

# Oracle9i

データベース・ユーティリティ

リリース 2 (9.2)

2002 年 7 月

部品番号 : J06265-01

ORACLE®

---

Oracle9i データベース・ユーティリティ, リリース 2 (9.2)

部品番号: J06265-01

原本名: Oracle9i Database Utilities, Release 2 (9.2)

原本部品番号: A96652-01

原本著者: Kathy Rich

原本協力者: Lee Barton、Ellen Batbouta、Janet Blowney、George Claborn、Jay Davison、William Fisher、Dean Gagne、John Galanes、John Kalogeropoulos、Jonathan Klein、Cindy Lim、Eric Magrath、Brian McCarthy、Ray Pfau、Rich Phillips、Paul Reilly、Mike Sakayeda、Francisco Sanchez、Jim Stenoish

Copyright © 1996, 2002, Oracle Corporation. All rights reserved.

Printed in Japan.

制限付権利の説明

プログラム（ソフトウェアおよびドキュメントを含む）の使用、複製または開示は、オラクル社との契約に記された制約条件に従うものとします。著作権、特許権およびその他の知的財産権に関する法律により保護されています。

当プログラムのリバース・エンジニアリング等は禁止されております。

このドキュメントの情報は、予告なしに変更されることがあります。オラクル社は本ドキュメントの無謬性を保証しません。

\* オラクル社とは、**Oracle Corporation**（米国オラクル）または**日本オラクル株式会社**（日本オラクル）を指します。

危険な用途への使用について

オラクル社製品は、原子力、航空産業、大量輸送、医療あるいはその他の危険が伴うアプリケーションを用途として開発されておりません。オラクル社製品を上述のようなアプリケーションに使用することについての安全確保は、顧客各位の責任と費用により行ってください。万一かかる用途での使用によりクレームや損害が発生いたしましても、日本オラクル株式会社と開発元である **Oracle Corporation**（米国オラクル）およびその関連会社は一切責任を負いかねます。当プログラムを米国国防総省の米国政府機関に提供する際には、『**Restricted Rights**』と共に提供してください。この場合次の Notice が適用されます。

Restricted Rights Notice

Programs delivered subject to the DOD FAR Supplement are "commercial computer software" and use, duplication, and disclosure of the Programs, including documentation, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement. Otherwise, Programs delivered subject to the Federal Acquisition Regulations are "restricted computer software" and use, duplication, and disclosure of the Programs shall be subject to the restrictions in FAR 52.227-19, Commercial Computer Software - Restricted Rights (June, 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065.

このドキュメントに記載されているその他の会社名および製品名は、あくまでその製品および会社を識別する目的にのみ使用されており、それぞれの所有者の商標または登録商標です。

---

---

# 目次

はじめに .....	xxxix
対象読者 .....	xxxix
構成 .....	xxxix
関連文書 .....	xxxix
表記規則 .....	xxxix

データベース・ユーティリティの新機能 .....	xxxix
リリース 2 (9.2) での Oracle9i ユーティリティの新機能 .....	xl
リリース 1 (9.0.1) での Oracle9i ユーティリティの新機能 .....	xli
Oracle8i ユーティリティの新機能 .....	xliv

## 第1部 エクスポートおよびインポート

### 1 エクスポート

エクスポート・ユーティリティとは .....	1-3
エクスポート・ユーティリティを使用する前に .....	1-4
catexp.sql または catalog.sql の実行 .....	1-4
十分なディスク領域の確認 .....	1-5
アクセス権限の確認 .....	1-5
エクスポート・ユーティリティの起動 .....	1-6
コマンドライン .....	1-6
対話方式のエクスポート・プロンプト .....	1-6
パラメータ・ファイル .....	1-6
SYSDBA でのエクスポート・ユーティリティの起動 .....	1-8
エクスポート・モード .....	1-9

表レベル・エクスポートおよびパーティション・レベル・エクスポート .....	1-13
表レベル・エクスポート .....	1-13
パーティション・レベル・エクスポート .....	1-13
処理上の制限事項 .....	1-13
<b>オンライン・ヘルプの利用</b> .....	1-14
<b>エクスポート・パラメータ</b> .....	1-14
BUFFER .....	1-18
例：バッファ・サイズの計算 .....	1-18
COMPRESS .....	1-18
CONSISTENT .....	1-19
CONSTRAINTS .....	1-21
DIRECT .....	1-21
FEEDBACK .....	1-21
FILE .....	1-22
FILESIZE .....	1-22
FLASHBACK_SCN .....	1-23
FLASHBACK_TIME .....	1-24
FULL .....	1-24
GRANTS .....	1-24
HELP .....	1-24
INDEXES .....	1-24
LOG .....	1-25
OBJECT_CONSISTENT .....	1-25
OWNER .....	1-25
PARFILE .....	1-25
QUERY .....	1-26
制限事項 .....	1-26
RECORDLENGTH .....	1-27
RESUMABLE .....	1-27
RESUMABLE_NAME .....	1-28
RESUMABLE_TIMEOUT .....	1-28
ROWS .....	1-28
STATISTICS .....	1-28
TABLES .....	1-29
表名の制限 .....	1-30

TABLESPACES .....	1-31
TRANSPORT_TABLESPACE .....	1-31
TRIGGERS .....	1-32
TTS_FULL_CHECK .....	1-32
USERID (ユーザー名 / パスワード) .....	1-32
VOLSIZE .....	1-33
パラメータ間の相互作用 .....	1-33
<b>エクスポート・セッションの例</b> .....	1-33
全データベース・モードでのエクスポート・セッションの例 .....	1-34
ユーザー・モードでのエクスポート・セッションの例 .....	1-37
表モードでのエクスポート・セッションの例 .....	1-38
例 1: DBA による 2 人のユーザーの表のエクスポート .....	1-39
例 2: ユーザーによる自分が所有する表のエクスポート .....	1-40
例 3: パターン一致を使用した様々な表のエクスポート .....	1-40
パーティション・レベル・エクスポートでのエクスポート・セッションの例 .....	1-41
例 1: パーティションを指定しない表のエクスポート .....	1-41
例 2: パーティションを指定した表のエクスポート .....	1-42
例 3: コンポジット・パーティションのエクスポート .....	1-43
<b>対話方式の使用</b> .....	1-44
制限事項 .....	1-48
<b>警告、エラーおよび完了メッセージ</b> .....	1-49
ログ・ファイル .....	1-49
警告メッセージ .....	1-49
リカバリ不能エラー・メッセージ .....	1-49
完了メッセージ .....	1-50
<b>終了コードによる結果の検査と表示</b> .....	1-50
<b>従来型パス・エクスポートおよびダイレクト・パス・エクスポート</b> .....	1-51
<b>ダイレクト・パス・エクスポートの起動</b> .....	1-53
ダイレクト・パス・エクスポートのセキュリティに関する考慮点 .....	1-53
ダイレクト・パス・エクスポートのパフォーマンスの問題 .....	1-54
<b>ネットワークに関する考慮点</b> .....	1-54
ネットワークを介してエクスポート・ファイルを転送する方法 .....	1-54
Oracle Net でのエクスポートおよびインポート .....	1-54
<b>キャラクタ・セットおよびグローバリゼーション・サポートに関する考慮点</b> .....	1-55
キャラクタ・セット変換 .....	1-55
変換によるキャラクタ・セットのソート順への影響 .....	1-56

マルチバイト・キャラクタ・セットおよびエクスポートとインポート .....	1-56
<b>インスタンス親和性とエクスポート .....</b>	<b>1-57</b>
<b>データベース・オブジェクトのエクスポートに関する考慮点 .....</b>	<b>1-57</b>
順序のエクスポート .....	1-57
LONG データ型および LOB データ型のエクスポート .....	1-57
外部関数ライブラリのエクスポート .....	1-58
オフライン・ローカル管理表領域のエクスポート .....	1-58
ディレクトリ別名のエクスポート .....	1-58
BFILE 列および属性のエクスポート .....	1-58
外部表 .....	1-58
オブジェクト型定義のエクスポート .....	1-59
ネストした表のエクスポート .....	1-59
アドバンスト・キュー (AQ) 表のエクスポート .....	1-59
シノニムのエクスポート .....	1-60
Java シノニムに関連して発生する可能性があるエクスポート・エラー .....	1-60
ファイングレイン・アクセス・コントロールのサポート .....	1-60
<b>トランスポータブル表領域 .....</b>	<b>1-61</b>
<b>読み込み専用データベースからのエクスポート .....</b>	<b>1-62</b>
<b>エクスポート・ユーティリティおよびインポート・ユーティリティを使用した     データベース移行のパーティション化 .....</b>	<b>1-62</b>
移行をパーティション化する場合のメリット .....	1-62
移行をパーティション化する場合のデメリット .....	1-62
エクスポート・ユーティリティおよびインポート・ユーティリティを使用した データベース移行のパーティション化方法 .....	1-63
<b>リリースおよびバージョンが異なるエクスポート・ユーティリティの使用法 .....</b>	<b>1-63</b>
リリースおよびバージョンが異なるエクスポート・ユーティリティおよび インポート・ユーティリティの使用上の制限 .....	1-64
リリースが異なるエクスポート・ユーティリティおよびインポート・ユーティリティの使用例 .....	1-65
Oracle9i データベースからの Oracle8 のエクスポート・ファイルの作成 .....	1-65
異なるリリースおよびバージョンを使用するときに発生する可能性があるエラー .....	1-66
EXP-00024 .....	1-66
IMP-00023 .....	1-67
EXP-00037 .....	1-67

## 2 インポート

インポート・ユーティリティとは .....	2-3
-----------------------	-----

表オブジェクト:インポートの順序 .....	2-4
<b>インポート・ユーティリティを使用する前に .....</b>	<b>2-5</b>
catexp.sql または catalog.sql の実行 .....	2-5
アクセス権限の確認 .....	2-6
オブジェクトをスキーマにインポートする方法 .....	2-6
権限のインポート .....	2-7
他のスキーマへのオブジェクトのインポート .....	2-7
システム・オブジェクトのインポート .....	2-8
<b>既存の表へのインポート .....</b>	<b>2-8</b>
データのインポート前に手動で表を作成する方法 .....	2-9
参照制約を使用禁止にする方法 .....	2-9
手動によるインポートの順序付け .....	2-10
<b>インポート操作上のスキーマおよびデータベース・トリガーの影響 .....</b>	<b>2-10</b>
<b>インポート・ユーティリティの起動 .....</b>	<b>2-11</b>
コマンドライン .....	2-11
対話方式のインポート・プロンプト .....	2-11
パラメータ・ファイル .....	2-11
SYSDBA でのインポート・ユーティリティの起動 .....	2-13
<b>インポート・モード .....</b>	<b>2-14</b>
<b>オンライン・ヘルプの利用 .....</b>	<b>2-15</b>
<b>インポート・パラメータ .....</b>	<b>2-15</b>
BUFFER .....	2-19
CHARSET .....	2-19
COMMIT .....	2-20
COMPILE .....	2-20
CONSTRAINTS .....	2-21
DATAFILES .....	2-21
DESTROY .....	2-21
FEEDBACK .....	2-22
FILE .....	2-22
FILESIZE .....	2-22
FROMUSER .....	2-23
FULL .....	2-23
GRANTS .....	2-24
HELP .....	2-24
IGNORE .....	2-24

INDEXES .....	2-25
INDEXFILE .....	2-25
LOG .....	2-26
PARFILE .....	2-26
RECORDLENGTH .....	2-26
RESUMABLE .....	2-27
RESUMABLE_NAME .....	2-27
RESUMABLE_TIMEOUT .....	2-27
ROWS .....	2-28
SHOW .....	2-28
SKIP_UNUSABLE_INDEXES .....	2-28
STATISTICS .....	2-29
STREAMS_CONFIGURATION .....	2-29
STREAMS_INSTANTIATION .....	2-30
TABLES .....	2-30
表名の制限 .....	2-31
TABLESPACES .....	2-32
TOID_NOVALIDATE .....	2-32
TOUSER .....	2-33
TRANSPORT_TABLESPACE .....	2-34
TTS_OWNERS .....	2-34
USERID (ユーザー名 / パスワード) .....	2-34
VOLSIZE .....	2-35
<b>インポート・セッションの例</b> .....	2-35
特定のユーザーの表を選択してインポートする例 .....	2-36
別のユーザーによってエクスポートされた表をインポートする例 .....	2-37
あるユーザーの表を別のユーザーへインポートする例 .....	2-38
パーティション・レベル・インポートでのインポート・セッションの例 .....	2-39
例 1: パーティション・レベル・インポート .....	2-39
例 2: コンポジット・パーティション表のパーティション・レベル・インポート .....	2-40
例 3: 別の列での表の再パーティション化 .....	2-41
パターン一致を使用して様々な表をインポートする例 .....	2-44
<b>対話方式の使用</b> .....	2-45
<b>警告、エラーおよび完了メッセージ</b> .....	2-47
ログ・ファイル .....	2-47



警告メッセージ .....	2-47
リカバリ不能エラー・メッセージ .....	2-47
完了メッセージ .....	2-48
<b>終了コードによる結果の検査と表示 .....</b>	<b>2-48</b>
<b>インポート中のエラー処理 .....</b>	<b>2-49</b>
行エラー .....	2-49
整合性制約違反 .....	2-49
無効なデータ .....	2-49
データベース・オブジェクトのインポートでのエラー .....	2-50
既存オブジェクト .....	2-50
順序 .....	2-50
リソース・エラー .....	2-51
ドメイン索引メタデータ .....	2-51
<b>表レベル・インポートおよびパーティション・レベル・インポート .....</b>	<b>2-51</b>
表レベル・インポートの使用に関するガイドライン .....	2-51
パーティション・レベル・インポートの使用に関するガイドライン .....	2-52
パーティションと表の間のデータ移行 .....	2-53
<b>索引作成およびメンテナンスの制御 .....</b>	<b>2-54</b>
索引作成の延期 .....	2-54
索引作成およびメンテナンスの制御 .....	2-54
索引更新延期の例 .....	2-54
<b>データベースの断片化を解消する方法 .....</b>	<b>2-55</b>
<b>ネットワークに関する考慮点 .....</b>	<b>2-56</b>
ネットワークを介してエクスポート・ファイルを転送する方法 .....	2-56
Oracle Net を使用したエクスポートおよびインポート .....	2-56
<b>キャラクタ・セットおよびグローバリゼーション・サポートに関する考慮点 .....</b>	<b>2-57</b>
キャラクタ・セット変換 .....	2-57
ユーザー・データ .....	2-57
DDL .....	2-57
インポートおよびシングルバイト・キャラクタ・セット .....	2-58
インポートおよびマルチバイト・キャラクタ・セット .....	2-58
<b>データベース・オブジェクトのインポートに関する考慮点 .....</b>	<b>2-58</b>
オブジェクト識別子のインポート .....	2-58
既存のオブジェクト表およびオブジェクト型の含まれている表のインポート .....	2-60
ネストした表のインポート .....	2-61
REF データのインポート .....	2-61

BFILE 列およびディレクトリ別名のインポート .....	2-62
外部関数ライブラリのインポート .....	2-62
ストアド・プロシージャ、ファンクションおよびパッケージのインポート .....	2-62
Java オブジェクトのインポート .....	2-63
外部表のインポート .....	2-63
AQ 表のインポート .....	2-63
LONG 列のインポート .....	2-63
ビューのインポート .....	2-64
パーティション表のインポート .....	2-64
ファイングレイン・アクセス・コントロールに対するサポート .....	2-65
マテリアライズド・ビューおよびスナップショット .....	2-65
スナップショット・ログ .....	2-66
スナップショット .....	2-66
スナップショットのインポート .....	2-66
異なるスキーマへのスナップショットのインポート .....	2-67
トランSPORTABLE表領域 .....	2-67
記憶域パラメータ .....	2-68
OPTIMAL パラメータ .....	2-68
OID 索引と LOB 列の記憶域パラメータ .....	2-68
記憶域パラメータの上書き .....	2-69
エクスポート・パラメータ COMPRESS .....	2-69
読み込み専用表領域 .....	2-69
表領域を削除する方法 .....	2-69
表領域を再編成する方法 .....	2-70
統計情報のインポート .....	2-70
エクスポート・ユーティリティおよびインポート・ユーティリティを使用した	
データベース移行のパーティション化 .....	2-71
移行をパーティション化する場合のメリット .....	2-71
移行をパーティション化する場合のデメリット .....	2-72
エクスポート・ユーティリティおよびインポート・ユーティリティを使用した	
データベース移行のパーティション化方法 .....	2-72
以前のリリースの Oracle のエクスポート・ファイルの使用法 .....	2-72
Oracle7 のエクスポート・ファイルの使用法 .....	2-72
DATE 列に関する CHECK 制約 .....	2-73
Oracle バージョン 6 のエクスポート・ファイルの使用法 .....	2-73
ユーザー権限 .....	2-73
CHAR 列 .....	2-73

整合性制約の状態 .....	2-73
デフォルト列値の長さ .....	2-73
Oracle バージョン 5 のエクスポート・ファイルの使用方法 .....	2-74
リリースおよびバージョンが異なるエクスポート・ユーティリティおよび インポート・ユーティリティの使用上の制限 .....	2-74
CHARSET パラメータ .....	2-75

## 第 II 部 SQL\*Loader

### 3 SQL\*Loader の概念

SQL*Loader の機能 .....	3-2
SQL*Loader 制御ファイル .....	3-3
入力データおよびデータ・ファイル .....	3-4
固定レコード形式 .....	3-4
可変レコード形式 .....	3-5
ストリーム・レコード形式 .....	3-6
論理レコード .....	3-7
データ・フィールド .....	3-8
LOBFILE および SDF .....	3-8
データ変換およびデータ型仕様 .....	3-9
廃棄レコードおよび拒否レコード .....	3-9
不良ファイル .....	3-9
SQL*Loader による拒否 .....	3-10
Oracle による拒否 .....	3-10
廃棄ファイル .....	3-10
ログ・ファイルおよびログ情報 .....	3-11
従来型パス・ロード、ダイレクト・パス・ロードおよび外部表ロード .....	3-11
従来型パス・ロード .....	3-11
ダイレクト・パス・ロード .....	3-12
パラレル・ダイレクト・パス .....	3-12
外部表ロード .....	3-12
オブジェクト、コレクションおよび LOB のロード .....	3-12
サポートされるオブジェクト型 .....	3-13
列オブジェクト .....	3-13
行オブジェクト .....	3-13
サポートされるコレクション型 .....	3-13

ネストした表 .....	3-13
VARRAY .....	3-13
サポートされる LOB 型 .....	3-14
パーティション・オブジェクトのサポート .....	3-14
アプリケーション開発:ダイレクト・パス・ロード API .....	3-15

## 4 SQL\*Loader コマンドライン・リファレンス

SQL*Loader の起動 .....	4-2
制御ファイルでのパラメータの指定 .....	4-3
コマンドライン・パラメータ .....	4-4
BAD (不良ファイル) .....	4-4
BINDSIZE (最大サイズ) .....	4-4
COLUMNARRAYROWS .....	4-4
CONTROL (制御ファイル) .....	4-5
DATA (データ・ファイル) .....	4-5
DATE_CACHE .....	4-5
DIRECT (データ・パス) .....	4-6
DISCARD (ファイル名) .....	4-6
DISCARDMAX (整数) .....	4-6
ERRORS (エラーの許容最大数) .....	4-7
EXTERNAL_TABLE .....	4-7
EXTERNAL_TABLE の使用上の制限 .....	4-9
FILE (ロード先ファイル) .....	4-9
LOAD (ロードするレコード) .....	4-9
LOG (ログ・ファイル) .....	4-9
MULTITHREADING .....	4-9
PARALLEL (パラレル・ロード) .....	4-10
PARFILE (パラメータ・ファイル) .....	4-10
READSIZE (読み込みバッファ・サイズ) .....	4-11
RESUMABLE .....	4-11
RESUMABLE_NAME .....	4-12
RESUMABLE_TIMEOUT .....	4-12
ROWS (1 回にコミットする行数) .....	4-12
SILENT (フィードバック・モード) .....	4-13
SKIP (スキップされるレコード) .....	4-14

SKIP_INDEX_MAINTENANCE .....	4-14
SKIP_UNUSABLE_INDEXES .....	4-14
STREAMSIZE .....	4-15
USERID (ユーザー名 / パスワード) .....	4-15
終了コードによる結果の検査と表示 .....	4-16

## 5 SQL\*Loader 制御ファイル・リファレンス

制御ファイルの内容 .....	5-2
制御ファイルのコメント .....	5-4
制御ファイル中でのコマンドライン・パラメータの指定 .....	5-4
OPTIONS 句 .....	5-4
ファイル名およびオブジェクト名の指定 .....	5-5
SQL および SQL*Loader の予約語と競合するファイル名 .....	5-5
SQL 文字列の指定 .....	5-5
オペレーティング・システムに関する考慮点 .....	5-5
完全パスの指定 .....	5-5
バックスラッシュ・エスケープ文字の使用 .....	5-6
移植不能文字列 .....	5-6
バックスラッシュのエスケープ .....	5-6
エスケープ文字が使用できない場合 .....	5-6
データ・ファイルの指定 .....	5-7
INFILE 構文の例 .....	5-9
複数のデータ・ファイルの指定 .....	5-9
BEGINDATA による制御ファイルのデータの識別 .....	5-10
データ・ファイル形式およびバッファリングの指定 .....	5-11
不良ファイルの指定 .....	5-11
不良ファイル名指定の例 .....	5-13
LOBFILE および SDF を使用した不良ファイルの処理方法 .....	5-13
拒否レコードの条件 .....	5-13
廃棄ファイルの指定 .....	5-14
制御ファイルでの廃棄ファイルの指定 .....	5-14
コマンドラインからの廃棄ファイルの指定 .....	5-15
廃棄ファイル名指定の例 .....	5-15
廃棄レコードの条件 .....	5-15
LOBFILE および SDF を使用した廃棄ファイルの処理方法 .....	5-16
廃棄レコード数の上限付け .....	5-16

<b>異なる文字コード体系の処理 .....</b>	<b>5-16</b>
マルチバイト（アジア系言語）・キャラクタ・セット .....	5-17
Unicode キャラクタ・セット .....	5-17
データベース・キャラクタ・セット .....	5-18
データ・ファイル・キャラクタ・セット .....	5-18
入力文字の変換 .....	5-19
CHARACTERSET パラメータ .....	5-19
制御ファイル・キャラクタ・セット .....	5-21
文字長セマンティクス .....	5-22
<b>中断されたロード .....</b>	<b>5-23</b>
中断された従来型パス・ロード .....	5-24
中断されたダイレクト・パス・ロード .....	5-24
領域エラーによって中断されたロード .....	5-24
エラーが最大数を越えたことによって中断されたロード .....	5-25
致命的エラーによって中断されたロード .....	5-25
[Ctrl] を押しながら [C] を押すことによって中断されたロード .....	5-25
ロードの中断後の表および索引の状態 .....	5-25
ログ・ファイルを使用したロード状態の確認 .....	5-25
単一表へのロードの継続 .....	5-26
<b>物理レコードからの論理レコードの作成 .....</b>	<b>5-26</b>
CONCATENATE を使用した論理レコードの作成 .....	5-26
CONTINUEIF を使用した論理レコードの作成 .....	5-27
<b>表への論理レコードのロード .....</b>	<b>5-30</b>
表名の指定 .....	5-31
INTO TABLE 句 .....	5-31
表固有のロード方法 .....	5-31
空の表へのデータのロード .....	5-32
空でない表へのデータのロード .....	5-32
表固有の OPTIONS パラメータ .....	5-33
条件に基づいたレコードのロード .....	5-34
LOBFILE および SDF での WHEN 句の使用 .....	5-34
デフォルトのデータ・デリミタ（区切り記号）の指定 .....	5-35
fields_spec .....	5-35
termination_spec .....	5-35
enclosure_spec .....	5-35
データが欠落しているショート・レコードの処理 .....	5-36

TRAILING NULLCOLS 句 .....	5-37
索引オプション .....	5-37
SORTED INDEXES 句 .....	5-37
SINGLEROW オプション .....	5-38
複数の INTO TABLE 句を使用するメリット .....	5-38
複数の論理レコードの抽出 .....	5-39
デリミタに基づく相対的な位置指定 .....	5-39
異なる入力レコード形式の区別 .....	5-40
POSITION パラメータに基づく相対的な位置指定 .....	5-40
異なる入力行オブジェクトのサブタイプの区別 .....	5-41
複数表へのデータのロード .....	5-42
要約 .....	5-43
バインド配列および従来型パス・ロード .....	5-43
バインド配列のサイズ要件 .....	5-43
バインド配列のパフォーマンスに関する考慮点 .....	5-44
行数およびバインド配列サイズの指定 .....	5-44
バインド配列サイズを確認するための計算 .....	5-45
長さインジケータのサイズの決定 .....	5-46
フィールド・バッファ・サイズの計算 .....	5-47
バインド配列用のメモリー所要量の最小化 .....	5-48
複数の INTO TABLE 句に対するバインド配列サイズの計算 .....	5-49

## 6 フィールド・リスト・リファレンス

フィールド・リストの内容 .....	6-2
データ・フィールドの位置指定 .....	6-3
タブを含むデータでの POSITION の使用 .....	6-4
複数表のロードでの POSITION の使用 .....	6-4
POSITION を使用した例 .....	6-5
列およびフィールドの指定 .....	6-5
FILLER フィールドの指定 .....	6-6
データ・フィールドのデータ型の指定 .....	6-7
SQL*Loader のデータ型 .....	6-7
移植不能なデータ型 .....	6-8
INTEGER( <i>n</i> ) .....	6-8
SMALLINT .....	6-9
FLOAT .....	6-9

DOUBLE .....	6-10
BYTEINT .....	6-10
ZONED .....	6-10
DECIMAL .....	6-11
VARGRAPHIC .....	6-11
VARCHAR .....	6-12
VARRAW .....	6-13
LONG VARRAW .....	6-14
移植可能なデータ型 .....	6-14
CHAR .....	6-14
日時データ型および期間データ型 .....	6-15
GRAPHIC .....	6-18
GRAPHIC EXTERNAL .....	6-18
数値型 EXTERNAL .....	6-19
RAW .....	6-19
VARCHARC .....	6-20
VARRAWC .....	6-20
システム固有のデータ型フィールド長の競合 .....	6-21
LENGTH-VALUE データ型のフィールド長 .....	6-21
データ型の変換 .....	6-22
日時データ型および期間データ型のデータ型変換 .....	6-22
デリミタの指定 .....	6-24
TERMINATED フィールド .....	6-24
ENCLOSED フィールド .....	6-24
データ中のデリミタ記号 .....	6-26
デリミタ付きデータの最大長 .....	6-27
デリミタを使用した後続の空白のロード .....	6-27
文字データ型フィールド長の競合 .....	6-27
事前にサイズが決まっているフィールド .....	6-27
デリミタ付きフィールド .....	6-28
日付フィールド・マスク .....	6-28
フィールド条件の指定 .....	6-29
フィールドと BLANKS の比較 .....	6-31
フィールドと文字列の比較 .....	6-31
WHEN、NULLIF および DEFAULTIF 句の使用 .....	6-32
異なるプラットフォーム間でのデータのロード .....	6-35
バイト順序 .....	6-36



バイト順序の指定 .....	6-37
バイト順序マーク (BOM) の使用 .....	6-38
BOM の確認の抑止 .....	6-39
<b>すべてが空白のフィールドのロード</b> .....	6-40
<b>空白の切捨て</b> .....	6-41
空白の切捨てが可能なデータ型 .....	6-43
空白の切捨てが可能なデータ型に対するフィールド長指定 .....	6-44
事前にサイズが決まっているフィールド .....	6-44
デリミタ付きフィールド .....	6-44
フィールドの相対的な位置指定 .....	6-45
フィールドの開始位置が指定されていない場合 .....	6-45
前のフィールドの終端がデリミタで指定されている場合 .....	6-45
前のフィールドに囲みデリミタおよび終了デリミタの両方が含まれる場合 .....	6-45
先頭の空白 .....	6-46
前のフィールドが空白で区切られている場合 .....	6-46
オプションの囲みデリミタ .....	6-46
後続の空白 .....	6-47
囲まれたフィールド .....	6-47
<b>空白の保存</b> .....	6-47
PRESERVE BLANKS オプション .....	6-48
空白で区切られている場合 .....	6-48
<b>フィールドへの SQL 演算子の適用</b> .....	6-48
フィールドの参照 .....	6-50
フィールド指定での SQL 演算子の共通使用 .....	6-51
SQL 演算子の組合せ .....	6-51
日付マスク付きの SQL 文字列の使用 .....	6-51
書式化されたフィールドの解析 .....	6-52
<b>SQL*Loader を使用した入力データの生成</b> .....	6-52
ファイルを使用しないデータのロード .....	6-52
列への定数値の設定 .....	6-53
CONSTANT パラメータ .....	6-53
列への式の値の設定 .....	6-53
EXPRESSION パラメータ .....	6-53
列へのデータ・ファイルのレコード番号の設定 .....	6-54
RECNUM パラメータ .....	6-54
列への現在の日付の設定 .....	6-54

SYSDATE パラメータ .....	6-54
列への一意の順序番号の設定 .....	6-54
SEQUENCE パラメータ .....	6-55
複数の表に対する順序番号の生成 .....	6-56
例:挿入ごとの順序番号の生成 .....	6-56

## 7 オブジェクト、LOB およびコレクションのロード

<b>列オブジェクトのロード</b> .....	7-2
ストリーム・レコード形式への列オブジェクトのロード .....	7-2
可変レコード形式への列オブジェクトのロード .....	7-3
ネストした列オブジェクトのロード .....	7-4
導出サブタイプを使用した列オブジェクトのロード .....	7-5
オブジェクトに対する NULL 値の指定 .....	7-6
NULL 属性の指定 .....	7-6
アトミック NULL の指定 .....	7-7
ユーザー定義コンストラクタを使用した列オブジェクトのロード .....	7-8
<b>オブジェクト表のロード</b> .....	7-12
サブタイプを使用したオブジェクト表のロード .....	7-14
<b>REF 列のロード</b> .....	7-15
実 REF 列 .....	7-15
主キー REF 列 .....	7-16
主キーが使用可能な有効範囲なし REF 列 .....	7-16
<b>LOB のロード</b> .....	7-18
プライマリ・データ・ファイルからの LOB データのロード .....	7-19
事前に決められたサイズのフィールドの LOB データ .....	7-19
デリミタ付きフィールドの LOB データ .....	7-20
Length-Value Pair フィールドの LOB データ .....	7-21
外部 LOBFILE (BFILE) からの LOB データのロード .....	7-22
LOBFILE からの LOB データのロード .....	7-23
動的および静的 LOBFILE 指定 .....	7-24
LOBFILE からの LOB データのロードの例 .....	7-24
LOBFILE から LOB をロードする場合の考慮点 .....	7-28
<b>コレクション (ネストした表および VARRAY) のロード</b> .....	7-29
ネストした表および VARRAY での制限事項 .....	7-30
SDF .....	7-31
<b>動的および静的 SDF 指定</b> .....	7-32

親表を子表から分割してのロード .....	7-33
VARRAY 列ロード時のメモリーの問題 .....	7-34

## 8 SQL\*Loader ログ・ファイル・リファレンス

ヘッダー情報 .....	8-2
グローバル情報 .....	8-2
表情報 .....	8-3
列情報 .....	8-4
位置 .....	8-4
長さ .....	8-4
デリミタ .....	8-4
データ型 .....	8-4
データ・ファイル情報 .....	8-5
表ロード情報 .....	8-5
サマリー統計 .....	8-6
Oracle のログ用統計レポート .....	8-7
単一パーティションのロード情報 .....	8-7
表ロード時の統計 .....	8-7
ダイレクト・パス・ロードおよびマルチスレッドのサマリー統計の追加 .....	8-8
EXTERNAL_TABLE=GENERATE_ONLY の使用時に作成されるログ・ファイル .....	8-8

## 9 従来型パス・ロードおよびダイレクト・パス・ロード

データのロード方法 .....	9-2
従来型パス・ロード .....	9-4
単一パーティションの従来型パス・ロード .....	9-4
従来型パスを使用する場合 .....	9-4
ダイレクト・パス・ロード .....	9-5
ダイレクト・パス・ロード時のデータ変換 .....	9-6
パーティション表またはサブパーティション表のダイレクト・パス・ロード .....	9-6
単一パーティションまたはサブパーティションのダイレクト・パス・ロード .....	9-7
ダイレクト・パス・ロードのメリット .....	9-8
ダイレクト・パス・ロード使用上の制限 .....	9-8
単一パーティションのダイレクト・パス・ロードでの制限 .....	9-9
ダイレクト・パスを使用する場合 .....	9-9
整合性制約 .....	9-10
ダイレクト・パスのフィールド・デフォルト .....	9-10

シノニムへのロード .....	9-10
<b>ダイレクト・パス・ロードの使用 .....</b>	<b>9-10</b>
ダイレクト・パス・ロードのセットアップ .....	9-10
ダイレクト・パス・ロードの指定 .....	9-11
索引の作成 .....	9-11
パフォーマンスの向上 .....	9-11
一時セグメント記憶域要件 .....	9-12
使用禁止状態 (Index Unusable) のままの索引 .....	9-12
データ・セーブを使用したデータ損失の防止 .....	9-13
ROWS パラメータの使用 .....	9-13
データ・セーブとコミット .....	9-14
ダイレクト・パス・ロード時のデータ・リカバリ .....	9-14
メディア・リカバリおよびダイレクト・パス・ロード .....	9-15
インスタンス・リカバリおよびダイレクト・パス・ロード .....	9-15
LONG 型データ・フィールドのロード .....	9-15
PIECED としてのデータのロード .....	9-16
<b>ダイレクト・パス・ロードのパフォーマンスの最適化 .....</b>	<b>9-17</b>
高速ロードのための記憶域の事前割当て .....	9-17
高速索引付けのためのデータの事前ソート .....	9-17
SORTED INDEXES 句 .....	9-18
未ソートのデータ .....	9-18
複数列索引 .....	9-18
最適ソート順序の選択方法 .....	9-19
データ・セーブの回数の削減 .....	9-19
REDO ログの最小限の使用 .....	9-19
アーカイブの使用禁止 .....	9-20
UNRECOVERABLE パラメータの指定 .....	9-20
NOLOG 属性の設定 .....	9-20
列配列の行数およびストリーム・バッファ・サイズの指定 .....	9-21
日付キャッシュの値の指定 .....	9-21
<b>複数 CPU システムのダイレクト・パス・ロードの最適化 .....</b>	<b>9-23</b>
<b>索引メンテナンスの回避 .....</b>	<b>9-24</b>
<b>ダイレクト・ロード、整合性制約およびトリガー .....</b>	<b>9-24</b>
整合性制約 .....	9-24
使用可能な制約 .....	9-25
使用禁止の制約 .....	9-25
制約を使用可能に戻す方法 .....	9-25

挿入トリガー .....	9-26
挿入トリガーの整合性制約への置換 .....	9-27
自動制約が使用できない場合 .....	9-27
準備 .....	9-27
更新トリガーの使用 .....	9-27
例外処理と同じ処理の実現 .....	9-28
ストアド・プロシージャの使用 .....	9-28
永続的に使用禁止のトリガーおよび制約 .....	9-29
従来型パスの同時ロードによるパフォーマンスの向上 .....	9-29
<b>パラレル・データ・ロード・モデル</b> .....	9-29
従来型パスによる同時ロード .....	9-30
ダイレクト・パスによるセグメント間同時処理 .....	9-30
ダイレクト・パスによるセグメント内同時処理 .....	9-30
パラレル・ダイレクト・パス・ロードの制限 .....	9-31
複数の SQL*Loader セッションの初期化 .....	9-31
パラレル・ダイレクト・パス・ロードのパラメータ .....	9-32
一時セグメントの指定 .....	9-32
パラレル・ダイレクト・パス・ロード後の制約の使用可能化 .....	9-33
主キー制約および一意制約 .....	9-34
一般的なパフォーマンス改善のヒント .....	9-34

## 10 SQL\*Loader の事例

<b>事例</b> .....	10-3
<b>事例用ファイル</b> .....	10-4
<b>各事例で使用する表</b> .....	10-5
emp 表の内容 .....	10-5
dept 表の内容 .....	10-5
<b>ロード結果の確認</b> .....	10-5
<b>参照および注意</b> .....	10-6
<b>事例 1: 可変長データのロード</b> .....	10-6
事例 1 の制御ファイル .....	10-6
事例 1 の実行 .....	10-7
事例 1 のログ・ファイル .....	10-8
<b>事例 2: 固定形式フィールドのロード</b> .....	10-9
事例 2 の制御ファイル .....	10-9
事例 2 のデータ・ファイル .....	10-10

事例 2 の実行 .....	10-10
事例 2 のログ・ファイル .....	10-11
<b>事例 3: 自由区分形式ファイルのロード .....</b>	<b>10-12</b>
事例 3 の制御ファイル .....	10-12
事例 3 の実行 .....	10-13
事例 3 のログ・ファイル .....	10-14
<b>事例 4: 結合された物理レコードのロード .....</b>	<b>10-15</b>
事例 4 の制御ファイル .....	10-16
事例 4 のデータ・ファイル .....	10-16
拒否レコード .....	10-17
事例 4 の実行 .....	10-17
事例 4 のログ・ファイル .....	10-18
事例 4 の不良ファイル .....	10-19
<b>事例 5: 複数表へのデータのロード .....</b>	<b>10-19</b>
事例 5 の制御ファイル .....	10-20
事例 5 のデータ・ファイル .....	10-21
事例 5 の実行 .....	10-21
事例 5 のログ・ファイル .....	10-22
事例 5 のロードされた表 .....	10-24
<b>事例 6: ダイレクト・パス・ロード方式を使用したデータのロード .....</b>	<b>10-25</b>
事例 6 の制御ファイル .....	10-26
事例 6 のデータ・ファイル .....	10-26
事例 6 の実行 .....	10-27
事例 6 のログ・ファイル .....	10-27
<b>事例 7: 書式化されたレポートからのデータの抽出 .....</b>	<b>10-29</b>
BEFORE INSERT トリガーの作成 .....	10-29
事例 7 の制御ファイル .....	10-30
事例 7 のデータ・ファイル .....	10-32
事例 7 の実行 .....	10-32
事例 7 のログ・ファイル .....	10-33
<b>事例 8: パーティション化された表のロード .....</b>	<b>10-34</b>
事例 8 の制御ファイル .....	10-35
表の作成 .....	10-35
事例 8 のデータ・ファイル .....	10-36
事例 8 の実行 .....	10-37

事例 8 のログ・ファイル .....	10-37
<b>事例 9: LOBFILE のロード (CLOB) .....</b>	<b>10-39</b>
事例 9 の制御ファイル .....	10-39
事例 9 のデータ・ファイル .....	10-40
事例 9 の実行 .....	10-42
事例 9 のログ・ファイル .....	10-42
<b>事例 10: REF フィールドおよび VARRAY のロード .....</b>	<b>10-43</b>
事例 10 の制御ファイル .....	10-44
事例 10 の実行 .....	10-45
事例 10 のログ・ファイル .....	10-46
<b>事例 11: Unicode キャラクタ・セットのデータのロード .....</b>	<b>10-48</b>
事例 11 の制御ファイル .....	10-48
事例 11 のデータ・ファイル .....	10-49
事例 11 の実行 .....	10-50
事例 11 のログ・ファイル .....	10-50
事例 11 のロードされた表 .....	10-52

## 第 III 部 外部表

### 11 外部表の概要

アクセス・ドライバ .....	11-2
外部表の制限事項 .....	11-3
データ・ファイルおよび出力ファイルの位置 .....	11-3
外部表を使用したデータのロード .....	11-5
外部表へのパラレル・アクセス .....	11-6
外部表使用時のパフォーマンスのヒント .....	11-6
SQL*Loader と外部表との処理内容の違い .....	11-7
複数のプライマリ入力データ・ファイル .....	11-7
構文およびデータ型 .....	11-7
拒否された行 .....	11-8
BOM .....	11-8
デフォルトのキャラクタ・セットおよび日付マスク .....	11-8

### 12 外部表アクセス・パラメータ

access_parameters 句 .....	12-2
---------------------------	------

<b>record_format_info 句</b> .....	12-3
FIXED .....	12-4
VARIABLE .....	12-4
DELIMITED BY .....	12-5
CHARACTERSET .....	12-6
DATA IS...ENDIAN .....	12-6
BYTE ORDER MARK (CHECK   NOCHECK) .....	12-7
STRING SIZES ARE IN .....	12-7
LOAD WHEN .....	12-8
BADFILE   NOBADFILE .....	12-8
DISCARDFILE   NODISCARDFILE .....	12-9
LOG FILE   NOLOGFILE .....	12-9
SKIP .....	12-9
READSIZE .....	12-10
DATE_CACHE .....	12-10
string .....	12-10
condition_spec .....	12-11
[directory object name:] filename .....	12-12
condition .....	12-13
range start : range end .....	12-13
<b>field_definitions 句</b> .....	12-14
delim_spec .....	12-15
例 : 終了デリミタを含む外部表 .....	12-17
例 : 囲みデリミタおよび終了デリミタを含む外部表 .....	12-17
例 : オプションの囲みデリミタを含む外部表 .....	12-17
trim_spec .....	12-18
MISSING FIELD VALUES ARE NULL .....	12-19
field_list .....	12-19
pos_spec 句 .....	12-21
start .....	12-21
* .....	12-21
increment .....	12-21
end .....	12-21
length .....	12-22
datatype_spec 句 .....	12-22
[UNSIGNED] INTEGER [EXTERNAL] [(len)] .....	12-24



DECIMAL [EXTERNAL] および ZONED [EXTERNAL] .....	12-24
ORACLE_DATE .....	12-24
ORACLE_NUMBER .....	12-24
DOUBLE [EXTERNAL] .....	12-25
FLOAT [EXTERNAL] .....	12-25
RAW .....	12-25
CHAR .....	12-25
date_format_spec .....	12-26
VARCHAR および VARRAW .....	12-27
VARCHARC および VARRAWC .....	12-28
init_spec 句 .....	12-29

## 第 IV 部 その他のユーティリティ

### 13 DBVERIFY: オフライン・データベース検査ユーティリティ

DBVERIFY を使用した単一データ・ファイルのディスク・ブロックの検査 .....	13-2
構文 .....	13-2
パラメータ .....	13-2
コマンドライン・インタフェース .....	13-3
DBVERIFY のサンプル出力 .....	13-3
DBVERIFY を使用したセグメントの検査 .....	13-4
構文 .....	13-5
パラメータ .....	13-5
コマンドライン・インタフェース .....	13-6

### 14 DBNEWID ユーティリティ

DBNEWID ユーティリティとは .....	14-2
DBID および DBNAME の変更による影響 .....	14-2
データベースの DBID および DBNAME の変更 .....	14-2
DBID およびデータベース名の変更 .....	14-3
データベース名のみの変更 .....	14-4
DBID の変更操作のトラブルシューティング .....	14-6
データベース名の変更操作のトラブルシューティング .....	14-6
DBNEWID ユーティリティの構文 .....	14-7
パラメータ .....	14-7

制限事項および使用上の注意 .....	14-8
DBNEWID ユーティリティの使用例 .....	14-9
DBID のみの変更 .....	14-9
DBID およびデータベース名の変更 .....	14-9
データベース名のみの変更 .....	14-9

## 15 メタデータ API の使用

メタデータ API の概要 .....	15-2
以前の方法によるメタデータの抽出 .....	15-2
メタデータ API の構成要素 .....	15-2
メタデータ API の機能 .....	15-3
インターネットによる計算 .....	15-3
メタデータ API の実装方法 .....	15-4
DBMS_METADATA とセキュリティ .....	15-4
DBMS_METADATA プログラム・インタフェース .....	15-5
DBMS_METADATA.FETCH_XML プロシージャの使用 .....	15-7
DBMS_METADATA.FETCH_DDL プロシージャの使用 .....	15-8
メタデータ API のプログラム・インタフェースに関するパフォーマンスのヒント .....	15-10
DBMS_METADATA ブラウザ・インタフェース .....	15-10
例: DBMS_METADATA ブラウザ・インタフェースの使用 .....	15-11
メタデータ API の例 .....	15-11
mddemo.sql .....	15-12
PAYROLL_DEMO の出力 .....	15-17

## 第 V 部 付録

### A SQL\*Loader の構文図

### B DB2/DXT ユーザーに対する注意事項

DB2 RESUME オプションの使用方法 .....	B-2
互換性維持のための機能 .....	B-2
LOG 文 .....	B-3
WORKDDN 文 .....	B-3
SORTDEVT 文および SORTNUM 文 .....	B-3
DISCARD の指定 .....	B-3

制限事項 ..... B-3

    FORMAT 文 ..... B-4

    PART 文 ..... B-4

    SQL/DS オプション ..... B-4

    DBCS GRAPHIC 型文字列 ..... B-4

DB2 と互換性のある文を含む SQL\*Loader の構文 ..... B-5

C   バックス正規形構文

索引

## 例目次

3-1	固定レコード形式でのデータのロード .....	3-5
3-2	可変レコード形式でのデータのロード .....	3-6
3-3	ストリーム・レコード形式でのデータのロード .....	3-7
5-1	サンプル制御ファイル .....	5-2
5-2	PRESERVE パラメータを使用しない CONTINUEIF THIS .....	5-29
5-3	PRESERVE パラメータを使用した CONTINUEIF THIS .....	5-29
5-4	PRESERVE パラメータを使用しない CONTINUEIF NEXT .....	5-30
5-5	PRESERVE パラメータを使用した CONTINUEIF NEXT .....	5-30
6-1	サンプル制御ファイルのフィールド・リスト・セクション .....	6-2
6-2	DEFAULTIF 句の無評価 .....	6-33
6-3	DEFAULTIF 句の評価 .....	6-34
6-4	DEFAULTIF 句を使用した位置の指定 .....	6-34
6-5	DEFAULTIF 句を使用したフィールド名の指定 .....	6-35
7-1	ストリーム・レコード形式への列オブジェクトのロード .....	7-3
7-2	可変レコード形式への列オブジェクトのロード .....	7-3
7-3	ネストした列オブジェクトのロード .....	7-4
7-4	サブタイプを使用した列オブジェクトのロード .....	7-5
7-5	NULLIF 句を使用した NULL 属性の指定 .....	7-6
7-6	FILLER フィールドを使用したデータのロード .....	7-7
7-7	属性値コンストラクタに一致するユーザー定義コンストラクタを使用した 列オブジェクトのロード .....	7-9
7-8	属性値コンストラクタに一致しないユーザー定義コンストラクタを使用した 列オブジェクトのロード .....	7-10
7-9	SQL 式を使用した、属性値コンストラクタに一致しない ユーザー定義コンストラクタでの列オブジェクトのロード .....	7-11
7-10	主キー OID を使用したオブジェクト表のロード .....	7-12
7-11	OID のロード .....	7-13
7-12	サブタイプを使用したオブジェクト表のロード .....	7-14
7-13	実 REF 列のロード .....	7-15
7-14	主キー REF 列のロード .....	7-16
7-15	事前に決められたサイズのフィールドの LOB データ .....	7-19
7-16	デリミタ付きフィールドの LOB データのロード .....	7-20
7-17	Length-Value Pair フィールドへの LOB データのロード .....	7-21
7-18	BFILE を使用したデータのロード：ファイル名のみを動的に指定 .....	7-22
7-19	BFILE を使用したデータのロード：ファイル名およびディレクトリ名を動的に指定 .....	7-23
7-20	LOBFILE 当たり 1 つの LOB を使用した LOB データのロード .....	7-25
7-21	事前に決められたサイズの LOB を使用した LOB データのロード .....	7-26
7-22	デリミタ付きフィールドの LOB を使用した LOB データのロード .....	7-27
7-23	Length-Value Pair を指定した LOB を使用した LOB データのロード .....	7-28
7-24	VARRAY およびネストした表のロード .....	7-30
7-25	ユーザー定義 SID を使用した親表のロード .....	7-33
7-26	ユーザー定義 SID を使用した子表（ネストした格納表）のロード .....	7-33
9-1	SQL*Loader の制御ファイルに対する日付書式の設定 .....	9-6

9-2	環境変数 NLS_DATE_FORMAT の設定 .....	9-6
-----	--------------------------------	-----

## 図目次

1-1	データベースのエクスポート .....	1-3
1-2	従来型パス・エクスポートおよびダイレクト・パス・エクスポートでの データベースの読み込み .....	1-52
2-1	エクスポート・ファイルのインポート .....	2-3
3-1	SQL*Loader の概要 .....	3-3
6-1	フィールド変換の例 .....	6-42
6-2	固定長フィールドの後の相対的な位置指定 .....	6-45
6-3	デリミタ付きフィールドの後の相対的な位置指定 .....	6-45
6-4	囲みデリミタの後の相対的な位置指定 .....	6-46
6-5	空白で区切られたフィールド .....	6-46
6-6	オプションの囲みデリミタ付きフィールド .....	6-47
9-1	SQL*Loader ダイレクト・パスおよび従来型パスでのデータベースの書き込み .....	9-3
15-1	DBMS_METADATA.FETCH_XML() の使用 .....	15-7
15-2	DBMS_METADATA.FETCH_DDL() の使用 .....	15-9

## 表目次

1-1	各モードでエクスポートおよびインポートされるオブジェクト .....	1-9
1-2	2 人のユーザーによる更新時のイベントの順序 .....	1-20
1-3	ダンプ・ファイルの最大サイズ .....	1-23
1-4	対話方式を使用したエクスポート・ユーティリティの起動 .....	1-44
1-5	エクスポート時の終了コード .....	1-50
1-6	リリースが異なるエクスポート・ユーティリティおよび インポート・ユーティリティの使用 .....	1-65
2-1	自分のスキーマにオブジェクトをインポートするために必要な権限 .....	2-6
2-2	権限のインポートに必要な権限 .....	2-7
2-3	対話方式を使用したインポート・ユーティリティの起動 .....	2-45
2-4	インポート時の終了コード .....	2-48
4-1	SQL*Loader の終了コード .....	4-16
5-1	INFILE 句のパラメータ .....	5-8
5-2	CONTINUEIF のパラメータ .....	5-27
5-3	固定長フィールド .....	5-47
5-4	非グラフィック・フィールド .....	5-47
5-5	グラフィック・フィールド .....	5-48
5-6	可変長フィールド .....	5-48
6-1	位置指定句のパラメータ .....	6-3
6-2	日時データ型および期間データ型のデータ型変換 .....	6-23
6-3	終了および囲みの指定に関するパラメータ .....	6-25
6-4	フィールド条件句のパラメータ .....	6-30
6-5	空白の切捨てに関する動作のサマリー .....	6-43
6-6	列指定に使用されるパラメータ .....	6-55
10-1	事例用ファイルおよび関連ファイル .....	10-4
14-1	DBNEWID ユーティリティのパラメータ .....	14-7
15-1	DBMS_METADATA プログラム・インタフェースのプロシージャ .....	15-5
15-2	DBMS_METADATA ブラウザ・インタフェースのプロシージャ .....	15-11
B-1	DB2 の関数とそれに相当する SQL*Loader のオプション .....	B-2
C-1	バックス正規形構文の記号および表記規則 .....	C-1





---

# はじめに

このマニュアルでは、Oracle9i データベース・ユーティリティを使用して、データ転送、データ・メンテナンスおよびデータベース管理を行う方法について説明します。

この章の内容は、次のとおりです。

- [対象読者](#)
- [このマニュアルの構成](#)
- [関連文書](#)
- [表記規則](#)

# 対象読者

このマニュアルは、データベース管理者（DBA）、アプリケーション・プログラマ、セキュリティ管理者、システム・オペレータを対象としています。また、次の作業を行う Oracle ユーザーも対象としています。

- エクスポート・ユーティリティおよびインポート・ユーティリティを使用した、データのアーカイブ、Oracle データベースのバックアップおよび Oracle データベース間のデータ移動
- SQL\*Loader を使用したオペレーティング・システムのファイルから Oracle 表へのデータのロード、または外部表機能を使用した外部ソースから Oracle 表へのデータのロード
- メタデータ API を使用した、データベース・オブジェクトに対するメタデータの完全な表現の抽出および処理
- DBNEWID ユーティリティを使用した、オペレーショナル・データベースの内部データベース識別子（DBID）およびデータベース名（DBNAME）の維持

このマニュアルを使用するには、『Oracle9i データベース概要』で説明されている SQL および Oracle の基礎知識が必要です。また、SQL\*Loader を使用するには、オペレーティング・システムのファイル管理機能の使用方法を理解しておく必要があります。

## このマニュアルの構成

このマニュアルは、次のように構成されています。

### 第1部「エクスポートおよびインポート」

#### 第1章「エクスポート」

この章では、エクスポート・ユーティリティを使用して Oracle データベースから転送可能なファイルへデータを書き込む方法を説明します。エクスポートの概要、エクスポート・モード、対話方式とコマンドライン方式、パラメータの指定およびエクスポート・オブジェクトのサポートについて説明します。エクスポート・セッションの例も示します。

#### 第2章「インポート」

この章では、インポート・ユーティリティを使用してエクスポート・ファイルのデータを Oracle データベースへ読み込む方法を説明します。インポートの概要、対話方式とコマンドライン方式、パラメータの指定およびインポート・オブジェクトのサポートについて説明します。インポート・セッションの例も示します。

## 第 II 部「SQL\*Loader」

### 第 3 章「SQL\*Loader の概念」

この章では、SQL\*Loader の機能について説明します。また、データ・ロードの概念（オブジェクト・サポートも含む）についても説明します。さらに、SQL\*Loader への入力、データベースの事前準備および SQL\*Loader からの出力についても説明します。

### 第 4 章「SQL\*Loader コマンドライン・リファレンス」

この章では、SQL\*Loader で使用するコマンドラインの構文について説明します。コマンドライン引数、SQL\*Loader のメッセージを抑制する方法、バインド配列のサイズ指定などについて説明します。

### 第 5 章「SQL\*Loader 制御ファイル・リファレンス」

この章では、SQL\*Loader の構成に使用する制御ファイルの構文、およびデータを Oracle の形式にマップする方法を SQL\*Loader に記述する方法について説明します。詳細な構文図を示すとともに、データ・ファイル、表と列、データの位置、ロードするデータの型と形式などの指定に関する情報についても説明します。

### 第 6 章「フィールド・リスト・リファレンス」

この章では、SQL\*Loader 制御ファイルのフィールド・リストのセクションについて説明します。フィールド・リストには、位置、データ型、条件、デリミタなどのロードするフィールドについての情報が提供されます。

### 第 7 章「オブジェクト、LOB およびコレクションのロード」

この章では、様々な形式での列オブジェクトのロード方法について説明します。オブジェクト表、REF 列、LOB およびコレクションのロード方法についても説明します。

### 第 8 章「SQL\*Loader ログ・ファイル・リファレンス」

この章では、ログ・ファイルに記述される情報について説明します。

### 第 9 章「従来型パス・ロードおよびダイレクト・パス・ロード」

この章では、従来型パス・ロードとダイレクト・パス・ロードの違いを説明します。ダイレクト・パス・ロードは、大量のデータを従来より高速にロードするための高パフォーマンス・オプションです。

### 第 10 章「SQL\*Loader の事例」

この章では、様々な事例から SQL\*Loader の機能を説明します。可変長データ、固定形式レコード、自由形式ファイルのロード方法、複数の物理レコードを 1 件の論理レコードとしてロードする方法、複数の表のロード方法、ダイレクト・パスを使用したロード方法、およびオブジェクト、コレクション、REF 列のロード方法について説明します。

## 第 III 部「外部表」

### 第 11 章「外部表の概要」

この章では、外部表の基本概念について説明します。

### 第 12 章「外部表アクセス・パラメータ」

この章では、外部表 API でのインタフェースに使用する接続パラメータについて説明します。

## 第 IV 部「その他のユーティリティ」

### 第 13 章「DBVERIFY: オフライン・データベース検査ユーティリティ」

この章では、オフライン・データベース検査ユーティリティである DBVERIFY の使用方法について説明します。

### 第 14 章「DBNEWID ユーティリティ」

この章では、DBNEWID ユーティリティを使用してデータベースの名前または ID（あるいはその両方）を変更する方法について説明します。

### 第 15 章「メタデータ API の使用」

データベース・オブジェクトに対するメタデータの完全な表現の抽出および処理が可能なメタデータ API について説明します。

## 第 V 部「付録」

### 付録 A「SQL\*Loader の構文図」

この付録では、SQL\*Loader の構文図について説明します。

### 付録 B「DB2/DXT ユーザーに対する注意事項」

この付録では、SQL\*Loader で使用するデータ定義言語（DDL）の構文と、DB2 ロード・ユーティリティの制御ファイルで使用する DDL の構文の違いについて説明します。DB2 ロード・ユーティリティに対する SQL\*Loader の拡張機能、DB2 の RESUME オプション、互換性を維持するためのオプションおよび SQL\*Loader に関する制限事項について説明します。

### 付録 C「バックス正規形構文」

この付録では、テキストによる構文図の説明に使用される、バックス正規形（BNF）の変形の記号および表記規則について説明します。

## 関連文書

詳細は、次の Oracle ドキュメントを参照してください。

特に Oracle9i ドキュメント・セットの次のドキュメントを参照してください。

- 『Oracle9i データベース概要』
- 『Oracle9i SQL リファレンス』
- 『Oracle9i データベース管理者ガイド』

ドキュメント・セットの多くのマニュアルで、Oracle データベース・サーバーのインストール時にデフォルトとしてインストールされるシード・データベースのサンプル・スキーマを使用しています。サンプル・スキーマの作成方法および使用方法の詳細は、『Oracle9i サンプル・スキーマ』を参照してください。

リリース・ノート、インストール・マニュアル、ホワイト・ペーパーまたはその他の関連文書は、OTN-J (Oracle Technology Network Japan) に接続すれば、無償でダウンロードできます。OTN-J を使用するには、オンラインでの登録が必要です。次の URL で登録できます。

<http://otn.oracle.co.jp/membership/>

すでに OTN-J のユーザー名およびパスワードを取得済であれば、次の OTN-J Web サイトの文書セクションに直接接続できます。

<http://otn.oracle.co.jp/document/>

## 表記規則

この項では、このマニュアルの本文およびコード例で使用される表記規則について説明します。この項の内容は次のとおりです。

- [本文中の表記規則](#)
- [コード例中の表記規則](#)

本文中の表記規則

本文では、特別な用語をより迅速に識別するために、各種の表記規則を使用します。次の表に、それらの表記規則を説明し、その使用例を示します。

規則	意味	例
太字	太字は、本文中で定義されている用語または用語集に記載されている用語（あるいはその両方）を示します。	この句を指定すると、 <b>索引構成表</b> が作成されます。
固定幅フォントの大文字	固定幅フォントの大文字は、システムが提供する要素を示します。このような要素には、パラメータ、権限、データ型、Recovery Manager キーワード、SQL キーワード、SQL*Plus またはユーティリティ・コマンド、パッケージおよびメソッドが含まれます。また、システムが提供する列名、データベース・オブジェクト、データベース構造、ユーザー名およびロールも含まれます。	NUMBER 列に対してのみに、この句を指定できます。  BACKUP コマンドを使用して、データベースのバックアップを取ることができます。  USER_TABLES データ・ディクショナリ・ビュー内の TABLE_NAME 列を問い合わせます。  DBMS_STATS.GENERATE_STATS プロシージャを使用します。
固定幅フォントの小文字	固定幅フォントの小文字は、実行可能ファイル、ファイル名、ディレクトリ名およびユーザーが提供する要素のサンプルを示します。このような要素には、コンピュータ名、データベース名、ネット・サービス名および接続識別子が含まれます。また、ユーザーが提供するデータベース・オブジェクトとデータベース構造、列名、パッケージ、クラス、ユーザー名、ロール、プログラム・ユニットおよびパラメータの値も含まれます。  <b>注意：</b> 大文字と小文字を組み合わせて使用するプログラム要素もあります。これらの要素は、記載されているとおりに入力してください。	sqlplus と入力して、SQL*Plus をオープンします。  パスワードは、orapwd ファイルで指定します。  /disk1/oracle/dbs ディレクトリ内のデータ・ファイルおよび制御ファイルのバックアップを取ります。  hr.departments 表には、department_id 列、department_name 列および location_id 列があります。  QUERY_REWRITE_ENABLED 初期化パラメータを true に設定します。  oe ユーザーとして接続します。  JRepUtil クラスが次のメソッドを実装します。  parallel_clause を指定できます。  Uold_release.SQL を実行します。ここで、old_release とはアップグレード前にインストールしたリリースを示します。
固定幅フォントの小文字のイタリック	固定幅フォントの小文字のイタリックは、プレースホルダまたは変数を示します。	

コード例中の表記規則

コード例では、SQL、PL/SQL、SQL\*Plus または他のコマンドライン文を説明します。コード例は、固定幅フォントで表示され、次の例に示すとおり通常のテキストと区別されます。

```
SELECT username FROM dba_users WHERE username = 'MIGRATE';
```

次の表に、コード例で使用する表記規則を説明し、その使用例を示します。

規則	意味	例
[ ]	大カッコは、任意に選択する 1 つ以上の項目を囲みます。大カッコは、入力しないでください。	DECIMAL (digits [ , precision ])
{ }	中カッコは、2 つ以上の項目を囲み、そのうちの 1 つの項目は必須です。中カッコは、入力しないでください。	{ENABLE   DISABLE}
	縦線は、大カッコまたは中カッコ内の 2 つ以上のオプションの選択項目を表します。オプションのうちの 1 つを入力します。縦線は、入力しないでください。	{ENABLE   DISABLE} [COMPRESS   NOCOMPRESS]
...	水平の省略記号は、次のいずれかを示します。 <ul style="list-style-type: none"><li>■ 例に直接関連しないコードの一部が省略されている。</li><li>■ コードの一部を繰り返すことができる。</li></ul>	CREATE TABLE ... AS subquery;  SELECT col1, col2, ... , coln FROM employees;
. . . .	垂直の省略記号は、例に直接関連しない複数の行が省略されていることを示します。	SQL> SELECT NAME FROM V\$DATAFILE; NAME ----- /fs1/dbs/tbs_01.dbf /fs1/dbs/tbs_02.dbf . . . /fs1/dbs/tbs_09.dbf 9 rows selected.
その他の句読点	大カッコ、中カッコ、縦線および省略記号以外の句読点は、表示されているとおりに入力する必要があります。	acctbal NUMBER(11,2); acct        CONSTANT NUMBER(4) := 3;
イタリック体	イタリック体は、特定の値を指定する必要があるプレースホルダや変数を示します。	CONNECT SYSTEM/system_password DB_NAME = database_name

規則	意味	例
大文字	大文字は、システムが提供する要素を示します。これらの用語は、ユーザー定義の用語と区別するために大文字で示されます。用語が大カッコ内にかぎり、表示されているとおりの順序および綴りで入力します。ただし、これらの用語は大 / 小文字が区別がされないため、小文字でも入力できます。	<pre>SELECT last_name, employee_id FROM employees;  SELECT * FROM USER_TABLES;  DROP TABLE hr.employees;</pre>
小文字	小文字は、ユーザー定義のプログラム要素を示します。たとえば、表名、列名、ファイル名などです。 <b>注意:</b> 大文字と小文字を組み合わせて使用するプログラム要素もあります。これらの要素は、記載されているとおりに入力してください。	<pre>SELECT last_name, employee_id FROM employees;  sqlplus hr/hr  CREATE USER mjjones IDENTIFIED BY ty3MU9;</pre>



---

# データベース・ユーティリティの新機能

ここでは、Oracle9i データベース・ユーティリティの新機能について説明します。また、各機能の参照先も示します。今回のリリースへアップグレードする場合のために、Oracle8i で導入された機能についても説明します。

この章の内容は、次のとおりです。

- リリース 2 (9.2) での Oracle9i ユーティリティの新機能
- リリース 1 (9.0.1) での Oracle9i ユーティリティの新機能
- Oracle8i ユーティリティの新機能

## リリース 2 (9.2) での Oracle9i ユーティリティの新機能

この項では、Oracle9i リリース 2 (9.2) で導入された新機能および拡張機能について説明します。

### エクスポート・ユーティリティおよびインポート・ユーティリティ

次に、エクスポート・ユーティリティおよびインポート・ユーティリティの新機能および拡張機能を示します。

- エクスポート・ユーティリティ用の新しい `OBJECT_CONSISTENT` パラメータ。これを使用すると、オブジェクトがパーティション化されている場合でも、読み込み専用トランザクションに各オブジェクトをエクスポートできます。詳細は、1-25 ページの「[OBJECT\\_CONSISTENT](#)」を参照してください。
- インポート・ユーティリティ用の新しい `STREAMS_CONFIGURATION` パラメータ。これを使用すると、エクスポート・ダンプ・ファイル内に存在するすべての一般的な Streams メタデータをインポートできます。詳細は、2-29 ページの「[STREAMS\\_CONFIGURATION](#)」を参照してください。
- インポート・ユーティリティ用の新しい `STREAMS_INSTANTIATION` パラメータ。これを使用すると、エクスポート・ダンプ・ファイル内に存在する Streams インスタンスエーション・メタデータをインポートできます。詳細は、2-30 ページの「[STREAMS\\_INSTANTIATION](#)」を参照してください。

### SQL\*Loader ユーティリティ

次に、SQL\*Loader の新機能および拡張機能を示します。

- 新しい日付キャッシュ機能。これによって、入力データ内に多数の重複する日付値が存在する場合に実際に実行される日付変換の回数が減ります。そのため、ダイレクト・パス・ロード中のパフォーマンスが向上します。詳細は、9-21 ページの「[日付キャッシュの値の指定](#)」を参照してください。
- 1 つ以上のユーザー定義コンストラクタの作成による、デフォルトの属性値コンストラクタの上書き。詳細は、7-8 ページの「[ユーザー定義コンストラクタを使用した列オブジェクトのロード](#)」を参照してください。

### 外部表

次に、外部表の新機能および拡張機能を示します。

- 新しい日付キャッシュ機能。これによって、入力データ内に多数の重複する日付値が存在する場合に実際に実行される日付変換の回数が減ります。そのため、ダイレクト・パス・ロード中のパフォーマンスが向上します。詳細は、11-6 ページの「[外部表使用時のパフォーマンスのヒント](#)」を参照してください。

## DBNEWID ユーティリティ

DBNEWID ユーティリティは、オペレーショナル・データベースの DBID および DBNAME を変更可能な新しいデータベース・ユーティリティです。詳細は、[第 14 章「DBNEWID ユーティリティ」](#)を参照してください。

## メタデータ API

メタデータ API によって、次の作業を集中的、かつ単純で柔軟な方法で実行できます。

- XML または作成用 DDL のいずれかとしての、データベース・オブジェクト（メタデータ）の完全な定義の抽出
- 業界標準の Extensible Stylesheet Language Transformation（XSLT）を介したメタデータの変換
- SQL DDL の生成によるデータベース・オブジェクトの再作成

メタデータ API は、Oracle9i リリース 1 (9.0.1) から使用可能でしたが、別のマニュアルで説明されていました。リリース 2 (9.2) からは、このマニュアルで説明されます。詳細は、[第 15 章「メタデータ API の使用」](#)を参照してください。

## リリース 1 (9.0.1) での Oracle9i ユーティリティの新機能

この項では、Oracle9i で導入された新機能および拡張機能について説明します。

### エクスポート・ユーティリティおよびインポート・ユーティリティ

次に、エクスポート・ユーティリティおよびインポート・ユーティリティの新機能および拡張機能を示します。

- 計算済オプティマイザ統計の拡張エクスポート / インポート機能。詳細は、次の項を参照してください。
  - エクスポート・ユーティリティでのこのパラメータの使用方法については、1-28 ページの「[STATISTICS](#)」を参照してください。
  - インポート・ユーティリティでのこのパラメータの使用方法については、2-29 ページの「[STATISTICS](#)」を参照してください。
  - 2-70 ページの「[統計情報のインポート](#)」を参照してください。
- 新しいパラメータ RESUMABLE、RESUMABLE\_NAME、RESUMABLE\_TIMEOUT、FLASHBACK\_SCN および FLASHBACK\_TIME の追加。詳細は、1-14 ページの「[エクスポート・パラメータ](#)」および 2-15 ページの「[インポート・パラメータ](#)」を参照してください。
- エクスポート・モードによる、表領域内のすべての表のダンプ。詳細は、1-31 ページの「[TABLESPACES](#)」を参照してください。

- エクスポート中の表名のパターン一致。詳細は、1-29 ページの「[TABLES](#)」を参照してください。
- インポート中の表名のパターン一致。詳細は、2-30 ページの「[TABLES](#)」を参照してください。
- インポート時のキャラクタ・セット変換の削減。詳細は、2-57 ページの「[キャラクタ・セット変換](#)」を参照してください。

## SQL\*Loader ユーティリティ

次に、SQL\*Loader の新機能および拡張機能を示します。

- プラットフォーム間で整数データ型および ZONED 型 / PACKED 型の 10 進データ型を正常にロード可能。SQL\*Loader を使用して、次のことが実行可能です。
  - バイト順序がターゲット・プラットフォームとは異なるプラットフォームで作成された 2 進整数データのロード
  - バイト順序がターゲット・プラットフォームとは異なるプラットフォームで作成されたバイナリ浮動小数点データのロード（ソース・システムおよびターゲット・システムで使用されている浮動小数点形式が同じ場合）
  - バイト単位での 2 進整数のサイズ指定、およびターゲット・プラットフォーム固有の整数サイズに依存しないロード
  - 整数値の符号付きまたは符号なしの指定
  - IBM 形式でエンコードされた EBCDIC ベースの ZONED 型 10 進データまたは PACKED 型 10 進データの指定

これらの拡張機能の詳細は、次の項を参照してください。

- 6-8 ページの「[INTEGER\(n\)](#)」
- 6-11 ページの「[DECIMAL](#)」
- 6-10 ページの「[ZONED](#)」
- 6-35 ページの「[異なるプラットフォーム間でのデータのロード](#)」
- XML 列のロードのサポート。詳細は、7-18 ページの「[LOB のロード](#)」を参照してください。
- サブタイプを持つオブジェクト表のロードのサポート。詳細は、7-14 ページの「[サブタイプを使用したオブジェクト表のロード](#)」を参照してください。
- 導出サブタイプを持つ列オブジェクトのロードのサポート。詳細は、7-5 ページの「[導出サブタイプを使用した列オブジェクトのロード](#)」を参照してください。
- SQL\*Loader での Unicode のサポート。このサポートで次のことが実行可能です。
  - SQL\*Loader データ・ファイルでの UTF16 キャラクタ・セットの使用

- SQL\*Loader 制御ファイルでの文字長セマンティクスのサポート
- SQL\*Loader を使用した、NCHAR、NVARCHAR2 および NCLOB データ型の列へのデータのロード（各国語キャラクタ・セットが AL16UTF16 の場合）
- SQL\*Loader データ・ファイルに対するバイト順序（ビッグ・エンディアンまたはリトル・エンディアン）の指定

**参照：** 次の項を参照してください。

- 5-16 ページの「異なる文字コード体系の処理」
- 6-7 ページの「SQL\*Loader のデータ型」
- 6-36 ページの「バイト順序」
- ANSI SQL 規格ドキュメントでの指定に対応した日時データ型および期間データ型のサポート。このサポートで次のことが実行可能です。
  - SQL\*Loader の従来型パス・モードおよびダイレクト・パス・モードの両方に対する日時データ型および期間データ型のロード
  - SQL\*Loader クライアントとデータベース・サーバーとの間の日時データ型変換および期間データ型変換の実行
  - ダイレクト・パス API を使用した、日時データ型および期間データ型のロード
 詳細は、6-15 ページの「日時データ型および期間データ型」を参照してください。
- 2 進整数 SMALLINT および INTEGER(n) に対して UNSIGNED パラメータを指定可能。詳細は、6-9 ページの「SMALLINT」および 6-8 ページの「INTEGER(n)」を参照してください。
- INTEGER パラメータ（たとえば、INTEGER(n)）に長さ指定を適用可能。詳細は、6-8 ページの「INTEGER(n)」を参照してください。
- 可能な場合、列配列をストリーム・バッファに変換し、パラレルでストリーム・バッファ・ロードを行うダイレクト・パス・ロードに対するマルチスレッド・ロード。詳細は、9-23 ページの「複数 CPU システムのダイレクト・パス・ロードの最適化」を参照してください。
- 新しい COLUMNARRAYROWS パラメータ（ダイレクト・パス・ロードでの列配列の行数に対する値の指定に使用）および新しい STREAMSIZE パラメータ（ダイレクト・パス・ストリーム・バッファのサイズの指定に使用）。詳細は、9-21 ページの「列配列の行数およびストリーム・バッファ・サイズの指定」を参照してください。
- 再開可能な領域割当てを有効または無効にする RESUMABLE、RESUMABLE\_NAME および RESUMABLE\_TIMEOUT パラメータの追加。詳細は、4-4 ページの「コマンドライン・パラメータ」を参照してください。

## 外部表

Oracle9i の外部表機能は、既存の SQL\*Loader 機能を補足します。この機能によって、データベースに表がある場合と同様に、外部ソースのデータにアクセスできます。

**参照：** 次の章を参照してください。

- [第 11 章「外部表の概要」](#)
- [第 12 章「外部表アクセス・パラメータ」](#)

## DBVERIFY ユーティリティ

DBVERIFY ユーティリティには、検査のための表セグメントまたは索引セグメントを指定できる追加のコマンドライン・インタフェースがあります。検査中のセグメントに行連鎖ポイントがあることを確認します。詳細は、13-4 ページの「[DBVERIFY を使用したセグメントの検査](#)」を参照してください。

## Oracle8i ユーティリティの新機能

この項で説明する Oracle8i の新機能を使用すると、データの転送、メンテナンスおよび管理を最適化できます。この項で説明する機能は、リリース 8.1.5、リリース 8.1.6 およびリリース 8.1.7 で追加された機能です。

### エクスポート・ユーティリティ

追加または拡張されたエクスポート機能は、次のとおりです。

- サブパーティションのエクスポート。詳細は、1-13 ページの「[表レベル・エクスポートおよびパーティション・レベル・エクスポート](#)」を参照してください。
- エクスポート・コマンドに複数のダンプ・ファイルが指定可能。パラメータの詳細は、1-22 ページの「[FILE](#)」および 1-22 ページの「[FILESIZE](#)」を参照してください。
- エクスポート・ユーティリティで表のアンロードに使用される SELECT 文に対して、条件式を指定可能。詳細は、1-26 ページの「[QUERY](#)」を参照してください。
- 各テーブル・ボリュームのエクスポート・ファイルに指定できる最大バイト数が増加。詳細は、1-33 ページの「[VOLSIZE](#)」を参照してください。
- LOB およびオブジェクトを含む表のエクスポートが可能（コマンドラインでダイレクト・パスが指定されている場合を含む）。詳細は、1-53 ページの「[ダイレクト・パス・エクスポートの起動](#)」を参照してください。
- インポート時にオブティマイザ統計情報を再計算するかわりに、計算済オブティマイザ統計情報のエクスポートおよびインポートが可能（この機能は、特定のエクスポートおよび特定の表に対してのみ使用可能）。詳細は、1-28 ページの「[STATISTICS](#)」を参照してください。

- ドメイン索引の開発者は、ODCIIndex インタフェースで新しい ODCIIndexGetMetadata 方式を使用して、索引と対応付けられたアプリケーション固有のメタデータのエクスポートが可能。詳細は、『Oracle8i データ・カートリッジ開発者ガイド』を参照してください。
- トランスポータブル表領域のメタデータのエクスポート。詳細は、1-31 ページの「[TRANSPORT\\_TABLESPACE](#)」を参照してください。

## インポート・ユーティリティ

追加または拡張されたインポート機能は、次のとおりです。

- サブパーティションのインポート。詳細は、2-51 ページの「[表レベル・インポートおよびパーティション・レベル・インポート](#)」を参照してください。
- インポート・コマンドに複数のダンプ・ファイルが指定可能。パラメータの詳細は、2-22 ページの「[FILE](#)」および 2-22 ページの「[FILESIZE](#)」を参照してください。
- インポート・パラメータ TOID\_NOVALIDATE を使用して、オブジェクト型（通常、カートリッジのインストールによって作成された型）の妥当性チェックを省略可能。詳細は、2-32 ページの「[TOID\\_NOVALIDATE](#)」を参照してください。
- 各テープ・ボリュームのエクスポート・ファイルに指定できる最大バイト数が増加。詳細は、2-35 ページの「[VOLSIZE](#)」を参照してください。
- ファイングレイン・アクセス・コントロールに対するサポート。詳細は、2-58 ページの「[データベース・オブジェクトのインポートに関する考慮点](#)」を参照してください。
- インポート時にオプティマイザ統計情報を再計算するかわりに、計算済オプティマイザ統計情報のエクスポートおよびインポートが可能（この機能は、特定のエクスポートおよび特定の表に対してのみ使用可能）。詳細は、2-29 ページの「[STATISTICS](#)」を参照してください。
- トランスポータブル表領域のメタデータのインポート。詳細は、2-34 ページの「[TRANSPORT\\_TABLESPACE](#)」を参照してください。

## SQL\*Loader ユーティリティ

追加または拡張された SQL\*Loader 機能は、次のとおりです。

- CONTINUEIF THIS および CONTINUEIF NEXT とともに使用する PRESERVE パラメータの追加。

PRESERVE パラメータを使用しない場合、継続フィールドは、論理レコードの作成時にすべての物理レコードから削除されます。継続フィールドが読み込まれるまでは、データは同じ論理レコードに連結します。

PRESERVE パラメータを使用した場合、継続フィールドは、論理レコードの作成時にもすべての物理レコードに保持されます。

詳細は、5-27 ページの「[CONTINUEIF を使用した論理レコードの作成](#)」を参照してください。

- 空白のみを含む DATE フィールドを NULL フィールドとしてロード可能。これによるエラーは発生しません。詳細は、6-15 ページの「[日時データ型および期間データ型](#)」を参照してください。
- リリース 8.1.5 での、特定 DDL 句の動作および制限事項の変更によるオブジェクトのサポート。この機能の詳細は、[第 7 章「オブジェクト、LOB およびコレクションのロード」](#)を参照してください。さらに、次の項の説明も参照してください。
  - 6-6 ページの「[FILLER フィールドの指定](#)」
  - 6-32 ページの「[WHEN、NULLIF および DEFAULTIF 句の使用](#)」
  - 6-48 ページの「[フィールドへの SQL 演算子の適用](#)」



# 第I部

---

## エクスポートおよびインポート

第I部では、Oracleのエクスポート・ユーティリティおよびインポート・ユーティリティについて説明します。この部に含まれる章は次のとおりです。

### 第1章「エクスポート」

この章では、エクスポート・ユーティリティを使用して Oracle データベースから転送可能なファイルヘデータを書き込む方法を説明します。エクスポートの概要、エクスポート・モード、対話方式とコマンドライン方式、パラメータの指定およびエクスポート・オブジェクトのサポートについて説明します。エクスポート・セッションの例も示します。

### 第2章「インポート」

この章では、インポート・ユーティリティを使用してエクスポート・ファイルのデータを Oracle データベースへ読み込む方法を説明します。インポートの概要、対話方式とコマンドライン方式、パラメータの指定およびインポート・オブジェクトのサポートについて説明します。インポート・セッションの例も示します。



---

# エクスポート

この章では、エクスポート・ユーティリティを使用して Oracle データベースのデータをバイナリ形式でオペレーティング・システム・ファイルに書き込む方法について説明します。書き込んだファイルは、データベース外に格納されるため、インポート・ユーティリティ（第2章を参照）を使用して他の Oracle データベースに読み込むことができます。

この章の内容は、次のとおりです。

- エクスポート・ユーティリティとは
- エクスポート・ユーティリティを使用する前に
- エクスポート・ユーティリティの起動
- エクスポート・モード
- オンライン・ヘルプの利用
- エクスポート・パラメータ
- エクスポート・セッションの例
- 対話方式の使用
- 警告、エラーおよび完了メッセージ
- 終了コードによる結果の検査と表示
- 従来型パス・エクスポートおよびダイレクト・パス・エクスポート
- ダイレクト・パス・エクスポートの起動
- ネットワークに関する考慮点
- キャラクタ・セットおよびグローバリゼーション・サポートに関する考慮点
- インスタンス親和性とエクスポート
- データベース・オブジェクトのエクスポートに関する考慮点
- トランスポートابل表領域

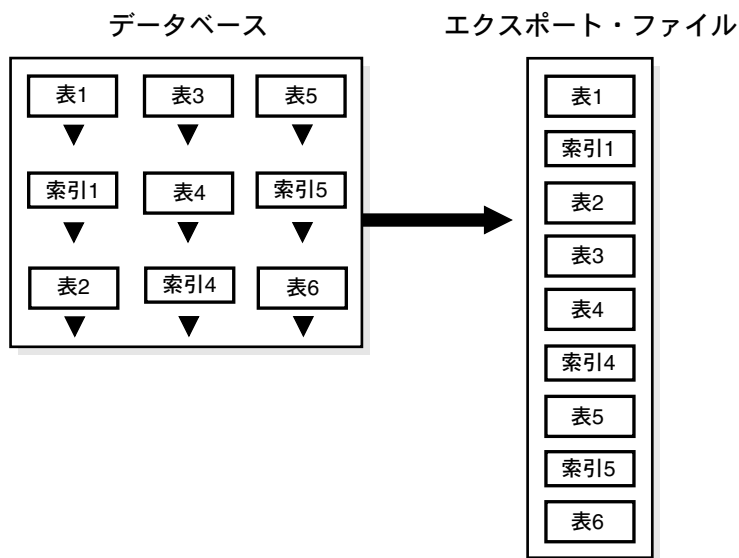
- 
- 読込み専用データベースからのエクスポート
  - エクスポート・ユーティリティおよびインポート・ユーティリティを使用したデータベース移行のパーティション化
  - リリースおよびバージョンが異なるエクスポート・ユーティリティの使用方法

## エクスポート・ユーティリティとは

エクスポート・ユーティリティを使用すると、異なるハードウェア構成およびソフトウェア構成のプラットフォーム上にある Oracle データベース間で、データ・オブジェクトの転送が簡単にできます。

Oracle データベースに対してエクスポートを実行すると、まずオブジェクト（表など）が抽出され、次にそれに関連するオブジェクト（索引、コメント、権限など）が抽出されます。抽出されたデータは、[図 1-1](#) に示すとおり、エクスポート・ファイルに書き込まれます。

図 1-1 データベースのエクスポート



エクスポート・ファイルは、通常、ディスクまたはテープにあるバイナリ形式の Oracle ダンプ・ファイルです。ダンプ・ファイルは、FTP を使用して別サイトに転送、または物理的に移送（テープの場合）できます。そのため、インポート・ユーティリティでエクスポート・ファイルを使用することで、ネットワークで接続されていないシステム上のデータベース間でデータを転送できます。また、標準のバックアップ手順以外のバックアップとしても使用できます。

エクスポート・ダンプ・ファイルは、Oracle のインポート・ユーティリティを使用した場合のみ読み込みが可能です。ダンプ・ファイルの作成に使用したエクスポート・ユーティリティより下位のバージョンのインポート・ユーティリティは使用できません。

また、実際にインポートを実行せずにエクスポート・ファイルの内容を表示することもできます。この場合は、インポート・ユーティリティの `SHOW` パラメータを使用します。詳細は、2-28 ページの「[SHOW](#)」を参照してください。

ASCII 固定形式ファイルまたは区切りファイルからデータをロードするには、`SQL*Loader` ユーティリティを使用します。

### 参照：

- 1-63 ページの「[リリースおよびバージョンが異なるエクスポート・ユーティリティの使用方法](#)」を参照してください。
- インポート・ユーティリティの詳細は、[第 2 章](#)を参照してください。
- `SQL*Loader` ユーティリティの詳細は、[第 II 部](#)を参照してください。
- エクスポート・ユーティリティおよびインポート・ユーティリティを使用した、オフライン・インスタンスエーションなどの Oracle Advanced Replication の機能については、『Oracle9i アドバンスド・レプリケーション』を参照してください。

## エクスポート・ユーティリティを使用する前に

エクスポート・ユーティリティを使用する前に、次のことを行う必要があります。

- `catexp.sql` または `catalog.sql` スクリプトの実行
- エクスポート・ファイルの書き込み先であるディスクまたはテープに十分な記憶域があることの確認
- 必要なアクセス権限を所有していることの確認

## catexp.sql または catalog.sql の実行

エクスポート・ユーティリティを使用するには、データベースを作成した後で、スクリプト `catexp.sql` または (`catexp.sql` を実行する) `catalog.sql` を実行する必要があります。

---

**注意：** スクリプト・ファイルの実際の名前は、システムによって異なります。スクリプトのファイル名およびそれらの実行方法については、ご使用のオペレーティング・システム固有の Oracle マニュアルを参照してください。

---

データベースに対して、`catexp.sql` または `catalog.sql` を実行するのは 1 回のみです。エクスポート・ユーティリティの実行前に、これらのスクリプトを再実行する必要はありません。スクリプトを実行すると、次の処理が行われ、データベースはエクスポートに備えて調整されます。

- データ・ディレクトリへの必要なエクスポート・ビューの作成
- EXP\_FULL\_DATABASE ロールの作成
- EXP\_FULL\_DATABASE ロールへのすべての必要な権限の割当て
- DBA ロールへの EXP\_FULL\_DATABASE の割当て
- インストールされている catexp.sql のバージョンの記録

## 十分なディスク領域の確認

エクスポートを実行する前に、エクスポート・ファイルの書き込み先であるディスク上またはテープ上に、十分な記憶領域があることを確認してください。十分な領域がない場合は、書き込み失敗というエラーでエクスポートの処理が中止されます。

表サイズを使用して、必要な最大容量を見積もることができます。表サイズは、Oracle データ・ディクショナリの USER\_SEGMENTS ビューで参照できます。次の問合せを行うと、すべての表に関するディスクの使用状況が表示されます。

```
SELECT SUM(BYTES) FROM USER_SEGMENTS WHERE SEGMENT_TYPE='TABLE';
```

問合せの結果には、LOB（ラージ・オブジェクト）列、VARRAY 列またはパーティション表に格納されているデータに使用されているディスク領域は含まれません。

**参照：** ディクショナリ・ビューの詳細は、『Oracle9i データベース・リファレンス』を参照してください。

## アクセス権限の確認

エクスポート・ユーティリティを使用するには、Oracle データベースに対する CREATE SESSION 権限が必要です。別のユーザーが所有する表をエクスポートする場合は、EXP\_FULL\_DATABASE ロールを使用可能にしておく必要があります。このロールは、すべての DBA に付与されています。

EXP\_FULL\_DATABASE ロールに含まれるシステム権限がない場合、別のユーザーのスキーマに格納されているオブジェクトをエクスポートすることはできません。シノニムを作成しても、別のユーザーのスキーマの表はエクスポートできません。

次のスキーマ名は予約済のため、エクスポートでは処理されません。

- ORDSYS
- MDSYS
- CTXSYS
- ORDPLUGINS
- LBACSYS

## エクスポート・ユーティリティの起動

次のいずれかの方法を使用して、エクスポート・ユーティリティの起動およびパラメータの指定ができます。

- コマンドライン
- 対話方式のエクスポート・プロンプト
- パラメータ・ファイル

これらのいずれかの方法でエクスポート・ユーティリティを起動する前に、使用可能なパラメータについての説明を必ず読んでください。詳細は、1-14 ページの「[エクスポート・パラメータ](#)」を参照してください。

### コマンドライン

次の構文を使用して、すべての有効なパラメータおよびその値をコマンドラインから指定できます。

```
exp username/password PARAMETER=value
```

または

```
exp username/password PARAMETER=(value1,value2,...,valuen)
```

システムでのコマンドラインの最大長を超える数のパラメータは指定できません。

### 対話方式のエクスポート・プロンプト

エクスポート・ユーティリティのプロンプトで各パラメータ値を入力する場合は、次の構文を使用して対話方式モードでエクスポート・ユーティリティを起動します。

```
exp username/password
```

一般的に使用される、値の入力が必要なパラメータがエクスポート・ユーティリティによって表示されます。この方法には下位互換性がありますが、他の方法よりも機能的に劣るためお勧めしません。詳細は、1-44 ページの「[対話方式の使用](#)」を参照してください。

### パラメータ・ファイル

パラメータ・ファイルに、すべての有効なパラメータおよびその値を指定できます。パラメータを1つのファイルに格納することによって、パラメータを簡単に変更または再利用できるため、エクスポート・ユーティリティの起動方法としてパラメータを1つのファイルに格納することをお勧めします。データベースごとに別のパラメータを使用する場合は、複数のパラメータ・ファイルを作成できます。



フラット・ファイル用のテキスト・エディタを使用してパラメータ・ファイルを作成します。コマンドライン・オプション `PARFILE=filename` を指定すると、エクスポート・ユーティリティは、コマンドラインからではなく指定されたファイルからパラメータを読み込みます。次に例を示します。

```
exp PARFILE=filename
exp username/password PARFILE=filename
```

最初の例では、コマンドラインで `username/password` を指定しないで、パラメータ・ファイルで指定する方法が示されています。ただし、セキュリティ上の理由から、この方法はお薦めしません。

パラメータ・ファイルは、次のいずれかの構文を使用して指定します。

```
PARAMETER=value
PARAMETER=(value)
PARAMETER=(value1, value2, ...)
```

次に、パラメータ・ファイル内のリストの一部を示します。

```
FULL=y
FILE=dba.imp
GRANTS=y
INDEXES=y
CONSISTENT=y
```

---

---

**注意：** パラメータ・ファイルの最大サイズは、オペレーティング・システムによって制限されます。また、パラメータ・ファイル名はオペレーティング・システムのネーミング規則に従います。詳細は、ご使用のオペレーティング・システム固有の Oracle マニュアルを参照してください。

---

---

シャープ（＃）記号を使用すると、パラメータ・ファイルにコメントを追加できます。シャープ（＃）の右側にある文字はすべて無視されます。

コマンドラインでのパラメータの入力と同時にパラメータ・ファイルを指定できます。実際、パラメータ・ファイルとコマンドラインの両方に同じパラメータを指定することもできます。コマンドラインでの `PARFILE` パラメータと他のパラメータの位置によって、優先されるパラメータが決まります。たとえば、パラメータ・ファイル `params.dat` でパラメータ `INDEXES=y` を指定し、エクスポート・ユーティリティを次のコマンドで起動するとします。

```
exp username/password PARFILE=params.dat INDEXES=n
```

この場合、`INDEXES=n` は `PARFILE=params.dat` の後にあるので、パラメータ・ファイルに指定されている `INDEXES` パラメータの値は、`INDEXES=n` によって上書きされます。

### 参照：

- エクスポート・パラメータの詳細は、1-14 ページの「[エクスポート・パラメータ](#)」を参照してください。
- リモートのデータベースからのエクスポートの指定方法の詳細は、1-54 ページの「[Oracle Net でのエクスポートおよびインポート](#)」を参照してください。

## SYSDBA でのエクスポート・ユーティリティの起動

SYSDBA は内部的に使用され、一般ユーザーとは異なる特別な機能を持ちます。そのため、通常、次の場合以外は、エクスポート・ユーティリティを SYSDBA で起動する必要はありません。

- オラクル社カスタマ・サポート・センターから要求された場合
- トランスポータブル表領域を使用している場合（1-61 ページの「[トランスポータブル表領域](#)」を参照）

SYSDBA でエクスポート・ユーティリティを起動するには、必要なパラメータまたはパラメータ・ファイル名を追加して次の構文を使用します。

```
exp \ 'username/password AS SYSDBA\ '
```

オプションで、インスタンス名の指定もできます。

```
exp \ 'username/password@instance AS SYSDBA\ '
```

ユーザー名またはパスワードを指定しないと、入力するように要求されます。

この例では、接続文字列全体が一重引用符およびバックスラッシュで囲まれています。これは、文字列 AS SYSDBA に空白が含まれるため、ほとんどのオペレーティング・システムで、文字列全体を一重引用符で囲むか、なんらかの方法でリテラルとしてマークする必要があります。オペレーティング・システムによっては、コマンドラインの一重引用符自体をエスケープする必要がある場合もあります。この例では、バックスラッシュがエスケープ文字として使用されます。バックスラッシュがない場合、エクスポート・ユーティリティで使用するコマンドライン解析機能で一重引用符として認識されるため、一重引用符はエクスポート・ユーティリティを起動する前に削除されてしまいます。

システムの特許文字および予約文字の詳細は、ご使用のオペレーティング・システム固有の Oracle マニュアルを参照してください。

エクスポート・ユーティリティで対話方式モードを使用する場合の詳細は、1-44 ページの「[対話方式の使用](#)」を参照してください。

# エクスポート・モード

エクスポート・ユーティリティには、次の 4 種類のモードがあります。

- 全データベース・モード
- ユーザー（所有者）・モード
- 表モード
- トランSPORTダブル表領域モード

表モードとユーザー・モードは、すべてのユーザーが使用できます。EXP\_FULL\_DATABASE ロールを持つユーザー（特権ユーザー）は、すべてのモードでエクスポートできます。表 1-1 に、各モードでエクスポートおよびインポートされるオブジェクトを示します。1-13 ページの「[処理上の制限事項](#)」も参照してください。

これらのモードのいずれかを指定するには、エクスポート・ユーティリティの起動時に、適切なパラメータ（FULL、OWNER、TABLES または TABLESPACES）を使用します。これらの各パラメータに対する構文の詳細は、1-14 ページの「[エクスポート・パラメータ](#)」を参照してください。

従来型パス・エクスポートまたはダイレクト・パス・エクスポートは、最初の 3 つのモードで使用できます。従来型パス・エクスポートとダイレクト・パス・エクスポートの違いは、1-51 ページの「[従来型パス・エクスポートおよびダイレクト・パス・エクスポート](#)」を参照してください。

**参照：**

- 『Oracle9i データベース管理者ガイド』を参照してください。
- トランSPORTダブル表領域の機能の詳細は、『Oracle9i データベース概要』を参照してください。

表 1-1 各モードでエクスポートおよびインポートされるオブジェクト

オブジェクト	表モード	ユーザー・モード	全データベース・モード	トランSPORTダブル表領域モード
分析クラスタ	No	Yes	Yes	No
分析表 / 統計	Yes	Yes	Yes	Yes
アプリケーション・コンテキスト	No	No	Yes	No
監査情報	Yes	Yes	Yes	No
B ツリー索引、ビットマップ索引、ドメイン・ファンクション索引	Yes <sup>1</sup>	Yes <sup>1</sup>	Yes	Yes

表 1-1 各モードでエクスポートおよびインポートされるオブジェクト（続き）

オブジェクト	表モード	ユーザー・モード	全データベース・モード	トランスポート ブル表領域 モード
クラスタ定義	No	Yes	Yes	Yes
列コメントおよび表コメント	Yes	Yes	Yes	Yes
データベース・リンク	No	Yes	Yes	No
デフォルト・ロール	No	No	Yes	No
ディメンション	No	Yes	Yes	No
ディレクトリ別名	No	No	Yes	No
外部表（データなし）	Yes	Yes	Yes	No
外部関数ライブラリ	No	Yes	Yes	No
表の所有者以外のユーザーが所有する索引	Yes（特権ユーザーのみ）	Yes	Yes	Yes
索引タイプ	No	Yes	Yes	No
Java リソースおよび Java クラス	No	Yes	Yes	No
ジョブ・キュー	No	Yes	Yes	No
ネストした表のデータ	Yes	Yes	Yes	Yes
オブジェクト権限	Yes（表および索引のみ）	Yes	Yes	Yes
表で使用するオブジェクト型定義	Yes	Yes	Yes	Yes
オブジェクト型	No	Yes	Yes	No
演算子	No	Yes	Yes	No
パスワード履歴	No	No	Yes	No
インスタンスの事後処理およびオブジェクト	No	No	Yes	No
スキーマの事後プロシージャ処理およびオブジェクト	No	Yes	Yes	No
表の事後処理	Yes	Yes	Yes	Yes

表 1-1 各モードでエクスポートおよびインポートされるオブジェクト（続き）

オブジェクト	表モード	ユーザー・モード	全データベース・モード	トランスポート ブル表領域 モード
表の事後プロシージャ処理およびオブジェクト	Yes	Yes	Yes	Yes
スキーマの事前プロシージャ・オブジェクトおよび処理	No	Yes	Yes	No
表の事前処理	Yes	Yes	Yes	Yes
表の事前プロシージャ処理	Yes	Yes	Yes	Yes
プライベート・シノニム	No	Yes	Yes	No
プロシージャ・オブジェクト	No	Yes	Yes	No
プロファイル	No	No	Yes	No
パブリック・シノニム	No	No	Yes	No
参照整合性制約	Yes	Yes	Yes	No
リフレッシュ・グループ	No	Yes	Yes	No
リソース・コスト	No	No	Yes	No
ロール権限	No	No	Yes	No
ロール	No	No	Yes	No
ロールバック・セグメント定義	No	No	Yes	No
表のセキュリティ・ポリシー	Yes	Yes	Yes	Yes
順序番号	No	Yes	Yes	No
スナップショット・ログ	No	Yes	Yes	No
スナップショットおよびマテリアライズド・ビュー	No	Yes	Yes	No
システム権限	No	No	Yes	No
表制約（主キー制約、一意制約、CHECK 制約）	Yes	Yes	Yes	Yes
表データ	Yes	Yes	Yes	No

表 1-1 各モードでエクスポートおよびインポートされるオブジェクト（続き）

オブジェクト	表モード	ユーザー・モード	全データベース・モード	トランスポート フル表領域 モード
表定義	Yes	Yes	Yes	Yes
表領域定義	No	No	Yes	No
表領域割当て制限	No	No	Yes	No
トリガー	Yes	Yes <sup>2</sup>	Yes <sup>3</sup>	Yes
他のユーザーが所有する トリガー	Yes（特権ユーザーのみ）	No	No	No
ユーザー定義	No	No	Yes	No
ユーザー・プロキシ	No	No	Yes	No
ユーザー・ビュー	No	Yes	Yes	No
ユーザー・ストアド・プロ シージャ、ユーザー・ ストアド・パッケージお よびユーザー・ストア ド・ファンクション	No	Yes	Yes	No

- <sup>1</sup> 非特権ユーザーがエクスポートおよびインポートできるのは、そのユーザー自身が所有する表に関する索引のみです。他のユーザーが所有する表に関する索引や、ユーザー自身が所有する表に関して他のユーザーが作成した索引はエクスポートできません。特権ユーザーは、エクスポートおよびインポート対象に指定したユーザーの表に関する索引が、表の所有者以外のユーザーが作成したものであっても、その索引をエクスポートおよびインポートできます。指定したユーザーが他のユーザーの表に関する索引を所有しているときは、エクスポートするユーザーのリストに表の所有者であるユーザーを指定しないかぎり、その索引はエクスポートされません。
- <sup>2</sup> 特権ユーザーも非特権ユーザーも、そのユーザー自身が所有するすべてのトリガーを（他のユーザーが所有する表に関するトリガーの場合でも）、エクスポートおよびインポートできます。
- <sup>3</sup> 全エクスポートでは、スキーマ SYS が所有するトリガーはエクスポートされません。SYS トリガーは、全インポートの前後のいずれかに手動で再作成する必要があります。SYS トリガーによってインポートの進行を妨げるような処理が定義されないように、SYS トリガーはインポートの後に再作成することをお薦めします。

## 表レベル・エクスポートおよびパーティション・レベル・エクスポート

表、パーティションおよびサブパーティションのエクスポートは、次のように実行できます。

- **表レベル・エクスポート**: 指定された表のすべてのデータをエクスポートします。
- **パーティション・レベル・エクスポート**: 指定されたソース・パーティションまたはサブパーティションのデータのみをエクスポートします。

どのモードの場合も、パーティション・データは、インポート時にパーティション単位またはサブパーティション単位で選択できる形式でエクスポートされます。

### 表レベル・エクスポート

表レベル・エクスポートでは、パーティション表または非パーティション表は、索引およびその他の表に依存するオブジェクトとともに全体的にエクスポートされます。パーティション表の場合は、すべてのパーティションおよびサブパーティションもエクスポートされます。ダイレクト・パス・エクスポートおよび従来型パス・エクスポートの両方とも、この点は同じです。表レベル・エクスポートは、いずれのエクスポート・モードでも実行できます。

### パーティション・レベル・エクスポート

パーティション・レベル・エクスポートでは、表の 1 つ以上のパーティションまたはサブパーティションを指定してエクスポートできます。パーティション・レベル・エクスポートは、表モードでのみ実行できます。

表レベル・エクスポートおよびパーティション・レベル・エクスポートの指定方法の詳細は、1-29 ページの「[TABLES](#)」を参照してください。

## 処理上の制限事項

エクスポート・ユーティリティおよびインポート・ユーティリティを使用してデータを処理する場合、次の制限が適用されます。

- Enterprise JavaBeans (EJB) を使用して作成された Java クラス、リソースおよびプロシージャは、エクスポート・ファイルには書き込まれません。
- RELY キーワードを使用して変更された制約は、エクスポートされると RELY 属性が失われます。
- 型定義が進化し、その進化した型を参照するデータがエクスポートされた場合、インポート・システムの型定義も同様に進化させる必要があります。

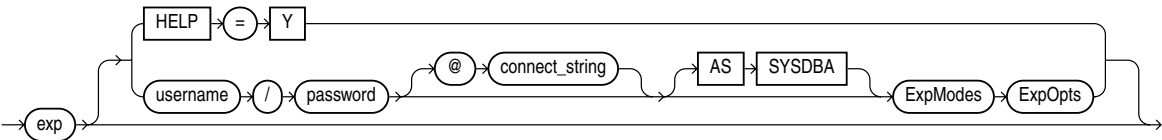
# オンライン・ヘルプの利用

エクスポート・ユーティリティには、オンライン・ヘルプが用意されています。ヘルプを起動するには、コマンドラインで `exp help=y` を入力します。

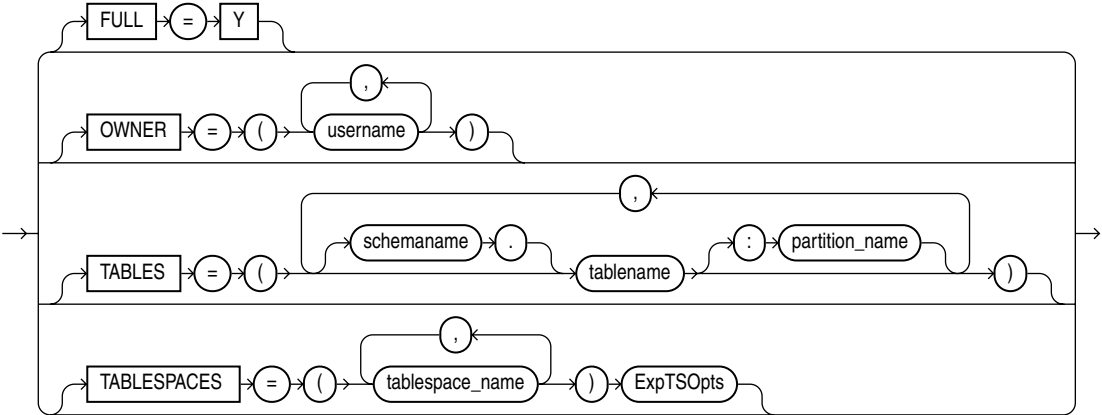
## エクスポート・パラメータ

パラメータ・ファイルまたはコマンドラインで指定できるパラメータの構文を次の図に示します。図の後に、各パラメータについて説明します。

### Export\_start

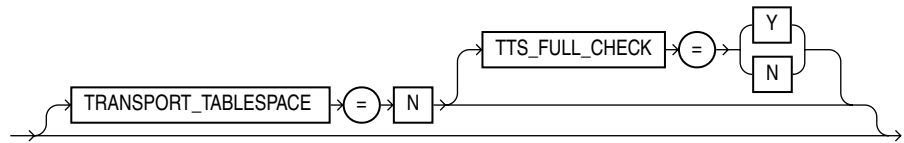


### ExpModes

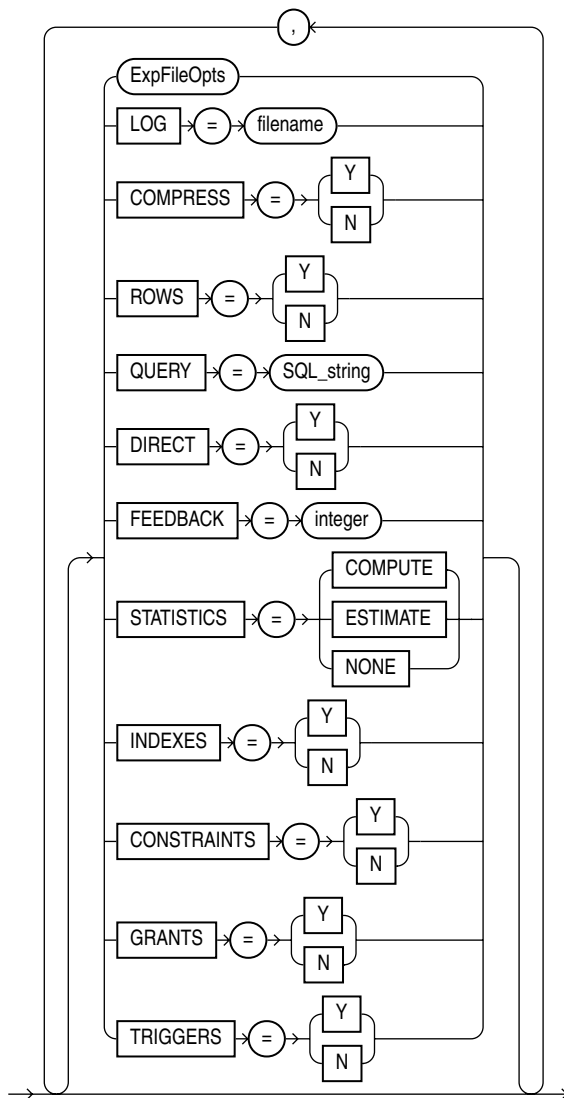




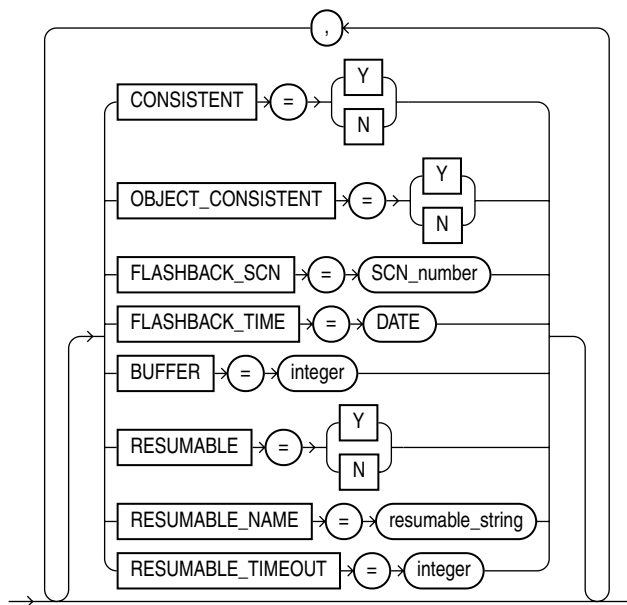
## ExpTSOpts (tablespaces\_spec)



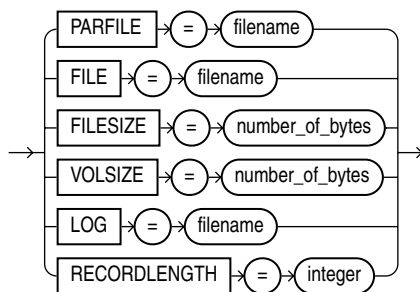
## ExpOpts



## ExpOpts\_continued



## ExpFileOpts



## BUFFER

デフォルト:オペレーティング・システムによって異なります。このパラメータのデフォルト値については、ご使用のオペレーティング・システム固有の Oracle マニュアルを参照してください。

行のフェッチに使用されるバッファのサイズをバイト単位で指定します。これにより、エクスポート・ユーティリティによってフェッチされる配列内の最大行数が決まります。バッファ・サイズの計算には、次の計算式を使用してください。

$$\text{buffer\_size} = \text{rows\_in\_array} * \text{maximum\_row\_size}$$

0 (ゼロ) を指定すると、1 回に 1 行のみフェッチされます。

LONG、LOB、BFILE、REF、ROWID、LOGICAL ROWID または DATE 型の列が含まれる表は、1 回に 1 行ずつフェッチされます。

---

---

**注意:** BUFFER パラメータを使用できるのは、従来型パス・エクスポートの場合のみです。ダイレクト・パス・エクスポートの場合は、このパラメータの指定による影響はありません。

---

---

### 例: バッファ・サイズの計算

ここでは、バッファ・サイズの計算方法の例を示します。

次の表を作成します。

```
CREATE TABLE sample (name varchar(30), weight number);
```

name 列の最大サイズは 30 で、インジケータ用に 2 バイトをプラスします。weight 列の最大サイズは 22 (Oracle の NUMBER データ型の内部表現のサイズ) で、インジケータ用に 2 バイトをプラスします。

したがって、行の最大サイズは 56 (30+2+22+2) になります。

100 行単位で配列の操作を実行するには、バッファ・サイズを 5600 に指定する必要があります。

## COMPRESS

デフォルト:y

エクスポート・ユーティリティおよびインポート・ユーティリティによる、表データの初期エクステンツの管理方法を指定します。

デフォルトの COMPRESS=y を指定すると、インポート時に表データを 1 つの初期エクステンツに整理統合するためのフラグが付きます。エクステンツ・サイズが大きい場合 (たとえ

ば、PCTINCREASE パラメータが指定されている場合)、データの格納に必要以上の領域が割り当てられます。

COMPRESS=n を指定すると、エクスポート・ユーティリティは、初期エクステンツのサイズおよび第 2 エクステンツのサイズが指定されている現行の記憶域パラメータを使用します。パラメータの値は、CREATE TABLE 文または ALTER TABLE 文で指定された値、あるいはデータベース・システムによって変更された値になります。たとえば、表が大きくなった場合、および PCTINCREASE パラメータに 0（ゼロ）以外の値が指定されている場合は、第 2 エクステンツのサイズが変更されることがあります。

---

---

**注意：** 実際に整理統合が実行されるのはインポート時ですが、COMPRESS パラメータを指定できるのはインポート時ではなくエクスポート時のみです。記憶域パラメータなどのデータ定義は、インポート・ユーティリティではなく、エクスポート・ユーティリティによって生成されるためです。したがって、エクスポート時に COMPRESS=y を指定した場合、そのデータは整理統合形式でのみインポートできます。

---

---

---

---

**注意：** LOB データは圧縮されません。LOB データに関しては、エクスポート時の初期エクステンツのサイズおよび第 2 エクステンツのサイズの値が使用されます。

---

---

## CONSISTENT

デフォルト : n

エクスポート・ユーティリティによって読み込まれたデータのある時点における一貫性を維持し、exp コマンドの実行中に変更されないようにするために、SET TRANSACTION READ ONLY 文を使用するかどうかを指定します。エクスポート開始後に、他のアプリケーションによってそのデータベースが更新されることがわかっている場合は、CONSISTENT=y を指定してください。

CONSISTENT=n を使用すると、通常、各表は 1 つのトランザクションでエクスポートされます。ただし、表の内部にネストした表がある場合は、外部表および各内部表は別のトランザクションでエクスポートされます。パーティション表の場合は、パーティションごとに別のトランザクションでエクスポートされます。

したがって、ネストした表やパーティション表が別のアプリケーションによって更新中の場合、エクスポートされるデータが一貫性を維持できないことがあります。このような危険性をできるだけ低くするために、これらの表のエクスポートは、更新中でないときに実行してください。

表 1-2 に、2 人のユーザーによるイベントの順序を示します。user1 が表のパーティションをエクスポートし、user2 が同じ表のデータを更新するとします。

表 1-2 2 人のユーザーによる更新時のイベントの順序

時系列	user1	user2
1	TAB:P1 のエクスポートを開始	アクティビティなし
2	アクティビティなし	TAB:P2 を更新 TAB:P1 を更新 トランザクションをコミット
3	TAB:P1 のエクスポートを終了	アクティビティなし
4	TAB:P2 をエクスポート	アクティビティなし

エクスポートで `CONSISTENT=y` を指定すると、`user2` によって実行された更新はエクスポート・ファイルには書き込まれません。

エクスポートで `CONSISTENT=n` を指定すると、`TAB:P1` に対する更新はエクスポート・ファイルには書き込まれません。ただし、`TAB:P2` に対する更新は `TAB:P2` のエクスポート開始前にコミットされているため、更新がエクスポート・ファイルに書き込まれます。その結果、`user2` のトランザクションは部分的にのみエクスポート・ファイルに書き込まれるため、エクスポート・ファイルではデータの一貫性を維持できません。

`CONSISTENT=y` を指定しているときに更新量が多いと、ロールバック・セグメントの使用量が大きくなります。また、ロールバック・セグメントをスキャンしてコミットされていないトランザクションを探すため、各表のエクスポートにかかる時間が長くなります。

`CONSISTENT=y` を指定する場合は、次のことに注意してください。

- ユーザー `SYS` として接続しているとき、または `AS SYSDBA` を使用しているとき（あるいはその両方）に実行するエクスポートでは、`CONSISTENT=y` はサポートされません。
- メタデータのエクスポートには、再帰的 `SQL` 内の `SYS` スキーマを使用する必要があります。この場合、`CONSISTENT=y` を指定しても無視されます。`CONSISTENT=y` を選択したエクスポートの処理中は、メタデータを変更しないことをお勧めします。
- エクスポートに必要な時間および領域を最小にするには、一貫性が要求される表をまとめてエクスポートし、残りの表は別途エクスポートします。  
  
たとえば、`CONSISTENT=y` を指定して `emp` 表および `dept` 表をまとめてエクスポートした後、残りの表をエクスポートします。
- 「スナップショットが古すぎます」というエラーは、ロールバック領域を使い果たし、コミットされたトランザクションの領域が新しいトランザクションのために再利用されたときに発生します。ロールバック・セグメント領域を再利用することによって、最小の領域でデータベースの整合性を維持できますが、読み込み一貫性のためのイメージを維持する時間が制限されます。

コミットされたトランザクションが上書きされた後で、データベースの読み込み一貫性を維持するために、上書きによって消去された情報が必要になった場合、「スナップショットが古すぎます」というエラーが発生します。

このエラーを回避するには、読み込み一貫性エクスポートにかかる時間をできるだけ短くします（エクスポートするオブジェクト数を制限し、可能な場合はデータベースのトランザクション率を低くします）。また、ロールバック・セグメントをできるだけ大きく設定しておきます。

**参照：** 1-25 ページの「[OBJECT\\_CONSISTENT](#)」を参照してください。

## CONSTRAINTS

デフォルト : y

表制約をエクスポートするかどうかを指定します。

## DIRECT

デフォルト : n

ダイレクト・パス・エクスポートと従来型パス・エクスポートのどちらを使用するかを指定します。

DIRECT=y を指定すると、エクスポート・ユーティリティが、（バッファを調べ）SQL コマンド処理レイヤーをバイパスしてデータを直接読み込み、データを抽出します。この方法は、従来型パス・エクスポートに比べて非常に高速です。

セキュリティおよびパフォーマンスに関する考慮点を含むダイレクト・パス・エクスポートの詳細は、1-53 ページの「[ダイレクト・パス・エクスポートの起動](#)」を参照してください。

## FEEDBACK

デフォルト : 0（ゼロ）

n 行分のエクスポートを 1 つのピリオドで示すプログレス・バーの表示を指定します。たとえば、FEEDBACK=10 を指定すると、10 行分のエクスポートが終了するたびにピリオドが 1 つ表示されます。FEEDBACK 値は、エクスポートされるすべての表に適用されるため、表単位では設定できません。

### FILE

デフォルト: `expdat.dmp`

エクスポート・ファイル名を指定します。デフォルトの拡張子は `.dmp` ですが、別の拡張子を指定できます。エクスポート・ユーティリティは、複数ファイルのエクスポートをサポートしているため (1-22 ページの「[FILESIZE](#)」パラメータを参照)、複数のファイル名を指定できます。次に例を示します。

```
exp scott/tiger FILE = dat1.dmp, dat2.dmp, dat3.dmp FILESIZE=2048
```

`FILESIZE` に指定した最大値までエクスポートが実行されると、現行のファイルへの書き込みは中止され、`FILE` パラメータで次のファイル名として指定した名前のエクスポート・ファイルがオープンされます。エクスポートが完了するまで、または `FILESIZE` の最大値に再度到達するまでエクスポートが続行されます。指定したエクスポート・ファイル名が十分でないためにエクスポートを完了できない場合は、ファイル名を追加するためのプロンプトが表示されます。

### FILESIZE

デフォルト: データは、[表 1-3](#) に示す最大サイズに到達するまで、1 つのファイルに書き込まれます。

エクスポート・ユーティリティでは複数のエクスポート・ファイルへの書き込みがサポートされており、インポート・ユーティリティでは複数のエクスポート・ファイルからの読み込みが可能です。`FILESIZE` パラメータの値 (バイト制限) を指定すると、エクスポート・ユーティリティによって、それぞれのダンプ・ファイルに指定したバイト数のみ書き込まれます。

エクスポートで書き込まれるデータの量が `FILESIZE` に指定した最大値を超えると、次のエクスポート・ファイルの名前が `FILE` パラメータで指定されるか (1-22 ページの「[FILE](#)」を参照)、または `FILE` パラメータで指定したすべての名前が使用されている場合は、新しいエクスポート・ファイル名を指定するためのプロンプトが表示されます。`FILESIZE` の値を指定しない場合 (0 の指定は、`FILESIZE` の指定なしを意味する) は、`FILE` パラメータで指定したファイルの数にかかわらず、1 つのファイルのみに書き込まれます。

---

---

**注意:** エクスポート・ファイルの領域要件が使用可能なディスク領域を超えている場合、エクスポートは異常終了するため、十分なディスク領域を確保した後で再度エクスポートする必要があります。

---

---

`FILESIZE` パラメータの最大値は、64 ビットで格納できる最大値と同じです。

[表 1-3](#) に、ご使用のオペレーティング・システムおよび Oracle データベース・サーバーのリリースによって異なるダンプ・ファイルの最大サイズを示します。



表 1-3 ダンプ・ファイルの最大サイズ

オペレーティング・システム	Oracle サーバーのリリース	最大サイズ
すべて	8.1.5 より前	2GB
32 ビット・システム	8.1.5	2GB
64 ビット・システム	8.1.5 以上	無制限
32 ビット・ファイルを扱う 32 ビット・システム	すべて	2GB
64 ビット・ファイルを扱う 32 ビット・システム	8.1.6 以上	無制限

**注意：** ファイルに格納可能な最大値は、オペレーティング・システムによって異なります。この最大値については、FILESIZE を指定する前に、ご使用のオペレーティング・システム固有の Oracle マニュアルで確認してください。また、エクスポートで指定するファイル・サイズが、インポートを実行するシステムでサポートされていることも確認してください。

FILESIZE の値は、数字に KB（キロバイト数）を付けて指定できます。たとえば、FILESIZE=2KB は、FILESIZE=2048 と同じです。同様に、MB はメガバイト（1024 × 1024）を、GB はギガバイト（1024 の 3 乗）を表します。B はバイトの省略です。この場合、最終的なファイルサイズの算出に乗算は不要です（FILESIZE=2048B は、FILESIZE=2048 と同じです）。

## FLASHBACK\_SCN

デフォルト：なし

エクスポートで使用するシステム変更番号（SCN）を指定して、フラッシュバックを使用可能にします。エクスポート・オペレーションは、この指定された SCN におけるデータの一貫性を維持したまま実行されます。

**参照：** フラッシュバックの使用の詳細は、『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。

## FLASHBACK\_TIME

デフォルト: なし

時間を指定します。エクスポート・ユーティリティによって、指定された時間に一番近い SCN が検索されます。この SCN を使用して、フラッシュバックを使用可能にします。エクスポート・オペレーションは、この SCN におけるデータの一貫性を維持したまま実行されます。

**参照:** フラッシュバックの使用方法的詳細は、『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。

## FULL

デフォルト: n

エクスポートが、全データベース・モードのエクスポートであること（データベース全体のエクスポート）を示します。全データベース・モードでエクスポートするには、FULL=y を指定します。このモードでのエクスポートには、EXP\_FULL\_DATABASE ロールが必要です。

## GRANTS

デフォルト: y

オブジェクト権限をエクスポートするかどうかを指定します。エクスポートされるオブジェクト権限は、エクスポート・モードが全データベース・モードかユーザー・モードかによって異なります。全データベース・モードでは、表に対するすべての権限がエクスポートされます。一方、ユーザー・モードでは、表の所有者が付与した権限のみがエクスポートされます。システム権限は常にエクスポートされます。

## HELP

デフォルト: なし

エクスポート・パラメータの説明を表示します。ヘルプを起動するには、コマンドラインで exp help=y を入力します。

## INDEXES

デフォルト: y

索引をエクスポートするかどうかを指定します。

## LOG

デフォルト: なし

情報メッセージおよびエラー・メッセージを受け取るファイル名を指定します。次に例を示します。

```
exp SYSTEM/password LOG=export.log
```

このパラメータを指定すると、メッセージはログ・ファイルに記録されるとともに端末画面に表示されます。

## OBJECT\_CONSISTENT

デフォルト: n

エクスポート・ユーティリティによって読み込まれたデータのある時点における一貫性を維持し、エクスポート中に変更されないようにするために、`SET TRANSACTION READ ONLY` 文を使用するかどうかを指定します。`OBJECT_CONSISTENT` を `y` に設定した場合、オブジェクトがパーティション化されている場合でも、読み込み専用トランザクションに各オブジェクトをエクスポートできます。これに対し、`CONSISTENT` パラメータを使用する場合、読み込み専用トランザクションは 1 つのみです。

**参照:** 1-19 ページの「[CONSISTENT](#)」を参照してください。

## OWNER

デフォルト: なし

ユーザー・モード・エクスポートでエクスポートすることを示します。エクスポートの対象となるオブジェクトを所有するユーザー名のリストを表示します。`DBA` ユーザーがエクスポートを起動している場合は、複数のユーザーがリストされる場合があります。

## PARFILE

デフォルト: なし

エクスポート・パラメータのリストが格納されているファイルのファイル名を指定します。パラメータ・ファイルの使用の詳細は、1-6 ページの「[エクスポート・ユーティリティの起動](#)」を参照してください。

## QUERY

デフォルト: なし

表モード・エクスポートの実行時に、一連の表から行のサブセットを選択できるようにします。QUERY パラメータの値は、TABLE パラメータにリストされたすべての表（または表パーティション）に適用される SQL の SELECT 文の WHERE 句を含む文字列です。

たとえば、ユーザー scott が、職種が SALESMAN で、給与が 1600 より少ない従業員のみをエクスポートする場合は、次のように指定します（この例は UNIX ベースの場合）。

```
exp scott/tiger TABLES=emp QUERY=\"WHERE job=\"SALESMAN\" and sal <1600\"
```

---

---

**注意：** QUERY パラメータの値には空白が含まれるため、ほとんどのオペレーティング・システムでは、すべての文字列 WHERE  
job=\"SALESMAN\" および sal<1600 を二重引用符で囲むか、なんらかの方法でリテラルとしてマークする必要があります。オペレーティング・システムの予約文字も、エスケープする必要があります。システムの特殊文字および予約文字の詳細は、ご使用のオペレーティング・システム固有のドキュメントを参照してください。

---

---

この問合せの実行時、エクスポート・ユーティリティによって、次のように SQL の SELECT 文が構築されます。

```
SELECT * FROM emp WHERE job='SALESMAN' and sal <1600;
```

QUERY パラメータに指定された値は、TABLE パラメータでリストされたすべての表（または表パーティション）に適用されます。たとえば、次の文では、問合せと一致する emp および bonus の両方の行がアンロードされます。

```
exp scott/tiger TABLES=emp,bonus QUERY=\"WHERE job=\"SALESMAN\" and sal<1600\"
```

また、エクスポート・ユーティリティによって次の SQL 文が実行されます。

```
SELECT * FROM emp WHERE job='SALESMAN' and sal <1600;
```

```
SELECT * FROM bonus WHERE job='SALESMAN' and sal <1600;
```

QUERY 句で指定された列が表にない場合は、エラー・メッセージが表示されて、行はエクスポートされません。

### 制限事項

- QUERY パラメータは、全データベース・モード、ユーザー・モードまたは表領域モードのエクスポートでは指定できません。
- QUERY パラメータは、すべての指定した表に適用される必要があります。

- QUERY パラメータは、ダイレクト・パス・エクスポート（DIRECT=y）では指定できません。
- QUERY パラメータは、内部にネストした表を含む表では指定できません。
- データが QUERY エクスポートの結果かどうかを、エクスポート・ファイルの内容から判断することはできません。

## RECORDLENGTH

デフォルト：オペレーティング・システムによって異なります。

ファイル・レコードの長さをバイト単位で指定します。RECORDLENGTH パラメータは、異なるデフォルト値を使用する別のオペレーティング・システムにエクスポート・ファイルを転送する場合に指定する必要があります。

このパラメータを指定しない場合は、ご使用のプラットフォーム固有のバッファ・サイズのデフォルト値が採用されます。バッファ・サイズのデフォルト値の詳細は、ご使用のオペレーティング・システム固有の Oracle マニュアルを参照してください。

RECORDLENGTH は、ご使用のシステムのバッファ・サイズの値以上の任意の値に設定できます（最大値は 64KB）。RECORDLENGTH パラメータの変更により影響を受けるのは、ディスクに書き出す前に累積されるデータのサイズのみです。オペレーティング・システム・ファイルのブロック・サイズには影響しません。

---

---

**注意：** このパラメータは、エクスポートの I/O バッファのサイズ指定に使用できます。

---

---

適切な値の判断や他のレコード・サイズでのファイルの作成については、ご使用のオペレーティング・システム固有の Oracle マニュアルを参照してください。

## RESUMABLE

デフォルト：n

再開可能な領域割当てを有効または無効にできます。このパラメータはデフォルトでは無効なため、関連する RESUMABLE\_NAME パラメータおよび RESUMABLE\_TIMEOUT パラメータを使用するには、RESUMABLE=y に設定する必要があります。

### 参照：

- 『Oracle9i データベース概要』を参照してください。
- 再開可能な領域割当ての詳細は、『Oracle9i データベース管理者ガイド』を参照してください。

## RESUMABLE\_NAME

デフォルト: 'User USERNAME (USERID), Session SESSIONID, Instance INSTANCEID'

再開可能な文を指定します。この値はユーザー定義のテキスト文字列で、USER\_RESUMABLE または DBA\_RESUMABLE ビューに挿入して、一時停止されている特定の再開可能な文を識別できます。

RESUMABLE パラメータを y に設定して、再開可能な領域割当てを有効にしていない場合、このパラメータは無視されます。

## RESUMABLE\_TIMEOUT

デフォルト: 7200 秒 (2 時間)

エラー修正に必要な時間を指定します。タイムアウト時間内にエラーを修正できない場合は、文の実行が強制終了されます。

RESUMABLE パラメータを y に設定して、再開可能な領域割当てを有効にしていない場合、このパラメータは無視されます。

## ROWS

デフォルト: y

表のデータ行をエクスポートするかどうかを指定します。

## STATISTICS

デフォルト: ESTIMATE

エクスポートされたデータのインポート時に生成されるデータベース・オブティマイザ統計のタイプを指定します。オプションは、ESTIMATE、COMPUTE および NONE です。インポート・パラメータの詳細は、2-29 ページの「[STATISTICS](#)」および 2-70 ページの「[統計情報のインポート](#)」を参照してください。

エクスポート・ユーティリティによって、ANALYZE 文および計算済統計情報がエクスポート・ファイルに書き込まれ、統計情報が再生成される場合もあります。

ただし、システムによって生成された名前を持つ列が表に含まれる場合、計算済オブティマイザ統計は、エクスポート時に使用されません。

次の場合は、エクスポート時に、計算済オブティマイザ統計に問題ありのフラグが付きます。

- エクスポート中に、行エラーが発生した場合
- クライアント・キャラクタ・セットまたは NCHAR キャラクタ・セットが、サーバー・キャラクタ・セットまたはサーバーの NCHAR と一致しない場合

- QUERY 句を指定した場合
- 一部のパーティションまたはサブパーティションのみがエクスポートされる場合

---

**注意：** ROWS=n を指定しても、計算済統計情報は、エクスポート・ファイルから削除されません。この機能により、本番データベースからの統計情報を使用して、非本番データベース上で問合せの実行計画のチューニングを行うことができます。

---

**参照：**『Oracle9i データベース概要』を参照してください。

## TABLES

デフォルト：なし

表モード・エクスポートでエクスポートすることを指定します。エクスポートの対象となる表名、パーティション名およびサブパーティション名をリストとして指定します。表名を指定するときに、次の項目を指定できます。

- *schemaname* には、表またはパーティションのエクスポート元のユーザーのスキーマ名を指定します。スキーマ名 ORDSYS、MDSYS、CTXSYS および ORDPLUGINS は、エクスポート・ユーティリティによって予約されています。
- *tablename* には、エクスポートされる表名を指定します。表レベル・エクスポートでは、パーティション表または非パーティション表全体をエクスポートできます。リストにパーティション表が含まれる場合、パーティション名を指定しないと、すべてのパーティションおよびサブパーティションがエクスポートされます。

表名は、任意の数の「%」パターン一致文字を含むことができます。各「%」パターン一致文字は、データベースの表オブジェクトと表名の 0 個以上の文字を一致させます。関連するスキーマにある、指定されたパターンと一致するすべての表は、それぞれの表名がパラメータで明示的に指定された場合と同様に、選択されてエクスポートされます。

- *partition\_name* は、エクスポートがパーティション・レベル・エクスポートであることを示します。パーティション・レベル・エクスポートでは、1 つの表に含まれる 1 つ以上のパーティションまたはサブパーティションをエクスポートできます。

構文の形式は、次のとおりです。

```
schemaname.tablename:partition_name
schemaname.tablename:subpartition_name
```

*tablename:partition\_name* を使用する場合、指定された表はパーティション化される必要があります。*partition\_name* はパーティションまたはサブパーティションのうちのいずれかの名前である必要があります。指定された表がパーティション化されていない場合、*partition\_name* は無視され、表全体がエクスポートされます。

パーティション・レベル・エクスポートの例は、1-41 ページの「パーティション・レベル・エクスポートでのエクスポート・セッションの例」を参照してください。

---

---

**注意：** UNIX など一部のオペレーティング・システムで、カッコなどの特殊文字を使用する場合は、特殊文字として扱われないように、その文字の前にエスケープ文字を使用する必要があります。UNIX では、次の例に示すように、エスケープ文字としてバックスラッシュ (\) を使用します。

---

---

```
TABLES=(emp,dept\)
```

---

---

## 表名の制限

表名には次の制限があります。

- デフォルトでは、表名は大文字でデータベースに格納されます。表名が大文字と小文字または小文字のみで表記され、大 / 小文字を区別する場合、名前を引用符で囲む必要があります。したがって、表名は、データベースに格納されている表名と完全に一致するように指定する必要があります。

ただし、オペレーティング・システムによっては、コマンドラインの引用符自体をエスケープする必要がある場合があります。次に、異なるエクスポート・モードで大 / 小文字の区別を保持する方法を示します。

- コマンドライン・モード

```
TABLES=\'Emp\'
```

- 対話方式モード

```
Table(T) to be exported: "Emp"
```

- パラメータ・ファイル・モード

```
TABLES='Emp'
```

- 表名を引用符で囲まないかぎり、コマンドラインで指定する表名にシャープ (#) 記号は使用できません。同様に、パラメータ・ファイルでは、表名が引用符で囲まれていないかぎり、表名にシャープ (#) 記号を使用すると、シャープ (#) 記号より右側の文字はコメントとして解釈されます。

たとえば、パラメータ・ファイルに次のコマンドラインが記述されている場合、emp# の右側はすべてコメントとして解釈されるため、表 dept および mydata はエクスポートされません。

```
TABLES=(emp#, dept, mydata)
```

ただし、次の例では、emp# が引用符で囲まれているため、3 つの表はすべてエクスポートされます。



```
TABLES=("emp#", dept, mydata)
```

---

**注意：** オペレーティング・システムによっては、一重引用符を使用する必要がある場合と、二重引用符を使用する必要がある場合があります。ご使用のオペレーティング・システム固有のドキュメントで確認してください。表のネーミング方法に制限があるオペレーティング・システムもあります。

たとえば、UNIX の C シェルではドル記号 (\$) やシャープ (#) (またはその他の特殊文字) には特別な意味があります。これらの文字をシェルを介してエクスポートするには、エスケープ文字を使用する必要があります。

---

複数の `schema.tablename:(sub)partition_name` 引数を指定する `TABLES` パラメータの場合、エクスポート・ユーティリティは、オブジェクトのリストを処理する前に複製を削除します。

## TABLESPACES

デフォルト: なし

表領域のすべての表がエクスポート・ダンプ・ファイルへエクスポートされるように指定します。これには、表領域のリストに含まれるすべての表およびパーティションを持つすべての表が含まれます。索引は、格納されている場所にかかわらず、表とともにエクスポートされます。

`TABLESPACES` を使用して表領域のすべての表をエクスポートするには、`EXP_FULL_DATABASE` ロールが必要です。

`TABLESPACES` を `TRANSPORT_TABLESPACE=y` と組み合わせて使用すると、表領域の一部のリストをデータベースからエクスポート・ファイルへエクスポートするように指定できます。

## TRANSPORT\_TABLESPACE

デフォルト: n

`y` を指定すると、トランスポートابل表領域のメタデータをエクスポートできるようになります。

**参照：**

- 1-61 ページの「[トランスポートابل表領域](#)」を参照してください。
- 『Oracle9i データベース管理者ガイド』を参照してください。
- 『Oracle9i データベース概要』を参照してください。

## TRIGGERS

デフォルト: `y`

トリガーをエクスポートするかどうかを指定します。

## TTS\_FULL\_CHECK

デフォルト: `FALSE`

`TRUE` を設定すると、エクスポート・ユーティリティによって、リカバリ・セット（リカバリの対象となる表領域のセット）にリカバリ・セット外のオブジェクト（またはその逆）に対する依存関係（特に `IN` ポインタ）がないことが確認されます。

## USERID（ユーザー名 / パスワード）

デフォルト: なし

エクスポートを実行するユーザーの `username/password`（およびオプションの接続文字列）を指定します。パスワードを指定しないと、入力するように要求されます。

`USERID` は、次のように指定できます。

`username/password AS SYSDBA`

または

`username/password@instance AS SYSDBA`

ユーザー `SYS` として接続する場合は、接続文字列に `AS SYSDBA` も指定する必要があります。オペレーティング・システムによっては、`AS SYSDBA` を特殊文字列とみなし、その文字列全体を引用符で囲む必要があります。詳細は、1-8 ページの「[SYSDBA でのエクスポート・ユーティリティの起動](#)」を参照してください。

**参照：**

- 『Oracle9i Heterogeneous Connectivity Administrator's Guide』を参照してください。
- Oracle Net に `@connect_string` を指定する方法の詳細は、ご使用の Oracle Net プロトコルのユーザーズ・ガイドを参照してください。

## VOLSIZE

各テープ・ボリュームのエクスポート・ファイルに最大バイト数を指定します。

VOLSIZE パラメータの最大値は、64 ビットで格納できる最大値と同じです。詳細は、ご使用のオペレーティング・システム固有の Oracle マニュアルを参照してください。

VOLSIZE の値は、数字に KB（キロバイト数）を付けて指定できます。たとえば、VOLSIZE=2KB は、VOLSIZE=2048 と同じです。同様に、MB はメガバイト（ $1024 \times 1024$ ）を、GB はギガバイト（ $1024$  の 3 乗）を表します。B はバイトの省略です。この場合、最終的なファイル・サイズの算出に乗算は不要です（VOLSIZE=2048B は VOLSIZE=2048 と同じ）。

## パラメータ間の相互作用

パラメータによっては、パラメータ間で競合することがあります。たとえば、TABLES および OWNER の両方を指定すると、競合が発生します。次のようなコマンドを指定した場合、エラーが発生し、エクスポートは終了します。

```
exp SYSTEM/password OWNER=jones TABLES=scott.emp
```

同様に、OWNER および TABLES も FULL=Y と競合します。

## エクスポート・セッションの例

この項では、次のタイプのエクスポート・セッションの例を示します。

- [全データベース・モードでのエクスポート・セッションの例](#)
- [ユーザー・モードでのエクスポート・セッションの例](#)
- [表モードでのエクスポート・セッションの例](#)
- [パーティション・レベル・エクスポートでのエクスポート・セッションの例](#)

各例で、コマンドライン方式およびパラメータ・ファイル方式の両方の使用方法を示します。

## 全データベース・モードでのエクスポート・セッションの例

全データベース・モードでエクスポートを実行できるのは、DBA ロールまたは EXP\_FULL\_DATABASE ロールを持つユーザーのみです。この例では、すべての GRANTS（付与されている権限）およびすべてのデータとともにデータベース全体をファイル dba.dmp にエクスポートします。

### パラメータ・ファイル方式

```
> exp SYSTEM/password PARFILE=params.dat
```

params.dat ファイルには、次の情報が格納されています。

```
FILE=dba.dmp
GRANTS=y
FULL=y
ROWS=y
```

### コマンドライン方式

```
> exp SYSTEM/password FULL=y FILE=dba.dmp GRANTS=y ROWS=y
```

### エクスポート・メッセージ

Export: Release 9.2.0.1.0 - Production on Wed Feb 27 16:52:15 2002

(c) Copyright 2002 Oracle Corporation. All rights reserved.

Connected to: Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production  
With the Partitioning and Oracle Data Mining options  
JServer Release 9.2.0.1.0 - Production  
Export done in WE8DEC character set and AL16UTF16 NCHAR character set

```
About to export the entire database ...
. exporting tablespace definitions
. exporting profiles
. exporting user definitions
. exporting roles
. exporting resource costs
. exporting rollback segment definitions
. exporting database links
. exporting sequence numbers
. exporting directory aliases
. exporting context namespaces
. exporting foreign function library names
. exporting PUBLIC type synonyms
. exporting private type synonyms
. exporting object type definitions
```

```

. exporting system procedural objects and actions
. exporting pre-schema procedural objects and actions
. exporting cluster definitions
. about to export SYSTEM's tables via Conventional Path ...
. . exporting table          AQ$_INTERNET_AGENTS          0 rows exported
. . exporting table          AQ$_INTERNET_AGENT_PRIVS      0 rows exported
. . exporting table          DEF$_AQCALL                   0 rows exported
. . exporting table          DEF$_AQERROR                   0 rows exported
. . exporting table          DEF$_CALLDEST                  0 rows exported
. . exporting table          DEF$_DEFAULTDEST               0 rows exported
. . exporting table          DEF$_DESTINATION               0 rows exported
. . exporting table          DEF$_ERROR                     0 rows exported
. . exporting table          DEF$_LOB                       0 rows exported
. . exporting table          DEF$_ORIGIN                    0 rows exported
. . exporting table          DEF$_PROPAGATOR                0 rows exported
. . exporting table          DEF$_PUSHED_TRANSACTIONS        0 rows exported
. . exporting table          DEF$_TEMP$LOB                  0 rows exported
. . exporting table          LOGSTDBY$APPLY_MILESTONE        0 rows exported
. . exporting table          LOGSTDBY$APPLY_PROGRESS        0 rows exported
. . exporting partition                                P0          0 rows exported
. . exporting table          LOGSTDBY$EVENTS                0 rows exported
. . exporting table          LOGSTDBY$PARAMETERS            0 rows exported
. . exporting table          LOGSTDBY$PLSQL                 0 rows exported
. . exporting table          LOGSTDBY$SCN                   0 rows exported
. . exporting table          LOGSTDBY$SKIP                  0 rows exported
. . exporting table          LOGSTDBY$SKIP_TRANSACTION      0 rows exported
. . exporting table          REPCAT$_AUDIT_ATTRIBUTE         2 rows exported
. . exporting table          REPCAT$_AUDIT_COLUMN           0 rows exported
. . exporting table          REPCAT$_COLUMN_GROUP           0 rows exported
. . exporting table          REPCAT$_CONFLICT               0 rows exported
. . exporting table          REPCAT$_DDL                    0 rows exported
. . exporting table          REPCAT$_EXCEPTIONS             0 rows exported
. . exporting table          REPCAT$_EXTENSION              0 rows exported
. . exporting table          REPCAT$_FLAVORS                0 rows exported
. . exporting table          REPCAT$_FLAVOR_OBJECTS         0 rows exported
. . exporting table          REPCAT$_GENERATED              0 rows exported
. . exporting table          REPCAT$_GROUPED_COLUMN         0 rows exported
. . exporting table          REPCAT$_INSTANTIATION_DDL      0 rows exported
. . exporting table          REPCAT$_KEY_COLUMNS            0 rows exported
. . exporting table          REPCAT$_OBJECT_PARMs           0 rows exported
. . exporting table          REPCAT$_OBJECT_TYPES           28 rows exported
. . exporting table          REPCAT$_PARAMETER_COLUMN       0 rows exported
. . exporting table          REPCAT$_PRIORITY               0 rows exported
. . exporting table          REPCAT$_PRIORITY_GROUP         0 rows exported
. . exporting table          REPCAT$_REFRESH_TEMPLATES      0 rows exported
. . exporting table          REPCAT$_REPCAT                 0 rows exported
. . exporting table          REPCAT$_REPCATLOG              0 rows exported

```

```

. . exporting table          REPCAT$_REPCOLUMN          0 rows exported
. . exporting table          REPCAT$_REPGROUP_PRIVS      0 rows exported
. . exporting table          REPCAT$_REPOBJECT           0 rows exported
. . exporting table          REPCAT$_REPPROP             0 rows exported
. . exporting table          REPCAT$_REPSHEMA            0 rows exported
. . exporting table          REPCAT$_RESOLUTION          0 rows exported
. . exporting table          REPCAT$_RESOLUTION_METHOD  19 rows exported
. . exporting table          REPCAT$_RESOLUTION_STATISTICS 0 rows exported
. . exporting table          REPCAT$_RESOL_STATS_CONTROL 0 rows exported
. . exporting table          REPCAT$_RUNTIME_PARMS       0 rows exported
. . exporting table          REPCAT$_SITES_NEW           0 rows exported
. . exporting table          REPCAT$_SITE_OBJECTS        0 rows exported
. . exporting table          REPCAT$_SNAPGROUP           0 rows exported
. . exporting table          REPCAT$_TEMPLATE_OBJECTS    0 rows exported
. . exporting table          REPCAT$_TEMPLATE_PARMS      0 rows exported
. . exporting table          REPCAT$_TEMPLATE_REFGROUPS  0 rows exported
. . exporting table          REPCAT$_TEMPLATE_SITES      0 rows exported
. . exporting table          REPCAT$_TEMPLATE_STATUS     3 rows exported
. . exporting table          REPCAT$_TEMPLATE_TARGETS    0 rows exported
. . exporting table          REPCAT$_TEMPLATE_TYPES      2 rows exported
. . exporting table          REPCAT$_USER_AUTHORIZATIONS 0 rows exported
. . exporting table          REPCAT$_USER_PARM_VALUES    0 rows exported
. . exporting table          SQLPLUS_PRODUCT_PROFILE     0 rows exported
. about to export OUTLN's tables via Conventional Path ...
. . exporting table          OL$                          0 rows exported
. . exporting table          OL$HINTS                    0 rows exported
. . exporting table          OL$NODES                    0 rows exported
. about to export DBSNMP's tables via Conventional Path ...
. about to export SCOTT's tables via Conventional Path ...
. . exporting table          BONUS                       0 rows exported
. . exporting table          DEPT                        4 rows exported
. . exporting table          EMP                          14 rows exported
. . exporting table          SALGRADE                     5 rows exported
. about to export ADAMS's tables via Conventional Path ...
. about to export JONES's tables via Conventional Path ...
. about to export CLARK's tables via Conventional Path ...
. about to export BLAKE's tables via Conventional Path ...
. . exporting table          DEPT                        8 rows exported
. . exporting table          MANAGER                     4 rows exported
. exporting synonyms
. exporting views
. exporting referential integrity constraints
. exporting stored procedures
. exporting operators
. exporting indextypes
. exporting bitmap, functional and extensible indexes
. exporting posttables actions

```

```
. exporting triggers
. exporting materialized views
. exporting snapshot logs
. exporting job queues
. exporting refresh groups and children
. exporting dimensions
. exporting post-schema procedural objects and actions
. exporting user history table
. exporting default and system auditing options
. exporting statistics
Export terminated successfully without warnings.
```

## ユーザー・モードでのエクスポート・セッションの例

ユーザー・モードのエクスポートは、1人以上のデータベース・ユーザーのバックアップに使用可能です。たとえば、削除されたユーザーの表を、DBA が一定の期間バックアップを取る場合などに有効です。ユーザー・モードは、ユーザーが自分自身のデータのバックアップを取る場合や、ある所有者のオブジェクトを別の所有者に移す場合にも適しています。次の例では、ユーザー scott が自分が所有する表をエクスポートします。

### パラメータ・ファイル方式

```
> exp scott/tiger PARFILE=params.dat
```

params.dat ファイルには、次の情報が格納されています。

```
FILE=scott.dmp
OWNER=scott
GRANTS=y
ROWS=y
COMPRESS=y
```

### コマンドライン方式

```
> exp scott/tiger FILE=scott.dmp OWNER=scott GRANTS=y ROWS=y COMPRESS=y
```

### エクスポート・メッセージ

```
Export: Release 9.2.0.1.0 - Production on Wed Feb 27 17:01:06 2002
```

```
(c) Copyright 2002 Oracle Corporation. All rights reserved.
```

```
Connected to: Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production
With the Partitioning and Oracle Data Mining options
JServer Release 9.2.0.1.0 - Production
Export done in WE8DEC character set and AL16UTF16 NCHAR character set
. exporting pre-schema procedural objects and actions
```

```
. exporting foreign function library names for user SCOTT
. exporting PUBLIC type synonyms
. exporting private type synonyms
. exporting object type definitions for user SCOTT
About to export SCOTT's objects ...
. exporting database links
. exporting sequence numbers
. exporting cluster definitions
. about to export SCOTT's tables via Conventional Path ...
. . exporting table                BONUS                0 rows exported
. . exporting table                DEPT                   4 rows exported
. . exporting table                EMP                   14 rows exported
. . exporting table                SALGRADE               5 rows exported
. exporting synonyms
. exporting views
. exporting stored procedures
. exporting operators
. exporting referential integrity constraints
. exporting triggers
. exporting indextypes
. exporting bitmap, functional and extensible indexes
. exporting posttables actions
. exporting materialized views
. exporting snapshot logs
. exporting job queues
. exporting refresh groups and children
. exporting dimensions
. exporting post-schema procedural objects and actions
. exporting statistics
Export terminated successfully without warnings.
```

## 表モードでのエクスポート・セッションの例

表モードでは、表データまたは表定義のみをエクスポートできます。(エクスポートする行がない場合は、CREATE TABLE 文がエクスポート・ファイルに書き込まれます。このとき、権限付与および索引が指定されると、これらもエクスポート・ファイルに書き込まれます。)

EXP\_FULL\_DATABASE ロールを持つユーザーは、表モードで TABLES=schemaname.tablename を指定することによって、どのユーザーのスキーマに属する表でもエクスポートできます。

schemaname を指定しない場合、その直前にエクスポートされたオブジェクトのスキーマ名がデフォルト値として採用されます。直前にエクスポートされたオブジェクトがない場合は、エクスポート実行者のスキーマがデフォルトの値になります。次の例では、a 表の場合は SYSTEM、c 表の場合は scott が、スキーマのデフォルトとして採用されます。

```
> exp SYSTEM/password TABLES=(a, scott.b, c, mary.d)
```



EXP\_FULL\_DATABASE ロールを持つユーザーは、他のユーザーが所有する依存オブジェクトのエクスポートも実行できます。権限を持たないユーザーは、そのユーザー自身が所有し、指定した表の依存オブジェクトのみをエクスポートできます。

表モードのエクスポートには、クラスタ定義がありません。このため、データはクラスタ化されていない表としてエクスポートされます。したがって、表モードは、表の非クラスタ化に使用できます。

## 例 1: DBA による 2 人のユーザーの表のエクスポート

この例では、DBA が 2 人のユーザーの表を指定してエクスポートします。

### パラメータ・ファイル方式

```
> exp SYSTEM/password PARFILE=params.dat
```

params.dat ファイルには、次の情報が格納されています。

```
FILE=expdat.dmp
TABLES=(scott.emp,blake.dept)
GRANTS=y
INDEXES=y
```

### コマンドライン方式

```
> exp SYSTEM/password FILE=expdat.dmp TABLES=(scott.emp,blake.dept) GRANTS=y-
INDEXES=y
```

### エクスポート・メッセージ

Export: Release 9.2.0.1.0 - Production on Wed Feb 27 17:01:35 2002

(c) Copyright 2002 Oracle Corporation. All rights reserved.

Connected to: Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production  
With the Partitioning and Oracle Data Mining options  
JServer Release 9.2.0.1.0 - Production  
Export done in WE8DEC character set and AL16UTF16 NCHAR character set

About to export specified tables via Conventional Path ...

Current user changed to SCOTT

.. exporting table	EMP	14 rows exported
--------------------	-----	------------------

Current user changed to BLAKE

.. exporting table	DEPT	8 rows exported
--------------------	------	-----------------

Export terminated successfully without warnings.

## 例 2: ユーザーによる自分が所有する表のエクスポート

この例では、ユーザー blake が自分が所有している表の中から選択した表をエクスポートします。

### パラメータ・ファイル方式

```
> exp blake/paper PARFILE=params.dat
```

params.dat ファイルには、次の情報が格納されています。

```
FILE=blake.dmp  
TABLES=(dept,manager)  
ROWS=y  
COMPRESS=y
```

### コマンドライン方式

```
> exp blake/paper FILE=blake.dmp TABLES=(dept, manager) ROWS=y COMPRESS=y
```

### エクスポート・メッセージ

```
Export: Release 9.2.0.1.0 - Production on Wed Feb 27 17:01:38 2002
```

```
(c) Copyright 2002 Oracle Corporation. All rights reserved.
```

```
Connected to: Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production  
With the Partitioning and Oracle Data Mining options  
JServer Release 9.2.0.1.0 - Production  
Export done in WE8DEC character set and AL16UTF16 NCHAR character set
```

```
About to export specified tables via Conventional Path ...
```

```
.. exporting table                DEPT                8 rows exported  
.. exporting table                MANAGER              4 rows exported
```

```
Export terminated successfully without warnings.
```

## 例 3: パターン一致を使用した様々な表のエクスポート

この例では、パターン一致を使用して、ユーザー scott およびユーザー blake の様々な表をエクスポートします。

### パラメータ・ファイル方式

```
> exp SYSTEM/password PARFILE=params.dat
```

params.dat ファイルには、次の情報が格納されています。

```
FILE=misc.dmp  
TABLES=(scott.%,blake.%,scott.%)
```

## コマンドライン方式

```
> exp SYSTEM/password FILE=misc.dmp TABLES=(scott.%P%,blake.%,scott.%S%)
```

## エクスポート・メッセージ

Export: Release 9.2.0.1.0 - Production on Wed Feb 27 17:01:40 2002

(c) Copyright 2002 Oracle Corporation. All rights reserved.

Connected to: Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production  
With the Partitioning and Oracle Data Mining options  
JServer Release 9.2.0.1.0 - Production  
Export done in WE8DEC character set and AL16UTF16 NCHAR character set

About to export specified tables via Conventional Path ...

Current user changed to SCOTT

.. exporting table	DEPT	4 rows exported
.. exporting table	EMP	14 rows exported

Current user changed to BLAKE

.. exporting table	DEPT	8 rows exported
.. exporting table	MANAGER	4 rows exported

Current user changed to SCOTT

.. exporting table	BONUS	0 rows exported
.. exporting table	SALGRADE	5 rows exported

Export terminated successfully without warnings.

## パーティション・レベル・エクスポートでのエクスポート・セッションの例

パーティション・レベル・エクスポートでは、エクスポートの対象を表のパーティションおよびサブパーティション単位で指定できます。

### 例 1: パーティションを指定しない表のエクスポート

従業員名にパーティションが指定されている emp 表があるとします。m および z の 2 つのパーティションがあります。次の例に示すように、パーティションを指定しないでエクスポートを実行すると、すべてのパーティションがエクスポートされます。

## パラメータ・ファイル方式

```
> exp scott/tiger PARFILE=params.dat
```

params.dat ファイルには、次の情報が格納されています。

```
TABLES=(emp)
ROWS=y
```

### コマンドライン方式

```
> exp scott/tiger TABLES=emp rows=y
```

### エクスポート・メッセージ

Export: Release 9.2.0.1.0 - Production on Wed Feb 27 17:01:53 2002

(c) Copyright 2002 Oracle Corporation. All rights reserved.

Connected to: Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production  
With the Partitioning and Oracle Data Mining options  
JServer Release 9.2.0.1.0 - Production  
Export done in WE8DEC character set and AL16UTF16 NCHAR character set

About to export specified tables via Conventional Path ...

. . exporting table	EMP	
. . exporting partition	M	8 rows exported
. . exporting partition	Z	6 rows exported

Export terminated successfully without warnings.

### 例 2: パーティションを指定した表のエクスポート

従業員名にパーティションが指定されている emp 表があるとします。m および z の 2 つのパーティションがあります。次の例に示すように、パーティションを指定して表をエクスポートすると、指定したパーティションのみがエクスポートされます。

### パラメータ・ファイル方式

```
> exp scott/tiger PARFILE=params.dat
```

params.dat ファイルには、次の情報が格納されています。

```
TABLES=(emp:m)  
ROWS=y
```

### コマンドライン方式

```
> exp scott/tiger TABLES=emp:m rows=y
```

### エクスポート・メッセージ

Export: Release 9.2.0.1.0 - Production on Wed Feb 27 17:01:55 2002

(c) Copyright 2002 Oracle Corporation. All rights reserved.

Connected to: Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production  
With the Partitioning and Oracle Data Mining options

```
JServer Release 9.2.0.1.0 - Production
Export done in WE8DEC character set and AL16UTF16 NCHAR character set

About to export specified tables via Conventional Path ...
.. exporting table                                EMP
.. exporting partition                            M           8 rows exported
Export terminated successfully without warnings.
```

### 例 3: コンポジット・パーティションのエクスポート

emp は、2 つのパーティション m および z で構成されているパーティション表です。emp は、コンポジット・メソッドでパーティション化されます。パーティション m にはサブパーティション sp1 および sp2 があり、パーティション z にはサブパーティション sp3 および sp4 があります。次の例に示すように、コンポジット・パーティション m をエクスポートする場合、すべてのサブパーティション (sp1 および sp2) がエクスポートされます。サブパーティション (sp4) を指定して表をエクスポートすると、指定したサブパーティションのみがエクスポートされます。

#### パラメータ・ファイル方式

```
> exp scott/tiger PARFILE=params.dat
```

params.dat ファイルには、次の情報が格納されています。

```
TABLES=(emp:m,emp:sp4)
ROWS=y
```

#### コマンドライン方式

```
> exp scott/tiger TABLES=(emp:m, emp:sp4) ROWS=y
```

#### エクスポート・メッセージ

```
Export: Release 9.2.0.1.0 - Production on Wed Feb 27 17:22:47 2002
```

```
(c) Copyright 2002 Oracle Corporation. All rights reserved.
```

```
Connected to: Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production
With the Partitioning and Oracle Data Mining options
JServer Release 9.2.0.1.0 - Production
Export done in WE8DEC character set and AL16UTF16 NCHAR character set
```

```
About to export specified tables via Conventional Path ...
.. exporting table                                EMP
.. exporting composite partition                  M
.. exporting subpartition                        SP1           1 rows exported
.. exporting subpartition                        SP2           3 rows exported
```

```
.. exporting composite partition          Z
.. exporting subpartition                 SP4      1 rows exported
Export terminated successfully without warnings.
```

対話方式の使用

エクスポート・ユーティリティは、パラメータを指定しないでコマンドラインから起動すると、対話方式で起動されます。コマンドライン対話方式では、エクスポート・ユーティリティのすべての機能に関してプロンプトが表示されるわけではありません。また、コマンドライン対話方式は下位互換用のみに提供されています。エクスポート・ユーティリティに対話方式のインタフェースを使用する場合は、Oracle Enterprise Manager (OEM) エクスポート・ウィザードの使用をお薦めします。

コマンドラインで `username/password` を指定しないと、入力するように求められます。

対話方式でエクスポート・ユーティリティを起動する場合、エクスポート・ユーティリティからの応答は、コマンドラインでの入力内容によって異なります。表 1-4 に、入力内容とそれに対する応答を示します。

表 1-4 対話方式を使用したエクスポート・ユーティリティの起動

入力内容	エクスポート・ユーティリティからの応答
exp username/password@instance as sysdba	エクスポート・セッションの起動
exp username/password@instance	エクスポート・セッションの起動
exp username/password as sysdba	エクスポート・セッションの起動
exp username/password	エクスポート・セッションの起動
exp username@instance as sysdba	パスワード入力を求めるプロンプトの表示
exp username@instance	パスワード入力を求めるプロンプトの表示
exp username	パスワード入力を求めるプロンプトの表示
exp username as sysdba	パスワード入力を求めるプロンプトの表示
exp / as sysdba	パスワード入力を求めるプロンプトの表示なし（オペレーティング・システム認証の使用）
exp /	パスワード入力を求めるプロンプトの表示なし（オペレーティング・システム認証の使用）
exp /@instance as sysdba	パスワード入力を求めるプロンプトの表示なし（オペレーティング・システム認証の使用）

表 1-4 対話方式を使用したエクスポート・ユーティリティの起動（続き）

入力内容	エクスポート・ユーティリティからの応答
exp /@instance	パスワード入力を求めるプロンプトの表示なし（オペレーティング・システム認証の使用）

エクスポート・ユーティリティの対話方式モードでは、SYSDBA で接続するか、@instance で接続するかを指定するプロンプトは表示されません。AS SYSDBA や @instance は、ユーザー名とともに指定する必要があります。

また、パスワードを指定しなかったためにプロンプトが表示されると、@instance 文字列を指定できなくなります。@instance はユーザー名を指定する場合のみ使用できます。

AS SYSDBA を使用してエクスポート・ユーティリティを起動する前に、正しいコマンドラインの構文について、1-8 ページの「[SYSDBA でのエクスポート・ユーティリティの起動](#)」を参照してください。

エクスポート・ユーティリティが起動された後、次のプロンプトが表示されます。他のプロンプトに対して入力した応答によって決まるプロンプトもあるため、エクスポート・セッションですべてのプロンプトが表示されるとはかぎりません。デフォルト値が表示されるプロンプトもあります。このデフォルト値が使用可能な場合は、[Enter] を押します。

Export: Release 9.2.0.1.0 - Production on Wed Feb 27 17:02:03 2002

(c) Copyright 2002 Oracle Corporation. All rights reserved.

```
Connected to: Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production
With the Partitioning and Oracle Data Mining options
JServer Release 9.2.0.1.0 - Production
Enter array fetch buffer size: 4096 >
Export file: expdat.dmp >
(1)E(ntire database), (2)U(sers), or (3)T(ables): (2)U >
Export grants (yes/no): yes >
Export table data (yes/no): yes >
Compress extents (yes/no): yes >
Export done in WE8DEC character set and AL16UTF16 NCHAR character set
```

```
About to export the entire database ...
. exporting tablespace definitions
. exporting profiles
. exporting user definitions
. exporting roles
. exporting resource costs
. exporting rollback segment definitions
. exporting database links
. exporting sequence numbers
```

```

. exporting directory aliases
. exporting context namespaces
. exporting foreign function library names
. exporting PUBLIC type synonyms
. exporting private type synonyms
. exporting object type definitions
. exporting system procedural objects and actions
. exporting pre-schema procedural objects and actions
. exporting cluster definitions
. about to export SYSTEM's tables via Conventional Path ...
. . exporting table          AQ$_INTERNET_AGENTS          0 rows exported
. . exporting table          AQ$_INTERNET_AGENT_PRIVS      0 rows exported
. . exporting table          DEF$_AQCALL                   0 rows exported
. . exporting table          DEF$_AQERROR                  0 rows exported
. . exporting table          DEF$_CALLDEST                  0 rows exported
. . exporting table          DEF$_DEFAULTDEST               0 rows exported
. . exporting table          DEF$_DESTINATION               0 rows exported
. . exporting table          DEF$_ERROR                     0 rows exported
. . exporting table          DEF$_LOB                       0 rows exported
. . exporting table          DEF$_ORIGIN                    0 rows exported
. . exporting table          DEF$_PROPAGATOR                0 rows exported
. . exporting table          DEF$_PUSHED_TRANSACTIONS       0 rows exported
. . exporting table          DEF$_TEMP$LOB                  0 rows exported
. . exporting table          LOGSTDBY$APPLY_MILESTONE       0 rows exported
. . exporting table          LOGSTDBY$APPLY_PROGRESS       0 rows exported
. . exporting partition      P0                             0 rows exported
. . exporting table          LOGSTDBY$EVENTS                0 rows exported
. . exporting table          LOGSTDBY$PARAMETERS            0 rows exported
. . exporting table          LOGSTDBY$PLSQL                 0 rows exported
. . exporting table          LOGSTDBY$SCN                   0 rows exported
. . exporting table          LOGSTDBY$SKIP                  0 rows exported
. . exporting table          LOGSTDBY$SKIP_TRANSACTION      0 rows exported
. . exporting table          REPCAT$_AUDIT_ATTRIBUTE        2 rows exported
. . exporting table          REPCAT$_AUDIT_COLUMN           0 rows exported
. . exporting table          REPCAT$_COLUMN_GROUP           0 rows exported
. . exporting table          REPCAT$_CONFLICT               0 rows exported
. . exporting table          REPCAT$_DDL                    0 rows exported
. . exporting table          REPCAT$_EXCEPTIONS             0 rows exported
. . exporting table          REPCAT$_EXTENSION              0 rows exported
. . exporting table          REPCAT$_FLAVORS                0 rows exported
. . exporting table          REPCAT$_FLAVOR_OBJECTS         0 rows exported
. . exporting table          REPCAT$_GENERATED              0 rows exported
. . exporting table          REPCAT$_GROUPED_COLUMN         0 rows exported
. . exporting table          REPCAT$_INSTANTIATION_DDL      0 rows exported
. . exporting table          REPCAT$_KEY_COLUMNS            0 rows exported
. . exporting table          REPCAT$_OBJECT_PARMS           0 rows exported
. . exporting table          REPCAT$_OBJECT_TYPES           28 rows exported

```



```

. . exporting table      REPCAT$_PARAMETER_COLUMN      0 rows exported
. . exporting table      REPCAT$_PRIORITY              0 rows exported
. . exporting table      REPCAT$_PRIORITY_GROUP        0 rows exported
. . exporting table      REPCAT$_REFRESH_TEMPLATES     0 rows exported
. . exporting table      REPCAT$_REPCAT               0 rows exported
. . exporting table      REPCAT$_REPCATALOG            0 rows exported
. . exporting table      REPCAT$_REPCOLUMN             0 rows exported
. . exporting table      REPCAT$_REPGROUP_PRIVS        0 rows exported
. . exporting table      REPCAT$_REPOBJECT             0 rows exported
. . exporting table      REPCAT$_REPPROP              0 rows exported
. . exporting table      REPCAT$_REPSHEMA             0 rows exported
. . exporting table      REPCAT$_RESOLUTION            0 rows exported
. . exporting table      REPCAT$_RESOLUTION_METHOD     19 rows exported
. . exporting table      REPCAT$_RESOLUTION_STATISTICS 0 rows exported
. . exporting table      REPCAT$_RESOL_STATS_CONTROL   0 rows exported
. . exporting table      REPCAT$_RUNTIME_PARMS         0 rows exported
. . exporting table      REPCAT$_SITES_NEW             0 rows exported
. . exporting table      REPCAT$_SITE_OBJECTS          0 rows exported
. . exporting table      REPCAT$_SNAPGROUP            0 rows exported
. . exporting table      REPCAT$_TEMPLATE_OBJECTS      0 rows exported
. . exporting table      REPCAT$_TEMPLATE_PARMS       0 rows exported
. . exporting table      REPCAT$_TEMPLATE_REFGROUPS    0 rows exported
. . exporting table      REPCAT$_TEMPLATE_SITES       0 rows exported
. . exporting table      REPCAT$_TEMPLATE_STATUS      3 rows exported
. . exporting table      REPCAT$_TEMPLATE_TARGETS     0 rows exported
. . exporting table      REPCAT$_TEMPLATE_TYPES       2 rows exported
. . exporting table      REPCAT$_USER_AUTHORIZATIONS   0 rows exported
. . exporting table      REPCAT$_USER_PARM_VALUES     0 rows exported
. . exporting table      SQLPLUS_PRODUCT_PROFILE      0 rows exported
. about to export OUTLN's tables via Conventional Path ...
. . exporting table      OL$                          0 rows exported
. . exporting table      OL$HINTS                     0 rows exported
. . exporting table      OL$NODES                     0 rows exported
. about to export DBSNMP's tables via Conventional Path ...
. about to export SCOTT's tables via Conventional Path ...
. . exporting table      BONUS                        0 rows exported
. . exporting table      DEPT                        4 rows exported
. . exporting table      EMP                        14 rows exported
. . exporting table      SALGRADE                    5 rows exported
. about to export ADAMS's tables via Conventional Path ...
. about to export JONES's tables via Conventional Path ...
. about to export CLARK's tables via Conventional Path ...
. about to export BLAKE's tables via Conventional Path ...
. . exporting table      DEPT                        8 rows exported
. . exporting table      MANAGER                    4 rows exported
. exporting synonyms
. exporting views

```

```
. exporting referential integrity constraints
. exporting stored procedures
. exporting operators
. exporting indextypes
. exporting bitmap, functional and extensible indexes
. exporting posttables actions
. exporting triggers
. exporting materialized views
. exporting snapshot logs
. exporting job queues
. exporting refresh groups and children
. exporting dimensions
. exporting post-schema procedural objects and actions
. exporting user history table
. exporting default and system auditing options
. exporting statistics
Export terminated successfully without warnings.
```

## 制限事項

対話方式を使用する場合は、次のことに注意してください。

- ユーザー・モードでは、データをエクスポートする前に、エクスポート対象とするすべてのユーザー名を入力するプロンプトが表示されます。ユーザー名のリストの入力後、[Enter] を押してカレントのエクスポート・セッションを開始します。
- 表モードでは、スキーマの接頭辞を指定しないと、エクスポート実行者のスキーマ、または現行のセッション中に最後にエクスポートされた表が格納されているスキーマがデフォルトの値になります。

たとえば、特権ユーザーである `beth` が表モードでエクスポートを実行している場合、他のユーザーのスキーマが指定されるまでは、すべての表は `beth` のスキーマにあると判断されます。他のユーザーのスキーマに属する表をエクスポートできるのは、特権ユーザー（`EXP_FULL_DATABASE` ロールを持つユーザー）のみです。

- 「エクスポートする表」という入力要求のプロンプトに対して表を指定しないと、エクスポート・ユーティリティは終了します。

## 警告、エラーおよび完了メッセージ

この項では、エクスポート・ユーティリティによって発行される異なるタイプのメッセージについて説明します。また、そのメッセージのログ・ファイルへの保存方法についても説明します。

### ログ・ファイル

すべてのエクスポート・メッセージは、ログ・ファイルに保存できます。この場合の保存方法は2つあります。1つ目は、LOG パラメータを使用する方法です（1-25 ページの「LOG」を参照）。2つ目は、システムでサポートされている場合にかぎりませんが、エクスポート・ユーティリティの出力をファイルにリダイレクトする方法です。リダイレクト先のファイルには、正常にアンロードされた場合はその内容が、またエラーが発生した場合はそのエラーに関する詳細情報が記録されます。出力のリダイレクトの詳細は、ご使用のオペレーティング・システム固有の Oracle マニュアルを参照してください。

### 警告メッセージ

リカバリ可能なエラーの場合、エクスポート処理は続行されます。たとえば、表のエクスポート中にエラーが発生した場合は、エラー・メッセージが表示され（またはログが記録され）、次の表にスキップして処理が続けられます。リカバリ可能なエラーは、警告と呼ばれます。

エクスポートでは、無効なオブジェクトに対しても警告が発行されます。

たとえば、表モード・エクスポートで、存在しない表を指定した場合、エクスポート・ユーティリティは他の表をすべてエクスポートします。それから、警告を発行して処理を正常に終了します。

### リカバリ不能エラー・メッセージ

エラーの中にはリカバリ不能なものもあり、このようなエラーが発生するとエクスポート・セッションは終了します。これらのエラーは、内部的な問題が原因であるか、またはメモリなどのリソースが使用できないか、リソースを使い果たしたことが原因で発生します。たとえば、catexp.sql スクリプトが実行されていない場合、エクスポート・ユーティリティによって、次のようなリカバリ不能エラー・メッセージが発行されます。

EXP-00024: エクスポート・ビューがインストールされていません。DBA に連絡してください。

## 完了メッセージ

問題なくエクスポートが完了した場合は、次のメッセージが表示されます。

エクスポートは警告なしで正常終了しました。

リカバリ可能エラーが1つ以上発生しても、エクスポートがそのまま続行され、処理が完了した場合は、次のメッセージが表示されます。

エクスポートは正常に終了しましたが、警告が発生しました。

リカバリ不能なエラーが発生した場合は、エクスポートは即時終了し、次のメッセージが表示されます。

エラーが発生したためエクスポートを終了します。

**参照：**『Oracle9i データベース・エラー・メッセージ』およびご使用のシステム固有の Oracle マニュアルを参照してください。

## 終了コードによる結果の検査と表示

エクスポート・ユーティリティでは、エクスポートの完了後、すぐに実行結果を確認できます。プラットフォームによっては、エクスポートの実行結果はログ・ファイルに記録されるのみでなく、プロセス終了コードにも通知されます。これによって、コマンドラインやスクリプトからの出力を確認できます。表 1-5 に、それぞれの結果に対応する終了コードを示します。

表 1-5 エクスポート時の終了コード

結果	終了コード
エクスポートは警告なしで正常終了しました。	EX_SUCC
エクスポートは正常に終了しましたが、警告が発生しました。	EX_OKWARN
エラーが発生したためエクスポートを終了します。	EX_FAIL

UNIX の場合、終了コードは次のようになります。

```
EX_SUCC    0
EX_OKWARN  0
EX_FAIL    1
```

## 従来型パス・エクスポートおよびダイレクト・パス・エクスポート

エクスポートでは次の2つの方式で表データをエクスポートできます。

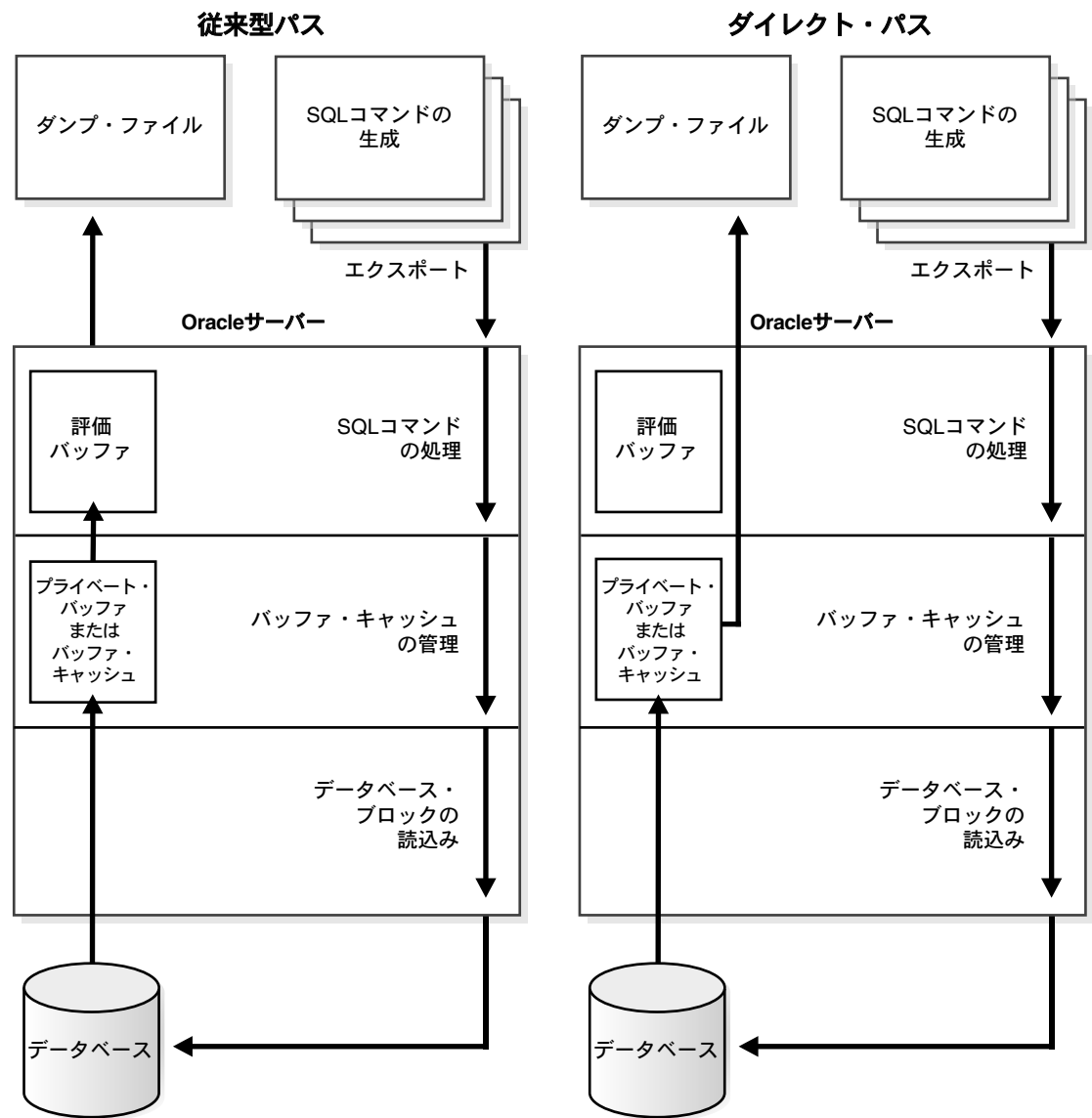
- 従来型パス・エクスポート
- ダイレクト・パス・エクスポート

従来型パス・エクスポートでは、SQL の `SELECT` 文によって、データベースの表からデータが抽出されます。データはディスクからバッファ・キャッシュに読み込まれ、行は評価バッファに転送されます。式の評価が終了すると、そのデータはエクスポートを実行するクライアントへ転送され、そこでエクスポート・ファイルに書き込まれます。

ダイレクト・パス・エクスポートでは、データがディスクからバッファ・キャッシュに読み込まれ、行がエクスポート・クライアントに直接転送されるため、従来型パス・エクスポートに比べて非常に高速です。評価バッファはバイパスします。データは、すでにエクスポート・ユーティリティが要求する形式になっているため、不要なデータ変換をする必要がありません。データはエクスポート・クライアントに転送され、このクライアントでエクスポート・ファイルに書き込まれます。

図 1-2 に、従来型パス・エクスポートとダイレクト・パス・エクスポートでのデータの抽出方法の違いを示します。

図 1-2 従来型パス・エクスポートおよびダイレクト・パス・エクスポートでのデータベースの読み込み



## ダイレクト・パス・エクスポートの起動

ダイレクト・パス・エクスポートを起動するには、コマンドライン方式またはパラメータ・ファイルのいずれかを使用する必要があります。対話方式でダイレクト・パス・エクスポートを起動することはできません。

ダイレクト・パス・エクスポートを使用するには、コマンドラインまたはパラメータ・ファイルで `DIRECT=y` パラメータを指定します。デフォルトは `DIRECT=n` です。この場合、従来型パスで表データが抽出されます。

また、エクスポートの `BUFFER` パラメータを使用できるのは、従来型パス・エクスポートのみのため注意してください。ダイレクト・パス・エクスポートでは、エクスポート・ファイルへの書き込みに使用するバッファのサイズは、`RECORDLENGTH` パラメータで指定します。

リリース 8.1.5 より前のエクスポート・ユーティリティの場合、オブジェクトおよび LOB を含む表に対してダイレクト・パス・エクスポートを使用できませんでした。使用しても、行はエクスポートされませんでした。この動作は変更されています。オブジェクトおよび LOB を含む表の行は、ダイレクト・パスが指定されていても、従来型パスを使用してエクスポートできます。インポート・ユーティリティによって、ダイレクト・パス・ダンプ・ファイル内でこれらの従来型パス表が正しく処理されます。

## ダイレクト・パス・エクスポートのセキュリティに関する考慮点

仮想プライベート・データベース (VPD) および Oracle Label Security は、ダイレクト・パス・エクスポート中には施行されません。

次のユーザーは、データベースからデータを抽出するために使用するエクスポート・モード、アプリケーションまたはユーティリティにかかわらず、VPD および Oracle Label Security を施行する必要はありません。

- データベース・ユーザー SYS
- 直接またはデータベース・ロールを介して、Oracle9i の EXEMPT ACCESS POLICY 権限を付与されたデータベース・ユーザー

EXEMPT ACCESS POLICY 権限を付与されたユーザーは、VPD および Oracle Label Security を施行する必要はありません。これは強力な権限であるため、慎重に管理する必要があります。この権限は、SELECT、INSERT、UPDATE および DELETE などの従来のオブジェクト権限の施行には影響しません。ユーザーが EXEMPT ACCESS POLICY 権限を付与されていても、これらの権限は施行されます。

### 参照：

- 1-60 ページの「[ファイングレイン・アクセス・コントロールのサポート](#)」を参照してください。
- 『Oracle9i アプリケーション開発者ガイド - 基礎編』も参照してください。

## ダイレクト・パス・エクスポートのパフォーマンスの問題

ダイレクト・パス・エクスポートの起動時に、RECORDLENGTH パラメータの値を大きくすると、パフォーマンスが向上する場合があります。実際のパフォーマンス向上の度合いは、次の要因によって異なります。

- DB\_BLOCK\_SIZE
- 表の列の型
- I/O レイアウト（エクスポート・ファイルの転送先ドライブは、データベース・ファイルが常駐するディスク・ドライブとは別にします。）

RECORDLENGTH の値は、次のように設定することをお勧めします。

- ファイル・システムの I/O ブロック・サイズの倍数であること。
- DB\_BLOCK\_SIZE の倍数であること。

## ネットワークに関する考慮点

この項では、ネットワークを介したエクスポートおよびインポートの実行時に考慮すべき点について説明します。

### ネットワークを介してエクスポート・ファイルを転送する方法

エクスポート・ファイルはバイナリ形式であるため、ネットワークを介してそのエクスポート・ファイルを転送するときは、バイナリ転送をサポートしているプロトコルを使用して、ファイルが破損しないようにしてください。たとえば、FTP などのファイル転送プロトコルを使用して、バイナリ・モードでファイルを転送します。エクスポート・ファイルをキャラクタ・モードで送信すると、ファイルのインポート時にエラーが発生します。

### Oracle Net でのエクスポートおよびインポート

Oracle Net を使用すると、ネットワークを介してエクスポートおよびインポートを実行できます。たとえば、エクスポート・ユーティリティをローカルで実行して、リモート Oracle データベースのデータをローカル・エクスポート・ファイルに書き込むことができます。また、インポート・ユーティリティをローカルで実行して、リモート Oracle データベースのデータを読み込むことができます。

Oracle Net でインポート・ユーティリティを使用するには、exp コマンドまたは imp コマンドに *username/password* を入力するときに接続修飾文字列 *@connect\_string* を指定する必要があります。この句の構文の詳細は、ご使用の Oracle Net プロトコルのユーザーズ・ガイドを参照してください。



**参照：** 次のマニュアルを参照してください。

- 『Oracle9i Net Services 管理者ガイド』
- 『Oracle9i Heterogeneous Connectivity Administrator's Guide』

## キャラクタ・セットおよびグローバリゼーション・サポートに関する考慮点

この項では、グローバリゼーション・サポートに関連するエクスポート・ユーティリティおよびインポート・ユーティリティの動作について説明します。

### キャラクタ・セット変換

エクスポート・ユーティリティは、常に、エクスポート・サーバーのキャラクタ・セットで Unicode データを含むユーザー・データをエクスポートします。キャラクタ・セットは、データベース作成時に指定されます。

データは、インポート・ユーティリティによって、自動的にインポート・サーバーのキャラクタ・セットに変換されます。

8 ビット・キャラクタ・セットのエクスポート・ファイルをインポートすると、8 ビット文字の一部が消去されることがあります（同等の 7 バイトに変換されます）。これが発生するのは、クライアント・システムに、システム固有の 7 ビット・キャラクタ・セットが存在するか、オペレーティング・システム環境変数 `NLS_LANG` が 7 ビット・キャラクタ・セットに設定されている場合です。アクセント記号が付いている文字からアクセント記号が消去されるのが最もよく見られる例です。

エクスポート・ユーティリティおよびインポート・ユーティリティでは、データをエクスポートまたはインポートする前に、必要なキャラクタ・セット変換に関する説明が表示されます。

---

---

**注意：** `CREATE TABLE` コマンドなどの DDL は、クライアントのキャラクタ・セットでエクスポートされます。

---

---

## 変換によるキャラクタ・セットのソート順への影響

エクスポート・キャラクタ・セットのソート順が、インポート・キャラクタ・セットと異なる場合、キャラクタ列をパーティション化した表では、結果が保証されません。たとえば、次のような ASCII キャラクタ・セットのデータベースの表定義について考えてみます。

```
CREATE TABLE partlist
(
  part      VARCHAR2(10),
  partno    NUMBER(2)
)
PARTITION BY RANGE (part)
(
  PARTITION part_low VALUES LESS THAN ('Z')
    TABLESPACE tbs_1,
  PARTITION part_mid VALUES LESS THAN ('z')
    TABLESPACE tbs_2,
  PARTITION part_high VALUES LESS THAN (MAXVALUE)
    TABLESPACE tbs_3
);
```

ASCII キャラクタ・セットでは、Z の後に z があるため、このパーティション・スキームには意味があります。

この表が EBCDIC キャラクタ・セット・ベースのデータベースにインポートされると、EBCDIC キャラクタ・セットでは、z が Z の前にあるため、part\_mid パーティションのすべての行が、part\_low パーティションに移行します。希望する結果を得るには、partlist 表の所有者は、インポート後に表を再パーティション化する必要があります。

**参照：**『Oracle9i Database グローバリゼーション・サポート・ガイド』を参照してください。

## マルチバイト・キャラクタ・セットおよびエクスポートとインポート

---

---

**注意：** エクスポート・クライアントとエクスポート・サーバーのキャラクタ・セット幅が異なる場合、変換によってデータが長くなると、データが切り捨てられることがあります。データが切り捨てられると、エクスポート・ユーティリティによって警告メッセージが表示されます。

---

---

## インスタンス親和性とエクスポート

インスタンス親和性を使用して、インポートおよびエクスポートするデータベース内のインスタンスにジョブを関連付けることができます。Oracle8、Oracle8i および Oracle9i を組み合わせて使用している場合は、互換性の問題に注意してください。

**参照：** 次のマニュアルを参照してください。

- 『Oracle9i データベース管理者ガイド』
- 『Oracle9i データベース・リファレンス』
- 『Oracle9i データベース移行ガイド』

## データベース・オブジェクトのエクスポートに関する考慮点

次の項からは、特定のデータベース・オブジェクトをエクスポートするときに考慮すべき点について説明します。

### 順序のエクスポート

エクスポート中にトランザクションが順序番号にアクセスすると、順序番号はスキップされる可能性があります。順序番号がスキップされないようにするには、エクスポート中に順序番号にアクセスしないようにします。

順序番号がスキップされる可能性があるのは、キャッシュされている順序番号が使用中の場合のみです。順序番号のキャッシュが割り当てられている場合は、現行のデータベースでこれらの順序番号を使用できます。エクスポートされる値は、その次（キャッシュされている値の後）の順序番号です。キャッシュされても使用されていない順序番号は、順序のインポート時に失われます。

### LONG データ型および LOB データ型のエクスポート

エクスポート時、LONG データ型は、セクション単位でフェッチされます。ただし、各行のすべてのデータ（LONG データ型を含む）を保持できるだけのメモリーが使用可能である必要があります。

LONG 列の長さは、最大 2GB です。

LOB 列のすべてのデータを同時にメモリーに置いておく必要はありません。LOB データのロードおよびアンロードはセクション単位で行われます。

## 外部関数ライブラリのエクスポート

外部関数ライブラリの内容は、エクスポート・ファイルにはエクスポートされません。全データベース・モード・エクスポートおよびユーザー・モード・エクスポートの場合、ライブラリの仕様（名前、位置）のみがエクスポートされます。データベースを新しい場所に移動する場合、ライブラリの実行可能ファイルを移動し、ライブラリの仕様を更新する必要があります。

## オフライン・ローカル管理表領域のエクスポート

エクスポート中のデータにオフライン・ローカル管理表領域が含まれている場合、表領域の定義を完全にエクスポートできず、エラー・メッセージが表示されます。データはインポートできますが、インポートの前に、まずオフライン・ローカル管理表領域を作成し、不完全な表領域を参照する DDL コマンドでエラーが発生しないようにする必要があります。

## ディレクトリ別名のエクスポート

ディレクトリ別名の定義は、全データベース・モード・エクスポートの場合のみエクスポートされます。データベースを新しい場所に移動する場合、データベース管理者は、その新しい場所に対応するようにディレクトリ別名を更新する必要があります。

ディレクトリ別名は、ユーザー・モード・エクスポートや表モード・エクスポートの場合はエクスポートされません。したがって、ディレクトリ別名を使用する前に、ターゲット・システムにディレクトリ別名が作成されていることを確認する必要があります。

## BFILE 列および属性のエクスポート

エクスポート・ファイルには、BFILE 列またはその属性から参照される外部ファイルの内容は格納されません。ファイルの名前およびディレクトリ別名のみが、エクスポート時にコピーされ、インポート時にリストアされます。旧ディレクトリからではファイルにアクセスできない場所にデータベースを移動する場合、DBA は、指定されたファイルが格納されているディレクトリを、アクセス可能な新しい場所へ移動する必要があります。

## 外部表

外部表の内容は、エクスポート・ファイルにはエクスポートされません。全データベース・モード・エクスポートおよびユーザー・モード・エクスポートの場合、表の仕様（名前、位置）のみがエクスポートされます。データベースを新しい場所に移動する場合、外部データを手動で移動して表の仕様を更新する必要があります。

## オブジェクト型定義のエクスポート

どのエクスポート・モードでも、エクスポートされる表で使用されているオブジェクト型定義に関する情報はエクスポートされます。オブジェクト名、オブジェクト識別子、オブジェクト構成などの情報は、ターゲット・システムでのオブジェクト型とエクスポート・ファイルに格納されているオブジェクト・インスタンスに整合性があることを検証するために必要となります。これによって、インポート時に、表に必要なオブジェクト型が同一のオブジェクト識別子で作成されます。

ただし、表モード、ユーザー・モードおよび表領域モードで、オブジェクト型に対する実行権がないユーザーがエクスポートを実行している場合は、表に必要なすべてのオブジェクト型定義が、エクスポート・ファイルに保持されるとはかぎりません。この場合、同じオブジェクト型の識別子および同じオブジェクトの構成を持つ型の存在を検証するために必要な情報のみが、インポートのターゲット・システムに書き込まれます。

DBA に協力を求めて型定義を作成するか、DBA が実行した全データベース・モードまたはユーザー・モードのエクスポートから型定義をインポートすることによって、ターゲット・システムに適切な型定義が確実に存在するようにしてください。

すべてのオブジェクト型定義を保持するには、定期的に全データベース・エクスポートを実行することが重要です。また、別のユーザーのスキーマに属するオブジェクト型定義を使用する場合は、DBA が、適切なユーザー・グループのユーザー・モードでエクスポートを実行する必要があります。たとえば、ユーザー `scott` が所有する `table1` に `blake` のオブジェクト型である `type1` が存在する場合、この表に必要な型定義を保持するには、DBA がユーザー・モードで `blake` および `scott` の両方を指定してエクスポートを実行する必要があります。

## ネストした表のエクスポート

ネストした表については、外部表がエクスポートされる場合は、必ず内部表のデータもエクスポートされます。内部のネストした表を指定することはできませんが、それらを個別にエクスポートすることはできません。

## アドバンスト・キュー（AQ）表のエクスポート

キューは表に実装されています。キューのエクスポートおよびインポートは、その基礎となるキュー表および関連するディクショナリ表のエクスポートおよびインポートになります。キューのエクスポートおよびインポートは、キュー表単位でのみ実行できます。

キュー表をエクスポートすると、表定義に関する情報とキュー・データの両方がエクスポートされます。キュー表データと表定義の両方がエクスポートされるため、キュー表のインポート時に、インポートを実行したユーザーがアプリケーション・レベルでのデータの整合性をメンテナンスすることになります。

**参照：**『Oracle9i アプリケーション開発者ガイド - アドバンスト・キューイング』を参照してください。

## シノニムのエクスポート

シノニムおよびもう 1 つのオブジェクトとして使用される名前を参照する、コンパイル済のオブジェクトをエクスポートする場合は、注意が必要です。これらのオブジェクトをエクスポートおよびインポートすると、強制的に再コンパイルされ、オブジェクトの定義が変更される可能性があります。

次の例では、この問題について説明します。

```
CREATE PUBLIC SYNONYM emp FOR scott.emp;  
  
CONNECT blake/paper;  
CREATE TRIGGER t_emp BEFORE INSERT ON emp BEGIN NULL; END;  
CREATE VIEW emp AS SELECT * FROM dual;
```

前述の例では、データベースがエクスポートされた場合、トリガーの `emp` に対する参照では、`scott` の表ではなく `blake` のビューが参照されます。このため、インポート時に `t_emp` トリガーを再確立しようとする、エラーになります。

### Java シノニムに関連して発生する可能性があるエクスポート・エラー

対応する `DBMS_JAVA` パッケージがないとき、または `Java` がロードされていないか、不適切にロードされているときに、エクスポート操作で `DBMS_JAVA` という名前のシノニムをエクスポートしようとした場合、エラーが発生して、エクスポートは終了します。この場合に生成されるエラー・メッセージには `EXP-00008`、`ORA-00904`、`ORA-29516` などがあります。

`Java` が使用可能な場合、エクスポート・ユーティリティを再実行する前に、`DBMS_JAVA` シノニムと `DBMS_JAVA` パッケージの両方が作成済みであり、有効であることを確認します。

`Java` が使用可能でない場合、エクスポート・ユーティリティを再実行する前に `Java` 関連のオブジェクトを削除します。

## ファイングレイン・アクセス・コントロールのサポート

ファイングレイン・アクセス・コントロール・ポリシーを使用可能にして、表をエクスポートできます。その場合は、次のことに注意してください。

- ファイングレイン・アクセス・コントロール・ポリシーが使用可能な表を含むエクスポート・ファイルからインポートする場合は、適切な権限（特に、表のセキュリティ・ポリシーを回復するための `DBMS_RLS` パッケージに対する `EXECUTE` 権限）が必要です。ファイングレイン・アクセス・ポリシーが使用可能な表をエクスポートするための正しい権限が付与されていない場合は、読込み権限のある行のみがエクスポートされます。
- `SELECT` 文でファイングレイン・アクセス・コントロールが使用可能な場合、ファイングレイン・アクセスにより問合せがライトされるため、従来型パス・エクスポートでは表全体をエクスポートできない場合があります。

- ユーザー SYS、EXPORT\_FULL\_DATABASE ロールを使用可能なユーザーまたは EXEMPT ACCESS POLICY を付与されたユーザーのみがファイングレイン・アクセス・コントロールを持つ表に対してダイレクト・パス・エクスポートを実行できます。

**参照：** ファイングレイン・アクセス・コントロールの詳細は、『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。

## トランスポートابل表領域

トランスポートابل表領域機能は、一連の表領域を、ある Oracle データベースから他の Oracle データベースに移動できる機能です。

一連の表領域を移動またはコピーするには、表領域を読み込み専用にし、表領域のデータ・ファイルをコピーしてから、エクスポートおよびインポートを使用して、データ・ディクショナリに格納されているデータベース情報（メタデータ）を移動します。データ・ファイルおよびメタデータのエクスポート・ファイルの両方を、ターゲット・データベースにコピーする必要があります。これらのファイルの転送は、オペレーティング・システムのコピー機能、バイナリ・モード FTP、CD-ROM への出力などのようなバイナリ・ファイルのコピー機能を使用して行われます。

データ・ファイルのコピーおよびメタデータのエクスポートの後、表領域を任意に読み書き両用モードにできます。

次のパラメータで、トランスポートابل表領域のメタデータのエクスポートを使用可能にできます。

- TABLESPACES
- TRANSPORT\_TABLESPACE

詳細は、1-31 ページの「[TABLESPACES](#)」および 1-31 ページの「[TRANSPORT\\_TABLESPACE](#)」を参照してください。

**参照：**

- トランスポートابل表領域の管理の詳細は、『Oracle9i データベース管理者ガイド』を参照してください。
- トランスポートابل表領域の詳細は、『Oracle9i データベース概要』を参照してください。

## 読み専用データベースからのエクスポート

ソース・データベースからメタデータを抽出するには、エクスポートで順序付けする句（ソート操作）を含む問合せを使用します。これらの問合せが成功するには、エクスポートを実行しているユーザーがディスク上のソート・セグメントを割り当てることができる必要があります。これらのソート・セグメントを読み専用データベースに割り当てするには、ユーザーの一時表領域が、ローカル管理一時表領域を示すように設定する必要があります。

**参照：** この環境の設定の詳細は、『Oracle9i Data Guard 概要および管理』を参照してください。

## エクスポート・ユーティリティおよびインポート・ユーティリティを使用したデータベース移行のパーティション化

エクスポート・ユーティリティおよびインポート・ユーティリティを使用して大規模データベースを移行する場合、移行を複数のエクスポート・ジョブおよびインポート・ジョブにパーティション化するとより効率的です。移行をパーティション化する場合は、次のメリットおよびデメリットに注意してください。

### 移行をパーティション化する場合のメリット

移行をパーティション化すると、次のメリットがあります。

- 多くのサブジョブをパラレルに実行できるため、移行に必要な時間を削減できます。
- 最初のエクスポート・ジョブが完了するとすぐにインポート・ユーティリティを起動できます。

### 移行をパーティション化する場合のデメリット

移行をパーティション化すると、次のデメリットがあります。

- エクスポートおよびインポートのプロセスがより複雑になります。
- ある型のオブジェクトに対する相互スキーマ参照のサポートは損なわれます。たとえば、異なるスキーマの表に対する外部キー制約を持つ表がスキーマに含まれている場合、その表を依存スキーマにインポートしたときに、必要な親レコードが存在しない場合があります。



## エクスポート・ユーティリティおよびインポート・ユーティリティを使用したデータベース移行のパーティション化方法

データベースの移行をパーティション化方法で実行するには、次の手順に従います。

1. データベースのすべての最上位メタデータに、次のコマンドを発行します。
  - a. `exp dba/password FILE=full FULL=y CONSTRAINTS=n TRIGGERS=n ROWS=n INDEXES=n`
  - b. `imp dba/password FILE=full FULL=y`
2. データベースの各スキーマ *n* に、次のコマンドを発行します。
  - a. `exp dba/password OWNER=scheman FILE=scheman`
  - b. `imp dba/password FILE=scheman FROMUSER=scheman TOUSER=scheman IGNORE=y`

すべてのエクスポートはパラレルで実行できます。full.dmp のインポートが完了すると、残りのインポートもパラレルで実行できます。

## リリースおよびバージョンが異なるエクスポート・ユーティリティの使用法

この項では、リリースが異なるエクスポート・ユーティリティおよび Oracle データベース・サーバーの使用に関連する互換性の問題について説明します。

リリースが異なる Oracle データベース・サーバー間でデータを移動させる場合、常に次の基本的な規則が適用されます。

- インポート・ユーティリティとデータのインポート先であるデータベース（ターゲット・データベース）のバージョンは同じである必要があります。
- エクスポート・ユーティリティのバージョンは、ソース・データベースまたはターゲット・データベースの下位の方のバージョンと同じである必要があります。

たとえば、上位バージョンのデータベースにインポートするエクスポート・ファイルを作成するには、ソース・データベースと同じバージョンのエクスポート・ユーティリティを使用します。それに対し、下位バージョンのデータベースにインポートするエクスポート・ファイルを作成するには、ターゲット・データベースと同じバージョンのエクスポート・ユーティリティを使用します。次に、バージョン固有の情報を示します。

- Oracle7 データベース・サーバーで Oracle バージョン 6 のエクスポート・ユーティリティを実行して、Oracle7 データベースから Oracle バージョン 6 のエクスポート・ファイルを作成する場合は、まず、Oracle7 データベースで catexp6.sql スクリプトを実行する必要があります。このスクリプトを実行すると、エクスポート・ユーティリティで Oracle7 データベースを Oracle バージョン 6 のデータベースとして認識可能なエクスポート・ビューが作成されます。

- Oracle8i データベースで Oracle7 のエクスポート・ユーティリティを実行して、Oracle8i データベースから Oracle7 のエクスポート・ファイルを作成する場合は、まず、Oracle8i データベースで catexp7.sql スクリプトを実行する必要があります。このスクリプトを実行すると、エクスポート・ユーティリティで Oracle8i データベースを Oracle7 データベースとして認識可能なエクスポート・ビューが作成されます。

---

**注意：** catexp6.sql スクリプトおよび catexp7.sql スクリプトは、ユーザー SYS が実行する必要があります。これらのスクリプトは、複数回実行する必要はありません。

---

- 通常は、Oracle8 のエクスポート・ユーティリティを使用して、Oracle9i サーバーからエクスポートし、Oracle8 のエクスポート・ファイルを作成できます。詳細は、1-65 ページの「[Oracle9i データベースからの Oracle8 のエクスポート・ファイルの作成](#)」を参照してください。

## リリースおよびバージョンが異なるエクスポート・ユーティリティおよびインポート・ユーティリティの使用上の制限

異なるリリースのエクスポート・ユーティリティおよびインポート・ユーティリティを使用する場合、次の制限が適用されます。

- エクスポート・ダンプ・ファイルは、特別なバイナリ形式で格納されているため、インポート・ユーティリティによる読込み専用です。
- すべてのエクスポート・ダンプ・ファイルは、上位リリースの Oracle データベース・サーバーにインポートできます。
- インポート・ユーティリティでは、リリース 5.1.22 以上のエクスポート・ユーティリティで作成されたエクスポート・ダンプ・ファイルの読込みができます。
- インポート・ユーティリティでは、上位のメンテナンス・リリースまたはバージョンのエクスポート・ユーティリティで作成されたエクスポート・ダンプ・ファイルの読込みはできません。たとえば、リリース 8.0 のインポート・ユーティリティでは、リリース 8.1 のエクスポート・ダンプ・ファイルはインポートできません。また、バージョン 7 のインポート・ユーティリティでは、バージョン 8 のエクスポート・ダンプ・ファイルはインポートできません。
- Oracle バージョン 6 以下のエクスポート・ユーティリティは、Oracle8 以上のデータベースには使用できません。
- 下位バージョンのエクスポート・ユーティリティを上位バージョンの Oracle データベース・サーバーで実行すると、下位バージョンに存在しないデータベース・オブジェクトのカテゴリは、常にエクスポートから除外されます。たとえば、Oracle7 データベース・サーバーにはパーティション表が存在しませんでした。そのため、Oracle8 のパー

ティション表を Oracle7 データベースに移動させる必要がある場合は、まず、表を非パーティション表に再編成する必要があります。

- ダイレクト・パスの場合も従来型パスの場合も、Oracle9i のエクスポート・ユーティリティを使用して作成されたエクスポート・ファイルは、旧リリースのインポート・ユーティリティとは互換性がないため、インポートに使用できるのは Oracle9i のインポート・ユーティリティのみです。下位互換性が問題となる場合は、Oracle9i データベースに対して下位のリリースまたはバージョンのエクスポート・ユーティリティを使用します。

## リリースが異なるエクスポート・ユーティリティおよびインポート・ユーティリティの使用例

表 1-6 に、異なるリリースの Oracle データベース・サーバー間でデータを移動させる場合に使用するエクスポート・ユーティリティおよびインポート・ユーティリティのリリースの例を示します。

**表 1-6 リリースが異なるエクスポート・ユーティリティおよびインポート・ユーティリティの使用**

エクスポート元 → インポート先	使用するエクスポート・ ユーティリティのリリース	使用するインポート・ユー ティリティのリリース
リリース 7.3.3 → リリース 8.1.6	リリース 7.3.3	リリース 8.1.6
リリース 8.1.6 → リリース 8.1.6	リリース 8.1.6	リリース 8.1.6
リリース 8.1.5 → リリース 8.0.6	リリース 8.0.6	リリース 8.0.6
リリース 8.1.7 → リリース 8.1.6	リリース 8.1.6	リリース 8.1.6
リリース 8.1.7 → リリース 7.3.4	リリース 7.3.4 <sup>1</sup>	リリース 7.3.4
リリース 1 (9.0.1) → リリース 8.1.6	リリース 8.1.6	リリース 8.1.6
リリース 1 (9.0.1) → リリース 2 (9.2)	リリース 1 (9.0.1)	リリース 2 (9.2)

<sup>1</sup> catexp7.sql を Oracle8i リリース 8.1.7 データベースで 1 度も実行していない場合、まずこれを行って Oracle7 データ・ディクショナリ・ビューを作成する必要があります。これを行わないかぎり、Oracle8i リリース 8.1.7 データベースでは、リリース 7.3.4 のエクスポート・ユーティリティを正常に実行できません。

## Oracle9i データベースからの Oracle8 のエクスポート・ファイルの作成

Oracle9i データベースから Oracle8 のエクスポート・ファイルを作成する場合、特別な処理は必要ありません。ただし、Oracle9i データベースで Oracle8 のエクスポート・ユーティリティを使用した場合、次の機能はサポートされません。

- ダイレクト・パス・ロード (DIRECT=y) を指定した場合、オブジェクトおよび LOB を含む表の行はエクスポートされません。
- ディメンションはエクスポートされません。
- ファンクションおよびドメイン索引はエクスポートされません。
- セカンダリ・オブジェクト (表、索引、順序など、ドメイン索引のサポートで作成されるもの) はエクスポートされません。
- ビュー、プロシージャ、ファンクション、パッケージ、型本体および Oracle9i の新機能への参照を含む型は、コンパイルされない場合があります。
- SQL の DDL が、SQL ではなくストアド・プロシージャとして実装されているオブジェクトはエクスポートされません。
- アクションが CALL 文であるトリガーは、エクスポートされません。
- 論理 ROWID 列、主キー参照またはユーザー定義 OID 列を含む表はエクスポートされません。
- 一時表はエクスポートされません。
- 索引構成表 (IOT) は圧縮前の状態に戻ります。
- パーティション化された IOT のパーティション情報が消失します。
- 索引タイプおよび演算子はエクスポートされません。
- ローカル管理表領域、一時表領域および UNDO 表領域はエクスポートされません。
- Java ソース、Java クラスおよび Java リソースはエクスポートされません。
- VARRAY 列の、可変幅 CLOB、コレクション拡張および LOB ストレージ句、またはネストした表の拡張はエクスポートされません。
- ファイングレイン・アクセス・コントロール・ポリシーは保持されません。
- 外部表はエクスポートされません。

## 異なるリリースおよびバージョンを使用するときに発生する可能性があるエラー

この項では、互換性がないリリースまたはバージョンのエクスポート・ユーティリティおよび Oracle データベース・サーバーを使用した場合に表示されるいくつかのエラー・メッセージについて簡単に説明します。

### EXP-00024

EXP-00024: エクスポート・ビューがインストールされていません。DBA に連絡してください。

原因: 必要なエクスポート・ビューがインストールされていません。

処置: DBA に連絡して、必要なビューのインストールを要求します。

### IMP-00023

IMP-00023: インポート・ビューがインストールされていません。DBA に連絡してください。

原因: 必要なインポート・ビューがインストールされていません。

処置: DBA に連絡して、必要なビューのインストールを要求します。

### EXP-00037

EXP-00037: エクスポート・ビューはデータベースのバージョンと互換性がありません。

原因: エクスポート・ユーティリティがデータベースのバージョンより上位のバージョンです。

処置: データベースと同じバージョンのエクスポート・ユーティリティを使用します。

#### 参照:

- 『Oracle9i データベース・エラー・メッセージ』を参照してください。
- 1-64 ページの「リリースおよびバージョンが異なるエクスポート・ユーティリティおよびインポート・ユーティリティの使用上の制限」も参照してください。
- 2-72 ページの「以前のリリースの Oracle のエクスポート・ファイルの使用方法」も参照してください。



---

# インポート

この章では、インポート・ユーティリティを使用して、エクスポート・ファイルを Oracle データベースに読み込む方法について説明します。インポート・ユーティリティでは、エクスポート・ユーティリティで作成されたエクスポート・ファイルのみ読み込みができます。データベースのエクスポート方法の詳細は、[第 1 章](#)を参照してください。他のオペレーティング・システム・ファイルのデータのロードについては、[第 II 部](#)を参照してください。

この章の内容は、次のとおりです。

- [インポート・ユーティリティとは](#)
- [インポート・ユーティリティを使用する前に](#)
- [既存の表へのインポート](#)
- [インポート操作上のスキーマおよびデータベース・トリガーの影響](#)
- [インポート・ユーティリティの起動](#)
- [インポート・モード](#)
- [オンライン・ヘルプの利用](#)
- [インポート・パラメータ](#)
- [インポート・セッションの例](#)
- [対話方式の使用](#)
- [警告、エラーおよび完了メッセージ](#)
- [終了コードによる結果の検査と表示](#)
- [インポート中のエラー処理](#)
- [表レベル・インポートおよびパーティション・レベル・インポート](#)
- [索引作成およびメンテナンスの制御](#)
- [データベースの断片化を解消する方法](#)

- 
- ネットワークに関する考慮点
  - キャラクタ・セットおよびグローバリゼーション・サポートに関する考慮点
  - データベース・オブジェクトのインポートに関する考慮点
  - マテリアライズド・ビューおよびスナップショット
  - トランスポータブル表領域
  - 記憶域パラメータ
  - 表領域を削除する方法
  - 表領域を再編成する方法
  - 統計情報のインポート
  - エクスポート・ユーティリティおよびインポート・ユーティリティを使用したデータベース移行のパーティション化
  - 以前のリリースの **Oracle** のエクスポート・ファイルの使用方法

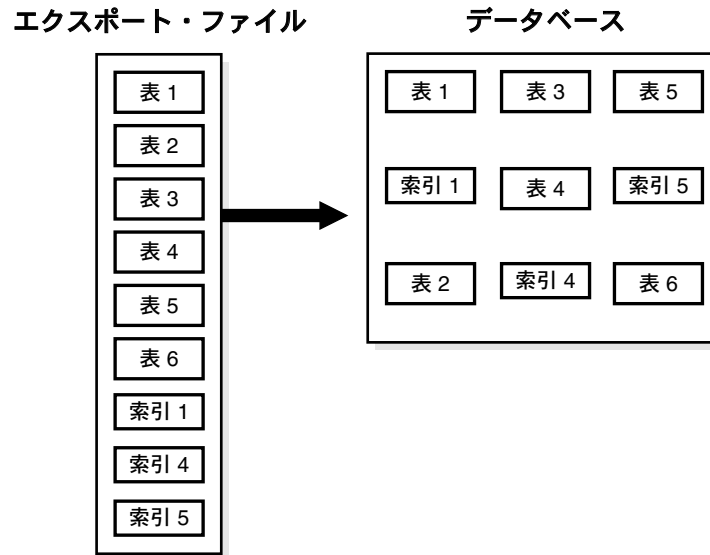


# インポート・ユーティリティとは

インポート・ユーティリティは、エクスポート・ダンプ・ファイルからオブジェクトの定義および表データを読み込みます。そして、データ・オブジェクトを Oracle データベースに挿入します。

図 2-1 に、エクスポート・ダンプ・ファイルからのインポートのプロセスを示します。

図 2-1 エクスポート・ファイルのインポート



エクスポート・ダンプ・ファイルは、Oracle のインポート・ユーティリティを使用した場合のみ読み込みが可能です。ダンプ・ファイルの作成に使用したエクスポート・ユーティリティより前のバージョンのインポート・ユーティリティは使用できません。

インポート・ユーティリティでは、リリース 5.1.22 以上のエクスポート・ユーティリティで作成されたエクスポート・ファイルの読み込みが可能です。

ASCII 固定形式ファイルまたは区切りファイルからデータをロードするには、SQL\*Loader ユーティリティを使用します。

## 参照：

- エクスポート・ユーティリティの詳細は、[第 1 章](#)を参照してください。
- SQL\*Loader ユーティリティの詳細は、[第 II 部](#)を参照してください。

## 表オブジェクト：インポートの順序

表オブジェクトは、エクスポート・ファイルから読み込まれたとおりにインポートされます。エクスポート・ファイルには、次の順序でオブジェクトが格納されています。

1. 型定義
2. 表定義
3. 表データ
4. 表索引
5. 整合性制約、ビュー、プロシージャおよびトリガー
6. ビットマップ索引、ファンクション索引およびドメイン索引

まず、新しい表が作成されます。次にデータがインポートされ、索引が作成されます。その後トリガーがインポートされ、整合性制約が新しい表で使用可能になり、ビットマップ索引、ファンクション索引またはドメイン索引（あるいはそのすべて）が作成されます。このようにインポートされると、表のインポート順序が原因でデータが拒否されることがなくなります。また、同じデータについて、トリガーが重複して2回（最初の挿入時に1回、インポート中に1回）起動すること也不再行します。

たとえば、`emp` 表に `dept` 表に対する参照整合性制約が指定されて、`emp` 表が最初にインポートされた場合、この制約が使用可能になっていると、まだ `dept` にインポートされていない部門を参照するすべての `emp` 行が拒否されます。

データが既存の表にインポートされた場合でも、インポートの順序によっては参照整合性のエラーが発生することがあります。前述の状況で、`emp` 表がすでに存在していて、参照整合性制約が使用可能になっている場合、多くの行が拒否されることがあります。

その表自体への参照整合性制約がある場合にも、同様の状況が発生します。たとえば、`emp` 表において `scott` の管理者が `drake` で、`drake` の行がまだロードされていない場合、インポートが終了した場合には有効になるとしても、`scott` の行はロードされません。

---

**注意：** このような理由から、既存の表にインポートする場合は、参照制約を使用禁止にすることをお薦めします。インポートの完了後、再度制約を使用可能にできます。

---

## インポート・ユーティリティを使用する前に

インポート・ユーティリティを使用する前に、次のことを行う必要があります。

- catexp.sql または catalog.sql スクリプトの実行
- 必要なアクセス権限を所有していることの確認

また、次の項も参照してください。

- 2-8 ページの「[既存の表へのインポート](#)」
- 2-10 ページの「[インポート操作上のスキーマおよびデータベース・トリガーの影響](#)」

### catexp.sql または catalog.sql の実行

インポート・ユーティリティを使用するには、データベースを作成または Oracle9i へ移行した後で、スクリプト catexp.sql または (catexp.sql を実行する) catalog.sql を実行する必要があります。

---

**注意：** スクリプト・ファイルの実際の名前は、システムによって異なります。スクリプトのファイル名およびそれらの実行方法については、ご使用のオペレーティング・システム固有の Oracle マニュアルを参照してください。

---

データベースに対して、catexp.sql または catalog.sql を実行するのは1回のみです。インポート・ユーティリティを実行するたびに、これらのスクリプトを再実行する必要はありません。スクリプトを実行すると、次の処理が行われ、データベースはインポートに備えて調整されます。

- IMP\_FULL\_DATABASE ロールへのすべての必要な権限の割当て
- DBA ロールへの IMP\_FULL\_DATABASE の割当て
- データ・ディクショナリへの必要なビューの作成

## アクセス権限の確認

この項では、インポート・ユーティリティを使用してユーザー自身の所有するスキーマまたは他のユーザーのスキーマにオブジェクトをインポートするために必要な権限について説明します。

インポート・ユーティリティを使用するには、Oracle データベース・サーバーにログインするための `CREATE SESSION` 権限が必要です。この権限は、データベースの作成時に設定される `CONNECT` ロールに含まれます。

他のユーザーが作成したエクスポート・ファイルをインポートすることもできます。ただし、`EXP_FULL_DATABASE` 権限を所有するユーザーが作成したエクスポート・ファイルをインポートするには、`IMP_FULL_DATABASE` 権限が必要です。通常、DBA には両方の権限が付与されています。

## オブジェクトをスキーマにインポートする方法

表 2-1 に、自分のスキーマにオブジェクトをインポートするために必要な権限を示します。これらのすべての権限は、あらかじめ `RESOURCE` ロールに含まれています。

表 2-1 自分のスキーマにオブジェクトをインポートするために必要な権限

オブジェクト	必要な権限（該当する場合の権限タイプ）
クラスタ	<code>CREATE CLUSTER</code> （システム）および表領域割当て制限、または <code>UNLIMITED TABLESPACE</code> （システム）
データベース・リンク	<code>CREATE DATABASE LINK</code> （システム）およびリモート・データベースの場合は <code>CREATE SESSION</code> （システム）
表のトリガー	<code>CREATE TRIGGER</code> （システム）
スキーマのトリガー	<code>CREATE ANY TRIGGER</code> （システム）
索引	<code>CREATE INDEX</code> （システム）および表領域割当て制限、または <code>UNLIMITED TABLESPACE</code> （システム）
整合性制約	<code>ALTER TABLE</code> （オブジェクト）
ライブラリ	<code>CREATE ANY LIBRARY</code> （システム）
パッケージ	<code>CREATE PROCEDURE</code> （システム）
プライベート・シノニム	<code>CREATE SYNONYM</code> （システム）
順序	<code>CREATE SEQUENCE</code> （システム）
スナップショット	<code>CREATE SNAPSHOT</code> （システム）
ストアド・ファンクション	<code>CREATE PROCEDURE</code> （システム）
ストアド・プロシージャ	<code>CREATE PROCEDURE</code> （システム）

表 2-1 自分のスキーマにオブジェクトをインポートするために必要な権限（続き）

オブジェクト	必要な権限（該当する場合の権限タイプ）
表データ	INSERT TABLE（オブジェクト）
表定義（コメントおよび監査オプションを含む）	CREATE TABLE（システム）および表領域割当て制限、または UNLIMITED TABLESPACE（システム）
ビュー	実表の場合は CREATE VIEW（システム） および SELECT（オブジェクト）、または SELECT ANY TABLE（システム）
オブジェクト型	CREATE TYPE（システム）
外部関数ライブラリ	CREATE LIBRARY（システム）
ディメンション	CREATE DIMENSION（システム）
演算子	CREATE OPERATOR（システム）
索引タイプ	CREATE INDEXTYPE（システム）

権限のインポート

他のユーザーに付与されている権限をインポートするには、インポートを開始したユーザーがそのオブジェクトの所有者であるか、With Grant Option 付きのオブジェクト権限を所有している必要があります。表 2-2 に、権限がターゲット・システムで有効となるために必要な条件を示します。

表 2-2 権限のインポートに必要な権限

権限	条件
オブジェクト権限	オブジェクトがユーザーのスキーマに存在しているか、 ユーザーが With Grant Option 付きのオブジェクト権限を所有しているか、または IMP_FULL_DATABASE ロールを使用可能にする必要があります。
システム権限	ユーザーが With Admin Option および SYSTEM 権限を所有している必要があります。

他のスキーマへのオブジェクトのインポート

オブジェクトを他のユーザーのスキーマにインポートする場合は、IMP\_FULL\_DATABASE ロールを使用可能にしておく必要があります。

### システム・オブジェクトのインポート

システム・オブジェクトを全データベース・エクスポート・ファイルからインポートする場合は、IMP\_FULL\_DATABASE ロールを使用可能にする必要があります。エクスポート・ファイルが全エクスポートの場合にパラメータ FULL を指定すると、次のシステム・オブジェクトがインポート対象に含まれます。

- プロファイル
- パブリック・データベース・リンク
- パブリック・シノニム
- ロール
- ロールバック・セグメント定義
- リソース・コスト
- 外部関数ライブラリ
- コンテキスト・オブジェクト
- システム・プロシージャ・オブジェクト
- システム監査オプション
- システム権限
- 表領域定義
- 表領域割当て制限
- ユーザー定義
- ディレクトリ別名
- システム・イベント・トリガー

## 既存の表へのインポート

この項では、既存の表にデータをインポートする場合に考慮すべき点について説明します。

## データのインポート前に手動で表を作成する方法

エクスポート・ファイルから表にデータをインポートする前に手動で表を作成する場合は、以前使用した表定義を使用するか、互換性のある形式を使用して表を作成します。たとえば、列幅の増加および列順序の変更はできますが、次のことはできません。

- NOT NULL 列の追加
- 互換性がないデータ型への列のデータ型の変更（たとえば、LONG 型から NUMBER 型への変更）
- 表で使用されているオブジェクト型定義の変更
- DEFAULT 列値の変更

---

**注意：** データをインポートする前に表を手動で作成した場合は、表がすでに存在するため、エクスポート・ダンプ・ファイルで CREATE TABLE 文を実行するとエラーが発生します。このエラーを回避して、表へのデータのロードを継続するには、インポートパラメータを IGNORE=y に設定します。設定しない場合、表作成エラーが発生し、データは表にロードされません。

---

## 参照制約を使用禁止にする方法

通常のインポートの順序では、参照制約はすべての表がインポートされた後にインポートされます。この順序でインポートすることによって、まだインポートされていないデータに対する参照整合性制約が存在する場合に発生するエラーを回避できます。

データが既存の表にロードされる場合に、このようなエラーが発生することがあります。たとえば、表 emp で mgr 列に参照整合性制約が定義されており、その制約によって表 emp 内のマネージャ番号が検証される場合、マネージャの行がインポートされていない時点では、従業員の行が完全に基準を満たしていても、参照整合性制約の違反になることがあります。

このようなエラーが発生すると、エラー・メッセージが生成され、失敗した行をバイパスして、引き続き他の行が表にインポートされます。制約を手動で使用禁止にすると、このエラーを回避できます。

複数の表にまたがって参照制約が存在すると、問題になることがあります。たとえば、emp 表の順序がエクスポート・ファイル内で dept 表より先であるにもかかわらず、emp 表から dept 表へ参照チェックが行われると、参照制約違反のため、emp 表のいくつかの行がインポートされないことがあります。

このようなエラーが発生しないようにするには、データを既存の表にインポートするときに、参照整合性制約を使用禁止にします。

## 手動によるインポートの順序付け

インポート後に制約が再び使用可能にされると、表全体がチェックされますが、大きな表の場合はチェックに時間がかかることがあります。表のチェックにかかる時間が長すぎる場合、インポートを手動で順序付ける方が効率的なこともあります。

そのためには、エクスポート・ファイルからのインポートを1回ではなく複数回に分けて実行します。まず、参照チェックのターゲットである表をインポートします。次に、これらの表を参照する表をインポートします。表が循環的に相互参照している場合、および表がその表自体を参照している場合を除き、この方法は有効です。

## インポート操作上のスキーマおよびデータベース・トリガーの影響

特定のスキーマに対する DDL イベントまたはデータベースに対する DDL 関連イベントで実行するように定義されているトリガーは、システム・トリガーです。これらのトリガーは、特定のインポート操作に悪い影響を与える場合があります。たとえば、表などのデータベース・オブジェクトを正常に再作成できない場合があります。これによって、トリガーが原因で問題が発生したということがわからないエラーが返されます。

データベース管理者およびシステム・トリガーを作成するユーザーは、このようなトリガーによって、ユーザーが、権限を持つデータベースの操作を実行できなくなることがないように確認する必要があります。システム・トリガーをテストするには、次の手順に従います。

1. トリガーを定義します。
2. データベース・オブジェクトを作成します。
3. 表モードまたはユーザーモードでオブジェクトをエクスポートします。
4. オブジェクトを削除します。
5. オブジェクトをインポートします。
6. オブジェクトが正常に再作成されていることを確認します。

---

**注意：** 全エクスポートでは、スキーマ SYS が所有するトリガーはエクスポートされません。SYS トリガーは、全インポートの前後のいずれかに手動で再作成する必要があります。SYS トリガーによってインポートの進行を妨げるような処理が定義されないように、SYS トリガーはインポートの後に再作成することをお勧めします。

---



## インポート・ユーティリティの起動

次のいずれかの方法を使用して、インポート・ユーティリティの起動およびパラメータの指定ができます。

- コマンドライン
- 対話方式のインポート・プロンプト
- パラメータ・ファイル

これらのいずれかの方法でインポート・ユーティリティを起動する前に、使用可能なパラメータについての説明を必ず読んでください。詳細は、2-15 ページの「[インポート・パラメータ](#)」を参照してください。

### コマンドライン

次の構文を使用して、すべての有効なパラメータおよびその値をコマンドラインから指定できます。

```
imp username/password PARAMETER=value
```

または

```
imp username/password PARAMETER=(value1,value2,...,valuen)
```

システムでのコマンドラインの最大長を超える数のパラメータは指定できません。

### 対話方式のインポート・プロンプト

インポート・ユーティリティのプロンプトで各パラメータ値を入力する場合は、次の構文を使用して対話方式モードでインポート・ユーティリティを起動します。

```
imp username/password
```

値を入力する各パラメータがインポート・ユーティリティによって表示されます。この方法は下位互換性に対して提供されていますが、他の方法よりも機能的に劣るためお薦めしません。詳細は、2-45 ページの「[対話方式の使用](#)」を参照してください。

### パラメータ・ファイル

パラメータ・ファイルに、すべての有効なパラメータおよびその値を指定できます。パラメータを1つのファイルに格納することによって、パラメータを簡単に変更または再利用できるため、インポート・ユーティリティの起動方法としてパラメータを1つのファイルに格納することをお薦めします。データベースごとに別のパラメータを使用する場合は、複数のパラメータ・ファイルを作成できるので、それぞれにパラメータ・ファイルを用意すると便利です。

フラット・ファイル用のテキスト・エディタを使用してパラメータ・ファイルを作成します。コマンドライン・オプション `PARFILE=filename` を指定すると、インポート・ユーティリティは、コマンドラインからではなく指定されたファイルからパラメータを読み込みます。次に例を示します。

```
imp PARFILE=filename
imp username/password PARFILE=filename
```

最初の例では、コマンドラインで `username/password` を指定しないで、パラメータ・ファイルで指定する方法が示されています。ただし、セキュリティ上の理由から、この方法はお薦めしません。

パラメータ・ファイルは、次のいずれかの構文を使用して指定します。

```
PARAMETER=value
PARAMETER=(value)
PARAMETER=(value1, value2, ...)
```

次に、パラメータ・ファイル内のリストの一部を示します。

```
FULL=y
FILE=dbay
INDEXES=y
CONSISTENT=y
```

---

---

**注意：** パラメータ・ファイルの最大サイズは、オペレーティング・システムによって制限されます。また、パラメータ・ファイル名はオペレーティング・システムのネーミング規則に従います。詳細は、ご使用のオペレーティング・システム固有の **Oracle** マニュアルを参照してください。

---

---

シャープ (#) 記号を使用すると、パラメータ・ファイルにコメントを追加できます。シャープ (#) の右側にある文字はすべて無視されます。

コマンドラインでのパラメータの入力と同時にパラメータ・ファイルを指定できます。実際、パラメータ・ファイルとコマンドラインの両方に同じパラメータを指定することもできます。コマンドラインでの `PARFILE` パラメータと他のパラメータの位置によって、優先されるパラメータが決まります。たとえば、パラメータ・ファイル `params.dat` でパラメータ `INDEXES=y` を指定し、インポート・ユーティリティを次のコマンドで起動するとします。

```
imp username/password PARFILE=params.dat INDEXES=n
```

この場合、`INDEXES=n` は `PARFILE=params.dat` の後にあるため、パラメータ・ファイルに指定されている `INDEXES` パラメータの値は、`INDEXES=n` によって上書きされます。

**参照：**

- インポート・パラメータの詳細は、2-15 ページの「[インポート・パラメータ](#)」を参照してください。
- リモートのデータベースからインポートを指定する方法の詳細は、2-56 ページの「[Oracle Net を使用したエクスポートおよびインポート](#)」を参照してください。

## SYSDBA でのインポート・ユーティリティの起動

SYSDBA は内部的に使用され、一般ユーザーとは異なる特別な機能を持ちます。そのため、通常、次の場合以外は、インポート・ユーティリティを SYSDBA で起動する必要はありません。

- オラクル社カスタマ・サポート・センターから要求された場合
- トランスポータブル表領域セットをインポートする場合

SYSDBA でインポート・ユーティリティを起動するには、必要なパラメータまたはパラメータ・ファイル名を追加して次の構文を使用します。

```
imp \'username/password AS SYSDBA\'
```

オプションで、インスタンス名の指定もできます。

```
imp \'username/password@instance AS SYSDBA\'
```

ユーザー名またはパスワードを指定しないと、入力するように要求されます。

この例では、接続文字列全体が一重引用符およびバックスラッシュで囲まれています。これは、文字列 `AS SYSDBA` に空白が含まれるため、ほとんどのオペレーティング・システムで、文字列全体を一重引用符で囲むか、なんらかの方法でリテラルとしてマークする必要があります。オペレーティング・システムによっては、コマンドラインの一重引用符自体をエスケープする必要がある場合もあります。この例では、バックスラッシュがエスケープ文字として使用されます。バックスラッシュがない場合、エクスポート・ユーティリティで使用するコマンドライン解析機能で一重引用符として認識されるため、一重引用符はエクスポート・ユーティリティを起動する前に削除されてしまいます。

システムの特許文字および予約文字の詳細は、ご使用のオペレーティング・システム固有の Oracle マニュアルを参照してください。

インポート・ユーティリティの対話方式モードを使用する場合の詳細は、2-45 ページの「[対話方式の使用](#)」を参照してください。

## インポート・モード

インポート・ユーティリティには、次の4種類のモードがあります。

- 全データベース・モード  
IMP\_FULL\_DATABASE ロールを持つユーザーのみが、全データベース・エクスポート・ダンプ・ファイルをインポートできます。FULL パラメータを使用してこのモードを指定します。
- トランスポータブル表領域モード  
特権ユーザーが、一連の表領域を、ある Oracle データベースから他の Oracle データベースに移動できます。TRANSPORT\_TABLESPACE パラメータを使用してこのモードを指定します。
- ユーザー（所有者）・モード  
所有するすべてのオブジェクト（表、権限、索引、プロシージャなど）をインポートできます。特権ユーザーがユーザー・モードでインポートする場合、指定したユーザー・グループのユーザーのスキーマにあるすべてのオブジェクトをインポートできます。FROMUSER パラメータを使用してこのモードを指定します。
- 表モード  
指定した表およびパーティションをインポートできます。特権ユーザーは、インポートする表を含むスキーマを指定して、その表を修飾できます。TABLES パラメータを使用してこのモードを指定します。

---

---

**注意：** 表モードを使用して ANYDATA 型の列を含む表をインポートする場合、次のエラーが発生する場合があります。

ORA-22370: Nonexistent type メソッドの使用方法が正しくありません。

このメッセージは、ANYDATA 列が、データベースに存在しない他の型に依存することを示します。表モードを使用して ANYDATA 型を使用する表をインポートする前に、手動で、ターゲット・データベース内に依存型を作成する必要があります。

---

---

表モードおよびユーザー・モードは、すべてのユーザーが使用できます。

IMP\_FULL\_DATABASE ロールを持つユーザー（特権ユーザー）は、すべてのモードでインポートできます。

IMP\_FULL\_DATABASE ロールを持つユーザーは、これらのモードのいずれかを指定する必要があります。指定しない場合、インポートはエラーになります。IMP\_FULL\_DATABASE ロールを持たないユーザーがこれらのモードをいずれも指定しない場合は、ユーザー・レベルのインポートが実行されます。

インポートされるオブジェクトは、選択したインポート・モードおよび使用したエクスポート・モードによって決まります。

**参照：**

- これらのパラメータの各構文の詳細は、2-15 ページの「[インポート・パラメータ](#)」を参照してください。
- 各エクスポート・モードでエクスポートされているオブジェクトのリストは、1-9 ページの[表 1-1](#)を参照してください。

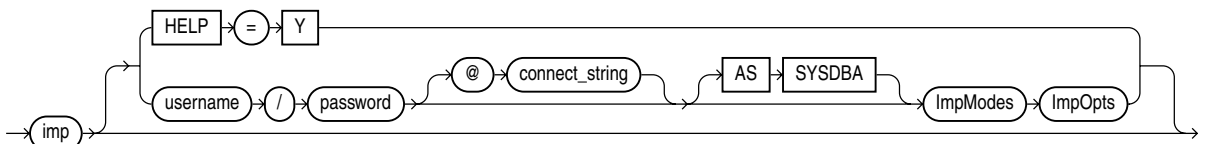
## オンライン・ヘルプの利用

インポート・ユーティリティにはオンライン・ヘルプが用意されています。ヘルプを起動するには、コマンドラインで `imp HELP=y` を入力します。

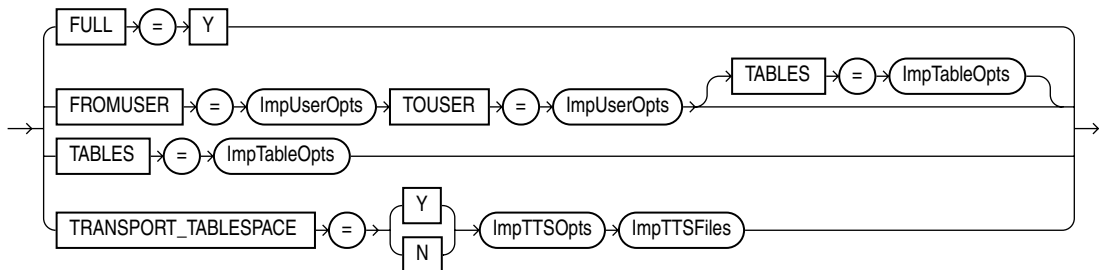
## インポート・パラメータ

パラメータ・ファイルまたはコマンドラインで指定可能なパラメータの構文を次の図に示します。図の後に、各パラメータについて説明します。

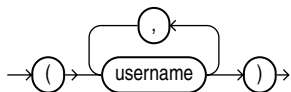
### Import\_start



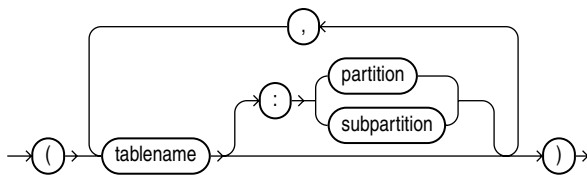
### ImpModes



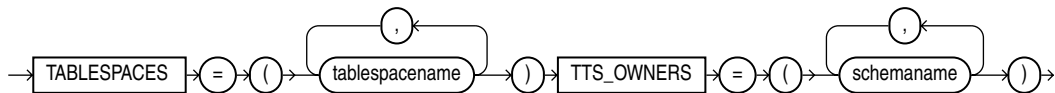
### ImpUserOpts



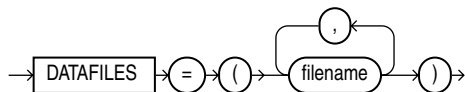
### ImpTableOpts



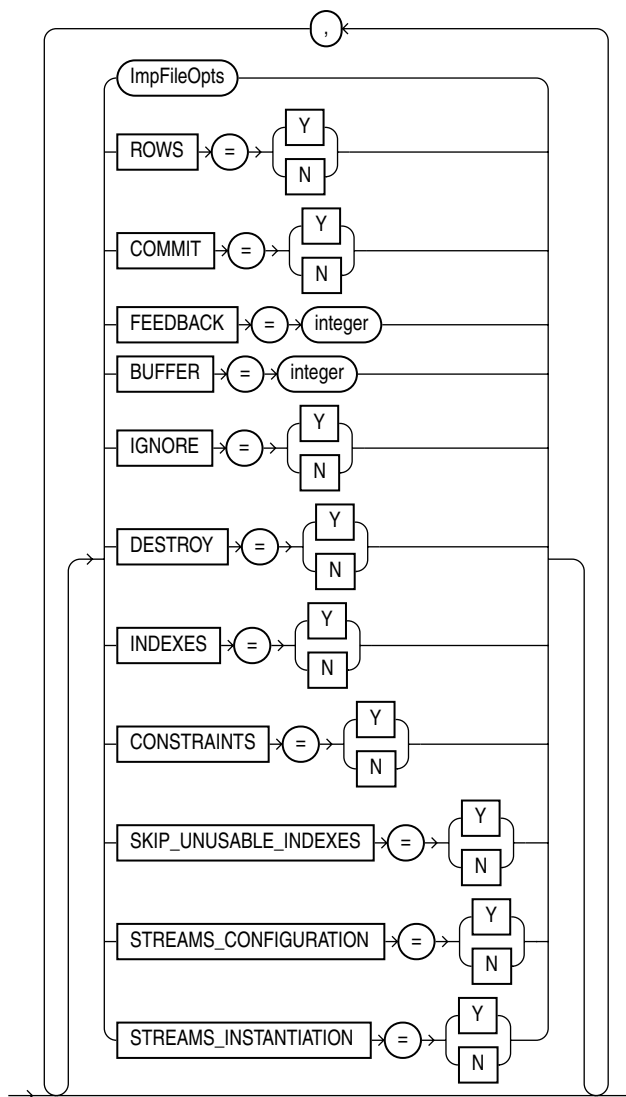
### ImpTTSOpts



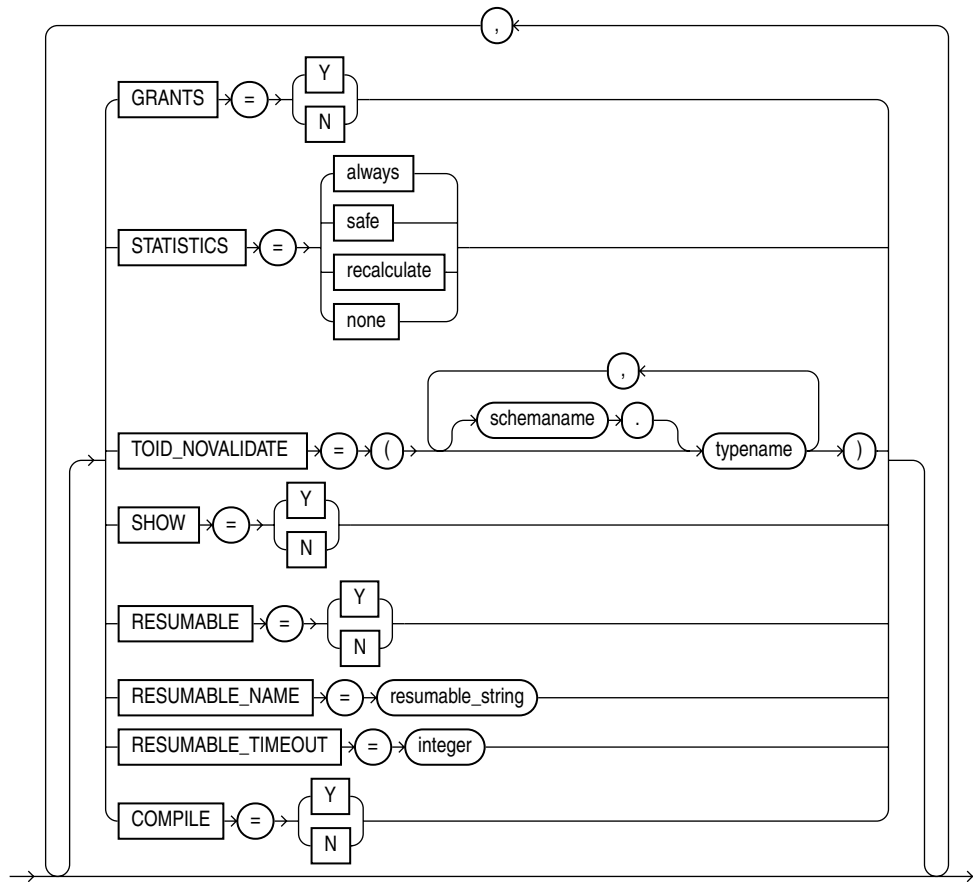
### ImpTTSTFiles



## ImpOpts

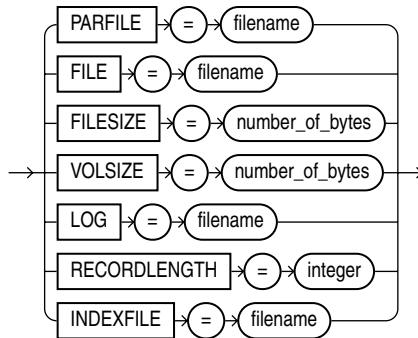


## ImpOpts\_continued





## ImpFileOpts



次に、パラメータの機能およびデフォルト値について説明します。

## BUFFER

デフォルト：オペレーティング・システムによって異なります。

BUFFER に指定された整数は、転送したデータ行を格納するバッファのバイト数です。

BUFFER によって、インポートで挿入される配列の行数が決定します。所定の行配列の挿入に必要なバッファ・サイズは、次のように計算できます。

$$\text{buffer\_size} = \text{rows\_in\_array} * \text{maximum\_row\_size}$$

LONG、LOB、BFILE、REF、ROWID、UROWID または DATE の列が含まれている表は、1 回に 1 行ずつ挿入されます。バッファ・サイズは (LOB および LONG 列の場合以外は)、行全体を格納できるだけの容量が必要です。バッファ・サイズが足りずに表の最長の行を格納できない場合、インポート・ユーティリティはさらに大きいサイズのバッファを割り当てようとします。

---

**注意：** このパラメータのデフォルト値については、ご使用のオペレーティング・システム固有の Oracle マニュアルを参照してください。

---

## CHARSET

このパラメータは、Oracle バージョン 5 およびバージョン 6 のエクスポート・ファイルにのみ適用されます。このパラメータはできるだけ使用しないでください。これは、下位バージョンとの互換性のためにのみ用意されているパラメータです。このパラメータは将来廃止される予定です。このパラメータの使用の詳細は、2-75 ページの「[CHARSET パラメータ](#)」を参照してください。

## COMMIT

デフォルト : n

配列を挿入するたびにコミットするかどうかを指定します。デフォルトでは、各表はロードされた後にのみコミットされ、エラーが発生した場合はロールバックを実行してから次のオブジェクトに進みます。

表にネストした表の列または属性が含まれている場合、ネストした表の内容は、それぞれ別の表としてインポートされます。したがって、ネストした表の内容は、常に、外部表をコミットしたトランザクションとは別のトランザクションとしてコミットされます。

パーティション表の場合に COMMIT=n を指定すると、エクスポート・ファイルのそれぞれのパーティションおよびサブパーティションは、別のトランザクションとしてインポートされます。

COMMIT=y を指定すると、ロールバック・セグメントが極端に大きくなることなく、大容量インポートのパフォーマンスが向上します。表に一意制約がある場合は、COMMIT=y と指定する必要があります。インポートが再開すると、すでにインポートされている行はすべて拒否され、リカバリ可能なエラーとして通知されます。

表に一意制約がない場合は、再度インポートを実行したときに、すでにインポート済の行もインポートされるので、行が二重にインポートされます。

LONG、LOB、BFILE、REF、ROWID、UROWID または DATE の列が含まれている表は、配列単位では挿入されません。COMMIT=y が指定されていると、各行の挿入後に表がコミットされます。

## COMPILE

デフォルト : y

パッケージ、プロシージャおよびファンクションを、作成時にインポート・ユーティリティでコンパイルするかどうかを指定します。

COMPILE=n の場合、これらのユニットは最初の使用時にコンパイルされます。たとえば、ドメイン索引作成に使用されるパッケージは、ドメイン索引作成時にコンパイルされます。

**参照：** 2-62 ページの「[ストアド・プロシージャ、ファンクションおよびパッケージのインポート](#)」を参照してください。

## CONSTRAINTS

デフォルト: y

表の制約をインポートするかどうかを指定します。デフォルトでは、制約をインポートします。制約をインポートしないようにするには、このパラメータ値を n に設定します。

IOT およびオブジェクト表の主キー制約は、常にインポートされます。

## DATAFILES

デフォルト: なし

TRANSPORT\_TABLESPACE に y を指定した場合、データベースに転送するデータ・ファイルを、このパラメータを使用して表示します。

**参照:** 2-34 ページの「[TRANSPORT\\_TABLESPACE](#)」を参照してください。

## DESTROY

デフォルト: n

データベースを構成している既存のデータ・ファイルを再利用するかどうかを指定します。DESTROY=y を指定して CREATE TABLESPACE コマンドの datafile 句に REUSE オプションを付けることによって、元のデータベースのデータ・ファイルの内容を削除した後でこれらのファイルを再利用します。

エクスポート・ファイルには、各表領域で使用されるデータ・ファイル名が格納されています。DESTROY=y を指定し、(テストや他の目的で) 同一システム上に 2 番目のデータベースを作成しようとする、表領域の作成時に、元のデータベースのデータ・ファイルが上書きされます。このような場合、デフォルトの DESTROY=n を指定すると、表領域作成時にすでにデータ・ファイルがある場合は、エラーが返されます。また、元のデータベースへインポートする必要がある場合は、IGNORE=y を指定し、既存のデータ・ファイルを置換せずに追加します。

---

---

**注意:** データ・ファイルが RAW デバイスに格納されている場合は、DESTROY=n を指定しても、ファイルは上書きされます。

---

---

## FEEDBACK

デフォルト: 0 (ゼロ)

$n$  行分のインポートを 1 つのピリオドで示すプログレス・バーの表示を指定します。たとえば、`FEEDBACK=10` を指定すると、10 行分のインポートが終了するたびにピリオドが 1 つ表示されます。`FEEDBACK` 値は、インポートされるすべての表に適用されるため、表単位では設定できません。

## FILE

デフォルト: `expdat.dmp`

インポートするエクスポート・ファイル名を指定します。デフォルトの拡張子は、`.dmp` です。エクスポート・ユーティリティは、複数ファイルのエクスポートをサポートしているため (次の `FILESIZE` パラメータの説明を参照)、複数のインポート・ファイル名が必要な場合もあります。次に例を示します。

```
imp scott/tiger IGNORE=y FILE = dat1.dmp, dat2.dmp, dat3.dmp FILESIZE=2048
```

ユーザー自身がエクスポートしたファイルでなくても指定できますが、そのファイルに対する読み権限が必要です。他のユーザーがエクスポートしたエクスポート・ファイルの場合は、`IMP_FULL_DATABASE` ロールが必要です。

## FILESIZE

デフォルト: オペレーティング・システムによって異なります。

エクスポート・ユーティリティでは複数のエクスポート・ファイルへの書き込みがサポートされており、インポート・ユーティリティでは複数のエクスポート・ファイルからの読み込みが可能です。エクスポート `FILESIZE` パラメータの値 (バイト制限) を指定すると、エクスポート・ユーティリティでは、それぞれのダンプ・ファイルに指定したバイト数のみが書き込まれます。インポート・ユーティリティでは、エクスポートの最大ダンプ・ファイル・サイズを指定するために、インポート `FILESIZE` パラメータを使用する必要があります。

---

---

**注意：** ファイルに格納可能な最大値は、オペレーティング・システムによって異なります。この最大値については、`FILESIZE` を指定する前に、ご使用のオペレーティング・システム固有の Oracle マニュアルで確認してください。

---

---

`FILESIZE` の値は、数字に `KB` (キロバイト数) を付けて指定できます。たとえば、`FILESIZE=2KB` は、`FILESIZE=2048` と同じです。同様に、`MB` はメガバイト ( $1024 \times 1024$ ) を、`GB` はギガバイト ( $1024$  の 3 乗) を表します。`B` はバイトの省略です。この場合、最終的なファイルサイズの算出に乗算は不要です (`FILESIZE=2048B` は、`FILESIZE=2048` と同じです)。

ダンプ・ファイルの最大サイズの詳細は、1-22 ページの「[FILESIZE](#)」を参照してください。

## FROMUSER

デフォルト: なし

インポートするスキーマをカンマで区切ったリスト。このパラメータは、IMP\_FULL\_DATABASE ロールを持つユーザーにのみ関係があります。このパラメータで、複数のスキーマを含むエクスポート・ファイル（たとえば、全エクスポート・ダンプ・ファイルまたは複数スキーマのユーザー、ユーザー・モードのエクスポート・ダンプ・ファイルなど）からスキーマのサブセットをインポートできます。

ファンクション索引、ファンクション、プロシージャ、トリガー、型本体、ビューなどの内部に表示されるスキーマ名は、FROMUSER または TOUSER の処理には影響されません。影響を受けるのは、オブジェクト名のみです。インポートの完了後、TOUSER スキーマに含まれる項目が古いスキーマ (FROMUSER) を参照しているかどうかを手動で確認し、必要に応じて修正する必要があります。

通常は、インポート・パラメータ TOUSER と FROMUSER を組み合わせて使用し、インポートのターゲットとなるスキーマの所有者ユーザー名のリストを指定します (2-33 ページの「[TOUSER](#)」を参照)。ただし、TOUSER を指定しない場合、次のようにインポートされます。

- エクスポート・ファイルが、全データベース・モードのダンプ・ファイル、または複数スキーマ、ユーザー・モードのエクスポート・ダンプ・ファイルの場合、FROMUSER のスキーマへオブジェクトをインポートします。
- エクスポート・ファイルが単一のスキーマで、ユーザー・モードのエクスポート・ダンプ・ファイルが権限のないユーザーに作成された場合、(インポート時に FROMUSER スキーマが存在するかどうかにかかわらず) インポートするユーザーのスキーマにオブジェクトを作成します。

---

---

**注意:** FROMUSER=SYSTEM を指定しても、システム・オブジェクトはインポートされず、ユーザー SYSTEM が所有するスキーマ・オブジェクトのみがインポートされます。

---

---

## FULL

デフォルト: n

エクスポート・ファイル全体をインポートするかどうかを指定します。

## GRANTS

デフォルト : y

オブジェクト権限をインポートするかどうかを指定します。

デフォルトでは、エクスポートされたオブジェクト権限はすべてインポートされます。ユーザー・モードでエクスポートが実行されている場合は、第 1 レベルのオブジェクト権限（所有者によって付与されているもの）のみがエクスポート・ファイルにインポートされます。

全データベース・モードでエクスポートが実行されている場合は、下位レベルのオブジェクト権限（With Grant Option で権限が付与されたユーザーによって付与されているもの）を含むすべての権限が、エクスポート・ファイルにインポートされます。GRANTS=n を指定すると、オブジェクト権限はインポートされません（GRANTS=n を指定しても、システム権限はインポートされます）。

---

---

**注意：** エクスポート・ユーティリティでは、セキュリティ上の理由から、インポートに影響するデータ・ディクショナリ・ビューの権限はエクスポートされません。このような権限がエクスポートされると、インポートしたユーザーが気付かないうちに、アクセス権が変更される場合があります。

---

---

## HELP

デフォルト : なし

インポート・パラメータの説明を表示します。ヘルプを起動するには、コマンドラインで `imp HELP=y` を入力します。

## IGNORE

デフォルト : n

オブジェクト作成エラーの処理方法を指定します。デフォルトの IGNORE=n が指定されている場合は、オブジェクト作成エラーがログに記録または表示されて、インポートが続行されます。

IGNORE=y を指定すると、データベース・オブジェクトの作成時に作成エラーが発生しても、このエラーは無視され、エラーはレポートされずに続きます。

無視されるのはオブジェクト作成エラーのみです。オペレーティング・システム、データベース、SQL などのエラーは無視されず、場合によっては処理が停止します。

IGNORE=y が指定され、1 つのエクスポート・ファイルから何回もリフレッシュが行われる場合、オブジェクトが何回も作成される場合があります（ただし、各オブジェクトには一意のシステム定義名が付けられます）。特定のオブジェクト（たとえば、制約など）に対しては、CONSTRAINTS=n を指定してインポートを実行すると、この問題を回避できます。

CONSTRAINTS=n を指定して全インポートを実行すると、表の制約はインポートされません。

表がすでに存在する場合、IGNORE=y を指定すると、行は既存の表にインポートされ、エラーやメッセージは出力されません。新しい記憶域パラメータを使用するため、またはクラスタにすでに表を作成済であるため、すでに存在する表にデータをインポートする場合があります。

表がすでに存在する場合、IGNORE=n を指定すると、エラーがレポートされ、表は、行が挿入されないままスキップされます。また、表に依存するオブジェクト（索引、権限、制約など）は作成されません。

---

---

**注意：** 既存の表へのインポート時に、表の列の索引が一意でない場合、行データが重複することがあります。

---

---

## INDEXES

デフォルト: y

索引をインポートするかどうかを指定します。LOB 索引、OID 索引、一意制約索引など、システムによって作成される索引は、このパラメータの指定に関係なく、インポート・ユーティリティによって自動的に再作成されます。

INDEXES=n を指定すると、すべてのユーザー作成索引をインポートの終了後に作成できます。

インポート時、ターゲット表にすでに索引が存在する場合は、データ挿入時にターゲット表の索引のメンテナンスを実行します。

## INDEXFILE

デフォルト: なし

索引作成文を受け取るファイルを指定します。

このパラメータを指定すると、指定したモードでの索引作成文は、データベース中に索引を作成するために使用されるのではなく、抽出されて指定のファイルに書き込まれます。データベース・オブジェクトはインポートされません。

インポート・パラメータ CONSTRAINTS に y を設定している場合、索引ファイルに表制約も書き込まれます。

その後、このファイルを編集して（記憶域パラメータの変更など）、索引を作成するための SQL スクリプトとして使用できます。

ファイル内で定義されている索引をさらに簡単に識別するために、エクスポート・ファイルの CREATE TABLE 文および CREATE CLUSTER 文がコメントとして含まれます。

この機能を使用するには、次の手順を実行します。

1. INDEXFILE パラメータを使用してインポートを行い、索引作成文のファイルを作成します。
2. ファイルを編集して、有効なパスワードを `connect` 文字列に追加します。
3. INDEXES=n を指定してインポートを再実行します。  
(この手順でデータベース・オブジェクトがインポートされますが、エクスポート・ファイルに格納されている索引定義は使用されません。)
4. 索引作成文のファイルを SQL スクリプトとして実行し、索引を作成します。

INDEXFILE パラメータを指定できるのは、FULL=y、FROMUSER、TOUSER または TABLES パラメータを指定した場合のみです。

## LOG

デフォルト: なし

情報メッセージおよびエラー・メッセージを受け取るファイルを指定します。ログ・ファイルを指定すると、端末画面およびログ・ファイルの両方にインポートに関する情報が書き込まれます。

## PARFILE

デフォルト: なし

インポート・パラメータのリストを格納するファイルのファイル名を指定します。パラメータ・ファイルの使用の詳細は、2-11 ページの「[インポート・ユーティリティの起動](#)」を参照してください。

## RECORDLENGTH

デフォルト: オペレーティング・システムによって異なります。

ファイル・レコードの長さをバイト単位で指定します。RECORDLENGTH パラメータは、異なるデフォルト値を使用する別のオペレーティング・システムにエクスポート・ファイルを転送する場合に指定する必要があります。

このパラメータを指定しない場合、ご使用のプラットフォーム固有の BUFSIZ のデフォルト値が採用されます。BUFSIZ のデフォルト値の詳細は、ご使用のオペレーティング・システム固有のドキュメントを参照してください。

RECORDLENGTH は、ご使用のシステムの BUFSIZ の値以上の任意の値に設定できます（最大値は 64KB です）。RECORDLENGTH パラメータの変更により影響を受けるのは、データベースに書き出す前に累積されるデータのサイズのみです。オペレーティング・システム・ファイルのブロック・サイズには影響しません。



このパラメータは、インポート・ユーティリティの I/O バッファのサイズ指定にも使用できます。

---

---

**注意：** 適切な値の判断や他のレコード・サイズでのファイルの作成については、ご使用のオペレーティング・システム固有の Oracle マニュアルを参照してください。

---

---

## RESUMABLE

デフォルト : n

再開可能な領域割当てを有効または無効にします。このパラメータはデフォルトでは無効なため、関連する RESUMABLE\_NAME パラメータおよび RESUMABLE\_TIMEOUT パラメータを使用するには、RESUMABLE=y に設定する必要があります。

### 参照：

- 『Oracle9i データベース概要』を参照してください。
- 再開可能な領域割当ての詳細は、『Oracle9i データベース管理者ガイド』を参照してください。

## RESUMABLE\_NAME

デフォルト : 'User USERNAME (USERID), Session SESSIONID, Instance INSTANCEID'

再開可能な文を指定します。この値はユーザー定義のテキスト文字列で、USER\_RESUMABLE または DBA\_RESUMABLE ビューに挿入して、一時停止されている特定の再開可能な文を識別できます。

RESUMABLE パラメータを y に設定して、再開可能な領域割当てを有効にしないかぎり、このパラメータは無視されます。

## RESUMABLE\_TIMEOUT

デフォルト : 7200 秒 (2 時間)

エラー修正に必要な時間を指定します。タイムアウト時間内にエラーを修正できない場合は、文の実行が強制終了されます。

RESUMABLE パラメータを y に設定して、再開可能な領域割当てを有効にしないかぎり、このパラメータは無視されます。

## ROWS

デフォルト : y

表のデータ行をインポートするかどうかを指定します。

## SHOW

デフォルト : n

SHOW=y を指定すると、エクスポート・ファイルの内容が画面に表示されますが、インポートは実行されません。エクスポート・ファイルに含まれる SQL 文は、インポート・ユーティリティでその文が実行される順序で表示されます。

SHOW パラメータを指定できるのは、FULL=y、FROMUSER、TOUSER または TABLES パラメータを指定した場合のみです。

## SKIP\_UNUSABLE\_INDEXES

デフォルト : n

(システムまたはユーザーのいずれかによって) 索引使用禁止に設定されている索引を、インポート・ユーティリティで作成するかどうかを指定します。他の索引（事前に索引使用禁止に設定されていない索引）に対しては、行の挿入時にメンテナンス処理が行われます。

このパラメータを使用すると、選択した索引パーティションに対する索引のメンテナンスを、行データの挿入が完了するまで延期できます。インポート後、影響を受けた索引パーティションの再作成が必要です。

---

---

**注意：** 一意で使用禁止状態の索引に対しては、索引メンテナンスをスキップできません。したがって、一意の索引は SKIP\_UNUSABLE\_INDEXES パラメータの影響を受けません。

---

---

INDEXES=n を指定して INDEXFILE パラメータを使用すると、索引の再作成を行う SQL スクリプトを作成できます。このパラメータを指定しないと、行挿入によって使用禁止索引をメンテナンスする場合、その行挿入によってエラーが発生します。

**参照：**『Oracle9i SQL リファレンス』の ALTER SESSION 文についての説明を参照してください。

## STATISTICS

デフォルト: ALWAYS

インポート時のデータベース・オブティマイザ統計の処理方法を指定します。

メニュー項目は次のとおりです。

- ALWAYS

データベース・オブティマイザ統計に問題があるかどうかにかかわらず、常にインポートします。

- NONE

データベース・オブティマイザ統計をインポートまたは再計算しません。

- SAFE

データベース・オブティマイザ統計に問題がない場合のみにインポートします。問題がある場合は、オブティマイザ統計を再計算します。

- RECALCULATE

データベース・オブティマイザ統計をインポートしません。かわりに、インポート時に再計算します。

**参照:**

- オブティマイザおよびオブティマイザが使用する統計の詳細は、『Oracle9i データベース概要』を参照してください。
- 1-28 ページの「[STATISTICS](#)」も参照してください。
- 2-70 ページの「[統計情報のインポート](#)」も参照してください。

## STREAMS\_CONFIGURATION

デフォルト: y

エクスポート・ダンプ・ファイル内に存在する一般的な Streams メタデータをインポートするかどうかを指定します。

**参照:** 『Oracle9i Streams』を参照してください。

## STREAMS\_INSTANTIATION

デフォルト:n

エクスポート・ダンプ・ファイル内に存在する Streams インスタンスエーション・メタデータをインポートするかどうかを指定します。ストリーム環境でインスタンスエーションの一部としてインポートする場合は、y を指定します。

**参照：**『Oracle9i Streams』を参照してください。

## TABLES

デフォルト:なし

表モード・インポートでインポートすることを指定します。インポートの対象となる表名、パーティション名およびサブパーティション名をリストとして指定します。表レベル・インポートでは、パーティション表または非パーティション表全体をインポートできます。TABLES パラメータでは、インポート対象は、表およびその関連オブジェクトに限定されます (1-9 ページの表 1-1 を参照)。TABLES パラメータでは、次の値を指定できます。

- **tablename** には、インポートされる表の名前を指定します。リストにパーティション表が含まれる場合、パーティション名を指定しないと、すべてのパーティションおよびサブパーティションがインポートされます。エクスポートされたすべての表をインポートするには、アスタリク (\*) のみを表名パラメータとして指定します。

**tablename** は、任意の数の「%」パターン一致文字を含むことができます。各「%」パターン一致文字は、エクスポート・ファイルの表名の 0 個以上の文字と一致します。リスト中の固有の表名の指定されたすべてのパターンと一致する名前を持つすべての表が、選択されてインポートされます。すべてのパターン一致文字で構成され、パーティション名を含まないリスト中の表名によって、エクスポートされたすべての表がインポートされます。

- **partition\_name** および **subpartition\_name** によって、パーティション表内で指定された 1 つ以上のパーティションまたはサブパーティションにインポートが制限されます。

構文の形式は、次のとおりです。

**tablename:partition\_name**

**tablename:subpartition\_name**

**tablename:partition\_name** を使用する場合、指定された表はパーティション化される必要があります、**partition\_name** はパーティションまたはサブパーティションのうちのいずれかの名前である必要があります。指定された表がパーティション化されていない場合、**partition\_name** は無視され、表全体がインポートされます。

一度に指定できる表の数は、コマンドラインの制限によって決まります。

エクスポート・ファイルの処理中、エクスポート・ファイルの各表名は、パラメータで指定された順に、リスト中の各表名と比較されます。あいまいな処理が行われたり、処理時間が極端に長くならないように、固有の表名がリストの始めに表示され、一般的な表名（パターンを使用したもの）がリストの終わりに表示される必要があります。

エクスポート時には表名をスキーマ名（scott.emp など）で修飾できますが、インポート時にはできません。次に、間違って指定された TABLES パラメータの例を示します。

```
imp SYSTEM/password TABLES=(jones.accts, scott.emp, scott.dept)
```

これらの表をインポートするには、次のように指定します。

```
imp SYSTEM/password FROMUSER=jones TABLES=(accts)
imp SYSTEM/password FROMUSER=scott TABLES=(emp,dept)
```

詳細は、2-44 ページの「[パターン一致を使用して様々な表をインポートする例](#)」を参照してください。

---

---

**注意：** UNIX など一部のオペレーティング・システムで、カッコなどの特殊文字を使用する場合は、特殊文字として扱われないように、その文字の前にエスケープ文字を使用する必要があります。UNIX では、次の例に示すように、エスケープ文字としてバックスラッシュ（\）を使用します。

```
TABLES=(emp,dept\)
```

---

---

## 表名の制限

表名には次の制限があります。

- デフォルトでは、表名は大文字でデータベースに格納されます。表名が大文字と小文字または小文字のみで表記され、大 / 小文字を区別する場合、名前を引用符で囲む必要があります。したがって、表名は、データベースに格納されている表名と完全に一致するように指定する必要があります。

ただし、オペレーティング・システムによっては、コマンドラインの引用符自体をエスケープする必要がある場合があります。次に、異なるインポート・モードで大 / 小文字の区別を保持する方法を示します。

- コマンドライン・モード

```
tables='"Emp\''
```

- 対話方式モード

```
Table(T) to be exported: "Exp"
```

- パラメータ・ファイル・モード

```
tables='"Emp"'
```

- 表名を引用符で囲まないかぎり、コマンドラインで指定する表名にシャープ（#）記号は使用できません。同様に、パラメータ・ファイルでは、表名が引用符で囲まれていないかぎり、表名にシャープ（#）記号を使用すると、シャープ（#）記号より右側の文字はコメントとして解釈されます。

たとえば、パラメータ・ファイルに次のコマンドラインが記述されている場合、emp# の右側はすべてコメントとして解釈されるため、表 dept および mydata はインポートされません。

```
TABLES=(emp#, dept, mydata)
```

ただし、次の例では、emp# が引用符で囲まれているため、3 つの表はすべてインポートされます。

```
TABLES=("emp#", dept, mydata)
```

---

---

**注意：** オペレーティング・システムによっては、一重引用符を使用する必要がある場合と、二重引用符を使用する必要がある場合があります。ご使用のオペレーティング・システム固有のドキュメントで確認してください。表のネーミング方法に制限があるオペレーティング・システムもあります。

たとえば、UNIX の C シェルではドル記号（\$）やシャープ（#）（またはその他の特殊文字）には特別な意味があります。これらの文字をシェルを介してインポートするには、エスケープ文字を使用する必要があります。

---

---

## TABLESPACES

デフォルト：なし

TRANSPORT\_TABLESPACE に y を指定した場合、データベースに転送する表領域を、このパラメータを使用して表示します。

詳細は、2-34 ページの「[TRANSPORT\\_TABLESPACE](#)」を参照してください。

## TOID\_NOVALIDATE

デフォルト：なし

型を参照している表のインポート時に、その名前の型がすでにデータベースに存在している場合は、その既存の型が、実際にその表で使用されているかどうか（実際は異なる型で、単に同じ名前であるだけではないか）を確認します。

この検証のために、型の一意の識別子（TOID）と、エクスポート・ファイルに格納された識別子が比較されます。TOID が一致しない場合、その表の行はインポートされません。

この妥当性チェックをしてはいけない型もあります（たとえば、その型がカートリッジのインストールによって作成された場合）。TOID\_NOVALIDATE パラメータを使用して、TOID と比較しない型を指定できます。

構文は次のようになります。

```
TOID_NOVALIDATE=([schemaname.]typename [, ...])
```

次に例を示します。

```
imp scott/tiger TABLE=foo TOID_NOVALIDATE=bar
imp scott/tiger TABLE=foo TOID_NOVALIDATE=(fred.type0,sally.type2,type3)
```

その型にスキーマ名を指定しない場合、インポートするユーザーのスキーマがデフォルトになります。たとえば、前述の最初の例では、型 bar は scott.bar がデフォルトになります。

通常のインポートでは、削除される型が含まれていると、次のように出力されます。

```
[...]
. importing IMP3's objects into IMP3
. . skipping TOID validation on type IMP2.TOIDTYP0
. . importing table                "TOIDTAB3"
[...]
```

---

---

**注意：** 型の識別子を比較しないように指定する場合は、ユーザーの責任において、インポートされる型の属性リストを既存の型の属性リストと一致させるようにしてください。これらの属性リストが一致しない場合、インポート結果は保証されません。

---

---

## TOUSER

デフォルト: なし

インポートの対象となるスキーマを所有するユーザー名のリストを指定します。このパラメータを使用するには、IMP\_FULL\_DATABASE ロールが必要です。オブジェクトがもともと入っていたスキーマと異なるスキーマにインポートする場合は、TOUSER を指定してください。次に例を示します。

```
imp SYSTEM/password FROMUSER=scott TOUSER=joe TABLES=emp
```

複数のスキーマを指定する場合、スキーマ名は対で指定します。次の例では、scott のオブジェクトを joe のスキーマに、fred のオブジェクトを ted のスキーマにインポートします。

```
imp SYSTEM/password FROMUSER=scott,fred TOUSER=joe,ted
```

FROMUSER リストが TOUSER リストより長い場合、残りのスキーマは、通常のデフォルトの規則に従って、FROMUSER スキーマにインポートされるか、またはインポートを実行するユーザーのスキーマにインポートされます。余分なオブジェクトが TOUSER スキーマにインポートされるようにするには、次の構文を使用します。

```
imp SYSTEM/password FROMUSER=scott,adams TOUSER=tet,tet
```

ユーザー tet は 2 回指定されています。

**参照：** FROMUSER および TOUSER を使用する場合の制限の詳細は、2-23 ページの「[FROMUSER](#)」を参照してください。

## TRANSPORT\_TABLESPACE

デフォルト : n

y を指定した場合、エクスポート・ファイルからトランспортаブルの表領域メタデータがインポートされます。

## TTS\_OWNERS

デフォルト : なし

TRANSPORT\_TABLESPACE に y を指定した場合、このパラメータを使用して、一連のトランспортаブル表領域のデータの所有者ユーザーを表示できます。

詳細は、2-34 ページの「[TRANSPORT\\_TABLESPACE](#)」を参照してください。

## USERID (ユーザー名 / パスワード)

デフォルト : なし

インポートを実行するユーザーの *username/password* (およびオプションの接続文字列) を指定します。

USERID は、次のように指定できます。

```
username/password AS SYSDBA
```

または

```
username/password@instance
```

または

```
username/password@instance AS SYSDBA
```

ユーザー SYS として接続する場合は、接続文字列に AS SYSDBA も指定する必要があります。オペレーティング・システムによっては、AS SYSDBA を特殊文字列とみなし、その文



文字列全体を引用符で囲む必要があります。詳細は、2-13 ページの「[SYSDBA でのインポート・ユーティリティの起動](#)」を参照してください。

**参照：**

- 『Oracle9i Heterogeneous Connectivity Administrator's Guide』を参照してください。
- Oracle Net に `@connect_string` を指定する方法の詳細は、ご使用の Oracle Net プロトコルのユーザーズ・ガイドを参照してください。

## VOLSIZE

各テーブル・ボリュームのエクスポート・ファイルに最大バイト数を指定します。

VOLSIZE パラメータの最大値は、64 ビットで格納できる最大値と同じです。詳細は、ご使用のオペレーティング・システム固有の Oracle マニュアルを参照してください。

VOLSIZE の値は、数字に KB（キロバイト数）を付けて指定できます。たとえば、VOLSIZE=2KB は、VOLSIZE=2048 と同じです。同様に、MB はメガバイト（1024 × 1024）を、GB はギガバイト（1024 の 3 乗）を表します。B はバイトの省略です。この場合、最終的なファイル・サイズの算出に乗算は不要です（VOLSIZE=2048B は、VOLSIZE=2048 と同じです）。

## インポート・セッションの例

この項では、インポート・セッションの例をいくつか取り上げ、パラメータ・ファイル方式およびコマンドライン方式の使用方法を示します。ここでは、次の 4 つのインポート・セッションの例を示します。

- [特定のユーザーの表を選択してインポートする例](#)
- [別のユーザーによってエクスポートされた表をインポートする例](#)
- [あるユーザーの表を別のユーザーへインポートする例](#)
- [パーティション・レベル・インポートでのインポート・セッションの例](#)
- [パターン一致を使用して様々な表をインポートする例](#)

## 特定のユーザーの表を選択してインポートする例

この例では、管理者が全データベース・エクスポート・ファイルを使用して、dept 表および emp 表を scott のスキーマにインポートします。

### パラメータ・ファイル方式

```
> imp SYSTEM/password PARFILE=params.dat
```

params.dat ファイルには、次の情報が格納されています。

```
FILE=dba.dmp  
SHOW=n  
IGNORE=n  
GRANTS=y  
FROMUSER=scott  
TABLES=(dept,emp)
```

### コマンドライン方式

```
> imp SYSTEM/password FILE=dba.dmp FROMUSER=scott TABLES=(dept,emp)
```

### インポート・メッセージ

```
Import: Release 9.2.0.1.0 - Production on Wed Feb 27 17:20:51 2002
```

```
(c) Copyright 2002 Oracle Corporation. All rights reserved.
```

```
Connected to: Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production  
With the Partitioning and Oracle Data Mining options  
JServer Release 9.2.0.1.0 - Production
```

```
Export file created by EXPORT:V09.02.00 via conventional path  
import done in WE8DEC character set and AL16UTF16 NCHAR character set  
. importing SCOTT's objects into SCOTT  
. . importing table "DEPT" 4 rows imported  
. . importing table "EMP" 14 rows imported  
Import terminated successfully without warnings.
```

## 別のユーザーによってエクスポートされた表をインポートする例

この例では、blake がエクスポートしたファイルから unit 表および manager 表を scott のスキーマにインポートします。

### パラメータ・ファイル方式

```
> imp SYSTEM/password PARFILE=params.dat
```

params.dat ファイルには、次の情報が格納されています。

```
FILE=blake.dmp
SHOW=n
IGNORE=n
GRANTS=y
ROWS=y
FROMUSER=blake
TOUSER=scott
TABLES=(unit,manager)
```

### コマンドライン方式

```
> imp SYSTEM/password FROMUSER=blake TOUSER=scott FILE=blake.dmp -
TABLES=(unit,manager)
```

### インポート・メッセージ

```
Import: Release 9.2.0.1.0 - Production on Wed Feb 27 17:21:40 2002
```

```
(c) Copyright 2002 Oracle Corporation. All rights reserved.
```

```
Connected to: Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production
With the Partitioning and Oracle Data Mining options
JServer Release 9.2.0.1.0 - Production
```

```
Export file created by EXPORT:V09.02.00 via conventional path
```

```
Warning: the objects were exported by BLAKE, not by you
```

```
import done in WE8DEC character set and AL16UTF16 NCHAR character set
.. importing table                "UNIT"                4 rows imported
.. importing table                "MANAGER"              4 rows imported
Import terminated successfully without warnings.
```

## あるユーザーの表を別のユーザーへインポートする例

この例では、DBA がユーザー `scott` のすべての表をユーザー `blake` のアカウントにインポートします。

### パラメータ・ファイル方式

```
> imp SYSTEM/password PARFILE=params.dat
```

params.dat ファイルには、次の情報が格納されています。

```
FILE=scott.dmp
FROMUSER=scott
TOUSER=blake
TABLES=(*)
```

### コマンドライン方式

```
> imp SYSTEM/password FILE=scott.dmp FROMUSER=scott TOUSER=blake TABLES=(*)
```

### インポート・メッセージ

```
Import: Release 9.2.0.1.0 - Production on Wed Feb 27 17:21:44 2002
```

```
(c) Copyright 2002 Oracle Corporation. All rights reserved.
```

```
Connected to: Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production
With the Partitioning and Oracle Data Mining options
JServer Release 9.2.0.1.0 - Production
```

```
Export file created by EXPORT:V09.02.00 via conventional path
```

```
Warning: the objects were exported by SCOTT, not by you
```

```
import done in WE8DEC character set and AL16UTF16 NCHAR character set
. importing SCOTT's objects into BLAKE
. . importing table          "BONUS"          0 rows imported
. . importing table          "DEPT"            4 rows imported
. . importing table          "EMP"             14 rows imported
. . importing table          "SALGRADE"         5 rows imported
Import terminated successfully without warnings.
```

## パーティション・レベル・インポートでのインポート・セッションの例

この項では、複数のパーティションがある表、パーティションとサブパーティションがある表、および異なる列で再パーティション化した表のインポートについて説明します。

### 例 1: パーティション・レベル・インポート

この例では、emp は、p1、p2 および p3 で構成されているパーティション表です。

表レベルのエクスポート・ファイルを作成するには、次のコマンドを使用します。

```
> exp scott/tiger TABLES=emp FILE=exmpexp.dat ROWS=y
```

#### インポート・メッセージ

Export: Release 9.2.0.1.0 - Production on Wed Feb 27 17:22:55 2002

(c) Copyright 2002 Oracle Corporation. All rights reserved.

Connected to: Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production  
With the Partitioning and Oracle Data Mining options  
JServer Release 9.2.0.1.0 - Production  
Export done in WE8DEC character set and AL16UTF16 NCHAR character set

About to export specified tables via Conventional Path ...

.. exporting table	EMP	
.. exporting partition	P1	7 rows exported
.. exporting partition	P2	12 rows exported
.. exporting partition	P3	3 rows exported

Export terminated successfully without warnings.

パーティション・レベルのインポートでは、インポートの対象に、エクスポートした表の特定のパーティションを指定できます。この例では、emp 表の p1 および p3 を指定します。

```
> imp scott/tiger TABLES=(emp:p1,emp:p3) FILE=exmpexp.dat ROWS=y
```

#### インポート・メッセージ

Import: Release 9.2.0.1.0 - Production on Wed Feb 27 17:22:57 2002

(c) Copyright 2002 Oracle Corporation. All rights reserved.

Connected to: Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production  
With the Partitioning and Oracle Data Mining options  
JServer Release 9.2.0.1.0 - Production

Export file created by EXPORT:V09.02.00 via conventional path

```
import done in WE8DEC character set and AL16UTF16 NCHAR character set
. importing SCOTT's objects into SCOTT
. . importing partition          "EMP": "P1"          7 rows imported
. . importing partition          "EMP": "P3"          3 rows imported
Import terminated successfully without warnings.
```

## 例 2: コンポジット・パーティション表のパーティション・レベル・インポート

この例では、コンポジット・パーティション表のパーティションおよびサブパーティションがインポートされます。emp は、2つのコンポジット・パーティション p1 および p2 のパーティション表です。P1 には、3つのサブパーティション p1\_sp1、p1\_sp2 および p1\_sp3 があります。P2 には2つのサブパーティション p2\_sp1 および p2\_sp2 があります。

表レベルのエクスポート・ファイルを作成するには、次のコマンドを使用します。

```
> exp scott/tiger TABLES=emp FILE=exmpexp.dat ROWS=y
```

### インポート・メッセージ

Export: Release 9.2.0.1.0 - Production on Wed Feb 27 17:23:06 2002

(c) Copyright 2002 Oracle Corporation. All rights reserved.

```
Connected to: Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production
With the Partitioning and Oracle Data Mining options
JServer Release 9.2.0.1.0 - Production
Export done in WE8DEC character set and AL16UTF16 NCHAR character set
```

About to export specified tables via Conventional Path ...

```
. . exporting table          EMP
. . exporting composite partition      P1
. . exporting subpartition      P1_SP1      2 rows exported
. . exporting subpartition      P1_SP2      10 rows exported
. . exporting subpartition      P1_SP3      7 rows exported
. . exporting composite partition      P2
. . exporting subpartition      P2_SP1      4 rows exported
. . exporting subpartition      P2_SP2      2 rows exported
Export terminated successfully without warnings.
```

次のインポート・コマンドでは、emp 表にあるコンポジット・パーティション p1 のサブパーティション p1\_sp2 および p1\_sp3 と、emp 表にあるコンポジット・パーティション p2 のすべてのサブパーティションがインポートされます。

```
> imp scott/tiger TABLES=(emp:p1_sp2,emp:p1_sp3,emp:p2) FILE=exmpexp.dat ROWS=y
```

**インポート・メッセージ**

Import: Release 9.2.0.1.0 - Production on Wed Feb 27 17:23:07 2002

(c) Copyright 2002 Oracle Corporation. All rights reserved.

Connected to: Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production  
With the Partitioning and Oracle Data Mining options  
JServer Release 9.2.0.1.0 - Production

Export file created by EXPORT:V09.02.00 via conventional path  
import done in WE8DEC character set and AL16UTF16 NCHAR character set  
. importing SCOTT's objects into SCOTT  
.. importing subpartition "EMP": "P1\_SP2" 10 rows imported  
.. importing subpartition "EMP": "P1\_SP3" 7 rows imported  
.. importing subpartition "EMP": "P2\_SP1" 4 rows imported  
.. importing subpartition "EMP": "P2\_SP2" 2 rows imported  
Import terminated successfully without warnings.

**例 3: 別の列での表の再パーティション化**

この例では、emp 表に、empno 列に基づく 2 つのパーティションがあります。emp 表を deptno 列で再パーティション化します。

別の列で表を再パーティション化するには、次の手順を実行してください。

1. エクスポートを実行して、データを保存します。
2. データベースから表を削除します。
3. 表を新しいパーティションに分割して再作成します。
4. 表データをインポートします。

次に、これらの手順の例を示します。

```
> exp scott/tiger table=emp file=empexp.dat
```

Export: Release 9.2.0.1.0 - Production on Wed Feb 27 17:22:19 2002

(c) Copyright 2002 Oracle Corporation. All rights reserved.

Connected to: Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production  
With the Partitioning and Oracle Data Mining options  
JServer Release 9.2.0.1.0 - Production  
Export done in WE8DEC character set and AL16UTF16 NCHAR character set

```
About to export specified tables via Conventional Path ...
. . exporting table                      EMP
. . exporting partition                  EMP_LOW      4 rows exported
. . exporting partition                  EMP_HIGH     10 rows exported
Export terminated successfully without warnings.
```

```
SQL> connect scott/tiger
Connected.
SQL> drop table emp cascade constraints;
Statement processed.
SQL> create table emp
2>  (
3>  empno    number(4) not null,
4>  ename     varchar2(10),
5>  job       varchar2(9),
6>  mgr       number(4),
7>  hiredate  date,
8>  sal       number(7,2),
9>  comm      number(7,2),
10>  deptno    number(2)
11>  )
12> partition by range (deptno)
13>  (
14>  partition dept_low values less than (15)
15>    tablespace tbs_1,
16>  partition dept_mid values less than (25)
17>    tablespace tbs_2,
18>  partition dept_high values less than (35)
19>    tablespace tbs_3
20>  );
Statement processed.
SQL> exit
```

```
> imp scott/tiger tables=emp file=empexp.dat ignore=y
```

```
Import: Release 9.2.0.1.0 - Production on Wed Feb 27 17:22:25 2002
```

```
(c) Copyright 2002 Oracle Corporation. All rights reserved.
```

```
Connected to: Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production
With the Partitioning and Oracle Data Mining options
JServer Release 9.2.0.1.0 - Production
```

```
Export file created by EXPORT:V09.02.00 via conventional path
import done in WE8DEC character set and AL16UTF16 NCHAR character set
. importing SCOTT's objects into SCOTT
```



```
.. importing partition          "EMP": "EMP_LOW"          4 rows imported
.. importing partition          "EMP": "EMP_HIGH"         10 rows imported
Import terminated successfully without warnings.
```

次の SELECT 文では、データは deptno 列でパーティション化されます。

```
SQL> connect scott/tiger
```

```
Connected.
```

```
SQL> select empno, deptno from emp partition (dept_low);
```

EMPNO	DEPTNO
7782	10
7839	10
7934	10

```
3 rows selected.
```

```
SQL> select empno, deptno from emp partition (dept_mid);
```

EMPNO	DEPTNO
7369	20
7566	20
7788	20
7876	20
7902	20

```
5 rows selected.
```

```
SQL> select empno, deptno from emp partition (dept_high);
```

EMPNO	DEPTNO
7499	30
7521	30
7654	30
7698	30
7844	30
7900	30

```
6 rows selected.
```

```
SQL> exit;
```

## パターン一致を使用して様々な表をインポートする例

この例では、パターン一致を使用して、ユーザー scott の様々な表をインポートします。

### パラメータ・ファイル方式

```
imp SYSTEM/password PARFILE=params.dat
```

params.dat ファイルには、次の情報が格納されています。

```
FILE=scott.dmp
IGNORE=n
GRANTS=y
ROWS=y
FROMUSER=scott
TABLES=(%d%,b%s)
```

### コマンドライン方式

```
imp SYSTEM/password FROMUSER=scott FILE=scott.dmp TABLES=(%d%,b%s)
```

### インポート・メッセージ

```
Import: Release 9.2.0.1.0 - Production on Wed Feb 27 17:22:25 2002
```

```
(c) Copyright 2002 Oracle Corporation. All rights reserved.
```

```
Connected to: Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production
With the Partitioning and Oracle Data Mining options
JServer Release 9.2.0.1.0 - Production
```

```
Export file created by EXPORT:V09.02.00 via conventional path
```

```
import done in US7ASCII character set and AL16UTF16 NCHAR character set
import server uses JA16SJIS character set (possible charset conversion)
. importing SCOTT's objects into SCOTT
. . importing table          "BONUS"          0 rows imported
. . importing table          "DEPT"           4 rows imported
. . importing table          "SALGRADE"        5 rows imported
Import terminated successfully without warnings.
```

## 対話方式の使用

インポート・ユーティリティは、パラメータを指定しないでコマンドラインから起動すると、対話方式で起動されます。対話方式では、インポート・ユーティリティのすべての機能に関してプロンプトが表示されるわけではありません。対話方式は下位互換用にのみ提供されています。

コマンドラインで `username/password` を指定しないと、入力するように求められます。

対話方式でインポート・ユーティリティを起動する場合、インポート・ユーティリティからの応答は、コマンドラインでの入力内容によって異なります。表 2-3 に、入力内容とそれに対する応答を示します。

表 2-3 対話方式を使用したインポート・ユーティリティの起動

入力内容	インポート・ユーティリティからの応答
<code>imp username/password@instance as sysdba</code>	インポート・セッションの起動
<code>imp username/password@instance</code>	インポート・セッションの起動
<code>imp username/password as sysdba</code>	インポート・セッションの起動
<code>imp username/password</code>	インポート・セッションの起動
<code>imp username@instance as sysdba</code>	パスワード入力を求めるプロンプトの表示
<code>imp username@instance</code>	パスワード入力を求めるプロンプトの表示
<code>imp username</code>	パスワード入力を求めるプロンプトの表示
<code>imp username as sysdba</code>	パスワード入力を求めるプロンプトの表示

インポート・ユーティリティの対話方式モードでは、`SYSDBA` で接続するか、`@instance` で接続するかを指定するプロンプトは表示されません。`AS SYSDBA` や `@instance` は、ユーザー名とともに指定する必要があります。

また、パスワードを指定しなかったためにプロンプトが表示されると、`@instance` 文字列を指定できなくなります。`@instance` は `username` を指定する場合のみ使用できます。

`AS SYSDBA` を使用してインポート・ユーティリティを起動する前に、正しいコマンドラインの構文について、2-13 ページの「[SYSDBA でのインポート・ユーティリティの起動](#)」を参照してください。

インポート・ユーティリティが起動された後、次のプロンプトが表示されます。他のプロンプトに対して入力した応答によって決まるプロンプトもあるため、インポート・セッション

ですべてのプロンプトが表示されるとはかぎりません。デフォルト値が表示されるプロンプトもあります。このデフォルト値が使用可能な場合は、[Enter]を押します。

Import: Release 9.2.0.1.0 - Production on Wed Feb 27 17:22:37 2002

(c) Copyright 2002 Oracle Corporation. All rights reserved.

Connected to: Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production  
With the Partitioning and Oracle Data Mining options  
JServer Release 9.2.0.1.0 - Production

Import file: expdat.dmp >  
Enter insert buffer size (minimum is 8192) 30720>  
Export file created by EXPORT:V09.02.00 via conventional path

Warning: the objects were exported by BLAKE, not by you

import done in WE8DEC character set and AL16UTF16 NCHAR character set  
List contents of import file only (yes/no): no >  
Ignore create error due to object existence (yes/no): no >  
Import grants (yes/no): yes >  
Import table data (yes/no): yes >  
Import entire export file (yes/no): no > y  
. importing BLAKE's objects into SYSTEM  
. . importing table "DEPT" 4 rows imported  
. . importing table "MANAGER" 3 rows imported  
Import terminated successfully without warnings.

「Import entire export file(yes/no):」プロンプトでNoを指定すると、インポート先のスキーマ名とそのスキーマ内のインポート対象の表名を次のように入力するように要求されます。

Enter table(T) or partition(T:P) names. Null list means all tables for user

NULL 値を入力すると、スキーマ中のすべての表がインポートの対象になります。対話方式を使用する場合は、1 度に 1 つのスキーマのみ指定できます。

## 警告、エラーおよび完了メッセージ

この項では、インポート・ユーティリティによって発行される異なるタイプのメッセージについて説明します。また、そのメッセージのログ・ファイルへの保存方法についても説明します。

### ログ・ファイル

すべてのインポート・メッセージは、ログ・ファイルに保存できます。この場合の保存方法は2つあります。1つ目は、LOG パラメータを使用する方法です。2つ目は、システムでサポートされている場合にかぎりますが、インポート・ユーティリティの出力をファイルにリダイレクトする方法です。リダイレクト先のファイルには、正常にロードされた場合はその内容が、またエラーが発生した場合はそのエラーに関する詳細情報が記録されます。

#### 参照：

- 2-26 ページの「LOG」を参照してください。
- 出力のリダイレクトの詳細は、ご使用のオペレーティング・システム固有の Oracle マニュアルを参照してください。

### 警告メッセージ

リカバリ可能なエラーの場合、インポート処理は続行されます。たとえば、表のインポート中にエラーが発生した場合は、エラー・メッセージが表示され（またはログが記録され）、次の表にスキップして処理が続けられます。リカバリ可能なエラーは、警告と呼ばれます。

インポートでは、無効なオブジェクトに対しても警告が発行されます。

たとえば、表モード・インポートで、存在しない表を指定した場合、インポート・ユーティリティでは他のすべての表がインポートされます。その後、警告が発行されて処理が正常に終了されます。

### リカバリ不能エラー・メッセージ

エラーの中にはリカバリ不能なものもあり、このようなエラーが発生するとインポート・セッションは終了します。これらのエラーは、内部的な問題が原因であるか、またはメモリーなどのリソースが使用できないか、リソースを使い果したことが原因で発生します。

## 完了メッセージ

問題なくインポートが完了した場合は、次のメッセージが表示されます。

インポートは警告なしで正常終了しました。

リカバリ可能なエラーが 1 つ以上発生しても、インポートがそのまま続行され、完了した場合は次のメッセージが表示されます。

インポートは正常に終了しましたが、警告が発生しました。

リカバリ不能なエラーが発生した場合、インポートは即時終了し、次のメッセージが表示されます。

エラーが発生したためインポートを終了します。

**参照：**『Oracle9i データベース・エラー・メッセージ』およびご使用のシステム固有の Oracle マニュアルを参照してください。

## 終了コードによる結果の検査と表示

インポート・ユーティリティでは、インポートの完了後、すぐに実行結果を確認できます。プラットフォームによっては、インポートの実行結果はログ・ファイルに記録されるのみでなく、プロセス終了コードにも通知されます。これによって、コマンドラインやスクリプトからの出力を確認できます。[表 2-4](#) に、それぞれの結果に対応する終了コードを示します。

表 2-4 インポート時の終了コード

結果	終了コード
インポートは警告なしで正常終了しました。	EX_SUCC
インポートは正常に終了しましたが、警告が発生しました。	EX_OKWARN
エラーが発生したためインポートを終了します。	EX_FAIL

UNIX の場合、終了コードは次のようになります。

```
EX_SUCC    0
EX_OKWARN  0
EX_FAIL    1
```

## インポート中のエラー処理

この項では、データベース・オブジェクトのインポート時に発生する可能性のあるエラーについて説明します。

### 行エラー

整合性制約違反またはデータが無効なために行のインポートが拒否されると、警告メッセージが表示されますが、その表の残りの行は引き続き処理されます。「`tablespace full`」というエラーなど、後続のすべての行に影響するエラーもあります。このようなエラーの場合には、現行の表の処理は停止され、次の表にスキップします。

`RESUMABLE=y` パラメータが指定されている場合、「`tablespace full`」エラーによって、インポートが一時停止することもあります。

### 整合性制約違反

次の整合性制約に違反している行があると行エラーが発生します。

- NOT NULL 制約
- 一意制約
- 主キー（NOT NULL および一意）制約
- 参照整合性制約
- CHECK 制約

**参照：** 次のマニュアルを参照してください。

- 『Oracle9i アプリケーション開発者ガイド - 基礎編』
- 『Oracle9i データベース概要』

### 無効なデータ

データベース内の表の列定義が、エクスポート・ファイル内の列定義と異なるときにも行エラーが発生します。無効データ・エラーは、新しい表の列より長いデータの挿入、無効なデータ型またはその他の `INSERT` エラーによって発生します。

## データベース・オブジェクトのインポートでのエラー

データベース・オブジェクトをインポートするときにエラーが発生する理由にはいろいろありますが、この項ではその理由について説明します。これらのエラーが発生すると、現行のデータベース・オブジェクトのインポートは中断されます。その後、インポート・ユーティリティでは、エクスポート・ファイルの次のデータベース・オブジェクトが継続して処理されます。

### 既存オブジェクト

インポートするオブジェクトがデータベース中にすでに存在していると、オブジェクト作成エラーが発生します。これ以降の処理は、IGNORE パラメータに指定されている値によって異なります。

IGNORE=n（デフォルト）が指定されている場合、エラーが報告され、次のデータベース・オブジェクトが継続して処理されます。現行のデータベース・オブジェクトは置き換えられません。オブジェクトが表の場合、エクスポート・ファイル内の行はインポートされません。

IGNORE=y が指定されている場合、オブジェクト作成エラーは報告されません。データベース・オブジェクトは置き換えられません。オブジェクトが表の場合、行がインポートされます。無視できるエラーはオブジェクト作成エラーのみです。他のすべてのエラー（オペレーティング・システムのエラー、データベースのエラー、SQL のエラーなど）は報告されます。また、処理が停止することもあります。

---

---

**注意：** IGNORE=y を指定した場合、表の 1 つ以上の列に対して一意制約を指定しないかぎり、その表に対して重複した行が挿入されます。たとえば、誤って 2 回インポートを実行した場合などがこれに該当します。

---

---

### 順序

インポート処理で、順序番号をエクスポート・ファイルの値に設定しなおす必要がある場合は、順序を削除してください。インポートでは、既存の順序の削除と再作成は行われません。そのため、順序は、インポートの前に削除されない場合、エクスポート・ファイルに保存されている値には設定されません。順序がすでに存在している場合、エクスポート・ファイルの CREATE SEQUENCE 文は失敗し、その順序はインポートされません。



## リソース・エラー

リソースの制限によって、オブジェクトがインポートされないことがあります。たとえば、表のインポート中に、内部的な問題またはメモリーなどのリソース不足によって、リソース・エラーが発生する場合があります。

行のインポート中にリソース・エラーが発生すると、現行の表の処理が中止され、次の表にスキップします。COMMIT=y を指定している場合、現行の表のインポート済の部分がコミットされます。指定していない場合は、現行の表の処理がロールバックされた後で、インポートが続行されます。2-20 ページの「COMMIT」を参照してください。

## ドメイン索引メタデータ

ドメイン索引は、無名 PL/SQL ブロックでインポートされる、アプリケーション固有のメタデータと関連付けることができます。これらの PL/SQL ブロックは、インポート時に CREATE INDEX 文より優先して実行されます。PL/SQL ブロックにエラーが発生した場合、メタデータが索引の一部分とみなされるため、関連付けられた索引は作成されません。

# 表レベル・インポートおよびパーティション・レベル・インポート

表、パーティションおよびサブパーティションのインポートは、次のように実行できます。

- 表レベル・インポート: エクスポート・ファイルのすべてのデータをインポートします。
- パーティション・レベル・インポート: 指定されたソース・パーティションまたはサブパーティションのデータのみをインポートします。

既存の表にデータをロードするときは、パラメータ IGNORE=y を指定します。詳細は、2-24 ページの「IGNORE」を参照してください。

## 表レベル・インポートの使用に関するガイドライン

表レベル・インポートでは、指定した各表に関して表のすべての行がインポートされます。表レベル・インポートの特長は次のとおりです。

- (TRANSPORT\_TABLESPACES を除く) どのエクスポート・モードでエクスポートされた場合でも、表レベル・インポートを使用してエクスポートされたすべての表をインポートできます。
- ユーザーは、表レベルでエクスポートされたすべての表（パーティション表または非パーティション表）、パーティションまたはサブパーティションを、同じ名前のターゲット表（パーティション表または非パーティション表）にインポートできます。

表が存在しない場合で、なおかつエクスポートされた表がパーティション表である場合は、表レベル・インポートによってパーティション表が作成されます。表が正常に作成されると、エクスポート・ファイルからすべてのソース・データが読み込まれ、ターゲット表に書

き込まれます。インポート後、ターゲット表には、エクスポート・ファイル内のソース表に対応付けられたすべてのパーティションおよびサブパーティションに関するパーティション定義が格納されます。この処理によって、ソース・パーティションの物理属性および論理属性（パーティションの境界を含む）が、インポート時に維持されます。

## パーティション・レベル・インポートの使用に関するガイドライン

パーティション・レベル・インポートを指定できるのは、表モードでインポートを実行する場合のみです。パーティション・レベル・インポートでは、エクスポート・ファイル内の特定のパーティションまたはサブパーティションを選択してデータをロードすることができます。パーティション・レベル・インポートを使用する場合は、次のガイドラインに注意してください。

- インポートでは、常にターゲット表のパーティション化スキーマに従って行が格納されます。
- パーティション・レベル・インポートでは、指定されたソース・パーティションまたはサブパーティションの行データのみ挿入されます。
- ターゲット表がパーティション表の場合、パーティション・レベル・インポートを行うと、そのターゲット表の最上位パーティションより上に入る行はすべて拒否されます。
- エクスポートされた表が非パーティション表の場合は、パーティション・レベル・インポートを実行できません。ただし、表レベル・インポートを使用すると、エクスポートされた非パーティション表からパーティション表をインポートできます。
- ソース表（エクスポート時に *tablename* に指定された表）がパーティション表で、エクスポート・ファイルに存在する場合のみ、パーティション・レベル・インポートは正常に実行されます。
- エクスポート・ファイルに存在しないパーティション名またはサブパーティション名を指定すると、警告が発行されます。
- パラメータの中で指定するパーティション名またはサブパーティション名には、エクスポート・ファイルにあるパーティションまたはサブパーティションのみを指定します。エクスポート・ファイルには、エクスポート・ソース・システム上の表のすべてのデータが含まれているとはかぎりません。
- ROWS=y（デフォルト）が指定されていて、表がインポート先のシステムに存在しない場合、表が作成され、すべての行が、ソース・パーティションまたはサブパーティションから、インポート先の表のパーティションまたはサブパーティションに挿入されます。
- ROWS=y（デフォルト）および IGNORE=y が指定されていて、対象となる表の表名がインポート前に存在している場合は、指定された表のパーティションまたはサブパーティションの行がすべて、同名の表に挿入されます。インポートでは、常に、ターゲット表の既存のパーティション化スキーマに従って行が格納されます。

- ROWS=n が指定されている場合、データはターゲット表に挿入されず、エクスポート・ファイル中の表およびパーティションまたはサブパーティションに関連する他のオブジェクトに対する処理が、継続して行われます。
- ターゲット表が非パーティション表の場合、パーティションおよびサブパーティションは表全体にインポートされます。1 つ以上のパーティションまたはサブパーティションを、エクスポート・ファイルからインポート・ターゲット・システム上の非パーティション表にインポートするには、IGNORE=y を指定します。

## パーティションと表の間のデータ移行

コンポジット・パーティションのパーティション名を指定しない場合、コンポジット・パーティション内のすべてのサブパーティションが、ソースとして使用されます。

次の例では、パーティション名によって指定されたパーティションは、コンポジット・パーティションです。すべてのサブパーティションがインポートされます。

```
imp SYSTEM/password FILE=expdat.dmp FROMUSER=scott TABLES=b:py
```

次の例では、表 scott.e のパーティション qc および qd の行データが、表 scott.e にインポートされます。

```
imp scott/tiger FILE=expdat.dmp TABLES=(e:qc, e:qd) IGNORE=y
```

インポート・ターゲット・データベースに表 e が存在しない場合は、表 e の作成後、同じパーティションにデータが挿入されます。インポート前にターゲット・システムに表 e が存在する場合、行データは、挿入可能な範囲を持つパーティションに挿入されます。行データは、最終的に qc および qd 以外の名前のパーティションに挿入することもできます。

---

---

**注意：** 既存の表にパーティション・レベル・インポートを実行する場合は、ターゲット・パーティションまたはサブパーティションを正しく設定し、IGNORE=y を指定してください。

---

---

## 索引作成およびメンテナンスの制御

この項では、索引作成およびメンテナンスに関連するインポートの動作について説明します。

### 索引作成の延期

インポート・ユーティリティには、索引の作成およびメンテナンスの実行を、インポートが完了し、エクスポート・データの挿入が終了するまで延期させる機能が用意されています。インポート完了後に索引の作成、再作成またはメンテナンスを実行すると、通常、その処理時間は、インポートで各行が挿入される度にメンテナンスを実行するより短くなります。

索引作成には時間がかかるため、他のすべてのオブジェクトのインポートが完了してから行った方が効率的です。インポートでは、インポートが完了するまで、索引作成を延期できます。延期するには、INDEXES=n（デフォルトは INDEXES=y）を指定します。その後、INDEXFILE パラメータを使用してインポートを実行し、SQL スクリプト内の未作成の索引定義を格納できます。索引作成文は、このように指定しない場合、インポート・ユーティリティから発行されますが、このように指定した場合、指定されたファイルに書き込まれます。

インポート完了後、索引を作成する必要があります。索引を作成するには、通常、CONNECT 文にパスワードを指定した後、INDEXFILE で指定したファイルの内容を SQL スクリプトとして使用します。

### 索引作成およびメンテナンスの制御

SKIP\_UNUSABLE\_INDEXES=y を指定すると、インポート前に索引使用禁止に設定されていた索引のメンテナンスはすべて延期されます。他の索引（事前に索引使用禁止に設定されていない索引）に対しては、行の挿入時にメンテナンス処理が行われます。これにより、既存の表をインポートする間、索引の更新が保存されます。

索引のメンテナンスが延期されると、その索引で設定されている既存の一意整合性制約に対して違反が発生することがあります。表に一意整合性制約が存在しても、INDEXES=n を指定してインポートした表内のキーの重複は回避できません。このため、その索引は、重複キーが削除されて索引が再構築されるまでは、UNUSABLE 状態となります。

#### 索引更新延期の例

パーティション p1 および p2 を持つパーティション表 t が、インポート・ターゲット・システムに存在するとします。また、パーティション p1 にローカル索引 p1\_ind、パーティション p2 にローカル索引 p2\_ind が存在するとします。このパーティション p1 には既存の表 t のデータが入っており、そのデータ量は、エクスポート・ファイル（expdat.dmp）を使用して挿入されるデータの量よりはるかに多いとします。一方、パーティション p2 はその逆であるとしてします。

表データ挿入時に p1\_ind の索引メンテナンスを実行すると、パーティション索引の再作成時に実行するより、処理効率が高くなります。p2\_ind については、この逆になります。

また、p2\_ind については、インポート中のローカル索引のメンテナンスを延期できます。延期するには、次の手順を実行します。

1. インポート前に、次の SQL 文を発行します。

```
ALTER TABLE t MODIFY PARTITION p2 UNUSABLE LOCAL INDEXES;
```

2. 次のインポート・コマンドを発行します。

```
imp scott/tiger FILE=expdat.dmp TABLES = (t:p1, t:p2) IGNORE=y SKIP_UNUSABLE_INDEXES=y
```

この例では、インポートの実行前に ALTER SESSION SET SKIP\_UNUSABLE\_INDEXES=y 文を実行します。

3. インポート後に次の SQL 文を発行します。

```
ALTER TABLE t MODIFY PARTITION p2 REBUILD UNUSABLE LOCAL INDEXES;
```

この例では、p1 のローカル索引 p1\_ind は、インポート中、表データがパーティション p1 に挿入されるときにメンテナンスされます。一方、p2 のローカル索引 p2\_ind は、インポート後の索引再作成時にメンテナンスされます。

## データベースの断片化を解消する方法

断片化とは、多数の小さな空き領域が散在しているデータベースの状態のことです。断片化しているデータベースを再編成すると、空き領域をより大きな連続したブロックとして使用できるようになります。次のように全データベース・エクスポートおよびインポートを実行することで、データベースの断片化を解消できます。

1. データベース全体のバックアップを取るために、全データベース・エクスポート (FULL=y) を実行します。
2. すべてのユーザーがログオフしてから、Oracle データベース・サーバーを停止します。
3. データベースを削除します。データベース削除の詳細は、ご使用のオペレーティング・システム固有の Oracle マニュアルを参照してください。
4. CREATE DATABASE 文を使用して、データベースを再作成します。
5. データベース全体をリストアするために、全データベース・インポート (FULL=y) を実行します。

**参照：** データベース作成の詳細は、『Oracle9i データベース管理者ガイド』を参照してください。

## ネットワークに関する考慮点

この項では、ネットワークを介したエクスポートおよびインポートの実行時に考慮すべき点について説明します。

### ネットワークを介してエクスポート・ファイルを転送する方法

エクスポート・ファイルはバイナリ形式であるため、ネットワークを介してそのエクスポート・ファイルを転送するときは、バイナリ転送をサポートしているプロトコルを使用して、ファイルが破損しないようにしてください。たとえば、FTP などのファイル転送プロトコルを使用して、バイナリ・モードでファイルを転送します。エクスポート・ファイルをキャラクター・モードで送信すると、ファイルのインポート時にエラーが発生します。

### Oracle Net を使用したエクスポートおよびインポート

Oracle Net を使用すると、ネットワークを介してエクスポートおよびインポートを実行できます。たとえば、エクスポート・ユーティリティをローカルで実行して、リモート Oracle データベースのデータをローカル・エクスポート・ファイルに書き込みできます。また、インポート・ユーティリティをローカルで実行して、リモート Oracle データベースのデータの読み込みができます。

Oracle Net でインポート・ユーティリティを使用するには、`exp` コマンドまたは `imp` コマンドに `username/password` を入力するときに接続修飾文字列 `@connect_string` を指定する必要があります。この句の構文の詳細は、ご使用の Oracle Net プロトコルのユーザーズ・ガイドを参照してください。

**参照：** 次のマニュアルを参照してください。

- 『Oracle9i Net Services 管理者ガイド』
- 『Oracle9i Heterogeneous Connectivity Administrator's Guide』

# キャラクタ・セットおよびグローバリゼーション・サポートに関する考慮点

この項では、エクスポートおよびインポート操作時に発生する、キャラクタ・セットの変換について説明します。

## キャラクタ・セット変換

次の各項で、ユーザー・データおよび DDL に適用される文字変換について説明します。

### ユーザー・データ

データ型が CHAR、VARCHAR2、NCHAR、NVARCHAR2、CLOB および NCLOB のデータは、ソース・データベースのキャラクタ・セットで、エクスポート・ファイルに直接書き込まれます。ソース・データベースのキャラクタ・セットが、インポート・データベースのキャラクタ・セットと異なる場合、キャラクタ・セット変換が実行されます。

### DDL

エクスポートおよびインポート操作時に、DDL に対して最高 3 回のキャラクタ・セット変換が必要です。

1. エクスポート・ファイルは、環境変数 `NLS_LANG` でユーザー・セッション用に指定されたキャラクタ・セットで書き出されます。`NLS_LANG` の値が、データベースのキャラクタ・セットと異なる場合は、キャラクタ・セット変換が実行されます。
2. エクスポート・ファイルのキャラクタ・セットが、インポート先ユーザー・セッション用のキャラクタ・セットと異なる場合、ユーザー・セッションのキャラクタ・セットに変換されます。シングルバイト・キャラクタ・セットの場合のみにこの変換が実行されます。マルチバイト・キャラクタ・セットの場合は、インポート・ファイルのキャラクタ・セットがエクスポート・ファイルのキャラクタ・セットと同じである必要があります。
3. ターゲット・データベースのキャラクタ・セットが、インポート先ユーザー・セッション用のキャラクタ・セットと異なる場合、3 回目のキャラクタ・セット変換が実行される場合があります。

キャラクタ・セット変換によるデータの損失を最小限にするには、エクスポート・データベース、エクスポート・ユーザー・セッション、インポート・ユーザー・セッションおよびインポート・データベースのすべてにおいて、同一のキャラクタ・セットを使用するようにしてください。

## インポートおよびシングルバイト・キャラクタ・セット

8 ビット・キャラクタ・セットのエクスポート・ファイルをインポートすると、8 ビット文字の一部が消去されることがあります（同等の7バイトに変換されます）。これが発生するのは、インポートを実行するシステムに、システム固有の7ビット・キャラクタ・セットが存在するか、オペレーティング・システム環境変数 `NLS_LANG` が7ビット・キャラクタ・セットに設定されている場合です。アクセント記号が付いている文字からアクセントが消去されるのが最もよく見られる例です。

このような状況を回避するために、オペレーティング・システム環境変数 `NLS_LANG` にエクスポート・ファイルのキャラクタ・セットを設定できます。

Oracle バージョン5 とバージョン6 のエクスポート・ファイルを、オペレーティング・システム固有のキャラクタ・セットとは異なるキャラクタ・セットまたは `NLS_LANG` 設定でインポートする場合は、`CHARSET` インポート・パラメータにエクスポート・ファイルのキャラクタ・セットを指定してください。

## インポートおよびマルチバイト・キャラクタ・セット

ターゲット・キャラクタ・セットに同等の文字がないエクスポート・ファイル中の文字は、変換時にデフォルトの文字に置換されます（デフォルトの文字は、ターゲット・キャラクタ・セットによって定義されます）。100% 完全に変換されるためには、ターゲット・キャラクタ・セットはソース・キャラクタ・セットのスーパーセットであるか、ソース・キャラクタ・セットと同等である必要があります。

**参照：**『Oracle9i Database グローバリゼーション・サポート・ガイド』を参照してください。

## データベース・オブジェクトのインポートに関する考慮点

次の項では、特定のデータベース・オブジェクトをインポートする場合に考慮すべき点について説明します。

### オブジェクト識別子のインポート

Oracle データベース・サーバーでは、オブジェクト型、オブジェクト表およびオブジェクト表内の行を一意に識別できるように、オブジェクト識別子が割り当てられます。オブジェクト識別子はインポート・ユーティリティによって保持されます。

型を参照している表のインポート時に、その名前の型がすでにデータベースに存在している場合は、その既存の型が、実際にその表で使用されているかどうか（実際は異なる型で、単に同じ名前であるだけではないか）を確認します。

この確認のために、型の一意の識別子（TOID）とエクスポート・ファイルに格納された識別子が比較されます。これらの識別子が一致する場合は、型の一意のハッシュ・コードとエ



クスポート・ファイルに格納されたハッシュ・コードが比較されます。TOID またはハッシュ・コードが一致しない場合、その表の行はインポートされません。

この妥当性チェックをしてはいけない型もあります（たとえば、その型がカートリッジのインストールによって作成された場合）。パラメータ TOID\_NOVALIDATE を使用して、TOID およびハッシュ・コードと比較しない型を指定できます。詳細は、2-32 ページの「TOID\_NOVALIDATE」を参照してください。

---

**注意：** 型比較は、不正なデータを発生させないための非常に重要な機能であるため、TOID\_NOVALIDATE の使用には、特に注意してください。この機能を使用禁止にする場合は、データ型の妥当性チェックとその処理について十分な知識を持つユーザーが行ってください。

---

次の基準によって、オブジェクト型、オブジェクト表およびオブジェクト表の行の処理方法が決まります。

- オブジェクト型に関して、IGNORE=y が指定されていて、オブジェクト型がすでに存在し、そのオブジェクト識別子、ハッシュ・コードおよび型記述子が一致する場合は、エラーは通知されません。オブジェクト識別子またはハッシュ・コードが一致しない場合、パラメータ TOID\_NOVALIDATE にそのオブジェクト型を無視する設定がされていないと、エラーが通知され、そのオブジェクト型を使用している表はインポートされません。
- オブジェクト型に関して、IGNORE=n が指定されていて、そのオブジェクト型がすでに存在する場合は、エラーが通知されます。オブジェクト識別子、ハッシュ・コードまたは型記述子が一致しない場合、パラメータ TOID\_NOVALIDATE にそのオブジェクト型を無視するように設定されていないと、エラーが通知され、そのオブジェクト型を使用している表はインポートされません。
- オブジェクト表に関して、IGNORE=y が指定されていて、表がすでに存在し、そのオブジェクト識別子、ハッシュ・コードおよび型記述子が一致する場合、エラーは通知されません。行はオブジェクト表にインポートされます。同じオブジェクト識別子の行が、すでにそのオブジェクト表に存在している場合、行のインポートはエラーになります。オブジェクト識別子、ハッシュ・コードまたは型記述子が一致しない場合、パラメータ TOID\_NOVALIDATE がそのオブジェクト型を無視するように設定されていないと、エラーが通知され、そのオブジェクト型を使用している表はインポートされません。
- オブジェクト表に関して、IGNORE=n が指定されていて、オブジェクト表がすでに存在する場合は、エラーが通知され、オブジェクト表はインポートされません。

インポート・ユーティリティにオブジェクト型とオブジェクト表に関するオブジェクト識別子が保持されるため、FROMUSER パラメータおよび TOUSER パラメータを使用して、あるユーザー・スキーマから別のユーザー・スキーマにオブジェクトをインポートする場合は、次のことを考慮してください。

- FROMUSER のオブジェクト型およびオブジェクト表がターゲット表にすでに存在する場合は、TOUSER のオブジェクト型およびオブジェクト表の識別子がすでに使用されているため、エラーが発生します。インポート開始前に、FROMUSER のオブジェクト型およびオブジェクト表をシステムから削除する必要があります。
- オブジェクト表作成時に、OID AS オプションを指定して他の表と同じオブジェクト識別子が割り当てられている場合は、同じオブジェクト識別子を持つ表を両方インポートすることはできません。1 つ目の表はインポートできますが、同じオブジェクト識別子がすでに使用されているため、2 つ目の表をインポートするとエラーになります。

## 既存のオブジェクト表およびオブジェクト型の含まれている表のインポート

表領域の使用法または表の記憶域パラメータを変更するため、インポートの前に表を作成することがよくあります。表を作成する場合、以前（記憶域パラメータ以外に対して）使用していた定義と同じ定義で作成するか、互換性のある形式で作成する必要があります。オブジェクト表や、オブジェクト型の列を含む表の場合は、形式の互換性がさらに制限されます。

オブジェクト表およびオブジェクト列を含む表の場合、表が参照する各オブジェクトは、その名前、構造およびバージョン情報をエクスポート・ファイルに書き出されます。エクスポート・ユーティリティでは、必要に応じて、異なるスキーマのオブジェクト型の情報も含まれます。

インポート・ユーティリティは、表データをインポートする前に、表に必要な各オブジェクト型の存在を確認します。この確認機能には、オブジェクト型の名前の確認、インポート・システムのオブジェクト型の構造およびバージョンと、エクスポート・ファイルに書き込まれた情報との比較が含まれます。

オブジェクト型名がインポート・システムで検出され、構造またはバージョンがエクスポート・ファイルと一致しない場合、エラー・メッセージが生成され、表データはインポートされません。

インポート・パラメータ `TOID_NOVALIDATE` を使用して、特定のオブジェクトのオブジェクト型の構造およびバージョンの確認機能を使用禁止にできます。

## ネストした表のインポート

内部のネストした表は外部表とは別にエクスポートされます。したがって、内部のネストした表が正しくインポートされない場合、次のような状況が予想されます。

- 内部にネストした表を持つ表がエクスポートされ、インポート時に、表も表内の行も削除されなかったとします。IGNORE=y パラメータが指定されている場合、外部表に各行を挿入すると、制約違反が発生します。ただし、内部のネストした表のデータは正常にインポートされることがあり、その場合、内部表の行データが重複します。
- 外部表へデータをインポート中にリカバリ不能なエラーが発生した場合は、外部表の残りのデータはスキップされますが、対応する内部表の行はスキップされません。その結果、内部表の行は外部表のどの行からも参照されなくなります。
- リカバリ可能なエラーの後で内部表への挿入がエラーになる場合、外部表の行はすでに外部表にインポートされています。また、外部表および他の内部表へのデータのインポートは続行されます。その結果、不完全な論理行が作成されます。
- 内部表へのデータのインポート中にリカバリ不能なエラーが発生した場合は、その内部表の残りのデータはスキップされますが、外部表やその他のネストした表はスキップされません。

常にログ・ファイルを調べて、外部表および内部表にエラーがないかどうかを確認する必要があります。データに一貫性を持たせるためには、表データの変更や削除が必要になることがあります。

内部にネストした表は、外部表とは別にインポートされるので、インポート中に、このネストした表のデータにアクセスしようとしても失敗することがあります。たとえば、内部表の行がインポートされる前に、外部表の行にアクセスすると、ユーザーには不完全な行が返されます。

## REF データのインポート

REF 列および属性には、参照されている型のインスタンスを示す ROWID が隠されていることがあります。インポート・ユーティリティでは、ターゲット・データベースに対する ROWID は、自動的に再設定されません。ROWID を適切な値に再設定するには、次のコマンドを実行します。

```
ANALYZE TABLE [schema.]table VALIDATE REF UPDATE;
```

**参照：** ANALYZE TABLE 文の詳細は、『Oracle9i SQL リファレンス』を参照してください。

## BFIL 列およびディレクトリ別名のインポート

BFIL 列および属性で参照されているデータは、ソース・データベースからターゲット・データベースへはコピーされません。BFIL 列で参照されているファイルの名前とディレクトリ別名が伝達されるのみです。BFIL 列および属性で参照されている実際のファイルは、DBA またはユーザーが移動してください。

BFIL 列を含む表データをインポートする場合、BFIL ロケータは、ディレクトリ別名およびエクスポート時のファイル名でインポートされます。インポート・ユーティリティでは、そのディレクトリ別名またはファイルが存在するかどうかの確認は行われません。ディレクトリ別名またはファイルが存在しない場合、ユーザーが BFIL データにアクセスするとエラーが発生します。

ディレクトリ別名に関しては、エクスポート・システムで使用しているオペレーティング・システムのディレクトリ構文がインポート・システムで有効でない場合でも、インポート時にエラーは通知されません。この場合、インポート後にそのファイルのデータにアクセスするとエラーが返されます。

ディレクトリ別名がインポート・システムで有効かどうかは、DBA またはユーザーが確認してください。

## 外部関数ライブラリのインポート

インポート・ユーティリティでは、外部関数ライブラリの参照先が正しいかどうかの確認は行われません。エクスポート・ファイル上のライブラリの指定で使用されているディレクトリやファイル名の形式がインポート・システムで無効であっても、インポート時にエラーは通知されません。この場合、インポート後にそのファンクションを呼び出そうとすると、エラーが返されます。

DBA またはユーザーが手動でライブラリを移動し、ライブラリの指定がインポート・システムで有効になるようにしてください。

## ストアド・プロシージャ、ファンクションおよびパッケージのインポート

ローカルのストアド・プロシージャ、ストアド・ファンクションまたはパッケージがインポートされるときインポート・ユーティリティの動作は、COMPILE パラメータが y または n に設定されているかどうかによって異なります。

ローカルのストアド・プロシージャ、ストアド・ファンクションまたはパッケージがインポートされ、COMPILE=y が指定されている場合は、プロシージャ、ファンクション、パッケージはインポート時に再コンパイルされ、元のタイムスタンプ仕様が保持されます。コンパイルが成功すると、リモート・プロシージャによってアクセスしてもエラーは発生しません。

COMPILE=n が指定されている場合も、プロシージャ、ストアド・ファンクションまたはパッケージはインポートされます。ただし、元のタイムスタンプは失われます。次に、プロシージャ、ファンクションまたはパッケージを使用するときに、コンパイルが実行されます。

**参照：** 2-20 ページの「[COMPILE](#)」を参照してください。

## Java オブジェクトのインポート

Java オブジェクトを任意のスキーマにインポートしても、リゾルバはインポート・ユーティリティによって変更されません（リゾルバとは、Java のフルネームの解決に使用されるスキーマのリストです）。インポートの終了後、明示的または暗黙的に再検証しないかぎり、すべてのユーザー・クラスが無効な状態のままになります。暗黙的な再検証は、クラスが最初に参照されるときに実行されます。明示的な再検証は、SQL の ALTER JAVA CLASS...RESOLVE 文を使用すると実行されます。いずれの方法でもクラスは正常に解決され、有効になります。

## 外部表のインポート

インポート・ユーティリティでは、外部表の参照先が正しいかどうかの確認は行われません。エクスポート・ファイルの表の指定で使用されているディレクトリやファイル名の形式がインポート・システムで無効であっても、インポート時にエラーは通知されません。この場合、インポート後にそのファンクションを呼び出そうとすると、エラーが返されます。

DBA またはユーザーが手動で表を移動し、表の指定がインポート・システムで有効になるようにしてください。

## AQ 表のインポート

キュー表をインポートすると、基礎となっているキューや関連するディクショナリ情報もインポートされます。キューのインポートは、キュー表単位のレベルでのみ実行できます。キュー表のインポートでは、エクスポートの表処理プロシージャの前後に、キュー・ディクショナリがメンテナンスされます。

**参照：** 『Oracle9i アプリケーション開発者ガイド - アドバンスト・キューイング』を参照してください。

## LONG 列のインポート

LONG 列の長さは、最大 2GB です。インポートおよびエクスポート時には、LONG 列は各行の残りのデータとともにメモリーに収まるサイズである必要があります。ただし、LONG データはセクション単位でロードされるため、LONG 列を格納するメモリーが連続している必要はありません。

インポート・ユーティリティを使用して、LONG 列を CLOB 列に変換できます。このためには、まず、新しい CLOB 列を指定する表を作成します。インポートを実行すると、LONG データは CLOB 形式に変換されます。同じ方法で、LONG RAW 列を BLOB 列に変換できます。

## ビューのインポート

ビューは、依存順序でエクスポートされます。状況によっては、サーバー・データベースから順序を取得するのではなく、エクスポートで順序付けをする必要があります。この場合、常に正しい順序を複製できるとはかぎりません。順序が正しくない場合、ビューのインポート時にコンパイル上の警告が発行され、そのビューに関する列コメントはインポートされません。

特に、`viewa` でストアド・プロシージャ `procb` が使用され、`procb` でビュー `viewc` が使用されている場合、エクスポート・ユーティリティでは、ビュー `viewa` と `viewc` の正しい順序付けはできません。`viewa` が `viewc` より先にエクスポートされ、`procb` がインポート・システムにすでに存在する場合は、`viewa` のインポート時にコンパイル上の警告が出されます。

ビューに関する権限は、ビューにコンパイル・エラーがあってもインポートされます。ビューの作成時に、そのビューの基礎になっているオブジェクト（たとえば、表、プロシージャ、他のビューなど）が存在していない場合、ビューにコンパイル・エラーが発生する場合があります。実表が存在しない場合、実表に対する権限を付与したユーザー自身が、その実表に対して `GRANT OPTION` 付きの適正な権限を持っているかどうかを、サーバーでは検証できません。権限を付与したユーザーが適正な権限を持っていない場合、インポートされなかった表の作成後にその表にアクセスしようとすると、エラーが発生します。

他のスキーマの表を参照しているビューをインポートする場合は、インポートを実行するユーザーに、`SELECT ANY TABLE` 権限が必要です。この権限がない場合、ビューは、コンパイルされていない状態でインポートされます。ロールに権限を付与するのみでは不十分です。ビューのコンパイルには、インポートするユーザーに直接権限を付与する必要があります。

## パーティション表のインポート

エクスポートしたパーティション表と同じパーティション名またはサブパーティション名を使用してパーティション表を作成するために、`SYS_pnnn` 形式の名前もインポートされます。同じ名前のパーティション表がすでに存在している場合、これ以降の処理は `IGNORE` パラメータに指定されている値によって異なります。

`SKIP_UNUSABLE_INDEXES=y` が指定されていないかぎり、インポート時に非パーティション索引またはパーティション索引が（索引使用禁止に設定されているか、またはその他の不適合が理由で）メンテナンスできない場合は、エクスポート・データはターゲット表にインポートできません。

## ファイングレイン・アクセス・コントロールに対するサポート

ファイングレイン・アクセス・コントロール・ポリシーを使用可能にして、表をエクスポートできます。その場合は、次のことに注意してください。

ファイングレイン・アクセス・コントロール・ポリシーをリストアするには、ファイングレイン・アクセス・コントロール・ポリシーが使用可能な表を含むエクスポート・ファイルからインポートを行うための次の権限が必要です。

- 表のセキュリティ・ポリシーを回復させるための DBMS\_RLS パッケージに対する EXECUTE 権限
- 使用可能な EXPORT\_FULL\_DATABASE ロールまたは付与された EXEMPT ACCESS POLICY 権限

ファイングレイン・アクセス・コントロール・ポリシーが使用可能な表を含むエクスポートファイルからインポートを行うための正しい権限が付与されていない場合は、警告メッセージが発行されます。したがって、セキュリティ上の理由から、そのような表をエクスポートおよびインポートするユーザーは、DBA である必要があります。

**参照：** ファイングレイン・アクセス・コントロールの詳細は、『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。

## マテリアライズド・ビューおよびスナップショット

---

**注意：** 特定の状況、特にデータ・ウェアハウスに関連する場合、スナップショットは、マテリアライズド・ビューと呼ばれます。この項では、そのような場合でもスナップショットという用語を使用します。

---

スナップショット・システムには、マスター表、オプションのスナップショット・ログおよびスナップショット自体の3つのオブジェクトがあり、相互に関連しています。表（マスター表、スナップショット・ログ表定義およびスナップショット表）は、個別にエクスポートできます。スナップショット・ログは、対応付けられたマスター表をエクスポートしないかぎり、エクスポートできません。スナップショットをエクスポートできるのは、全データベース・エクスポートおよびユーザー・モード・エクスポートの場合のみです。表モードではエクスポートできません。

この項では、これらのオブジェクトのインポート時に高速リフレッシュが受ける影響について説明します。

**参照：** インポート固有の移行と互換性、およびスナップショットとスナップショット・ログの詳細は、『Oracle9i アドバンスド・レプリケーション』を参照してください。

## スナップショット・ログ

インポート先のデータベースにマスター表がすでに存在し、そのマスター表にスナップショット・ログがある場合は、ダンプ・ファイルのスナップショット・ログがインポートされます。

ROWID スナップショット・ログのエクスポートでは、スナップショット・ログに記録されている ROWID はインポートした後には意味を持ちません。このため、各 ROWID のスナップショットによる最初的高速リフレッシュは失敗し、完全リフレッシュが必要であることを示すエラーが発生します。

リフレッシュのエラーを回避するには、ROWID のスナップショット・ログをインポートしてから完全リフレッシュを実行してください。完全リフレッシュを実行すると、後続的高速リフレッシュが適切に行われます。これに対し、主キー・スナップショット・ログをエクスポートした場合は、インポートした後でも、主キーのスナップショットにより、高速リフレッシュを実行できます。

**参照：** 主キーのスナップショットの詳細は、『Oracle9i アドバンスド・レプリケーション』を参照してください。

## スナップショット

エクスポート・ファイルからリストアされたスナップショットは、前の状態に戻ってしまいます。インポートでは、最後のリフレッシュが実行された時刻が、スナップショット表定義の一部としてインポートされます。次のリフレッシュ時刻を計算する機能もインポートされます。

各リフレッシュによって、署名が付けられます。高速リフレッシュでは、スナップショットを最新に保つため、その署名の時刻から日付を決定するログ・エントリが使用されます。高速リフレッシュが完了した時点で署名は削除され、新しい署名が付けられます。他のスナップショットのリフレッシュに必要でないログ・エントリ（残っている最も古い署名よりも前の時刻を持つすべてのログ・エントリ）も削除されます。

### スナップショットのインポート

エクスポート・ファイルからスナップショットをリストアすると、問題が発生する場合があります。

スナップショットが時刻 A にリフレッシュされ、時刻 B にエクスポートされ、時刻 C に再びリフレッシュされたとします。破損などの問題が発生したため、スナップショットを削除してインポートしなおすことによってリストアする必要があります。新しくインポートしたスナップショットには、時刻 A に実行した最後のリフレッシュが記録されていますが、高速リフレッシュに必要なログ・エントリが存在しなくなっている可能性があります。ログ・エントリが存在する場合は（たとえば、リフレッシュする必要のある別のスナップショットに必要なため）、このエントリが使用され、高速リフレッシュは正常に完了します。ログ・エントリが存在しない場合、高速リフレッシュは失敗し、完全リフレッシュが必要であることを示すエラーが発生します。



## 異なるスキーマへのスナップショットのインポート

スナップショット、スナップショット・ログおよび関連項目は、DDL 文で明示的に指定されたスキーマ名でエクスポートされます。したがって、スナップショットおよびその関連項目を異なるスキーマにインポートすることはできません。

FROMUSER および TOUSER を使用してスナップショット・データをインポートしようとする、インポート・ログ・ファイルにエラーが書き込まれ、その項目はインポートされません。

## トランスポートابل表領域

トランスポートابل表領域によって、一連の表領域を、ある Oracle データベースから他のデータベースに移動できます。

そのためには、表領域を読み込み専用にし、表領域のデータ・ファイルをコピーしてから、エクスポート・ユーティリティおよびインポート・ユーティリティを使用して、データ・ディクショナリに格納されているデータベース情報（メタデータ）を移動します。データ・ファイルおよびメタデータのエクスポート・ファイルの両方を、ターゲット・データベースにコピーする必要があります。これらのファイルの転送は、オペレーティング・システムのコピー機能、バイナリ・モード FTP、CD-ROM への出力などのような、フラットまたはバイナリ・ファイルのコピー機能を使用して行われます。

データ・ファイルのコピーおよびメタデータのインポート後、表領域を任意に読み書き両用モードにできます。

トランスポートابل表領域のメタデータを含むエクスポート・ファイルの作成については、1-61 ページの「[トランスポートابل表領域](#)」を参照してください。

次のパラメータで、トランスポートابل表領域のメタデータのインポートを使用可能にできます。

- TRANSPORT\_TABLESPACE
- TABLESPACES
- DATAFILES
- TTS\_OWNERS

詳細は、2-34 ページの「[TRANSPORT\\_TABLESPACE](#)」、2-32 ページの「[TABLESPACES](#)」、2-21 ページの「[DATAFILES](#)」および 2-34 ページの「[TTS\\_OWNERS](#)」を参照してください。

### 参照：

- 表領域を他のデータベースに移動またはコピーする方法の詳細は、『Oracle9i データベース管理者ガイド』を参照してください。
- トランスポートابل表領域の機能の詳細は、『Oracle9i データベース概要』を参照してください。

## 記憶域パラメータ

デフォルトでは、表は、元の表領域にインポートされます。

その表領域が存在しない場合、またはユーザーがその表領域に十分な割当て制限を持っていない場合、次の表の場合を除いて、そのユーザーにはデフォルトの表領域が割り当てられません。

- パーティション表
- 特定の型の表
- LOB 型列、VARRAY 型列または OPAQUE 型列を含む表
- オーバーフロー・セグメントがある IOT を含む表

ユーザーがデフォルトの表領域に対する十分な割当て制限を持っていない場合、そのユーザーの表はインポートされません。この制限の利用方法の詳細は、2-70 ページの「[表領域を再編成する方法](#)」を参照してください。

### OPTIMAL パラメータ

ロールバック・セグメントのための記憶域パラメータ OPTIMAL は、エクスポートおよびインポート時には保持されません。

### OID 索引と LOB 列の記憶域パラメータ

表は、その表の現行の記憶域パラメータを使用してエクスポートされます。オブジェクト表に関しては、OIDINDEX の作成時に、OIDINDEX の現行の記憶域パラメータおよび名前が設定されている場合は、それらを使用して作成されます。LOB 型列、VARRAY 型列または OPAQUE 型列が含まれている表に関しては、LOB 型データ、VARRAY 型データまたは OPAQUE 型データは、それらの現行の記憶域パラメータを使用して作成されます。

エクスポートの前に、ユーザーが既存の表の記憶域パラメータを変更する場合がありますが、このような場合、表は変更された記憶域パラメータを使用してエクスポートされます。ただし、LOB データの記憶域パラメータは、エクスポートの前には変更できません（たとえば、LOB 列のチャンク・サイズ、LOB 列が CACHE または NOCACHE か、など）。

LOB データと LOB 索引は、格納している表と同じ表領域に常駐することはできません。このデータの表領域は、インポート時に読み込み / 書き込みが可能である必要があります。そうでない場合、表はインポートされません。

LOB データまたは LOB 索引がインポート時に存在しない表領域にある場合、またはユーザーがその表領域に対して必要な割当て制限を持っていない場合、表はインポートされません。表領域の句は、表に関する句も含めて複数の句を同時に指定できるため、インポート時にエラーが発生しても、インポート・ユーティリティではどの表領域句が原因のエラーかを特定できません。

## 記憶域パラメータの上書き

インポート・ユーティリティを使用してデータをインポートする前に、別の記憶域パラメータで、事前に大きな表を作成した方がよい場合があります。その場合は、コマンドラインまたはパラメータ・ファイルに `IGNORE=y` を指定します。

## エクスポート・パラメータ COMPRESS

エクスポート時のデフォルトによって、初期エクステンツにインポートされる表のすべてのデータを整理統合するように、記憶域パラメータが調整されます。初期エクステンツのサイズを元のまま保つには、エクスポート時にエクステンツが整理統合されないように `COMPRESS=n` を指定します。詳細は、1-18 ページの「[COMPRESS](#)」を参照してください。

## 読み込み専用表領域

読み込み専用表領域はエクスポート可能です。インポートでは、表領域がターゲット・データベース内に存在しない場合、読み込み / 書き込み表領域として表領域が作成されます。読み込み専用機能が必要な場合は、インポート後にその表領域を手動で読み込み専用にしてください。

ターゲット・データベース内に表領域がすでに存在し、読み込み専用である場合は、インポート前にこの表領域を読み込み / 書き込み可能にする必要があります。

## 表領域を削除する方法

インポート前に、オブジェクトに別の表領域を使用するように再定義すると、表領域を削除できます。imp コマンドの発行時には、`IGNORE=y` を指定します。

表領域を削除するには、通常、全データベース・エクスポートを実行し、(ログオフの前に) 削除する表領域と同名の表領域をブロック数 0 (ゼロ) で作成します。`IGNORE=y` が指定されていると、インポート時にその表領域に関する `CREATE TABLESPACE` 文はエラーとなります。これにより、削除対象である不要な表領域は作成されません。

その表領域のすべてのオブジェクト（ただし、パーティション表、特定の型の表、LOB 列または `VARRAY` 列を含む表またはオーバーフロー・セグメントのある `IOT` を除く）が、そのオブジェクトの所有者のデフォルトの表領域にインポートされます。インポート・ユーティリティでは、エラーの原因となった表領域を特定できません。かわりに、ユーザー自身が表の作成後、`IGNORE=y` を指定して表のインポートを実行する必要があります。

その表領域が存在しない場合、またはデフォルトの表領域に対するユーザーの割当て制限が十分でない場合は、オブジェクトはデフォルトの表領域にインポートされません。

## 表領域を再編成する方法

ユーザーの割当て制限が十分な場合、そのユーザーの表はエクスポート元と同じ表領域にインポートされます。その表領域がもう存在しないか、またはユーザーの割当て制限が十分でない場合は、そのユーザーに対するデフォルトの表領域が適用されます。ただし、パーティション表、LOB 列または VARRAY 列が含まれている表、特定の型の表およびオーバーフロー・セグメントのある IOT には適用されません。この条件を利用して、表領域間でユーザーの表を移動できます。

たとえば、全データベース・エクスポートを実行した後、joe の表を表領域 A から表領域 B に移動する必要があるとします。この場合には、次の手順を実行します。

1. joe が UNLIMITED TABLESPACE 権限を持っている場合、その権限を取り消します。表領域 A に対する joe の割当て制限を 0（ゼロ）に設定します。さらに、このような権限または割当て制限を含む可能性のあるすべてのロールを取り消します。

権限の取消しはカスケード化されません。したがって、joe によって他のロールを付与されたユーザーは影響を受けません。

2. joe の表をエクスポートします。
3. 表領域 A から joe の表を削除します。
4. joe に表領域 B の割当て制限を付与し、joe のデフォルトの表領域とします。
5. joe の表をインポートします（デフォルトでは、joe の表は表領域 B にインポートされます）。

## 統計情報のインポート

統計情報がエクスポート時に必要で、表にアナライザ統計が利用できる場合、エクスポートによって ANALYZE 文が発行され、表の統計情報が再計算されてダンプ・ファイルに書き込まれます。ほとんどの場合、表、索引および列に対する計算済オブティマイザ統計情報も、ダンプ・ファイルにエクスポートされます。エクスポート・パラメータの詳細は、1-28 ページの「[STATISTICS](#)」を、インポート・パラメータの詳細は、2-29 ページの「[STATISTICS](#)」を参照してください。

ANALYZE 文の実行には時間がかかるため、通常のインポートでは、エクスポートによって保存される ANALYZE 文を計算するのではなく、表（およびその索引や列）の計算済オブティマイザ統計情報を使用してください。デフォルトでは、エクスポート・ダンプ・ファイルにある計算済統計情報が使用されます。

エクスポート・ユーティリティによって、計算済統計情報に問題ありというフラグが付けられる場合もあります。詳細は、1-28 ページの「[STATISTICS](#)」を参照してください。問題のない統計情報のみがインポートされる場合もあります。また、次の場合には、計算済統計情報がインポートされないこともあります。

- ダンプ・ファイル、インポート・クライアント、インポート・データベース間のキャラクタ・セット変換（計算済統計情報で暗黙的に照合順序が変更されている可能性が高いため）。
- 表のインポート時に行エラーが発生した場合。
- パーティション・レベル・インポートが実行された場合（列統計情報が、すでに正確ではないため）。

---

**注意：** ROWS=n を指定しても、計算済統計情報は使用できません。この機能により、本番データベースからの統計情報を使用して、非本番データベース上で問合せの実行計画のチューニングを行うことができます。このような場合は、STATISTICS=SAFE を指定する必要があります。

---

場合によっては、インポート時に、計算済統計情報ではなく、常に ANALYZE 文を使用する必要があります。たとえば、分散データベースから収集した統計情報は、そのデータが圧縮形式でインポートされると、適切でなくなる場合があります。このような場合は、インポート時に STATISTICS=RECALCULATE を指定して、統計情報を再計算する必要があります。

インポート時に統計情報を確定しない場合は、STATISTICS=NONE を指定する必要があります。

## エクスポート・ユーティリティおよびインポート・ユーティリティを使用したデータベース移行のパーティション化

エクスポート・ユーティリティおよびインポート・ユーティリティを使用して大規模データベースを移行する場合、移行を複数のエクスポート・ジョブおよびインポート・ジョブにパーティション化するとより効率的です。移行をパーティション化する場合は、次のメリットおよびデメリットに注意してください。

### 移行をパーティション化する場合のメリット

移行をパーティション化すると、次のメリットがあります。

- 多くのサブジョブを平行に実行できるため、移行に必要な時間を削減できます。
- 最初のエクスポート・ジョブが完了するとすぐにインポート・ユーティリティを起動できます。

## 移行をパーティション化する場合のデメリット

移行をパーティション化すると、次のデメリットがあります。

- エクスポートおよびインポートのプロセスがより複雑になります。
- 特定の型のオブジェクトに対する相互スキーマ参照のサポートが損なわれます。たとえば、別のスキーマの表に対する外部キー制約を持つ表がスキーマに含まれている場合、その表を依存スキーマにインポートしたときに、必要な親レコードが存在しない場合があります。

## エクスポート・ユーティリティおよびインポート・ユーティリティを使用したデータベース移行のパーティション化方法

データベースの移行をパーティション化方法で実行するには、次の手順に従います。

1. データベースのすべての最上位メタデータに、次のコマンドを発行します。
  - a. `exp dba/password FILE=full FULL=y CONSTRAINTS=n TRIGGERS=n ROWS=n INDEXES=n`
  - b. `imp dba/password FILE=full FULL=y`
2. データベースの各スキーマ *n* に、次のコマンドを発行します。
  - a. `exp dba/password OWNER=scheman FILE=scheman`
  - b. `imp dba/password FILE=scheman FROMUSER=scheman TOUSER=scheman IGNORE=y`

すべてのエクスポートは平行で実行できます。full.dmp のインポートが完了すると、残りのインポートも平行で実行できます。

## 以前のリリースの Oracle のエクスポート・ファイルの使用法

次の項では、以前のリリースの Oracle データベース・サーバーから Oracle9i サーバーにデータをインポートする場合に、考慮すべき点について説明します。

**参照：** 1-63 ページの「[リリースおよびバージョンが異なるエクスポート・ユーティリティの使用法](#)」を参照してください。

## Oracle7 のエクスポート・ファイルの使用法

この項では、Oracle7 のデータベースのデータを、Oracle9i サーバーにインポートする場合のガイドラインおよび制限について説明します。

**参照：** 『Oracle9i データベース移行ガイド』を参照してください。

## DATE 列に関する CHECK 制約

Oracle9i では、DATE 列に関する CHECK 制約を有効にするために、TO\_DATE 関数を使用して、日付書式を指定する必要があります。Oracle8i より前のバージョンでは、この関数は必要なかったため、Oracle8i より前のバージョンの Oracle データベースからデータをインポートした場合は、TO\_DATE 関数が使用されていない場合があります。このような場合、制約は Oracle9i データベースにインポートされますが、ディクショナリで無効のフラグが付けられます。

カタログ・ビュー DBA\_CONSTRAINTS、USER\_CONSTRAINTS および ALL\_CONSTRAINTS を使用すると、制約を識別できます。データベースに無効な日付制約があると、警告メッセージが発行されます。

## Oracle バージョン 6 のエクスポート・ファイルの使用方法

この項では、Oracle バージョン 6 のデータベースのデータを、Oracle9i サーバーにインポートする場合のガイドラインおよび制限について説明します。

### ユーザー権限

ユーザー定義は、Oracle データベースへのインポート時に CREATE USER 文によって作成されます。したがって、旧バージョンのエクスポートで作成されたエクスポート・ファイルからインポートするときは、ユーザーが自動的に CREATE SESSION 権限を付与されることはありません。

### CHAR 列

Oracle バージョン 6 の CHAR 列は、自動的に Oracle の VARCHAR2 データ型に変換されます。

### 整合性制約の状態

NOT NULL 制約は ENABLED としてインポートされます。その他の制約はすべて DISABLED としてインポートされます。

### デフォルト列値の長さ

表のデフォルトの列値が列の最大サイズを超えていると、Oracle9i へのインポートの実行時に次のエラーが発生します。

ORA-01401: 列に挿入した値が大きすぎます。

Oracle バージョン 6 では、CREATE TABLE 文で列はチェックされず、列がデフォルト値を保持できるだけの長さかどうかを確認されませんでした。そのため、このような表をバージョン 6 のデータベースにインポートすることができました。ただし、Oracle9i サーバーでは、CREATE TABLE 文で列がチェックされます。そのため、バージョン 6 のデータベースにインポートできた表が、Oracle9i にはインポートできない場合があります。

関数によって返される値がデフォルトの場合、その関数によって返される可能性のある最大値を保持できるだけの列の長さが必要になります。長さが足りない場合、エクスポート・ファイルに記録されている CREATE TABLE 文によって、インポート時にエラーが返されます。

---

**注意：** Oracle7 では、USER 関数の最大値が大きくなったため、USER のデフォルト値を持つ列の場合、長さが足りないことがあります。USER 関数によって返される最大サイズを判別するには、次の SQL 文を実行してください。

```
DESCRIBE user_sys_privs
```

表示される USERNAME 列の長さが、USER 関数によって戻される列の最大長です。

---

**参照：**『Oracle9i データベース移行ガイド』を参照してください。

## Oracle バージョン 5 のエクスポート・ファイルの使用方法

Oracle9i のインポート・ユーティリティでは、Oracle リリース 5.1.22 以上によって作成されたエクスポート・ダンプ・ファイルが読み込まれます。次のことに注意してください。

- CHAR 列は自動的に VARCHAR2 に変換されます。
- NOT NULL 制約は ENABLED としてインポートされます。
- インポートするクラスタの索引が自動的に作成されます。

## リリースおよびバージョンが異なるエクスポート・ユーティリティおよびインポート・ユーティリティの使用上の制限

異なるリリースのエクスポート・ユーティリティおよびインポート・ユーティリティを使用する場合、次の制限が適用されます。

- エクスポート・ダンプ・ファイルは、特別なバイナリ形式で格納されているため、インポート・ユーティリティによる読み込み専用です。
- すべてのエクスポート・ダンプ・ファイルは、上位リリースの Oracle データベース・サーバーにインポートできます。
- エクスポート・ダンプ・ファイルは、下位のバージョンまたはリリースのインポート・ユーティリティでは読み込みができません。したがって、リリース 8.0 のインポート・ユーティリティでは、リリース 8.1 のエクスポート・ファイルはインポートできません。また、バージョン 7 のインポート・ユーティリティでは、バージョン 8 のエクスポート・ダンプ・ファイルはインポートできません。



- インポート・ユーティリティでは、リリース 5.1.22 以上のエクスポート・ユーティリティで作成されたエクスポート・ダンプ・ファイルの読み込みができます。
- インポート・ユーティリティでは、上位のメンテナンス・リリースまたはバージョンのエクスポート・ユーティリティで作成されたエクスポート・ダンプ・ファイルの読み込みはできません。たとえば、リリース 8.0 のインポート・ユーティリティでは、リリース 8.1 のエクスポート・ダンプ・ファイルはインポートできません。また、バージョン 7 のインポート・ユーティリティでは、バージョン 8 のエクスポート・ダンプ・ファイルはインポートできません。
- Oracle バージョン 6 以下のエクスポート・ユーティリティは、Oracle8 以上のデータベースには使用できません。
- 下位バージョンのエクスポート・ユーティリティを上位バージョンの Oracle データベース・サーバーで実行すると、下位バージョンに存在しないデータベース・オブジェクトのカテゴリは、常にエクスポートから除外されます。たとえば、Oracle7 データベース・サーバーにはパーティション表が存在しませんでした。そのため、Oracle8 のパーティション表を Oracle7 データベースに移動させる必要がある場合は、まず、表を非パーティション表に再編成する必要があります。
- ダイレクト・パスの場合も従来型パスの場合も、Oracle9i のエクスポート・ユーティリティを使用して作成されたエクスポート・ファイルは、旧リリースのインポート・ユーティリティとは互換性がないため、インポートに使用できるのは Oracle9i のインポート・ユーティリティのみです。下位互換性が問題となる場合は、Oracle9i データベースに対して下位のリリースまたはバージョンのエクスポート・ユーティリティを使用します。
- Oracle8i リリース 8.1.7 データベースからのジョブ・キューは、下位のリリースのデータベースにインポートできません。したがって、インポートの終了後、手動でジョブを再開する必要があります。

## CHARSET パラメータ

デフォルト: なし

このパラメータは、Oracle バージョン 5 およびバージョン 6 のエクスポート・ファイルにのみ適用されます。このパラメータはできるだけ使用しないでください。これは、下位バージョンとの互換性のためにのみ用意されているパラメータです。このパラメータは将来廃止される予定です。

Oracle バージョン 5 およびバージョン 6 のエクスポート・ファイルには、データベース・キャラクタ・セット識別子が含まれていません。ただし、バージョン 5 またはバージョン 6 のエクスポート・ファイルでは、ユーザー・セッションで使用されているキャラクタ・セットが ASCII かまたは EBCDIC かということが認識されます。

このパラメータを使用して、エクスポート時に実際に使用されるキャラクタ・セットを示します。指定したキャラクタ・セットがエクスポート・ファイル内のキャラクタ・セットに基づく ASCII または EBCDIC かどうかを検証されます。

CHARSET パラメータの値を指定しなかった場合、エクスポート・ファイルのキャラクタ・セットが ASCII のときは、インポート時に、ユーザー・セッションのキャラクタ・セットは ASCII であると検証されます。または、エクスポート・ファイルのキャラクタ・セットが EBCDIC のときは、インポート時に、ユーザー・セッションのキャラクタ・セットは EBCDIC であると検証されます。

Oracle7 以上を使用している場合、キャラクタ・セットはエクスポート・ファイル内で指定され、現行のデータベースで使用されているキャラクタ・セットへ自動的に変換されます。このパラメータは、エクスポート・ファイルのキャラクタ・セットが、要求されている値と一致するかどうかを確認するためのみに使用されます。キャラクタ・セットが一致しない場合はエラーになります。

# 第II部

---

## SQL\*Loader

第II部では、SQL\*Loader ユーティリティについて説明します。この部に含まれる章は、次のとおりです。

### 第3章「SQL\*Loader の概念」

この章では、SQL\*Loader の機能について説明します。また、データ・ロードの概念（オブジェクト・サポートも含む）についても説明します。さらに、SQL\*Loader への入力、データベースの事前準備および SQL\*Loader からの出力についても説明します。

### 第4章「SQL\*Loader コマンドライン・リファレンス」

この章では、SQL\*Loader で使用するコマンドラインの構文について説明します。コマンドライン引数、SQL\*Loader のメッセージを抑制する方法、バインド配列のサイズ指定などについて説明します。

### 第5章「SQL\*Loader 制御ファイル・リファレンス」

この章では、SQL\*Loader の設定に使用する制御ファイルの構文、およびデータを Oracle の形式にマップする方法を SQL\*Loader に記述する方法について説明します。詳細な構文図を示すとともに、データ・ファイル、表と列、データの位置、ロードするデータの型と形式などの指定に関する情報についても説明します。

### 第6章「フィールド・リスト・リファレンス」

この章では、SQL\*Loader 制御ファイルのフィールド・リストのセクションについて説明します。フィールド・リストには、位置、データ型、条件、デリミタなど、ロードするフィールドの情報が提供されます。

### 第7章「オブジェクト、LOB およびコレクションのロード」

この章では、オブジェクト表および REF 列のロードに加えて、様々な形式でのオブジェクトのロード方法について説明します。また、LOB および列のロードについても説明します。

### 第8章「SQL\*Loader ログ・ファイル・リファレンス」

この章では、ログ・ファイルに記述される情報について説明します。

## 第9章「従来型パス・ロードおよびダイレクト・パス・ロード」

この章では、従来型パス・ロードとダイレクト・パス・ロードの違いを説明します。ダイレクト・パス・ロードは、大量のデータを従来より高速にロードするための高パフォーマンス・オプションです。

## 第10章「SQL\*Loader の事例」

この章では、様々な事例から SQL\*Loader の機能を説明します。可変長データ、固定形式レコード、自由形式ファイルのロード方法、複数の物理レコードを1件の論理レコードとしてロードする方法、複数の表のロード方法、ダイレクト・パスを使用したロード方法、およびオブジェクト、コレクション、REF 列のロード方法について説明します。

---

## SQL\*Loader の概念

この章では、SQL\*Loader による Oracle データベースへのデータのロードについて、基本的な概念を説明します。この章の内容は、次のとおりです。

- SQL\*Loader の機能
- SQL\*Loader 制御ファイル
- 入力データおよびデータ・ファイル
- LOBFILE および SDF
- データ変換およびデータ型仕様
- 廃棄レコードおよび拒否レコード
- ログ・ファイルおよびログ情報
- 従来型パス・ロード、ダイレクト・パス・ロードおよび外部表ロード
- オブジェクト、コレクションおよび LOB のロード
- パーティション・オブジェクトのサポート
- アプリケーション開発:ダイレクト・パス・ロード API

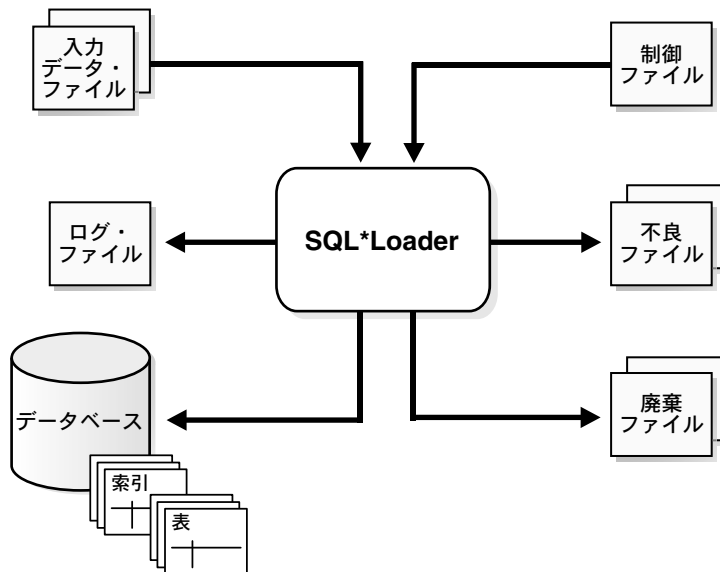
## SQL\*Loader の機能

SQL\*Loader を使用して、外部ファイルのデータを Oracle データベースの表にロードします。強力なデータ解析エンジンによって、あらゆるデータ形式のデータ・ファイルに対応できます。SQL\*Loader を使用して、次のことが可能です。

- 同一のロード・セッションでの複数のデータ・ファイルからのデータのロード。
- 同一のロード・セッションでの複数の表へのデータのロード。
- データのキャラクタ・セットの指定。
- ロード・データの選択（レコード値に基づいたロード）。
- ロード前の、SQL 関数を使用したデータ処理。
- 指定した列に対する、一意の順序キーの生成。
- オペレーティング・システムのファイル・システムを使用したデータ・ファイルへのアクセス。
- ディスク、テープまたは Named Pipe からのデータのロード。
- 高度なエラー・レポートの生成による、トラブルシューティングの支援。
- 複合オブジェクト・リレーショナル・データの任意のロード。
- セカンダリ・データ・ファイル（SDF）を使用した、LOB およびコレクションのロード。
- 従来型パス・ロードまたはダイレクト・パス・ロードの使用。従来型パス・ロードでは高い柔軟性を、ダイレクト・パス・ロードでは優れたロード・パフォーマンスを提供します。詳細は、[第 9 章](#)を参照してください。
- DB2 ロード・ユーティリティの制御ファイルをほとんど変更せずに、SQL\*Loader 制御ファイルとして使用。詳細は、[付録 B](#)を参照してください。

一般的な SQL\*Loader セッションでは、SQL\*Loader の動作を制御する制御ファイルと 1 つ以上のデータ・ファイルが入力用に使用されます。SQL\*Loader の出力先は、データがロードされる Oracle データベース、ログ・ファイル、不良ファイルで、廃棄ファイルに出力される場合もあります。[図 3-1](#) に、SQL\*Loader セッションの流れの例を示します。

図 3-1 SQL\*Loader の概要



## SQL\*Loader 制御ファイル

制御ファイルは、SQL\*Loader が解釈できる言語で記述されたテキスト・ファイルです。制御ファイルは、データの場所、データの分析と解釈方法、データの挿入先などを SQL\*Loader に通知します。

制御ファイルには、大きく分けて 3 つのセクションがあります。

第 1 セクションには、セッション全体の情報が記述されます。たとえば、次のような情報です。

- バインドサイズ、行、スキップ・レコードなどのようなグローバル・オプション
- 入力データの配置先を指定する INFILE 句
- ロードされるデータ

第 2 セクションは、1 つ以上の INTO TABLE ブロックで構成されています。それぞれのブロックには、表名、その表の列などの、データがロードされる表についての情報が含まれています。

第 3 セクションはオプションで、このセクションがある場合は、入力データが記述されます。

制御ファイルの構文には、次の注意事項があります。

- 構文は、自由形式で記述できます（文は複数行になってもかまいません）。
- 大文字と小文字は、一重引用符または二重引用符で囲まれた文字列の場合のみ区別され、それ以外では区別されません。
- 先頭にハイフンを2つ (--) 続けて入力することによって、コメントを挿入できます。ハイフンから行の終わりまでがコメントになります。オプションである第3セクションでは、二重ハイフンがコメントとしてではなくデータとして解釈されるため、このセクションでのコメントはサポートされません。
- CONSTANT キーワードは、SQL\*Loader では特別な意味があり予約されています。競合の可能性を回避するために、表または列の名前に CONSTANT という語を使用しないことをお勧めします。

**参照：** 制御ファイルの構文およびセマンティクスの詳細は、[第5章](#)を参照してください。

## 入力データおよびデータ・ファイル

制御ファイルに指定された1つ以上のファイルなどから、SQL\*Loader にデータが読み込まれます。SQL\*Loader から見ると、データ・ファイルのデータはレコードとして構成されています。データ・ファイルには、固定レコード形式、可変レコード形式またはストリーム・レコード形式があります。レコード形式は、INFILE パラメータを使用して制御ファイルに指定することができます。レコード形式が指定されない場合、デフォルトはストリーム・レコード形式になります。

---

**注意：** 制御ファイル内部でデータが指定されている場合（INFILE \* が制御ファイルに指定されている場合）、そのデータはデフォルトでレコード終了記号を使用したストリーム・レコード形式として解釈されます。

---

### 固定レコード形式

固定レコード形式のファイルでは、データ・ファイルにあるすべてのレコードが同じバイト長です。この形式は柔軟性はありませんが、その結果、可変長またはストリーム形式よりも高いパフォーマンスを得ることができます。また、固定形式は簡単に指定できます。次に例を示します。

```
INFILE datafile_name "fix n"
```

ここでは、特殊なデータ・ファイルが、SQL\*Loader によって全レコード *n* バイト長の固定レコード形式で解釈されるように指定しています。



例 3-1 に、固定レコード形式で解釈されるようにデータ・ファイルを指定する制御ファイルを示します。この例では、5 つの物理レコードがあります。ピリオド (.) が空白を示すと想定すると、第 1 の物理レコードは [001,・d,.] で、11 バイト（シングルバイト・キャラクタ・セットと想定）です。第 2 のレコードは [0002,fghi,\n] で、改行文字（11 バイト目）が続きます。改行文字は、固定レコード形式では必要ありません。

文字長セマンティクスが使用されているファイルでも、長さは常にバイト単位で解析されます。これは、文字長セマンティクスで処理されるフィールドとバイト長セマンティクスで処理されるフィールドがファイル内に混在する可能性があるため必要です。詳細は、5-22 ページの「[文字長セマンティクス](#)」を参照してください。

### 例 3-1 固定レコード形式でのデータのロード

```
load data
infile 'example.dat' "fix 11"
into table example
fields terminated by ',' optionally enclosed by '"'
(col1, col2)
```

```
example.dat:
001,   cd, 0002,fghi,
00003,lmn,
1, "pqrs",
0005,uvwxx,
```

## 可変レコード形式

可変レコード形式のファイルでは、文字フィールドの各レコード長がデータ・ファイルの各レコードの開始位置に含まれています。この形式は、固定レコード形式より柔軟性があり、ストリーム・レコード形式よりパフォーマンスに優れています。可変レコード形式の場合、たとえば、次のように指定できます。

```
INFILE "datafile_name" "var n"
```

*n* には、レコード長フィールドのバイト数を指定します。指定しない場合、長さは 5 バイトとみなされます。*n* に、40 より大きい値を指定すると、エラーになります。

例 3-2 に、ファイル名が `example.dat` でレコード長フィールドが 3 バイト長の可変レコード形式の入力データに対する制御ファイルの指定を示します。`example.dat` データ・ファイルは、3 つの物理レコードで構成されています。1 つ目のレコードは 009（すなわち 9）バイト長、2 つ目のレコードは 010（すなわち 1 バイトの改行を含めて 10）バイト長、3 つ目は 012（同様に 1 バイトの改行を含む）バイト長で指定されています。改行文字は、可変レコード形式では必要ありません。ここでは、シングルバイト・キャラクタ・セットであるとします。

文字長セマンティクスが使用されているファイルでも、長さは常にバイト単位で解析されます。これは、文字長セマンティクスで処理されるフィールドとバイト長セマンティクスで処

理されるフィールドがファイル内に混在する可能性があるため必要です。詳細は、5-22 ページの「[文字長セマンティクス](#)」を参照してください。

### 例 3-2 可変レコード形式でのデータのロード

```
load data
infile 'example.dat' "var 3"
into table example
fields terminated by ',' optionally enclosed by '"'
(col1 char(5),
 col2 char(7))

example.dat:
009hello,cd,010world,im,
012my,name is,
```

## ストリーム・レコード形式

ストリーム・レコード形式では、レコードをサイズで指定してではなく、SQL\*Loader でレコード終了記号を読み込むことによって、レコードが確認されます。ストリーム・レコード形式は最も柔軟性のある形式ですが、パフォーマンスに影響する場合があります。ストリーム・レコード形式として指定するには、次のように指定します。

```
INFILE datafile_name ["str terminator_string"]
```

*terminator\_string* には、次の '*char\_string*' または *X'hex\_string*' を指定します。

- '*char\_string*' は、一重引用符または二重引用符で囲まれた文字列です。
- *X'hex\_string*' は、16 進形式のバイト列です。

*terminator\_string* に、特別な（印字不可能な）文字が含まれる場合、*X'hex\_string*' で指定する必要があります。ただし、一部の印字不可能な文字列（'*char\_string*'）はバックスラッシュを使用すると指定できます。次に例を示します。

- \n LF
- \t 水平タブ
- \f 改ページ
- \v 垂直タブ
- \r 改行

セッションに対して NLS\_LANG パラメータに指定されているキャラクタ・セットが、データ・ファイルのキャラクタ・セットと異なる場合、文字列はデータ・ファイルのキャラクタ・セットに変換されます。これは、SQL\*Loader でデフォルトのレコード終了記号が確認される前に行われます。

16 進文字列はデータ・ファイルのキャラクタ・セット内にあるとみなされ、変換は実行されません。

UNIX ベースのプラットフォームでは、*terminator\_string* を指定しない場合、デフォルトで LF 文字 `\n` が使用されます。

Windows NT では、*terminator\_string* を指定しない場合、`\n` または `\r\n` のうちデータ・ファイル内で先に現れる文字がレコード終了記号として使用されます。データ・ファイルの 1 つ以上のレコードで、フィールドに `\n` が埋め込まれていることがわかっている場合に、`\r\n` をレコード終了記号として使用するには、*terminator\_string* を指定する必要があります。

例 3-3 に、*terminator\_string* が文字列 `'|\n'` で指定されている場合に、ストリーム・レコード形式でデータをロードする方法を示します。バックスラッシュを使用すると、文字列に印字可能な改行文字を指定することができます。

### 例 3-3 ストリーム・レコード形式でのデータのロード

```
load data
infile 'example.dat' "str '|\n'"
into table example
fields terminated by ',' optionally enclosed by ''
(col1 char(5),
 col2 char(7))
```

```
example.dat:
hello,world,|
james,bond,|
```

## 論理レコード

入力データは、指定されたレコード形式で物理レコードに編成されます。デフォルトでは、物理レコードは論理レコードですが、複数の物理レコードが 1 件の論理レコードに結合される場合もあります。

次のいずれかの方法によって、論理レコード形式に結合されます。

- 物理レコードの固定数を、それぞれの論理レコードの形式に組み合わせます。
- 特定の条件が真である場合、物理レコードを論理レコードに組み合わせます。

#### 参照：

- 5-26 ページの「[物理レコードからの論理レコードの作成](#)」を参照してください。
- 連続フィールドを使用して、複数の物理レコードを 1 つの論理レコードに結合する方法の例については、10-15 ページの「[事例 4: 結合された物理レコードのロード](#)」を参照してください。

## データ・フィールド

論理レコードが作成されると、フィールドが設定されます。フィールド設定では、制御ファイルのフィールド指定を使用して、論理レコードのデータのどの部分が制御ファイルのフィールドに対応しているのかを SQL\*Loader で判断します。2 つ以上のフィールド指定に同じデータを使用できます。また、論理レコードに、制御ファイルのフィールド指定で使用されないデータを含めることもできます。

ほとんどの場合、制御ファイルのフィールドに、論理レコードの特定の位置や長さの指定が必要です。この部分は、次のような形式で指定します。

- データ・フィールドの開始バイト位置または終了位置（あるいはその両方）を指定できます。この指定形式に柔軟性はありませんが、フィールド設定によって高パフォーマンスが得られます。
- 特殊なデータ・フィールドの区切り（囲みまたは終了（あるいはその両方））文字列を指定できます。デリミタ付きデータ・フィールドは、データ・フィールドの開始バイト位置が指定されている場合を除いて、直前のデータ・フィールドの終了位置から始まるとみなされます。
- バイト・オフセットまたはデータ・フィールドの長さ（あるいはその両方）を指定できます。この方法では、各フィールドは、直前のフィールドが終了した位置から、指定されたバイト数の位置で始まり、指定された長さの位置で終了します。
- Length-value データ型が使用できます。この場合、データ・フィールドの最初の *n* バイト数に、データ・フィールドの残りの長さについての情報が含まれています。

**参照：** 次の項を参照してください。

- 6-7 ページの「SQL\*Loader のデータ型」
- 6-24 ページの「デリミタの指定」

## LOBFILE および SDF

LOB データは、非常に長いデータであるため、LOBFILE からロードすると有効です。LOBFILE では、LOB データのインスタンスは、フィールド（事前に決められたサイズ、デリミタ付き、Length-Value）内にあるとみなされますが、これらのフィールドは、レコードに編成されていません（LOBFILE にはレコードの概念がありません）。そのため、レコードを扱うことによって発生する処理のオーバーヘッドを回避できます。このようなデータの編成方法は、LOB のロードにとって理想的です。

たとえば、従業員名、従業員 ID および従業員の履歴をロードする場合に LOBFILE を使用するとします。従業員名および従業員 ID をメイン・データ・ファイルから読み込み、非常に長い従業員の履歴を LOBFILE から読み込むことができます。

また、簡単に XML データをロードする場合に LOBFILE を使用するとします。XML 列を使用して、構造化データおよび半構造化データのモデルを保持できます。そのようなデータは、非常に長いデータです。

SDF とプライマリ・データ・ファイルの概念は類似しています。プライマリ・データ・ファイルと同様に、SDF は、レコードおよびフィールドで構成されたレコードの集まりです。SDF は、制御ファイルごとに指定されます。SDF をデータ・ソースとして命名できるのは、`collection_fld_spec` のみです。

SDF を指定するには、SDF パラメータを使用します。SDF パラメータの後に、ファイル指定文字列、または 1 つ以上のファイル指定文字列を含むデータ・フィールドにマップされた FILLER フィールドを指定します。

**参照：** 次の項を参照してください。

- 7-23 ページの「[LOBFILE からの LOB データのロード](#)」
- 7-31 ページの「[SDF](#)」

## データ変換およびデータ型仕様

従来型パス・ロード中に、データ・ファイルのデータ・フィールドが、データベースの列に変換されます（概念的にはダイレクト・パス・ロードと同様ですが、実装は異なります）。変換には、次の 2 つの手順があります。

1. SQL\*Loader で、制御ファイルにあるフィールド指定を使用してデータ・ファイルの形式を解析し、次に、入力データを解析します。そのデータを使用して SQL INSERT 文に対応するバインド配列を移入します。
2. Oracle データベース・サーバーがデータを受け取り、INSERT 文を実行してデータベースにデータを格納します。

Oracle データベース・サーバーでは、列のデータ型を使用して最終的な格納形式にデータを変換します。データ・ファイル内のフィールドとデータベース内の列の違いに注意する必要があります。また、SQL\*Loader 制御ファイルで定義されているフィールドのデータ型が、データベースの列のデータ型と同じではないことにも注意してください。

## 廃棄レコードおよび拒否レコード

入力ファイルから読み込まれたすべてのレコードが、データベースに挿入されるわけではありません。挿入されないレコードは、不良ファイルまたは廃棄ファイルに書き込まれます。

### 不良ファイル

不良ファイルには、SQL\*Loader または Oracle データベース・サーバーによって拒否レコードが書き込まれます。次に、その理由を示します。

## SQL\*Loader による拒否

入力形式が不適切なデータ・ファイル・レコードは、SQL\*Loader によって拒否されます。たとえば、2 番目の囲みデリミタがない場合や、デリミタ付きフィールドが最大長を超えている場合、レコードは、SQL\*Loader によって拒否されます。拒否レコードは、不良ファイルに書き込まれます。

## Oracle による拒否

データ・ファイル・レコードは、SQL\*Loader によって受け取られた後、Oracle データベース・サーバーに送られ、行として表に挿入されます。Oracle データベース・サーバーによって有効であると判断された行は、表に挿入されます。行が無効であると判断された場合、レコードは拒否され、不良ファイルに書き込まれます。行が無効であると判断される例としては、キーが重複している場合、必須入力フィールドに対応するデータが NULL 値の場合、またはフィールドに Oracle データ型ではないデータ型が指定された場合が考えられます。

**参照：** 次の項を参照してください。

- 5-11 ページの「[不良ファイルの指定](#)」
- 10-15 ページの「[事例 4: 結合された物理レコードのロード](#)」にある不良ファイルの使用例

## 廃棄ファイル

SQL\*Loader の実行によって、廃棄ファイルをコールするファイルが作成されることがあります。ファイルが作成されるのは必要な場合のみで、廃棄ファイルを使用可能にすることを指定してある場合にかぎります。廃棄ファイルには、制御ファイルに指定されているレコード選択基準に一致しなかったため、ロード対象から除外されたレコードが入ります。

したがって、廃棄ファイルには、データベースのどの表にも挿入されなかったレコードが格納されます。廃棄ファイルに格納可能なレコードの最大数を指定できます。レコードのデータがいずれかの表に書き込まれる場合、このレコードは廃棄ファイルには書き込まれません。

**参照：** 次の項を参照してください。

- 10-15 ページの「[事例 4: 結合された物理レコードのロード](#)」
- 5-14 ページの「[廃棄ファイルの指定](#)」

## ログ・ファイルおよびログ情報

SQL\*Loader で処理が開始されると、ログ・ファイルが作成されます。ログ・ファイルを作成できない場合、処理は終了します。このログ・ファイルにはロード中に発生したエラーに関する記述など、ロードに関する詳細情報が記録されます。

**参照：** 次の項を参照してください。

- [第 8 章「SQL\\*Loader ログ・ファイル・リファレンス」](#)
- [第 10 章「SQL\\*Loader の事例」](#) の「ログ・ファイルのサンプル」

## 従来型パス・ロード、ダイレクト・パス・ロードおよび外部表ロード

SQL\*Loader でデータをロードするには、次の方法があります。

- [従来型パス・ロード](#)
- [ダイレクト・パス・ロード](#)
- [外部表ロード](#)

### 従来型パス・ロード

従来型パス・ロードでは、入力レコードがフィールド指定を基に解析され、各データ・フィールドが対応するバインド配列にコピーされます。バインド配列がいっぱいになるか、または最終レコードが読み込まれた時点で配列の挿入が実行されます。

**参照：** 次の項を参照してください。

- [9-2 ページの「データのロード方法」](#)
- [5-43 ページの「バインド配列および従来型パス・ロード」](#)

SQL\*Loader では、バインドの配列に挿入後、LOB フィールドが格納されます。そのため、LOB フィールドの処理にエラーが発生した場合（たとえば、LOBFILE がないなど）、LOB フィールドは空のままになります。配列の挿入の実行後に LOB データがロードされるため、BEFORE および AFTER 行トリガーは LOB 列に対して機能しない場合もあります。これは、SQL\*Loader で列に LOB の内容をロードする機会を持つ前にトリガーが起動されるためです。たとえば、LOB 列 C1 にデータをロードして、BEFORE 行トリガーを使用してその LOB 列の内容を調べ、その結果を基に、他の列 C2 にロードされる値を導出するとします。これは、トリガーの起動時に LOB の内容がロードされていないため不可能です。

## ダイレクト・パス・ロード

ダイレクト・パス・ロードでは、入力レコードがフィールド指定を基に解析され、入力フィールド・データが列のデータ型に変換されて列配列が作成されます。この列配列は、Oracle データベース・ブロック形式でデータ・ブロックを作成するブロック・フォーマットに渡されます。新しくフォーマットされたデータベース・ブロックはデータベースに直接書き込まれるため、RDBMS による処理の大部分が省略されます。ダイレクト・パス・ロードによる処理は、従来型パス・ロードと比較すると非常に高速ですが、制限事項がいくつかあります。

**参照：** 9-5 ページの「[ダイレクト・パス・ロード](#)」を参照してください。

## パラレル・ダイレクト・パス

パラレル・ダイレクト・パス・ロードでは、複数のダイレクト・パス・ロード・セッションで同じデータ・セグメントを同時にロードできます（セグメント内の並列化が可能です）。パラレル・ダイレクト・パスには、ダイレクト・パスより多くの制約事項があります。

**参照：** 9-29 ページの「[パラレル・データ・ロード・モデル](#)」を参照してください。

## 外部表ロード

1 回の外部表ロードで、データ・ファイルのデータに 1 つの外部表が作成されます。INSERT 文を実行して、データ・ファイルのデータをターゲット表に挿入します。

従来型パス・ロードおよびダイレクト・パス・ロードではなく、外部表ロードを使用した場合のメリットは、次のとおりです。

- 外部表ロードでは、パラレルでデータ・ファイルをロードできます。データ・ファイルが大きい場合、パラレルでファイルをロードできます。
- 外部表ロードでは、外部表の作成に使用される INSERT 文の一部として SQL 関数および PL/SQL ファンクションを使用することによってロードされるデータを変更できます。

**参照：** 次の章を参照してください。

- [第 11 章「外部表の概要」](#)
- [第 12 章「外部表アクセス・パラメータ」](#)

## オブジェクト、コレクションおよび LOB のロード

オブジェクト、コレクションおよび LOB のバルクロードに SQL\*Loader を使用できます。オブジェクトの概念および Oracle のオブジェクト・サポートの実装の詳細は、『Oracle9i データベース概要』および『Oracle9i データベース管理者ガイド』を参照してください。



## サポートされるオブジェクト型

SQL\*Loader では、次の2つのオブジェクト型のロードがサポートされています。

### 列オブジェクト

表の列が、なんらかのオブジェクト型である場合、その列のオブジェクトは列オブジェクトと呼ばれます。概念的には、そのようなオブジェクトは、行の単一の列位置に全体が格納されます。これらのオブジェクトにはオブジェクト識別子がなく、参照することはできません。

列オブジェクトのオブジェクト型が NOT FINAL であると宣言されると、SQL\*Loader で導出された型（またはサブタイプ）を列オブジェクトにロードできます。

### 行オブジェクト

これらのオブジェクトはオブジェクト表と呼ばれる表に格納され、オブジェクト表にはオブジェクトの属性に対応する列があります。さらに、そのオブジェクト表にはシステムが生成する SYS\_NC\_OID\$ という列があり、その列に、表の各オブジェクトに対してシステムが生成する一意の識別子（OID）が格納されます。他の表の列は、これらのオブジェクトを OID を使用して参照できます。

列オブジェクトのオブジェクト型が NOT FINAL であると宣言されると、SQL\*Loader で導出された型（またはサブタイプ）を列オブジェクトにロードできます。

**参照：** 次の項を参照してください。

- 7-2 ページの「[列オブジェクトのロード](#)」
- 7-12 ページの「[オブジェクト表のロード](#)」

## サポートされるコレクション型

SQL\*Loader では、次の2つのコレクション型のロードがサポートされています。

### ネストした表

ネストした表は、別の表に列があるように見える表です。別の表に対して実行できるすべての操作は、ネストした表に対しても実行できます。

### VARRAY

VARRAY は、可変サイズの配列です。配列は、要素と呼ばれる、一連の組込み型またはオブジェクトの順序付けられた集合です。各配列の要素は同一の型であり、VARRAY 内の要素の位置に対応する一意の番号（index）を持ちます。

VARRAY 型の作成時に、最大数を指定する必要があります。VARRAY 型を宣言すると、リレーショナル表の列のデータ型、オブジェクト型属性、または PL/SQL 変数として使用することができます。

**参照：** SQL\*Loader 制御ファイルの DDL を使用してこれらのコレクション型をロードする方法の詳細は、7-29 ページの「[コレクション（ネストした表および VARRAY）のロード](#)」を参照してください。

## サポートされる LOB 型

LOB は、ラージ・オブジェクト型です。今回のリリースの SQL\*Loader では、4 つの LOB 型のロードをサポートしています。

- BLOB：構造化されていないバイナリ・データを含む LOB。
- CLOB：文字データを含む LOB。
- NCLOB：データベースの各国語キャラクタ・セットの文字を含む LOB。
- BFILE：データベースの表領域ではなく、サーバー側のオペレーティング・システム・ファイルに格納されている BLOB。

LOB は列データ型で、NCLOB 以外は、オブジェクトの属性データ型です。LOB は実際の値を持ち、その値は NULL でも「値なし（空）」でもかまいません。

**参照：** SQL\*Loader 制御ファイルの DDL を使用してこれらの LOB 型をロードする方法の詳細は、7-18 ページの「[LOB のロード](#)」を参照してください。

## パーティション・オブジェクトのサポート

SQL\*Loader では、データベース内のパーティション・オブジェクトのロードをサポートしています。Oracle データベースでは、グループ化されたパーティション（部分）で構成される表または索引が、パーティション・オブジェクトに相当します。一般に、パーティションは共通の論理属性によってグループ化されます。たとえば、2000 年度の売上データを、月別にパーティション化するとします。この場合、各月のデータは、売上表の中のそれぞれ別のパーティションに保存されます。このパーティションはそれぞれ、データベース内の異なるセグメントに保存されます。また、パーティションごとに異なる物理属性を指定できます。

次に、パーティション・オブジェクトがサポートされたことによって、SQL\*Loader でロードが可能になったものを示します。

- パーティション表中の個別パーティション
- パーティション表中の全パーティション
- 非パーティション表

## アプリケーション開発：ダイレクト・パス・ロード API

アプリケーション開発のために、ダイレクト・パス・ロード API が提供されています。詳細は、『Oracle Call Interface プログラマーズ・ガイド』を参照してください。



---

## SQL\*Loader コマンドライン・リファレンス

この章では、SQL\*Loader を起動するために使用するコマンドライン・パラメータについて説明します。この章の内容は、次のとおりです。

- [SQL\\*Loader の起動](#)
- [コマンドライン・パラメータ](#)
- [終了コードによる結果の検査と表示](#)

## SQL\*Loader の起動

SQL\*Loader を起動すると、セッションの特性を確立するために特定のパラメータを指定できます。パラメータは任意の順序で入力できます。オプションとして、カンマで区切ることもできます。パラメータに値を指定するか、場合によっては、値を入力しないでデフォルトを指定できます。

次に例を示します。

```
SQLLDR CONTROL=foo.ctl, LOG=bar.log, BAD=baz.bad, DATA=etc.dat
        USERID=scott/tiger, ERRORS=999, LOAD=2000, DISCARD=toss.dis,
        DISCARDMAX=5
```

任意のパラメータを指定しないで SQL\*Loader を起動すると、次のようなヘルプ画面が表示されます。使用可能なパラメータおよびそれらのデフォルト値が示されます。

```
sqlldr
```

```
...
```

```
SQL*Loader: Release 9.2.0.1.0 - Production on Wed Feb 27 12:06:17 2002
```

```
(c) Copyright 2002 Oracle Corporation. All rights reserved.
```

```
Usage: SQLLDR keyword=value [,keyword=value,...]
```

```
Valid Keywords:
```

```
    userid -- ORACLE username/password
    control -- Control file name
    log -- Log file name
    bad -- Bad file name
    data -- Data file name
    discard -- Discard file name
    discardmax -- Number of discards to allow (Default all)
    skip -- Number of logical records to skip (Default 0)
    load -- Number of logical records to load (Default all)
    errors -- Number of errors to allow (Default 50)
    rows -- Number of rows in conventional path bind array or between
direct path data saves
    (Default: Conventional path 64, Direct path all)
    bindsize -- Size of conventional path bind array in bytes (Default 256000)
    silent -- Suppress messages during run (header,feedback,errors,discards,partitions)
    direct -- use direct path (Default FALSE)
    parfile -- parameter file: name of file that contains parameter specifications
    parallel -- do parallel load (Default FALSE)
    file -- File to allocate extents from
skip_unusable_indexes -- disallow/allow unusable indexes or index partitions (Default FALSE)
skip_index_maintenance -- do not maintain indexes, mark affected indexes as unusable (Default FALSE)
```

```

readsize -- Size of Read buffer (Default 1048576)
external_table -- use external table for load; NOT_USED, GENERATE_ONLY, EXECUTE (Default NOT_USED)
columnarrayrows -- Number of rows for direct path column array (Default 5000)
streamsize -- Size of direct path stream buffer in bytes (Default 256000)
multithreading -- use multithreading in direct path
resumable -- enable or disable resumable for current session (Default FALSE)
resumable_name -- text string to help identify resumable statement
resumable_timeout -- wait time (in seconds) for RESUMABLE (Default 7200)
date_cache -- size (in entries) of date conversion cache (Default 1000)

```

PLEASE NOTE: Command-line parameters may be specified either by position or by keywords.

An example of the former case is 'sqlldr scott/tiger foo'; an example of the latter is 'sqlldr control=foo userid=scott/tiger'. One may specify parameters by position before but not after parameters specified by keywords. For example, 'sqlldr scott/tiger control=foo logfile=log' is allowed, but 'sqlldr scott/tiger control=foo log' is not, even though the position of the parameter 'log' is correct.

---

**注意：** SQL\*Loader を起動するコマンドは、オペレーティング・システムによって異なります。この章の例では、UNIX での名前 `sqlldr` を使用しています。ご使用のシステムの正しいコマンドの詳細は、オペレーティング・システム固有の Oracle マニュアルを参照してください。

---

**参照：** すべてのコマンドライン・パラメータの詳細は、4-4 ページの「[コマンドライン・パラメータ](#)」を参照してください。

## 制御ファイルでのパラメータの指定

コマンドラインの長さが、ご使用のシステムの許容最大長を超える場合は、コマンドライン・パラメータを制御ファイルで指定できます。制御ファイルでのパラメータの指定方法の詳細は、5-4 ページの「[OPTIONS 句](#)」を参照してください。

PARFILE パラメータで指定した別のファイルにも指定できます。値がほとんど変更されないパラメータを指定する場合は、前述のいずれかの方法で指定してください。このようにして指定されたパラメータは、コマンドラインから別の指定を行うことによって、取り消すことができます。

**参照：**

- SQL\*Loader 制御ファイルの詳細は、[第 5 章](#)を参照してください。
- 4-10 ページの「[PARFILE \(パラメータ・ファイル\)](#)」を参照してください。

## コマンドライン・パラメータ

この項では、SQL\*Loader の各コマンドライン・パラメータについて説明します。ここで使用されているコマンドライン・パラメータのデフォルト値および最大値は、UNIX ベースのシステムでの値です。これらの値はオペレーティング・システムによって異なります。詳細は、ご使用のオペレーティング・システム固有の Oracle マニュアルを参照してください。

### BAD（不良ファイル）

デフォルト：拡張子が .bad のデータ・ファイル名

SQL\*Loader によって作成される不良ファイルの名前を指定します。このファイルには、挿入中にエラーの原因となったレコード、または形式が不適切なレコードが格納されます。ファイル名を指定しない場合、デフォルトが使用されます。

コマンドラインで指定された不良ファイル名は、制御ファイルの最初の INFILE 文に関連する不良ファイル名になります。つまり、制御ファイル中で指定した不良ファイル名よりも、コマンドラインで指定した不良ファイル名の方が優先されます。

**参照：** 不良ファイルの形式の詳細は、5-11 ページの「[不良ファイルの指定](#)」を参照してください。

### BINDSIZE（最大サイズ）

デフォルト：このパラメータのデフォルト値を参照するには、4-2 ページの「[SQL\\*Loader の起動](#)」で説明したように、任意のパラメータを指定しないで SQL\*Loader を起動します。

バインド配列の最大サイズ（バイト単位）を指定します。BINDSIZE で指定されたバインド配列サイズは、デフォルト・サイズ（システムによって異なる）および ROWS に基づいて計算されたサイズよりも優先されます。

**参照：** 次の項を参照してください。

- 5-43 ページの「[バインド配列および従来型パス・ロード](#)」
- 4-11 ページの「[READSIZE（読み込みバッファ・サイズ）](#)」

### COLUMNARRAYROWS

デフォルト：このパラメータのデフォルト値を参照するには、4-2 ページの「[SQL\\*Loader の起動](#)」で説明したように、任意のパラメータを指定しないで SQL\*Loader を起動します。

行数を指定してダイレクト・パス配列に割り当てます。このパラメータの値は、SQL\*Loader では計算されません。値を指定するか、またはデフォルトを使用する必要があります。



**参照：** 次の項を参照してください。

- 5-26 ページの「[CONCATENATE を使用した論理レコードの作成](#)」
- 9-21 ページの「[列配列の行数およびストリーム・バッファ・サイズの指定](#)」

## CONTROL（制御ファイル）

デフォルト：なし

データのロード方法を記述する制御ファイルの名前を指定します。ファイルの拡張子またはファイル・タイプを指定していない場合は、デフォルトで .ctl になります。省略すると、SQL\*Loader からファイル名の入力要求されます。

SQL\*Loader 制御ファイル名に特殊文字が含まれている場合、オペレーティング・システムによっては、その文字をエスケープする必要があります。また、オペレーティング・システム上でファイル・システム・パスの中にバックスラッシュが使用されている場合は、複数のエスケープ文字を使用するか、または引用符でパスを囲む必要があります。詳細は、ご使用のオペレーティング・システム固有の Oracle マニュアルを参照してください。

**参照：** SQL\*Loader 制御ファイルの詳細は、[第 5 章](#)を参照してください。

## DATA（データ・ファイル）

デフォルト：拡張子が .dat の制御ファイル名

ロードするデータが入っているデータ・ファイルの名前を指定します。ファイルの拡張子またはファイル・タイプを指定していない場合は、デフォルトで .dat になります。

コマンドラインでデータ・ファイルを指定し、INFILE で制御ファイル内のデータ・ファイルを指定する場合は、コマンドラインで指定されたデータが最初に処理されます。制御ファイル内で指定された最初のデータ・ファイルは無視されます。制御ファイル内で指定された他のすべてのデータ・ファイルは処理されます。

ファイル処理オプションを指定する場合は、制御ファイルからのデータのロード時に、警告メッセージが発行されます。

## DATE\_CACHE

デフォルト：使用可能（1000 要素）。日付キャッシュ機能を完全に使用禁止にするには、0（ゼロ）に設定します。

日付キャッシュ・サイズ（エントリ数）を指定します。たとえば、DATE\_CACHE=5000 を指定すると、作成された日付キャッシュごとに最大 5000 の一意の日付エントリを含めることができます。必要に応じて、すべての表に固有の日付キャッシュが作成されます。日付キャッシュは、表に格納するためにデータ型変換が必要な日付値またはタイムスタンプ値が 1 つ以上ロードされた場合にのみ作成されます。

日付キャッシュ機能は、ダイレクト・パス・ロードでのみ使用できます。デフォルトでは、この機能が使用可能です。デフォルトの日付キャッシュ・サイズは 1000 要素です。デフォルトのサイズを使用し、1000 を超える一意の入力値がロードされると、日付キャッシュ機能はこの表に対して自動的に使用禁止となります。ただし、デフォルトを変更して 0（ゼロ）以外の日付キャッシュ・サイズを指定し、キャッシュ量がこのサイズを超えた場合、キャッシュは使用禁止になりません。

ログ・ファイルに含まれる日付キャッシュ統計（エントリ数、ヒット数、ミス数）を使用して、将来同様のロードを行う場合に備えてキャッシュのサイズを調整できます。

**参照：** 9-21 ページの「[日付キャッシュの値の指定](#)」を参照してください。

## DIRECT（データ・パス）

デフォルト : false

従来型パスまたはダイレクト・パスのどちらの方法でデータをロードするかを指定します。true 値はダイレクト・パス・ロードを指定します。false 値は従来型パス・ロードを指定します。

**参照：** 第 9 章「[従来型パス・ロードおよびダイレクト・パス・ロード](#)」を参照してください。

## DISCARD（ファイル名）

デフォルト : 拡張子が .bsc のデータ・ファイル名

SQL\*Loader で作成する廃棄ファイル（オプション）を指定します。このファイルには、表に挿入されず、拒否もされなかったレコードが保存されます。

コマンドラインで指定された廃棄ファイル名は、制御ファイルの最初の INFILE 文に関連する廃棄ファイル名となります。つまり、制御ファイル中で指定する廃棄ファイル名よりも、コマンドラインで指定した廃棄ファイル名の方が優先されます。

**参照：** 廃棄ファイルの形式の詳細は、3-9 ページの「[廃棄レコードおよび拒否レコード](#)」を参照してください。

## DISCARDMAX（整数）

デフォルト : ALL

廃棄レコードの最大数を指定します。廃棄レコード件数がここで指定した数に達すると、ロードは中止されます。最初の廃棄レコードで中止するには、1 を指定します。

## ERRORS（エラーの許容最大数）

デフォルト：このパラメータのデフォルト値を参照するには、4-2 ページの「[SQL\\*Loader の起動](#)」で説明したように、任意のパラメータを指定しないで SQL\*Loader を起動します。

挿入エラーの許容最大数を指定します。発生したエラーの数が ERRORS に指定された値を超えると、ロード処理が中止されます。エラーを許容しない場合は、ERRORS=0 を設定します。エラーを無制限に許容する場合は、非常に大きい値を指定します。

単一の表をロードする場合、エラーの数がこの上限を超えると、ロード処理が中止されます。ただし、その前に挿入されたデータはコミットされます。

SQL\*Loader では、すべての表の間でレコードの整合性を保つように処理されます。つまり、複数の表をロードする場合は、エラーの数がこの上限を超えても、すぐにはロード処理が中止されません。SQL\*Loader では、複数の表のロードに対して許容最大数のエラーが検出されても、行のロードは続行され、すでに表にロードされた有効な行は確実にすべての表にロードされます。また、拒否された行はすべての表から削除されます。

どのような場合でも、SQL\*Loader では、エラーになったレコードが不良ファイルに書き込まれます。

## EXTERNAL\_TABLE

デフォルト：NOT\_USED

外部表オプションを使用してデータをロードするかどうかを SQL\*Loader に指定します。EXTERNAL\_TABLE には、次の 3 つの値を指定できます。

- NOT\_USED: デフォルト値。従来型パス・モードまたはダイレクト・パス・モードのいずれかを使用して、ロードを行います。
- GENERATE\_ONLY: 制御ファイルに記述されているとおり、SQL\*Loader ログ・ファイル内の外部表を使用してロードを行うために必要なすべての SQL 文を書き込みます。これらの SQL 文は、編集およびカスタマイズできます。実際のロードは、SQL\*Loader を使用せずに、SQL\*Plus でこれらの文を実行して、後で行うことができます。この SQL\*Loader ログ・ファイルの例は、8-8 ページの「[EXTERNAL\\_TABLE=GENERATE\\_ONLY の使用時に作成されるログ・ファイル](#)」を参照してください。
- EXECUTE: 外部表を使用してロードを行うために必要な SQL 文を実行します。ただし、SQL 文からエラーが返されると、ロードは停止します。SQL 文は、実行されたとおりログ・ファイルに書き込まれます。つまり、SQL 文からエラーが返されると、ロードに必要な残りの SQL 文は制御ファイルには書き込まれません。

外部表オプションで、データベース内のディレクトリ・オブジェクトを使用して、すべてのデータ・ファイルがどこに格納されるか、および出力ファイルがどこで作成されるかを指定します。データ・ファイルを含むディレクトリ・オブジェクトには、READ アクセスが、出力ファイルが作成されるディレクトリ・オブジェクトには、WRITE アクセスが必要です。データ・ファイルまたは出力ファイル格納用の既存のディレクトリ・オブジェクトがない場

合は、SQL\*Loader で SQL 文を生成して作成します。EXECUTE オプションを指定する場合は、CREATE ANY DIRECTORY 権限が必要です。

---

**注意：** SQL\*Loader で EXTERNAL\_TABLE=EXECUTE 修飾子を指定すると、データのロードに使用可能な外部表が作成された後、INSERT 文が実行され、データがロードされます。外部表のすべてのファイルがディレクトリ・オブジェクト内に存在すると識別される必要があります。

SQL\*Loader では、アクセス権限を所有している既存のディレクトリ・オブジェクトの使用がサポートされています。ただし、一致するディレクトリ・オブジェクトは検出されません。一致するディレクトリ・オブジェクトが検出されないため、SQL\*Loader で一時ディレクトリ・オブジェクトの作成が試行されます。新しいディレクトリ・オブジェクトを作成する権限を所有していない場合、この操作は正常に実行されません。

この問題を解決するには、EXTERNAL\_TABLE=GENERATE\_ONLY を使用して、SQL\*Loader で実行が試行される SQL 文を作成します。これらの SQL 文を抽出し、参照するディレクトリ・オブジェクトを、アクセス権限を所有しているディレクトリ・オブジェクトに変更します。その後、これらの SQL 文を実行します。

---

複数表のロードを行う場合は、SQL\*Loader で次の手順を実行します。

1. 任意の表にロードされるデータ・ファイルのすべてのフィールドを記述する表を、データベースに作成します。
2. INSERT 文を作成して、データの外部表記述からこの表をロードします。
3. 制御ファイルのすべての表に INSERT 文を実行します。

実行例については、事例 5 (10-19 ページの「[事例 5: 複数表へのデータのロード](#)」) に EXTERNAL\_TABLE=GENERATE\_ONLY パラメータを追加して実行します。外部表に一意の名前を保証するため、SQL\*Loader ではすべてのフィールド用に生成された名前を使用します。これは、フィールド名が制御ファイル内の異なる表の間で一意でないことがあるためです。

**参照：** 次の章を参照してください。

- [第 11 章「外部表の概要」](#)
- [第 12 章「外部表アクセス・パラメータ」](#)

## EXTERNAL\_TABLE の使用上の制限

EXTERNAL\_TABLE 修飾子を使用する場合、次の制限が適用されます。

- SQL\*Loader を介して外部表からデータベース表にデータを挿入する場合、ユリウス日は使用できません。この問題を解決するには、次の例に示すとおり、TO\_DATE および TO\_CHAR を使用してユリウス日書式を変換します。  

```
TO_CHAR(TO_DATE(:COL1, 'MM-DD-YYYY'), 'J')
```
- 外部表からデータベース表にデータを挿入する場合、オブジェクト要素には組み関数および SQL 文字列は使用できません。

## FILE (ロード先ファイル)

デフォルト: なし

エクステントを割り当てるデータベース・ファイルを指定します。このパラメータは、パラレル・ロードでのみ使用します。SQL\*Loader の異なるプロセスに対する FILE パラメータの値を変えることによって、データがシステムにロードされときのディスクの競合を最小限に抑えることができます。

**参照:** 9-29 ページの「[パラレル・データ・ロード・モデル](#)」を参照してください。

## LOAD (ロードするレコード)

デフォルト: すべてのレコードがロードされます。

指定した件数のレコードをスキップした後に、ロードする論理レコード件数の最大数を指定します。実際のレコード件数が指定された最大数より少ない場合、エラーは発生しません。

## LOG (ログ・ファイル)

デフォルト: 拡張子が .log の制御ファイル名

SQL\*Loader によって作成されるログ・ファイルを指定します。このファイルには、ロード処理に関するログ情報が保存されます。

## MULTITHREADING

デフォルト: 複数 CPU システムでは true で、単一 CPU システムでは false。

このパラメータは、ダイレクト・パス・ロードでのみ使用できます。

デフォルトでは、マルチスレッド・オプションは常に true に設定され、複数 CPU システム上で使用可能です。この場合の複数 CPU システムの定義は、2 つ以上の CPU を持つ単一システムです。

単一 CPU システムでは、マルチスレッドはデフォルトで `false` に設定されています。2 つの単一 CPU システム間でマルチスレッドを使用するには、マルチスレッドを使用可能にする必要があります。デフォルトでは、使用可能になっていません。マルチスレッドを使用可能にすることによって、サーバー・システムでのストリームのロードと並行して、クライアント・システムでストリームを作成できます。

マルチスレッド機能は、オペレーティング・システムに依存します。すべてのオペレーティング・システムがマルチスレッドをサポートしているわけではありません。

**参照：** 9-23 ページの「[複数 CPU システムのダイレクト・パス・ロードの最適化](#)」を参照してください。

## PARALLEL (パラレル・ロード)

デフォルト: `false`

ダイレクト・ロード時に複数の同時セッションによって同じ表にデータをロードできるかどうかを指定します。

**参照：** 9-29 ページの「[パラレル・データ・ロード・モデル](#)」を参照してください。

## PARFILE (パラメータ・ファイル)

デフォルト: なし

コマンドラインで頻繁に使用するパラメータを記述したファイルを指定します。たとえば、コマンドラインで次のように指定します。

```
sqlldr PARFILE=example.par
```

パラメータ・ファイルには次のような内容を記述できます。

```
USERID=scott/tiger
CONTROL=example.ctl
ERRORS=9999
LOG=example.log
```

---

---

**注意：** 通常は問題ありませんが、システムによっては、パラメータ指定の中で等号 (=) の前後に空白を挿入できないものもあります。

---

---

## READSIZE (読み込みバッファ・サイズ)

デフォルト: このパラメータのデフォルト値を参照するには、4-2 ページの「[SQL\\*Loader の起動](#)」で説明したように、任意のパラメータを指定しないで SQL\*Loader を起動します。

READSIZE パラメータは、データ・ファイルからデータを読み込む場合にのみ使用されます。制御ファイルからレコードを読み込む場合は、常に 64KB という値が READSIZE として使用されます。

デフォルトを使用しない場合は、READSIZE パラメータを使用して読み込みバッファのサイズ (バイト単位) を指定できます。指定可能な最大サイズは、ダイレクト・パス・ロードおよび従来型パス・ロードの両方で 20MB です。

従来型パスの方法では、バインド配列は、読み込みバッファのサイズによって制限されます。そのため、読み込みバッファを大きくすると、コミットを実行する前に、より多くのデータを読み込むことができるという利点があります。

次に例を示します。

```
sqlldr scott/tiger CONTROL=ulcas1.ctl READSIZE=1000000
```

この場合、コミットを実行する前に、SQL\*Loader によって、外部のデータ・ファイルから 1,000,000 バイト単位で読み込みを実行できます。

---

---

**注意:** READSIZE に BINDSIZE より小さい値を指定した場合、READSIZE の値は増加します。

---

---

READSIZE パラメータは、LOB に影響しません。LOB 読み込みバッファのサイズは 64KB に固定されています。

詳細は、4-4 ページの「[BINDSIZE \(最大サイズ\)](#)」を参照してください。

## RESUMABLE

デフォルト: false

再開可能な領域割当てを有効または無効にします。このパラメータはデフォルトでは無効なため、関連する RESUMABLE\_NAME パラメータおよび RESUMABLE\_TIMEOUT パラメータを使用するには、RESUMABLE=true を設定する必要があります。

**参照:** 次のマニュアルを参照してください。

- 『Oracle9i データベース概要』
- 『Oracle9i データベース管理者ガイド』

## RESUMABLE\_NAME

デフォルト: 'User USERNAME (USERID), Session SESSIONID, Instance INSTANCEID'

再開可能な文を指定します。この値はユーザー定義のテキスト文字列で、USER\_RESUMABLE または DBA\_RESUMABLE ビューに挿入して、一時停止されている特定の再開可能な文を識別できます。

RESUMABLE パラメータを true に設定して、再開可能な領域割当てを有効にしないかぎり、このパラメータは無視されます。

## RESUMABLE\_TIMEOUT

デフォルト: 7200 秒 (2 時間)

エラー修正に必要な時間を指定します。タイムアウト時間内にエラーを修正できない場合は、文の実行が強制終了されます。

RESUMABLE パラメータを true に設定して、再開可能な領域割当てを有効にしないかぎり、このパラメータは無視されます。

## ROWS (1 回にコミットする行数)

デフォルト: このパラメータのデフォルト値を参照するには、4-2 ページの「[SQL\\*Loader の起動](#)」で説明したように、任意のパラメータを指定しないで SQL\*Loader を起動します。

従来型パス・ロードの場合: ROWS でバインド配列の行数を指定します。詳細は、5-43 ページの「[バインド配列および従来型パス・ロード](#)」を参照してください。

ダイレクト・パス・ロードの場合: ROWS でデータ・ファイルからデータのセーブ前に読み込む行数を指定します。デフォルトでは、ロード終了時に、すべての行の読み込みおよびデータ・セーブが 1 回実行されます。詳細は、9-13 ページの「[データ・セーブを使用したデータ損失の防止](#)」を参照してください。

ダイレクト・ロードでは、パフォーマンスを最適化するために、システム I/O ブロックと同じサイズで同じ形式のバッファを使用します。このバッファが一杯になった時点で、データがデータベースに書き込まれるため、指定した ROWS の値は目安として使用されます。



## SILENT（フィードバック・モード）

SQL\*Loader を開始すると、次のようなヘッダー・メッセージが画面に表示され、ログ・ファイルに書き込まれます。

```
SQL*Loader: Release 9.2.0.1.0 - Production on Wed Feb 27 14:33:54 2002
```

```
(c) Copyright 2002 Oracle Corporation. All rights reserved.
```

また、SQL\*Loader 実行中には次の例のようなフィードバック・メッセージも画面に表示されます。

```
Commit point reached - logical record count 20
```

SQL\*Loader で次のようなデータ・エラー・メッセージが表示される場合もあります。

```
Record 4: Rejected - Error on table EMP
```

```
ORA-00001: 一意制約 <name> に反しています
```

1 つ以上の値を持つ SILENT を指定して、これらのメッセージを抑止できます。

たとえば、画面に通常表示されるヘッダーとフィードバック・メッセージが表示されないようにするには、コマンドラインの引数で次のように指定します。

```
SILENT=(HEADER, FEEDBACK)
```

適切な値を使用して、次のいずれかの処理を抑止します（複数も可）。

- **HEADER** - 画面に通常表示される SQL\*Loader のヘッダー・メッセージを非表示にします。ただし、ヘッダー・メッセージはログ・ファイルに出力されます。
- **FEEDBACK** - 画面に通常表示される「commit point reached」フィードバック・メッセージを非表示にします。
- **ERRORS** - レコードに Oracle エラーが発生したためそのレコードが不良ファイルに書き込まれた場合、データ・エラー・メッセージがログ・ファイルに出力されないようにします。ただし、拒否レコード件数は出力されます。
- **DISCARDS** - レコードが廃棄ファイルに書き込まれた場合に、そのことを示すメッセージがログ・ファイルに出力されないようにします。
- **PARTITIONS** - パーティション表のダイレクト・ロード中、ログ・ファイルに対するパーティションごとの統計情報の書込みを使用禁止にします。
- **ALL** - すべての抑止値（HEADER、FEEDBACK、ERRORS、DISCARDS および PARTITIONS パラメータ）を実装します。

## SKIP（スキップされるレコード）

デフォルト：レコードは1件もスキップされません。

ファイルの先頭から何件の論理レコードをロード対象外とするかを指定します。

SKIP を指定すると、なんらかの理由で中断されたロードが継続されます。このパラメータは、従来型ロードおよび単一表のダイレクト・ロードで使用できます。複数の表のダイレクト・ロードに関しては、各表に対して同じ件数のレコードをロードする場合のみ使用できません。複数の表にそれぞれ異なる件数のレコードをダイレクト・ロードする場合は、使用できません。

**参照：** 5-23 ページの「[中断されたロード](#)」を参照してください。

## SKIP\_INDEX\_MAINTENANCE

デフォルト：false

ダイレクト・パス・ロードの索引メンテナンスを停止します。これは、従来型パス・ロードには適用できません。このオプションを指定すると、索引キーが付けられた索引パーティションには、索引キーのかわりに索引使用禁止が設定されます。これは、索引セグメントと、その索引が付けられているデータとの整合性がとれないためです。ロードの影響を受けない索引セグメントについては、ロード前の索引使用禁止状態が保持されます。

SKIP\_INDEX\_MAINTENANCE パラメータ：

- ローカル索引とグローバル索引の両方に適用できます。
- 索引を持つオブジェクトの平行ロードを実行できます（PARALLEL パラメータとともに指定）。
- グローバル索引を持つ表に単一パーティションをロードできます（INTO TABLE 句で PARTITION パラメータを指定）。
- ロードによって索引使用禁止状態に設定された索引や索引パーティションのリストを（SQL\*Loader ログ・ファイルに）作成します。

## SKIP\_UNUSABLE\_INDEXES

デフォルト：false

従来型パス・ロードおよびダイレクト・パス・ロードの両方に適用できます。

SKIP\_UNUSABLE\_INDEXES=true を指定すると、表のロードとともに、ロード開始前の状態が索引使用禁止（IU）である索引もロードされます。ロード時に IU 状態でない索引は、SQL\*Loader によってメンテナンスされます。一方、ロード時に IU 状態の索引はメンテナンスの対象にはならず、ロードが完了しても状態は IU のままです。

ただし、一意で IU 状態の索引に対しては、索引メンテナンスをスキップできません。この原則は、DML 操作の場合にも、ダイレクト・パスで DML と整合性を持つロードを行う場合にも適用されます。

SKIP\_UNUSABLE\_INDEXES=false を指定してロードを実行すると、従来型パス・ロードとダイレクト・パス・ロードとは、次のように処理内容が異なります。

- 従来型パス・ロードでは、挿入時に索引の更新が必要なレコードが存在する場合、そのレコードは挿入されずに拒否されます。
- ダイレクト・パス・ロードでは、使用不可 (unusable) 状態の索引に対して索引メンテナンスが必要なレコードが検出された時点で、ロード処理は終了します。

## STREAMSIZE

デフォルト: このパラメータのデフォルト値を参照するには、4-2 ページの「SQL\*Loader の起動」で説明したように、任意のパラメータを指定しないで SQL\*Loader を起動します。

ダイレクト・パス・ストリームに対して、サイズをバイト単位で指定します。

**参照:** 9-21 ページの「列配列の行数およびストリーム・バッファ・サイズの指定」を参照してください。

## USERID (ユーザー名 / パスワード)

デフォルト: なし

各ユーザーの Oracle ユーザー名 / パスワードを指定します。省略すると、システムから入力を要求されます。スラッシュのみを入力すると、デフォルトとしてオペレーティング・システムのログイン名が USERID に適用されます。

また、ユーザー SYS として接続する場合は、接続文字列に AS SYSDBA を指定する必要があります。次に例を示します。

```
sqlldr \SYS/password AS SYSDBA\ foo.ctl
```

**注意：** この例では、接続文字列全体が一重引用符およびバックスラッシュで囲まれています。これは、文字列 AS SYSDBA に空白が含まれるため、ほとんどのオペレーティング・システムで、文字列全体を一重引用符で囲むか、なんらかの方法でリテラルとしてマークする必要があるためです。オペレーティング・システムによっては、コマンドラインの一重引用符自体をエスケープする必要がある場合もあります。この例では、バックスラッシュがエスケープ文字として使用されます。バックスラッシュがない場合、SQL\*Loader で使用されるコマンドライン解析機能での一重引用符として認識されるため、一重引用符は SQL\*Loader を起動する前に削除されてしまいます。

システムの特許文字および予約文字の詳細は、ご使用のオペレーティング・システム固有の Oracle マニュアルを参照してください。

## 終了コードによる結果の検査と表示

Oracle SQL\*Loader では、SQL\*Loader の完了後、すぐに実行結果を確認できます。プラットフォームによっては、SQL\*Loader の実行結果はログ・ファイルに記録されるのみでなく、プロセス終了コードにも通知されます。この Oracle SQL\*Loader の機能によって、コマンドラインやスクリプトから SQL\*Loader を起動したときにもその結果を確認できます。表 4-1 に、それぞれの結果に対応する終了コードを示します。

表 4-1 SQL\*Loader の終了コード

結果	終了コード
すべての列が正常にロードされた	EX_SUCC
すべての行または一部の行が拒否された	EX_WARN
すべての行または一部の行が拒否された	EX_WARN
ロードが中断された	EX_WARN
コマンドラインまたは構文エラー	EX_FAIL
SQL*Loader に対してリカバリ不能な Oracle エラー	EX_FAIL
OS 関連エラー（ファイルのオープン / クローズ、malloc など）	EX_FAIL

UNIX の場合、終了コードは次のようになります。

EX\_SUCC 0  
EX\_FAIL 1  
EX\_WARN 2  
EX\_FTL 3

Windows NT の場合、終了コードは次のようになります。

```
EX_SUCC 0
EX_WARN 2
EX_FAIL 3
EX_FTL  4
```

SQL\*Loader が 0（ゼロ）以外の終了コードを返した場合、システム・ログ・ファイルおよび SQL\*Loader ログ・ファイルで、詳細な診断情報を確認してください。

UNIX では、シェルの終了コードを調べてロード結果を確認できます。たとえば、SQL\*Loader コマンドをスクリプトで使用する、スクリプト内で終了コードを確認できます。

```
#!/bin/sh
sqlldr scott/tiger control=ulcase1.ctl log=ulcase1.log
retcode=`echo $?`
case "$retcode" in
0) echo "SQL*Loader execution successful" ;;
1) echo "SQL*Loader execution exited with EX_FAIL, see logfile" ;;
2) echo "SQL*Loader execution exited with EX_WARN, see logfile" ;;
3) echo "SQL*Loader execution encountered a fatal error" ;;
*) echo "unknown return code" ;;
esac
```



---

## SQL\*Loader 制御ファイル・リファレンス

この章では、SQL\*Loader 制御ファイルについて説明します。この章の内容は、次のとおりです。

- 制御ファイルの内容
- 制御ファイル中でのコマンドライン・パラメータの指定
- ファイル名およびオブジェクト名の指定
- データ・ファイルの指定
- BEGINDATA による制御ファイルのデータの識別
- データ・ファイル形式およびバッファリングの指定
- 不良ファイルの指定
- 廃棄ファイルの指定
- 異なる文字コード体系の処理
- 中断されたロード
- 物理レコードからの論理レコードの作成
- 表への論理レコードのロード
- 索引オプション
- 複数の INTO TABLE 句を使用するメリット
- バインド配列および従来型パス・ロード

## 制御ファイルの内容

SQL\*Loader 制御ファイルは、DDL 命令を含むテキスト・ファイルです。DDL を使用して、SQL\*Loader セッションの次の項目を制御します。

- SQL\*Loader によってロードされるデータの位置
- SQL\*Loader によるデータ書式設定の方法
- データのロード時の SQL\*Loader の設定（メモリー管理、拒否レコード、ロード処理の中断など）
- SQL\*Loader によるロード中のデータの処理方法

SQL\*Loader の DDL 構文図については、[付録 A](#) を参照してください。

SQL\*Loader 制御ファイルを作成するには、vi や xemacs.create などのテキスト・エディタを使用します。

通常、制御ファイルには、次の 3 つの項目が次の順序で含まれます。

- セッション全体の情報
- 表およびフィールド・リストの情報
- 入力データ（オプションの項目）

[例 5-1](#) にサンプル制御ファイルを示します。

### 例 5-1 サンプル制御ファイル

```
1  -- This is a sample control file
2  LOAD DATA
3  INFILE 'sample.dat'
4  BADFILE 'sample.bad'
5  DISCARDFILE 'sample.dsc'
6  APPEND
7  INTO TABLE emp
8  WHEN (57) = '.'
9  TRAILING NULLCOLS
10 (hiredate SYSDATE,
    deptno POSITION(1:2)  INTEGER EXTERNAL(2)
        NULLIF deptno=BLANKS,
    job      POSITION(7:14)  CHAR  TERMINATED BY WHITESPACE
        NULLIF job=BLANKS  "UPPER(:job)",
    mgr      POSITION(28:31) INTEGER EXTERNAL
        TERMINATED BY WHITESPACE, NULLIF mgr=BLANKS,
    ename     POSITION(34:41) CHAR
        TERMINATED BY WHITESPACE  "UPPER(:ename)",
    empno     POSITION(45)  INTEGER EXTERNAL
        TERMINATED BY WHITESPACE,
```



```

sal    POSITION(51) CHAR  TERMINATED BY WHITESPACE
      "TO_NUMBER(:sal, '$99,999.99')",
comm   INTEGER EXTERNAL ENCLOSED BY '(' AND '%'
      ":comm * 100"
)

```

このサンプル制御ファイルの左側の数字は、実際の制御ファイルでは表示されません。これらの数字は、次の説明の番号に対応しています。

1. 制御ファイルにコメントを入力します。詳細は、5-4 ページの「[制御ファイルのコメント](#)」を参照してください。
2. LOAD DATA 文によって、SQL\*Loader で新しくデータ・ロードが開始されます。構文の詳細は、[付録 A](#) を参照してください。
3. INFILE には、ロードするデータが入っているデータ・ファイルの名前を指定します。詳細は、5-7 ページの「[データ・ファイルの指定](#)」を参照してください。
4. BADFILE パラメータには、拒否レコードが書き込まれるファイルの名前を指定します。詳細は、5-11 ページの「[不良ファイルの指定](#)」を参照してください。
5. DISCARDFILE パラメータには、廃棄レコードが書き込まれるファイルの名前を指定します。詳細は、5-14 ページの「[廃棄ファイルの指定](#)」を参照してください。
6. APPEND パラメータは、空ではない表にデータをロードするときに使用できるオプションの 1 つです。詳細は、5-32 ページの「[空でない表へのデータのロード](#)」を参照してください。

空の表にデータをロードするには、INSERT パラメータを使用します。詳細は、5-32 ページの「[空の表へのデータのロード](#)」を参照してください。

7. INTO TABLE 句を使用して、表、フィールドおよびデータ型を識別できます。この句で、データ・ファイル中のレコードとデータベース中の表の関係を定義します。詳細は、5-31 ページの「[表名の指定](#)」を参照してください。
8. WHEN 句には、1 つ以上のフィールド条件を指定します。SQL\*Loader で、この条件に基づいてデータをロードするかどうかを判断します。詳細は、5-34 ページの「[条件に基づいたレコードのロード](#)」を参照してください。
9. TRAILING NULLCOLS 句を使用すると、相対位置に指定した列がレコード中に存在しない場合、その列の値は NULL として処理されます。詳細は、5-36 ページの「[データが欠落しているショート・レコードの処理](#)」を参照してください。
10. 制御ファイルの残りの部分には、ロード中の表の列形式の詳細を参照できるフィールド・リストが含まれます。制御ファイルのこの部分の詳細は、[第 6 章](#)を参照してください。

### 制御ファイルのコメント

コメントはファイル中のコマンド部分のどこにでも記述できますが、データの部分には記述できません。コメントの前にハイフンを2つ続けて記述します。次に例を示します。

```
--This is a comment
```

二重ハイフンの右側のすべてのテキストは行末まで無視されます。制御ファイルのコメントの例は、10-12 ページの「[事例 3: 自由区分形式ファイルのロード](#)」参照してください。

### 制御ファイル中でのコマンドライン・パラメータの指定

OPTIONS 句は、制御ファイルを、通常使用するオプションの組合せで起動する場合に使用します。OPTIONS 句は、LOADDATA 文の前に置きます。

### OPTIONS 句

OPTIONS 句を使用すると、実行時のパラメータをコマンドラインではなく制御ファイル中で指定することができます。OPTIONS 句では、次のパラメータを指定できます。これらのパラメータの詳細は、[第 4 章](#)を参照してください。

```
BINDSIZE = n
COLUMNARRAYROWS = n
DIRECT = {TRUE | FALSE}
ERRORS = n
LOAD = n
MULTITHREADING = {TRUE | FALSE}
PARALLEL = {TRUE | FALSE}
READSIZE = n
RESUMABLE = {TRUE | FALSE}
RESUMABLE_NAME = 'text string'
RESUMABLE_TIMEOUT = n
ROWS = n
SILENT = {HEADERS | FEEDBACK | ERRORS | DISCARDS | PARTITIONS | ALL}
SKIP = n
SKIP_INDEX_MAINTENANCE = {TRUE | FALSE}
SKIP_UNUSABLE_INDEXES = {TRUE | FALSE}
STREAMSIZE = n
```

次に例を示します。

```
OPTIONS (BINDSIZE=100000, SILENT=(ERRORS, FEEDBACK) )
```

---

---

**注意：** コマンドラインで指定された値は、制御ファイルの OPTIONS 句で指定された値よりも優先されます。

---

---

## ファイル名およびオブジェクト名の指定

通常、オブジェクト名（表名や列名など）の指定に関して、SQL\*Loader は SQL 標準規則に準拠しています。この項では、次の項目について説明します。

- [SQL および SQL\\*Loader の予約語と競合するファイル名](#)
- [SQL 文字列の指定](#)
- [オペレーティング・システムに関する考慮点](#)

### SQL および SQL\*Loader の予約語と競合するファイル名

SQL および SQL\*Loader の予約語は、二重引用符で囲みます。SQL\*Loader の予約語は CONSTANT のみです。

二重引用符は、オブジェクト名に SQL が認識可能な文字（\$, #, \_）以外の特殊文字が含まれている場合、またはオブジェクト名に大 / 小文字の区別が必要な場合に使用する必要があります。

**参照：**『Oracle9i SQL リファレンス』を参照してください。

### SQL 文字列の指定

SQL 文字列を指定する場合は、二重引用符で囲みます。SQL 文字列を使用すると、SQL 演算子をデータフィールドに適用できます。

**参照：** 6-48 ページの「[フィールドへの SQL 演算子の適用](#)」を参照してください。

### オペレーティング・システムに関する考慮点

次の項では、ご使用のオペレーティング・システムによって異なる仕様について説明します。

#### 完全パスの指定

完全パス名を指定すると、問題が発生する場合があります。これは、特殊文字の使用が原因でオペレーティング・システム依存の非互換が発生するためです。多くの場合、パス名を二重引用符で囲んで指定すると、このエラーは回避できます。

この方法で解決できない場合は、オペレーティング・システム固有のドキュメントで、他の解決法を参照してください。

## バックスラッシュ・エスケープ文字の使用

DDL 構文では、（ご使用のオペレーティング・システムでエスケープ文字の使用が許可されている場合）二重引用符の前にエスケープ文字のバックスラッシュ「\」を付けて、二重引用符で区切られた文字列の中で二重引用符を使用できます。一重引用符で区切られた文字列中で一重引用符を使用する場合も、その前にバックスラッシュを付けます。

たとえば、`homedir\data\norm\mydata` という文字列には、二重引用符が含まれています。二重引用符の直前にバックスラッシュを付けることによって、二重引用符を文字列として使用できます。

```
INFILE 'homedir\data\'norm\mydata'
```

また、バックスラッシュを 2 回続けて書くことによって、バックスラッシュ自体を文字列中表示できます。

次に例を示します。

"so\"far"	or	'so\'far'	is parsed as	so"far
"'so\\far'"	or	'\\'so\\far\\''	is parsed as	'so\far'
"so\\\\far"	or	'so\\\\far'	is parsed as	so\\far

---

**注意：** 先頭位置の二重引用符はエスケープできません。そのため、先頭に引用符の付く文字列は作成しないでください。

---

## 移植不能文字列

SQL\*Loader 制御ファイルには、オペレーティング・システム間で移植不能な、2 種類の文字列があります。*filename* 文字列および *file processing option* 文字列です。SQL\*Loader 制御ファイルを別のオペレーティング・システム用に変換した場合、これらの文字列を修正する必要があります。SQL\*Loader 制御ファイルのこれ以外の文字列はすべて、異なるオペレーティング・システム間で移植可能です。

## バックスラッシュのエスケープ

オペレーティング・システムで、パス名のディレクトリを区切る文字としてバックスラッシュが使用されている場合、およびオペレーティング・システムで実行している Oracle データベース・サーバーのバージョンで、ファイル名およびその他の移植不能文字列に対してバックスラッシュをエスケープ文字として使用可能な場合は、パス名のディレクトリ間の区切りにバックスラッシュを 2 つ続けて指定して、パス名全体を一重引用符で囲みます。

必要なエスケープ文字または使用可能なエスケープ文字の詳細は、ご使用のオペレーティング・システム固有の Oracle マニュアルを参照してください。

## エスケープ文字が使用できない場合

現在使用しているバージョンの Oracle データベース・サーバーでは、移植不能文字列に対してエスケープ文字を使用できない場合があります。エスケープ文字が禁止されている場

合、バックスラッシュは、エスケープ文字ではなく通常の文字として処理されます（ただし、移植不能文字列以外の文字列では使用可能）。この場合、次のようなパス名を指定できます。

```
INFILE 'topdir\mydir\myfile'
```

バックスラッシュを2つ続ける必要はありません。

バックスラッシュはエスケープ文字として認識されないため、一重引用符で囲まれた文字列の中に、一重引用符で囲まれた別の文字列を埋め込むことはできません。これは、二重引用符についても同様です。二重引用符で囲まれた文字列は、二重引用符で区切られた他の文字列の中に埋め込むことはできません。

## データ・ファイルの指定

ロードするデータを含むデータ・ファイルを指定するには、`INFILE` 句にファイル名を続け、必要の場合はファイル処理オプション文字列を続けます。指定するファイルが複数ある場合は、`INFILE` 句を複数使用できます。

---

---

**注意：** コマンドラインからデータ・ファイルを指定する場合は、4-4 ページの「[コマンドライン・パラメータ](#)」で説明している `DATA` パラメータを使用します。コマンドラインでファイル名を指定すると、制御ファイルの最初の `INFILE` 句が上書きされます。

---

---

ファイル名が指定されない場合は、デフォルトで制御ファイル名の拡張子を `.dat` にしたものが採用されます。

ロードするデータを制御ファイル内にも記述した場合は、ファイル名にアスタリスク (\*) を指定してください。指定の詳細は、5-10 ページの「[BEGINDATA による制御ファイルのデータの識別](#)」を参照してください。

---

---

**注意：** ここで説明する内容は、プライマリ・データ・ファイルのみに適用されます。`LOBFILE` または `SDF` には適用されません。

`LOBFILE` の詳細は、7-23 ページの「[LOBFILE からの LOB データのロード](#)」を参照してください。

`SDF` の詳細は、7-31 ページの「[SDF](#)」を参照してください。

---

---

INFILE 句の構文は次のとおりです。

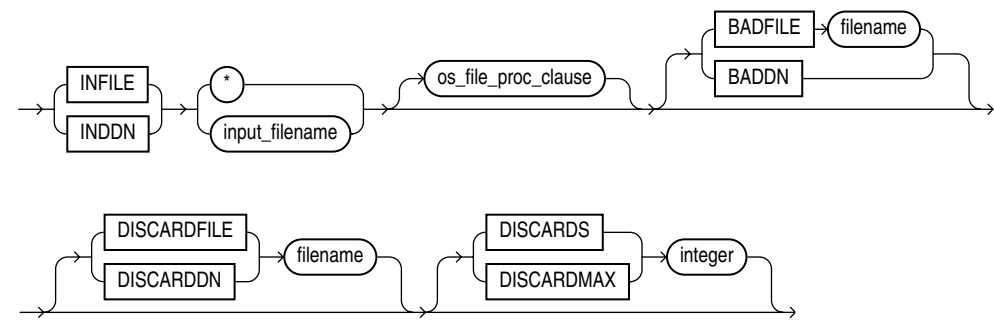


表 5-1 では、INFILE 句のパラメータについて説明します。

表 5-1 INFILE 句のパラメータ

パラメータ	説明
INFILE または INDDN	データ・ファイル指定が続くことを示します。  INDDN は、DB2 互換性が必要な場合に使用します。
input_filename	データが入っているファイルの名前。  ファイル名中に空白やその他の句読点文字が含まれている場合は、一重引用符で囲む必要があります。詳細は、5-5 ページの「 <a href="#">ファイル名およびオブジェクト名の指定</a> 」を参照してください。
*	データが制御ファイル中にある場合は、ファイル名のかわりにアスタリスク (*) を指定します。制御ファイルおよびデータ・ファイルの両方にデータがある場合は、読み込むデータをまずアスタリスクで指定する必要があります。
os_file_proc_clause	ファイル処理オプション文字列。データ・ファイルの形式を指定します。また、この指定によってデータ・ファイルの読込みを最適化できます。この文字列の構文は、ご使用のオペレーティング・システムに依存します。詳細は、5-11 ページの「 <a href="#">データ・ファイル形式およびバッファリングの指定</a> 」を参照してください。

## INFILE 構文の例

次に、INFILE 構文を指定する様々な方法を示します。

- 制御ファイルにデータがある場合

```
INFILE *
```

- デフォルトの拡張子 .dat を持つファイル foo にデータがある場合

```
INFILE foo
```

- フルパスに指定されたファイル datafile.dat にデータがある場合

```
INFILE 'c:/topdir/subdir/datafile.dat'
```

---

**注意：** ファイル名に空白やその他の句読点文字が含まれている場合は、ファイル名を一重引用符で囲む必要があります。ファイル名指定の詳細は、5-5 ページの「[ファイル名およびオブジェクト名の指定](#)」を参照してください。

---

## 複数のデータ・ファイルの指定

1 回の SQL\*Loader の実行で複数のデータ・ファイルのデータをロードする場合は、各データ・ファイルに対して INFILE 文を指定します。このとき、レコードのレイアウトは同じである必要がありますが、データ・ファイルに同じファイル処理オプションは必要ありません。たとえば、最初と 2 番目のファイルが異なるファイル処理オプション文字列で指定されていても、3 番目のファイルが制御ファイル中のデータで構成されていても実行することができます。

各データ・ファイルに、個別の不良ファイルおよび廃棄ファイルを指定することもできます。このような場合、個別の不良ファイルおよび廃棄ファイルは、各データ・ファイル名の直後に宣言する必要があります。次に、4 つのデータ・ファイルを個別の不良ファイルおよび廃棄ファイルとともに指定している制御ファイルの例を示します。

```
INFILE mydat1.dat BADFILE mydat1.bad DISCARDFILE mydat1.dis
INFILE mydat2.dat
INFILE mydat3.dat DISCARDFILE mydat3.dis
INFILE mydat4.dat DISCARDMAX 10 0
```

- mydat1.dat では、不良ファイルと廃棄ファイルの両方が明示的に指定されています。したがって、必要に応じて両方のファイルが作成されます。
- mydat2.dat では、不良ファイルも廃棄ファイルも指定されていません。そのため、不良ファイルのみが必要に応じて作成されます。不良ファイルが作成された場合、デフォルトのファイル名および拡張子 mydat2.bad が使用されます。一方、廃棄ファイルについては、行が廃棄されても廃棄ファイルは作成されません。

- mydat3.dat では、必要に応じてデフォルトの不良ファイルが作成されます。また、必要に応じて、指定された名前の廃棄ファイル (mydat3.dis) も作成されます。
- mydat4.dat では、必要に応じてデフォルトの不良ファイルが作成されます。DISCARDMAX オプションが指定されているため、SQL\*Loader によって廃棄ファイルが使用されることを前提として、廃棄ファイルがデフォルト名 (mydat4.dsc) で作成されます。

## BEGINDATA による制御ファイルのデータの識別

データが制御ファイルにある場合、ファイル名ではなくアスタリスクが INFILE 句の後に続きます。実際のデータは、ロード構成の指定後、制御ファイルに書き込まれます。

最初のデータ・レコードの前に、BEGINDATA パラメータを指定します。構文は次のとおりです。

```
BEGINDATA  
data
```

BEGINDATA パラメータを使用する場合は、次のことに注意してください。

- 制御ファイルにデータが含まれているにもかかわらず、BEGINDATA パラメータを省略すると、SQL\*Loader でデータが制御情報として解釈されるため、エラー・メッセージが発行されます。データが個別ファイルにある場合は、BEGINDATA パラメータは使用できません。
- BEGINDATA を含む行がデータの先頭行と解釈されてしまうため、BEGINDATA パラメータと同じ行には、空白またはその他の文字を入力しないでください。
- コメントもデータとして認識されてしまうため、BEGINDATA の後に続けて記述しないでください。

### 参照：

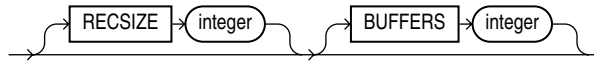
- INFILE の使用例については、5-7 ページの「[データ・ファイルの指定](#)」を参照してください。
- 10-6 ページの「[事例 1: 可変長データのロード](#)」も参照してください。



## データ・ファイル形式およびバッファリングの指定

SQL\*Loader の設定時、制御ファイルの形式およびバッファリングの指定に対して、オペレーティング・システムに依存したファイル処理オプション文字列 (`os_file_proc_clause`) を制御ファイルに指定できます。

たとえば、ご使用のオペレーティング・システムに、次のようなオプション文字列の構文があります。



この構文では、RECSIZE は固定長レコードのサイズ、BUFFERS は非同期 I/O を行うためのバッファ数です。

mydata.dat というデータ・ファイルの宣言で、そのレコード長を 80 バイトとし、SQL\*Loader の I/O 処理で使用するバッファ数を 8 とする場合、次のように指定します。

```
INFILE 'mydata.dat' "RECSIZE 80 BUFFERS 8"
```

ファイル処理オプション文字列の構文の詳細は、ご使用のオペレーティング・システム固有の Oracle マニュアルを参照してください。

---

**注意：** この例では、ファイル名に一重引用符を使用し、その他の指定には二重引用符を使用しています。

---

## 不良ファイルの指定

SQL\*Loader を実行すると、不良ファイルまたは拒否ファイルと呼ばれるファイルが生成されることがあります。これらのファイルには、書式エラーまたは Oracle エラーが発生したためにロードを拒否されたレコードが格納されます。不良ファイルが作成されるように指定した場合は、次の規則に従って作成されます。

- レコードが 1 つ以上拒否された場合、不良ファイルが作成され、拒否レコードがログされます。
- レコードが 1 つも拒否されない場合、不良ファイルは作成されません。この場合、次の実行のために不良ファイルを再度初期化する必要があります。
- 不良ファイルが作成される場合、同じ名前の既存のファイルは上書きされます。そのため、残しておきたいファイルが上書きされないようにする必要があります。

---

**注意：** システムによっては、同じ名前のファイルがすでに存在する場合、そのファイルを上書きせずに新しいファイルを作成することもあります。ご使用のシステムでこのような処理が行われるかどうかは、そのオペレーティング・システム固有の Oracle マニュアルで確認してください。

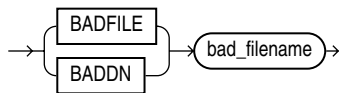
---

不良ファイルの名前を指定する場合は、BADFILE パラメータ（または DB2 互換性が必要な場合は BADDN パラメータ）の後にファイル名を指定します。不良ファイルの名前を指定しなかった場合は、デフォルトで、データ・ファイル名に拡張子（またはファイル・タイプ）.bad を付けたものがファイル名になります。コマンドラインから不良ファイルを指定する場合は、4-4 ページの「**コマンドライン・パラメータ**」で説明している BAD パラメータを使用します。

コマンドラインから指定した不良ファイル名は、制御ファイル中の最初に記述される INFILE 句または INDDN 句に対して指定されたものとみなされます。この場合、前述の句の中で不良ファイル名が指定されていても、コマンドラインから指定した不良ファイル名の方が優先されます。

生成される不良ファイルのファイル形式やレコード形式は、データ・ファイルと同じです。したがって、データを修正した後そのまま再ロードすることができます。ストリーム・レコード形式のデータ・ファイルに関しては、データ・ファイルにあるレコード終了記号が不良ファイル内でも使用されます。

不良ファイルの構文は次のとおりです。



BADFILE または BADDN パラメータの後には、不良ファイルのファイル名を指定します。（DB2 互換性が必要な場合は、BADDN を使用します。）

*bad\_filename* パラメータには、使用するプラットフォームに有効なファイル名を指定します。ファイル名中に空白やその他の句読点文字が含まれている場合は、一重引用符で囲む必要があります。

## 不良ファイル名指定の例

ファイル名 `foo` およびデフォルトのファイル拡張子（またはファイル・タイプ）`.bad` で不良ファイルを指定するには、次のとおり入力します。

```
BADFILE foo
```

ファイル名 `bad0001` およびファイル拡張子（またはファイル・タイプ）`.rej` で不良ファイルを指定するには、次のいずれかの行を入力します。

```
BADFILE bad0001.rej  
BADFILE '/REJECT_DIR/bad0001.rej'
```

## LOBFILE および SDF を使用した不良ファイルの処理方法

LOBFILE および SDF のデータは、拒否された行がある場合、不良ファイルには書き込まれません。LOB のロード中にエラーが発生した場合、その行は拒否されません。この場合、LOB 列は、空のまま（NULL ではなく長さが 0（ゼロ）バイト）になります。ただし、LOBFILE を使用して XML 列をロードし、この LOB データのロード中にエラーが発生した場合、XML 列は NULL になります。

## 拒否レコードの条件

レコードは、次の理由で拒否されます。

1. レコードの挿入時に、指定されたデータ型と実際のデータが適合しないなどの Oracle エラーが発生する場合。
2. レコードが不適切にフォーマットされて、SQL\*Loader がフィールドの境界を検索できない場合。
3. レコードが制約に違反するか、または一意の索引を非一意にしようとする場合。

デリミタの指定が不適切であっても、WHEN 句の指定条件に基づいてデータを評価できる場合は、条件判断が行われ、データが挿入または拒否されます。

前述のリストの 2 番目の理由でロードが拒否された場合、従来型パス・ロードでもダイレクト・パス・ロードでも表に行は書き込みできません。

さらに、前述のリストの 1 番目または 3 番目の理由でいずれかの表に違反がある場合、従来型パス・ロードでは、表に行は書き込みできません。行はその表に対して拒否され、拒否ファイルに書き込まれます。

ログ・ファイルには、拒否レコードそれぞれについての Oracle エラー情報が記録されます。拒否レコードの例の詳細は、10-15 ページの「[事例 4: 結合された物理レコードのロード](#)」を参照してください。

## 廃棄ファイルの指定

SQL\*Loader を実行すると、ロード条件を満たさないレコードがある場合、廃棄ファイルが生成されることがあります。このファイルに出力されたレコードは廃棄レコードと呼ばれます。廃棄レコードは、制御ファイル中の WHEN 句の指定を満たすことができなかったものです。これらのレコードは、拒否レコードとは異なります。廃棄レコードに必ず不良データがあるとは限りません。また、廃棄レコードに対して、挿入は行われません。

廃棄ファイルは次の規則に従って生成されます。

- 廃棄ファイル名を指定し、1 つ以上のレコードが制御ファイルに指定したすべての WHEN 句の条件を満たさない場合に生成されます。(廃棄ファイルが生成される場合、同じ名前の既存のファイルは上書きされます。そのため、残しておきたいファイルが上書きされないようにする必要があります。)
- 廃棄レコードがない場合、廃棄ファイルは生成されません。

制御ファイル内から廃棄ファイルを作成するには、DISCARDFILE *filename*、DISCARDN *filename* (DB2)、DISCARDS または DISCARDMAX のいずれかを指定します。

コマンドラインから廃棄ファイルを作成するには、DISCARD または DISCARDMAX のいずれかを指定します。

このように、名前を指定して廃棄ファイルを直接指定することも、また、廃棄数の最大値を指定することで間接的に廃棄ファイルの生成を指示することもできます。

生成される廃棄ファイルのファイル形式やレコード形式は、データ・ファイルと同じです。ストリーム・レコード形式のデータ・ファイルに関しては、データ・ファイルにあるレコード終了記号が廃棄ファイル内でも使用されます。

## 制御ファイルでの廃棄ファイルの指定

廃棄ファイル名を指定する場合、DISCARDFILE または DISCARDN (DB2 互換用) パラメータの後にファイル名を指定します。



DISCARDFILE または DISCARDN パラメータの後には、廃棄ファイルのファイル名を指定します。(DB2 互換性が必要な場合は、DISCARDN を使用します。)

*discard\_filename* パラメータには、使用するプラットフォームに有効なファイル名を指定します。ファイル名中に空白やその他の句読点文字が含まれている場合は、一重引用符で囲む必要があります。

デフォルトのファイル名は、データ・ファイル名にデフォルトのファイル拡張子（またはファイル・タイプ）.dsc を付けたものが採用されます。廃棄ファイル名をコマンドラインか

ら指定した場合は、制御ファイル中で指定されたファイル名よりも、コマンドラインからの指定ファイル名が優先されます。生成される廃棄ファイルと同じ名前のファイルがすでに存在する場合は、既存ファイルが上書きされるか、新しいバージョンのファイルが生成されます。どちらの処理が行われるかは、使用するオペレーティング・システムによって異なります。

## コマンドラインからの廃棄ファイルの指定

コマンドラインから廃棄ファイルを指定する方法の詳細は、4-6 ページの「[DISCARD \(ファイル名\)](#)」を参照してください。

コマンドラインで指定したファイル名で、制御ファイルに指定した廃棄ファイルが上書きされます。

## 廃棄ファイル名指定の例

次に、制御ファイル内から廃棄ファイルの名前を指定する様々な方法を示します。

- ファイル名 `circular` およびデフォルトのファイル拡張子（またはファイル・タイプ）`.dsc` で廃棄ファイルを指定するには、次のとおり入力します。

```
DISCARDFILE circular
```

- ファイル名 `notappl` およびファイル拡張子（またはファイル・タイプ）`.may` で廃棄ファイルを指定するには、次のとおり入力します。

```
DISCARDFILE notappl.may
```

- 廃棄ファイル `forget.me` にフルパスを指定するには、次のとおり入力します。

```
DISCARDFILE '/discard_dir/forget.me'
```

## 廃棄レコードの条件

レコードに対して `INTO TABLE` 句が指定されていない場合、そのレコードは廃棄されます。レコードの廃棄は、SQL\*Loader 制御ファイル中に記述されたすべての `INTO TABLE` 句が `WHEN` 句を持っていて、レコードがそれらのどの条件にも該当しない場合、またはすべてのフィールドが `NULL` である場合に発生します。

`INTO TABLE` 句が `WHEN` 句なしで指定されている場合、レコードは廃棄されません。すべてのレコードに関して、指定の表への挿入が試みられます。このとき、レコードが拒否されることはありますが、廃棄されることはありません。

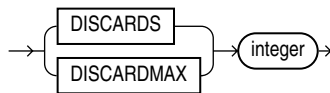
10-29 ページの「[事例 7: 書式化されたレポートからのデータの抽出](#)」にある廃棄ファイルの使用例を参照してください。

## LOBFILE および SDF を使用した廃棄ファイルの処理方法

LOBFILE および SDF のデータは、廃棄された行がある場合も、廃棄ファイルには書き込まれません。

### 廃棄レコード数の上限付け

各データ・ファイルに対して廃棄されるレコード数の上限を整数で指定できます。これには、次のような句を使用します。



廃棄レコード数がこの数 (*integer* で指定) に達すると、そのデータ・ファイルの処理は終了します。次のデータ・ファイルがあれば、その処理が開始されます。

データ・ファイルごとに異なる数の上限を設定できます。ただし、1 回のみ廃棄数を指定した場合は、指定した廃棄上限値はすべてのファイルに適用されます。

廃棄数の上限値が指定されていて、廃棄ファイル名の指定がない場合、SQL\*Loader によってデフォルトのファイル名とファイル拡張子（またはファイル・タイプ）で廃棄ファイルが作成されます。

## 異なる文字コード体系の処理

SQL\*Loader では、(キャラクタ・セットまたはコード・ページと呼ばれる) 異なる文字コード体系がサポートされます。SQL\*Loader では、Oracle グローバリゼーション・サポート・テクノロジーの機能を使用して、様々な文字コード体系 (シングルバイトおよびマルチバイト) を扱うことができます。

**参照：**『Oracle9i Database グローバリゼーション・サポート・ガイド』を参照してください。

通常、シフト・センシティブ文字データのロードは、ASCII データまたは EBCDIC データのロードに比べて非常に遅くなります。シフト・センシティブ文字データを最速でロードするには、デリミタのない固定位置フィールドを使用します。パフォーマンスを向上させるには、次の点に注意してください。

- フィールド・データに、同数のシフトアウト / シフトイン・バイトが含まれる必要があります。
- フィールドは、シングル・バイト・モードで開始および終了する必要があります。
- 最初のバイトをシフトアウト、最後のバイトをシフトインにできます。

- 最初と最後の文字は、マルチバイトにできません。
- 空白が保存され、マルチバイトの空白を確認する必要がある場合、より遅いパスが使用されます。これは、シングルのバイトの空白の削除が実行された後、シフトインのバイトがフィールドの最後のバイトとなる場合に、発生する可能性があります。

次の項では、サポートしている文字コード体系について簡単に説明します。

## マルチバイト（アジア系言語）・キャラクタ・セット

マルチバイト・キャラクタ・セットは、アジア系言語をサポートするために使用されます。データをマルチバイト形式でロードしたり、データベース・オブジェクト名（フィールド、表など）をマルチバイト文字で指定することもできます。制御ファイルでは、コメントおよびオブジェクト名にもマルチバイト文字を使用できます。

## Unicode キャラクタ・セット

SQL\*Loader では、Unicode キャラクタ・セットのデータのロードがサポートされています。

Unicode は、エンコーディングされたユニバーサル・キャラクタ・セットであり、単一のキャラクタ・セットではほとんどの言語による情報の格納をサポートします。Unicode では、プラットフォーム、プログラムまたは言語に関係なく、すべての文字に一意的なコード値が指定されます。Unicode には、2 つの異なるエンコーディング（UTF-16 および UTF-8）が存在します。

---

---

**注意：** このマニュアルでは、UTF-16 および UTF16 の両方の用語が使用されています。UTF-16 という用語は、一般的に Unicode の UTF-16 エンコーディングを意味します。UTF16（ハイフンなし）という用語は、キャラクタ・セットの固有の名前で、UTF-16 エンコーディングの使用時に CHARACTERSET パラメータに指定する必要のある用語です。UTF-8 および UTF8 の場合も同様です。

---

---

Unicode の UTF-16 エンコーディングは、固定幅マルチバイト・エンコーディングで、文字コード 0x0000 ～ 0x007f は、シングルバイト ASCII コードの 0x00 ～ 0x7f と同じ意味です。

Unicode の UTF-8 エンコーディングは、可変幅マルチバイト・エンコーディングで、文字コード 0x00 ～ 0x7f が ASCII と同じ意味です。UTF-8 の文字は 1 バイト、2 バイトまたは 3 バイトの長さになります。

**参照：**

- 10-48 ページの「[事例 11: Unicode キャラクタ・セットのデータのロード](#)」を参照してください。
- Unicode エンコーディングの詳細は、『Oracle9i Database グローバリゼーション・サポート・ガイド』を参照してください。

## データベース・キャラクタ・セット

Oracle データベース・サーバーでは、SQL の CHAR データ型 (CHAR、VARCHAR2、CLOB および LONG) に格納されたデータ、表名などの識別子、SQL 文および PL/SQL ソース・コードに対して、データベース・キャラクタ・セットが使用されます。シングルバイト・キャラクタ・セットおよび ASCII 文字または EBCDIC 文字のいずれかを含む可変幅キャラクタ・セットのみがデータベース・キャラクタ・セットとしてサポートされます。マルチバイト固定幅キャラクタ・セット (たとえば、AL16UTF16) は、データベース・キャラクタ・セットとしてサポートされません。

SQL の NCHAR データ型 (NCHAR、NVARCHAR および NCLOB) に格納されたデータ用のデータベースでは、代替キャラクタ・セットを使用できます。代替キャラクタ・セットは、データベース各国語キャラクタ・セットと呼ばれます。Unicode キャラクタ・セットのみが、データベース各国語キャラクタ・セットとしてサポートされます。

## データ・ファイル・キャラクタ・セット

デフォルトでは、データ・ファイルには、NLS\_LANG パラメータで定義されているキャラクタ・セットが使用されます。NLS\_LANG でサポートされているデータ・ファイル・キャラクタ・セットは、データベース・キャラクタ・セットとしてサポートされているものと同じです。SQL\*Loader では、データ・ファイル内の Oracle でサポートされているすべてのキャラクタ・セットがサポートされます (データベース・キャラクタ・セットとしてサポートされていないものもサポートされます)。

たとえば、SQL\*Loader では、データ・ファイル内のマルチバイト固定幅キャラクタ・セット (AL16UYF16、JA16EUCFIXED など) がサポートされます。また、SQL\*Loader では、リトル・エンディアン・バイト順序による UTF-16 エンコーディングもサポートされます。ただし、Oracle データベース・サーバーでは、ビッグ・エンディアン・バイト順序による UTF-16 エンコーディングのみが、データベース・キャラクタ・セットとしてではなく、データベース・各国語キャラクタ・セットとしてのみサポートされます。

データ・ファイルのキャラクタ・セットは、NLS\_LANG パラメータ、または SQL\*Loader の CHARACTERSET パラメータを使用して設定できます。



## 入力文字の変換

CHARACTERSET パラメータが指定されていない場合、すべてのデータ・ファイルに対するデフォルトのキャラクタ・セットは、NLS\_LANG パラメータで定義されたセッション・キャラクタ・セットです。入力データ・ファイルで使用されるキャラクタ・セットは、CHARACTERSET パラメータで指定できます。

SQL\*Loader には、データ・ファイル・キャラクタ・セット、データベース・キャラクタ・セットおよびデータベース各国語キャラクタ・セットが異なる場合、データ・ファイルのデータをデータベース・キャラクタ・セットまたはデータベース各国語キャラクタ・セットに自動的に変換する機能もあります。

データのキャラクタ・セット変換が必要な場合、ターゲットのキャラクタ・セットは、ソースのデータ・ファイル・キャラクタ・セットのスーパーセットである必要があります。そうでない場合、ターゲット・キャラクタ・セットに同じ文字がない文字は、置換文字（通常、疑問符 (?) などのデフォルトの文字）に変換されます。これによって、データが失われます。

データベース・キャラクタ・タイプ CHAR および VARCHAR2 のサイズは、バイト単位（バイト長セマンティクス）または文字単位（文字長セマンティクス）で指定できます。バイト単位で指定され、データ・キャラクタ・セットの変換が必要な場合、変換される文字のソース・キャラクタ・セットより多くのバイトがターゲット・キャラクタ・セットで使用されると、変換後の値はソースの値より大きくなります。これによって、ターゲットの大きい値がデータベースの列のサイズを超える場合、次のエラー・メッセージがレポートされます。

ORA-01401: 列に挿入した値が大きすぎます。

データベースの列サイズを文字単位で指定するか、または制御ファイルの文字サイズを使用してデータを記述することによっても、この問題を回避できます。また、最大列サイズがバイト単位で、変換後の値を保持できる大きさであることを確認する方法もあります。

### 参照:

- データベースの文字列セマンティクスの詳細は、『Oracle9i データベース概要』を参照してください。
- 5-22 ページの「[文字長セマンティクス](#)」も参照してください。

## CHARACTERSET パラメータ

CHARACTERSET パラメータを指定して、SQL\*Loader に入力データ・ファイルのキャラクタ・セットを示します。CHARACTERSET パラメータが指定されていない場合、すべてのデータ・ファイルに対するデフォルトのキャラクタ・セットは、NLS\_LANG パラメータで定義されたセッション・キャラクタ・セットです。文字データ（SQL\*Loader データ型の CHAR、VARCHAR、VARCHARC、数値型 EXTERNAL および日時データ型と期間データ型のフィールド）のみが、データ・ファイルのキャラクタ・セットに影響されます。

CHARACTERSET の構文は次のとおりです。

```
CHARACTERSET char_set_name
```

`char_set_name` 変数で、キャラクタ・セット名を指定します。通常、指定された名前は Oracle がサポートしているキャラクタ・セットの名前である必要があります。

Unicode の UTF-16 エンコーディングに関しては、AL16UTF16 ではなく UTF16 という名前を使用します。AL16UTF16 は、UTF-16 でエンコーディングされたデータに対して Oracle がサポートするキャラクタ・セット名であり、ビッグ・エンディアン・バイト順序の UTF-16 データのみに使用されます。ただし、データ・ファイルを作成するシステムのバイト順序を使用してデータを設定できるため、データ・ファイルのデータはビッグ・エンディアンまたはリトル・エンディアンのいずれにもなります。したがって、異なるキャラクタ・セット名 (UTF16) が使用されます。キャラクタ・セット名 AL16UTF16 もサポートされます。ただし、リトル・エンディアン・バイト順序のデータ・ファイルに対して AL16UTF16 を指定すると、SQL\*Loader では、警告メッセージが発行され、データ・ファイルがビッグ・エンディアンとして処理されます。

CHARACTERSET パラメータは、LOBFILE および SDF のみでなくプライマリ・データ・ファイルに対しても指定できます。異なる入力データ・ファイルには、異なるキャラクタ・セットを指定できます。INFILE パラメータの前に指定される CHARACTERSET パラメータは、プライマリ・データ・ファイルのリスト全体に適用されます。CHARACTERSET パラメータがプライマリ・データ・ファイルに対して指定される場合、指定された値は、LOBFILE および SDF のデフォルトとしても使用されます。LOBFILE または SDF 仕様で CHARACTERSET パラメータを指定すると、デフォルト設定は上書きされます。

CHARACTERSET パラメータで設定されたキャラクタ・セットは、制御ファイルのデータ (INFILE で指定されている) には適用されません。NLS\_LANG パラメータで指定されているセッション・キャラクタ・セット以外のキャラクタ・セットでデータをロードするには、そのデータを別のデータ・ファイルに置く必要があります。

### 参照：

- 6-36 ページの「[バイト順序](#)」を参照してください。
- サポートされているキャラクタ・セットの名前の詳細は、『Oracle9i Database グローバリゼーション・サポート・ガイド』を参照してください。
- 5-21 ページの「[制御ファイル・キャラクタ・セット](#)」も参照してください。
- リトル・エンディアンの UTF-16 でエンコーディングされたデータを含むデータ・ファイルをロードする例については、10-48 ページの「[事例 11: Unicode キャラクタ・セットのデータのロード](#)」を参照してください。

## 制御ファイル・キャラクタ・セット

SQL\*Loader 制御ファイル自体のキャラクタ・セットは、NLS\_LANG パラメータで指定されているセッション・キャラクタ・セットであることが前提です。制御ファイル・キャラクタ・セットがデータ・ファイル・キャラクタ・セットと異なる場合は、次のことに注意してください。SQL\*Loader 制御ファイルで文字列として指定されたデリミタおよび比較句の値は、比較が行われる前に、制御ファイル・キャラクタ・セットからデータ・ファイル・キャラクタ・セットに変換されます。正しく指定するには、文字列値ではなく 16 進文字列を指定してください。

16 進文字列が Unicode の UTF-16 エンコーディングのデータ・ファイルで使用される場合、ビッグ・エンディアンシステムとリトル・エンディアンシステムでバイト順序が異なります。たとえば、ビッグ・エンディアンシステムでは、UTF-16 の「,」（カンマ）は X'002c' です。リトル・エンディアンシステムでは X'2c00' です。SQL\*Loader では、常に、ビッグ・エンディアン形式で 16 進文字列を指定する必要があります。必要に応じて、比較が行われる前に、SQL\*Loader によってバイトが交換されます。これによって、ビッグ・エンディアンおよびリトル・エンディアンの両システム上の制御ファイルで同じ構文を使用することができます。

Unicode の UTF-16 エンコーディングのストリーム形式のデータ・ファイルで使用されるレコード終了記号のデフォルトは、UTF-16 では「\n」（ビッグ・エンディアンのシステムでは 0x000A で、リトル・エンディアンのシステムでは 0x0A00）です。INFILE 行上の "STR 'char\_str'" 指定または "STR x'hex\_str'" 指定を使用して、これらのデフォルト設定を上書きできます。たとえば、次のいずれかの行を使用して、「\n」のかわりに 'ab' をレコード終了記号として使用できます。

```
INFILE myfile.dat "STR 'ab'"
```

```
INFILE myfile.dat "STR x'00410042'"
```

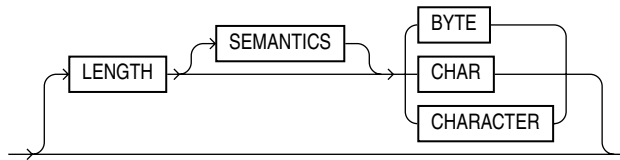
BEGINDATA 文の後に指定するデータも、NLS\_LANG パラメータで指定されたセッション・キャラクタ・セットであることが前提です。

SQL\*Loader のデータ型（CHAR、VARCHAR、VARCHARC、DATE および数値型 EXTERNAL）に関して、SQL\*Loader では、バイト単位（バイト長セマンティクス）または文字単位（文字長セマンティクス）のいずれかで指定される文字フィールドの長さがサポートされます。たとえば、制御ファイルでの CHAR(10) 指定は、10 バイトまたは 10 文字を意味します。データ・ファイルでシングル・バイト・キャラクタ・セットが使用されている場合、これらは同じです。ただし、データ・ファイルでマルチバイト・キャラクタ・セットが使用されている場合、異なることもあります。

キャラクタ・セット変換中に文字列の拡張によって発生する挿入エラーを回避するには、データ・ファイルおよびターゲット・データベースの列の両方で文字長セマンティクスを使用します。

## 文字長セマンティクス

バイト長セマンティクスは、UTF-16 キャラクタ・セット（デフォルトで文字長セマンティクスを使用します）を使用するデータ・ファイル以外のすべてのデータ・ファイルのデフォルトです。デフォルトに上書きするには、次の構文で示すとおり、CHAR または CHARACTER を指定します。



LENGTH パラメータは、SQL\*Loader 制御ファイルの CHARACTERSET パラメータの後に置かれます。LENGTH パラメータは、LOBFILE データ・ファイルおよび SDF に関してのみではなく、プライマリ・データ・ファイルに関する構文の指定にも適用されます。異なる入力データ・ファイルには、異なる文字セマンティクスを指定することができます。ただし、INFILE パラメータの前の LENGTH 指定は、プライマリ・データ・ファイルのリスト全体に適用されます。プライマリ・データ・ファイルに対する LENGTH 指定は、LOBFILE および SDF に対するデフォルトとして使用されます。LOBFILE および SDF 仕様で LENGTH を指定することによって、デフォルトを上書きできます。CHARACTERSET パラメータとは異なり、LENGTH パラメータは、制御ファイル内に含まれるデータ（INFILE \* 構文）に適用されます。

LENGTH パラメータに対しては、CHAR のかわりに CHARACTER を指定できます。

文字長セマンティクスが SQL\*Loader のデータ・ファイルに対して使用されている場合は、次の SQL\*Loader データ型で文字長セマンティクスが使用されます。

- CHAR
- VARCHAR
- VARCHARC
- DATE
- 数値型 EXTERNAL （INTEGER、FLOAT、DECIMAL および ZONED）

VARCHAR データ型では、長さサブフィールドはバイナリの SMALLINT 長さサブフィールドです。ただし、その値は文字単位の文字列の長さを示します。

次のデータ型では、データ・ファイルで文字長セマンティクスが使用されていてもバイト長セマンティクスを使用します。これは、データがバイナリであるか、または ZONED および DECIMAL の場合は、特別にバイナリにエンコーディングされた形式であるためです。

- INTEGER
- SMALLINT

- FLOAT
- DOUBLE
- BYTEINT
- ZONED
- DECIMAL
- RAW
- VARRAW
- VARRAWC
- GRAPHIC
- GRAPHIC EXTERNAL
- VARGRAPHIC

POSITION パラメータの開始引数および終了引数は、文字長セマンティクスがデータ・ファイルで使用されていても、バイト単位で解釈されます。これは、異なるデータ型のデータが混在するデータ・ファイル（一部では文字長セマンティクスを使用し、一部ではバイト長セマンティクスを使用する）を処理する場合に必要です。SMALLINT 長さフィールドおよび文字データを含む VARCHAR データ型で位置を処理する場合にも必要です。SMALLINT 長さフィールドは、システムによって一定のバイト数（通常、2 バイト）を必要とします。ただし、その値は、文字列の長さを文字単位で示します。

データ・ファイルの文字長セマンティクスは、データベースの列で文字長セマンティクスが使用されているかどうかに関係なく使用できます。したがって、データ・ファイルおよびデータベースの列で、同じまたは異なる長さのセマンティクスを使用できます。

## 中断されたロード

ロードは様々な理由で中断されます。主な理由は領域エラーで、SQL\*Loader で使用されるデータ行および索引エン트리用の領域の不足によって発生します。エラーの最大数を越えた場合、サーバーから SQL\*Loader に予期しないエラーが返された場合、レコードがデータ・ファイルに対して長すぎた場合、または [Ctrl] を押しながら [C] を押した場合にも、ロードが中断されます。

ロードが中断された場合の SQL\*Loader の動作は、ロードのモードが従来型パス・ロードかダイレクト・パス・ロードか、およびロードが中断された理由によって異なります。また、中断されたロードを継続する場合、SKIP パラメータを使用するかどうか、およびこのパラメータに指定する値は、状況によって異なります。次の項に例を示します。

**参照：** 4-14 ページの「[SKIP（スキップされるレコード）](#)」を参照してください。

## 中断された従来型パス・ロード

従来型パス・ロードでは、バインド配列のすべてのデータがすべての表にロードされた後で、データがコミットされます。ロードが中断された場合は、直前のコミット操作の時点までに処理された行のみがロードされます。データの部分コミットは行われません。

## 中断されたダイレクト・パス・ロード

ダイレクト・パス・ロードでは、中断されたロードの動作は、ロードが中断された理由によって異なります。

### 領域エラーによって中断されたロード

制御ファイルに 1 つの INTO TABLE 文が存在し、領域エラーが発生した場合は、次のとおりロードが中断されます。

- 非パーティション表、パーティション表の 1 つのパーティションまたはコンポジット・パーティション表の 1 つのサブパーティションにデータをロードすると、SQL\*Loader によって、エラーの発生前にロードされた行と同じ数の行がコミットされます。ROWS パラメータが指定されているかどうかは関係ありません。
- 複数のサブパーティションにデータをロードする（パーティション表、コンポジット・パーティション表またはコンポジット・パーティション表の 1 つのパーティションにデータをロードする）と、ロードが中断されます。ROWS が指定されていないかぎり、データは保存されません。指定されている場合、コミット済みのすべてのデータが保存されます。

制御ファイルに複数の INTO TABLE 文が存在し、これらの表の 1 つで領域エラーが発生した場合は、次のとおりロードが中断されます。

- 非パーティション表、パーティション表の 1 つのパーティションまたはコンポジット・パーティション表の 1 つのサブパーティションにデータをロード中に領域エラーが発生すると、SQL\*Loader によって、データ・ファイルから読み込み済みのデータの他の表へのロードが試行されます。次に、エラーの発生前にロードされた行と同じ数の行がコミットされます。ROWS パラメータが指定されているかどうかは関係ありません。この場合、各表に異なる数の行がロードされることがあります。ロードを継続するには、表ごとに異なる SKIP パラメータの値を指定する必要があります。SKIP パラメータの値がすべての表で同じ場合にのみ、SQL\*Loader によってその値が報告されます。
- 複数のサブパーティションにデータをロード（パーティション表、コンポジット・パーティション表またはコンポジット・パーティション表の 1 つのパーティションにデータをロード）中に領域エラーが発生すると、すべての表でロードが中断されます。ROWS が指定されていないかぎり、データは保存されません。指定されている場合、コミット済みのすべてのデータが保存されます。ロードを継続する場合、SKIP パラメータに指定する値は、すべての表で同じになります。

## エラーが最大数を越えたことによって中断されたロード

エラーが最大数を越えると、SQL\*Loader によって、すべての表へのレコードのロードが停止され、その時点までに終了している処理がコミットされます。ロードを継続する場合、SKIP パラメータに指定する値は表によって異なります。SKIP パラメータの値がすべての表で同じ場合にのみ、SQL\*Loader によってその値が報告されます。

## 致命的エラーによって中断されたロード

致命的エラーが発生した場合は、ロードが停止されます。ロードの開始時に ROWS が指定されていないかぎり、データは保存されません。指定されている場合、コミット済のすべてのデータが保存されます。SKIP パラメータの値がすべての表で同じ場合にのみ、SQL\*Loader によってその値が報告されます。

## [Ctrl] を押しながら [C] を押すことによって中断されたロード

SQL\*Loader によるデータの保存中に [Ctrl] を押しながら [C] を押すと、データの保存は継続され、保存の完了後にロードが停止されます。データを保存中でない場合は、コミットされていない処理をコミットせずにロードが停止されます。SKIP パラメータの値は、すべての表で同じになります。

## ロードの中断後の表および索引の状態

ロードが中断されると、すでにロードされたデータはそのまま表に残り、その表は有効な状態のままとなります。従来型パスを使用した場合、すべての索引は有効な状態のままとなります。

ダイレクト・パス・ロード方法を使用した場合は、領域の不足した索引はすべて使用禁止状態のままになります。このような索引は、ロードを継続する前に削除しておく必要があります。このような索引は、継続処理をする前、またはロードがすべて終了した後に再作成できます。

これ以外の索引は、他にエラーが発生していないかぎり有効です。索引が使用禁止状態のままになるその他の理由については、9-12 ページの「[使用禁止状態 \(Index Unusable\) のままの索引](#)」を参照してください。

## ログ・ファイルを使用したロード状態の確認

SQL\*Loader のログ・ファイルには、表や索引の状態、および入力データ・ファイルから読み込まれた論理レコード数の情報が記録されます。これらの情報は、中断されたロードを再開する場合に利用できます。

## 単一表へのロードの継続

ダイレクト・パス・ロードまたは従来型パス・ロードを、終了前に SQL\*Loader で中断する必要がある場合は、すでにコミットされているか、またはセーブポイントが付けられている行が存在しています。中断されたロードを継続するには、SKIP パラメータを使用して、前回のロードによってすでに処理されている論理レコードの数を指定します。ロードの中断時に、次のようなメッセージ形式でログ・ファイルに SKIP の値が書き込まれます。

Specify SKIP=1001 when continuing the load.

SKIP パラメータの値を指定するこのメッセージの前には、ロードが中断された理由を示すメッセージが書き込まれています。

複数表へのロードの場合は、SKIP パラメータの値がすべての表で同じ場合にのみ、値が表示されます。

**参照：** 4-14 ページの「SKIP (スキップされるレコード)」を参照してください。

## 物理レコードからの論理レコードの作成

Oracle9i は、64KB より大きいユーザー定義レコードをサポートしているため (4-11 ページの「READSIZE (読込みバッファ・サイズ)」を参照)、論理レコードを複数の物理レコードに細分化する必要性が少なくなります。ただし、細分化する必要性が完全になるわけではありません。複数の物理レコードを結合して 1 つの論理レコードに戻すには、対象とするデータによって、次の句のうちの 1 つを指定します。

- `CONCATENATE`
- `CONTINUEIF`

## CONCATENATE を使用した論理レコードの作成

常に一定の数の物理レコードを結合して 1 つの論理レコードを構成する場合、`CONCATENATE` を使用します。次の例では、*integer* によって結合する物理レコードの数を指定します。

```
CONCATENATE integer
```

`CONCATENATE` に指定された整数値によって、SQL\*Loader で列配列の各行に割り当てられる物理レコード構造の数が決まります。COLUMNARRAYROWS のデフォルト値が大きいため、`CONCATENATE` にも大きい値を指定すると、メモリーの過剰割当てが発生する可能性があります。この場合は、COLUMNARRAYROWS パラメータの値を小さくして列配列の行の数を減らすことによって、パフォーマンスを向上できます。



参照： 次の項を参照してください。

- 4-4 ページの「COLUMNARRAYROWS」
- 9-21 ページの「列配列の行数およびストリーム・バッファ・サイズの指定」

CONTINUEIF を使用した論理レコードの作成

継続する物理レコードの数が異なる場合、CONTINUEIF を使用します。パラメータ CONTINUEIF を使用する場合は、後に条件を指定します。この条件は、各物理レコードが読み込まれるたびに評価されます。たとえば、2 つのレコードがあって、その最初のレコードの 80 バイト目にシャープ記号 (#) がある場合、2 つのレコードは結合されます。この場合、指定の位置に他の文字があると、2 番目のレコードは最初のレコードに結合されません。

次に、CONTINUEIF の完全な構文を示します。この構文で、さらに柔軟な処理を実行できます。

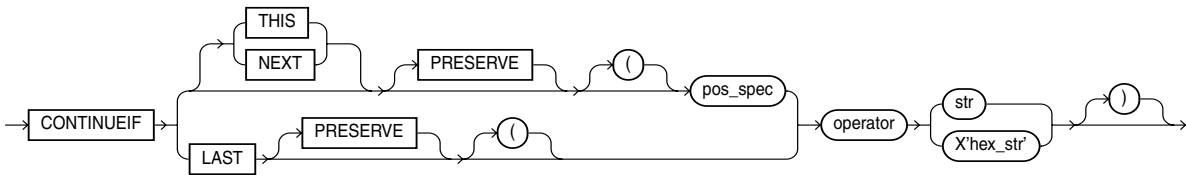


表 5-2 では、CONTINUEIF 句のパラメータについて説明します。

表 5-2 CONTINUEIF のパラメータ

パラメータ	説明
THIS	現在のレコードについて条件が真のとき、次の物理レコードを読み込んで現在のレコードに連結します。この処理は、条件が偽になるまで繰り返されます。条件が偽になると、現在の物理レコードが現在の論理レコードを構成する最後の物理レコードになります。THIS はデフォルトです。
NEXT	次のレコードで条件が真になると、現在の物理レコードを現在の論理レコードに連結します。この処理は、条件が偽になるまで繰り返されます。
operator	サポートされているのは等号演算子と不等号演算子です。  等号演算子を指定すると、フィールドと比較文字列が完全に一致したときに、条件が真となります。不等号演算子を指定した場合は、どの文字が異なっていても、条件は真となります。

表 5-2 CONTINUEIF のパラメータ (続き)

パラメータ	説明
LAST	レコードの最後の文字 (空白以外) に対して、THIS と同様の条件判断を行います。現在の物理レコードの最後の文字 (空白以外) が条件を満たしていると、次の物理レコードが読み込まれ、現在の物理レコードに連結されます。この処理は、条件が偽になるまで繰り返されます。現在のレコードに関して条件が偽になると、現在の物理レコードが現在の論理レコードを構成する最後の物理レコードになります。
pos_spec	物理レコード中の列の開始番号と終了番号です。  列番号は 1 から始まります。ハイフンまたはコロンを使用することもできます (start-end または start:end)。  end を省略すると、継続フィールドの長さには、指定されたバイト列 (X' hex_string') または文字列 (char_string) の長さが取られます。end が指定されていて、それによって決まる継続フィールドの長さが指定されたバイト列または文字列の長さとは異なる場合は、短い方に不足分を埋めるための文字が追加されます。文字列の場合は空白が追加され、16 進数のバイト列の場合は 0 (ゼロ) が追加されます。
str	start と end で定義された継続フィールドと比較する文字列です。比較演算の内容は、operator で指定された演算子によって異なります。文字列は一重引用符または二重引用符で囲みます。文字列は 1 文字ずつ比較され、必要があれば、空白埋め文字が右側に追加されます。
X'hex-str'	16 進形式で指定された文字列で、用途は str と同じです。たとえば、X'1FB033 と表記した場合は、1F、B0、33 の値を持つ 3 バイトの 16 進文字列を意味します。
PRESERVE	論理レコードには 'char_string' または X'hex_string' が含まれます。デフォルトでは、これらは排除されます。

**注意：** CONTINUEIF 句の中で指定する位置は、各物理レコード中の位置を示します。物理レコード中の位置が参照されるのは、この場合のみです。これ以外の場合、参照先は論理レコードとなります。

PRESERVE パラメータを使用しない場合、継続フィールドは、論理レコードの作成時にすべての物理レコードから削除されます。継続フィールドが読み込まれるまでは、データは同じ論理レコードに連結します。

PRESERVE パラメータを使用した場合、継続フィールドは、論理レコードの作成時にもすべての物理レコードに保持されます。

例 5-2 ～例 5-5 に、PRESERVE パラメータの有無にかかわらず、CONTINUEIF THIS および CONTINUEIF NEXT の使用例を示します。

### 例 5-2 PRESERVE パラメータを使用しない CONTINUEIF THIS

ここで、物理レコードは次のような 14 バイトのレコードであるとしています。また、ピリオドは空白を表します。

```
%aaaaaaaa....
%bbbbbbbb....
..cccccccc....
%dddddddddd..
%eeeeeeeeee..
..ffffffffff..
```

次の例では、CONTINUEIF THIS 句に PRESERVE パラメータは使用されていません。

```
CONTINUEIF THIS (1:2) = '%%'
```

したがって、論理レコードは次のように作成されます。

```
aaaaaaaa....bbbbbbbb....cccccccc....
dddddddddd..eeeeeeeeee..ffffffffff..
```

列 1 および列 2 (たとえば、物理レコード 1 の %) は、論理レコード作成時に物理レコードから削除されることに注意してください。

### 例 5-3 PRESERVE パラメータを使用した CONTINUEIF THIS

例 5-2 の物理レコードと同じ物理レコードがあるとします。

次の例では、CONTINUEIF THIS 句に PRESERVE パラメータが使用されています。

```
CONTINUEIF THIS PRESERVE (1:2) = '%%'
```

したがって、論理レコードは次のように作成されます。

```
%aaaaaaaa....%bbbbbbbb....cccccccc....
%dddddddddd..%eeeeeeeeee..ffffffffff..
```

列 1 および列 2 は、論理レコード作成時に物理レコードから削除されないことに注意してください。

### 例 5-4 PRESERVE パラメータを使用しない CONTINUEIF NEXT

ここで、物理レコードは次のような 14 バイトのレコードとします。また、ピリオドは空白を表します。

```
..aaaaaaaa....  
%bbbbbbbbb....  
%cccccccc....  
..dddddddddd..  
%eeeeeeeeeee..  
%ffffffffffff..
```

次の例では、CONTINUEIF NEXT 句に PRESERVE パラメータは使用されていません。

```
CONTINUEIF NEXT (1:2) = '%'
```

したがって、論理レコードは次のように作成されます（例 5-2 と同じ結果）。

```
aaaaaaaa....bbbbbbbbb....cccccccc....  
dddddddddd..eeeeeeeeeee..ffffffffffff..
```

### 例 5-5 PRESERVE パラメータを使用した CONTINUEIF NEXT

例 5-4 の物理レコードと同じ物理レコードがあるとします。

次の例では、CONTINUEIF NEXT 句に PRESERVE パラメータが使用されています。

```
CONTINUEIF NEXT PRESERVE (1:2) = '%'
```

したがって、論理レコードは次のように作成されます。

```
..aaaaaaaa....%bbbbbbbbb....%cccccccc....  
..dddddddddd..%eeeeeeeeeee..%ffffffffffff..
```

**参照：** CONTINUEIF 句の例の詳細は、10-15 ページの「[事例 4: 結合された物理レコードのロード](#)」を参照してください。

## 表への論理レコードのロード

この項では、次の内容について説明します。

- ロードする表の指定方法
- 表にロードするレコードの指定方法
- レコードに対するデフォルトのデータ・デリミタ
- データが欠落しているショート・レコードの処理方法

## 表名の指定

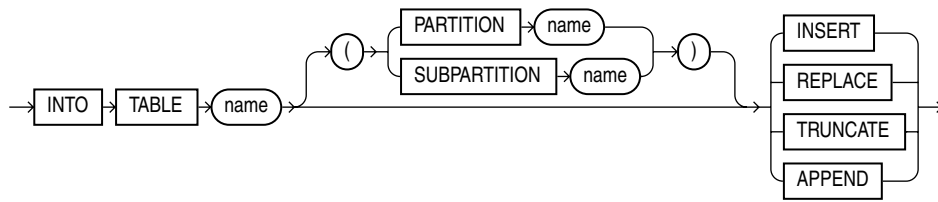
LOAD DATA 文の INTO TABLE 句を使用して、表、フィールドおよびデータ型を識別できます。この句で、データ・ファイル中のレコードとデータベース中の表の関係を定義します。フィールドやデータ型の指定については後述します。

### INTO TABLE 句

INTO TABLE 句が持つ様々な機能の 1 つに、データのロード先となる表を指定する機能があります。複数の表にロードするには、それぞれの表に対して INTO TABLE 句を指定する必要があります。

INTO TABLE 句を記述する場合、キーワード INTO TABLE の次に、データを受け取る Oracle の表名を指定します。

構文は次のようになります。



表名には、すでに存在する表を指定してください。表名が SQL や SQL\*Loader の予約済キーワードと同じ場合、表名に特殊文字が含まれる場合、または表名の中の大 / 小文字を区別する必要がある場合は、表名を二重引用符で囲みます。

```

INTO TABLE scott."CONSTANT"
INTO TABLE scott."Constant"
INTO TABLE scott."-CONSTANT"

```

ユーザーが表をロードするには、INSERT 権限が必要です。表がユーザーのスキーマにない場合、ユーザーはシノニムを使用して表を参照するか、またはスキーマ名を表名の一部として含める（たとえば、scott.emp）必要があります。

## 表固有のロード方法

表のロード時に、INTO TABLE 句を使用して、表固有のロード方法（INSERT、APPEND、REPLACE または TRUNCATE）を指定できます。指定したロード方法は、その表のみに適用されます。表固有のロード方法は、グローバルな表のロード方法よりも優先されます。INTO TABLE 句の前にロード方法が特に指定されていない場合、グローバルな表のロード方法は、デフォルトで INSERT となります。次の項では、これらのオプションを使用して空および空でない表へデータをロードする方法について説明します。

## 空の表へのデータのロード

ロード先の表が空の場合は、INSERT オプションを使用します。

**INSERT** これは、SQL\*Loader のデフォルトの方法です。ロードする前に表を空にする必要があります。表に行が存在する場合はエラーが返され、ロードが終了します。10-6 ページの「[事例 1: 可変長データのロード](#)」の例を参照してください。

## 空でない表へのデータのロード

ロード先の表にすでにデータが存在する場合は、3 つのオプションがあります。

- APPEND
- REPLACE
- TRUNCATE

---

---

**注意：** REPLACE または TRUNCATE を指定すると、個々の行ではなく表全体が置き換えられます。行の削除が成功した後に、コミットが実行されます。ロード前に表にあったデータは、事前にエクスポートまたはそれに相当するユーティリティで保存しておかないかぎり、リカバリできません。

---

---

---

---

**注意：** この項で説明する機能は、DB2 キーワードの RESUME に対応しています。DB2 を使用している場合は、[付録 B](#) の RESUME に関する説明も参照してください。

---

---

**APPEND** 表にデータがすでに存在する場合、SQL\*Loader で新しい行が表に追加されます。データが存在しない場合には、単に新しい行がロードされます。APPEND オプションを使用するには、SELECT 権限が必要です。10-12 ページの「[事例 3: 自由区分形式ファイルのロード](#)」の例を参照してください。

**REPLACE** REPLACE を使用すると、表の既存の行はすべて削除され、新しくデータがロードされます。この場合、その表がロード実行者のスキーマ内に存在するか、ロード実行者がその表に対して DELETE 権限を持っている必要があります。10-15 ページの「[事例 4: 結合された物理レコードのロード](#)」の例を参照してください。

行削除によって、その表に定義された削除トリガーが起動します。DELETE CASCADE が表に指定されている場合、カスケード化された削除が同様に実行されます。削除のカスケード化の詳細は、『Oracle9i データベース概要』の「データ整合性」を参照してください。

**既存の行の更新** REPLACE は、個々の行ではなく表を置換する方法です。既存レコードに NULL 列があっても、そのレコードは更新されません。既存の行を更新するには、次の手順を実行してください。

1. データを作業表にロードします。
2. 相関副問合せを指定した、SQL 言語の UPDATE 文を使用します。
3. 作業表を削除します。

詳細は、『Oracle9i SQL リファレンス』の「UPDATE 文」を参照してください。

**TRUNCATE** SQL の TRUNCATE 文によって短時間で効率的に表またはクラスタから行を削除して、最大限の処理パフォーマンスを実現します。TRUNCATE 文を実行する前に、表の参照整合性制約を使用禁止にします。参照整合性制約が使用禁止でない場合、SQL\*Loader によってエラーが返されます。

整合性制約が使用禁止になると、その表に対しては DELETE CASCADE は定義されません。DELETE CASCADE 機能が必要な場合は、ロードを開始する前に、表の内容を手動で削除する必要があります。

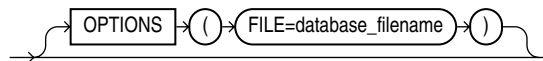
この場合、表がロード実行者のスキーマにあるか、ロード実行者が DROP ANY TABLE 権限を所有している必要があります。

**参照：** TRUNCATE 文の詳細は、『Oracle9i データベース管理者ガイド』を参照してください。

## 表固有の OPTIONS パラメータ

パラレル・ロードでは、個々の表に対して OPTIONS パラメータを指定できます（このパラメータは、パラレル・ロードの場合のみ有効です）。

OPTIONS パラメータの構文は次のとおりです。

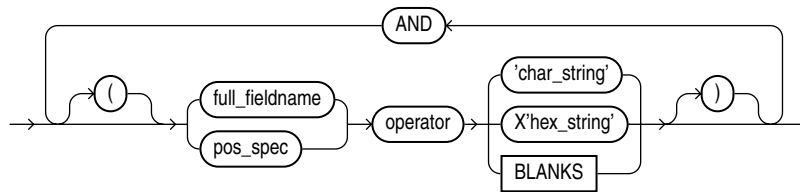


**参照：** 9-32 ページの「[パラレル・ダイレクト・パス・ロードのパラメータ](#)」を参照してください。

## 条件に基づいたレコードのロード

WHEN 句を使用して論理レコード中の条件をテストし、その論理レコードをロードするか廃棄するかを選択できます。

WHEN 句を表名の後に記述し、その後にフィールド条件を 1 つ以上指定します。  
field\_condition の構文は、次のとおりです。



たとえば、次のように指定すると、第 5 列の値が q であるレコードがすべてロードされます。

```
WHEN (5) = 'q'
```

WHEN 句では各条件の前に AND を使用して、複数の条件を設定できます。小カッコの指定は任意ですが、AND によって複数の条件を設定している次のような場合は、あいまいさ为了避免のために必ず使用してください。

```
WHEN (deptno = '10') AND (job = 'SALES')
```

### 参照：

- SQL\*Loader による NULLIF、DEFAULTIF および WHEN 句の評価方法については、6-32 ページの「[WHEN、NULLIF および DEFAULTIF 句の使用](#)」を参照してください。
- WHEN 句の使用例については、10-19 ページの「[事例 5: 複数表へのデータのロード](#)」を参照してください。

## LOBFILE および SDF での WHEN 句の使用

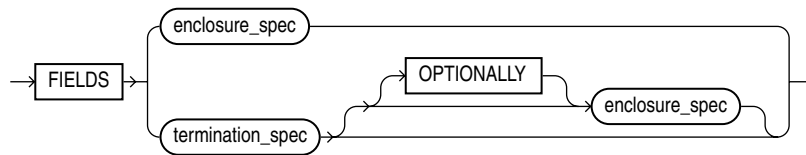
LOBFILE または SDF のレコードが廃棄されると、SQL\*Loader によってその LOBFILE または SDF 内の対応するデータがスキップされます。



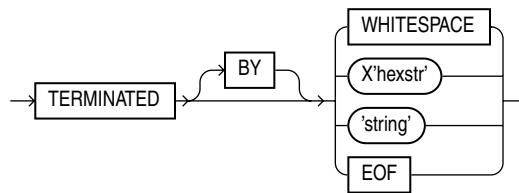
## デフォルトのデータ・デリミタ（区切り記号）の指定

データ・ファイル中のデータフィールドがすべて同じ文字で終了している場合、FIELDS 句を使用してデフォルトのデリミタ（区切り記号）を指定することができます。fields\_spec、termination\_spec および enclosure\_spec 句の構文は、次のとおりです。

### fields\_spec

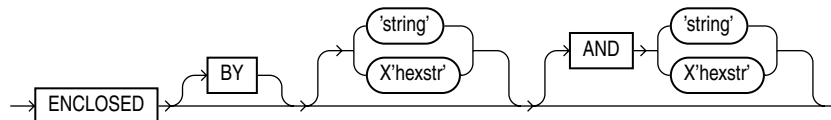


### termination\_spec



**注意：** 終了記号の文字列には、1 つ以上の文字が含まれます。また、`TERMINATED BY EOF` は、`LOBFILE` からの `LOB` のロードにのみ適用されます。

### enclosure\_spec



**注意：** 囲み文字列には、1 つ以上の文字が含まれます。

特定の列に対して別のデリミタを使用する場合は、その列名の後に適用するデリミタを指定します。10-12 ページの「[事例 3: 自由区分形式ファイルのロード](#)」の例を参照してください。

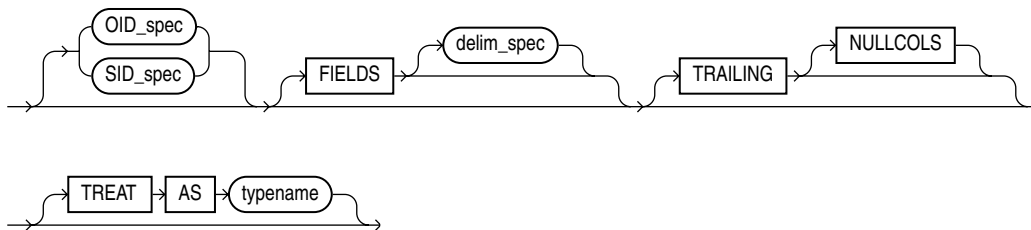
**参照：**

- 構文の詳細は、6-24 ページの「[デリミタの指定](#)」を参照してください。
- 7-23 ページの「[LOBFILE からの LOB データのロード](#)」も参照してください。

## データが欠落しているショート・レコードの処理

制御ファイルの定義で指定したフィールドの数が、実際にレコード中に存在するフィールドより多い場合、残りの（指定された余分の）列に NULL 値を設定するか、またはエラーを出力するかが SQL\*Loader によって判断されます。

制御ファイルの定義でフィールドの開始位置として論理レコードの終了位置よりも後の位置が明示的に指定されている場合、SQL\*Loader によって、このフィールドに NULL 値が設定されます。フィールドが（次に示す例の中の `dname` および `loc` のように）相対位置に定義されていて、そのフィールドが現れる前にレコードのデータが終わった場合は、SQL\*Loader によってこのフィールドに NULL 値が設定されるか、またはエラーが出力されます。SQL\*Loader による処理は、TRAILING NULLCOLS 句（次の構文図を参照）を指定しているかどうかによって決まります。



## TRAILING NULLCOLS 句

TRAILING NULLCOLS 句を使用すると、相対位置で指定した列がレコード中に存在しない場合、その列の値は NULL として処理されます。

たとえば、次のようなデータについて考えます。

10 Accounting

このデータが次の制御ファイルで読み込まれ、そのレコードは `dname` の後で終了するとします。

```
INTO TABLE dept
  TRAILING NULLCOLS
( deptno CHAR TERMINATED BY " ",
  dname  CHAR TERMINATED BY WHITESPACE,
  loc    CHAR TERMINATED BY WHITESPACE
)
```

この場合、その後の `loc` フィールドには NULL 値が設定されています。この例で TRAILING NULLCOLS 句を指定しなかった場合は、データ欠落のためエラーとなります。

**参照：** TRAILING NULLCOLS の例については、10-29 ページの「[事例 7: 書式化されたレポートからのデータの抽出](#)」を参照してください。

## 索引オプション

この項では、索引エントリの作成方法を制御する次の SQL\*Loader オプションについて説明します。

- SORTED INDEXES
- SINGLEROW

## SORTED INDEXES 句

SORTED INDEXES 句はダイレクト・パス・ロードに適用できます。このオプションを指定すると、入力データはロード前に指定の索引でソートされていることが SQL\*Loader で認識されるため、ロード時には SQL\*Loader のパフォーマンスが最適化されます。

**参照：** 9-18 ページの「[SORTED INDEXES 句](#)」を参照してください。

## SINGLEROW オプション

SINGLEROW オプションは、APPEND オプションを使用して限られたメモリーでダイレクト・パス・ロードを実行する場合、または少数のレコードを大規模な表にロードする場合に使用します。このオプションを使用すると、各索引エントリが、一度に 1 レコードずつ直接索引に挿入されます。

表に行を追加 (APPEND) する場合、デフォルトでは SINGLEROW は使用されません。この場合、索引エントリは個別の一時記憶域に置かれ、ロード終了時に元の索引とマージされます。この方法では、パフォーマンスが向上して最適な索引が作成されますが、記憶域も余分に必要となります。マージが実行されている間は、元の索引、新しい索引、および新しいエントリのための領域が同時に記憶域を占有します。

SINGLEROW オプションの場合、新しい索引エントリや新しい索引のための記憶域は必要ありません。結果としてできる索引は、新しくソートされたもののほど最適化されていない可能性があります。作成するための記憶域は少なく済みます。ただし、SINGLEROW を指定すると、各索引が挿入されるたびに UNDO 情報が追加で作成されるため、時間がかかります。このオプションは、次の場合に使用してください。

- 使用可能な記憶域が限られている場合。
- ロードされる行数が表のサイズに比べて小さい場合（比率が 1:20 以下かどうかが目安になります）。

## 複数の INTO TABLE 句を使用するメリット

複数の INTO TABLE 句を指定すると、次の処理が可能になります。

- 異なる表へのデータのロード
- 1 つの入力レコードからの複数の論理レコードの抽出
- 異なる入力レコード形式の区別
- 異なる入力行オブジェクト・サブタイプの区別

ここでは、まず、複数の INTO TABLE 句で同じ表を参照するという最も一般的な使用方法を示します。この項では、複数の INTO TABLE 句の様々な指定方法を説明します。また、POSITION パラメータの指定方法についても説明します。

---

---

**注意：** INTO TABLE 句を複数指定した場合、次の INTO TABLE 句の処理に移ったときのフィールド・スキャンは、前回フィールド・スキャンを停止した位置から続行されます。この項の残りの部分では、このようなスキャン時の動作を利用して INTO TABLE 句を指定する方法について詳しく説明します。また、固定フィールド位置や POSITION パラメータを使用した別の手順についても説明します。

---

---

## 複数の論理レコードの抽出

データ記憶域および転送メディアには、物理レコードが固定長のものがあります。データ・レコードが比較的短い場合、メディアを効率的に使用するために複数の論理レコードをまとめて 1 つの物理レコードに記録できます。

ここでは、入力ファイルの 1 件の物理レコードを 2 件の論理レコードとみなし、INTO TABLE 句を 2 回指定して emp 表にデータをロードする方法について説明します。たとえば、次のようなデータの例を考えます。

```
1119 Smith      1120 Yvonne
1121 Albert     1130 Thomas
```

次の制御ファイルを使用して論理レコードを抽出します。

```
INTO TABLE emp
(empno POSITION(1:4)  INTEGER EXTERNAL,
  ename POSITION(6:15) CHAR)
INTO TABLE emp
(empno POSITION(17:20) INTEGER EXTERNAL,
  ename POSITION(21:30) CHAR)
```

## デリミタに基づく相対的な位置指定

同じレコードを、別の指定方法でロードできます。次の制御ファイルでは、絶対的位置を指定するかわりに相対的な位置を指定しています。ここでは、各フィールドが空白 (" ") 1 文字、またはいくつかの空白やタブ (WHITESPACE) で区切られていることを示しています。

```
INTO TABLE emp
(empno INTEGER EXTERNAL TERMINATED BY " ",
  ename CHAR                TERMINATED BY WHITESPACE)
INTO TABLE emp
(empno INTEGER EXTERNAL TERMINATED BY " ",
  ename CHAR)                TERMINATED BY WHITESPACE)
```

この例では、2 番目の empno フィールドは、別の INTO TABLE 句に指定されていますが、1 番目の ename の直後に指定されていることに注意してください。2 番目の INTO TABLE 句に対して、レコードの先頭からのフィールド・スキャンのしなおしは行われません。かわりに、前に行われたスキャンの状態がそのまま残されてスキャンが継続されます。

レコードのスキャンを特定の位置から強制的に開始するには、POSITION パラメータを使用します。詳細は、5-40 ページの「異なる入力レコード形式の区別」および 5-42 ページの「複数表へのデータのロード」を参照してください。

## 異なる入力レコード形式の区別

通常、データ・ファイルには様々な形式のレコードが含まれています。ここでは、次のようなデータについて考えます。この例では、emp 表および dept 表のレコードが、データ中に混在しています。

```
1 50 Manufacturing      - DEPT record
2 1119 Smith           50   - EMP record
2 1120 Snyder          50
1 60 Shipping
2 1121 Stevens         60
```

これら 2 つの形式は、レコード ID フィールドで区別されます。部門レコードの最初の列は 1、従業員レコードの最初の列は 2 になります。このデータをロードするため、次の制御ファイルではフィールドの正確な位置を指定しています。

```
INTO TABLE dept
  WHEN recid = 1
    (recid FILLER POSITION(1:1)  INTEGER EXTERNAL,
     deptno POSITION(3:4)  INTEGER EXTERNAL,
     dname  POSITION(8:21) CHAR)
INTO TABLE emp
  WHEN recid <> 1
    (recid FILLER POSITION(1:1)  INTEGER EXTERNAL,
     empno POSITION(3:6)  INTEGER EXTERNAL,
     ename  POSITION(8:17)  CHAR,
     deptno POSITION(19:20) INTEGER EXTERNAL)
```

## POSITION パラメータに基づく相対的な位置指定

前述の例のレコードは、デリミタ付きのデータとしてもロードできます。ただし、その場合は POSITION パラメータを使用する必要があります。次の制御ファイルを使用します。

```
INTO TABLE dept
  WHEN recid = 1
    (recid FILLER INTEGER EXTERNAL TERMINATED BY WHITESPACE,
     deptno INTEGER EXTERNAL TERMINATED BY WHITESPACE,
     dname  CHAR TERMINATED BY WHITESPACE)
INTO TABLE emp
  WHEN recid <> 1
    (recid FILLER POSITION(1) INTEGER EXTERNAL TERMINATED BY ' ',
     empno  INTEGER EXTERNAL TERMINATED BY ' ',
     ename  CHAR TERMINATED BY WHITESPACE,
     deptno INTEGER EXTERNAL TERMINATED BY ' ')
```

2 番目の INTO TABLE 句の POSITION パラメータは、このデータを正しくロードするために必要です。このように指定すると、2 つ目の書式に一致するデータを確認するときのフィールド・スキャンは、列 1 から開始されます。この POSITION 指定がない場合、

SQL\*Loader では、recid フィールドが dname フィールドの後にあるものとしてスキャンされます。

## 異なる入力行オブジェクトのサブタイプの区別

通常、データ・ファイルには、同じ行オブジェクト型から継承された行オブジェクトで構成される単一のレコードが含まれています。たとえば、次のような単純なオブジェクト型およびオブジェクト表の定義について考えてみます。ここでは、NOT FINAL のベース・オブジェクト型が、ベース型を継承する 2 つのオブジェクト・サブタイプとともに定義されています。

```
CREATE TYPE person_t AS OBJECT
  (name   VARCHAR2(30),
   age    NUMBER(3)) not final;

CREATE TYPE employee_t UNDER person_t
  (empid   NUMBER(5),
   deptno  NUMBER(4),
   dept    VARCHAR2(30)) not final;

CREATE TYPE student_t UNDER person_t
  (stdid   NUMBER(5),
   major   VARCHAR2(20)) not final;

CREATE TABLE persons OF person_t;
```

次の入力データ・ファイルには、これらの行オブジェクト・サブタイプが混在しています。これらのサブタイプは、型 ID フィールドで区別されます。person\_t オブジェクトでは、最初の列に P があり、employee\_t オブジェクトでは E、student\_t オブジェクトでは S があります。

```
P,James,31,
P,Thomas,22,
E,Pat,38,93645,1122,Engineering,
P,Bill,19,
P,Scott,55,
S,Judy,45,27316,English,
S,Karen,34,80356,History,
E,Karen,61,90056,1323,Manufacturing,
S,Pat,29,98625,Spanish,
S,Cody,22,99743,Math,
P,Ted,43,
E,Judy,44,87616,1544,Accounting,
E,Bob,50,63421,1314,Shipping,
S,Bob,32,67420,Psychology,
E,Cody,33,25143,1002,Human Resources,
```

次の制御ファイルでは、POSITION パラメータに基づく相対的な位置指定によって、このデータをロードします。固有のオブジェクト型名を持つ TREAT AS 句の使用方法に注意してください。これによって、オブジェクト表のすべての入力行オブジェクトが名前付きオブジェクト型の定義に準拠することが、SQL\*Loader で認識されます。

```
INTO TABLE persons
REPLACE
WHEN typid = 'P' TREAT AS person_t
FIELDS TERMINATED BY ","
  (typid    FILLER  POSITION(1) CHAR,
   name     CHAR,
   age      CHAR)

INTO TABLE persons
REPLACE
WHEN typid = 'E' TREAT AS employee_t
FIELDS TERMINATED BY ","
  (typid    FILLER  POSITION(1) CHAR,
   name     CHAR,
   age      CHAR,
   empid    CHAR,
   deptno   CHAR,
   dept     CHAR)

INTO TABLE persons
REPLACE
WHEN typid = 'S' TREAT AS student_t
FIELDS TERMINATED BY ","
  (typid    FILLER  POSITION(1) CHAR,
   name     CHAR,
   age      CHAR,
   stdid    CHAR,
   major    CHAR)
```

**参照：** オブジェクト型のロードの詳細は、7-2 ページの「[列オブジェクトのロード](#)」参照してください。

## 複数表へのデータのロード

複数の INTO TABLE 句で POSITION 句を指定することによって、1 件のレコードのデータを正規化された複数の表にロードできます。詳細は、10-19 ページの「[事例 5: 複数表へのデータのロード](#)」を参照してください。



## 要約

複数の INTO TABLE 句を指定すると、1 件の入力レコードから複数の論理レコードを抽出できます。また、同一ファイル中の異なる形式のレコードを区別できます。

デリミタ付きデータの場合、期待する結果を得るには、POSITION パラメータを正しく指定する必要があります。

複数の INTO TABLE 句で POSITION パラメータを指定しない場合、1 件の（デリミタ付きデータ）入力レコードの異なる部分が処理されます。これによって、1 件のレコードから複数の表へのデータ・ロードが可能になります。複数の INTO TABLE 句で POSITION パラメータを指定すると、同一のレコードを異なる方法で処理できます。つまり、1 つの入力ファイルで複数の形式を識別できます。

## バインド配列および従来型パス・ロード

SQL\*Loader では、データのデータベースへの転送時に SQL 配列インタフェース・オプションが使用されます。まず、一度に複数の行が読み込まれてバインド配列に格納されます。SQL\*Loader から Oracle データベースに INSERT コマンドが送られると、配列全体が一度に挿入されます。バインド配列内の行が挿入された後で、COMMIT が発行されます。

バインド配列サイズを決定する必要があるのは、SQL\*Loader の従来型パス・オプションを使用する場合のみです。ダイレクト・パス・ロードでは、Oracle SQL インタフェースではなくダイレクト・パス API が使用されるため、バインド配列サイズを決定する必要はありません。

**参照：** ダイレクト・パス・ロードの概要については、『Oracle Call Interface プログラマーズ・ガイド』を参照してください。

## バインド配列のサイズ要件

バインド配列には、1 行以上が入る領域を確保してください。行の最大長が、BINDSIZE パラメータで指定されたバインド配列のサイズを超えると、SQL\*Loader からエラーが出力されます。通常、バインド配列内に入る範囲の行が格納されます。この場合の読み込み行数の上限は、ROWS パラメータで指定された行数となります。

BINDSIZE パラメータおよび ROWS パラメータについては、4-4 ページの「[コマンドライン・パラメータ](#)」を参照してください。

バインド配列全体が連続するメモリーを占有する必要はありませんが、バインド配列内の各フィールドを格納するバッファには連続するメモリーが必要です。オペレーティング・システムで、フィールド格納用として連続するメモリーを確保できないと、SQL\*Loader からエラーが出力されます。

## バインド配列のパフォーマンスに関する考慮点

バインド配列を大きくすると、Oracle データベース・サーバーへのコール数を最小に、またパフォーマンスを最大にできます。一般に、バインド配列サイズを大きくする場合、100 行まではサイズの増加に比例してパフォーマンスが大幅に向上します。ただし、100 行を超えるバインド配列サイズを設定しても、パフォーマンスはそれほど向上しません。したがって、一般に配列サイズ（バイト単位）は 100 行が目安となります。

一般に、サイズが適切であれば、SQL\*Loader で効果的に処理できます。通常は、この項で説明するような細かい計算をする必要はありません。この項は、パフォーマンスを最大にする場合、またはメモリー使用量を確認する場合に参照してください。

## 行数およびバインド配列サイズの指定

バインド配列サイズを指定する場合に、コマンドライン・パラメータ BINDSIZE（4-4 ページの「[BINDSIZE（最大サイズ）](#)」を参照）または制御ファイル中の OPTIONS 句（5-4 ページの「[OPTIONS 句](#)」を参照）を使用すると、バインド配列の上限値が設定されます。バインド配列は、この上限値を超えることはありません。

初期化の段階で、SQL\*Loader では単一行のロードに必要なサイズがバイト単位で決定されます。このサイズが指定された最大値を超える場合は、エラーが返され、ロードが終了します。

次に SQL\*Loader では、このサイズとロードする行数が掛け合されます。このとき、ロードする行数は、コマンドライン・パラメータ ROWS（4-12 ページの「[ROWS（1 回にコミットする行数）](#)」を参照）で指定されていても、制御ファイル中の OPTIONS 句（5-4 ページの「[OPTIONS 句](#)」を参照）で指定されていてもかまいません。

このサイズがバインド配列の最大値を超えないかぎり、ロードは継続されます。SQL\*Loader では、バインド配列の最大サイズの限界まで行数は拡張されません。行数とバインド配列の最大サイズの両方が指定された場合、SQL\*Loader では、これらの値の小さい方がバインド配列に適用されます。

バインド配列の最大サイズが小さく、指定の行数を格納できない場合は、その最大サイズに収まる分の行数が採用されます。

## バインド配列サイズを確認するための計算

バインド配列サイズは、配列内の行数に各行の最大長を掛け合せた値となります。行の最大長は、次のように、フィールドの最大長の合計にオーバーヘッドを加えた値となります。

```
bind array size =
    (number of rows) * ( SUM(fixed field lengths)
                        + SUM(maximum varying field lengths)
                        + ( (number of varying length fields)
                          * (size of length indicator) )
                      )
```

ほとんどのフィールドのサイズは、固定長です。このような固定長フィールドの場合、ロードされる各行のサイズは同じです。固定長フィールドについては、6-7 ページの「[SQL\\*Loader のデータ型](#)」で説明するとおり、フィールド・サイズがフィールドの最大長（バイト）となります。そのため、オーバーヘッドは発生しません。

行によってサイズが変化するフィールドには、次のようなものがあります。

- CHAR
- DATE
- INTERVAL DAY TO SECOND
- INTERVAL DAY TO YEAR
- LONG VARRAW
- 数値型 EXTERNAL
- TIME
- TIMESTAMP
- TIME WITH TIME ZONE
- TIME WITH TIME ZONE
- VARCHAR
- VARCHARC
- VARGRAPHIC
- VARRAW
- VARRAWC

これらのデータ型の最大長の詳細は、6-7 ページの「[SQL\\*Loader のデータ型](#)」を参照してください。ここでの最大長とは、入力データ・レコードの中でフィールドが占有できる長さを、バイト数で表したものです。この最大長は、バインド配列の中で各フィールドが占有する格納領域のサイズも表しています。バインド配列には、サイズが変化するこれらのフィールドについてのオーバーヘッドも含まれます。

文字データ型（CHAR、DATE および数値型 EXTERNAL）がデリミタ付きで指定された場合は、これらのフィールドに対して指定されたフィールド長が最大長となります。逆に、デリミタなしでこれらのデータ型が指定された場合は、レコードのサイズは固定ですが、挿入時にフィールド中の空白文字が切り捨てられるため、フィールド長は変化します。したがって、これらのデータ型は、たとえ固定長フィールドであっても内部的には可変長フィールドとして扱われます。

長さインジケータは、バインド配列内のそれぞれのフィールドに格納されています。バインド配列におけるフィールドの領域として、そのフィールドの可能最大長のデータを格納できるだけのサイズが確保されています。一方、実際のフィールド長は、行ごとに長さインジケータで示されます。

---

---

**注意：** 従来型パス・ロードでは、バインド配列のサイズの割当て時に LOBFILE は含まれません。

---

---

## 長さインジケータのサイズの決定

ほとんどのシステムでは、長さインジケータのサイズは2バイトです。まれに3バイトのシステムもあります。長さインジケータのサイズを調べるには、次の制御ファイルを作成して実行します。

```
OPTIONS (ROWS=1)
LOAD DATA
INFILE *
APPEND
INTO TABLE DEPT
(deptno POSITION(1:1) CHAR(1))
BEGINDATA
a
```

この制御ファイルは、1行のバインド配列を使用して、1バイトの CHAR をロードします。この例では、実際にデータはロードされません。これは、a を数値型の列（deptno）にロードすると、変換エラーが発生するためです。このときのログ・ファイルに示されたバインド配列サイズから、（文字フィールドの長さである）1 を引いた値が、フィールド長のインジケータのサイズとなります。

---

---

**注意：** これと同様の方法で、計算しないでバインド配列サイズを求めることもできます。制御ファイルにデータを記述せず、ROWS=1 と指定して実行すると、1行のデータに必要なメモリーのサイズがわかります。このサイズとバインド配列に格納する行数を掛け合せば、バインド配列サイズを判断できます。

---

---

## フィールド・バッファ・サイズの計算

表 5-3 ～表 5-6 の表に、各データ型のメモリ要件を示します。「L」は制御ファイルで指定したデータ長で、「P」は精度です。「S」はフィールド長インジケータのサイズです。これらの値の詳細は、6-7 ページの「SQL\*Loader のデータ型」を参照してください。

表 5-3 固定長フィールド

データ型	バイト単位のサイズ (オペレーティング・システムによって異なる)
INTEGER	C 言語の INT データ型に相当するサイズ
INTEGER (N)	N バイト
SMALLINT	C 言語の SHORT INT データ型に相当するサイズ
FLOAT	C 言語の FLOAT データ型に相当するサイズ
DOUBLE	C 言語の DOUBLE データ型に相当するサイズ
BYTEINT	C 言語の UNSIGNED CHAR データ型に相当するサイズ
VARRAW	UNSIGNED SHORT に 4096 バイトまたは <i>max_length</i> に指定した値をプラスしたサイズ
LONG VARRAW	UNSIGNED INT に 4096 バイトまたは <i>max_length</i> に指定した値をプラスしたサイズ
VARCHARC	2 つの数値で構成されます。最初に長さを指定し、次に（オプションで） <i>max_length</i> （デフォルトは 4096 バイト）を指定します。
VARRAWC	このデータ型は、RAW データ用です。2 つの数値で構成されます。最初に長さを指定し、次に（オプションで） <i>max_length</i> （デフォルトは 4096 バイト）を指定します。

表 5-4 非グラフィック・フィールド

データ型	デフォルト・サイズ	指定するサイズ
(PACKED) DECIMAL 型	なし	(N+1)/2 切上げ
ZONED	なし	P
RAW	なし	L
CHAR 型 (デリミタなし)	1	L+S
日時データ型および期間データ型 (デリミタなし)	なし	L+S
数値型 EXTERNAL 型 (デリミタなし)	なし	L+S

表 5-5 グラフィック・フィールド

データ型	デフォルト・サイズ	POSITION での 長さの指定	DATATYPE での 長さの指定
GRAPHIC	なし	L	2 × L
GRAPHIC EXTERNAL	なし	L - 2	2 × (L-2)
VARGRAPHIC	4KB × 2	L+S	(2 × L) +S

表 5-6 可変長フィールド

データ型	デフォルト・サイズ	最大長の 指定 (L)
VARCHAR	4KB	L+S
CHAR 型 (デリミタ付き)	255	L+S
日時データ型および期間データ型 (デリミタ付き)	255	L+S
数値型 EXTERNAL 型 (デリミタ付き)	255	L+S

バインド配列用のメモリー所要量の最小化

VARCHAR 型、VARGRAPHIC 型フィールド、およびデリミタ付きの CHAR 型、DATE 型、数値型 EXTERNAL 型フィールドの場合、そのデータ型に割り当てられているデフォルト・サイズに特に注意してください。このデフォルト・サイズによっては、メモリーを大量に使用することがあります。特に、デフォルト・サイズにバインド配列の行数を掛け合せると、使用するメモリーは非常に大きくなります。これらのフィールドに対しては、最大長としてできるだけ小さな値を指定してください。次の例について考えてみます。

CHAR(10) TERMINATED BY ", "

バイト長セマンティクスを使用する場合、次の例では、バインド配列で (10+2) × 64=768 バイトのメモリーを使用します (ここでは、長さインジケータを 2 バイトで一度に 64 行ロードするとして計算しています)。

同じ例で文字長セマンティクスを使用する場合、バインド配列では ((10+s) +2) × 64 バイトのメモリーを使用します (ここでは、「s」がデータ・ファイル・キャラクタ・セット中の文字のバイト単位での最大値です)。

ここで、次の例について考えてみます。

CHAR TERMINATED BY ", "

バイト長セマンティクスまたは文字長セマンティクスを使用しているかどうかにかかわらず、この例では、 $(255+2) \times 64=16,448$  バイトが必要になります。これは、デリミタ付きフィールドのデフォルト最大長が 255 バイトであるためです。この指定によって、バインド配列に入る行数が大きく違ってきます。

## 複数の INTO TABLE 句に対するバインド配列サイズの計算

制御ファイルに複数の INTO TABLE 句が指定されている場合のバインド配列サイズの計算は、複数の INTO TABLE 句が指定されていない場合と同様に行います。言い換えると、制御ファイルに指定されているフィールド全体を 1 つの長いデータ構造体、つまりバインド配列の中の 1 行のデータ構造体であると考えます。

データ・レコード中の同じフィールドを複数の INTO TABLE 句が参照する場合は、そのフィールドが参照されると、常に、バインド配列に追加の領域が必要となります。このようなフィールドについては、特にバッファの割当てを最小限に抑える必要があります。

---

---

**注意：** CONSTANT、EXPRESSION、RECNUM、SYSDATE および SEQUENCE の各関数を指定すると、SQL\*Loader によってデータが生成されます。このようにして生成されたデータは、バインド配列の領域を必要としません。

---

---





---

## フィールド・リスト・リファレンス

この章では、SQL\*Loader 制御ファイルのフィールド・リストについて説明します。この章の内容は、次のとおりです。

- [フィールド・リストの内容](#)
- [データ・フィールドの位置指定](#)
- [列およびフィールドの指定](#)
- [SQL\\*Loader のデータ型](#)
- [フィールド条件の指定](#)
- [WHEN、NULLIF および DEFAULTIF 句の使用](#)
- [異なるプラットフォーム間でのデータのロード](#)
- [バイト順序](#)
- [すべてが空白のフィールドのロード](#)
- [空白の切捨て](#)
- [空白の保存](#)
- [フィールドへの SQL 演算子の適用](#)
- [SQL\\*Loader を使用した入力データの生成](#)

## フィールド・リストの内容

SQL\*Loader 制御ファイルのフィールド・リストには、位置、データ型、条件、デリミタなど、ロードするフィールドの情報が提供されます。

例 6-1 に、第 5 章で説明したサンプル制御ファイルのフィールド・リスト・セクションを示します。

### 例 6-1 サンプル制御ファイルのフィールド・リスト・セクション

```
.
.
.
1  (hiredate  SYSDATE,
2      deptno  POSITION(1:2)  INTEGER EXTERNAL(2)
          NULLIF deptno=BLANKS,
3      job     POSITION(7:14)  CHAR  TERMINATED BY WHITESPACE
          NULLIF job=BLANKS  "UPPER(:job)",
      mgr      POSITION(28:31) INTEGER EXTERNAL
          TERMINATED BY WHITESPACE, NULLIF mgr=BLANKS,
      ename    POSITION(34:41) CHAR
          TERMINATED BY WHITESPACE  "UPPER(:ename)",
      empno    POSITION(45)   INTEGER EXTERNAL
          TERMINATED BY WHITESPACE,
      sal      POSITION(51)   CHAR  TERMINATED BY WHITESPACE
          "TO_NUMBER(:sal, '$99,999.99')",
4      comm    INTEGER EXTERNAL  ENCLOSED BY '(' AND '%'
          ":comm * 100"
    )
```

このサンプル制御ファイルの左側の数字は、実際の制御ファイルでは表示されません。これらの数字は、次の説明の番号に対応しています。

1. SYSDATE に、列を現在のシステム日付に設定します。詳細は、6-54 ページの「[列への現在の日付の設定](#)」を参照してください。
2. POSITION に、データ・フィールドの位置を指定します。詳細は、6-3 ページの「[データ・フィールドの位置指定](#)」を参照してください。

INTEGER EXTERNAL は、フィールド用のデータ型です。6-7 ページの「[データ・フィールドのデータ型の指定](#)」および 6-19 ページの「[数値型 EXTERNAL](#)」を参照してください。

NULLIF 句は、フィールド条件の指定に使用する句の 1 つです。詳細は、6-32 ページの「[WHEN、NULLIF および DEFAULTIF 句の使用](#)」を参照してください。

このサンプルでは、BLANKS を使用して、フィールドを空白と比較しています。詳細は、6-31 ページの「[フィールドと BLANKS の比較](#)」を参照してください。

- 3. `TERMINATED BY WHITESPACE` 句は、フィールドを指定できるデリミタの 1 つです。詳細は、6-24 ページの「[TERMINATED フィールド](#)」を参照してください。
- 4. `ENCLOSED BY` 句は、もう 1 つの使用可能なフィールド・デリミタです。詳細は、6-47 ページの「[囲まれたフィールド](#)」を参照してください。

## データ・フィールドの位置指定

データ・ファイルからデータをロードする場合は、フィールドの位置と長さを `SQL*Loader` に対して明示する必要があります。論理レコード中のフィールド位置は、列指定 (`columnspec`) の中で `POSITION` 句を使用して指定します。このときフィールド位置は、絶対位置で指定することも、前のフィールドからの相対位置で指定することもできます。`POSITION` に対する引数は、カッコで囲む必要があります。文字長セマンティクスをデータ・ファイルに使用しても、`start`、`end` および `integer` は、常にバイト単位です。

位置指定 (`pos_spec`) 句の構文は次のとおりです。

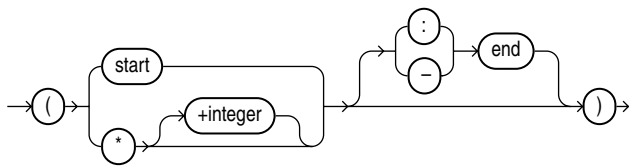


表 6-1 では、位置指定句のパラメータについて説明します。

表 6-1 位置指定句のパラメータ

パラメータ	説明
<code>start</code>	論理レコード中のデータ・フィールドの開始位置です。論理レコードの先頭バイト位置は 1 となります。
<code>end</code>	論理レコード中のデータ・フィールドの終了位置です。 <code>start-end</code> と表記することも、 <code>start:end</code> と表記することもできます。 <code>end</code> を省略した場合、フィールド長は、データ・ファイル中のデータ型から導出されます。 <code>CHAR</code> 型では、 <code>start</code> または <code>end</code> を省略し、長さ指定 ( <code>CHAR(n)</code> ) をしない場合、データ長が 1 として扱われます。データ型から長さを導出できない場合は、エラー・メッセージが出力されます。
<code>*</code>	対象となるデータ・フィールドが前のフィールドの直後にあることを示します。制御ファイル中の最初のデータ・フィールドに対して <code>*</code> を指定した場合、そのフィールドは論理レコードの先頭であると判断されます。位置指定に <code>*</code> を使用した場合、フィールド長はデータ型から導出されます。

表 6-1 位置指定句のパラメータ (続き)

パラメータ	説明
+ <i>INTEGER</i>	+ <i>integer</i> を指定してオフセットを使用すると、前フィールドの終了位置直後の位置から現行のフィールドをオフセットできます。この場合、現行のフィールドの値は、+ <i>integer</i> で指定されたバイト数分スキップした後で読み込まれます。

POSITION を完全に省略することも可能です。省略した場合のデータ・フィールドの位置指定は、POSITION(\*) と指定した場合と同じです。

タブを含むデータでの POSITION の使用

フィールド位置の指定で、データ・ファイル中にタブが含まれている場合は注意が必要です。書式化されたレポートのデータのロード時に SQL\*Loader の拡張 SQL 文字列機能を使用する場合、次のようなエラーが発生することがあります。

- レポートの印刷出力を見て、すべての文字位置を正確に調べ、制御ファイルを作成します。
- このロードは正常に実行されず、「無効数字」および「欠落フィールド」エラーが返されます。

このようなエラーは、データにタブが含まれているときに発生します。紙に出力した場合、各タブの幅は数列分に広がります。ただし、データ・ファイルでは、それぞれのタブは 1 文字のままです。そのため、SQL\*Loader は、データ・ファイルの読み込み時に、誤った POSITION 指定を参照することになります。

この問題を解決するには、データ・ファイル中のタブを探して該当箇所の POSITION 指定を調整するか、フィールドをデリミタで区切ります。

参照： 6-24 ページの「デリミタの指定」を参照してください。

複数表のロードでの POSITION の使用

複数表のロードでは、複数の INTO TABLE 句を指定します。このとき最初の表の最初の列に対して POSITION(\*) を使用すると、論理レコードの先頭から相対的に位置が計算されます。2 番目以降の表の最初の列に対して POSITION(\*) が使用された場合は、その時点で最後にロードされた表の最終列から相対的に位置が計算されます。

したがって、2 番目以降の INTO TABLE 句の処理開始時に、位置が自動的に論理レコードの先頭に設定されるわけではありません。このため、複数の INTO TABLE 句を指定して、同一物理レコード中の異なる箇所を処理できます。例については、5-39 ページの「複数の論理レコードの抽出」を参照してください。

論理レコード中のデータには、2 つの表の両方ではなく、片方の表のみにロードするデータもあります。その場合は、POSITION をリセットする必要があります。このとき、位置指定

を省略するか、または INTO TABLE 句で先頭フィールドに対する POSITION(\*+n) を指定するかわりに、POSITION(1) または POSITION(n) を指定します。

## POSITION を使用した例

```
siteid POSITION (*) SMALLINT
siteloc POSITION (*) INTEGER
```

これらが最初の 2 つの列を指している場合、siteid は 1 列目から始まり、siteloc がその次の列から始まります。

```
ename POSITION (1:20) CHAR
empno POSITION (22-26) INTEGER EXTERNAL
allow POSITION (*+2) INTEGER EXTERNAL TERMINATED BY "/"
```

列 ename は位置 1 ～ 20 を占める文字データで、その次の列 empno は位置 22 ～ 26 を占める数値型データです。列 allow は empno の終了位置直後の位置 (27) から +2 の位置、つまり位置 29 から始まり、スラッシュが検出されるまで継続するデータとなります。

## 列およびフィールドの指定

表の列はいくつでもロードできます。データベース中に定義されていて制御ファイル中で指定されていない列には、NULL 値が割り当てられます。

列指定 (columnspec) には、列名とその列に入る値の指定を記述します。これらの列指定はカンマで区切って、全体を小カッコで囲みます。

```
(columnspec, columnspec, ...)
```

それぞれの列名は、INTO TABLE 句で指定した表中の列名に対応させてください。列名に SQL や SQL\*Loader の予約語または特殊文字が含まれているか、大 / 小文字の区別がある場合は、列名を引用符で囲みます。

列の値を SQL\*Loader で生成する場合は、列指定の中で RECNUM パラメータ、SEQUENCE パラメータ、または CONSTANT パラメータを指定します。詳細は、6-52 ページの「[SQL\\*Loader を使用した入力データの生成](#)」を参照してください。

データ・ファイルから列の値を読み込む場合は、列値に対応するデータ・フィールドを指定します。このとき、列指定 (columnspec) には、データベース表中の列を示す列名 (column name) およびデータ・レコード中のフィールドを示すフィールド指定 (field specification) を指定します。フィールド指定には、フィールドの位置、データ型、NULL 値の制限およびデフォルト値を指定します。

列オブジェクトのロード時に、必ずしもすべての属性を指定する必要はありません。指定しなかった属性には、NULL が設定されます。

## FILLER フィールドの指定

FILLER で指定された FILLER フィールドは、データ・ファイルをマップしたフィールドで、データベースの列と対応しません。FILLER フィールドは、データ・ファイルがマップされているデータ・フィールドから割り当てられた値です。

FILLER フィールドに関しては次のことに注意してください。

- FILLER フィールドの構文は、フィールド名の後ろに FILLER を付けること以外は、列ベースのフィールドと同じです。
- FILLER フィールドには名前がありますが、表にはロードされません。
- FILLER フィールドは、引数として `init_specs`（たとえば、`NULLIF` および `DEFAULTIF`）に使用できます。
- FILLER フィールドは、引数として指示句（たとえば、`SID`、`OID`、`REF` および `BFILE`）に使用できます。
- FILLER フィールドは、`NULLIF` 句、`DEFAULTIF` 句および `WHEN` 句のフィールド条件指定で使用できます。ただし、SQL 文字列では使用できません。
- FILLER フィールドの指定に、`NULLIF` 句または `DEFAULTIF` 句を含めることはできません。
- FILLER フィールドは、`TRAILING NULLCOLS` が指定および適用される場合、`NULL` で初期化されます。他のフィールドが、無効な FILLER フィールドを参照している場合は、エラーになります。
- FILLER フィールドは、オブジェクトのフィールド・リスト内部または `VARRAY` の定義の内部を含む、データ・ファイルのどこにでも指定できます。
- バイナリ配列の FILLER には領域が割り当てられていないため、SQL 文字列を FILLER フィールドの一部としては指定できません。

---

**注意：** この項で説明する内容は、`BOUND FILLER` を使用したバウンド FILLER の指定にも適用されます。唯一の例外として、バイナリ配列のバウンド FILLER には領域が割り当てられているため、このバウンド FILLER を使用して、SQL 文字列をフィールドの一部として指定できます。

---

FILLER フィールド指定の例を次に示します。

```
field_1_count FILLER char,  
field_1 varray count(field_1_count)  
(  
    filler_field1 char(2),  
    field_1 column object  
(  
        attr1 char(2),
```

```
        filler_field2 char(2),
        attr2 char(2),
    )
    filler_field3 char(3),
)
filler_field4 char(6)
```

## データ・フィールドのデータ型の指定

フィールドのデータ型を指定することによって、SQL\*Loader でフィールドのデータが処理される方法が決まります。たとえば、INTEGER データ型を指定するとバイナリ・データとして処理され、INTEGER EXTERNAL を指定すると数字を表す文字データとして処理されます。CHAR を指定したフィールドには、すべての文字データを含むことができます。

各フィールドに対して指定できるデータ型は 1 つのみです。データ型を指定しない場合は、CHAR が想定されます。

SQL\*Loader データ型から Oracle データ型への変換処理および SQL\*Loader の各データ型の詳細は、6-7 ページの「[SQL\\*Loader のデータ型](#)」を参照してください。

データ型を指定する前に、フィールドの位置を指定する必要があります。

## SQL\*Loader のデータ型

SQL\*Loader データ型は、移植可能なデータ型と移植不能なデータ型に分類されます。さらに、これら 2 つのグループは、LENGTH-VALUE データ型および VALUE データ型に分類されます。

移植可能なデータ型および移植不能なデータ型は、データ型のプラットフォーム依存性によって分類されます。プラットフォーム依存性は、異なるプラットフォームのバイト順序スキーマ（ビッグ・エンディアンとリトル・エンディアン）の違い、プラットフォームのビット数（16 ビット、32 ビット、64 ビット）の違い、符号付き数表現のスキーマの違い（2 の補数と 1 の補数）などの様々な理由から存在します。バイト順序スキーマ、プラットフォームのワード長などの場合は、SQL\*Loader で、プラットフォーム依存性を解決するメカニズムが提供されます。これらのメカニズムの詳細は、該当するデータ型の説明を参照してください。

移植可能なデータ型および移植不能なデータ型の両方とも VALUE または LENGTH-VALUE をとることができます。VALUE データ型のデータ・フィールド部分は単一であるとしします。LENGTH-VALUE データ型では、データ・フィールドが 2 つのサブフィールド（length サブフィールドが value サブフィールドの長さを指定）で構成される必要があります。

## 移植不能なデータ型

移植不能なデータ型は、VALUE データ型および LENGTH-VALUE データ型に分類されます。移植不能な VALUE データ型は次のとおりです。

- INTEGER(*n*)
- SMALLINT
- FLOAT
- DOUBLE
- BYTEINT
- ZONED
- (PACKED) DECIMAL

移植不能な LENGTH-VALUE データ型は次のとおりです。

- VARGRAPHIC
- VARCHAR
- VARRAW
- LONG VARRAW

移植不能なデータ型の構文の詳細は、A-9 ページの「[datatype\\_spec](#)」構文図を参照してください。

### INTEGER(*n*)

データは、フルワード 2 進整数 (*n* は、1、2、4 または 8 のオプションで提供された長さ) です。長さが指定されていない場合、その長さはバイト単位で、特定のプラットフォームでの C 言語の LONG INT のサイズに基づいて決まります。

INTEGER は、バイト・サイズ、バイト順序および符号付きの値の表現がシステム間で異なるため、移植不能です。ただし、符号付きの値の表現がシステム間で同じ場合は、SQL\*Loader を使用して、正しい結果で INTEGER データにアクセスできます。長さ指定 (*n*) で INTEGER を指定し、必要に応じて適切な方法でデータのバイト順序を指定すると、SQL\*Loader を使用してシステム間で正しい結果でデータにアクセスできます。長さ指定なしに INTEGER を指定すると、C 言語の LONG INT の長さが両方のシステムで同じバイト数である場合のみ、SQL\*Loader を使用して正しい結果でデータにアクセスできます。その場合も、必要に応じて、適切な方法でデータのバイト順序を指定する必要があります。

2 進整数の長さを明示的に指定すると、ワード長が SQL\*Loader で使用されているものとは異なるプラットフォーム上で入力データを作成する場合に有効です。たとえば、2 進整数を含む入力データは、64 ビットのプラットフォームで作成され、32 ビットのプラットフォーム上の SQL\*Loader を使用しているデータベースにロードされます。この場合、



INTEGER (8) を指定して、SQL\*Loader によってその整数を 4 バイトの量ではなく、8 バイトの量として処理します。

デフォルトでは、INTEGER は SIGNED 量として処理されます。SQL\*Loader で、符号なしの量として処理する場合は、UNSIGNED を指定します。デフォルトの動作に戻すには、SIGNED を指定します。

**参照：** 6-35 ページの「異なるプラットフォーム間でのデータのロード」を参照してください。

## SMALLINT

データはハーフワードの 2 進整数で表現されます。このフィールド長には、使用しているシステムのハーフワード整数の長さが取られます。デフォルトでは、SIGNED 量として処理されます。SQL\*Loader で、符号なしの量として処理する場合は、UNSIGNED を指定します。デフォルトの動作に戻すには、SIGNED を指定します。

SMALLINT は、SHORT INT の長さが同じバイト数のシステム間のみで、正しい結果でロードできます。バイト順序がシステム間で異なる場合は、適切な方法でデータのバイト順序を指定します。詳細は、6-36 ページの「バイト順序」を参照してください。

---

---

**注意：** このデータ型のフィールド長は、C 言語の SHORT INT データ型と同じです。フィールド長を決定する 1 つの方法は、データを入れずに小さい制御ファイルを作成し、その結果のログ・ファイルを調べることです。このデータ型のサイズは、制御ファイルを使用して変更はできません。詳細は、ご使用のオペレーティング・システム固有の Oracle ドキュメントを参照してください。

---

---

## FLOAT

データは単精度浮動小数点 2 進数で表現されます。POSITION 句で end を指定すると end は無視されます。このフィールド長には、システムの単精度浮動小数点 2 進数の長さ（C 言語のデータ型 FLOAT に相当する長さ）が取られます。このデータ型のサイズは、制御ファイルを使用しては変更できません。

FLOAT は、FLOAT の表現に互換性があり、その長さが同じシステム間のみで、正しい結果の出るロードができます。バイト順序が 2 つのシステム間で異なる場合は、適切な方法でデータのバイト順序を指定します。詳細は、6-36 ページの「バイト順序」を参照してください。

## DOUBLE

データは倍精度浮動小数点 2 進数で表現されます。POSITION 句で *end* を指定すると *end* は無視されます。このフィールド長には、使用しているシステムの倍精度浮動小数点 2 進数の長さ（C 言語のデータ型 DOUBLE または LONG FLOAT に相当する長さ）が取られます。このデータ型のサイズは、制御ファイルを使用しては変更できません。

DOUBLE は、DOUBLE の表現に互換性があり、その長さが同じシステム間のみで、正しい結果でロードできます。バイト順序が 2 つのシステム間で異なる場合は、適切な方法でデータのバイト順序を指定します。詳細は、6-36 ページの「[バイト順序](#)」を参照してください。

## BYTEINT

2 進数で表されている 1 バイト分のデータを 10 進数に直した値がロードされます。たとえば、入力文字 `x"1C"` は 28 としてロードされます。BYTEINT フィールドの長さは、常に 1 バイトになります。POSITION(*start:end*) を指定すると、*end* は無視されます。（データ型は、C 言語の UNSIGNED CHAR です。）

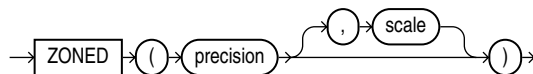
このデータ型の構文の例は次のとおりです。

```
(column1 position(1) BYTEINT,
column2 BYTEINT,
...
)
```

## ZONED

ZONED 型のデータは、ZONED 型の 10 進数形式で表現されます。つまり、10 進数の各 1 桁が 1 バイト（COBOL の SIGN TRAILING フィールドに相当）で表され、最終バイトに符号が入ります。このフィールド長には、精度（桁数）として指定された長さが取られます。

ZONED データ型の構文は次のとおりです。



ここでの *precision* は数字の桁数です。scale（指定されている場合）は（暗黙の）小数点の右側の桁数です。次の例では、位置 32 から始まる 8 桁の整数を表します。

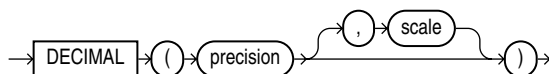
```
sal    POSITION(32)    ZONED(8),
```

Oracle データベース・サーバーでは、ZONED 型のデータが ASCII ベースのプラットフォームで生成される場合、VAX/VMS ZONED 型 10 進数形式を使用します。EBCDIC ベースのプラットフォームで生成される ZONED 型 10 進データのロードも可能です。この場合、Oracle では「ESA/390 操作の原理」で指定されている IBM 形式を使用します。使用される形式は、入力データ・ファイルのキャラクタ・セット・エンコーディングによって異なります。詳細は、5-19 ページの「[CHARACTERSET パラメータ](#)」を参照してください。

## DECIMAL

DECIMAL 型のデータは、PACKED 型の 10 進数形式で記述されます。つまり、10 進数の各 2 桁が 1 バイトで表され、最終バイトに 1 桁と符号が入ります。DECIMAL フィールドでは暗黙の小数点位置を指定できるため、分数の値を表すこともできます。

DECIMAL データ型の構文は次のとおりです。



*precision* パラメータは、数値の桁数です。DECIMAL フィールドのバイト長は、桁数から計算します。 $(N+1)/2$  を求め、その小数点以下を切り上げた値がバイト長となります。

*scale* パラメータは、小数点の右側にくる桁数のことで、スケール変更係数と呼びます。デフォルト値は 0 (ゼロ) です (整数となります)。全体の桁数より大きい数は指定できませんが、負数は指定できません。

次に例を示します。

```
sal DECIMAL (7,2)
```

この例では、+12345.67 の形式の数値がロードされます。このフィールドは、データ・レコード中で 4 バイトを占有します。(DECIMAL フィールドのバイト長は、 $(N+1)/2$  の小数点以下を切り上げた値になります。ここで、N は数値の桁数です。また、1 は符号用として追加されています。)

## VARGRAPHIC

このデータは、可変長のダブルバイト文字列です。length サブフィールドおよびダブルバイト文字 (DBCS) の文字列で構成されます。DBCS は、Oracle データベース・サーバーではサポートされていないため、SQL\*Loader を使用してシングルバイトとして読み込み、RAW データとしてロードします。RAW データ型と同様、VARGRAPHIC フィールドは変更されずに指定の列に格納されます。

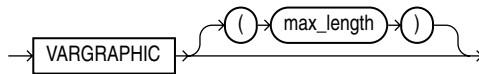
---

**注意：** length サブフィールドのサイズには、システム上の SQL\*Loader の SMALLINT データ型の長さ (C 言語の SHORT INT 型に相当する長さ) が取られます。詳細は、6-9 ページの「[SMALLINT](#)」を参照してください。

---

VARGRAPHIC は、SHORT INT の長さが同じバイト数のシステム間でのみ、正しい結果でロードできます。バイト順序がシステム間で異なる場合は、適切な方法で length サブフィールドのバイト順序を指定します。詳細は、6-36 ページの「[バイト順序](#)」を参照してください。

VARGRAPHIC データ型の構文は次のとおりです。



現行のフィールドの長さは、先頭の 2 バイトで示されます。VARGRAPHIC データ型に指定する最大長 (maximum\_length) には、length サブフィールドの長さは含まれません。この最大長には、グラフィック (ダブルバイト) 文字の文字数を指定します。フィールドのバイト単位の最大長を決定するには、この maximum\_length の値を 2 倍します。

フィールド最大長のデフォルトは、グラフィック文字で 2KB、つまり 4KB (2 × 2KB) です。必要なメモリーを最小限にするには、このような可変フィールドに対して、できるだけ最大長を指定します。

VARGRAPHIC 文の前に位置指定が指定されている場合、(pos\_spec を使用して) グラフィック文字の 1 文字目ではなく、length サブフィールドの位置がわかります。

pos\_spec (start:end) を指定すると、end の位置によってそのフィールドの最大長が決まります。ここでの start や end は、そのファイルにおける 1 バイト単位の文字位置を示します。したがって、(end+1) から start の値を引くと、フィールドの実際のバイト長が求められます。最大長を指定した場合は、その最大長の方が、位置指定から計算された最大長より優先されます。

VARGRAPHIC フィールドのフィールド長全体が、読み込まれる前に論理レコードの終わりで切り捨てられた場合、警告が出力されます。VARGRAPHIC 型のフィールド長は、そのフィールドの各入力データ中に埋め込まれているため、そのフィールド長の方が正確であるとみなされます。

VARGRAPHIC データに対してはデリミタを使用できません。

## VARCHAR

VARCHAR フィールドは、LENGTH-VALUE データ型です。バイナリの length サブフィールドおよびその長さを持つ文字列で構成されます。データ・ファイルに文字長セマンティクスが使用されないかぎり、長さはバイト単位です。文字長セマンティクスが使用される場合は、文字単位になります。詳細は、5-22 ページの「[文字長セマンティクス](#)」を参照してください。

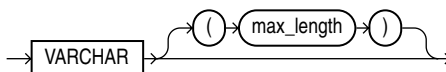
VARCHAR フィールドは、SHORT データ・フィールド INT の長さが同じバイト数のシステム間のみで、正しい結果でロードできます。バイト順序がシステム間で異なる場合、または VARCHAR フィールドに UTF-16 キャラクタ・セットのデータが含まれている場合は、適切な方法で length サブフィールドおよびデータのバイト順序を指定します。データのバイト順序は、UTF-16 キャラクタ・セットに対してのみ問題となります。詳細は、6-36 ページの「[バイト順序](#)」を参照してください。

---

**注意：** length サブフィールドのサイズには、システム上の SQL\*Loader の SMALLINT データ型の長さ (C 言語の SHORT INT 型に相当する長さ) が取られます。詳細は、6-9 ページの「[SMALLINT](#)」を参照してください。

---

VARCHAR データ型の構文は次のとおりです。



制御ファイルに指定する最大長 (maximum\_length) には、length サブフィールドのサイズを含むことはできません。VARCHAR データ型にオプションで最大長を指定すると、そのサイズ分のバッファがこのフィールドに対してバイト単位で割り当てられます。ただし、文字長セマンティクスがデータ・ファイルに使用される場合、バイト単位のバッファ・サイズは、キャラクタ・セット中の最大限の文字のバイト単位のサイズの max\_length 倍となります。詳細は、5-22 ページの「[文字長セマンティクス](#)」を参照してください。

デフォルトの最大サイズは 4KB です。データのロードに必要な最小限の値を最大値として指定することによって、SQL\*Loader で使用されるメモリーを最小限に抑えることができます。特に、VARCHAR フィールドを多数使用する場合有効です。

POSITION 句を使用する場合、指定する位置は、テキスト文字の先頭ではなく、length サブフィールドのバイト単位の位置になります。POSITION (start:end) と指定すると、end の位置によってそのフィールドの最大長が決まります。したがって、(end + 1) から start の値を引くと、フィールドの実際のバイト長が求められます。最大長を指定した場合は、その最大長の方が POSITION 句から計算された長さよりも優先されます。

VARCHAR フィールドのフィールド長全体が読み込まれる前に、論理レコードの終わりで切り捨てられた場合、警告が出力されます。VARGRAPHIC 型のフィールド長は、そのフィールドの各入力データ中に埋め込まれているため、そのフィールド長の方が正確であるとみなされます。

VARGRAPHIC データに対してはデリミタを使用できません。

## VARRAW

VARRAW は、2 バイトのバイナリの length サブフィールドおよびその後続く RAW 文字列の VALUE サブフィールドで構成されています。

デフォルトでは、VARRAW は、length サブフィールドが 2 バイトで、最大サイズが 4KB の VARRAW になります。VARRAW (65000) は、length サブフィールドが 2 バイトで、最大サイズが 65000 バイトの VARRAW になります。

VARRAW フィールドは、適切な方法で length サブフィールドのバイト順序を指定すると、異なるバイト順序のシステム間でロードできます。詳細は、6-36 ページの「[バイト順序](#)」を参照してください。

## LONG VARRAW

LONG VARRAW は、2 バイトの length サブフィールドではなく、4 バイトの length サブフィールドを持つ VARRAW です。

デフォルトでは、LONG VARRAW は、length サブフィールドが 4 バイトで、最大サイズが 4KB の VARRAW になります。LONG VARRAW(300000) は、length サブフィールドが 4 バイトで、最大サイズが 300000 バイトの VARRAW になります。

LONG VARRAW フィールドは、適切な方法で length サブフィールドのバイト順序を指定すると、異なるバイト順序のシステム間でロードできます。詳細は、6-36 ページの「[バイト順序](#)」を参照してください。

## 移植可能なデータ型

移植可能なデータ型は、VALUE データ型および LENGTH-VALUE データ型に分類されます。移植可能な VALUE データ型は次のとおりです。

- CHAR
- 日時データ型および期間データ型
- GRAPHIC
- GRAPHIC EXTERNAL
- 数値型 EXTERNAL (INTEGER、FLOAT、DECIMAL および ZONED)
- RAW

移植可能な LENGTH-VALUE データ型は次のとおりです。

- VARCHARC
- VARRAWC

移植可能なデータ型の構文は、A-9 ページの「[datatype\\_spec](#)」の構文図を参照してください。

文字データ型には、CHAR 型、DATE 型および数値型 EXTERNAL 型があります。これらのフィールドにはデリミタを使用できます。また、制御ファイルにフィールド長（または最大長）を指定することができます。

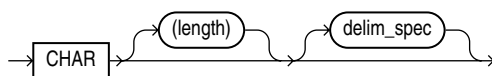
## CHAR

このデータ・フィールドには、文字データが入ります。データ長には最大長を指定します（オプション）。データ長については、次のことにも注意してください。

- データ長が指定されていない場合、データ長は POSITION 句の指定から導出されます。
- データ長が指定されている場合は、POSITION 句で指定したデータ長より優先されます。

- データ長も位置も指定されていない場合、フィールドが区切られていないかぎり、CHAR のデータ長は 1 文字とみなされます。
  - デリミタ付き CHAR フィールドに長さが指定されている場合は、その長さが最大長として使用されます。
  - デリミタ付き CHAR フィールドに長さが指定されていない場合は、デフォルトの 255 バイトが最大長として使用されます。
  - デリミタ付きで 255 バイトを超える CHAR フィールドの場合、最大長を指定する必要があります。そうしない場合、データ・ファイルのフィールドが最大長を超えているというエラーを受信します。

CHAR データ型の構文は次のとおりです。



**参照：** 6-24 ページの「デリミタの指定」を参照してください。

## 日時データ型および期間データ型

日時データ型には次のものがあります。

- DATE
- TIME
- TIMESTAMP
- TIME WITH TIME ZONE
- TIMESTAMP WITH TIME ZONE

日時データ型の値は、**Datetimes** と呼ばれる場合があります。

期間データ型には次のものがあります。

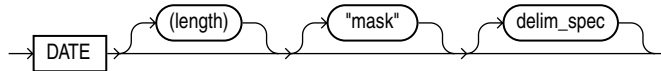
- INTERVAL YEAR TO MONTH
- INTERVAL DAY TO SECOND

期間データ型の値は、**Intervals** と呼ばれる場合があります。

日時および期間の両方ともフィールドで構成されています。これらのフィールドの値によってデータ型の値が決定されます。

**参照：** 日時データ型および期間データ型の詳細は、『Oracle9i SQL リファレンス』を参照してください。

**DATE** DATE フィールドには文字データが入り、その文字データは、指定された日付マスクを使用して Oracle の日付に変換されます。DATE フィールドの構文は次のとおりです。



次に例を示します。

```

LOAD DATA
INTO TABLE dates (col_a POSITION (1:15) DATE "DD-Mon-YYYY")
BEGINDATA
1-Jan-1991
1-Apr-1991 28-Feb-1991

```

デリミタがない場合、空白は無視され、日付は左から右に構文解析されます (DATE フィールドのデータがすべて空白の場合、NULL フィールドとしてロードされます)。

可変長の日付マスクを指定していない場合、データ長の指定はオプションになります。データ・ファイルに文字長セマンティクスが使用されないかぎり、長さはバイト単位です。文字長セマンティクスが使用される場合は、文字単位になります。詳細は、5-22 ページの「[文字長セマンティクス](#)」を参照してください。

前述の例では、日付マスク "DD-Mon-YYYY" は、バイト長セマンティクスで 11 バイトあります。そのため、SQL\*Loader によってこのフィールドの最大文字数が 11 文字とみなされ、前述の指定は正しく処理されます。ただし、次のように指定される場合は注意が必要です。

```
DATE "Month dd, YYYY"
```

この場合、日付マスクは 14 バイトです。"September 30, 1991" などのように 14 バイトを超える長さの値が指定されている場合は、長さを指定する必要があります。

同様に、ユリウス日 (日付マスク「J」) の場合も長さを指定する必要があります。日付文字列の長さがマスクの長さ (マスク内のバイト数) をを超える可能性がある場合は、必ずフィールド長を指定してください。

長さを明示的に指定していない場合は、POSITION 句から長さが求められます。マスクを使用するときは、データ長がマスクの長さ以下であることが確実にないかぎり、常に長さを指定してください。

長さを明示的に指定した場合、この長さは、POSITION 句で指定された長さよりも優先されます。これらはいずれも、マスクから求められる長さより優先されます。マスクについては、Oracle 日付マスクとして有効なものを指定します。マスクの指定を省略すると、デフォルトの Oracle 日付マスク「dd-mon-yy」が使用されます。

データ長は小カッコで囲み、マスクは引用符で囲む必要があります。DATE データ型の使用例については、10-12 ページの「[事例 3: 自由区分形式ファイルのロード](#)」を参照してください。



DATE 型のフィールドでは、デリミタも使用できます。詳細は、6-24 ページの「[デリミタの指定](#)」を参照してください。

**TIME** TIME データ型には、時、分、秒の値が格納されます。次に例を示します。

```
09:26:50
```

**TIMESTAMP** TIMESTAMP データ型は、DATE データ型の拡張です。DATE データ型の年、月、日に加えて、TIME データ型の時、分、秒の値を格納します。TIMESTAMP データ型の例を次に示します。

```
TIMESTAMP '1999-01-31 09:26:50'
```

日付値を指定する場合、時間構成要素を指定しないと、デフォルト時間の 12:00:00 AM（真夜中）が採用されます。

**TIME WITH TIME ZONE** TIME WITH TIME ZONE データ型は、タイム・ゾーンの置換えを含む TIME データ型の変形です。タイム・ゾーンの置換えは、ローカル時刻と UTC の差（時および分）です。

LOCAL オプションが指定されている場合、データベースに格納されているデータはデータベース・タイム・ゾーンに指定されます。データが取得されると、ユーザーのローカル・セッション・タイム・ゾーンに返されます。

**TIMESTAMP WITH TIME ZONE** TIMESTAMP WITH TIME ZONE データ型は、タイム・ゾーンの置換えを含む TIMESTAMP データ型の変形です。タイム・ゾーンの置換えは、ローカル時刻と UTC の差（時および分）です。

LOCAL オプションが指定されている場合、データベースに格納されているデータはデータベース・タイム・ゾーンに指定されます。データが取得されると、ユーザーのローカル・セッション・タイム・ゾーンに返されます。

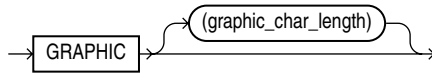
**INTERVAL YEAR TO MONTH** INTERVAL YEAR TO MONTH データ型は、YEAR および MONTH 日時フィールドを使用して一定期間を格納します。

**INTERVAL DAY TO SECOND** INTERVAL DAY TO SECOND データ型は、DAY および SECOND 日時フィールドを使用して一定期間を格納します。

## GRAPHIC

このデータは、ダブルバイト文字（DBCS）の文字列データです。Oracle データベース・サーバーでは DBCS はサポートされていないため、SQL\*Loader は DBCS を 1 バイトずつ読み込みます。RAW データ型と同様、GRAPHIC フィールドは変更されずに指定の列に格納されます。

GRAPHIC データ型の構文は次のとおりです。



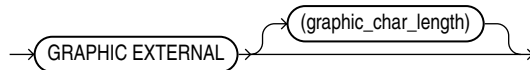
GRAPHIC 型および GRAPHIC EXTERNAL 型では、POSITION (*start:end*) を指定すると、論理レコードにおけるフィールドの正確な位置が決まります。

ただし、GRAPHIC (EXTERNAL) データ型にデータ長を指定する場合は、ダブルバイト・グラフィック文字の文字数を指定します。この値を 2 倍してフィールドのバイト長が求められます。グラフィック文字の長さを指定した場合は、POSITION 句から求められたデータ長は無視されます。GRAPHIC データ型の指定では、データ・フィールドの区切りの指定はできません。

## GRAPHIC EXTERNAL

DBCS フィールドがシフトイン / シフトアウト文字で囲まれている場合は、GRAPHIC EXTERNAL 型を使用します。このデータ型は、データの先頭と最後の文字（シフトイン / シフトアウト文字）がロードされないことを除き、GRAPHIC 型とほぼ同じです。

GRAPHIC EXTERNAL データ型の構文は次のとおりです。



GRAPHIC は、ダブルバイト文字のデータであることを示します。EXTERNAL は、先頭と最後の文字が無視されることを示します。graphic\_char\_length の値は、DBCS のデータ長を指定します（6-18 ページの [GRAPHIC](#) を参照）。

たとえば、[] をシフトイン / シフトアウト文字とし、# を任意のダブルバイト文字とします。

#### を表現する場合、POSITION (1:4) GRAPHIC または POSITION (1) GRAPHIC (2) と指定します。

[####] を表現する場合、POSITION (1:6) GRAPHIC EXTERNAL または POSITION (1) GRAPHIC EXTERNAL (2) と指定します。

## 数値型 EXTERNAL

数値型 EXTERNAL データ型は、数値データ型（INTEGER、FLOAT、DECIMAL および ZONED）に EXTERNAL、オプションのデータ長およびデリミタ指定を指定したものです。データ・ファイルに文字長セマンティクスが使用されないかぎり、長さはバイト単位です。文字長セマンティクスが使用される場合は、文字単位になります。詳細は、5-22 ページの「[文字長セマンティクス](#)」を参照してください。

このデータ型は、判読可能な文字形式の数値データです。長さ、位置およびデリミタについては、CHAR データと同じ規則が数値型 EXTERNAL にも適用されます。これらの規則の詳細は、6-14 ページの「[CHAR](#)」を参照してください。

数値型 EXTERNAL データ型の構文の詳細は、A-9 ページの「[datatype\\_spec](#)」を参照してください。

---

**注意：** このデータは、バイナリ表現ではなく、文字形式の数字になります。したがって、これらのデータ型の処理方法は、DEFAULTIF を使用する場合を除き、CHAR と同じです。デフォルトを NULL にする場合は CHAR を使用します。デフォルトを 0（ゼロ）にする場合は EXTERNAL を使用します。詳細は、6-32 ページの「[WHEN、NULLIF および DEFAULTIF 句の使用](#)」を参照してください。

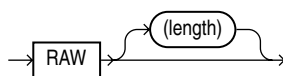
---

FLOAT EXTERNAL データは科学表記法または通常表記法のどちらででも指定できます。「5.33」と「533E-2」は両方とも同じ値の正しい表現です。

## RAW

RAW 型のバイナリ・データが無条件で RAW 型のデータベース列にロードされた場合、Oracle データベース・サーバーによるデータ変換は行われません。CHAR 列にロードした場合は、Oracle によって、16 進数にデータ変換されます。DATE 型や数値型の列にはロードできません。

RAW データ型の構文は次のとおりです。



ここで、length には制御ファイルに指定されたバイト数を指定します。この長さは、データベース中のターゲット列の長さでメモリ・リソースの許容範囲内であれば自由に指定できます。データ・ファイルに文字長セマンティクスが使用される場合でも、長さは常にバイト単位です。RAW データ・フィールドに対してはデリミタは使用できません。

## VARCHARC

VARCHARC データ型は、文字の `length` サブフィールドおよびその後続く文字列 `VALUE` サブフィールドで構成されます。

VARCHARC に対する宣言には、`length` サブフィールドの長さが含まれます。また、オプションで文字列の最大サイズがその後続く場合もあります。データ・ファイルにバイト長セマンティクスが使用される場合、長さおよび最大サイズは両方ともバイト単位です。データ・ファイルに文字長セマンティクスが使用される場合、長さおよび最大サイズは両方とも文字単位です。最大サイズが指定されていない場合、バイト長セマンティクスまたは文字長セマンティクスのいずれが使用されていても、デフォルトは **4KB** です。

次に例を示します。

- 少なくとも長さサブフィールドに値を指定する必要があるため、VARCHARC はエラーになります。
- データ・ファイルにバイト長セマンティクスが使用されている場合、VARCHARC (7) は、`length` サブフィールドが 7 バイトで、最大サイズが **4KB** (デフォルト) の VARCHARC になります。文字長セマンティクスが使用されている場合は、`length` サブフィールドが 7 文字で、最大サイズが **4KB** (デフォルト) の VARCHARC になります。最大サイズが指定されていない場合、バイト長セマンティクスまたは文字長セマンティクスのいずれが使用されていても、常にデフォルトの **4KB** が使用されることに注意してください。
- データ・ファイルにバイト長セマンティクスが使用されている場合、VARCHARC (3,500) は、`length` サブフィールドが 3 バイトで、最大サイズが 500 バイトの VARCHARC になります。文字長セマンティクスが使用されている場合は、`length` サブフィールドが 3 文字で、最大サイズが 500 文字の VARCHARC になります。

詳細は、5-22 ページの「[文字長セマンティクス](#)」を参照してください。

## VARRAWC

VARRAWC データ型は、RAW 文字列の `VALUE` サブフィールドで構成されています。

次に例を示します。

- VARRAWC はエラーになります。
- VARRAWC (7) は、`length` サブフィールドが 7 バイトで、最大サイズが **4KB** (デフォルト) の VARRAWC になります。
- VARCHARC (3,500) は、`length` サブフィールドが 3 バイトで、最大サイズが 500 バイトの VARRAWC になります。

### システム固有のデータ型フィールド長の競合

フィールド長を指定する方法は数通りあります。それぞれの指定方法で異なる値を指定して、値が競合する場合は、そのうちの 1 つの値が優先されます。競合が発生した時点で警告が出されます。どのフィールド長を採用するかは、次の規則に基づいて決定されます。

- 1. POSITION 句で指定されたバイト数に関係なく、SMALLINT、FLOAT および DOUBLE のデータ・サイズは固定長です。
- 2. DECIMAL、INTEGER、ZONED、GRAPHIC、GRAPHIC EXTERNAL または RAW で指定されたフィールド長（または精度）が、POSITION (start:end) から計算されたサイズと異なる場合は、指定されたフィールド長（または精度）を採用します。
- 3. 文字フィールドまたは VARGRAPHIC フィールドにおいて指定された最大長が、POSITION (start:end) から計算されたフィールド長と異なる場合は、指定された最大長を採用します。

たとえば、システム固有のデータ型 INTEGER が 4 バイトであるときに、次のようなフィールドが指定されたとします。

```
column1 POSITION(1:6) INTEGER
```

この場合、警告が出力され、正しいフィールド長である 4 バイトが採用されます。実際に使用されたフィールド長は、ログ・ファイル内の列表の「Len」という見出しの箇所に記録されます。

Column Name	Position	Len	Term	Encl	Datatype
COLUMN1	1:6	4			INTEGER

### LENGTH-VALUE データ型のフィールド長

制御ファイルで、LENGTH-VALUE データ型の最大長（VARCHAR、VARCHARC、VARGRAPHIC、VARRAW および VARRAWC）を指定できます。指定される最大長は、フィールドにバイト長セマンティクスが使用される場合はバイト単位で、文字長セマンティクスが使用される場合は文字単位です。最大長が指定されていない場合は、デフォルトで 4096 バイトとなります。フィールド長が最大長を超える場合、レコードは拒否され、次のエラーが表示されます。

可変長フィールドが最大長を超えています。

## データ型の変換

制御ファイルに指定したデータ型で、データ・ファイル中のデータをどのように解釈するかを、SQL\*Loader に対して指定します。一方、サーバーでは、これとは別にデータベース中の列に対してデータ型を定義します。これらの 2 つのデータ型を対応付ける手がかりとなるのが、制御ファイル中に指定している列名です。

SQL\*Loader では、入力ファイル中のフィールドからデータが抽出されます。抽出処理は、制御ファイルに指定したデータ型に基づいて行われます。次に SQL\*Loader では、そのフィールドがサーバーに送信されます。送信されたフィールドは、該当する列に（行挿入配列の一部として）格納されます。

SQL\*Loader またはサーバーでは、変換の必要なデータに対してデータ変換が行われ、適切な内部形式でデータが格納されます。これには、データ・ファイルのキャラクタ・セットとデータベースのキャラクタ・セットが異なる場合に、データ・ファイルのデータをデータベース用に変換する機能も含まれます。

入力ファイル中のデータ型は、Oracle 表における列のデータ型に一致している必要はありません。データ型が一致していない場合、Oracle データベース・サーバーで自動的に変換が行われます。ただし、変換が正常に実行されたか、エラーは発生していないかについては実行後に確認する必要があります。たとえば、データ・ファイルでは CHAR 型であるフィールドを、NUMBER 型のデータベース列にロードしたとします。この場合、その文字フィールドの値が有効な数値となっているかどうかを必ず確認してください。

---

**注意：** SQL\*Loader には、NUMBER または VARCHAR2 などの Oracle 内部データ型に関するデータ型指定は定義されていません。SQL\*Loader のデータ型として扱えるのは、テキスト・エディタで作成できるデータ（文字データ型）、および標準プログラミング言語で作成できるデータ（システム固有のデータ型）のみです。ただし、NUMBER および VARCHAR2 のようなデータ型は、SQL\*Loader では認識されませんが、これらのデータ型またはその他のデータ型のデータベース列に、Oracle データベース・サーバーで変換可能なデータをロードすることはできます。

---

## 日時データ型および期間データ型のデータ型変換

表 6-2 に、Oracle データベース・データ型と SQL\*Loader 制御ファイルの日時データ型および期間データ型の間でサポートされている変換およびサポートされていない変換を示します。

この表で使用されている Oracle データベース・データ型の略称は次のとおりです。

- N: NUMBER
- C: CHAR または VARCHAR2
- D: DATE
- T: TIME および TIME WITH TIME ZONE

TS: TIMESTAMP および TIMESTAMP WITH TIME ZONE

YM: INTERVAL YEAR TO MONTH

DS: INTERVAL DAY TO SECOND

SQL\*Loader データ型でも、D、T、TS、YM および DS に関しては、表の略称の定義は同じです。ただし、前述のとおり、SQL\*Loader には、NUMBER、CHAR、VARCHAR2 などの Oracle 内部データ型に関するデータ型指定は定義されていません。ただし、Oracle データベース・サーバーで変換可能なデータは、これらのデータ型やその他のデータ型のデータベース列にロードできます。

この表の読み方の例を、SQL\*Loader データ型 DATE（略称 D）の行で示します。行全体を見ると、Oracle データベース・データ型の CHAR、VARCHAR2、DATE、TIMESTAMP および TIMESTAMP WITH TIMEZONE データ型に対してデータ型変換がサポートされていることを確認できます。ただし、Oracle データベース・データ型の NUMBER、TIME、TIME WITH TIME ZONE、INTERVAL YEAR TO MONTH または INTERVAL DAY TO SECOND データ型に対しては変換がサポートされていません。

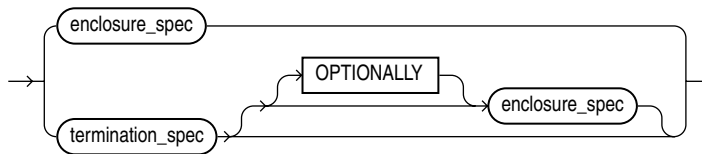
表 6-2 日時データ型および期間データ型のデータ型変換

SQL*Loader のデータ型	Oracle データベース・データ型（変換のサポート）
N	N（可）、C（可）、D（不可）、T（不可）、TS（不可）、YM（不可）、DS（不可）
C	N（可）、C（可）、D（可）、T（可）、TS（可）、YM（可）、DS（可）
D	N（不可）、C（可）、D（可）、T（不可）、TS（可）、YM（不可）、DS（不可）
T	N（不可）、C（可）、D（不可）、T（可）、TS（可）、YM（不可）、DS（不可）
TS	N（不可）、C（可）、D（可）、T（可）、TS（可）、YM（不可）、DS（不可）
YM	N（不可）、C（可）、D（不可）、T（不可）、TS（不可）、YM（可）、DS（不可）
DS	N（不可）、C（可）、D（不可）、T（不可）、TS（不可）、YM（不可）、DS（可）

## デリミタの指定

CHAR、日時、期間または数値型 EXTERNAL 型のフィールドの境界は、特定のデリミタ文字を使用して、入力データ・レコード中に指定することもできます。RAW データ型でもデリミタを使用できます。ただし、RAW データ型が入力 LOBFILE にある場合、およびデリミタが TERMINATED BY EOF（ファイルの終わり）の場合のみです。データ型指定の後にデリミタを指定して、フィールドをどのように区切るかを指定します。

次の構文に示すとおり、デリミタ付きデータは、終了デリミタまたは囲みデリミタで区切られます。



デリミタの指定には、TERMINATED BY 句または ENCLOSED BY 句（あるいはその両方）も使用できます。両方とも指定する場合は TERMINATED BY 句を先に指定してください。

### TERMINATED フィールド

TERMINATED フィールドには、フィールドの開始位置から最初のデリミタ文字までのデータが読み込まれます（デリミタ文字自体は読み込まれません）。終了デリミタが最初の列位置にあれば、そのフィールドは NULL となります。

TERMINATED BY WHITESPACE を指定すると、最初に空白文字（スペース、タブ、空白、LF、改ページまたは改行）が現れるまでデータが読み込まれます。空白文字が現れると、次に空白以外の文字が現れるまで連続する空白文字は読み込まれません。したがって、フィールド値の間に入る空白は、いくつあってもかまいません。この構文の詳細は、6-25 ページの「[終了および囲みの指定に関する構文](#)」を参照してください。

### ENCLOSED フィールド

ENCLOSED フィールドの読み込みでは、空白以外の文字が現れるまで、空白文字はスキップされます。このとき、現れた空白以外の文字がデリミタの場合は、次のデリミタまでのデータが読み込まれます。現れた空白以外の文字がデリミタでない場合は、エラーとなります。

デリミタ文字が 2 つ続けて現れた場合は、1 つのデリミタ文字のみがデータ値の一部として扱われます。たとえば 'DON'T' は、DON'T として格納されます。ただし、フィールドに 2 つのデリミタのみ含まれている場合は NULL 値となります。この構文の詳細は、6-25 ページの「[終了および囲みの指定に関する構文](#)」を参照してください。



終了および囲みの指定に関する構文

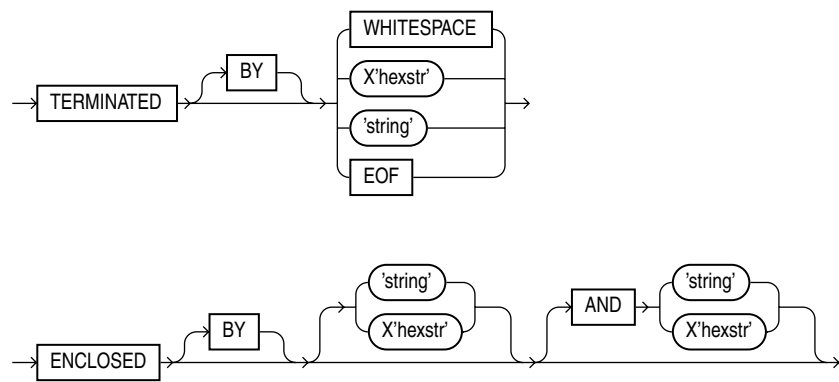


表 6-3 では、終了および囲みの指定に関する構文について説明します。

表 6-3 終了および囲みの指定に関するパラメータ

パラメータ	説明
TERMINATED	データは、最初にデリミタが現れるまで読み込まれます。
BY	可読性を向上させるために使用されるオプションのワードです。
WHITESPACE	デリミタにはスペース、タブ、空白、LF、改ページまたは改行を含むあらゆる空白文字を使用できます（ただし、使用できるのは TERMINATED のみで、ENCLOSED には使用できません）。
OPTIONALLY	指定する文字でデータを囲むこともできます。SQL*Loader では、この指定文字が最初に現れたところから、次に同じ文字が現れたところまでのデータ値が読み込まれます。データが囲まれていない場合は、終了デリミタ付きのフィールドとして読み込まれます。オプションで囲みデリミタを指定する場合は、必ず TERMINATED BY 句を指定してください。その場合、フィールド定義の一部としてローカルに指定しても、FIELDS 句の中でグローバルに指定してもかまいません。
ENCLOSED	データは 2 つのデリミタで囲まれます。
string	デリミタは文字列です。
X'hexstr'	デリミタには 1 文字を指定しますが、ここでは文字コード体系における X'hexstr' によって指定される値で、文字を指定します。たとえば、X'1F'（10 進数の 31）などのように指定します。「X」は大文字でも小文字でも使用できます。

表 6-3 終了および囲みの指定に関するパラメータ（続き）

パラメータ	説明
AND	後続の囲みデリミタを指定する場合に使用します。後続の囲みデリミタには、先頭の囲みデリミタとは異なる文字を指定できます。AND 句を指定しないと、先頭の囲みデリミタと後続の囲みデリミタは同じ文字とみなされます。
EOF	ファイル全体が LOB にロードされたことを示します。これは、LOB ファイルからデータがロードされる場合のみ有効です。EOF によって終了するフィールドは囲むことができません。

各指定方法によるデリミタの指定例およびそれぞれの場合の実際のデータの例を示します。

TERMINATED BY ','	a data string,
ENCLOSED BY '''	"a data string"
TERMINATED BY ',' ENCLOSED BY '''	"a data string",
ENCLOSED BY '(' AND ')'	(a data string)

データ中のデリミタ記号

デリミタとして定義した句読点を、データの中でも使用する必要があります。このような場合、デリミタ文字を 2 つ続けて記述すると、この文字は 1 文字のみ指定されたものと解釈され、データの一部として組み込まれます。たとえば、データベースに次の文字列を格納するとします。

(The delimiters are left parentheses, (, and right parentheses, ).)

フィールド指定は次のようにします。

ENCLOSED BY "(" AND ")"

この場合、データベースには次の文字列が格納されます。

The delimiters are left parentheses, (, and right parentheses, ).

このため、隣接するフィールドが同じデリミタを使用すると、問題が発生します。たとえば、次のように指定されている場合、

field1 TERMINATED BY "/"  
field2 ENCLOSED BY "/"

次のデータは正しく解釈されます。

This is the first string/        /This is the second string/

ただし、field1 および field2 が次のように隣接している場合、誤った処理が行われま  
す。

```
This is the first string//This is the second string/
```

この場合、このデータ全体が、中央に 1 つの「/」のみを持つ単一の文字列とみなされ、field1 に属するものと解釈されてしまいます。

## デリミタ付きデータの最大長

デリミタ付きデータの最大長のデフォルトは、255 バイトです。したがって、デリミタ付きフィールドでは、バインド配列に対して記憶域が大量に使用される場合があります。フィールドが 255 バイトより短い場合、最大長にはできるだけ小さい値を指定してください。フィールドが 255 バイトより長い場合は、フィールド長指定子または POSITION 句を使用して、フィールドに最大長を指定する必要があります。

## デリミタを使用した後続の空白のロード

後続の空白は、PRESERVE BLANKS を指定しないかぎり、デリミタなしのデータ型ではロードされません。たとえば、データ・フィールド長が 9 文字で、DANIELbbb という値のデータがあるとしします。ここでの bbb は 3 つの空白を示します。このとき、CHAR(9) と宣言されていると、Oracle データベースには「DANIEL」がロードされます。

この例で後続の空白も必要な場合は、CHAR(9) TERMINATED BY ':' と宣言し、さらにデータ・ファイルにコロンを追加してフィールドを DANIELbbb: とします。このフィールドは、後続の空白とともに、「DANIEL    」 としてロードされます。TERMINATED BY 句を使用しないで PRESERVE BLANKS を指定し、同じ結果を得ることもできます。

**参照：** 次の項を参照してください。

- 6-41 ページの「空白の切捨て」
- 6-47 ページの「空白の保存」

## 文字データ型フィールド長の競合

CHAR 型、DATE 型、数値型 EXTERNAL 型の文字データ型の場合は、そのフィールド長を制御ファイルに複数指定できます。複数指定したときの長さが異なり、値が競合する場合は、そのうちの 1 つが優先されます。競合が発生すると、警告が出力されます。この項では、指定された長さのうちのどれが優先されるかについて説明します。

## 事前にサイズが決まっているフィールド

前述のデータ型のフィールドに対して開始位置と終了位置を指定すると、そのフィールド長は指定された開始 / 終了位置から求められます。データ型指定の中で長さを指定し、終了位置は指定しない場合、データ型指定の中で指定された長さがそのフィールド長になります。開始位置、終了位置および長さがすべて指定されていて、その長さが異なる場合は、次のように、データ型指定の中で指定されている長さがフィールド長として使用されます。

POSITION(1:10) CHAR(15)

この場合、このフィールド長は 15 になります。

## デリミタ付きフィールド

デリミタ付きフィールドに長さを指定した場合、または開始位置と終了位置から長さを計算できる場合は、その長さがフィールドの最大長となります。指定される最大長は、フィールドにバイト長セマンティクスが使用される場合はバイト単位で、文字長セマンティクスが使用される場合は文字単位です。長さが指定されていない場合、または開始位置と終了位置から長さが計算できない場合は、最大長のデフォルトは 255 バイトです。実際の長さはデリミタの位置によって変わりますが、長さの上限はこの最大長の値となります。

フィールドにデリミタのみでなく、開始位置および終了位置が指定されている場合は、位置指定のみが影響します。囲みデリミタまたは終了デリミタは無視されます。

デリミタが見つからない場合は、レコードの終わりがフィールドの終端となります。TRAILING NULLCOLS が指定されている場合は、残りのフィールドには NULL 値が設定されます。デリミタまたはレコードの終端で区切った結果、フィールド長が最大長よりも大きくなる場合は、SQL\*Loader によってレコードが拒否され、エラーが返されます。

## 日付フィールド・マスク

マスクを指定した場合、日付フィールド長は、使用するマスクによって異なります。指定されたマスクによって形式が決定され、SQL\*Loader ではその形式に基づいてレコード中のデータが解釈されます。たとえば、次のようなマスクを指定したとします。

```
"Month dd, yyyy"
```

この場合、「May 3, 1991」はレコード（バイト長セマンティクスを含む）中で 11 バイトを占有し、「January 31, 1992」は 16 バイトを占有することになります。

ただし、開始位置および終了位置を指定すると、この位置指定から計算されるフィールド長は、マスクから求められるフィールド長よりも優先されます。DATE(12) のようにフィールド長が指定された場合は、このフィールド長が最優先となります。日付フィールドが、終了デリミタまたは囲みデリミタでも区切られている場合は、制御ファイル中で指定された長さがそのフィールドの最大長と解釈されます。

**参照：** DATE フィールドの詳細は、6-15 ページの「[日時データ型および期間データ型](#)」を参照してください。

## フィールド条件の指定

フィールド条件とは、論理レコード中のフィールドに関して、それが真か偽かを評価する条件を記述したものです。WHEN 句の他、NULLIF 句や DEFAULTIF 句の中で使用します。

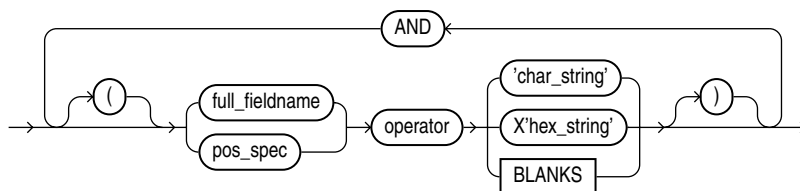
フィールド条件は、CONTINUEIF 句の中で指定する条件と同様ですが、次の 2 つの点で異なります。第 1 に、フィールド条件で指定する位置は、物理レコードではなく論理レコードの位置を示します。第 2 に、論理レコード内の位置またはデータ・ファイル内のフィールド名 (FILLER フィールドを含む) のいずれかを指定できます。

---

**注意：** フィールド条件は、SDF のフィールドに基づくことができません。

---

field\_condition 句の構文は次のとおりです。



pos\_spec 句の構文は次のとおりです。

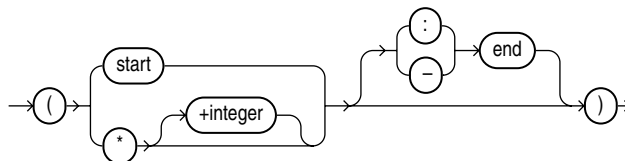


表 6-4 では、フィールド条件句のパラメータについて説明します。位置指定パラメータの詳細は、表 6-1 を参照してください。

表 6-4 フィールド条件句のパラメータ

パラメータ	説明
pos_spec	<p>論理レコード中の比較対象フィールドの開始および終了位置です。それらの位置は小カッコで囲んでください。<i>start-end</i>と表記することも、<i>start:end</i>と表記することもできます。</p> <p>開始位置の指定は、列番号、*（次の列）または *+n（次の列にオフセット分を加算）の形式で指定できます。</p> <p>終了位置を省略した場合、フィールド長は、比較文字列の長さから判断されます。対象フィールドと比較文字列の長さが異なるときは、短い方を埋めるために文字列が追加されます。文字列の場合は空白が追加され、16 進数のバイト列の場合は 0（ゼロ）が追加されます。</p>
start	論理レコード中の比較対象フィールドの開始位置です。
end	論理レコード中の比較対象フィールドの終了位置です。
full_fieldname	<p><i>full_fieldname</i> には、ドット表記法を使用してフィールドのフルネームを指定します。フィールド <i>col2</i> が列オブジェクト <i>col1</i> の属性の場合、指示句の中で <i>col2</i> を参照するときは、<i>col1.col2</i> と表記してください。同じエンティティを参照および命名している列名およびフィールド名があっても、列名には、エンティティのフルネームを指定できない（ドット表記法がない）ため、別のものとして認識されます。</p>
operator	比較演算子として、等価または不等価を示す記号を指定します。
char_string	比較フィールドとの比較に使用する文字列で、一重引用符または二重引用符で囲んで指定します。比較の結果が真の場合は、現在のレコードが表に挿入されます。
X'hex_string'	16 進数の文字列で、16 進数 2 桁がフィールドの 1 バイトに相当します。一重引用符または二重引用符で囲まれます。比較の結果が真の場合は、現在のレコードが表に挿入されます。
BLANKS	フィールドが完全に空白かどうかをテストできます。BLANKS の指定は、デリミタ付きデータのロード時にフィールド長が予測できない場合、または空白が複数あるマルチバイト・キャラクタ・セットを使用する場合に必要となります。

## フィールドと BLANKS の比較

BLANKS パラメータを使用すると、長さが不明なフィールドのデータが空白かどうかを知ることができます。

次の指定を実行すると、空白のフィールドに NULL 値を設定できます。

```
full_fieldname ... NULLIF column_name=BLANKS
```

BLANKS パラメータが認識できるのは空白のみです。タブは認識できません。BLANKS は、どのようなフィールド比較の場合でも、比較文字列のかわりに指定できます。列の値がすべて空白のときにのみ条件が真となります。

BLANKS は、固定長フィールドに対しても指定できます。その場合は、対象フィールドに合った長さの空白文字列を指定したのと同じことになります。たとえば、次の 2 つの指定はどちらも同じことを意味します。

```
fixed_field CHAR(2) NULLIF fixed_field=BLANKS  
fixed_field CHAR(2) NULLIF fixed_field=" "
```

マルチバイト・キャラクタ・セットには複数の空白が存在することもあります。このようなキャラクタ・セットには、空白文字列を指定するかわりに BLANKS を使用します。

文字列は特定の空白文字の組合せのみに一致しますが、BLANKS パラメータは様々な空白文字の組合せに一致します。マルチバイト・キャラクタ・セットの詳細は、5-17 ページの「[マルチバイト（アジア系言語）・キャラクタ・セット](#)」を参照してください。

## フィールドと文字列の比較

データ・フィールドが、それより短い比較文字列と比較された場合は、その文字列が埋め込まれます。文字データ型の文字列には、次のように空白が埋め込まれます。

```
NULLIF (1:4)=" "
```

この例では、位置 1:4 にあるデータが 4 つの空白と比較されます。位置 1:4 のデータが空白 4 つであれば、この句は真となります。

次の句のように、16 進文字列には 16 進数のゼロが埋め込まれます。

```
NULLIF (1:4)=X'FF'
```

この句で、データ位置 1:4 を 16 進数のバイト列 'FF000000' と比較します。

## WHEN、NULLIF および DEFAULTIF 句の使用

次の説明は、スカラー・フィールドに適用されます。非スカラー・フィールド（列オブジェクト、LOB およびコレクション）はより複雑なため、WHEN、NULLIF および DEFAULTIF 句は異なる方法で処理されます。

WHEN、NULLIF または DEFAULTIF 句は、フィールド名を指定するか、フィールド位置を指定するかによって、結果が異なります。

WHEN、NULLIF または DEFAULTIF 句でフィールド名を指定すると、これらの句は、SQL\*Loader によってフィールドの評価値と比較されます。評価値には、切り捨てられた空白文字が考慮されます。空白およびタブの切捨ての詳細は、6-41 ページの「[空白の切捨て](#)」を参照してください。

WHEN、NULLIF または DEFAULTIF 句で位置を指定すると、これらの句は、SQL\*Loader によってデータ・ファイル内の元の論理レコードと比較されます。この場合、論理レコードでは空白の切捨ては行われません。

フィールドに切り捨てられた空白がある場合、あるいは WHEN、NULLIF または DEFAULTIF 句に空白およびタブが含まれているか、BLANKS パラメータが使用されている場合は、異なる結果が得られます。名前で指定されているフィールドおよび位置で指定されているフィールドに対して同じ結果が必要な場合は、PRESERVE BLANKS オプションを使用します。PRESERVE BLANKS オプションによって、フィールド値の評価時に SQL\*Loader によって空白文字が切り捨てられないようにします。

WHEN、NULLIF または DEFAULTIF 句は、SQL\*Loader の処理手順によっても結果に影響します。SQL\*Loader は次の手順を順番に実行します。ただし、すべての手順を実行するわけではありません。フィールドが設定されると、設定作業の残りの手順は無視されます。たとえば、手順 5 でフィールドが設定された場合、SQL\*Loader は手順 6 には進みません。

1. SQL\*Loader で、入力レコードの各フィールドの値が評価され、空白およびタブの切り捨てに関する既存のガイドラインに従って、切り捨てる必要のある空白が切り捨てられます。
2. 各レコードに対して、SQL\*Loader で表の WHEN 句が評価されます。
3. レコードが表の WHEN 句を満たす場合、または WHEN が指定されていない場合は、SQL\*Loader で、NULLIF 句の各フィールドが確認されます。
4. NULLIF 句が存在する場合は、SQL\*Loader で評価されます。
5. NULLIF 句が満たされている場合は、SQL\*Loader はフィールドが NULL に設定されます。
6. NULLIF 句が満たされていない場合、または NULLIF 句がない場合は、SQL\*Loader のフィールド評価によってフィールド長が確認されます。フィールドがフィールド評価の結果、0 の長さを持っている場合（たとえば、NULL フィールドまたは結果として NULL フィールドとなる空白の切捨て）は、SQL\*Loader でフィールドが NULL に設定されます。この場合、フィールドに指定された DEFAULTIF 句は評価されません。



7. 指定された NULLIF 句が偽の場合、または NULLIF 句がない場合、およびフィールドがフィールド評価の結果、0 の長さを持たない場合は、SQL\*Loader で、DEFAULTIF 句に対するフィールドが確認されます。
8. DEFAULTIF 句が存在する場合は、SQL\*Loader で評価されます。
9. DEFAULTIF 句が満たされていて、データ・ファイルのフィールドが数値フィールドの場合、フィールドは 0 に設定されます。フィールドが数値フィールドではない場合、NULL に設定されます。次のフィールドは数値フィールドで、DEFAULTIF 句を満たす場合は、0 に設定されます。
  - BYTEINT
  - SMALLINT
  - INTEGER
  - FLOAT
  - DOUBLE
  - ZONED
  - (PACKED) DECIMAL
  - 数値型 EXTERNAL (INTEGER、FLOAT、DECIMAL および ZONED)
10. DEFAULTIF 句が満たされていない場合、または DEFAULTIF 句がない場合は、SQL\*Loader によって、手順 1 の評価値でフィールドが設定されます。

SQL\*Loader の操作順序が原因で、予期しない結果となる場合があります。たとえば、DEFAULTIF 句は、数値フィールドを 0 ではなく NULL に設定しているように見える場合があります。

例 6-2 ～例 6-5 に、異なる条件での結果を示します。これらの例では、空白またはスペースはピリオド (.) で示されています。col1 および col2 は、データベースの VARCHAR2 (5) 列です。

#### 例 6-2 DEFAULTIF 句の無評価

制御ファイルが次のように指定されています。

```
(col1 POSITION (1:5),
 col2 POSITION (6:8) CHAR INTEGER EXTERNAL DEFAULTIF col1 = 'aname')
```

データ・ファイルには、次のものが含まれます。

```
aname...
```

例 6-2 では、行の col1 が aname と評価され、col2 が長さが 0 (「...」ですが、後続の空白は位置フィールドでは切り捨てられます) である NULL と評価されます。

SQL\*Loader で col2 にロードされた最終値が確認される場合、WHEN 句および NULLIF 句は評価されません。フィールド長（フィールド評価の結果、0 と評価された長さ）が確認されます。したがって、SQL\*Loader では、col2 の最終値に NULL が設定されます。DEFAULTIF 句は評価されず、行は col1 の場合は aname、col2 の場合は NULL としてロードされます。

### 例 6-3 DEFAULTIF 句の評価

制御ファイルが次のように指定されています。

```
.
.
.
PRESERVE BLANKS
.
.
.
(col1 POSITION (1:5),
 col2 POSITION (6:8) INTEGER EXTERNAL DEFAULTIF col1 = 'aname')
```

データ・ファイルには、次のものが含まれます。

```
aname...
```

例 6-3 では、行の col1 は再び「aname」と評価されます。col2 は、PRESERVE BLANKS が指定された場合、後続の空白が切り捨てられないため、「...」と評価されます。

SQL\*Loader で col2 にロードされた最終値が確認される場合、WHEN 句および NULLIF 句は評価されません。0 でなく 3 と評価されたフィールド評価からフィールド長が確認されます。

次に、SQL\*Loader では DEFAULTIF 句が評価されます。col1 が「aname」で「aname」と同じであるため、真と評価されます。

col2 は数値フィールドであるため、SQL\*Loader では col2 の最終値に「0」が設定されます。行は、col1 の場合は「aname」、col2 の場合は「0」としてロードされます。

### 例 6-4 DEFAULTIF 句を使用した位置の指定

制御ファイルが次のように指定されています。

```
(col1 POSITION (1:5),
 col2 POSITION (6:8) INTEGER EXTERNAL DEFAULTIF (1:5) = BLANKS)
```

データ・ファイルには、次のものが含まれます。

```
.....123
```

例 6-4 では、行の col1 が、長さが 0 (..... ですが、後続の空白は切り捨てられます) の NULL と評価されます。col2 は、123 と評価されます。

SQL\*Loader で col2 にロードされる最終値が設定される場合、WHEN 句および NULLIF 句は評価されません。0 でなく 3 と評価されたフィールド評価からフィールド長が確認されます。

次に、SQL\*Loader では DEFAULTIF 句が評価されます。..... である (1:5) を、真と評価されている BLANKS と比較されます。したがって、col2 が数値フィールド (integer EXTERNAL は数値) のため、SQL\*Loader では col2 の最終値に「0」が設定されます。行は、col1 の場合は NULL、col2 の場合は 0 としてロードされます。

#### 例 6-5 DEFAULTIF 句を使用したフィールド名の指定

制御ファイルが次のように指定されています。

```
(col1 POSITION (1:5),
 col2 POSITION(6:8) INTEGER EXTERNAL DEFAULTIF col1 = BLANKS)
```

データ・ファイルには、次のものが含まれます。

```
.....123
```

例 6-5 では、行の col1 が、長さが 0 (..... ですが、後続の空白は切り捨てられます) の NULL と評価されます。col2 は、123 と評価されます。

SQL\*Loader で col2 の最終値が確認される場合、WHEN 句および NULLIF 句は評価されません。0 でなく 3 と評価されたフィールド評価からフィールド長が確認されます。

次に、SQL\*Loader では DEFAULTIF 句が評価されます。評価の一部として、SQL\*Loader では、col1 のフィールド評価が NULL であることが確認されます。NULL のため、DEFAULTIF 句は偽と評価されます。したがって、SQL\*Loader では col2 の最終値に、フィールド評価の元の値である 123 が設定されます。行は、col1 の場合は NULL、col2 の場合は 123 としてロードされます。

## 異なるプラットフォーム間でのデータのロード

データ・ファイルを作成するプラットフォームと、そのデータ・ファイルのロード先となるプラットフォームが異なる場合は、そのデータをターゲット・システムが読み込み可能な形式で作成する必要があります。たとえば、ソース・システムでは浮動小数点の内部表現に 16 バイトを使用し、ターゲット・システムでは浮動小数点を 12 バイトで表現しているとします。この場合、ソース・システムで生成されたデータを、ターゲット・システムに直接読み込ませることはできません。

この問題を解決する方法として、Oracle Net データベース・リンクを使用してデータをロードし、データ型の自動変換機能を利用する方法があります。前述のような問題が発生した場合は、できるだけこの方法を使用してください。この場合、SQL\*Loader はソース・システムで実行する必要があります。

プラットフォーム間のロードに関する問題は、通常、システム固有のデータ型に関連して発生します。フィールドに 0（ゼロ）を追加してフィールド長を長くするか、またはフィールドの一部分のみを読み込んでフィールド長を短くする（4 バイト整数を使用しているシステム上に 8 バイト整数を読み込む場合、またはその逆のパターンがこれに相当）ことによって、問題を回避できる場合もあります。ただし、データ型の実装に互換性がない場合は、この方法で問題の解決はできません。

Oracle Net データベース・リンクが使用できず、ターゲット・システム上で実行している SQL\*Loader を使用してデータ・ファイルにアクセスする必要がある場合は、移植可能な SQL\*Loader データ型（たとえば、CHAR、DATE、VARCHARC、数値型 EXTERNAL）のみを使用してください。このようにして作成したデータ・ファイルは、システム固有のデータ型を使用して作成したデータ・ファイルよりサイズが大きくなる場合があります。そのため、ロードに時間がかかりますが、異なるプラットフォームに直接転送することができます。

バイト順序スキームまたはシステム固有の整数の長さが、入力データが作成されるプラットフォームと SQL\*loader を実行するプラットフォームの間で異なることが事前にわかっている場合は、適切な方法で、データのバイト順序またはシステム固有の整数の長さを指定します。バイト順序を指定する方法には、BYTEORDER パラメータを使用する方法、またはファイルにバイト順序マーク（BOM）を設定する方法があります。この 2 つの方法の詳細は、6-36 ページの「**バイト順序**」を参照してください。これらの方法を実行すると、非互換性を排除し、プラットフォーム間での正常なデータ・ロードを実現できます。バイト順序が SQL\*Loader のデフォルトと異なる場合は、バイト順序を指定する必要があります。

## バイト順序

---

---

**注意：** この項の説明は、SQL\*Loader を実行するシステムとは異なるバイト順序スキームを持つシステム上で入力データを作成する場合のみに適用されます。それ以外の場合は、次の項に進んでください。

---

---

SQL\*Loader を使用して、SQL\*Loader が実行されているシステムとはバイト順序が異なるシステム上で作成されたデータ・ファイルからデータをロードできます。

デフォルトでは、SQL\*Loader で、すべてのデータ・ファイルに対するバイト順序として実行されているシステムのバイト順序が使用されます。たとえば、Sun SPARC Solaris システム上では、SQL\*Loader でビッグ・エンディアン・バイト順序が使用されます。Intel または Intel と互換性のある PC 上では、SQL\*Loader でリトル・エンディアン・バイト順序が使用されます。

バイト順序は、データが一度に偶数バイト（通常、2 バイト、4 バイトまたは 8 バイト）書き込まれる場合、および読み込まれる場合の結果に影響します。次に例をいくつか示します。

- 2 バイト整数値の 1 は、ビッグ・エンディアン・システムでは 0x0001、リトル・エンディアン・システムでは 0x0100 として書き込まれます。

- 4 バイト整数の 66051 は、ビッグ・エンディアン・システムでは 0x00010203、リトル・エンディアン・システムでは 0x03020100 として書き込まれます。

バイト順序は、UTF16 キャラクタ・セットの文字データが 2 バイトのエントリとして書き込まれる場合、および読み込まれる場合の結果にも影響します。たとえば、文字「a」（ASCII では 0x61）は、ビッグ・エンディアン・システムでは 0x0061、リトル・エンディアン・システムでは 0x6100 として書き込まれます。

Oracle でサポートされるすべてのキャラクタ・セットでは、UTF16 を除いて、一度に 1 バイト書き込まれます。そのため、UTF8 などのマルチバイト・キャラクタ・セットの場合も、文字の書込み、読み込みには、システムのバイト順序に関係なく、すべてのシステムで同じ方法が使用されます。したがって、UTF16 キャラクタ・セットのデータは、バイト順序依存のため移植不能です。Oracle でサポートされる他のすべてのキャラクタ・セットのデータは移植可能です。

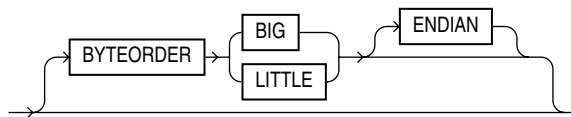
データ・ファイルのバイト順序が問題となるのは、バイト順序依存データを含むデータ・ファイルが、SQL\*Loader が実行されているシステムとは異なるバイト順序のシステムで作成される場合のみです。SQL\*Loader でデータのバイト順序を認識できる場合、必要に応じてバイトを入れ替えて、データが正常にターゲット・データベースにロードされることを確認します。バイト・スワップとは、ビッグ・エンディアン形式のデータをリトル・エンディアン形式に変換すること、またはその逆の変換を意味します。

SQL\*Loader にデータのバイト順序を指定するには、BYTEORDER パラメータを使用するか、またはそのファイルにバイト順序マーク（BOM）を設定します。この 2 つの方法のいずれも使用しない場合、SQL\*Loader ではデータを正常にデータ・ファイルにロードできません。

**参照：** SQL\*Loader でバイト・スワップを処理する方法の例の詳細は、10-48 ページの「[事例 11: Unicode キャラクタ・セットのデータのロード](#)」を参照してください。

## バイト順序の指定

入力データ・ファイルのデータのバイト順序を指定するには、SQL\*Loader 制御ファイルの次の構文を使用します。



BYTEORDER パラメータには、次の特長があります。

- BYTEORDER は、SQL\*Loader 制御ファイルの LENGTH パラメータの後に置かれます。
- 異なるデータ・ファイルに対して異なるバイト順序を指定することができます。ただし、INFILE パラメータの前の BYTEORDER 指定は、プライマリ・データ・ファイルのリスト全体に適用されます。

- プライマリ・データ・ファイルに対する BYTEORDER 指定は LOBFILE および SDF に対するデフォルトとしても使用されます。このデフォルトを上書きするには、LOBFILE または SDF 指定を使用して BYTEORDER を指定します。
- BYTEORDER パラメータは、制御ファイル内のデータには適用されません。
- BYTEORDER パラメータは次のものに適用されます。
  - － 2進 INTEGER データおよび SMALLINT データ
  - － 可変長フィールドのバイナリ長 (VARCHAR、VARGRAPHIC、VARRAW および LONG VARRAW データ型用)
  - － UTF16 キャラクタ・セットのデータ・ファイルの文字データ
  - － FLOAT および DOUBLE データ型 (データが書き込まれたシステムに、SQL\*Loader が実行されているシステム上のデータと互換性のある浮動小数点の表現がある場合)
- BYTEORDER パラメータは、次のものには適用されません。
  - － RAW データ型 (RAW、VARRAW または VARRAWC)
  - － 図形データ型 (GRAPHIC、VARGRAPHIC または GRAPHIC EXTERNAL)
  - － UTF16 以外のキャラクタ・セットのデータ・ファイルの文字データ
  - － ZONED データ型または (PACKED) DECIMAL データ型

## バイト順序マーク (BOM) の使用

Unicode エンコーディング (UTF-16 または UTF-8) が使用されているデータ・ファイルには、ファイルの最初の数バイトにバイト順序マーク (BOM) が含まれている場合があります。キャラクタ・セット UTF-16 が使用されているデータ・ファイルでは、ファイルの最初の 2 バイトの値 0xFEFF は、ファイルがビッグ・エンディアン of データを含んでいることを示す BOM です。0xFFFE という値は、ファイルにリトル・エンディアンのデータが含まれていることを示す BOM です。

第 1 プライマリ・データ・ファイルで UTF16 キャラクタ・セットが使用され、BOM も使用されている場合は、SQL\*Loader でそのマークを読み込み、解釈してすべてのプライマリ・データ・ファイルのバイト順序を決定します。SQL\*Loader で、BOM を読み込んで解釈し、スキップして、BOM の直後のバイトからデータを処理し始めます。BOM 設定は、第 1 プライマリ・データ・ファイルに対する BYTEORDER 指定より優先されます。第 1 プライマリ・データ・ファイル以外のデータ・ファイルの BOM は、バイト順序競合の確認のみに使用されます。データ・ファイルの処理中に SQL\*Loader で使用されるバイト順序の設定は、変更しません。

つまり、第 1 プライマリ・データ・ファイルに対するバイト順序インジケータの優先順位は次のようになります。

- 第1プライマリ・データ・ファイルのBOM（データ・ファイルで、バイト順序依存のUnicodeキャラクタ・セット（UTF16）が使用され、BOMが存在する場合）
- BYTEORDERパラメータの値（INFILEパラメータの前に指定された場合）
- SQL\*Loaderが実行されているシステムのバイト順序

UTF8キャラクタ・セットが使用されているデータ・ファイルでは、最初の3バイトの0xEFBBBFというBOMによって、ファイルにUTF8データが含まれていることが示されます。UTF8のデータはバイト順序依存ではないため、BOMではデータのバイト順序を指定しません。UTF8のBOMが検出された場合は、SQL\*LoaderでBOMをスキップしますが、データ・ファイルの処理のためのバイト順序の設定変更は実行しません。

SQL\*Loaderによって、まず、定義済の優先順位を使用して第1プライマリ・データ・ファイルのバイト順序設定を確立します。このバイト順位設定は、すべてのプライマリ・データ・ファイルに使用されます。別のプライマリ・データ・ファイルでキャラクタ・セットUTF16が使用され、BOMも含まれている場合、そのBOMの値は、第1プライマリ・データ・ファイルで確立されたバイト順序設定と比較されます。BOMの値が第1プライマリ・データ・ファイルのバイト順序設定と一致する場合は、SQL\*LoaderでそのBOMをスキップし、そのバイト順序設定を使用して、BOMの直後のバイトからデータを処理し始めます。BOMの値が第1プライマリ・データ・ファイルで確立されたバイト順序設定と一致しない場合は、SQL\*Loaderで、エラー・メッセージを発行し、処理を停止します。

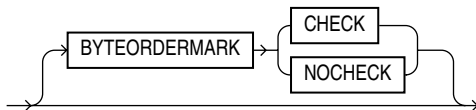
LOBFILEまたはSDFが制御ファイルで指定される場合、ファイルを処理する準備が整うと、SQL\*Loaderで各LOBFILEおよびSDFのバイト順序設定を確立します。LOBFILEおよびSDFのデフォルトのバイト順序設定は、第1プライマリ・データ・ファイルで確立されたバイト順序設定です。これは、BYTEORDERパラメータがLOBFILEまたはSDFで指定される場合は、上書きされます。いずれの場合も、LOBFILEまたはSDFでUTF16キャラクタ・セットが使用され、BOMが含まれている場合、BOMの値はファイルのバイト順序と比較されます。BOMの値がファイルのバイト順序設定と一致する場合は、SQL\*LoaderでそのBOMをスキップし、そのバイト順序設定を使用して、BOMの直後のバイトからデータを処理し始めます。BOMの値が一致しない場合、SQL\*Loaderからエラーが発行され、処理が停止します。

つまり、LOBFILEおよびSDFに対するバイト順序インジケータの優先順位は次のようになります。

- LOBFILEまたはSDFで指定されたBYTEORDERパラメータ値
- 第1プライマリ・データ・ファイルで確立されたバイト順序

## BOMの確認の抑止

Unicodeキャラクタ・セットのデータ・ファイルには、ファイルの最初のバイトにBOMと一致するバイナリ・データが含まれています。たとえば、integer(2)型の値0xFEFF = 65279小数点は、UTF16のビッグ・エンディアンBOMと一致します。この場合、SQL\*Loaderを使用してデータ・ファイルの最初のバイトをデータとして読込みできます。また、値NOCHECKでBYTEORDERMARKパラメータを指定して、SQL\*LoaderによってBOMの確認もできます。BYTEORDERMARKパラメータの構文は次のとおりです。



BYTEORDERMARK NOCHECK を指定すると、SQL\*Loader では BOM が確認されず、データ・ファイルのすべてのデータがデータとして読む込まれます。

BYTEORDERMARK CHECK を指定すると、SQL\*Loader で BOM が確認されます。これは、Unicode キャラクタ・セットのデータ・ファイルのデフォルトの動作です。ただし、この仕様は説明のために制御ファイルで 사용되는場合があります。Unicode 以外のキャラクタ・セットが使用されているデータ・ファイルに BYTEORDERMARK CHECK を指定すると、エラーが発生します。

BYTEORDERMARK パラメータには、次の特長があります。

- SQL\*Loader 制御ファイル内のオプションの BYTEORDER パラメータの後に置かれます。
- LOBFILE データ・ファイルおよび SDF に関してのみではなくプライマリ・データ・ファイルに関する構文の指定にも適用されます。
- 異なるデータ・ファイルに異なる BYTEORDERMARK 値を指定することができます。ただし、INFILE パラメータの前の BYTEORDERMARK 指定は、プライマリ・データ・ファイルのリスト全体に適用されます。
- LOBFILE または SDF で Unicode 以外のキャラクタ・セットが使用された場合に値 CHECK が無視される以外は、プライマリ・データ・ファイルの BYTEORDERMARK 指定は LOBFILE および SDF のデフォルトとしても使用されます。LOBFILE および SDF のデフォルト設定は、LOBFILE または SDF 仕様部で BYTEORDERMARK を指定して、上書きできます。

## すべてが空白のフィールドのロード

数値フィールドまたは完全に空白のフィールドが原因で、レコードは拒否されます。これらのフィールドのいずれかを NULL としてロードするには、BLANKS パラメータを持つ NULLIF 句を使用します。

空白のフィールドが CHAR 型で、囲みデリミタによって囲まれている場合は、囲みデリミタで囲まれている空白部分がロードされます。囲みデリミタで囲まれていない場合は、そのフィールドは NULL としてロードされます。

DATE フィールドのデータがすべて空白の場合、NULL フィールドとしてロードされます。



**参照：**

- NULLIF 句を使用して、すべてが空白のフィールドを NULL としてロードする方法の例の詳細は、10-25 ページの「[事例 6: ダイレクト・パス・ロード方式を使用したデータのロード](#)」を参照してください。
- 6-41 ページの「[空白の切捨て](#)」も参照してください。
- 6-47 ページの「[空白の保存](#)」も参照してください。

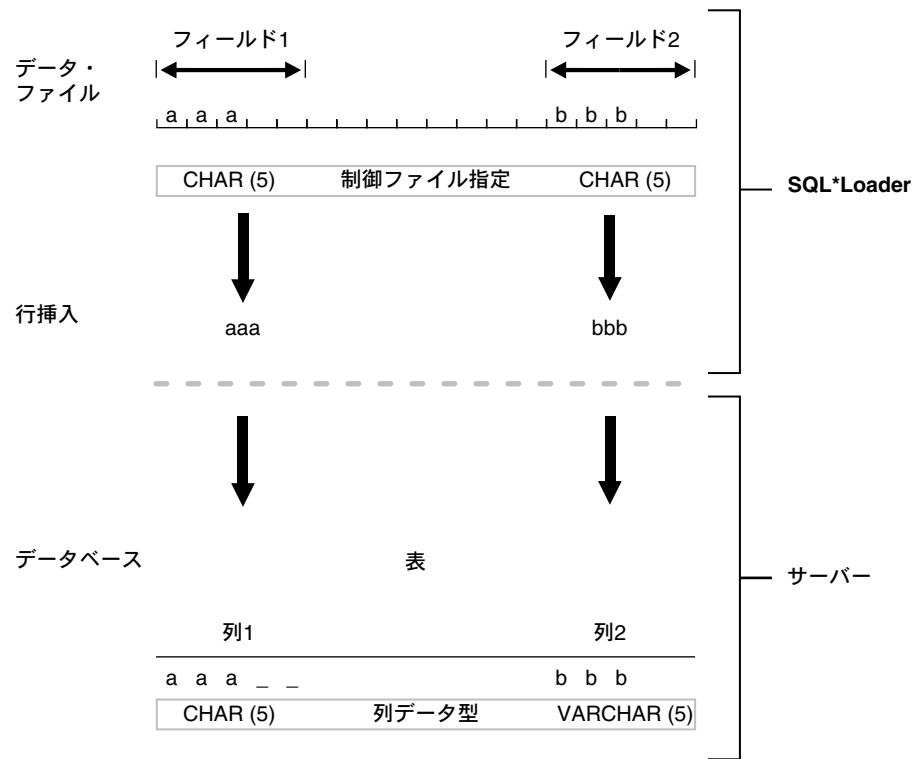
## 空白の切捨て

空白、タブおよびその他の印字されない文字（改行、LF など）が空白を構成します。先頭の空白は、フィールドの開始位置にあります。後続の空白は、フィールドの終了位置にあります。フィールドの指定方法にもよりますが、空白は、フィールドのデータベースへの挿入時に、データの一部として含めることも切り捨てることもできます。これについて、[図 6-1](#)に示します。この図では、2 つの CHAR フィールドがデータ・レコードに定義されています。

フィールド指定は制御ファイルに含まれています。制御ファイルの CHAR 指定は、データベースの CHAR 指定と同じではありません。制御ファイル内で CHAR として定義されたデータ・フィールドは、SQL\*Loader に行挿入の作成方法を指定するのみです。Oracle データベース・サーバーで必要な変換を行い、データベースの CHAR、VARCHAR2、NCHAR、NVARCHAR、NUMBER または DATE 列にデータを挿入できるようになります。

デフォルトでは、SQL\*Loader を使用して CHAR データから後続の空白を削除した後、このデータをデータベースに渡します。その後、[図 6-1](#)に示すように、フィールド 1 とフィールド 2 は 3 バイトのフィールドとしてデータベースに渡されます。ただし、データを表に挿入する場合は処理が異なります。

図 6-1 フィールド変換の例



列 1 は、データベース内で長さ 5 の固定長 CHAR 列として定義されます。そのため、データ (aaa) は 5 バイトの幅を保持したまま、その列で左揃えにされます。余った右側の部分は空白で埋められます。一方、列 2 は、最大長 5 バイトの可変長フィールドとして定義されています。その列 (bbb) のデータも左揃えにされますが、長さは 3 バイトのままです。

表 6-5 に、PRESERVE BLANKS が指定されていない場合に空白が入力データ・フィールドから切り捨てられるケースおよびその処理方法を示します。空白の切捨てを回避する方法については、6-47 ページの「空白の保存」を参照してください。

表 6-5 空白の切捨てに関する動作のサマリー

指定	データ	結果	先頭の空白 <sup>1</sup>	後続の空白 <sup>1</sup>
サイズ指定あり	__aa__	__aa	あり	なし
終了デリミタ	__aa_,	__aa__	あり	あり <sup>2</sup>
囲みデリミタ	"__aa__"	__aa__	あり	あり
終了と囲み	"__aa__",	__aa__	あり	あり
オプションの囲み（あり）	"__aa__",	__aa__	あり	あり
オプションの囲み（なし）	__aa_,	aa__	なし	あり
前のフィールドが空白で区切られている場合	__aa__	aa <sup>3</sup>	なし	<sup>3</sup>

<sup>1</sup> 空白のみのフィールドが切り捨てられた場合、値は NULL になります。  
<sup>2</sup> 空白で終了するフィールドを除きます。  
<sup>3</sup> 後続の空白があるかどうかは、表中の他の項目によって示されるとおり、現行のフィールドの指定によって異なります。

この項の残りの部分では、空白の切捨てに関する次の項目について説明します。

- [空白の切捨てが可能なデータ型](#)
- [空白の切捨てが可能なデータ型に対するフィールド長指定](#)
- [フィールドの相対的な位置指定](#)
- [先頭の空白](#)
- [後続の空白](#)
- [囲まれたフィールド](#)

空白の切捨てが可能なデータ型

次の説明は、データ型が文字データ型であるフィールドにのみ適用できます。

- CHAR データ型
- 日時データ型および期間のデータ型
- 数値型 EXTERNAL データ型
  - INTEGER EXTERNAL
  - FLOAT EXTERNAL
  - (PACKED) DECIMAL EXTERNAL

- ZONED (10 進) EXTERNAL

---

**注意：** VARCHAR 型および VARCHARC 型フィールドも文字データを含みますが、フィールド中の空白は切り捨てられません。これらのフィールドは、データ・ファイルのフィールド中の空白文字をすべて含みます。

---

## 空白の切捨てが可能なデータ型に対するフィールド長指定

フィールド長の指定には 2 通りの方法があります。制御ファイルで位置指定またはデータ型および長さについてフィールド長の定数を定義した場合、そのフィールドは、事前にサイズが決まっていることになります。一方、フィールド長を事前に指定しないでレコード中のインジケータでフィールド長を決める場合は、そのフィールドを囲みデリミタまたは終了デリミタのいずれかを使用して区切ります。

開始値および終了値を持つ位置指定は、囲みデリミタまたは終了デリミタが定義されるフィールドに対して定義されます。囲みデリミタまたは終了デリミタは無視されます。

### 事前にサイズが決まっているフィールド

フィールドのサイズを事前に決定するには、フィールドの開始位置と終了位置を指定するか、フィールド長を指定します。それぞれの指定例を示します。

```
loc POSITION(19:31)
loc CHAR(14)
```

2 番目の例では、フィールド位置は指定されていませんが、フィールド長は事前に指定されています。

### デリミタ付きフィールド

デリミタとは、フィールドの境界を指定する文字のことです。

囲みデリミタは、次の例 ("\_" が空白またはタブを表す) 中の引用符のように、フィールドを囲みます。

```
"__aa__"
```

一方、終了デリミタは、フィールドの終わりを示します。次の例中のカンマがこれに相当します。

```
__aa__,
```

デリミタに使用する文字は、制御句 TERMINATED BY および ENCLOSED BY を使用して指定します。次に指定例を示します。

```
loc TERMINATED BY "." OPTIONALLY ENCLOSED BY '||'
```

この項では、次の条件で **SQL\*Loader** を使用してフィールドの開始位置を決定する方法について説明します。

- ### フィールドの開始位置が指定されていない場合

フィールドの開始位置が指定されていない場合は、前のフィールドの終了位置の直後の位置が、そのフィールドの開始位置となります。図 6-2 に、前のフィールド（フィールド 1）のサイズが事前に決定されている場合の例を示します。

CHAR(9)のフィールド1 「,」で終了するフィールド2

a, a, a, a, , b, b, b, b, \0

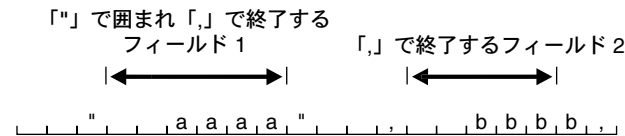
前のフィールド（フィールド 1）の終端がデリミタで指定されている場合、次のフィールドはそのデリミタの直後から開始します。この例を図 6-3 に示します。

「,」で終了するフィールド1 「,」で終了するフィールド2

a a a a , b b b b

フィールドが囲みデリミタと終了デリミタの両方で指定された場合、その次のフィールドは終了デリミタの直後の位置から開始します。この例を図 6-4 に示します。囲みデリミタから終了デリミタまでの間に空白以外の文字がある場合は、エラーが出力されます。

図 6-4 囲みデリミタの後の相対的な位置指定



先頭の空白

図 6-4 の例では、2つのフィールドはともに先頭の空白が付いた状態で格納されます。ただし、次のような場合、先頭の空白はフィールドのデータに含まれません。

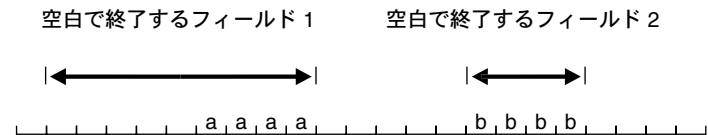
- 前のフィールドが空白で区切られて（終了して）いて、現行のフィールドの開始位置が指定されていない場合。
- フィールドに対してオプションの囲みデリミタが指定されているにもかかわらず、その囲みデリミタが使用されていない場合。

これらの事例については、次の項で例示します。

前のフィールドが空白で区切られている場合

前のフィールドが `TERMINATED BY WHITESPACE` で区切られていると、そのフィールドの後に続く空白はすべてデリミタとみなされます。この場合、次のフィールドは、次に空白以外の文字が現れた位置から開始します。この例を図 6-5 に示します。

図 6-5 空白で区切られたフィールド



このようなケースは、前述の例のように前のフィールドが `TERMINATED BY WHITESPACE` 句で明示的に指定された場合に発生します。グローバルに `FIELDS TERMINATED BY WHITESPACE` 句が指定された場合も、このケースに相当します。

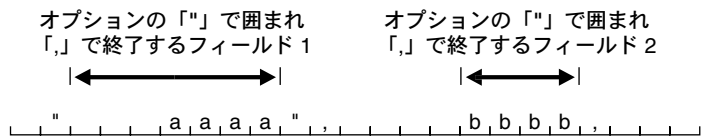
オプションの囲みデリミタ

オプションの囲みデリミタが指定されているにもかかわらず、それが使用されていない場合も、先頭の空白は切り捨てられます。

オプションの囲みデリミタが指定されると、`SQL*Loader` によって前方スキャンが実行され、最初の囲みデリミタが検索されます。囲みデリミタがない場合は、空白がスキップされ、

フィールドから削除されます。最初に見つかった空白以外の文字がフィールドの開始と判断されます。この例を図 6-6 のフィールド 2 に示します（フィールド 1 では、SQL\*Loader によってフィールドの囲みデリミタが検出されたため空白が含まれます）。

図 6-6 オプションの囲みデリミタ付きフィールド



前のフィールドが TERMINATED BY WHITESPACE で区切られた場合と異なり、前述のように指定された場合は、現行のフィールドの開始位置が指定されていても先頭の空白は切り捨てられます。

**注意：** 囲みデリミタが存在する場合は、最初の囲みデリミタの後の先頭の空白はそのままデータとして保持されますが、この囲みデリミタの前にある空白は切り捨てられます。図 6-6 のフィールド 1 の最初の引用符がこのケースに相当します。

## 後続の空白

後続の空白は、フィールドが文字データ型で事前にフィールド・サイズが決まっている場合、常に切り捨てられます。後続の空白が常に切り捨てられるのは、これらのフィールドのみです。

## 囲まれたフィールド

フィールドが囲みデリミタで囲まれている場合、または、図 6-6 の最初のフィールドのように終了デリミタと囲みデリミタの両方で区切られている場合、囲みデリミタの外側にある空白は、フィールドの一部とはみなされません。囲みデリミタの内側に空白があれば、それが先頭の空白または後続の空白のいずれであっても、フィールドの一部とみなされます。

## 空白の保存

CHAR、DATE および数値型 EXTERNAL 型のすべてのフィールド中で空白の切捨てを回避するには、制御ファイルに PRESERVE BLANKS を指定します。空白の切捨てについては、6-41 ページの「空白の切捨て」を参照してください。

## PRESERVE BLANKS オプション

PRESERVE BLANKS オプションには、次の特長があります。

- オプションの囲みデリミタがない場合、先頭の空白はそのまま残ります。
- 事前にフィールド・サイズが指定された場合にも、後続の空白を残すことができます。

たとえば、次のフィールドについて考えてみます。ここでは、アンダースコアは空白を表します。

```
__aa__
```

このフィールドが次の制御句でロードされる場合、PRESERVE BLANKS を指定すると、先頭の空白および後続の空白の両方とも保存されます。PRESERVE BLANKS を指定しない場合、先頭の空白は切り捨てられます。

```
TERMINATED BY ',' OPTIONALLY ENCLOSED BY '''
```

---

**注意：** BLANKS はオプションではなく必須です。2 語とも指定する必要があります。

---

### 空白で区切られている場合

前のフィールドの終端が空白で区切られている場合、PRESERVE BLANKS を指定しても、次の（現在の）フィールドの先頭の空白は切り捨てられます。ただし、その空白を含む次の（現在の）フィールドが POSITION 句で指定されている場合は、先頭の空白は保存されます。このような POSITION 句の指定がない場合は、SQL\*Loader によって前のフィールドの終わりにあるすべての空白を読み込まずにスキャンが実行され、次に空白以外の文字またはタブ以外の文字が現れた位置が次のフィールドの開始位置と認識されます。

## フィールドへの SQL 演算子の適用

SQL 文字列を使用することによって、様々な SQL 演算子をフィールド・データに適用できます。SQL 文字列には、任意に組み合わせた SQL 式を組み込むことができます。ただし、この SQL 式は、Oracle データベースによって INSERT 文中の VALUES 句に対して有効であると認識されたものにかぎります。通常、ターゲット列のデータ型と互換性のある単一の値を返す SQL 関数を使用できます。SQL 文字列は、列オブジェクトおよびコレクションなどのユーザー定義の複合型に加えて、単純なスカラー列型にも適用できます。式の詳細は、『Oracle9i SQL リファレンス』を参照してください。

列名と SQL 文字列中の列名は、引用符も含め、正確に一致している必要があります。次に制御ファイルでの指定例を示します。

```
LOAD DATA
  INFILE *
  APPEND INTO TABLE XXX
```



```
( "Last"    position(1:7)    char    "UPPER(:\"Last\")"
  FIRST    position(8:15)   char    "UPPER(:FIRST)"
)
BEGINDATA
Phil Locke
Jason Durbin
```

次の要件および制限は SQL 文字列を使用している場合に適用されます。

- SQL 文字列を指定する位置は、その列に関するその他の指定がすべて記述された後になります。
- SQL 文字列は、二重引用符で囲んで記述します。
- SQL 文字列内の列名を引用符で囲むには、エスケープ文字を使用する必要があります。  
前述の例では、Last を二重引用符で囲んで大文字および小文字の組合せを保持しています。また、二重引用符を使用すると、バックスラッシュ（エスケープ）文字も使用する必要があります。
- SQL 文字列に列オブジェクトの属性を参照する列名が含まれている場合は、全フィールド名を使用し、引用符で囲む必要があります。
- SQL 文字列の評価は、NULLIF 句または DEFAULTIF 句の後、日時マスクよりも前に行われます。
- Oracle データベース・サーバーで SQL 文字列を認識できない場合は、エラーが発生してロードは終了します。文字列が認識されても、データベース・エラーが発生すると、エラーの発生した行は拒否されます。
- フィールド指定で EXPRESSION パラメータを使用する場合は、SQL 文字列が必要です。
- SQL 文字列にバインド変数が含まれる場合、バインド変数は 4000 バイトを超えることができません。超えた場合は、レコードが拒否されます。
- SQL 文字列では、OID、SID、REF または BFILE を使用してロードしたフィールドは参照できません。また、FILLER フィールドも参照できません。
- ダイレクト・パス・モードでは、SQL 文字列での VARRAY、ネストした表または LOB 列の参照はできません。これには、列オブジェクトの属性である VARRAY、ネストした表または LOB 列も含まれます。
- SQL 文字列は、RECNUM、SEQUENCE、CONSTANT または SYSDATE フィールドでは使用できません。
- SQL 文字列は、LOB、BFILE、XML 列またはコレクションの要素であるファイルでは使用できません。
- ダイレクト・パス・モードでは、SQL 文字列の式の評価後に返される最後の結果はスカラー・データ型である必要があります。つまり、ダイレクト・パス・ロードの実行時、式によってはオブジェクトまたはコレクション・データを返さない場合があります。

## フィールドの参照

レコード中のフィールドを参照する場合は、フィールド名の前にコロン (:) を付けます。このように指定すると、現在のレコードのフィールド値が代入されます。SQL 文字列で、前にコロン (:) が付いたフィールド名もバインド変数として参照されます。次の例では、制御ファイルの現行のフィールドおよび他のフィールドの参照方法を示します。

```
LOAD DATA
INFILE *
APPEND INTO TABLE YYY
(
  field1 POSITION(1:6) CHAR "LOWER(:field1)"
  field2 CHAR TERMINATED BY ','
        NULLIF ((1) = 'a') DEFAULTTIF ((1)= 'b')
        "RTRIM(:field2)"
  field3 CHAR(7) "TRANSLATE(:field3, ':field1', ':1')",
  field4 COLUMN OBJECT
  (
    attr1 CHAR(3) "UPPER(:\"FIELD4.ATTR3\")",
    attr2 CHAR(2),
    attr3 CHAR(3) ":\FIELD4.ATTR1\" + 1"
  ),
  field5 EXPRESSION "MYFUNC(:FIELD4, SYSDATE)"
)
BEGINDATA
ABCDEF1234511 ,:field1500YYabc
abcDEF67890 ,:field2600ZZghl
```

前述の例では、次のことに注意してください。

- :field1 のみが一重引用符で囲まれていないため、列名として解釈されます。':field1' および ':1' は、変更されずに TRANSLATE 関数に渡されるテキスト・リテラルです。引用符で囲まれた文字列中で引用符を使用する方法の詳細は、5-5 ページの「[ファイル名およびオブジェクト名の指定](#)」を参照してください。
- 各入力データ読込みでは、バインド変数で参照されたフィールドの値はバインド変数に置き換えられます。たとえば、最初のレコードの値 ABCDEF は、最初のフィールド :field1 にマップされます。この値は、引数として LOWER 関数に渡されます。
- バインド変数が列オブジェクトへの参照の場合、全フィールド名は大文字で、引用符で囲まれる必要があります。たとえば、:\"FIELD4.ATTR1\"、:\"FIELD4.ATTR3\" などです。
- SQL 文字列のバインド変数によって現行のフィールドを参照する必要はありません。前述の例では、フィールド FIELD4.ATTR1 に対する SQL 文字列のバインド変数によって、フィールド FIELD4.ATTR3 を参照しています。フィールド FIELD4.ATTR1 は、入力レコードの値 500 および 600 にマップされます。ただし、対応する列に格納された最終値は ABC および GHL です。

- field5 は、入力レコードのどのフィールドにもマップされません。ターゲット列に格納されている値は、2つの引数をとる PL/SQL 関数 MYFUNC の実行結果です。  
EXPRESSION パラメータの使用には、入力データがフィールドにマップされないため、SQL 文字列を使用して列の最終値を計算する必要があります。

## フィールド指定での SQL 演算子の共通使用

次のタスクでは、共通して SQL 演算子を使用されています。

- 暗黙の小数点が付いている EXTERNAL データをロードするには、次のように指定します。

```
field1 POSITION(1:9) DECIMAL EXTERNAL(8) ":field1/1000"
```

- また、長すぎるフィールドを切り捨てるには、次のように指定します。

```
field1 CHAR TERMINATED BY ", " "SUBSTR(:field1, 1, 10)"
```

## SQL 演算子の組合せ

複数の演算子を次の例のように組み合わせることができます。

```
field1 POSITION(*+3) INTEGER EXTERNAL
      "TRUNC(RPAD(:field1,6,'0'), -2)"
field1 POSITION(1:8) INTEGER EXTERNAL
      "TRANSLATE(RTRIM(:field1),'N/A', '0')"
```

```
field1 CHAR(10)
      "NVL( LTRIM(RTRIM(:field1)), 'unknown' )"
```

## 日付マスク付きの SQL 文字列の使用

日付マスクと併用する場合、日付マスクは SQL 文字列の後で評価されます。次のように指定されるフィールドがあるとします。

```
field1 DATE "dd-mon-yy" "RTRIM(:field1)"
```

SQL\*Loader の内部で次の文が生成され、挿入されます。

```
TO_DATE(RTRIM(<field1_value>), 'dd-mon-yyyy')
```

DATE フィールド・データ型を使用する場合、日付マスクなしの SQL 文字列は使用できないことに注意してください。これは、SQL\*Loader で、DATE パラメータの後の最初の引用符付き文字列が日付マスクであるとみなされるためです。たとえば、次のフィールド指定では、エラー（ORA-01821: 日付書式コードが無効です）が発生します。

```
field1 DATE "RTRIM(TO_DATE(:field1, 'dd-mon-yyyy'))"
```

この場合、単純な解決策としては、CHAR データ型を使用します。

## 書式化されたフィールドの解析

TO\_CHAR 演算子を使用して、書式化された日付および数値を格納できます。次に例を示します。

```
field1 ... "TO_CHAR(:field1, '$09999.99')"
```

この例では、数値型の入力データを、書式化された形式で格納することができます。この場合、field1 はデータベース中ではキャラクタ列です。この指定にある書式化文字（ドル記号やピリオドなど）は、データとともにそのままフィールドに格納されます。

ただし、このような値を数量や日付として格納すると、より柔軟な処理を行うことができます。この場合、データベース内の値に算術関数を指定しても、書式化された値を選択してレポートを作成することができます。

10-29 ページの「[事例 7: 書式化されたレポートからのデータの抽出](#)」にある、SQL 文字列を使用して書式化されたレポートからデータをロードする例を参照してください。

## SQL\*Loader を使用した入力データの生成

この項では、データ・ファイルからデータを読み込むのではなく、SQL\*Loader でデータを生成してデータベース・レコードに格納するパラメータについて説明します。ここで説明するパラメータは次のとおりです。

- [CONSTANT パラメータ](#)
- [EXPRESSION パラメータ](#)
- [RECNUM パラメータ](#)
- [SYSDATE パラメータ](#)
- [SEQUENCE パラメータ](#)

## ファイルを使用しないデータのロード

フィールドの指定として順序、レコード番号、システム日付、定数および SQL 文字列式のみを指定して SQL\*Loader でデータを生成できます。

SQL\*Loader を使用して、LOAD 文で指定された数のレコードを挿入します。この場合、SKIP パラメータは指定できません。

SQL\*Loader は、このような場合用に最適化されています。生成された指定のみが使用されていることを検出すると、SQL\*Loader では、常に、指定されたすべてのデータ・ファイルが無視されます。I/O の読み込みは実行されません。

バインド配列用のメモリーも不要です。制御ファイル中に WHEN 句が指定されている場合は、SQL\*Loader でデータの評価が必要であると判断され、入力レコードが読み込まれます。

## 列への定数値の設定

これは、生成するデータとしては最も単純な形式です。ロード中であっても、何回ロードしても、このデータは変わりません。

### CONSTANT パラメータ

列に定数値を設定するには、CONSTANT を指定して、その後に値を指定します。

```
CONSTANT value
```

CONSTANT データは、SQL\*Loader では、文字入力として認識されます。このデータは、必要に応じてデータベース列のデータ型に変換されます。

値を引用符で囲むこともできます。特に、指定する値に空白や予約語が含まれている場合は、必ず引用符で囲んでください。また、ターゲット列に対して必ず有効な値を指定してください。指定した値が無効な場合は、すべてのレコードが拒否されてしまいます。

2 の 32 乗 -1 (4,294,967,295) より大きい数値は引用符で囲んでください。

---

---

**注意：** CONSTANT パラメータを使用して列に NULL を設定することはできません。列を NULL に設定する場合は、その列については何も指定しないでください。これによって、Oracle でレコードをロードするときに、その列に自動的に NULL が設定されます。CONSTANT および値の組合せを指定すると、列指定は完結します。

---

---

## 列への式の値の設定

列名の後に EXPRESSION パラメータを使用して、SQL 演算子または特別に書き込まれた PL/SQL 関数によって返された値に、その列を設定します。EXPRESSION パラメータの後に続く SQL 文字列に、演算子または関数が指定されます。演算子または関数に必要なパラメータが適切に指定され、その演算子または関数によって返された結果が、ロード中の列のデータ型と互換性がある場合、任意の式を使用できます。

### EXPRESSION パラメータ

列名、EXPRESSION パラメータおよび SQL 文字列の組合せは、完全なフィールド指定です。

```
column_name EXPRESSION "SQL string"
```

## 列へのデータ・ファイルのレコード番号の設定

RECNUM パラメータを列名の後に指定すると、レコードのロード元である論理レコードの番号が、その列に設定されます。レコードの番号は、最初のデータ・ファイルの先頭をレコード 1 として、順番にカウントされます。RECNUM は、各論理レコードが構成されるたびに増えていきます。廃棄、スキップ、拒否またはロードされたレコードもカウントされます。SKIP=10 と指定した場合、最初にロードされるレコードの RECNUM の値は 11 になります。

### RECNUM パラメータ

列名および RECNUM の組合せを指定すると、列指定は完結します。

```
column_name RECNUM
```

## 列への現在の日付の設定

SYSDATE を使用して列を指定すると、SQL 言語の SYSDATE パラメータで定義されているものと同じ、現在のシステム日付が列に設定されます。詳細は、『Oracle9i SQL リファレンス』の「DATE データ型」を参照してください。

### SYSDATE パラメータ

列名と SYSDATE パラメータの組合せを指定すると、列指定は完結します。

```
column_name SYSDATE
```

データベースの列のデータ型は、CHAR または DATE 型にしてください。列が CHAR 型の場合、日付は 'dd-mon-yy' の書式でロードされます。ロード後は、この書式でのみ日付のロードができます。システム日付を DATE 列にロードすると、そのシステム日付は、時間と日付を含む様々な書式でロードできます。

新しいシステム日付 / 時間は、従来型パス・ロードで挿入されたレコードの各配列や、ダイレクト・パス・ロード中にロードされた各レコード・ブロックで使用されます。

## 列への一意の順序番号の設定

SEQUENCE パラメータは、特定の列に対して一意の値を設定します。SEQUENCE の値は、ロードされたレコードまたは拒否されたレコードが発生するたびに増加します。廃棄またはスキップされたレコードに対しては、値は増加しません。

SEQUENCE パラメータ

列名と SEQUENCE パラメータの組合せを指定すると、列指定は完結します。

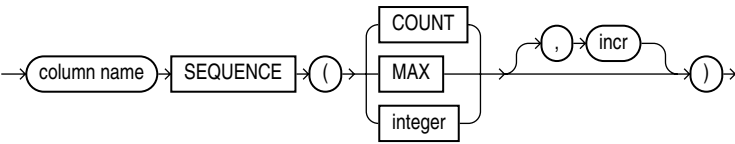


表 6-6 では、列指定に使用されるパラメータについて説明します。

表 6-6 列指定に使用されるパラメータ

パラメータ	説明
<i>column_name</i>	データベース内の、順序を割り当てる列の名前。
SEQUENCE	列の値を指定するには、この SEQUENCE を使用します。
COUNT	順序番号は表中にすでにあるレコード数で始まり、以降は増分値を加えた値が設定されます。
MAX	順序番号は列の現在の最大値で始まり、以降は増分値を加えた値が設定されます。
INTEGER	先頭となる特定の順序番号を指定します。
<i>incr</i>	レコードがロードまたは拒否された後の順序番号の増分値。これはオプションで、デフォルトは 1 です。

レコードが拒否された場合（書式エラーがあるか、または Oracle エラーが発生した場合）でも、その欠落を埋めるために生成された順序番号が変更されることはありません。たとえば、ある列に関して、4 つの行に順序番号 10、12、14 および 16 が割り当てられ、番号 12 の行が拒否されたとします。この場合、挿入された 3 つの行の番号は 10、14 および 16 であって、10、12 および 14 ではありません。このような処理が行われることによって、データ・エラーが発生しても、挿入時の順番を保持できます。拒否されたデータを修正して再度挿入する場合は、列の順序番号に一致するようにデータを手動で設定できます。

SEQUENCE パラメータの使用例の詳細は、10-12 ページの「[事例 3: 自由区分形式ファイルのロード](#)」を参照してください。

## 複数の表に対する順序番号の生成

一意の順序番号は、各表への挿入時に生成されるのではなく、各論理入力レコードに対して生成されます。したがって、データを複数の表に挿入する場合、同じ順序番号を使用することができます。この仕様は、多くの場合に有効です。

ただし、INTO TABLE 句ごとに別の順序番号を生成する場合があります。たとえば、各入力レコード中に 3 つの論理レコードが定義された形式のデータについて考えます。この場合、INTO TABLE 句を 3 つ使用して、レコードの 3 つの異なる部分を同じ表に挿入するように指定できます。SEQUENCE (MAX) を使用すると、各表の最大値が採用されるため、順序番号に一貫性がなくなります。

このレコードの順序番号を生成する場合は、挿入する 3 つの論理レコードそれぞれに対して、重複しない番号を生成する必要があります。1 レコード当たりの表挿入の回数を順序番号の増分値として使用し、各挿入の順序番号をその続き番号で始めるようにします。

### 例：挿入ごとの順序番号の生成

次に示す部門名を dept 表にロードするとします。各入力レコードには部門名が 3 つ入っています。この部門番号を自動生成する方法について考えます。

```
Accounting      Personnel      Manufacturing
Shipping        Purchasing      Maintenance
...
```

部門番号を重複しないように生成するには、次のような制御ファイル・エントリを作成します。

```
INTO TABLE dept
(deptno SEQUENCE(1, 3),
  dname  POSITION(1:14) CHAR)
INTO TABLE dept
(deptno SEQUENCE(2, 3),
  dname  POSITION(16:29) CHAR)
INTO TABLE dept
(deptno SEQUENCE(3, 3),
  dname  POSITION(31:44) CHAR)
```

最初の INTO TABLE 句で生成される部門番号は 1 で、2 番目では 2 が、3 番目では 3 が生成されます。すべての INTO TABLE 句で増分値は 3 が使用されます（増分値は各レコードに含まれる部門名の数と一致します）。この制御ファイルでロードを実行すると、Accounting 部門は部門番号 1、Personnel 部門は部門番号 2、Manufacturing 部門は部門番号 3 でロードされます。

また、次のレコードになると、順序番号は増分値分だけ増加するので、Shipping 部門は部門番号 4、Purchasing 部門は部門番号 5 でロードされ、以降も同様にロードされます。



---

# オブジェクト、LOB およびコレクションのロード

この章の内容は、次のとおりです。

- 列オブジェクトのロード
- オブジェクト表のロード
- REF 列のロード
- LOB のロード
- コレクション（ネストした表および VARRAY）のロード
- 動的および静的 SDF 指定
- 親表を子表から分割してのロード

## 列オブジェクトのロード

制御ファイルの列オブジェクトは、その属性によって記述されています。列オブジェクトの基になるオブジェクト型が `NOT FINAL` であると宣言されると、制御ファイルの列オブジェクトは、基になるオブジェクト型から導出されたサブタイプの（導出および宣言された）属性によって記述されます。データ・ファイルでは、列オブジェクトの各属性に対応するデータは、単純なりレーショナル列に対応するデータ・フィールドと同様の形式でデータ・ファイルに記述されています。

---

---

**注意：** `SQL*Loader` による列オブジェクトなどの複合データ型のサポートによって、制御ファイル内に、2つの同じフィールド名が存在する可能性があります。1つは列に対応し、もう1つは列オブジェクトの属性に対応する場合があります。制御ファイルに同名のフィールドが存在する場合、特定の句がフィールド（たとえば、`WHEN`、`NULLIF`、`DEFAULTIF`、`SID`、`OID`、`REF`、`BFILE` など）を参照できるため、名前の重複が発生する場合があります。

そのため、フィールドを参照する句を使用する場合は、フルネームで指定する必要があります。たとえば、フィールド `f1d1` が `COLUMN OBJECT` に指定されており、そこにフィールド `f1d2` が含まれる場合、`NULLIF` などの句で `f1d2` を指定する場合は、フィールドのフルネーム `f1d1.f1d2` を使用する必要があります。

---

---

次に、列オブジェクトのロードに関する例を示します。

- ストリーム・レコード形式への列オブジェクトのロード
- 可変レコード形式への列オブジェクトのロード
- ネストした列オブジェクトのロード
- 導出サブタイプを使用した列オブジェクトのロード
- オブジェクトに対する `NULL` 値の指定
- ユーザー定義コンストラクタを使用した列オブジェクトのロード

## ストリーム・レコード形式への列オブジェクトのロード

例 7-1 に、事前にサイズが決まっているフィールドにデータがある例を示します。終了文字は、物理レコードの終わりを示します。オペレーティング・システムのファイル処理句 (`os_file_proc_clause`) のカスタム・レコード・セパレータを使用して、物理レコードの終わりを示すこともできます。

**例 7-1 ストリーム・レコード形式への列オブジェクトのロード****制御ファイルの内容**

```

LOAD DATA
INFILE 'sample.dat'
INTO TABLE departments
  (dept_no    POSITION(01:03)    CHAR,
   dept_name  POSITION(05:15)    CHAR,
1  dept_mgr   COLUMN OBJECT
   (name      POSITION(17:33)    CHAR,
    age       POSITION(35:37)    INTEGER EXTERNAL,
    emp_id    POSITION(40:46)    INTEGER EXTERNAL) )

```

**データ・ファイル (sample.dat)**

```

101 Mathematics  Johny Quest      30    1024
237 Physics      Albert Einstein  65    0000

```

**注意：**

1. この列オブジェクトの型指定は、ネストした列オブジェクトの記述にも繰り返し使用できます。

**可変レコード形式への列オブジェクトのロード**

例 7-2 に、デリミタ・フィールドにデータがある例を示します。

**例 7-2 可変レコード形式への列オブジェクトのロード****制御ファイルの内容**

```

LOAD DATA
1 INFILE 'sample.dat' "var 6"
INTO TABLE departments
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
2 (dept_no
   dept_name,
   dept_mgr   COLUMN OBJECT
   (name      CHAR(30),
    age       INTEGER EXTERNAL(5),
    emp_id    INTEGER EXTERNAL(5)) )

```

### データ・ファイル (sample.dat)

```
3  000034101,Mathematics,Johny Q.,30,1024,  
    000039237,Physics,"Albert Einstein",65,0000,
```

#### 注意:

1. "var" 文字列には、各レコードの先頭にある長さフィールドのバイト数（この例では 6）が含まれます。値が指定されない場合、デフォルトは 5 バイトです。可変レコードの最大サイズは、2 の 32 乗 -1 で、それ以上の値を指定するとエラーになります。
2. 位置を指定しなくても、一般構文では同じ結果（列オブジェクトの名前の後に、カッコで囲まれた属性性のリストが続く）になります。また、省略された型指定については、デフォルトで長さが 255 の CHAR 型になります。
3. 最初の 6 バイト（斜体で示した部分）に、次のレコードの長さを指定します。これらの長さ指定には、emp\_id フィールドの後の終了記号のために無視される改行文字も含まれます。

## ネストした列オブジェクトのロード

例 7-3 に、ネストした列オブジェクト（他の列オブジェクト内にネストした 1 つの列オブジェクト）の制御ファイルの記述方法を示します。

### 例 7-3 ネストした列オブジェクトのロード

#### 制御ファイルの内容

```
LOAD DATA  
INFILE `sample.dat`  
INTO TABLE departments_v2  
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '''  
    (dept_no      CHAR(5),  
     dept_name    CHAR(30),  
     dept_mgr     COLUMN OBJECT  
       (name      CHAR(30),  
        age       INTEGER EXTERNAL(3),  
        emp_id    INTEGER EXTERNAL(7),  
1    em_contact  COLUMN OBJECT  
       (name      CHAR(30),  
        phone_num CHAR(20))))
```

### データ・ファイル (sample.dat)

```
101,Mathematics,Johny Q.,30,1024,"Barbie",650-251-0010,  
237,Physics,"Albert Einstein",65,0000,Wife Einstein,654-3210,
```

**注意：**

1. このエントリでは、列オブジェクトにネストした列オブジェクトを指定します。

**導出サブタイプを使用した列オブジェクトのロード**

例 7-4 に、NOT FINAL のベース・オブジェクト型を拡張して新しく導出されたサブタイプを作成する例を示します。表定義の列オブジェクトは、ベース・オブジェクト型であると宣言されますが、サブタイプがベース・オブジェクト型から導出される場合は、SQL\*Loader によってサブタイプを列オブジェクトにロードできます。

**例 7-4 サブタイプを使用した列オブジェクトのロード****オブジェクト型定義**

```
CREATE TYPE person_type AS OBJECT
  (name      VARCHAR(30),
   ssn       NUMBER(9)) not final;

CREATE TYPE employee_type UNDER person_type
  (empid     NUMBER(5));

CREATE TABLE personnel
  (deptno    NUMBER(3),
   deptname  VARCHAR(30),
   person    person_type);
```

**制御ファイルの内容**

```
LOAD DATA
  INFILE 'sample.dat'
  INTO TABLE personnel
  FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
    (deptno      INTEGER EXTERNAL(3),
     deptname    CHAR,
  1  person      COLUMN OBJECT TREAT AS employee_type
      (name      CHAR,
       ssn       INTEGER EXTERNAL(9),
  2  empid       INTEGER EXTERNAL(5)))
```

**データ・ファイル (sample.dat)**

```
101,Mathematics,Johny Q.,301189453,10249,
237,Physics,"Albert Einstein",128606590,10030,
```

注意：

- 1. TREAT AS 句で指定すると、SQL\*Loader では、実際の宣言型が person\_type である列オブジェクト person が、導出型 employee\_type であると宣言されたかのように処理されます。
- 2. empid 属性は employee\_type の属性であるため、ここで使用できます。TREAT AS 句が指定されていない場合、この属性は、列の宣言型の属性ではないためエラーを返します。

## オブジェクトに対する NULL 値の指定

非スカラー・データ型で NULL 値を指定する場合、スカラー・データ型で指定するよりも複雑です。オブジェクトは、その属性のサブセットを NULL にするか、すべての属性を NULL (NULL オブジェクトにかぎります) にするか、またはオブジェクト自身を NULL (アトミック NULL オブジェクト) にできます。

### NULL 属性の指定

オブジェクト列に対応するフィールドでは、NULLIF 句を使用して、特殊属性を NULL に初期化するフィールド条件を指定できます。例 7-5 に例を示します。

例 7-5 NULLIF 句を使用した NULL 属性の指定

#### 制御ファイルの内容

```
LOAD DATA
INFILE 'sample.dat'
INTO TABLE departments
  (dept_no      POSITION(01:03)    CHAR,
   dept_name    POSITION(05:15)    CHAR NULLIF dept_name=BLANKS,
   dept_mgr     COLUMN OBJECT
1  ( name      POSITION(17:33)    CHAR NULLIF dept_mgr.name=BLANKS,
1  age        POSITION(35:37)    INTEGER EXTERNAL
                                NULLIF dept_mgr.age=BLANKS,
1  emp_id     POSITION(40:46)    INTEGER EXTERNAL
                                NULLIF dept_mgr.emp_id=BLANKS))
```

#### データ・ファイル (sample.dat)

```
2 101          Johnty Quest          1024
   237 Physics Albert Einstein    65    0000
```

**注意：**

1. 各属性に対応する NULLIF 句は、属性値を NULL にする条件を示します。
2. dept\_mgr の age 属性の値は NULL です。dept\_name の値も NULL です。

**アトミック NULL の指定**

列オブジェクトが NULL 値（アトミック NULL）を取る条件を制御ファイルで指定するには、NULLIF 句で使用するオブジェクトの名前は、マップ・フィールドの論理的な組合せに基づいている必要があります（たとえば、例 7-5 では、指定されたマップ・フィールドは、dept\_no、dept\_name、name、age および emp\_id です。dept\_mgr は、データ・ファイルのどのフィールドにも対応していない（マップされていない）ため、指定されたマップ・フィールドではありません）。

前述の方法は使用可能ですが、オブジェクトが NULL 値を取るための条件が、マップ・フィールドに依存していない場合は、理想的な方法ではありません。このような場合は、FILLER フィールドを使用できます。

FILLER フィールドをデータ・ファイルのフィールドにマップし（列オブジェクトがアトミック NULL かどうかを示す）、その FILLER フィールドを列オブジェクトの NULLIF 句のフィールド条件で使用できます。この例を例 7-6 に示します。

**例 7-6 FILLER フィールドを使用したデータのロード****制御ファイルの内容**

```
LOAD DATA
INFILE 'sample.dat'
INTO TABLE departments_v2
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
  (dept_no          CHAR(5),
   dept_name        CHAR(30),
1  is_null          FILLER CHAR,
2  dept_mgr         COLUMN OBJECT NULLIF is_null=BLANKS
   (name            CHAR(30) NULLIF dept_mgr.name=BLANKS,
    age              INTEGER EXTERNAL(3) NULLIF dept_mgr.age=BLANKS,
    emp_id           INTEGER EXTERNAL(7)
                        NULLIF dept_mgr.emp_id=BLANKS,
    em_contact       COLUMN OBJECT NULLIF is_null2=BLANKS
      (name          CHAR(30)
                        NULLIF dept_mgr.em_contact.name=BLANKS,
      phone_num      CHAR(20)
                        NULLIF dept_mgr.em_contact.phone_num=BLANKS)),
1  is_null2         FILLER CHAR)
```

### データ・ファイル (sample.dat)

```
101,Mathematics,n,Johnny Q.,,1024,"Barbie",608-251-0010,,  
237,Physics,,,"Albert Einstein",65,0000,,650-654-3210,n,
```

#### 注意：

1. FILLER フィールド（データ・ファイルがマップされており、対応する列がない）は CHAR 型（デリミタ付きフィールドであるため、CHAR のデフォルトは CHAR(255)）のフィールドです。NULLIF 句は、FILLER フィールド自体には使用できません。
2. is\_null フィールドが空白の場合は、NULL（アトミック NULL）の値を取得します。

参照： 6-6 ページの「[FILLER フィールドの指定](#)」を参照してください。

## ユーザー定義コンストラクタを使用した列オブジェクトのロード

Oracle9i データベース・サーバーでは、すべてのオブジェクト型に対してデフォルトのコンストラクタが自動的に提供されます。このコンストラクタを使用するには、コンストラクタに対するコールで、その型のすべての属性を引数として指定する必要があります。オブジェクトの新しいインスタンスが作成されると、その属性は引数リスト内の対応する値を取ります。このコンストラクタは、属性値コンストラクタと呼ばれます。SQL\*Loader では、列オブジェクトのロード時に、デフォルトで属性値コンストラクタが使用されます。

1 つ以上のユーザー定義コンストラクタを作成することによって、属性値コンストラクタを上書きできます。ユーザー定義コンストラクタを作成する場合、オブジェクトの新しいインスタンスが作成されると、常に、ユーザー定義論理を実行する型本体を指定する必要があります。ユーザー定義コンストラクタの引数リストは、属性値コンストラクタと同じ場合もありますが、型本体で実装される論理は異なります。

ユーザー定義コンストラクタのファンクションの引数リストが属性値コンストラクタの引数リストと一致する場合、従来型パスとダイレクト・パスでは SQL\*Loader の動作が異なります。従来型パス・モードでは、ユーザー定義コンストラクタがコールされます。ダイレクト・パス・モードでは、属性値コンストラクタがコールされます。この相違点を例 7-7 に示します。



## 例 7-7 属性値コンストラクタに一致するユーザー定義コンストラクタを使用した列オブジェクトのロード

### オブジェクト型定義

```
CREATE TYPE person_type AS OBJECT
  (name      VARCHAR(30),
   ssn       NUMBER(9)) not final;

CREATE TYPE employee_type UNDER person_type
  (empid     NUMBER(5),
   -- User-defined constructor that looks like an attribute-value constructor
   CONSTRUCTOR FUNCTION
     employee_type (name VARCHAR2, ssn NUMBER, empid NUMBER)
     RETURN SELF AS RESULT);

CREATE TYPE BODY employee_type AS
  CONSTRUCTOR FUNCTION
    employee_type (name VARCHAR2, ssn NUMBER, empid NUMBER)
    RETURN SELF AS RESULT AS
  -- This UDC makes sure that the name attribute is in uppercase.
  BEGIN
    SELF.name := UPPER(name);
    SELF.ssn  := ssn;
    SELF.empid := empid;
    RETURN;
  END;

CREATE TABLE personnel
  (deptno     NUMBER(3),
   deptname   VARCHAR(30),
   employee   employee_type);
```

### 制御ファイルの内容

```
LOAD DATA
  INFILE *
  REPLACE
  INTO TABLE personnel
  FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
    (deptno      INTEGER EXTERNAL(3),
     deptname    CHAR,
     employee    COLUMN OBJECT
       (name     CHAR,
        ssn      INTEGER EXTERNAL(9),
        empid    INTEGER EXTERNAL(5)))

  BEGINDATA
```

```
1 101,Mathematics,Johnny Q.,301189453,10249,
   237,Physics,"Albert Einstein",128606590,10030,
```

### 注意：

1. この制御ファイルが従来型パス・モードで実行された場合、名前フィールドの Johnny Q. および Albert Einstein は、両方とも大文字でロードされます。これは、従来型パス・モードではユーザー定義コンストラクタがコールされるためです。これに対し、この制御ファイルがダイレクト・パス・モードで実行された場合、名前フィールドは入力データに表示されているとおりにロードされます。これは、ダイレクト・パス・モードでは属性値コンストラクタがコールされるためです。

引数リストが属性値コンストラクタと一致しないユーザー定義コンストラクタの作成もできます。この場合は、従来型パス・モードおよびダイレクト・パス・モードの両方で、属性値コンストラクタがコールされます。例 7-8 に示す定義について考えてみます。

### 例 7-8 属性値コンストラクタに一致しないユーザー定義コンストラクタを使用した列オブジェクトのロード

#### オブジェクト型定義

```
CREATE SEQUENCE employee_ids
  START      WITH 1000
  INCREMENT BY 1;

CREATE TYPE person_type AS OBJECT
  (name      VARCHAR(30),
   ssn       NUMBER(9)) not final;

CREATE TYPE employee_type UNDER person_type
  (empid      NUMBER(5),
   -- User-defined constructor that does not look like an attribute-value
   -- constructor
   CONSTRUCTOR FUNCTION
     employee_type (name VARCHAR2, ssn NUMBER)
     RETURN SELF AS RESULT);

CREATE TYPE BODY employee_type AS
  CONSTRUCTOR FUNCTION
    employee_type (name VARCHAR2, ssn NUMBER)
    RETURN SELF AS RESULT AS
  -- This UDC makes sure that the name attribute is in lowercase and
  -- assigns the employee identifier based on a sequence.
    nextid      NUMBER;
    stmt        VARCHAR2(64);
  BEGIN
```

```

stmt := 'SELECT employee_ids.nextval FROM DUAL';
EXECUTE IMMEDIATE stmt INTO nextid;

SELF.name := LOWER(name);
SELF.ssn := ssn;
SELF.empid := nextid;
RETURN;
END;

CREATE TABLE personnel
(deptno NUMBER(3),
deptname VARCHAR(30),
employee employee_type);

```

例 7-7 で説明した制御ファイルがこれらの定義に従って使用された場合、名前フィールドは、入力データに表示されているとおりに（大文字と小文字の組合せで）ロードされます。これは、従来型パス・モードおよびダイレクト・パス・モードの両方で属性値コンストラクタがコールされるためです。

SQL 式でユーザー定義コンストラクタを明示的に参照することによって、従来型パス・モードを使用してこの表をロードすることもできます。例 7-9 に例を示します。

#### 例 7-9 SQL 式を使用した、属性値コンストラクタに一致しないユーザー定義コンストラクタでの列オブジェクトのロード

##### 制御ファイルの内容

```

LOAD DATA
  INFILE *
  REPLACE
  INTO TABLE personnel
  FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
    (deptno      INTEGER EXTERNAL(3),
     deptname    CHAR,
     name        BOUNDFILLER CHAR,
     ssn         BOUNDFILLER INTEGER EXTERNAL(9),
     employee    EXPRESSION "employee_type(:NAME, :SSN)")

  BEGINDATA
1 101,Mathematics,Johnny Q.,301189453,
  237,Physics,"Albert Einstein",128606590,

```

### 注意：

1. この場合、従業員列オブジェクトが、SQL 式を使用してロードされます。この式によって、正しい数の引数が含まれているユーザー定義コンストラクタが起動されます。名前フィールドの Johny Q. および Albert Einstein は、両方とも小文字でロードされます。また、各行の従業員列オブジェクトの従業員識別子は、employee\_ids 順序番号から値を取ります。

例 7-9 に示した制御ファイルがダイレクト・パス・モードで使用された場合は、次のエラーが通知されます。

```
SQL*Loader-00951: once/load 初期化呼出しでエラーが発生しました。  
ORA-26052: SQL 式の型 121 ( 列 EMPLOYEE 上 ) はサポートされていません。
```

## オブジェクト表のロード

オブジェクト表のロードに必要な制御ファイルの構文は、典型的なリレーショナル表のロードの場合とほぼ同じです。例 7-10 に、主キー OID を使用したオブジェクト表のロード例を示します。

### 例 7-10 主キー OID を使用したオブジェクト表のロード

#### 制御ファイルの内容

```
LOAD DATA  
INFILE 'sample.dat'  
DISCARDFILE 'sample.dsc'  
BADFILE 'sample.bad'  
REPLACE  
INTO TABLE employees  
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'  
  (name      CHAR(30)                NULLIF name=BLANKS,  
   age       INTEGER EXTERNAL(3)     NULLIF age=BLANKS,  
   emp_id    INTEGER EXTERNAL(5))
```

#### データ・ファイル (sample.dat)

```
Johny Quest, 18, 007,  
Speed Racer, 16, 000,
```

前述の制御ファイルを見ただけでは、ロードされる表がシステム生成 OID (実 OID) を持つオブジェクト表か、主キー OID を持つオブジェクト表か、またはリレーショナル表かを判断できません。

すでに実 OID を含むデータをロードし、新しい OID を生成するのではなく、データ・ファイル内の既存の OID を使用するように指定する必要がある場合もあります。そのような場合は、INTO TABLE 句に続けて OID 句を使用します。

OID (*fieldname*)

この場合、*fieldname* には、実 OID を含むデータ・ファイルにマップされた、フィールド指定リストのフィールド名（通常は FILLER フィールド）を指定します。SQL\*Loader では、その指定された OID が、正しい形式で、グローバルな独自性を保持した OID であるとみなされます。そのため、Oracle の OID ジェネレータを使用して OID を生成し、ロードされた OID の一意性を確保する必要があります。

また、その OID 句は、主キー OID ではなく、システム生成の OID でのみ使用できます。

例 7-11 に、行オブジェクトを使用した実 OID のロード例を示します。

#### 例 7-11 OID のロード

##### 制御ファイルの内容

```
LOAD DATA
  INFILE 'sample.dat'
  INTO TABLE employees_v2
1  OID (s_oid)
  FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
    (name      CHAR(30)                NULLIF name=BLANKS,
     age       INTEGER EXTERNAL(3)     NULLIF age=BLANKS,
     emp_id    INTEGER EXTERNAL(5),
2    s_oid     FILLER CHAR(32))
```

##### データ・ファイル (sample.dat)

```
3  Johnny Quest, 18, 007, 21E978406D3E41FCE03400400B403BC3,
   Speed Racer, 16, 000, 21E978406D4441FCE03400400B403BC3,
```

##### 注意：

1. OID 句には、s\_oid のロード・フィールドが OID を含むように指定します。カッコが必要です。
2. s\_oid に有効な 16 進数が含まれていない場合、そのレコードは拒否されます。
3. データ・ファイルの OID は文字列で、32 バイトの 16 進数として解釈されます。32 バイトの 16 進数は、後で 16 バイトの RAW に変換されてオブジェクト表に格納されます。

## サブタイプを使用したオブジェクト表のロード

オブジェクト表の列オブジェクトが NOT FINAL に基づいている場合、SQL\*Loader によって導出サブタイプをオブジェクト表にロードできます。前述のとおり、オブジェクト表に導出サブタイプをロードする場合に必要な構文は、典型的なリレーショナル表のロードの場合とほぼ同じです。ただし、この場合、実際に使用するサブタイプがオブジェクト表で有効であるかどうかを SQL\*Loader で判断できるように、サブタイプに名前を指定する必要があります。この概念を例 7-12 に示します。

### 例 7-12 サブタイプを使用したオブジェクト表のロード

#### オブジェクト型定義

```
CREATE TYPE employees_type AS OBJECT
  (name      VARCHAR2(30),
   age       NUMBER(3),
   emp_id    NUMBER(5)) not final;

CREATE TYPE hourly_emps_type UNDER employees_type
  (hours     NUMBER(3));

CREATE TABLE employees_v3 OF employees_type;
```

#### 制御ファイルの内容

```
LOAD DATA

  INFILE 'sample.dat'
  INTO TABLE employees_v3
1  TREAT AS hourly_emps_type
  FIELDS TERMINATED BY ','
    (name      CHAR(30),
     age       INTEGER EXTERNAL(3),
     emp_id    INTEGER EXTERNAL(5),
2  hours      INTEGER EXTERNAL(2))
```

#### データ・ファイル (sample.dat)

```
Johny Quest, 18, 007, 32,
Speed Racer, 16, 000, 20,
```

**注意：**

1. TREAT AS 句で指定すると、SQL\*Loader では、実際の宣言型が `employee_type` であるオブジェクト表が、`hourly_emps_type` 型であると宣言されたかのように処理されます。
2. `hours` 属性は `hourly_emps_type` の属性であるため、ここで使用できます。TREAT AS 句が指定されていない場合、この属性は、オブジェクト表の宣言型の属性ではないためエラーを返します。

## REF 列のロード

SQL\*Loader では、実 REF 列（参照しているオブジェクトの実 OID を含む REF）、主キー REF 列、および主キーを使用可能な有効範囲なし REF 列をロードできます。

### 実 REF 列

実 REF 列をロードする場合、SQL\*Loader では、REF 列を構築する実 OID が残りのデータとともにデータ・ファイル内にあるとみなされます。REF 列に対応するフィールドの記述は、列名の後に REF 句を記述することによって行います。

REF 句には、引数として表名と OID を取ります。その引数は、定数として、または（FILLER フィールドを使用して）動的に指定できます。適切な構文の詳細は、A-7 ページの [ref\\_spec](#) を参照してください。例 7-13 に、実 REF のロード例を示します。

#### 例 7-13 実 REF 列のロード

##### 制御ファイルの内容

```
LOAD DATA
INFILE 'sample.dat'
INTO TABLE departments_alt_v2
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
  (dept_no      CHAR(5),
   dept_name    CHAR(30),
1 dept_mgr     REF(t_name, s_oid),
   s_oid        FILLER CHAR(32),
   t_name       FILLER CHAR(30))
```

##### データ・ファイル (sample.dat)

```
22345, QuestWorld, 21E978406D3E41FCE03400400B403BC3, EMPLOYEES_V2,
23423, Geography, 21E978406D4441FCE03400400B403BC3, EMPLOYEES_V2,
```

**注意：**

1. 指定した表が存在しない場合、レコードは拒否されます。また、dept\_mgr フィールド自体には、データ・ファイルのフィールドはマップされません。

## 主キー REF 列

主キー REF 列をロードするには、SQL\*Loader 制御ファイルのフィールドで列名の後に REF 句を記述する必要があります。REF 句には、カンマで区切ったフィールド名および定数値のリストが引数として必要です。最初の引数は表名で、その後にロードする REF 列の基になる主キー OID を指定する引数を記述します。適切な構文の詳細は、A-7 ページの「[ref\\_spec](#)」を参照してください。

SQL\*Loader では、引数の順序は、参照されている表で主キー OID を作成する列の相対順序に一致しているとみなされます。例 7-14 に、主キー REF 列のロード例を示します。

### 例 7-14 主キー REF 列のロード

**制御ファイルの内容**

```
LOAD DATA
INFILE 'sample.dat'
INTO TABLE departments_alt
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
  (dept_no          CHAR(5),
   dept_name        CHAR(30),
   dept_mgr         REF(CONSTANT 'EMPLOYEES', emp_id),
   emp_id           FILLER CHAR(32))
```

**データ・ファイル (sample.dat)**

```
22345, QuestWorld, 007,
23423, Geography, 000,
```

## 主キーが使用可能な有効範囲なし REF 列

主キーが使用可能な有効範囲なし REF 列によって、システム生成および主キーの両方の型の REF を参照できます。このような REF 列をロードするための構文は、実 REF 列または主キー REF 列をロードする場合と同じです。例 7-13「[実 REF 列のロード](#)」および例 7-14「[主キー REF 列のロード](#)」を参照してください。

主キーが使用可能な有効範囲なし REF 列をロードする場合は、次の制限が適用されます。

- 単一表へのロード中は、この列からシステム生成または主キーのいずれかの型の REF のみを参照できます。両方は参照できません。両方の型の参照を試行すると、データ行が拒否され、参照表名が無効であることを示すエラー・メッセージが表示されます。



- この列に有効範囲なし主キー REF をロードする場合、単一表へのロード中は、1 つのオブジェクト表のみ参照できます。複数の有効範囲なし主キー REF（いくつかの REF はオブジェクト表 X を指し、別の REF はオブジェクト表 Y を指す）をロードする場合、次のいずれかの操作を実行する必要があります。
  - 単一表へのロードを 2 回実行します。
  - 有効範囲なし主キー REF のオブジェクト表名など、WHEN 句によってデータのいくつかの要素が指定されている複数の INTO TABLE 句を使用して、ロードを 1 回実行します。次に例を示します。

```
LOAD DATA
INFILE 'data.dat'

INTO TABLE orders_apk
APPEND
when CUST_TBL = "CUSTOMERS_PK"
fields terminated by ","
(
  order_no position(1) char,
  cust_tbl FILLER      char,
  cust_no FILLER      char,
  cust   REF (cust_tbl, cust_no) NULLIF order_no='0'
)

INTO TABLE orders_apk
APPEND
when CUST_TBL = "CUSTOMERS_PK2"
fields terminated by ","
(
  order_no position(1) char,
  cust_tbl FILLER      char,
  cust_no FILLER      char,
  cust   REF (cust_tbl, cust_no) NULLIF order_no='0'
)
```

この 2 つの方法のいずれも使用しない場合、データ行が拒否され、参照されている表の名前が無効であることを示すエラー・メッセージが表示されます。

- コレクション内の有効範囲なし主キー REF は、SQL\*Loader ではサポートされていません。
- この REF 列にシステム生成の REF をロードする場合も、7-15 ページの「[実 REF 列](#)」で説明されている制限が適用されます。
- この REF 列に主キー REF をロードする場合も、7-16 ページの「[主キー REF 列](#)」で説明されている制限が適用されます。

---

---

**注意：** 主キーが使用可能な有効範囲なし REF 列の場合、SQL\*Loader では、(REF ディレクティブまたはデータ行のいずれかから) 最初の有効な解析済オブジェクト表が取得され、オブジェクト表の OID 型を使用してその単一表へのロードで参照可能な REF 型が決定されます。

---

---

## LOB のロード

LOB は、ラージ・オブジェクト型です。SQL\*Loader では、次の LOB 型がサポートされています。

- BLOB: 非構造化バイナリ・データを含む内部 LOB。
- CLOB: 文字データを含む内部 LOB。
- NCLOB: 各国語キャラクタ・セットの文字を含む内部 LOB。
- BFILE: サーバー側のオペレーティング・システム・ファイルのデータベース表領域外に格納される BLOB。

LOB は列データ型で、NCLOB 以外は、オブジェクトの属性データ型です。LOB は実際の値を持ち、その値は NULL でも「値なし (空)」でもかまいません。

XML 列は、SYS.XMLTYPE 型であると宣言された列です。SQL\*Loader は、XML 列を CLOB と同様に処理します。次の項で説明するプライマリ・データ・ファイルまたは LOBFILES からの LOB データのロード方法は、XML 列のロードに適用できます。

---

---

**注意：** LOB フィールドに SQL 文字列は指定できません。これは、LOBFILE\_spec を指定する場合も同じです。

---

---

LOB は非常に大きなデータであるため、SQL\*Loader では、LOB データをプライマリ・データ・ファイル (残りのデータを持つインライン) または LOBFILE のいずれかからロードできます。この項では、次の項目について説明します。

- [プライマリ・データ・ファイルからの LOB データのロード](#)
- [外部 LOBFILE \(BFILE\) からの LOB データのロード](#)
- [LOBFILE からの LOB データのロード](#)

## プライマリ・データ・ファイルからの LOB データのロード

プライマリ・データ・ファイルから内部 LOB（BLOB、CLOB および NCLOB）または XML 列をロードするには、次の標準 SQL\*Loader 形式を使用できます。

- 事前にサイズが決まっているフィールド
- デリミタ付きフィールド
- Length-Value Pair フィールド

これらの各形式については、次の項で説明します。

### 事前に決められたサイズのフィールドの LOB データ

例 7-15 に示すように、これは LOB データをロードする場合、最も高速で、概念的に単純な形式です。

---

---

**注意：** ロードする LOB データは、サイズが均等ではないため、サイズが小さいデータ・フィールドに空白を埋め込み、全 LOB データが同じサイズになるようにできます。

---

---

この形式で LOB をロードするには、ロード時のデータ型として CHAR または RAW を使用する必要があります。

#### 例 7-15 事前に決められたサイズのフィールドの LOB データ

##### 制御ファイルの内容

```
LOAD DATA
INFILE 'sample.dat' "fix 501"
INTO TABLE person_table
  (name          POSITION(01:21)          CHAR,
1 "RESUME"      POSITION(23:500)         CHAR  DEFAULTIF "RESUME"=BLANKS)
```

##### データ・ファイル (sample.dat)

```
Johny Quest      Johny Quest
                  500 Oracle Parkway
                  jquest@us.oracle.com ...
```

**注意：**

1. RESUME が含まれているデータ・フィールドが空の場合、NULL の LOB ではなく、空の LOB になります。DEFAULTIF 句のかわりに NULLIF 句を使用した場合、結果は逆になります。また、ロード時に、CHAR 以外にも SQL\*Loader のデータ型を使用できます。たとえば、BLOB のロード時、RAW データ型を使用する場合があります。

**デリミタ付きフィールドの LOB データ**

この形式で、同じ列（データ・ファイルのフィールド）で異なるサイズの LOB を、問題なく処理できます。ただし、このような柔軟性によって、SQL\*Loader で区切り文字列を探してデータをスキャンする必要があるため、パフォーマンスに影響します。

単一キャラクタのデリミタで、文字列のデリミタを指定する場合は、データ・ファイルのキャラクタ・セットに注意してください。データ・ファイルのキャラクタ・セットが制御ファイルのキャラクタ・セットと異なる場合は、デリミタを 16 進文字列の表記法で指定できます (*X'hexadecimal string'*)。デリミタを実際に 16 進文字列で指定する場合は、入力データ・ファイルのキャラクタ・セット中の有効な文字で指定する必要があります。一方、16 進文字列で指定しない場合、デリミタは、クライアント（制御ファイル）のキャラクタ・セットで指定してください。この場合、デリミタは、SQL\*Loader によってデータ・ファイル内で検索される前に、データ・ファイルのキャラクタ・セットに変換されます。

次のことに注意してください。

- 文字列デリミタを使用した区切り構文がサポートされます（囲みデリミタを区切りとします）。
- マルチ・キャラクタの囲みデリミタの前に空白は入れられません。
- フィールドが WHITESPACE で終わる場合、先頭の空白は切り捨てられます。

例 7-16 にデリミタ付きフィールドへの LOB データのロードの例を示します。

**例 7-16 デリミタ付きフィールドの LOB データのロード****制御ファイルの内容**

```
LOAD DATA
INFILE 'sample.dat' "str '|' "
INTO TABLE person_table
FIELDS TERMINATED BY ','
  (name          CHAR(25),
1  "RESUME"      CHAR(507) ENCLOSED BY '<startlob>' AND '<endlob>')
```

**データ・ファイル (sample.dat)**

```
Johny Quest,<startlob>          Johny Quest
                                500 Oracle Parkway
                                jquest@us.oracle.com ...   <endlob>

2 |Speed Racer, .....
```

**注意：**

1. <startlob> および <endlob> は、囲み文字列です。デフォルトのバイト長セマンティクスでは、CHAR(507) を使用して読み込むことができる LOB の最大長は 507 バイトです。文字長セマンティクスが使用された場合、最大長は 507 文字になります。詳細は、5-22 ページの「[文字長セマンティクス](#)」を参照してください。
2. レコード・セパレータ '|' は、<endlob> のすぐ後にあり、その後に改行文字が続く場合、改行は、次のレコードの一部として解釈されます。代替方法は、レコード・セパレータに改行部分を作成することです（たとえば '|' \n'、または 16 進では X'7C0A'）。

**Length-Value Pair フィールドの LOB データ**

VARCHAR、VARCHARC または VARRAW データ型を使用して、Length-Value Pair フィールドで編成された LOB データをロードできます。このロード方法を使用すると、デリミタ付きフィールドを使用するより高いパフォーマンスを得ることができます。ただし、柔軟性は損なわれます（たとえば、ロード前に各 LOB の長さの確認が必要です）。例 7-17 に、Length-Value Pair フィールドの LOB データのロード例を示します。

**例 7-17 Length-Value Pair フィールドへの LOB データのロード****制御ファイルの内容**

```
LOAD DATA
1 INFILE 'sample.dat' "str '<endrec>\n'"
  INTO TABLE person_table
  FIELDS TERMINATED BY ','
    (name      CHAR(25),
2    "RESUME"   VARCHARC(3,500))
```

**データ・ファイル (sample.dat)**

```
Johnny Quest,479          Johnny Quest
                           500 Oracle Parkway
                           jquest@us.oracle.com
                           ... <endrec>
3   Speed Racer,000<endrec>
```

**注意：**

1. バックスラッシュ・エスケープ文字がサポートされていない場合、例の中でレコード・セパレータとして使用されている文字列は、16 進数の表記法で表現されます。
2. "RESUME" は、CLOB 列に対応するフィールドです。制御ファイルでは、VARCHARC がそのフィールドで、フィールド長は 3 バイト、最大サイズは 500 バイト（バイト長セマンティクス）です。文字長セマンティクスが使用された場合、長さは 3 文字で最大サイズは 500 文字です。詳細は、5-22 ページの「[文字長セマンティクス](#)」を参照してください。
3. VARCHARC の length サブフィールドは、0（サブフィールドの値が空）です。このため、LOB インスタンスは、空に初期化されます。

## 外部 LOBFILE (BFILE) からの LOB データのロード

BFILE データ型には、データベースの外側にあるオペレーティング・システム・ファイルの非構造化バイナリ・データが格納されます。BFILE 列または属性には、データを含む外部ファイルを示すロケータが格納されます。BFILE としてロードされるファイルは、ロード時に存在している必要はなく、後で作成できます。SQL\*Loader では、必要なオブジェクト（サーバーのファイル・システム上の物理ディレクトリに対する論理的な別名）がすでに作成されていると想定しています。詳細は、『Oracle9i アプリケーション開発者ガイド - ラージ・オブジェクト』を参照してください。

BFILE 列に対応する制御ファイルのフィールドの記述は、列名の後に BFILE 句を記述して行います。BFILE 句には、引数として DIRECTORY OBJECT (server\_directory の別名) 名の後に BFILE 名が必要です。いずれの引数も文字列定数として指定するか、他のフィールドを介して動的にロードできます。詳細は、『Oracle9i SQL リファレンス』を参照してください。

次の 2 つの BFILE のロード例を示します。[例 7-18](#) では、1 つのファイル名のみを動的に指定する方法を、[例 7-19](#) では、BFILE と DIRECTORY OBJECT の両方を動的に指定する方法を示します。

### 例 7-18 BFILE を使用したデータのロード：ファイル名のみを動的に指定

#### 制御ファイルの内容

```
LOAD DATA
INFILE sample.dat
INTO TABLE planets
FIELDS TERMINATED BY ','
  (pl_id      CHAR(3),
   pl_name    CHAR(20),
   fname      FILLER CHAR(30),
1  pl_pict    BFILE(CONSTANT "scott_dir1", fname))
```

**データ・ファイル (sample.dat)**

```
1,Mercury,mercury.jpeg,
2,Venus,venus.jpeg,
3,Earth,earth.jpeg,
```

**注意：**

1. ディレクトリ名は引用符に囲まれており、そのまま使用されるため、文字列は大文字にせずそのまま指定します。

**例 7-19 BFILE を使用したデータのロード：ファイル名およびディレクトリ名を動的に指定****制御ファイルの内容**

```
LOAD DATA
INFILE sample.dat
INTO TABLE planets
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
  (pl_id    NUMBER(4),
   pl_name  CHAR(20),
   fname    FILLER CHAR(30),
1  dname    FILLER CHAR(20),
   pl_pict  BFILE(dname, fname) )
```

**データ・ファイル (sample.dat)**

```
1, Mercury, mercury.jpeg, scott_dir1,
2, Venus, venus.jpeg, scott_dir1,
3, Earth, earth.jpeg, scott_dir2,
```

**注意：**

1. dname は、ロードしたファイルに対応するディレクトリ名を含む、データ・ファイルのフィールドにマップされています。

**LOBFILE からの LOB データのロード**

LOB データは、非常に長いデータであるため、プライマリ・データ・ファイルからではなく、LOBFILE からロードすると有効です。LOBFILE では、LOB データのインスタンスは、フィールド（事前に決められたサイズ、デリミタ付き、Length-Value）内にあるとみなされますが、これらのフィールドは、レコードに編成されていません（LOBFILE にはレコードの概念がありません）。そのため、レコードを扱うことによって発生する処理のオーバーヘッドを回避できます。このようなデータの編成方法は、LOB のロードにとって理想的です。

LOBFILE からロードした LOB をメモリーに合せる必要はありません。SQL\*Loader では、64KB 単位で LOBFILE が読み込まれます。

LOBFILE のデータ・フィールドは、次のいずれかの型です。

- ファイルの内容全体を読み込む単一の LOB フィールド
- サイズが決められたフィールド（固定長フィールド）
- デリミタ付きフィールド（TERMINATED BY または ENCLOSED BY）

PRESERVE BLANKS 句は、LOBFILE から読み込むフィールドには使用できません。

- Length-Value Pair フィールド（可変長フィールド）: SQL\*Loader の VARRAW、VARCHAR、VARCHARC などのデータ型は、この型のフィールドに使用されます。

これらの各フィールド型の使用例については、7-24 ページの「[LOBFILE からの LOB データのロードの例](#)」を参照してください。前述のすべてのフィールド型は、XML 列のロードに使用できます。

LOBFILE の構文については、A-8 ページの「[lobfile\\_spec](#)」を参照してください。

### 動的および静的 LOBFILE 指定

LOBFILE を静的に指定（制御ファイルにファイル名を指定）するか、または動的に指定（FILLER フィールドをファイル名のソースとして使用）できます。いずれの場合も、LOBFILE が EOF で終了しない場合は、LOBFILE の終わりに到達するとファイルがクローズされ、そのファイルからさらにデータを読み込む場合は、空のフィールドからデータを読み込むことになります。

ただし、LOBFILE を EOF で終了する場合は、ファイルからデータを読み込むと、常に、ファイル全体が戻されます。

同じ LOBFILE を、2 つの異なるフィールドのソースとして指定しないでください。指定すると、通常、2 つのフィールドで、データが別々に読み込まれます。

### LOBFILE からの LOB データのロードの例

この項では、LOBFILE の異なるフィールド型からデータをロードする例を示します。

**ファイル当たり 1 つの LOB** [例 7-20](#) では、各 LOBFILE は、それぞれ 1 つの LOB のソースです。この方法で編成された LOB データをロードするには、LOBFILE データ型の指定に従った列またはフィールド名を使用します。



## 例 7-20 LOBFILE 当たり 1 つの LOB を使用した LOB データのロード

### 制御ファイルの内容

```
LOAD DATA
INFILE 'sample.dat'
  INTO TABLE person_table
  FIELDS TERMINATED BY ','
  (name      CHAR(20),
1  ext_fname  FILLER CHAR(40),
2  "RESUME"   LOBFILE(ext_fname) TERMINATED BY EOF)
```

### データ・ファイル (sample.dat)

```
Johny Quest,jqresume.txt,
Speed Racer,'/private/sracer/srresume.txt',
```

### SDF (jqresume.txt)

```
Johny Quest
500 Oracle Parkway
...
```

### SDF (srresume.txt)

```
Speed Racer
400 Oracle Parkway
...
```

### 注意：

1. FILLER フィールドは、SQL\*Loader の CHAR データ型を使用して読み込まれる、40 バイトのデータ・フィールドにマップされています。ここでは、デフォルトのバイト長セマンティクスの使用を想定しています。文字長セマンティクスが使用された場合、フィールドは 40 文字データ・フィールドにマップされます。
2. SQL\*Loader では、FILLER フィールド ext\_fname の LOBFILE 名が使用されます。(CHAR データ型を使用する) LOBFILE の最初のバイトから EOF 文字までのデータがロードされます。既存の LOBFILE が指定されていない場合、RESUME フィールドは空に初期化されます。

**事前に決められたサイズの LOB** 例 7-21 では、制御ファイルの特定の列にロードする LOB のサイズを指定します。ロード時、列にロードした LOB データは、指定したサイズとみなされます。事前に決められたサイズのフィールドでは、データ解析機能を最適に実行できません。ただし、すべての LOB データが必ずしも同じサイズであるとはかぎりません。

**例 7-21 事前に決められたサイズの LOB を使用した LOB データのロード****制御ファイルの内容**

```
LOAD DATA
INFILE 'sample.dat'
INTO TABLE person_table
FIELDS TERMINATED BY ','
      (name      CHAR(20),
1  "RESUME"      LOBFILE(CONSTANT '/usr/private/jquest/jqresume.txt')
                  CHAR(2000))
```

**データ・ファイル (sample.dat)**

```
Johny Quest,
Speed Racer,
```

**SDF (jqresume.txt)**

```
      Johny Quest
500 Oracle Parkway
...
      Speed Racer
400 Oracle Parkway
...
```

**注意：**

1. このエントリでは、現行のロード・セッション中、最後にロードされたバイト位置に続けてロードを開始し、CHAR データ型を使用して、jqresume.txt LOBFILE から 2000 バイトのデータをロードするように指定しています。ここでは、デフォルトのバイト長セマンティクスの使用を想定しています。文字長セマンティクスが使用された場合、SQL\*Loader では、最後にロードされた文字の直後の文字から順番に、2000 文字のデータがロードされます。詳細は、5-22 ページの「[文字長セマンティクス](#)」を参照してください。

**デリミタ付きフィールドの LOB** 例 7-22 では、LOBFILE がデリミタ付きフィールドである場合の、LOB データの例を示します。この形式では、サイズの異なる LOB を同じ列にロードしても、問題は発生しません。ただし、このような柔軟性によって、SQL\*Loader で区切り文字列を探してデータをスキャンする必要があるため、パフォーマンスに影響します。

**例 7-22 デリミタ付きフィールドの LOB を使用した LOB データのロード****制御ファイルの内容**

```
LOAD DATA
INFILE 'sample.dat'
INTO TABLE person_table
FIELDS TERMINATED BY ','
      (name      CHAR(20),
1  "RESUME"      LOBFILE( CONSTANT 'jqresume') CHAR(2000)
      TERMINATED BY "<endlob>\n")
```

**データ・ファイル (sample.dat)**

```
Johny Quest,
Speed Racer,
```

**SDF (jqresume.txt)**

```
      Johny Quest
500 Oracle Parkway
... <endlob>
      Speed Racer
400 Oracle Parkway
... <endlob>
```

**注意：**

1. CHAR の最大長に 2000 が指定されているため、SQL\*Loader で、フィールドの最大長を推測でき、メモリーの使用量を最適化できます。最大長を指定する場合、小さすぎる値は指定しないように注意してください。TERMINATED BY 句は、LOB のロードを終了する文字列を指定します。かわりに、ENCLOSED BY 句も使用できます。ENCLOSED BY 句を使用すると、LOBFILE (LOBFILE 内の LOB には順序が不要) 内での LOB の相対的な位置指定に関して、多少柔軟に対応できます。

**Length-Value Pair で指定した LOB** 例 7-23 では、LOBFILE の各 LOB の先頭でデータ長が定義されています。VARCHAR、VARCHARC または VARRAW データ型を使用して、この方法で編成された LOB データをロードできます。

このロード方法を使用すると、デリミタ付きフィールドを使用するより高いパフォーマンスを得ることができます。ただし、柔軟性は損なわれます（たとえば、各 LOB のロード前に、LOB の長さの確認が必要です）。

**例 7-23 Length-Value Pair を指定した LOB を使用した LOB データのロード****制御ファイルの内容**

```
LOAD DATA
INFILE 'sample.dat'
INTO TABLE person_table
FIELDS TERMINATED BY ','
  (name          CHAR(20),
1  "RESUME"      LOBFILE(CONSTANT 'jqresume') VARCHARC(4,2000))
```

**データ・ファイル (sample.dat)**

```
Johny Quest,
Speed Racer,
```

**SDF (jqresume.txt)**

```
2      0501Johny Quest
        500 Oracle Parkway
        ...
3      0000
```

**注意：**

1. VARCHARC(4,2000) のエントリによって、LOBFILE の LOB が Length-Value Pair 形式であり、最初の 4 バイトが長さを示すことを SQL\*Loader に指定します。この 2000 という値は、フィールドの最大サイズが 2000であることを示します。ここでは、デフォルトのバイト長セマンティクスの使用を想定しています。文字長セマンティクスが使用された場合、最初の 4 文字は文字単位の長さとして解釈されます。フィールドの最大サイズは 2000 文字です。詳細は、5-22 ページの「[文字長セマンティクス](#)」を参照してください。
2. Johny Quest の前の 0501 は、次の 501 文字が LOB のデータであることを示します。
3. このエントリは、LOB が空である (NULL ではない) ことを示します。

**LOBFILE から LOB をロードする場合の考慮点**

LOBFILE から LOB をロードする場合は、次のことに注意してください。

- LOB および XML 列のみが LOBFILE からロードできます。
- 特定の LOB のロードが失敗した場合、その LOB を含むレコードは拒否されません。かわりに、そのレコードの LOB は、空の LOB になります。XML 列の場合、LOB のロード中に失敗すると NULL 値が挿入されます。
- LOB 型の列に対するフィールドの最大長を指定する必要はありません。ただし、最大長を指定すると、メモリ使用量の最適化のヒントに使用されます。そのため、最大長の指定では、実際の最大長よりも小さい値を指定しないでください。

- LOBFILE からデータをロード中、位置指定 (pos\_spec) はできません。
- NULLIF または DEFAULTIF のフィールド条件は、LOBFILE から読み込まれたフィールドに基づくことはできません。
- 存在しない LOBFILE をフィールドのデータ・ソースに指定すると、そのフィールドは初期化されて空になります。そのフィールドを空にできない場合は、NULL に初期化されます。
- 表レベル・デリミタは、LOBFILE から読み込まれるフィールドには指定できません。
- 従来型パス・モードで SQL 式を使用して XML 列をロードするか、LOB 列を参照する場合は、SQL\*Loader で LOB データを一時 LOB として処理する必要があります。このような場合のロード・パフォーマンスを最適化するには、『Oracle9i アプリケーション開発者ガイド - ラージ・オブジェクト』の一時 LOB のパフォーマンスに関するガイドラインを参照してください。

## コレクション（ネストした表および VARRAY）のロード

LOB と同様、コレクションも、プライマリ・データ・ファイル（データ・インライン）または SDF（データ・アウトライン）のいずれかからロードできます。SDF の詳細は、7-31 ページの「[SDF](#)」を参照してください。

コレクション・データをロードする場合、コレクションに属するデータのインスタンスが終了したことを SQL\*Loader に伝える機能が必要です。これには、2 つの方法があります。

- それぞれのネストした表または VARRAY インスタンスにロードされる行または要素の数を指定するには、DDL 構文の COUNT 関数を使用します。COUNT に指定する値は、数字または数字を含む文字列のいずれかで、制御ファイルで COUNT 句よりも先に記述する必要があります。この位置の依存性は、COUNT 句に固有です。COUNT (0) または COUNT(cnt\_field) を指定して、カレント行の cnt\_field が 0 の場合は、NULLIF 句によって上書きされないかぎり、空のコレクション (NULL ではない) になります。詳細は、A-12 ページの「[count\\_spec](#)」を参照してください。
- TERMINATED BY および ENCLOSED BY 句を使用して、一意のコレクション・デリミタを指定します。SDF 句が使用されている場合、この方法は使用できません。

制御ファイルでは、コレクションは、列オブジェクトと同様に記述します。7-2 ページの「[列オブジェクトのロード](#)」を参照してください。一部、次のような相違点があります。

- コレクションの記述には、前述の 2 つの機能を使用します。
- コレクションの記述には、SDF を指定できます。
- 同じ SDF のファイル上にないかぎり、NULLIF または DEFAULTIF 句では SDF のフィールドを参照できません。
- フィールド名を引数として使用する句には、同じコレクションのフィールドに対する DDL 指定でないかぎり、コレクション内のフィールド名を使用できません。

- フィールド・リストには、非 FILLER フィールドが1つと、複数の FILLER フィールドが含まれている必要があります。VARRAY が列オブジェクトの VARRAY の場合、列オブジェクトの属性は、ネストしたフィールド・リストに記述されます。

## ネストした表および VARRAY での制限事項

ネストした表および VARRAY には次の制限事項があります。

- `field_list` には、`collection fld_spec` を含めることはできません。
- VARRAY 内にネストした `col_obj_spec` には、`collection fld_spec` を含めることはできません。
- `field_list` の一部として指定した `column_name` は、VARRAY パラメータを前に付けた `column_name` と同一である必要があります。

例 7-24 に、VARRAY およびネストした表のロード例を示します。

### 例 7-24 VARRAY およびネストした表のロード

#### 制御ファイルの内容

```
LOAD DATA
INFILE 'sample.dat' "str '\n' "
INTO TABLE dept
REPLACE
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
(
  dept_no      CHAR(3),
  dname        CHAR(25) NULLIF dname=BLANKS,
1  emps        VARRAY TERMINATED BY ':'
  (
    emps        COLUMN OBJECT
    (
      name      CHAR(30),
      age       INTEGER EXTERNAL(3),
2  emp_id     CHAR(7) NULLIF emps.emps.emp_id=BLANKS
    )
  ),
3  proj_cnt    FILLER CHAR(3),
4  projects    NESTED TABLE SDF (CONSTANT "pr.txt" "fix 57") COUNT (proj_cnt)
  (
    projects    COLUMN OBJECT
    (
      project_id  POSITION (1:5) INTEGER EXTERNAL(5),
      project_name POSITION (7:30) CHAR
                      NULLIF projects.projects.project_name = BLANKS
    )
  )
)
```

```
)
)
```

### データ・ファイル (sample.dat)

```
101,MATH,"Napier",28,2828,"Euclid", 123,9999:0
210,"Topological Transforms",:2
```

### SDF (pr.txt)

```
21034 Topological Transforms
77777 Impossible Proof
```

#### 注意：

1. TERMINATED BY 句では、VARRAY のインスタンス終了記号（COUNT 句は使用されていないことに注意）を指定します。
2. この FILLER フィールドの存在によって発生するフィールド名の競合は、フルネームによるフィールド参照（ドット表記法を使用）によって解決されます。
3. proj\_cnt は、COUNT 句に対する引数として使用する FILLER フィールドです。
4. このエントリには、次のものを指定します。
  - pr.txt と呼ばれる SDF をデータのソースとして指定します。また、SDF 内で固定レコード形式を指定します。
  - COUNT 句が 0 の場合、コレクションは空に初期化されます。コレクションを空に初期化するもう 1 つの方法は、DEFAULTIF 句を使用する方法です。ネストした表のフィールドの記述に対応するメイン・フィールド名は、そのネストした非 FILLER フィールドのフィールド名、特に、列オブジェクトのフィールド名の記述と同じです。

## SDF

SDF とプライマリ・データ・ファイルの概念は類似しています。プライマリ・データ・ファイルと同様に、SDF は、レコードおよびフィールドで構成されたレコードの集まりです。SDF は、制御ファイルごとに指定されます。SDF は、大きいネストした表および VARRAY をロードする場合に有効です。

---

**注意：** SDF をデータ・ソースとして命名できるのは、collection\_fld\_spec のみです。

---

SDF を指定するには、SDF パラメータを使用します。SDF パラメータの後に、ファイル指定文字列、または 1 つ以上のファイル指定文字列を含むデータ・フィールドにマップされた FILLER フィールドを指定します。

プライマリ・データ・ファイルについては、各 SDF に対して次の指定ができます。

- レコード形式（固定、ストリームまたは可変）の指定。また、ストリーム・レコード形式が使用される場合、レコード・セパレータを指定できます。
- レコード・サイズの指定
- CHARACTERSET 句を使用した、SDF のキャラクタ・セットの指定（5-16 ページの「異なる文字コード体系の処理」を参照）。
- 特に SDF 指定（SDF 指定を含むコレクションのすべてのメンバー・フィールドまたは属性で、LOBFILE フィールドを含むフィールドを除く）のあるフィールドに対するデフォルトのデリミタの指定（デリミタ指定を使用）

SDF については、次のことにも注意してください。

- 存在しない SDF をフィールドのデータ・ソースに指定すると、そのフィールドは初期化されて空になります。そのフィールドを空にできない場合は、NULL に初期化されます。
- 表レベル・デリミタは、SDF から読み込まれるフィールドには指定できません。
- 64KB を超える SDF をロードするには、READSIZE パラメータを使用してより大きな物理レコード・サイズを指定できます。コマンドラインからでも、OPTIONS 句の一部としても READSIZE パラメータを指定できます。

**参照：** 次の項を参照してください。

- 4-11 ページの「[READSIZE（読み込みバッファ・サイズ）](#)」
- 5-4 ページの「[OPTIONS 句](#)」
- A-11 ページの「[sdf\\_spec](#)」

## 動のおよび静的 SDF 指定

SDF を静的に指定（実際のファイル名を指定）するか、または動的に指定（FILLER フィールドをファイル名のソースとして使用）できます。いずれの場合も、SDF の EOF に到達するとファイルはクローズされ、そのファイルからさらにデータを読み込む場合は、空のフィールドからデータを読み込むことになります。

動的セカンダリ・ファイル指定では、動作は多少異なります。参照ファイルの指定が変更されると、常に、古いファイルはクローズされ、データは新しい参照先ファイルの最初から読み込まれます。

このようなデータ・ソース・ファイルの動的な切替えは、リセットの効果があります。たとえば、SQL\*Loader を使用して、現行のファイルから前回オープンしていたファイルに切り替える場合、前回オープンしていたファイルを再オープンし、そのファイルの最初からデータが読み込まれます。



同じ SDF を、2 つの異なるフィールドのソースとして指定しないでください。指定すると、通常、2 つのフィールドは、データを別々に読み込みます。

## 親表を子表から分割してのロード

ネストした表の列を含む表をロードする場合、親表を子表から分割してロードする場合があります。SID がロード時にわかっている場合（SID がデータとともにデータ・ファイルにある場合）、親表と子表を別々にロードできます。

例 7-25 および例 7-26 に、ユーザー定義 SID を使用した親表および子表のロード方法を示します。

### 例 7-25 ユーザー定義 SID を使用した親表のロード

#### 制御ファイルの内容

```
LOAD DATA
INFILE 'sample.dat' "str '|' \n" "
INTO TABLE dept
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
TRAILING NULLCOLS
( dept_no      CHAR(3),
  dname        CHAR(20) NULLIF dname=BLANKS ,
  mysid        FILLER CHAR(32),
1 projects    SID(mysid))
```

#### データ・ファイル (sample.dat)

```
101,Math,21E978407D4441FCE03400400B403BC3,|
210,"Topology",21E978408D4441FCE03400400B403BC3,|
```

#### 注意：

1. mysid は、実際の SID を含むデータ・ファイルのフィールドにマップされている FILLER フィールドで、SID 句に対する引数として指定できます。

### 例 7-26 ユーザー定義 SID を使用した子表（ネストした格納表）のロード

#### 制御ファイルの内容

```
LOAD DATA
INFILE 'sample.dat'
INTO TABLE dept
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
TRAILING NULLCOLS
1 SID(sidsrc)
(project_id    INTEGER EXTERNAL(5),
```

```
project_name  CHAR(20) NULLIF project_name=BLANKS,  
sidsrc FILLER  CHAR(32))
```

### データ・ファイル (sample.dat)

```
21034, "Topological Transforms", 21E978407D4441FCE03400400B403BC3,  
77777, "Impossible Proof", 21E978408D4441FCE03400400B403BC3,
```

### 注意：

1. 表レベルの SID 句を指定すると、SQL\*Loader によって、ネストした表の記憶表がロードされます。sidsrc は、実際の SID のソースである、FILLER フィールド名です。

## VARRAY 列ロード時のメモリーの問題

次に、VARRAY 列をロードする場合の注意事項を示します。

- VARRAY は、データベースにロードされる前にクライアント・メモリーに作成されます。VARRAY の各要素には、データベースにロードする前に、4 バイトのクライアント・メモリーが必要です。そのため、1000 個の要素を持つ VARRAY をロードする場合は、VARRAY をデータベースにロードする前に、それぞれの VARRAY インスタンスに、4000 バイト以上のクライアント・メモリーが必要です。多くの場合、SQL\*Loader では、VARRAY の構築やロードに、2～3 倍のメモリー量を必要とします。
- BINDSIZE パラメータを使用して、SQL\*Loader でレコードのロードに割り当てるメモリーの量を指定します。BINDSIZE に指定された値に応じて、SQL\*Loader で、ロード中の各フィールドのサイズを考慮し、1 回のトランザクションでロードできる行数を判断します。行数が多ければトランザクションは少なくなり、パフォーマンスは向上します。

ただし、システムのメモリー量に制限がある場合、パフォーマンスを優先せず、SQL\*Loader で算出した値より低い値を ROWS に指定できます。

- 非常に大きい VARRAY または多数の小さい VARRAY が原因で、ロード中にメモリーが不足する場合があります。この場合は、BINDSIZE または ROWS により小さい値を指定し、再ロードします。

---

## SQL\*Loader ログ・ファイル・リファレンス

SQL\*Loader の実行が開始すると、ログ・ファイルが作成されます。ログ・ファイルには、ロードの詳細な情報が含まれています。

ログ・ファイル・エントリのほとんどは、SQL\*Loader が正常に実行されたことの記録です。ただし、エラーが発生してログ・ファイル・エントリが作成される場合もあります。たとえば、制御ファイルの解析中にエラーが検出されると、そのエラーはログ・ファイルに書き込まれます。

この章の内容は、次のとおりです。

- [ヘッダー情報](#)
- [グローバル情報](#)
- [表情報](#)
- [データ・ファイル情報](#)
- [表ロード情報](#)
- [サマリー統計](#)
- [ダイレクト・パス・ロードおよびマルチスレッドのサマリー統計の追加](#)
- [EXTERNAL\\_TABLE=GENERATE\\_ONLY の使用時に作成されるログ・ファイル](#)

## ヘッダー情報

ヘッダー・セクションには、次のエントリが含まれます。

- 実行日
- ソフトウェアのバージョン番号

次に例を示します。

```
SQL*Loader: Release 9.2.0.1.0 - Production on Wed Feb 27 11:07:28 2002
```

```
(c) Copyright 2002 Oracle Corporation. All rights reserved.
```

## グローバル情報

グローバル情報セクションには、次のエントリが含まれます。

- すべての入出力ファイル名
- コマンドライン引数のエコー
- 継続文字仕様

データが制御ファイルにある場合、データ・ファイルは「\*」として示されます。

次に例を示します。

```
Control File:      LOAD.CTL
Data File:         LOAD.DAT
  Bad File:        LOAD.BAD
  Discard File:    LOAD.DSC
```

```
(Allow all discards)
```

```
Number to load: ALL
Number to skip: 0
Errors allowed: 50
Bind array:      64 rows, maximum of 65536 bytes
Continuation:    1:1 = '*', in current physical record
Path used:       Conventional
```

# 表情報

表情報セクションには、ロードされる各表について次のエントリが含まれます。

- 表名。
- ロード条件（指定されている場合）。ここでは、すべてのレコードがロードされたのか、WHEN 句で指定された条件を満たすレコードのみがロードされたのかが示されます。
- INSERT、APPEND または REPLACE の設定。
- 列情報。次のようなエントリが書き込まれます。
  - 列名。
  - 列の位置、長さ、デリミタまたはデータ型（データ・ファイルに定義されている場合）。列の詳細は、8-4 ページの「[列情報](#)」を参照してください。
  - RECNUM、SEQUENCE、CONSTANT または EXPRESSION（指定されている場合）。
  - DEFAULTIF または NULLIF（指定されている場合）。

次に例を示します。

Table EMP, loaded from every logical record.  
Insert option in effect for this table: REPLACE

Column Name	Position	Len	Term	Encl	Datatype
empno	1:4	4			CHARACTER
ename	6:15	10			CHARACTER
job	17:25	9			CHARACTER
mgr	27:30	4			CHARACTER
sal	32:39	8			CHARACTER
comm	41:48	8			CHARACTER
deptno	50:51	2			CHARACTER

Column empno is NULL if empno = BLANKS  
Column mgr is NULL if mgr = BLANKS  
Column sal is NULL if sal = BLANKS  
Column comm is NULL if comm = BLANKS  
Column deptno is NULL if deptno = BLANKS

## 列情報

この項では、SQL\*Loader ログ・ファイルの表情報セクションで説明した列情報について説明します。

### 位置

次に、位置列で行われる可能性のある事項を示します。

- 位置を指定した場合は、バイト長セマンティクスが使用されているか、または文字長セマンティクスが使用されているかにかかわらず、位置は 1 で始まるバイト単位の値になります。
- 開始位置および終了位置は、コロンで区切られます。
- 開始位置のみを指定した場合は、その位置のみが表示されます。
- 開始位置も終了位置も指定しなかった場合は、最初のフィールドに FIRST が表示され、他のフィールドに NEXT が表示されます。
- 開始位置が他の情報から導出された場合は、DERIVED が表示されます。

### 長さ

長さは、見出しの Len の下にバイト単位で表示されます。この値によってフィールドの最大サイズがわかります。この長さには、埋め込まれた長さフィールドのサイズも含まれます。サイズは、バイト長セマンティクスか文字長セマンティクスによって異なります。たとえば、バイト長セマンティクスの VARCHAR (2,10) の場合は、長さは 2 (長さフィールドのサイズ) +10 (フィールドの最大サイズ) =12 バイトになります。文字長セマンティクスの VARCHAR (2,10) の場合は、長さはデータ・ファイルのキャラクタ・セットで文字の最大サイズを使用してバイト単位で計算されます。

指定された最大長を持ったフィールドの場合は、Len 列にアスタリスク (\*) が 1 つ書き込まれます。

### デリミタ

デリミタは、見出し Term (終了デリミタ) および Encl (囲みデリミタ) の下に表示されます。デリミタがオプションの場合は、デリミタの前に 0 が置かれ、カッコで囲まれて表示されます。

### データ型

データ型は、制御ファイルで指定されたものが表示されます。

SQL\*Loader 制御ファイルに、日時および期間のデータ型をロードする指示句が含まれる場合、ログ・ファイルには見出し Datatype の下に DATE、DATETIME または INTERVAL パラメータが含まれます。適用されると、DATE、DATETIME または INTERVAL パラメータには対応するマスクが続きます。次に例を示します。

Table emp, loaded from every logical record.  
 Insert option in effect for this table: REPLACE

Column Name	Position	Len	Term	Encl	Datatype
-----	-----	---	---	---	-----
coll	NEXT	*			DATETIME HH.MI.SSXXFF AM

## データ・ファイル情報

データ・ファイル情報セクションは、データ・エラーが発生したデータ・ファイルについてのみ生成されます。このセクションには、次のエントリが含まれます。

- SQL\*Loader および Oracle のデータ・レコード・エラー
- 廃棄されたレコード

次に例を示します。

Record 2: Rejected - Error on table EMP.

ORA-00001: 一意制約 <name> に反しています

Record 8: Rejected - Error on table emp, column deptno.

ORA-01722: 数値が無効です。

Record 3: Rejected - Error on table proj, column projno.

ORA-01722: 数値が無効です。

## 表ロード情報

表ロード情報セクションには、ロードされた表ごとに次のエントリが含まれます。

- ロードされた行の数
- ロード対象だったが、データ・エラーのためロードを拒否された行の数
- WHEN 句で指定された条件に一致せず、廃棄された行の数
- 関連するフィールドがすべて NULL だった行の数
- 日付キャッシュの統計（適用される場合）

次に例を示します。

Table EMP:

25000 Rows successfully loaded.

2 Rows not loaded due to data errors.

0 Rows not loaded because all WHEN clauses were failed.

0 Rows not loaded because all fields were null.

```
Date Cache:
Max Size: 2000
Entries: 1000
Hits: 11000
Misses: 0
```

**参照：** 日付キャッシュの最大サイズを調整することによってパフォーマンスを向上させる方法の詳細は、9-21 ページの「[日付キャッシュの値の指定](#)」を参照してください。

## サマリー統計

サマリー統計セクションには、次の情報が表示されます。

- 使用された領域のサイズ。
  - バインド配列のサイズ (BINDSIZE の指定に基づいて実際に使用されたサイズ)
  - その他のオーバーヘッドのサイズ (BINDSIZE にかかわらず常に必要となるサイズ)
- ロードの累積統計。すべてのデータ・ファイルに関して、次のレコードの数が示されます。
  - スキップされたレコード
  - 読み込まれたレコード
  - 拒否レコード
  - 廃棄レコード
- 実行開始および終了時刻。
- 総経過時間。
- 総 CPU タイム。(すべてのファイル I/O を含む。ただし、バックグラウンドでの Oracle の CPU タイムは含みません。)

次に例を示します。

```
Space allocated for bind array: 65336 bytes (64 rows)
Space allocated for memory less bind array: 6470 bytes
```

```
Total logical records skipped: 0
Total logical records read: 7
Total logical records rejected: 0
Total logical records discarded: 0
```

```
Run began on Wed Feb 27 10:46:53 1990
Run ended on Wed Feb 27 10:47:17 1990
```



Elapsed time was: 00:00:15.62  
CPU time was: 00:00:07.76

## Oracle のログ用統計レポート

ログ・ファイル用統計レポートは、ロードのタイプによってその内容が異なります。

- 非パーティション表については、従来型ロードの場合もダイレクト・ロードの場合も、Oracle7 以上では統計レポートに関する変更はありません。
- パーティション表をダイレクト・ロードした場合は、表レベル統計セクションの次に、パーティション別統計セクションが表示されます。
- 単一パーティションをロードした場合は、そのパーティション名が、表レベル統計セクション中に表示されます。

### 単一パーティションのロード情報

単一パーティションのロード時のログ情報は、次のようになります。

- 表の列の説明の中にパーティション名が示されます。
- エラー・メッセージの中にパーティション名が示されます。
- 統計一覧の中にパーティション名が示されます。

### 表ロード時の統計

表のロード時の統計ログは、次のようになります。

- パーティション表をダイレクト・パス・ロードした場合、パーティション別の統計が記録されます。
- 従来型パス・ロードの場合は、パーティション別の統計は記録されません。
- 非パーティション表をロードする場合、統計レポートに関して Oracle7 以上での変更はありません。

非パーティション表については、従来型ロードの場合もダイレクト・ロードの場合も、Oracle7 以上では統計レポートに関する変更はありません。

メディア・リカバリが使用可能ではないにもかかわらず、ロギングを要求した場合、ロード時の情報はログに記録されません。

## ダイレクト・パス・ロードおよびマルチスレッドのサマリー統計の追加

ダイレクト・パス・ロードの場合、ログには次の追加データが含まれます（ログ・ファイルの番号は異なる場合があります）。

```
Column array rows:      20000
Stream buffer bytes:    256000
```

これらの統計の根拠は、9-21 ページの「[列配列の行数およびストリーム・バッファ・サイズの指定](#)」を参照してください。

複数 CPU システム上のダイレクト・パス・ロードでは、オプションでマルチスレッドを使用できます。マルチスレッドが使用可能な場合（デフォルト）は、次追加統計がログに記録されます（ログの番号は異なる場合があります）。

```
Total stream buffers loaded by SQL*Loader main thread:  102
Total stream buffers loaded by SQL*Loader load thread:   200
```

マルチスレッドの詳細は、9-23 ページの「[複数 CPU システムのダイレクト・パス・ロードの最適化](#)」を参照してください。

## EXTERNAL\_TABLE=GENERATE\_ONLY の使用時に作成されるログ・ファイル

外部表の機能を使用すると、制御ファイルに記述されているとおり、ロードに必要なすべての SQL コマンドを SQL\*Loader ログ・ファイルに記述することができます。これを行うには、EXTERNAL\_TABLE パラメータを GENERATE\_ONLY に設定します。実際のロードは、SQL\*Loader を使用せずに、SQL\*Plus でこれらの文を実行して、後で行うことができます。

EXTERNAL\_TABLE=GENERATE\_ONLY の使用時に作成されるログ・ファイルの例を生成するには、事例 1（10-6 ページの「[事例 1: 可変長データのロード](#)」）の次のコマンドを実行します。

```
sqlldr scott/tiger ulcase1 EXTERNAL_TABLE=GENERATE_ONLY
```

次に、結果のログ・ファイルを示します。

```
SQL*Loader: Release 9.2.0.1.0 - Production on Wed Feb 27 11:07:28 2002
```

```
(c) Copyright 2002 Oracle Corporation. All rights reserved.
```

```
Control File:    ulcase1ctl
Data File:       ulcase1ctl
Bad File:        ulcase1.bad
Discard File:    none specified
```

(Allow all discards)

Number to load: ALL  
 Number to skip: 0  
 Errors allowed: 50  
 Continuation: none specified  
 Path used: External Table

Table DEPT, loaded from every logical record.  
 Insert option in effect for this table: INSERT

Column Name	Position	Len	Term	Encl	Datatype
DEPTNO	FIRST	*	,	O(")	CHARACTER
DNAME	NEXT	*	,	O(")	CHARACTER
LOC	NEXT	*	,	O(")	CHARACTER

CREATE DIRECTORY statements needed for files

```
CREATE DIRECTORY SYS_SQLLDR_XT_TMPDIR_00000 AS '/private/adestore/krich/.ade/view_
storage/krich_dev/rdbms/demo'
```

CREATE TABLE statement for external table:

```
CREATE TABLE "SYS_SQLLDR_X_EXT_DEPT"
(
  DEPTNO NUMBER(2),
  DNAME VARCHAR2(14),
  LOC VARCHAR2(13)
)
ORGANIZATION external
(
  TYPE oracle_loader
  DEFAULT DIRECTORY SYS_SQLLDR_XT_TMPDIR_00000
  ACCESS PARAMETERS
  (
    RECORDS DELIMITED BY NEWLINE CHARACTERSET US7ASCII
    BADFILE 'SYS_SQLLDR_XT_TMPDIR_00000': 'ulcase1.bad'
    LOGFILE 'ulcase1.log_xt'
    READSIZE 1048576
    SKIP 20
    FIELDS TERMINATED BY "," OPTIONALLY ENCLOSED BY "'" LDRTRIM
    REJECT ROWS WITH ALL NULL FIELDS
  )
```

```
        DEPTNO CHAR(255)
            TERMINATED BY "," OPTIONALLY ENCLOSED BY '"',
        DNAME CHAR(255)
            TERMINATED BY "," OPTIONALLY ENCLOSED BY '"',
        LOC CHAR(255)
            TERMINATED BY "," OPTIONALLY ENCLOSED BY '"',
    )
)
location
(
    'ulcase1.ctl'
)
)REJECT LIMIT UNLIMITED
```

INSERT statements used to load internal tables:

```
-----
INSERT /*+ append */ INTO DEPT
(
    DEPTNO,
    DNAME,
    LOC
)
SELECT
    DEPTNO,
    DNAME,
    LOC
FROM "SYS_SQLLDR_X_EXT_DEPT"
```

statements to cleanup objects created by previous statements:

```
-----
DROP TABLE "SYS_SQLLDR_X_EXT_DEPT"
DROP DIRECTORY SYS_SQLLDR_XT_TMPDIR_00000
```

Run began on Wed Feb 27 11:07:28 2002

Run ended on Wed Feb 27 11:07:34 2002

Elapsed time was: 00:00:06.13

CPU time was: 00:00:00.20

**参照：**

- 4-7 ページの「[EXTERNAL\\_TABLE](#)」を参照してください。
- 第 III 部「外部表」も参照してください。

---

## 従来型パス・ロードおよびダイレクト・パス・ロード

この章では、SQL\*Loader の従来型パス・ロードとダイレクト・パス・ロードの方法について説明します。この章の内容は、次のとおりです。

- データのロード方法
- 従来型パス・ロード
- ダイレクト・パス・ロード
- ダイレクト・パス・ロードの使用
- ダイレクト・パス・ロードのパフォーマンスの最適化
- 複数 CPU システムのダイレクト・パス・ロードの最適化
- 索引メンテナンスの回避
- ダイレクト・ロード、整合性制約およびトリガー
- パラレル・データ・ロード・モデル
- 一般的なパフォーマンス改善のヒント

ダイレクト・パス・ロードを使用した例は、10-25 ページの「[事例 6: ダイレクト・パス・ロード方式を使用したデータのロード](#)」を参照してください。その他の事例では、従来型パス・ロードを使用しています。

## データのロード方法

SQL\*Loader でデータをロードするには、次の 2 つの方法があります。

- 従来型パス・ロード
- ダイレクト・パス・ロード

従来型パス・ロードでは、Oracle データベースの表に対して (1 つ以上の) SQL INSERT 文が実行されます。ダイレクト・パス・ロードでは、Oracle データ・ブロックをフォーマットし、データ・ブロックを直接データ・ファイルに書き込むため、Oracle データベースのオーバーヘッドが大幅に削減されます。ダイレクト・パス・ロードでは、データベース・リソースに対して他のユーザーとの競合が発生しないため、ディスク速度に近い速度でデータをロードできます。この章では、ダイレクト・パス・ロード固有の問題 (制限、セキュリティ、バックアップなど) について説明します。

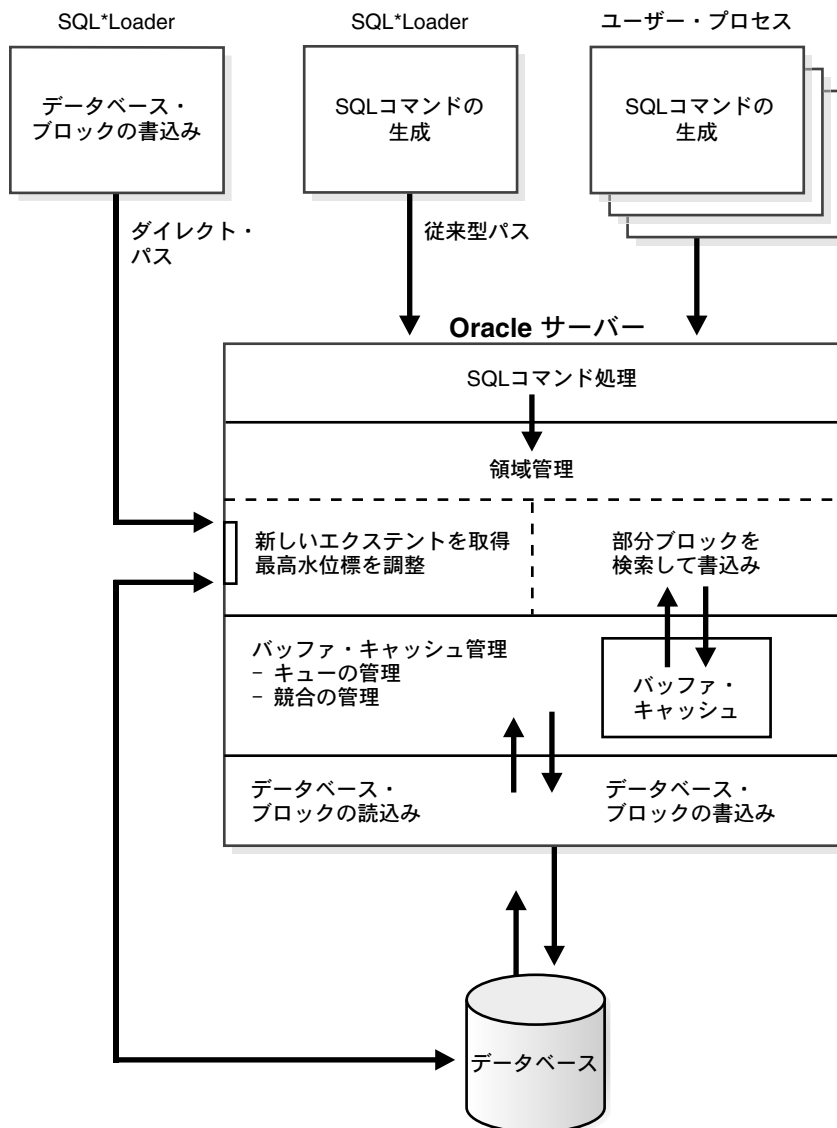
データをロードする表はデータベース中に存在している必要があります。SQL\*Loader では、表は作成されません。すでにデータが含まれているか、または空である既存の表にロードされます。

ロードを実行するには、次の権限が必要です。

- ロードする表についての INSERT 権限。
- ロードする表にすでにデータが存在するために、REPLACE オプションまたは TRUNCATE オプションを使用して古いデータを削除してから新しくデータをロードする場合には、その表についての DELETE 権限。

[図 9-1](#) に、従来型パス・ロードおよびダイレクト・パス・ロードでのデータベースへの書き込み方法を示します。

図 9-1 SQL\*Loader ダイレクト・パスおよび従来型パスでのデータベースの書き込み



## 従来型パス・ロード

従来型パス・ロード（デフォルト）では、SQL INSERT 文とバインド配列バッファを使用して、データをデータベース表にロードします。この方法は、すべての Oracle のツール製品および Applications で使用されます。

SQL\*Loader で従来型パス・ロードを実行する場合、バッファ・リソースに関して他のすべてのプロセスと同等の処理が行われるため、競合が発生します。このため、ロードにかなりの時間がかかります。また、SQL コマンドが生成され、Oracle に渡されてから実行されるため、さらにオーバーヘッドが発生します。

挿入が発生すると、常に、Oracle データベース・サーバーで空き領域のあるブロック（ディスク内に散在して、部分的に書込み可能なブロック）が検索され、そこにデータが書き込まれます。通常のデータベース使用の場合はそれほどでもありませんが、このアクションは大量データのロード速度を大幅に低下させることがあります。

**参照：** 5-24 ページの「[中断された従来型パス・ロード](#)」を参照してください。

## 単一パーティションの従来型パス・ロード

従来型パス・ロードでは、SQL INSERT 文を使用します。ただし、従来型パスで単一パーティションに対してロードする場合は、SQL\*Loader で次のような形式の INSERT 文のパーティション拡張構文を使用します。

```
INSERT INTO TABLE T PARTITION (P) VALUES ...
```

Oracle カーネルの SQL レイヤーでは、挿入される行が指定のパーティションに対応するかどうかを判断します。行が指定のパーティションに対応しない場合、その行は拒否され、そのことを示すエラー・メッセージが SQL\*Loader ログ・ファイルに記録されます。

## 従来型パスを使用する場合

ロードを高速にするには、従来型パス・ロードよりダイレクト・パス・ロードを使用します。ただし、ダイレクト・パス・ロードにはいくつかの制限があるため、従来型パス・ロードを使用する必要がある場合もあります。次のような場合には、従来型パス・ロードを使用します。

- ロードと並行して索引付き表にアクセスする場合、またはロードと並行して索引なしの表に挿入または更新を行う場合。

ダイレクト・パス・ロード（パラレル・ロードは除く）を使用するには、SQL\*Loader に、表への排他的書込み権限と、すべての索引への排他的読込み権限および書込み権限が必要です。

- データをクラスタ表にロードする場合。



ダイレクト・パス・ロードでは、クラスタ表に対するロードはサポートされていません。

- 比較的少数の行を索引付きの大きな表にロードする場合。

ダイレクト・パス・ロードでは、既存の索引を新しい索引キーとマージするために、既存の索引をコピーします。既存の索引が非常に大きく、新しいキーの数が非常に少ない場合は、索引をコピーする時間が、ダイレクト・パス・ロードで節約できる時間を相殺してしまうことがあります。

- 参照整合性制約および列チェック整合性制約のある大きな表に、比較的少数の行をロードする場合。

これらの制約は、ダイレクト・パスでロードした行には適用できないため、ロードが継続している間は使用禁止になります。そして、ロードが完了した時点で表全体に適用されます。表が非常に大きく、新しい行の数が少ない場合は、この処理にかかる時間が節約した時間より多くなる可能性があります。

- レコードのロード時に、次のような状況でレコードが拒否されることを確認する場合。
  - － レコードの挿入で Oracle エラーが発生した場合。
  - － レコードが間違っていてフォーマットされたため、SQL\*Loader でフィールドの境界を見つけられない場合。
  - － レコードが制約に違反した場合、または一意の索引を非一意にしようとした場合。

## ダイレクト・パス・ロード

バインド配列バッファに書き込むかわりに、SQL INSERT 文を使用して、バインド配列を Oracle データベース・サーバーに渡します。ダイレクト・パス・ロードは、ダイレクト・パス API を使用して、ロードされるデータをサーバーのロード・エンジンに渡します。ロード・エンジンは、渡されたデータから列配列構造体を作成します。

ダイレクト・パス・ロード・エンジンは、列配列構造体を使用して Oracle データ・ブロックをフォーマットし、索引キーを作成します。新しくフォーマットされたデータベース・ブロックが直接データベースに書き込まれます（ホスト・プラットフォームが非同期 I/O をサポートしている場合、非同期書き込みを使用して 1 つの I/O 要求に対して複数のブロックを書き込むことができます）。

内部的には、フォーマットされたブロック用に複数のバッファが使用されます。ホスト・プラットフォームで非同期 I/O が可能な場合は、あるバッファに書き込んでいる間に 1 つ以上のバッファへの書き込みが行われます。この場合、I/O を伴う処理がオーバーラップするため、ロード・パフォーマンスが向上します。

**参照：** 5-24 ページの「[中断されたダイレクト・パス・ロード](#)」を参照してください。

## ダイレクト・パス・ロード時のデータ変換

ダイレクト・パス・ロード時には、サーバー側ではなくクライアント側でデータ変換が発生します。初期化パラメータ・ファイルの NLS パラメータ（サーバー側の言語ハンドル）は使用されません。この動作を変更するには、SQL\*Loader の制御ファイルに書式マスクを指定する（初期化パラメータ・ファイルに NLS パラメータを設定することと同じ）か、または適切な環境変数を設定します。たとえば、フィールドに日付書式を指定するには、[例 9-1](#) に示すとおり SQL\*Loader の制御ファイルに日付書式を設定するか、または[例 9-2](#) に示すとおり環境変数 NLS\_DATE\_FORMAT を設定します。

### 例 9-1 SQL\*Loader の制御ファイルに対する日付書式の設定

```
LOAD DATA
INFILE 'data.dat'
INSERT INTO TABLE emp
FIELDS TERMINATED BY "|"
(
  EMPNO NUMBER(4) NOT NULL,
  ENAME CHAR(10),
  JOB CHAR(9),
  MGR NUMBER(4),
  HIREDATE DATE 'YYYYMMDD',
  SAL NUMBER(7,2),
  COMM NUMBER(7,2),
  DEPTNO NUMBER(2)
)
```

### 例 9-2 環境変数 NLS\_DATE\_FORMAT の設定

UNIX の Bourne または Korn シェルの場合：

```
% NLS_DATE_FORMAT='YYYYMMDD'
% export NLS_DATE_FORMAT
```

UNIX の csh の場合：

```
%setenv NLS_DATE_FORMAT='YYYYMMDD'
```

## パーティション表またはサブパーティション表のダイレクト・パス・ロード

パーティション表またはサブパーティション表へロードする場合は、SQL\*Loader によって、行がパーティション化され、索引（索引もパーティション化できます）がメンテナンスされます。パーティション表またはサブパーティション表のダイレクト・パス・ロードは、パーティションまたはサブパーティションの多い表の場合、非常に多くのリソースを使用することに注意してください。

---

**注意：** 複数のパーティションに対してダイレクト・パス・ロードを実行する場合に領域エラーが発生すると、ロードは直前のコミット・ポイントにロールバックされます。コミット・ポイントが存在しない場合、ロード全体がロードバックされます。これによって、領域エラー後に検出されたデータが異なるパーティションに書き込まれることがなくなります。

ROWS パラメータを使用して、コミット・ポイントの頻度を指定できます。ROWS パラメータが指定されていない場合は、ロード全体がロールバックされます。

---

## 単一パーティションまたはサブパーティションのダイレクト・パス・ロード

パーティション表の単一パーティションまたはサブパーティションをロードするとき、SQL\*Loader によって、行がパーティション化され、SQL\*Loader 制御ファイルに指定されたパーティションまたはサブパーティションにマップできない行はすべて拒否されます。ロードされるデータのパーティションまたはサブパーティションに対応するローカル索引パーティションは、SQL\*Loader によってメンテナンスされます。単一パーティションまたはサブパーティションのダイレクト・パス・ロードでは、グローバル索引はメンテナンスされません。ただし、ダイレクト・パスで単一パーティションに対してロードする場合、SQL\*Loader では次のような形式の LOAD 文のパーティション拡張構文を使用します。

```
LOAD INTO TABLE T PARTITION (P) VALUES ...
```

```
LOAD INTO TABLE T SUBPARTITION (P) VALUES ...
```

パーティション表またはサブパーティション表の 1 つのパーティションをロードしている間も、その表の他のパーティションに対しては DML 操作やダイレクト・パス・ロードを行うことができます。

ダイレクト・パス・ロードでは、データベース処理は最小限に抑えられますが、ロードの開始時と終了時に、それぞれロードの初期化と終了処理のために Oracle データベース・サーバーに対するコールが数回実行されます。また、ロードの初期化中に必要な DML ロックがかけられ、ロード終了時に解除されます。ロード中には、索引キーが作成されてソートに使用されたり、必要に応じて新しいエクステントを取得するために領域管理ルーチンを使用することによって、データ・セーブポイントの上限（最高水位標）を調整するなどの動作も発生します。上限の調整については、9-13 ページの「[データ・セーブを使用したデータ損失の防止](#)」参照してください。

## ダイレクト・パス・ロードのメリット

ダイレクト・パス・ロードは、従来型パス・ロードよりも処理が高速です。その理由は次のとおりです。

- 一部使用中の部分ブロックを使用しないため、空きブロックを探すための読み込み処理が不要で、書き込みも少なくなります。
- SQL\*Loader は SQL INSERT 文を実行しないため、Oracle データベースの処理負荷が減ります。
- 表と索引をロード開始時にロックするように Oracle に要求し、ロード完了時にロック解除を要求します。これに対し、従来型パス・ロードでは、行配列ごとに Oracle を 1 回コールして、SQL INSERT 文を処理します。
- マルチ・ブロックの非同期 I/O を使用してデータベース・ファイルに書き込みます。
- ダイレクト・パス・ロードを使用するプロセスは、Oracle のバッファ・キャッシュを使用するのではなく、そのプロセス独自の書き込み I/O を実行します。これによって、他の Oracle ユーザーとの競合を最少にします。
- SORTED INDEXES オプションを指定すると、使用システムまたはインストール環境に固有の高性能ソート・ルーチンを使用して、データを事前にソートしておくことができます。
- ロードする表が空の場合、事前ソート・オプションを使用すると索引作成のソート段階とマージ段階を省略できます。索引は、データの挿入時に作成されます。
- インスタンス障害からのリストアに、REDO ログ・ファイル・エントリは必要ありません。したがって、次の場合は、ロード時のログを記録する時間は不要です。
  - Oracle の動作モードが NOARCHIVELOG の場合
  - UNRECOVERABLE パラメータの値が Y の場合
  - ロードするオブジェクトの NOLOG 属性が設定されている場合

詳細は、9-15 ページの「[インスタンス・リカバリおよびダイレクト・パス・ロード](#)」を参照してください。

## ダイレクト・パス・ロード使用上の制限

ダイレクト・パス・ロードを使用するには、次の条件を満たしている必要があります。

- 表がクラスタ化されていないこと。
- ロードする表に未処理のアクティブ・トランザクションがないこと。

この条件が満たされているかどうかを確認するには、MONITOR TABLE という OEM コマンドを使用して、ロードする表のオブジェクト ID を検索します。次に、MONITOR LOCK コマンドを使用して、表にロックがかけられているかどうか調べます。

- Oracle9i より前のバージョンの Oracle データベース・サーバーでは、クライアントとサーバーが同じバージョンである場合のみ、SQL\*Loader のダイレクト・パス・ロードを実行できます。また、Oracle9i のデータのダイレクト・パス・ロードは、以前のバージョンのデータベースに対しては実行できません。たとえば、ダイレクト・パス・ロードを使用して、Oracle9i リリース 1 (9.0.1) のデータベースから Oracle8i リリース 8.1.7 のデータベースに、データはロードできません。

ただし、Oracle9i 以上のバージョンでは、クライアントおよびサーバーの両方が Oracle9i 以上であるかぎり、異なるバージョン間で SQL\*Loader のダイレクト・パス・ロードを実行できます。たとえば、Oracle9i リリース 1 (9.0.1) のデータベースから Oracle9i リリース 2 (9.2) のデータベースへのダイレクト・パス・ロードを実行できます。

次の機能は、ダイレクト・パス・ロードでは使用できません。

- VARRAY のロード
- 親表を子表とともにロード
- BFILE 列のロード

## 単一パーティションのダイレクト・パス・ロードでの制限

前述の制限に加え、単一のパーティションをロードするときには次の制限があります。

- パーティションのある表に、グローバル索引が定義されていないこと。
- パーティションのある表に対して、参照制約および CHECK 制約が使用禁止 (Disable) であること。
- トリガーが使用禁止 (Disable) であること。

## ダイレクト・パスを使用する場合

前述の制限のいずれにも該当しない場合は、次のようなときにダイレクト・パス・ロードを使用してください。

- 短時間で大量のデータをロードする必要がある場合。ダイレクト・パス・ロードによって、大量のデータを高速ロードし、索引付けできます。表が空であるかどうかにかかわらず、データをロードできます。
- 最大のパフォーマンスを得るため、データをパラレルでロードする場合。詳細は、9-29 ページの「[パラレル・データ・ロード・モデル](#)」を参照してください。

### 整合性制約

ダイレクト・パス・ロード時には、すべての整合性制約が適用されます。ただし、すべての制約が同時に適用されるとはかぎりません。ロード中には NOT NULL 制約が施行されます。これらの制約に従っていないレコードは拒否されます。

ロード中およびロード後に、一意制約が施行されます。一意制約に違反するレコードは拒否されます（制約違反が検出されると、そのレコードはメモリー内で使用不可になります）。

他の行または表に依存する整合性制約（参照制約など）は、ダイレクト・パス・ロード実行前に使用禁止になります。そのため、ロード後に再び使用可能にする必要があります。REENABLE を指定すると、これらの制約はロード終了時に自動的に使用可能に戻されます。制約が再び使用可能になった時点で、表全体（すべての行）に対してチェックが行われます。このチェックでエラーが見つかった行は、指定されたエラー・ログに書き込まれます。詳細は、9-24 ページの「[ダイレクト・ロード、整合性制約およびトリガー](#)」を参照してください。

### ダイレクト・パスのフィールド・デフォルト

ダイレクト・パス・ロードを使用する場合、データベースに定義されているデフォルトの列指定は使用できません。デフォルト値を設定するフィールドに対しては、DEFAULTIF 句を使用して指定する必要があります。DEFAULTIF 句が指定されておらず、フィールドが NULL である場合は、NULL 値がデータベースに挿入されます。

### シノニムへのロード

ダイレクト・パス・ロードでは、表のシノニムにデータをロードできます。ただし、そのためには、そのシノニムが表を直接指している必要があります。シノニムがビューのシノニムであるか、他のシノニム用のシノニムである場合は、データはロードできません。

## ダイレクト・パス・ロードの使用

この項では、SQL\*Loader のダイレクト・パス・ロードの使用方法を説明します。

### ダイレクト・パス・ロードのセットアップ

ダイレクト・パス・ロード用にデータベースを準備するには、セットアップ・スクリプトの catldr.sql を実行し、必要なビューを作成します。このスクリプトは、ダイレクト・ロードを行う予定のデータベースそれぞれに対して 1 回のみ実行します。データベースのインストール時に、ダイレクト・ロードを実行することがわかっている場合は、データベースのインストール中にこのスクリプトを実行することもできます。

## ダイレクト・パス・ロードの指定

SQL\*Loader をダイレクト・パス・ロード・モードで起動するには、次の形式で、コマンドラインまたはパラメータ・ファイル（使用している場合）の DIRECT パラメータに true を設定します。

DIRECT=true

### 参照：

- 10-25 ページの「[事例 6: ダイレクト・パス・ロード方式を使用したデータのロード](#)」を参照してください。
- ダイレクト・パス・ロードのパフォーマンスの最適化に使用可能なパラメータの詳細は、9-17 ページの「[ダイレクト・パス・ロードのパフォーマンスの最適化](#)」を参照してください。
- 複数 CPU システムまたは分散システムでダイレクト・パス・ロードを使用する場合は、9-23 ページの「[複数 CPU システムのダイレクト・パス・ロードの最適化](#)」を参照してください。

## 索引の作成

一時記憶域を使用すると、ダイレクト・パス・ロードのパフォーマンスが向上します。各ブロックがフォーマットされた後、新しい索引キーがソート（一時）セグメントに挿入されます。ロードが終了すると、古い索引と新しいキーがマージされ、新しい索引が作成されます。古い索引、ソート（一時）セグメント、新しい索引セグメントでは、すべてのマージが完了するまで記憶域が必要です。最後に、古い索引と一時セグメントが削除されます。

従来型パス・ロードでは、行が挿入されるたびに索引が更新されます。この方法では一時記憶域は不要ですが、処理に時間がかかります。

## パフォーマンスの向上

メモリーに制限があるシステムでパフォーマンスを向上するには、SINGLEROW パラメータを使用します。詳細は、5-38 ページの「[SINGLEROW オプション](#)」を参照してください。

---

**注意：** ダイレクト・ロード時にデータの事前ソートを指定してあり、既存の索引が空である場合、一時セグメントは不要で、マージも行われません。この場合は、索引にキーが直接付加されます。詳細は、9-17 ページの「[ダイレクト・パス・ロードのパフォーマンスの最適化](#)」を参照してください。

---

複数の索引が作成されると、古い索引の他に、各索引に対応する一時セグメントが同時に存在するようになります。次に、新しいキーは一度に 1 索引ずつ古い索引とマージされます。新しい各索引が作成されると、古い索引とそれに対応する一時セグメントは削除されます。

**参照：** 索引のサイズを計算する方法および記憶域パラメータを設定する方法の詳細は、『Oracle9i データベース管理者ガイド』を参照してください。

## 一時セグメント記憶域要件

新しい索引キーの格納に必要な一時セグメント領域の大きさ（バイト単位）を計算するには、次の式を使用します。

### 1.3 \* key\_storage

この式では、キー記憶域が次のように定義されます。

```
key_storage = (number_of_rows) *  
              ( 10 + sum_of_column_sizes + number_of_columns )
```

この式における列（column）とは、索引の列を意味します。ここでは、1 列につき 1 バイトを使用しています。さらに、ROWID やその他のオーバーヘッドとして 1 行につき 10 バイトを計算に入れています。

定数 1.3 は、ソートに必要な追加領域の平均的な大きさを反映しています。この値は、データの順序がきわめてランダムな場合に有効です。データが逆の順序に並んでいると、ソートには 2 倍のキー記憶域が必要となるため、そのときは定数値を 2.0 とします。ただし、これは最悪の場合です。

データが完全にソートされている場合は、索引エントリを格納できる領域のみが必要なため、そのときの定数の値は 1.0 に下がります。詳細は、9-17 ページの「[高速索引付けのためのデータの事前ソート](#)」を参照してください。

## 使用禁止状態（Index Unusable）のままの索引

ロードされているデータ・セグメントが、その索引の索引セグメントより新しいものになると、SQL\*Loader によって索引が索引使用禁止状態になります。

SQL 文が索引使用禁止状態の索引を参照しようとすると、エラーが発生します。ダイレクト・パス・ロードの実行時に、次のような状況が発生すると、索引またはパーティション索引のパーティションは索引使用禁止状態になります。

- SQL\*Loader で索引のための領域が少なくなり、索引が更新できない場合。
- データが SORTED INDEXES 句で指定した順序になっていない場合。
- インスタンス障害が発生したか、または索引作成中に Oracle シャドウ・プロセスが失敗した場合。
- 一意の索引内に重複キーがある場合。
- データ・セーブポイントを使用中、データ・セーブポイント発生後にロードが正常に実行されないか、またはキーボードからの中断によって終了された場合。



ある索引が索引使用禁止状態かどうかを調べるには、次に示す簡単な問合せを実行します。

```
SELECT INDEX_NAME, STATUS
FROM USER_INDEXES
WHERE TABLE_NAME = 'tablename';
```

表の所有者でない場合は、USER\_INDEXES のかわりに、ALL\_INDEXES または DBA\_INDEXES を検索してください。

ある索引パーティションが使用禁止状態かどうかを調べるには、次に示す問合せを実行します。

```
SELECT INDEX_NAME,
PARTITION_NAME,
STATUS FROM USER_IND_PARTITIONS
WHERE STATUS != 'VALID';
```

表の所有者でない場合は、USER\_IND\_PARTITIONS のかわりに、ALL\_IND\_PARTITIONS および DBA\_IND\_PARTITIONS を検索します。

## データ・セーブを使用したデータ損失の防止

データ・セーブを使用して、インスタンス障害によるデータ損失を回避できます。前回のセーブポイント実行前にロードされたデータはすべて、インスタンス障害に対して保護されます。インスタンス障害が発生した後でロードを続行するには、障害前に処理された入力ファイルの行数を調べ、SKIP パラメータを使用して処理済の行をスキップします。

表に索引がある場合には、まず索引を削除してからロードを続行します。ロードの終了後、索引を再作成してください。メディア障害およびインスタンス・リカバリの詳細は、9-14 ページの「[ダイレクト・パス・ロード時のデータ・リカバリ](#)」を参照してください。

---

---

**注意：** SQL\*Loader ではデータのロードが完了するまで索引が作成されないため、索引はデータ・セーブでは保護されません（事前ソートされたデータを空の表にロードする場合にかぎり、ロード中に索引が作成されますが、その場合も索引は保護されません）。

---

---

## ROWS パラメータの使用

ROWS パラメータには、ダイレクト・パス・ロードでデータ・セーブを行う間隔を設定します。ROWS に指定する値は、データベースへの挿入を保存する前に、SQL\*Loader によって入力ファイルから読み込まれる行数です。

データ・セーブに対して指定する行数は、概数です。ダイレクト・ロードでは、常に Oracle データベース・ブロックと同じ形式のデータ・バッファをデータの処理単位としています。したがって、データ・セーブされるデータ行の実際数は、データベース・ブロックにおける行数の倍数に切り上げられます。

SQL\*Loader では、必ずデータベース・ブロックの充填に必要な行数が読み込まれます。廃棄されるか、受け付けを拒否されたレコードは削除され、残されたレコードがデータベースに挿入されます。セーブする前に挿入される実際の行数は、指定された行数をデータベース・ブロックの行数の倍数に切り上げた値から、廃棄および拒否されたレコード数を引いた値となります。

データ・セーブは負荷の高い操作です。データ・セーブの間隔が 15 分以上になるように、ROWS を十分高い値に設定してください。これによって、長時間のダイレクト・パス・ロードの実行中にインスタンス障害が発生したときに、損失する作業量の上限（最高水位標）を指定できます。ROWS に小さい数値を設定すると、パフォーマンスが低下します。

### データ・セーブとコミット

従来型ロードでは、ROWS はコミットの前に読み込む行数を意味します。ダイレクト・ロードにおけるデータ・セーブは、従来型ロードにおけるコミットと同様ですが、異なる部分もあります。

類似点は次のとおりです。

- データ・セーブを行うと他のユーザーもその行を参照できます。
- データ・セーブ後は、行をロールバックできません。

一方、従来型との主な相違点は、ダイレクト・パス・ロードのデータ・セーブではロードが完了するまで索引が使用できない（索引使用禁止状態）ということです。

## ダイレクト・パス・ロード時のデータ・リカバリ

ダイレクト・パス・ロード方法を指定すると、SQL\*Loader のデータ・リカバリ機能が完全にサポートされます。リカバリには大きく分けて 2 種類あります。

- メディア・リカバリは、損失したデータベース・ファイルをリカバリします。データベース・ファイルを損失した場合に、それをリカバリできるようにするには、ARCHIVELOG モードで実行する必要があります。
- インスタンス・リカバリは、障害が発生する前にメモリー内でデータが変更されたが、ディスクに書き込まれる前に障害のため失われてしまったというシステム障害をリカバリします。Oracle データベース・サーバーでは、REDO ログ・ファイルがアーカイブされていない場合も、インスタンス障害をリカバリできます。

**参照：** リカバリの詳細は、『Oracle9i データベース管理者ガイド』を参照してください。

## メディア・リカバリおよびダイレクト・パス・ロード

REDO ログ・ファイル・アーカイブ機能が使用可能になっている（ARCHIVELOG モードで実行している）場合、ダイレクト・パスでロードしたデータは、SQL\*Loader によってログに記録されます。それによって、メディア・リカバリが可能になります。REDO ログ・ファイル・アーカイブの機能が使用可能になっていない（NOARCHIVELOG モードで実行している）場合、メディア・リカバリはできません。

ロード中に失われたデータベース・ファイルをリカバリするには、従来型パスでロードしたデータをリカバリするときと同じ方法を使用してください。

1. 影響を受けたデータベース・ファイルの最新のバックアップをリストアします。
2. RECOVER コマンドを使用して、表領域をリカバリします。

**参照：** RECOVER コマンドの詳細は、『Oracle9i ユーザー管理バックアップおよびリカバリ・ガイド』を参照してください。

## インスタンス・リカバリおよびダイレクト・パス・ロード

データベース・ファイルは、SQL\*Loader によって直接書き込まれます。そのため、インスタンスを再起動すると、最後にデータをセーブした時点までに挿入したすべての行が、自動的にデータベース・ファイルに存在します。変更が REDO ログ・ファイルに記録されていなくても、インスタンス・リカバリは可能です。

インスタンス障害が発生すると、作成中の索引は索引使用禁止状態のままになります。使用禁止状態の索引は、表またはパーティションを使用する前に再構築する必要があります。索引が索引使用禁止状態のままであるかどうかを調べる方法については、9-12 ページの「[使用禁止状態 \(Index Unusable\) のままの索引](#)」を参照してください。

## LONG 型データ・フィールドのロード

SQL\*Loader の最大バッファ・サイズよりも長いデータをダイレクト・パスでロードするには、LOB を使用します。大きい値のストリーム・サイズを使用してこれを行うと、パフォーマンスを向上できます。

**参照：** 次の項を参照してください。

- 7-18 ページの「[LOB のロード](#)」
- 9-21 ページの「[列配列の行数およびストリーム・バッファ・サイズの指定](#)」

次の項で説明するように、PIECED パラメータを使用すると、最大バッファ・サイズより長いデータもロードできますが、LOB を使用することをお勧めします。

## PIECED としてのデータのロード

データが論理レコードの最終列である場合、PIECED パラメータを使用すると、データをセクションごとにロードできます。

列を PIECED と宣言することによって、LONG フィールドを複数の物理レコード（ピース）に分割することが、ダイレクト・パス・ローダーに通知されます。この場合、SQL\*Loader では、LONG フィールドの各ピースが物理レコード内で検索された順序で処理されます。レコードが処理される前に、フィールドのすべてのピースが読み込まれます。SQL\*Loader では、格納前の LONG フィールドは具体化されません。ただし、レコードが処理される前に、フィールドのすべての部分が読み込まれます。

列を PIECED と宣言する場合、次の制約が適用されます。

- このオプションはダイレクト・パスでのみ有効です。
- 1 つの表につき 1 フィールドのみを PIECED にできます。
- PIECED フィールドは論理レコードの最終フィールドである必要があります。
- WHEN 句、NULLIF 句または DEFAULTIF 句では、PIECED フィールドを使用できません。
- 論理レコード内の PIECED フィールドの領域は、他のフィールドの領域と重複してはいけません。
- PIECED に対応するデータベースの列を索引に含めることはできません。
- 拒否されたレコードに PIECED フィールドが含まれている場合は、不良ファイルからそのレコードをロードできません。

たとえば、1 つの PIECED フィールドが 3 つのレコードにまたがっているとします。SQL\*Loader では、最初のレコードから PIECED フィールドの第一分割がロードされ、次に同じバッファを使用して 2 番目のレコードから第二分割がロードされます。その後、同じバッファを使用して同様に 3 番目のレコードがロードされます。ここでエラーが検出されると、最初の 2 つのレコードはすでにバッファには存在しないため、3 番目のレコードのみが不良ファイルに書き込まれます。その結果、不良ファイルにあるレコードが無効となります。

## ダイレクト・パス・ロードのパフォーマンスの最適化

ダイレクト・パス・ロードでは、使用する時間と一時記憶域を制御できます。

時間を最小化するには、次の方法があります。

- 記憶域の事前割当て
- データの事前ソート
- データ・セーブの回数の削減
- REDO ログの最小限の使用
- 配列行の列数およびストリーム・バッファのサイズを指定
- 日付キャッシュの値の指定

領域を最小化するには、次の方法があります。

- ロード前のデータのソート時に、最も多くの一時記憶域を必要とする索引でデータをソートします。
- ロード中の索引メンテナンスを回避します。

### 高速ロードのための記憶域の事前割当て

SQL\*Loader では、必要に応じて自動的に表にエクステン트가追加されますが、これには時間がかかります。新しい表へ高速にロードするには、表の作成に必要なエクステン트를事前に割り当ててください。

表に必要な領域を計算するには、『Oracle9i データベース管理者ガイド』のデータベース・ファイルの管理の説明を参照してください。必要な領域を割り当てるには、SQL の CREATE TABLE 文で INITIAL または MINEXTENTS 句を使用します。

別の方法として、エクステン트의割当て回数が減るようにエクステン트의サイズを十分に大きくする方法もあります。

### 高速索引付けのためのデータの事前ソート

索引付き列を基準にしてデータを事前ソートすると、ダイレクト・パス・ロードのパフォーマンスを改善できます。事前ソートを行うと、ロード時の一時記憶要件を最小限に抑えることができます。また、事前ソートでは、ご使用のオペレーティング・システムまたはアプリケーション用に最適化された高性能ソート・ルーチンを利用できます。

データが事前ソートされていて既存の索引が空でない場合は、事前ソートによって、新しいキーに必要な一時セグメント領域の大きさを最小にできます。ソート・ルーチンは、新しい各キーをキー・リストに追加します。

ソート用の追加領域は必要なく、キーのための領域のみが必要となります。必要な記憶域の大きさを計算するには、ソート係数として 1.3 ではなく 1.0 を使用してください。必要な記

憶域要件の見積りについては、9-12 ページの「[一時セグメント記憶域要件](#)」を参照してください。

事前ソートを指定していて既存の索引が空である場合は、最大効率が実現します。新しいキーが索引に挿入されるのみです。一時セグメントと新しい索引が古い空の索引と同時に存在するのではなく、新しい索引のみが存在します。したがって、一時記憶域は不要であり、時間も短縮できます。

**SORTED INDEXES 句**

データを事前ソートしている索引を指定します。この句は、ダイレクト・パス・ロードでのみ使用できます。10-25 ページの「[事例 6: ダイレクト・パス・ロード方式を使用したデータのロード](#)」を参照してください。

一般に、SORTED INDEXES 句では 1 つの索引のみを指定します。通常、これは、ある索引でソートされたデータは、別の索引にとって正しい順序とは限らないためです。ただし、複数の索引のデータの順序が同じである場合は、索引すべてを同時に指定できます。

SORTED INDEXES 句で指定した索引はすべて、ダイレクト・パス・ロードを開始する前に作成する必要があります。

**未ソートのデータ**

SORTED INDEXES 句で索引を指定しても、データがその索引でソートされていない場合は、ロード終了時に索引は索引使用禁止状態のままになります。データは存在していますが、索引を使用しようとするとエラーになります。索引使用禁止状態の索引がある場合は、ロード後に再構築してください。

**複数列索引**

SORTED INDEXES 句で複数列の索引を指定する場合は、まず索引の最初の列で順序付けが行われ、次に 2 番目の列で順序付けが行われるように、データをソートしてください。

たとえば、索引の最初の列に都市名があり、2 番目の列に名前の名字がある場合、次のリストのように都市別順で、同じ都市の中では名字順に並ぶようにデータをソートします。

Albuquerque	Adams
Albuquerque	Hartstein
Albuquerque	Klein
...	...
Boston	Andrews
Boston	Bobrowski
Boston	Heigham
...	...

## 最適ソート順序の選択方法

ダイレクト・パス・ロードのパフォーマンスを最大限に引き出すには、最も大きな一時セグメント領域を必要とする索引に基づいて、データを事前ソートしてください。たとえば、主キーが1つの数値列で、2次キーが3つのテキスト列で構成される場合、2次キーで事前ソートすることによって、ソート時間と記憶要件の両方を最小にできます。

最も大きな記憶域を必要とする索引がどれであるかを知るには、次の手順に従ってください。

1. 各索引について、その索引のすべての列の幅を加算します。
2. 単表のロードの場合は、最大幅を持った索引を選択します。
3. 複数表へのロードの場合は、各表について最大幅を持つ索引がどれかを調べます。各表にロードされる行数が同じ場合は、最大幅を持つ索引を選択します。通常は、各表にロードされる行数は同じです。
4. 複数表へのロードにおいて、索引付きの表にロードされる行数が表によって異なる場合は、手順3で確認した各索引の幅と、その索引にロードされる行数を掛け合せます。結果が最も大きい値の索引を選択します。

## データ・セーブの回数の削減

ROWS 値が小さいことが原因でデータ・セーブが頻繁に発生する場合、ダイレクト・パス・ロードのパフォーマンスは低下します。ダイレクト・パス・ロードは従来型ロードより何倍も高速なため、ダイレクト・ロードの場合には、ROWS の値は従来型ロードの場合よりかなり大きくする必要があります。

データ・セーブ時には、SQL\*Loader のすべてのバッファへの書き込みが正常に終了するまで、ロードは停止します。ROWS の値は、安全性を確保できる範囲で、できるだけ大きくしてください。数千行をロードしてみて、1行当たりの平均ロード時間を計ってみることをお勧めします。その値から、ROWS に設定する値が求められます。

たとえば、1分当たり 20,000 行がロードされるとします。この場合、処理途中に実行するセーブの間隔を 10 分以内にするには、ROWS を 200,000 (20,000 行 / 分 × 10 分間) に設定してください。

## REDO ログの最小限の使用

ダイレクト・ロードを大幅に高速化する 1 つの方法は、REDO ログの使用を最小限に抑えることです。それには 3 通りの方法があります。アーカイブを使用禁止にする方法、ロードを UNRECOVERABLE に指定する方法、ロードされるオブジェクトに NOLOG 属性を設定する方法です。この項では、すべての方法を説明します。

## アーカイブの使用禁止

アーカイブが使用禁止の場合、ダイレクト・パス・ロードでは全体イメージの REDO ログは生成されません。ARCHIVELOG および NOARCHIVELOG パラメータを使用して、アーカイブ・モードを設定します。アーカイブの詳細は、『Oracle9i データベース管理者ガイド』を参照してください。

## UNRECOVERABLE パラメータの指定

時間および REDO ログ・ファイルの領域を節約するには、データのロード時に UNRECOVERABLE パラメータを使用します。UNRECOVERABLE を指定してロードすると、ロードされたデータは REDO ログ・ファイルに記録されません。かわりに、操作を無効にするために必要な REDO ログ（無効 REDO ログ）が生成されます。

UNRECOVERABLE パラメータは、ロード・セッション中にロードされたすべてのオブジェクト（データ・セグメントおよび索引セグメントの両方）に適用されます。このため、ロードされた表についてはメディア・リカバリはできません。ただし、他のユーザーが行ったデータベース変更のログは、引き続き記録されます。

---

---

**注意：** データ・ロードは記録されないため、必要な場合はロード後にデータのバックアップを取ってください。

---

---

UNRECOVERABLE パラメータを指定してロードしたデータについてメディア・リカバリが必要になった場合、ロードしたデータ・ブロックには、論理的に破損したというマークが付けられます。

データをリカバリするには、データを削除して再作成します。データがリカバリ不能にならないように、データのロード後、すぐにバックアップを取ってください。

デフォルトでは、ダイレクト・パス・ロードは RECOVERABLE です。

## NOLOG 属性の設定

データまたは索引のセグメントに NOLOG 属性が指定されていると、そのセグメントに対する全体イメージの REDO ログは使用できません（無効 REDO ログが生成されます）。NOLOG 属性を使用すると、ログが記録されないオブジェクトに対してより優れた制御が可能です。



## 列配列の行数およびストリーム・バッファ・サイズの指定

列配列の行数によって、ストリーム・バッファが作成される前にロードされた行数を判断します。STREAMSIZE パラメータで、クライアントからサーバーへ送ったデータのストリーム・サイズ（バイト単位）を指定します。

列配列の行数の値を指定するには、COLUMNARRAYROWS パラメータを使用します。

ダイレクト・パス・ストリーム・バッファのサイズを指定するには、STREAMSIZE パラメータを使用します。

これらのパラメータの最適な値は、使用しているシステム、入力データ型および Oracle の列データ型によって異なります。独自の構成用に最適な値を使用することで、SQL\*Loader のログ・ファイルでの経過時間が少なくなります。

これらのパラメータのデフォルト値のリストは、4-2 ページの「[SQL\\*Loader の起動](#)」で説明したとおり、パラメータを指定しないで SQL\*Loader を起動すると参照できます。

---

**注意：** ページングが過剰に発生すると、パフォーマンスが大幅に低下するため、ページング・アクティビティのプロセスを監視する必要があります。過剰なページングを回避するには、READSIZE、STREAMSIZE および COLUMNARRAYROWS の値を小さくする必要があります。

---

複数 CPU システムでダイレクト・パス・ロードを実行する場合、列配列の行数およびストリーム・バッファのサイズを指定すると、特に有効です。詳細は、9-23 ページの「[複数 CPU システムのダイレクト・パス・ロードの最適化](#)」を参照してください。

## 日付キャッシュの値の指定

同じ日付値またはタイムスタンプ値のロードが何度も行われるダイレクト・パス・ロードを実行する場合、総ロード時間の大部分が日付およびタイムスタンプのデータの変換に使用される可能性があります。特に、複数の日付列がロードされる場合にこのような状況が発生します。この場合、SQL\*Loader の日付キャッシュを使用することによってパフォーマンスを向上できます。

日付キャッシュを使用すると、入力データ内に多数の重複する日付値が存在する場合、日付変換が実行される回数が減ります。この機能を使用すると、ロード中に予測される一意の日付の数を指定できます。

日付キャッシュは、デフォルトで使用可能です。日付キャッシュ機能を使用禁止にするには、0（ゼロ）に設定します。

デフォルトの日付キャッシュ・サイズは 1000 要素です。デフォルトのサイズを使用し、1000 を超える一意の入力値がロードされると、日付キャッシュはこの表に対して自動的に使用禁止となります。これによって、過剰および不要なルックアップ時間によって、パフォーマンスが低下する可能性がなくなります。ただし、デフォルトを使用するかわりに 0（ゼロ）以外の値を日付キャッシュに指定し、キャッシュ量がこの値を超えた場合、日付キャッシュ

は使用禁止になりません。最大値を超えた入力データは、適切な変換ルーチンによって明示的に変換されます。

日付キャッシュは、1つの表のみに対応付けできます。複数の表で日付キャッシュの共有はできません。次のすべての条件を満たす場合にのみ、表に対して日付キャッシュが作成されます。

- DATE\_CACHE パラメータが 0（ゼロ）以外に設定されている
- 表への格納のためにデータ型変換が必要な、1つ以上の日付値またはタイムスタンプ値（あるいはその両方）がロードされている
- ダイレクト・パス・ロードでロードされている

日付キャッシュの統計はログ・ファイルに書き込まれます。これらの統計を使用して、次のとおりダイレクト・パス・ロードのパフォーマンスを向上できます。

- キャッシュ・エントリの数が増え、キャッシュ・サイズより小さく、キャッシュ・ミスがない場合は、キャッシュ・サイズをより小さい値に設定できます。
- キャッシュ・ヒット（重複値が存在するエントリ）の数が小さく、キャッシュ・ミスの数が多い場合は、キャッシュ・サイズを大きくする必要があります。キャッシュ・サイズを大きくしすぎると、過剰なページング、過度のメモリー使用量などの他の問題が発生する場合があります。
- ほぼすべての入力データ値が一意の場合、日付キャッシュを使用してもパフォーマンスは向上しないため、使用する必要はありません。

---

**注意：** 日付キャッシュがデフォルトで使用可能な場合、最大値を超えたため使用禁止になると、日付キャッシュの統計は SQL\*Loader のログ・ファイルに書き込まれません。

---

キャッシュ・サイズを大きくしてもパフォーマンスが向上しない場合は、デフォルトの動作に戻すか、またはキャッシュ・サイズを 0（ゼロ）に設定します。パフォーマンス全体の向上は、ロードされる他の列のデータ型によっても異なります。ロードされる日付列の総数がロードされる他のデータ型より大きい場合、パフォーマンスは大幅に向上します。

### 参照：

- 4-5 ページの「[DATE\\_CACHE](#)」を参照してください。
- SQL\*Loader のログ・ファイルに書き込まれる日付キャッシュの統計の例は、8-5 ページの「[表ロード情報](#)」を参照してください。

## 複数 CPU システムのダイレクト・パス・ロードの最適化

複数 CPU システムでダイレクト・パス・ロードを実行する場合、SQL\*Loader ではデフォルトでマルチスレッドが使用されます。この場合の複数 CPU システムは、2 つ以上の CPU を持つ単一のシステムとして定義されます。

マルチスレッド・ロードとは、可能な場合、列配列をストリーム・バッファに変換し、ストリーム・バッファ・ロードが平行で実行されることを表します。この最適化は、次の場合に最も効果的です。

- 列配列が、ロード用に複数のダイレクト・パス・ストリーム・バッファを生成できる十分な大きさの場合。
- 入力フィールドのデータ型から Oracle の列データ型へのデータ変換が必要な場合。  
変換は、ストリーム・バッファのロード時に平行で実行されます。

この処理の状態は、次の例に示すとおり、SQL\*Loader のログ・ファイルに記録されます。

Total stream buffers loaded by SQL*Loader main thread:	47
Total stream buffers loaded by SQL*Loader load thread:	180
Column array rows:	1000
Stream buffer bytes:	256000

この例では、SQL\*Loader のロード・スレッドがメイン・スレッドをオフロードしています。これによって、ロード・スレッドがサーバーで現行のストリームをロードする一方で、メイン・スレッドは、次のストリーム・バッファを作成できます。

ロード・スレッドを使用して、できるだけ多くのストリーム・バッファ・ロードを実行することが目的です。これによって、列配列の行数の増加、またはストリーム・バッファのサイズの削減（あるいはその両方）を実現できます。SQL\*Loader のログ・ファイルの経過時間を監視することで、変更による効果を確認できます。詳細は、9-21 ページの「[列配列の行数およびストリーム・バッファ・サイズの指定](#)」を参照してください。

単一 CPU システム上では、最適化はデフォルトで無効になります。サーバーが他のシステム上にある場合は、マルチスレッドを手動で有効にするとパフォーマンスが向上します。

マルチスレッド・オプションを有効または無効にするには、SQL\*Loader のコマンドラインで MULTITHREADING パラメータを使用するか、または SQL\*Loader の制御ファイルに MULTITHREADING パラメータを指定します。

**参照：** ダイレクト・パス・ロードの概要については、『Oracle Call Interface プログラマーズ・ガイド』を参照してください。

## 索引メンテナンスの回避

従来型パスとダイレクト・パスの両方について、SQL\*Loader では表のすべての既存の索引がメンテナンスされます。

索引のメンテナンスを回避するには、次のいずれかの方法を使用します。

- ロードを始める前に索引を削除します。
- ロードを始める前に、選択した索引または索引パーティションを索引使用禁止状態に設定し、SKIP\_UNUSABLE\_INDEXES パラメータを使用します。
- SKIP\_INDEX\_MAINTENANCE パラメータを使用します（ダイレクト・パスの場合に限られるため、注意して使用してください）。

索引のメンテナンスを回避すると、ダイレクト・パス・ロード中に必要な領域を最小限にできます。その方法は次のとおりです。

- 一度に索引を作成できるため、各索引を別々に作成する場合に必要なソート用の（一時）セグメント領域を削減できます。
- 索引の作成時に、索引セグメントは1つのみ存在します（これに対し、新しいキーを古いキーにマージして新しい索引を作成するときには、一時的に3つのセグメントが存在します）。

表の全行数に対してロードする行数が多い場合、索引のメンテナンスを避けることは合理的です。ただし、比較的少数の行を大きな表に追加する場合は、索引の再ソートに非常に時間がかかることがあります。そのような場合は、従来型パス・ロードを使用するか、SQL\*Loader の SINGLEROW パラメータを使用します。詳細は、5-38 ページの「[SINGLEROW オプション](#)」を参照してください。

## ダイレクト・ロード、整合性制約およびトリガー

従来型パス・ロードでは、行配列の挿入には標準 SQL INSERT 文を使用します。このとき、整合性制約および挿入トリガーは自動的に適用されます。ただし、ダイレクト・パスでデータをロードする場合は、SQL\*Loader では一部の整合性制約およびすべてのデータベース・トリガーが使用禁止になります。このセクションでは、これらの機能に関するダイレクト・パス・ロードの使用について説明します。

### 整合性制約

整合性制約には、ダイレクト・パス・ロード時に自動的に使用禁止になるものがあります。また、使用禁止にならないものもあります。制約の詳細は、『Oracle9i アプリケーション開発者ガイド - 基礎編』の「制約によるデータ整合性のメンテナンス」を参照してください。

## 使用可能な制約

ダイレクト・パス・ロードでも有効な制約は次のとおりです。

- NOT NULL 制約
- 一意制約
- 主キー制約（NOT NULL 列における一意制約）

NOT NULL 制約は列配列の作成時にチェックされます。NOT NULL 制約に違反する行はすべて拒否されます。

一意制約は、ロードの最後で索引が再作成されるときに検証されます。一意制約の違反が検出されると、索引は索引使用禁止状態のままになります。詳細は、9-12 ページの「[使用禁止状態 \(Index Unusable\) のままの索引](#)」を参照してください。

## 使用禁止の制約

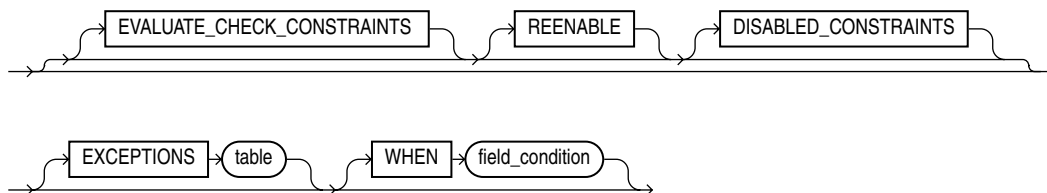
次の制約は、ダイレクト・パス・ロード時にデフォルトで自動的に使用禁止になります。

- CHECK 制約
- 参照制約（外部キー）

EVALUATE\_CHECK\_CONSTRAINTS 句を指定すると、CHECK 制約の使用禁止を変更できます。SQL\*Loader では、ダイレクト・パス・ロード中に CHECK 制約が評価されます。CHECK 制約に違反する行はすべて拒否されます。

## 制約を使用可能に戻す方法

REENABLE 句を指定しておくで、ロードの完了時に、整合性制約が自動的に使用可能に戻されます。REENABLE 句の構文は次のとおりです。



DISABLED\_CONSTRAINTS パラメータはオプションで、読みやすくするために使用します。EXCEPTIONS 句を使用する場合は、指定する表がすでに存在していて、その表への挿入が可能である必要があります。ここで指定する表には、整合性制約のいずれかに違反したすべての行の ROWID が格納されます。また、違反があった制約名も格納されます。この例外表の作成方法の詳細は、『Oracle9i SQL リファレンス』を参照してください。

SQL\*Loader ログ・ファイルには、使用禁止となっていた制約、および使用可能に戻された制約が記録されます。また、エラーが原因で制約を使用可能に戻せなかった場合または検査

できなかった場合はそのエラーが記録されます。さらに、ロードした表のそれぞれに指定された例外表の名前もログ・ファイルに書き込まれます。

REENABLE 句を使用しない場合は、表のすべての行が検査されるときに制約を手動で使用可能に戻す必要があります。ここで新しいデータにエラーが見つかったと、エラー・メッセージが生成されます。例外表を指定した場合は、違反のあった制約名と不良データの ROWID が、その表に書き込まれます。

REENABLE 句を使用すると、制約を自動的に使用可能に戻し、新しい行すべてを検証できます。新しいデータにエラーが見つからなかった場合は、検証された制約に自動的にマークが付けられます。新しいデータにエラーが見つかった場合は、ログ・ファイルにエラー・メッセージが書き込まれ、SQL\*Loader によって制約の状態に ENABLE NOVALIDATE というマークが付けられます。例外表を指定した場合は、違反のあった制約名と不良データの ROWID が、その表に書き込まれます。

---

**注意：** 通常、表制約が ENABLE NOVALIDATE の状態のままになっている場合、新しいデータは表へ挿入されますが、新しい無効なデータは挿入されません。ただし、SQL\*Loader のダイレクト・パス・ロードでは、この規則は施行されません。したがって、次のダイレクト・パス・ロードが無効なデータで実行される場合、無効なデータは挿入されますが、前述されたのと同じエラー・レポートおよび例外表の処理が実行されます。この例では、各ロードの前に外部表がクリーン・アウトされていない場合、その表には、重複するエントリが含まれます。重複するエントリは、次のような問合せを実行することによって、簡単に削除できます。

---

```
SELECT UNIQUE * FROM exceptions_table;
```

---

---

**注意：** 参照整合性の再検証は、表全体に対して実行する必要があります。そのため、少数の行を非常に大きい表にロードするときは、ダイレクト・パスではなく従来型パスを使用した方がパフォーマンスが向上することがあります。

---

## 挿入トリガー

ダイレクト・パス・ロードが始まると、表挿入トリガーも使用禁止になります。行のロードおよび索引の再作成が完了すると、使用禁止になっていたトリガーはすべて使用可能に戻されます。ログ・ファイルには、ロード時に使用禁止になっていたすべてのトリガーのリストが示されます。トリガーを使用可能に戻すときに 1 つでもエラーがあると、使用可能にできません。

整合性制約と異なり、挿入トリガーは、使用可能に戻っても表全体に対して再び適用されません。つまり、ダイレクト・パスでロードされた行に対しては、挿入トリガーは起動しません。ダイレクト・パスでロードした場合は、新しい行への挿入トリガーに相当する処理はすべて、アプリケーション側で実行する必要があります。

## 挿入トリガーの整合性制約への置換

アプリケーションは整合性制約を実行する場合、通常は挿入トリガーを使用します。アプリケーションが使用する挿入トリガーのほとんどは単純なため、Oracle の自動整合性制約に置き換えることができます。

## 自動制約が使用できない場合

挿入トリガーは、Oracle の自動整合性制約に置き換えられない場合があります。たとえば、挿入トリガーの中で表のロックアップ関数を使用して整合性チェックを行っている場合、自動制約は使用できません。これは、自動制約はカレント行における定数および列以外は参照できないためです。このようなトリガーと同じ処理を実現する方法が2つあります。この項ではその方法について説明します。

## 準備

いずれの方法の場合も、表に対して事前に行うべき作業があります。次に示す一般的なガイドラインに従って、表を準備してください。

1. ロードの前に、表に1バイトまたは1文字分の列 (VARCHAR2) を追加します。この列は、各行が「旧データ」か「新データ」かを示すためのものです。
2. この列の値が NULL の場合は「旧データ」を示すことにします。これは、NULL 列であれば領域を使用せずに済むためです。
3. ロード時に SQL\*Loader の CONSTANT パラメータを使用して、ロードしたすべての行に「新データ」を示すフラグを付けます。

この手順に従って準備すると、新しくロードした行が識別できるため、古い行に影響を与えずに新しいデータを操作できます。

## 更新トリガーの使用

一般に、挿入トリガーと同じ処理を実現する場合は、データベース更新トリガーを使用します。これは最も単純な方法です。例外を呼び出さない挿入トリガーの場合は、常にこの方法を使用できます。

1. 挿入トリガーと同じ処理を行う更新トリガーを作成します。  
トリガーをコピーします。「new.column\_name」というすべての箇所を「old.column\_name」に変更してください。
2. 現行の更新トリガーがある場合は、それを新しい更新トリガーに置き換えます。
3. 「新データ」のフラグを NULL に変更して、表を更新します。これによって、更新トリガーが起動します。
4. 元の更新トリガーがある場合は、それをリストアします。

トリガーの動作によっては、この操作中に表に対する排他的更新アクセスが必要となる場合もあります。排他的更新アクセスを実行すると、他のユーザーが修正する行に、誤ってこのトリガーが適用されることはありません。

## 例外処理と同じ処理の実現

挿入トリガーの中で例外を呼び出している場合、それと同じ処理を行うには、さらに作業が必要です。例外を呼び出すということは、表にその行を挿入しないということです。この処理を更新トリガーで実現するには、ロードした行に削除フラグを付けておく必要があります。

この場合、「新データ」列を削除フラグに使用することはできません。更新トリガーでは、その起動元である列を修正できないためです。したがって、表にもう 1 列追加する必要があります。ここで追加する列は、削除する行を示すためのものです。NULL 値の場合は、行が有効であることを示します。挿入トリガーで例外を呼び出すと、常に、更新トリガーによって追加列にフラグが設定されます。これによって、その行が無効であることが示されます。

要約すると、挿入トリガーで例外を呼び出している場合は、次の条件を満たすと、同じ処理を更新トリガーで実現できます。

- 表に列を 2 つ追加する（通常は NULL）。
- 表を排他的に更新できる（必要な場合）。

## ストアド・プロシージャの使用

次に示すプロシージャは、どのような場合でも使用できますが、その実装はより複雑になります。このプロシージャは、挿入トリガーで例外を呼び出すときに使用できます。そのとき、2 番目の列を追加する必要はありません。また、更新トリガーとは処理が異なるため、表に対する排他的アクセス権限がなくても使用できます。

1. 次のようにして、挿入トリガーと同じ処理を行うストアド・プロシージャを作成します。
  - a. 表から新しい行をすべて選択するように、カーソルを宣言します。
  - b. 処理ループの中でカーソルをオープンし、1 回に 1 行ずつフェッチします。
  - c. 挿入トリガーでの操作を実行します。
  - d. 操作が正常に終了した場合は、「新データ」のフラグを NULL に変更します。
  - e. 操作が失敗した場合は、「新データ」のフラグを「不良データ」に変更します。
2. SQL\*Plus などの管理ツールを使用して、このストアド・プロシージャを実行します。
3. プロシージャの実行後、表の中に「不良データ」フラグの付いた行がないかどうかを調べます。
4. 不良の行を更新または削除します。
5. 挿入トリガーを使用可能に戻します。



**参照：** カーソル管理の詳細は、『PL/SQL ユーザーズ・ガイドおよびリファレンス』を参照してください。

## 永続的に使用禁止のトリガーおよび制約

SQL\*Loader では、トリガーおよび制約を使用禁止にするために、ロードされる表にいくつかのロックを獲得する必要があります。競合するプロセスが表のトリガーまたは制約を使用可能にしているときに、SQL\*Loader でその表のトリガーまたは制約を使用禁止にしようとした場合、SQL\*Loader ではその表に関して排他的アクセス権を獲得することはできません。

この場合、SQL\*Loader では、できるかぎり問題のないように処理が実行されます。ロード終了前に、SQL\*Loader では使用禁止のトリガーおよび制約が使用可能に戻されます。ただし、表ロックが原因で SQL\*Loader の処理を継続できなくなった場合は、SQL\*Loader でトリガーや制約を使用可能にする処理も実行されないことがあります。この場合、トリガーおよび制約は、手動で使用可能にするまで永続的に使用できない状態になります。

このような状況はまれですが、発生する可能性があります。このような状況を回避するには、ダイレクト・ロードの処理中は、表のトリガーまたは制約を使用可能にするアプリケーションを実行しないことをお勧めします。

適切なロックを獲得できなかったためにダイレクト・ロードが異常終了した場合は、ログ・ファイルを確認してください。ログ・ファイルには、使用禁止になっていたトリガーおよび制約と、それらを使用可能に戻そうと試みた履歴が記録されます。SQL\*Loader によって使用可能に戻せなかったトリガーや制約は、『Oracle9i SQL リファレンス』で説明されている ALTER TABLE 文の ENABLE 句を使用して、手動で使用可能にする必要があります。

## 従来型パスの同時ロードによるパフォーマンスの向上

トリガーまたは整合性制約の問題があっても、より高速なロードを実現する場合は、従来型パスによる同時ロードの使用を考えてください。つまり、複数 CPU システムで同時に複数のセッションでロードを実行します。入力データ・ファイルを論理レコードの境界で別々のファイルに分割し、それらの各入力データ・ファイルを従来型パス・ロード・セッションでロードします。このロードには、次のような特長があります。

- 複数 CPU システムでの単一従来型パス・ロードよりは速くなりますが、ダイレクト・ロードほど速くはありません。
- トリガーが起動されて整合性制約がロードされた行に適用され、標準 DML 実行ロジックによって索引がメンテナンスされます。

## パラレル・データ・ロード・モデル

この項では、データのロードに必要な所要時間を最小限にするために使用される、並列処理の 3 つの基本モデルについて説明します。

- 従来型パスによる同時ロード

- ダイレクト・パス・ロードによるセグメント間同時処理
- ダイレクト・パス・ロードによるセグメント内同時処理

## 従来型パスによる同時ロード

同時に複数の従来型パス・ロード・セッションを実行する方法の詳細は、9-29 ページの「[従来型パスの同時ロードによるパフォーマンスの向上](#)」を参照してください。同一または異なるオブジェクトを制限なしで同時にロードする場合に、この方法を使用できます。

## ダイレクト・パスによるセグメント間同時処理

セグメント間同時処理は、異なるオブジェクトを同時にロードする場合に使用できます。この方法は、異なる表の同時ダイレクト・パス・ロード、または同じ表の異なるパーティションの同時ダイレクト・パス・ロードに適用できます。

1つのパーティションのダイレクト・パス・ロードを行う場合は、次のことを考慮します。

- ローカル索引は、ロードによってメンテナンスされます。
- グローバル索引は、ロードではメンテナンスできません。
- 参照整合性および CHECK 制約は使用禁止にする必要があります。
- トリガーは使用禁止にする必要があります。
- 入力データは事前にパーティション化する必要があります（パーティション化しない場合、多くのレコードが拒否され、パフォーマンスが低下します。）

## ダイレクト・パスによるセグメント内同時処理

SQL\*Loader では、複数のセッションを同時に実行して、同一の表またはパーティション表の同一パーティションに対してダイレクト・パス・ロードを実行できます。複数の SQL\*Loader セッションを実行すると、システムで使用可能なリソースを与えられればダイレクト・パス・ロードのパフォーマンスが向上します。

このデータ・ロード方法は、DIRECT および PARALLEL パラメータに true を設定することによって使用でき、「パラレル・ダイレクト・パス・ロード」とも呼ばれます。

並列化はユーザーによって管理されるものであることを理解しておいてください。PARALLEL パラメータに true を設定した場合、複数の同時ダイレクト・パス・ロード・セッションのみが可能になります。

## パラレル・ダイレクト・パス・ロードの制限

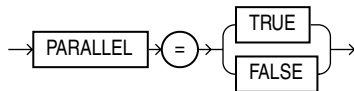
パラレル・ダイレクト・パス・ロードには次の制限があります。

- ローカル索引もグローバル索引もロードによってメンテナンスできません。
- 参照整合性および CHECK 制約は使用禁止にする必要があります。
- トリガーは使用禁止にする必要があります。
- 行は追加 (APPEND) のみできます。REPLACE、TRUNCATE および INSERT は使用できません (これは、個別のロードによって整合性がとられないためです)。パラレル・ロードの前に表を切り捨てる必要がある場合は、手動で行ってください。

1 つのパーティションのパラレル・ダイレクト・パス・ロードを行う場合は、まず、データをパーティション化してください (そうしない場合は、パーティション不一致によるレコード拒否のオーバーヘッドのため、ロード速度が遅くなります)。

## 複数の SQL\*Loader セッションの初期化

入力元となるデータ・ファイルは、SQL\*Loader セッションごとに異なります。同じ表にダイレクト・ロードを実行するセッションすべてに対して、PARALLEL 句に true を設定する必要があります。構文は次のとおりです。



PARALLEL は、コマンドラインまたはパラメータ・ファイルに指定できます。また、OPTION 句を使用して制御ファイルに指定することもできます。

たとえば、1 つの表について 3 つの SQL\*Loader ダイレクト・パス・ロード・セッションを起動するには、オペレーティング・システムのプロンプトで、次のコマンドを実行します。

```

sqlldr USERID=scott/tiger CONTROL=load1.ctl DIRECT=TRUE PARALLEL=true
sqlldr USERID=scott/tiger CONTROL=load2.ctl DIRECT=TRUE PARALLEL=true
sqlldr USERID=scott/tiger CONTROL=load3.ctl DIRECT=TRUE PARALLEL=true
  
```

このコマンドは、別々のセッションで実行するか、またはオペレーティング・システムがサポートしている場合には別々のバックグラウンド・ジョブとして実行してください。複数の制御ファイルを使用していることに注意してください。そうすることによって、ダイレクト・パス・ロードで使用するファイルをより柔軟に指定できます。

---

**注意：** パラレル・ロード時には、索引はメンテナンスされません。ロード完了後に、索引をすべて手動で（再）作成または再構築する必要があります。パラレル・ロード後に大きな索引を構築する場合、パラレル索引作成機能またはパラレル索引再構築機能を使用すると処理を高速化できます。

---

PARALLEL 句を使用してロードを実行すると、SQL\*Loader によって同時実行セッションごとに一時セグメントが作成されます。それらの一時セグメントは、ロード完了時にマージされます。マージによって作成されたセグメントは、セグメントの最高水位標より上にあるデータベースの既存のセグメントに追加されます。各ローダー・セッションの各セグメントで使用された最後のエクステントは、空き領域をすべて切り捨ててから、SQL\*Loader セッションの他のエクステントと組み合わせることができます。

## パラレル・ダイレクト・パス・ロードのパラメータ

パラレル・ダイレクト・パス・ロードを実行する場合、SQL\*Loader によって割り当てられる一時セグメントの属性を指定してできるオプションがあります。

### 一時セグメントの指定

最大の出入カスループットを得るために、同時実行のダイレクト・パス・ロード・セッションで、各ファイルを別のディスクに置いて使用することをお薦めします。ロードされるオブジェクト（表またはパーティション）の表領域にある有効なデータ・ファイルであればどれでも、OPTIONS 句の FILE パラメータを使用してファイル名を指定できます。

次に例を示します。

```
LOAD DATA
INFILE 'load1.dat'
INSERT INTO TABLE emp
OPTIONS(FILE='/dat/data1.dat')
(empno POSITION(01:04) INTEGER EXTERNAL NULLIF empno=BLANKS
...

```

同時実行する各 SQL\*Loader セッションのコマンドラインでも、FILE パラメータを指定できます。ただし、その指定は、そのセッションでロードされるすべてのオブジェクトにグローバルに適用されます。

**FILE パラメータの使用** パラレル・ダイレクト・パス・ロードでは、Oracle データベース・サーバーの FILE パラメータに次の制限があります。

- **非パーティション表の場合：**指定されたファイルは、ロードする表と同じ表領域に存在する必要があります。

- **パーティション表の1つのパーティションをロードする場合:** 指定したファイルは、ロードするパーティションの表領域に存在する必要があります。
- **パーティション表の表全体をロードする場合:** 指定されたファイルは、ロードするすべてのパーティションと同じ表領域に存在する必要があります。つまり、すべてのパーティションは同じ表領域に存在する必要があります。

**STORAGE パラメータの使用** STORAGE パラメータを使用して、パラレル・ダイレクト・パス・ロード用に割り当てられる一時セグメントの記憶域属性を指定できます。STORAGE パラメータが使用されない場合は、ロードされるオブジェクト（表、パーティション）が存在するセグメントの記憶域属性が使用されます。また、STORAGE パラメータが指定されていない場合は、SQL\*Loader で EXTENTS 用にデフォルトの 2KB が使用されます。

```
OPTIONS(STORAGE=(MINEXTENTS n1 MAXEXTENTS n2 INITIAL n3 [K|M]
NEXT n4 [K|M] PCTINCREASE n5))
```

たとえば、次の STORAGE 句が使用されます。

```
OPTIONS (STORAGE=(INITIAL 100M NEXT 100M PCTINCREASE 0))
```

STORAGE パラメータを使用できるのは制御ファイル内のみで、コマンドラインでは使用できません。PCTINCREASE を 0（ゼロ）に設定すること、INITIAL または NEXT 値を設定すること以外には、STORAGE パラメータは使用しないでください（将来的には無視されるようになる可能性があります）。

## パラレル・ダイレクト・パス・ロード後の制約の使用可能化

すべてのデータのロード完了後に、制約およびトリガーを手動で使用可能にしてください。

それぞれの SQL\*Loader セッションで、ダイレクト・パス・ロードの後に表の制約が使用可能に戻されることがあるため、あるセッションで、他のセッションがデータのロードを終える前に、制約が使用可能に戻される可能性があります。この場合、ロードを完了する最初のセッションでは、残りのセッションで表のロックが共有されているため、制約を使用可能にできません。

ダイレクト・パス・ロードの後、一部の制約が使用可能に戻っていない可能性があるため、ロードの完了後、制約の状態を調べ、制約が使用可能になったことを確認する必要があります。

## 主キー制約および一意制約

主キーおよび一意制約が有効の場合、表に索引が作成されます。その後、その表が非常に大きい場合は、ダイレクト・パス・ロード後に索引を使用可能にするまでに非常に長い時間がかかる可能性があります。ロード後に、これらの制約を手動で有効にしてください（また、自動で有効にする機能を指定しないでください）。これによって、必要な索引をパラレルに手動で作成し、制約を有効にするまでの時間を節約できます。

**参照：**『Oracle9i データベース・パフォーマンス・ガイドおよびリファレンス』を参照してください。

## 一般的なパフォーマンス改善のヒント

ロードするデータの形式について制御が可能な場合は、次に示すヒントを利用してロード・パフォーマンスを改善できます。

- 論理レコードの処理を効率化します。
  - 物理レコードと論理レコードの 1 対 1 のマップを使用します（`continueif`、`concatenate` を使用しない）。
  - ソフトウェアで物理レコードの境界を簡単に判断できるようにします。ファイル処理オプション文字列「`FIX nnn`」または「`VAR`」を使用します。ほとんどのプラットフォーム（UNIX、NT など）では、デフォルト（ストリーム・モード）を使用する場合、**SQL\*Loader** で各物理レコードをスキャンしてレコード終了記号（改行文字）を探す必要があります。
- フィールド設定を効率化します。フィールド設定とは、データ・ファイルのフィールドを、ロードされる表の対応する列にマップする処理です。マップ機能は、制御ファイルのフィールド記述によって制御されます。フィールド設定は（データ変換とともに）、ほとんどのロードで CPU サイクルを最も使用する処理です。
  - デリミタ付きのフィールドを使用しないようにします。固定位置のフィールドを使用します。デリミタ付きフィールドを使用すると、**SQL\*Loader** で入力データをスキャンしてデリミタを見つけなければなりません。固定位置フィールドを使用すると、フィールド設定は単純なポインタの算出によって（非常に速く）行われます。
  - （`PRESERVE BLANKS` を使用する）必要がない場合は、空白を切り捨てないでください。
- 変換を効率化します。**SQL\*Loader** では、キャラクタ・セット変換、データ型変換などのいくつかの変換が行われます。こうした変換がない場合、処理は最も速くなります。
  - 可能な場合は、シングルバイト・キャラクタ・セットを使用します。
  - キャラクタ・セット変換はできるだけ行わないようにします。**SQL\*Loader** では、次の 4 つのキャラクタ・セットがサポートされています。

- \* クライアント・キャラクタ・セット (クライアント sqlldr プロセスの NLS\_LANG)
- \* データ・ファイル・キャラクタ・セット (通常、クライアント・キャラクタ・セットと同じ)
- \* データベース・サーバー・キャラクタ・セット
- \* データベース・サーバーの各国語キャラクタ・セット

すべてのキャラクタ・セットが同じ場合、パフォーマンスは最適化されます。ダイレクト・パスでは、データ・ファイル・キャラクタ・セットとデータベース・サーバー・キャラクタ・セットが同じである場合、最適なパフォーマンスが得られます。キャラクタ・セットが同じ場合、キャラクタ・セット変換用バッファは割り当てられません。

- ダイレクト・パス・ロードを使用します。
- SORTED INDEX 句を使用します。
- NULLIF 句および DEFAULTIF 句は必要な場合以外は使用しないようにします。これらの句は、関連する句を持つ各列にロードされるすべての行について、評価される必要があります。
- 可能な場合は、パラレル・ダイレクト・パス・ロードおよびパラレル索引作成を使用します。
- CONCATENATE 句および COLUMNARRAYROWS 句の両方に大きい値を指定する場合は、パフォーマンスへの影響に注意します。詳細は、5-26 ページの「[CONCATENATE を使用した論理レコードの作成](#)」を参照してください。

さらに、11-6 ページの「[外部表使用時のパフォーマンスのヒント](#)」で説明しているパフォーマンスのヒントは、SQL\*Loader にも適用できます。





---

## SQL\*Loader の事例

この章では、事例を通して、SQL\*Loader の機能をいくつか説明します。次のように、簡単な例から複雑な例の順で掲載しています。

---

**注意：** この章では、UNIX 固有の起動コマンドである `sqlldr` などを使用します。オペレーティング・システムで使用する正しいコマンドの詳細は、ご使用のオペレーティング・システム固有の Oracle マニュアルを参照してください。

---

この章の内容は、次のとおりです。

- 事例
- 事例用ファイル
- 各事例で使用する表
- ロード結果の確認
- 参照および注意
- 事例 1: 可変長データのロード
- 事例 2: 固定形式フィールドのロード
- 事例 3: 自由区分形式ファイルのロード
- 事例 4: 結合された物理レコードのロード
- 事例 5: 複数表へのデータのロード
- 事例 6: ダイレクト・パス・ロード方式を使用したデータのロード
- 事例 7: 書式化されたレポートからのデータの抽出
- 事例 8: パーティション化された表のロード
- 事例 9: LOBFILE のロード (CLOB)

- 
- 事例 10: REF フィールドおよび VARRAY のロード
  - 事例 11: Unicode キャラクタ・セットのデータのロード

## 事例

この章で示す事例は、次のとおりです。

- **事例 1: 可変長データのロード** (10-6 ページ) : ストリーム形式のレコードをロードします。ここで扱うレコードのフィールドは、カンマで区切るか、または引用符で囲まれています。データは制御ファイルの終わりに入っています。
- **事例 2: 固定形式フィールドのロード** (10-9 ページ) : 別のデータ・ファイルからデータをロードします。
- **事例 3: 自由区分形式ファイルのロード** (10-12 ページ) : フィールドがデリミタ付きで順序番号が付いている、ストリーム形式のレコードのデータをロードします。データは制御ファイルの終わりに入っています。
- **事例 4: 結合された物理レコードのロード** (10-15 ページ) : 複数の物理レコードを結合して、データベースの 1 行に対応する論理レコードを構成します。
- **事例 5: 複数表へのデータのロード** (10-19 ページ) : 1 回の実行で、データを複数の表にロードします。
- **事例 6: ダイレクト・パス・ロード方式を使用したデータのロード** (10-25 ページ) : ダイレクト・パス・ロード方法を使用してデータをロードします。
- **事例 7: 書式化されたレポートからのデータの抽出** (10-29 ページ) : 書式化されたレポートからデータを抽出します。
- **事例 8: パーティション化された表のロード** (10-34 ページ) : パーティション表をロードします。
- **事例 9: LOBFILE のロード (CLOB)** (10-39 ページ) : FILLER フィールド (res\_file) を使用して、複数の LOBFILE を emp 表にロードし、resume と呼ばれる CLOB 列を emp 表に付加します。
- **事例 10: REF フィールドおよび VARRAY のロード** (10-43 ページ) : OID を主キーとして利用するカスタマ表をロードして、注文した商品を VARRAY に格納します。カスタマ表および VARRAY の注文商品を参照する注文表をロードします。
- **事例 11: Unicode キャラクタ・セットのデータのロード** (10-48 ページ) : Unicode キャラクタ・セットの UTF16 データを、リトル・エンディアンバイト順序でロードします。この事例は、文字長セマンティクスを使用します。

# 事例用ファイル

SQL\*Loader の配布メディアには、各事例に関する次のファイルが含まれます。

- 制御ファイル (ulcase5.ctl など)
- データ・ファイル (ulcase5.dat など)
- セットアップ・ファイル (ulcase5.sql など)

事例用のサンプル・データが制御ファイルに含まれている場合は、その事例用の .dat ファイルはありません。

事例用の特別なセットアップ手順がない場合は、その事例用の .sql ファイルがないこともあります。スクリプトの開始（セットアップ）および終了（クリーンアップ）は各事例番号の後の S および E で示されます。

表 10-1 に、各事例に関連のあるファイルを示します。

表 10-1 事例用ファイルおよび関連ファイル

事例	.ctl	.dat	.sql
1	あり	なし	あり
2	あり	あり	なし
3	あり	なし	あり
4	あり	あり	あり
5	あり	あり	あり
6	あり	あり	あり
7	あり	あり	あり (S、E)
8	あり	あり	あり
9	あり	あり	あり
10	あり	なし	あり
11	あり	あり	あり

**注意：** 事例用ファイルの実際の名前は、オペレーティング・システムによって異なります。ご使用のオペレーティング・システム固有の Oracle マニュアルで、正確な名前を参照してください。

## 各事例で使用する表

事例は、ユーザー ID `scott/tiger` のユーザーが、デモンストレーション用の標準 Oracle データベースの `emp` 表および `dept` 表を所有している場合を想定して作成してあります（事例の中では、さらに列が追加されていることもあります）。

### emp 表の内容

<code>empno</code>	<code>NUMBER(4) NOT NULL,</code>
<code>ename</code>	<code>VARCHAR2(10),</code>
<code>job</code>	<code>VARCHAR2(9),</code>
<code>mgr</code>	<code>NUMBER(4),</code>
<code>hiredate</code>	<code>DATE,</code>
<code>sal</code>	<code>NUMBER(7,2),</code>
<code>comm</code>	<code>NUMBER(7,2),</code>
<code>deptno</code>	<code>NUMBER(2)</code>

### dept 表の内容

<code>deptno</code>	<code>NUMBER(2) NOT NULL,</code>
<code>dname</code>	<code>VARCHAR2(14),</code>
<code>loc</code>	<code>VARCHAR2(13)</code>

## ロード結果の確認

ロード結果を確認するには、SQL\*Plus を起動して、事例でロードされた表から選択操作を行います。次のように行います。

1. システム・プロンプトで次のように入力して、SQL\*Plus を `scott/tiger` で起動します。

```
sqlplus scott/tiger
```

SQL プロンプトが表示されます。

2. SQL プロンプトで `SELECT` 文を使用して、事例がロードされた表からすべての行を選択します。たとえば、`emp` 表がロードされた場合、次のように入力します。

```
SQL> SELECT * FROM emp;
```

`emp` 表の各行の内容が表示されます。

## 参照および注意

各事例が対象としている SQL\*Loader の機能の詳細は、各事例の最初の項を参照してください。

各事例の制御ファイルとログ・ファイルの左側に示す数字は、ファイル内に実際に存在する数字ではありません。これらの数字は、その後の注意書きの番号と対応しています。制御ファイルを記述する場合、この番号は付けなくても構いません。

## 事例 1: 可変長データのロード

ここでは、次のことを説明します。

- 1 つの表と 3 つの列がロードされることを示す簡単な制御ファイルを使用します。
- 制御ファイルがロードするデータを含んでいるため、別のデータ・ファイルは存在しません。詳細は、5-10 ページの「[BEGIN DATA による制御ファイルのデータの識別](#)」を参照してください。
- 2 種類のデリミタ付きフィールド (TERMINATED BY で指定したデリミタおよび ENCLOSED BY で指定したデリミタ) を使用して、ストリーム形式のデータをロードします。詳細は、6-44 ページの「[空白の切捨てが可能なデータ型に対するフィールド長指定](#)」を参照してください。

## 事例 1 の制御ファイル

制御ファイルは ulcase1.ctl です。

```
1)  LOAD DATA
2)  INFILE *
3)  INTO TABLE dept
4)  FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
5)  (deptno, dname, loc)
6)  BEGIN DATA
    12,RESEARCH,"SARATOGA"
    10,"ACCOUNTING",CLEVELAND
    11,"ART",SALEM
    13,FINANCE,"BOSTON"
    21,"SALES",PHILA.
    22,"SALES",ROCHESTER
    42,"INT'L","SAN FRAN"
```

**注意：**

1. 制御ファイルの先頭には、LOAD DATA 文が必要です。
2. INFILE \* は、データが外部ファイルではなく制御ファイル内にあることを示します。
3. データをロードする表 (dept) を識別するために、INTO TABLE 文が必要です。デフォルトでは、SQL\*Loader でレコードを挿入するには、表は空である必要があります。
4. FIELDS TERMINATED BY は、データがカンマで終わることを示します。また、一重引用符で囲むこともできます。データ型のデフォルトはどのフィールドでも CHAR です。
5. ロードする列の名前をカッコで囲んで指定します。データ型および長さは指定されないため、デフォルトは最大 255 バイトの CHAR 型になります。
6. BEGINDATA はデータの始まりを指定します。

## 事例 1 の実行

次に示す手順で、事例を実行します。

1. システム・プロンプトで次のように入力して、SQL\*Plus を scott/tiger で起動します。

```
sqlplus scott/tiger
```

SQL プロンプトが表示されます。

2. SQL プロンプトで、この事例用の SQL スクリプトを次のように実行します。

```
SQL> @ulcase1
```

事例用の表が準備および移入されると、システム・プロンプトに戻ります。

3. システム・プロンプトで、次のように入力し、SQL\*Loader を起動して事例を実行します。

```
sqlldr USERID=scott/tiger CONTROL=ulcase1.ctl LOG=ulcase1.log
```

SQL\*Loader によって dept 表にデータがロードされ、ログ・ファイルが作成されると、システム・プロンプトに戻ります。事例の実行結果を確認するには、ログ・ファイルを調べます。

## 事例 1 のログ・ファイル

次に、ログ・ファイルの一部を示します。

```
Control File:  ulcase1ctl
Data File:     ulcase1ctl
  Bad File:    ulcase1.bad
  Discard File: none specified

(Allow all discards)

Number to load: ALL
Number to skip: 0
Errors allowed: 50
Bind array:    64 rows, maximum of 256000 bytes
Continuation:  none specified
Path used:     Conventional
```

Table DEPT, loaded from every logical record.  
Insert option in effect for this table: INSERT

Column Name	Position	Len	Term	Encl	Datatype
-----					
1) DEPTNO	FIRST	*	,	O(")	CHARACTER
DNAME	NEXT	*	,	O(")	CHARACTER
2) LOC	NEXT	*	,	O(")	CHARACTER

Table DEPT:  
  7 Rows successfully loaded.  
  0 Rows not loaded due to data errors.  
  0 Rows not loaded because all WHEN clauses were failed.  
  0 Rows not loaded because all fields were null.

```
Space allocated for bind array:          49536 bytes (64 rows)
Read  buffer bytes: 1048576
```

```
Total logical records skipped:    0
Total logical records read:       7
Total logical records rejected:    0
Total logical records discarded:   0
```

Run began on Wed Feb 27 14:10:13 2002  
Run ended on Wed Feb 27 14:10:14 2002

```
Elapsed time was:    00:00:01.53
CPU time was:       00:00:00.20
```



**注意：**

1. 各フィールドの位置と長さは、入力ファイルのデリミタに基づいてレコードごとに判断されます。
2. `o(")` は、オプションとして、引用符でデータを囲むこともできることを示します。

## 事例 2: 固定形式フィールドのロード

ここでは、次のことを説明します。

- 別のデータ・ファイルを使用します。詳細は、5-7 ページの「[データ・ファイルの指定](#)」を参照してください。
- データ変換を行います。詳細は、6-22 ページの「[データ型の変換](#)」を参照してください。

この事例では、フィールドの位置とデータ型を明示的に指定します。

### 事例 2 の制御ファイル

制御ファイルは `ulcase2.ct1` です。

```

1)  LOAD DATA
2)  INFILE 'ulcase2.dat'
3)  INTO TABLE emp
4)  (empno          POSITION(01:04)    INTEGER EXTERNAL,
     ename          POSITION(06:15)    CHAR,
     job            POSITION(17:25)    CHAR,
     mgr            POSITION(27:30)    INTEGER EXTERNAL,
     sal            POSITION(32:39)    DECIMAL EXTERNAL,
     comm           POSITION(41:48)    DECIMAL EXTERNAL,
5)  deptno          POSITION(50:51)    INTEGER EXTERNAL)
```

**注意：**

1. 制御ファイルの先頭には、LOAD DATA 文が必要です。
2. データを含むファイルの名前は、INFILE パラメータの後に置かれます。
3. データをロードする表を識別するために、INTO TABLE 文が必要です。
4. 番号 4) と 5) の間の行は、列名およびその列にロードするデータ・ファイル中のデータの位置を示します。empno、ename、job などは、emp 表にある列の名前です。データ型 (INTEGER EXTERNAL、CHAR、DECIMAL EXTERNAL) は、emp 表の列のデータ型ではなくデータ・ファイル内のデータ・フィールドのデータ型です。
5. 一連の列指定は、カッコで囲みます。

## 事例 2 のデータ・ファイル

ファイル `ulcase2.dat` のデータの一部を示します。空白フィールドは自動的に NULL に設定されます。

7782	CLARK	MANAGER	7839	2572.50		10
7839	KING	PRESIDENT		5500.00		10
7934	MILLER	CLERK	7782	920.00		10
7566	JONES	MANAGER	7839	3123.75		20
7499	ALLEN	SALESMAN	7698	1600.00	300.00	30
7654	MARTIN	SALESMAN	7698	1312.50	1400.00	30
7658	CHAN	ANALYST	7566	3450.00		20
7654	MARTIN	SALESMAN	7698	1312.50	1400.00	30

## 事例 2 の実行

次に示す手順で、事例を実行します。事例 1 を実行してある場合、手順 1 および手順 2 を省略できます。`ulcase1.sql` スクリプトは、事例 1 および事例 2 で共通です。

1. システム・プロンプトで次のように入力して、SQL\*Plus を `scott/tiger` で起動します。

```
sqlplus scott/tiger
```

SQL プロンプトが表示されます。

2. SQL プロンプトで、この事例用の SQL スクリプトを次のように実行します。

```
SQL> @ulcase1
```

事例用の表が準備および移入されると、システム・プロンプトに戻ります。

3. システム・プロンプトで、次のように入力し、SQL\*Loader を起動して事例を実行します。

```
sqlldr USERID=scott/tiger CONTROL=ulcase2.ctl LOG=ulcase2.log
```

SQL\*Loader によって表にデータがロードされ、ログ・ファイルが作成されると、システム・プロンプトに戻ります。事例の実行結果を確認するには、ログ・ファイルを調べます。

この例で `emp` 表にロードされたレコードには、部門番号が付けられています。`dept` 表が最初にロードされていないと、参照整合性チェックによってそれらのレコードは拒否されます（参照整合性制約が `emp` 表で有効な場合）。

## 事例 2 のログ・ファイル

次に、ログ・ファイルの一部を示します。

```
Control File:  ulcase2ctl
Data File:     ulcase2.dat
  Bad File:    ulcase2.bad
  Discard File: none specified

(Allow all discards)

Number to load: ALL
Number to skip: 0
Errors allowed: 50
Bind array:     64 rows, maximum of 256000 bytes
Continuation:   none specified
Path used:      Conventional
```

Table EMP, loaded from every logical record.  
Insert option in effect for this table: INSERT

Column Name	Position	Len	Term Encl	Datatype
EMPNO	1:4	4		CHARACTER
ENAME	6:15	10		CHARACTER
JOB	17:25	9		CHARACTER
MGR	27:30	4		CHARACTER
SAL	32:39	8		CHARACTER
COMM	41:48	8		CHARACTER
DEPTNO	50:51	2		CHARACTER

Table EMP:

```
  7 Rows successfully loaded.
  0 Rows not loaded due to data errors.
  0 Rows not loaded because all WHEN clauses were failed.
  0 Rows not loaded because all fields were null.
```

```
Space allocated for bind array:          3840 bytes (64 rows)
Read  buffer bytes: 1048576
```

```
Total logical records skipped:          0
Total logical records read:              7
Total logical records rejected:          0
Total logical records discarded:         0
```

Run began on Wed Feb 27 14:17:39 2002

Run ended on Wed Feb 27 14:17:39 2002

Elapsed time was: 00:00:00.81

CPU time was: 00:00:00.15

## 事例 3: 自由区分形式ファイルのロード

ここでは、次のことを説明します。

- ストリーム形式のデータ（ENCLOSED BY で指定した文字で囲まれたデータおよび TERMINATED BY で指定した文字で区切られているデータ）をロードします。詳細は、6-44 ページの「[デリミタ付きフィールド](#)」を参照してください。
- DATE データ型を使用して日付をロードします。詳細は、6-15 ページの「[日時データ型および期間データ型](#)」を参照してください。
- SEQUENCE 番号を使用して、ロードしたデータに一意キーを生成します。詳細は、6-54 ページの「[列への一意の順序番号の設定](#)」を参照してください。
- APPEND を使用して、新しいレコードを挿入する前に表を空にする必要がないことを指定します。詳細は、5-31 ページの「[表固有のロード方法](#)」を参照してください。
- 制御ファイルでコメントを使用します。コメントは先頭に二重ハイフンを付けます。詳細は、5-4 ページの「[制御ファイルのコメント](#)」を参照してください。

## 事例 3 の制御ファイル

この制御ファイルからは、事例 2 と同じ表がロードされます。ただし、列が 3 つ (hiredate, projno, loadseq) 追加されています。デモンストレーション用の emp 表には、projno 列および loadseq 列が含まれていません。この制御ファイルをテストする場合は、次のコマンドを入力して emp 表にこの 2 つの列を追加します。

```
ALTER TABLE emp ADD (projno NUMBER, loadseq NUMBER);
```

データは、事例 2 の形式とは異なります。引用符で囲まれているデータと、カンマで区切られているデータがあり、deptno と projno の値はコロンで区切られています。

```
1)  -- Variable-length, delimited, and enclosed data format
   LOAD DATA
2)  INFILE *
3)  APPEND
   INTO TABLE emp
4)  FIELDS TERMINATED BY "," OPTIONALLY ENCLOSED BY '"'
   (empno, ename, job, mgr,
5)  hiredate DATE(20) "DD-Month-YYYY",
   sal, comm, deptno CHAR TERMINATED BY ':',
   projno,
6)  loadseq SEQUENCE(MAX,1))
```

```

7)  BEGINDATA
8)  7782, "Clark", "Manager", 7839, 09-June-1981, 2572.50,, 10:101
    7839, "King", "President", , 17-November-1981,5500.00,,10:102
    7934, "Miller", "Clerk", 7782, 23-January-1982, 920.00,, 10:102
    7566, "Jones", "Manager", 7839, 02-April-1981, 3123.75,, 20:101
    7499, "Allen", "Salesman", 7698, 20-February-1981, 1600.00,
    (same line continued)                300.00, 30:103
    7654, "Martin", "Salesman", 7698, 28-September-1981, 1312.50,
    (same line continued)                1400.00, 3:103
    7658, "Chan", "Analyst", 7566, 03-May-1982, 3450,, 20:101

```

#### 注意：

1. ファイル内の任意のコマンドラインにコメントを入力できます。ただし、データにはコメントを表示できません。コメントは二重ハイフンの後に記述します。二重ハイフンは、コマンドライン内の任意の位置に置くことができます。
2. INFILE \* は、データが制御ファイルの終わりにあることを示します。
3. APPEND は、表内にすでに行が含まれていても、データをロードすることを示します。表を空にする必要はありません。
4. データ・フィールドの終了記号のデフォルトはカンマです。また、二重引用符 (") で囲むこともできます。
5. hiredate 列にロードするデータの書式は、DD-Month-YYYY です。データ・フィールドの長さは、最大の 20 に指定されます。最大長は、デフォルトのバイト長セマンティクスを使用したバイト単位です。文字長セマンティクスが使用された場合、長さは文字単位になります。長さの指定がない場合、日付フィールドの長さは、日付マスクの長さにより異なります。
6. SEQUENCE 関数は、loadseq 列に一意の番号を生成します。この関数は loadseq 列の現在の最大値を認識し、行を挿入するたびに増分値 (1) を追加して loadseq の値を求めます。
7. BEGINDATA は、制御情報の終わりとデータの始めを示します。
8. 物理レコードはいずれも 1 件の論理レコードと対応しますが、フィールドの長さがそれぞれ異なるため、レコードの長さも異なることがあります。また、いくつかの行の comm 列が NULL 値となることもあります。

## 事例 3 の実行

次に示す手順で、事例を実行します。

1. システム・プロンプトで次のように入力して、SQL\*Plus を scott/tiger で起動します。  

```
sqlplus scott/tiger
```

SQL プロンプトが表示されます。

- 2. SQL プロンプトで、この事例用の SQL スクリプトを次のように実行します。

```
SQL> @ulcase3
```

事例用の表が準備および移入されると、システム・プロンプトに戻ります。

- 3. システム・プロンプトで、次のように入力し、SQL\*Loader を起動して事例を実行します。

```
sqlldr USERID=scott/tiger CONTROL=ulcase3.ctl LOG=ulcase3.log
```

SQL\*Loader によって表にデータがロードされ、ログ・ファイルが作成されると、システム・プロンプトに戻ります。事例の実行結果を確認するには、ログ・ファイルを調べます。

### 事例 3 のログ・ファイル

次に、ログ・ファイルの一部を示します。

```
Control File:   ulcase3.ctl
Data File:     ulcase3.ctl
Bad File:      ulcase3.bad
Discard File:  none specified
```

```
(Allow all discards)
```

```
Number to load: ALL
Number to skip: 0
Errors allowed: 50
Bind array:     64 rows, maximum of 256000 bytes
Continuation:   none specified
Path used:      Conventional
```

```
Table EMP, loaded from every logical record.
Insert option in effect for this table: APPEND
```

Column Name	Position	Len	Term	Encl	Datatype
EMPNO	FIRST	*	,	O (")	CHARACTER
ENAME	NEXT	*	,	O (")	CHARACTER
JOB	NEXT	*	,	O (")	CHARACTER
MGR	NEXT	*	,	O (")	CHARACTER
HIREDATE	NEXT	20	,	O (")	DATE DD-Month-YYYY
SAL	NEXT	*	,	O (")	CHARACTER
COMM	NEXT	*	,	O (")	CHARACTER
DEPTNO	NEXT	*	:	O (")	CHARACTER

```
PROJNO                                NEXT      *      ,  O(") CHARACTER
LOADSEQ                                SEQUENCE (MAX, 1)
```

Table EMP:

```
7 Rows successfully loaded.
0 Rows not loaded due to data errors.
0 Rows not loaded because all WHEN clauses were failed.
0 Rows not loaded because all fields were null.
```

Space allocated for bind array: 134976 bytes (64 rows)

Read buffer bytes: 1048576

```
Total logical records skipped:      0
Total logical records read:          7
Total logical records rejected:      0
Total logical records discarded:     0
```

Run began on Wed Feb 27 14:25:29 2002

Run ended on Wed Feb 27 14:25:30 2002

Elapsed time was: 00:00:00.81

CPU time was: 00:00:00.15

## 事例 4: 結合された物理レコードのロード

ここでは、次のことを説明します。

- CONTINUEIF を使用して、複数の物理レコードを結合し 1 つの論理レコードを作成します。5-27 ページの「[CONTINUEIF を使用した論理レコードの作成](#)」を参照してください。
- 負数を挿入します。
- REPLACE を指定して、新しいデータが挿入される前に表を空にします。5-31 ページの「[表固有のロード方法](#)」を参照してください。
- DISCARDFILE を使用して、制御ファイル内で廃棄ファイルを指定します。5-14 ページの「[廃棄ファイルの指定](#)」を参照してください。
- DISCARDMAX を使用して、廃棄レコードの最大数を指定します。5-14 ページの「[廃棄ファイルの指定](#)」を参照してください。
- 一意の索引での値の重複、または無効なデータ値を防ぐため、レコードを拒否します。5-13 ページの「[拒否レコードの条件](#)」を参照してください。

## 事例 4 の制御ファイル

制御ファイルは ulcase4.ctl です。

```
LOAD DATA
INFILE 'ulcase4.dat'
1) DISCARDFILE 'ulcase4.dsc'
2) DISCARDMAX 999
3) REPLACE
4) CONTINUEIF THIS (1) = '*'
INTO TABLE emp
(empno      POSITION(1:4)      INTEGER EXTERNAL,
ename      POSITION(6:15)     CHAR,
job         POSITION(17:25)    CHAR,
mgr         POSITION(27:30)    INTEGER EXTERNAL,
sal         POSITION(32:39)    DECIMAL EXTERNAL,
comm        POSITION(41:48)    DECIMAL EXTERNAL,
deptno      POSITION(50:51)    INTEGER EXTERNAL,
hiredate    POSITION(52:60)    INTEGER EXTERNAL)
```

**注意：**

- 1. DISCARDFILE は、ulcase4.dsc という名前の廃棄ファイルを指定します。
- 2. DISCARDMAX は、実行終了までに処理できる廃棄レコードの最大数を、999 に設定します（実際は、すべてのレコードを廃棄できます）。
- 3. REPLACE は、データのロード先の表にすでにデータが含まれている場合、新しいデータのロード前に SQL\*Loader でそのデータを削除することを指定します。
- 4. CONTINUEIF THIS は、現在のレコードの 1 列目にアスタリスクがある場合に、その次の物理レコードがこの現在のレコードに追加され、論理レコードが形成されることを指定します。そのため、各物理レコードの 1 列目には、アスタリスクまたはデータ以外の値が必要となります。

## 事例 4 のデータ・ファイル

次に、この事例のデータ・ファイル ulcase4.dat を示します。1 列目にアスタリスクがあります。表示されていませんが、20 列目には改行文字があります。また、clark のコミッションは -10 で、この値は SQL\*Loader によって負数に変換されてからロードされます。

```
*7782 CLARK
MANAGER      7839 2572.50   -10   25 12-NOV-85
*7839 KING
PRESIDENT    5500.00           25 05-APR-83
*7934 MILLER
CLERK        7782 920.00           25 08-MAY-80
*7566 JONES
MANAGER      7839 3123.75           25 17-JUL-85
```



```
*7499 ALLEN
SALESMAN 7698 1600.00 300.00 25 3-JUN-84
*7654 MARTIN
SALESMAN 7698 1312.50 1400.00 25 21-DEC-85
*7658 CHAN
ANALYST 7566 3450.00 25 16-FEB-84
*      CHEN
ANALYST 7566 3450.00 25 16-FEB-84
*7658 CHIN
ANALYST 7566 3450.00 25 16-FEB-84
```

## 拒否レコード

最後の 2 件のレコードは、次の場合に拒否されます。empno 列に一意の索引が作成されている場合は、chin と chan の empno が等しいため、chin のレコードは受付けを拒否されます。empno 列が NOT NULL と定義されている場合は、empno に値が存在しないため chen のレコードは拒否されます。

## 事例 4 の実行

次に示す手順で、事例を実行します。

1. システム・プロンプトで次のように入力して、SQL\*Plus を scott/tiger で起動します。

```
sqlplus scott/tiger
```

SQL プロンプトが表示されます。

2. SQL プロンプトで、この事例用の SQL スクリプトを次のように実行します。

```
SQL> @ulcase4
```

事例用の表が準備および移入されると、システム・プロンプトに戻ります。

3. システム・プロンプトで、次のように入力し、SQL\*Loader を起動して事例を実行します。

```
sqlldr USERID=scott/tiger CONTROL=ulcase4.ctl LOG=ulcase4.log
```

SQL\*Loader によって表にデータがロードされ、ログ・ファイルが作成されると、システム・プロンプトに戻ります。事例の実行結果を確認するには、ログ・ファイルを調べます。

事例 4 のログ・ファイル

次に、ログ・ファイルの一部を示します。

Control File:   ulcase4.ctl  
Data File:       ulcase4.dat  
  Bad File:       ulcase4.bad  
  Discard File: ulcase4.dis  
  (Allow 999 discards)

Number to load: ALL  
Number to skip: 0  
Errors allowed: 50  
Bind array:       64 rows, maximum of 256000 bytes  
Continuation:    1:1 = 0X2a(character '\*'), in current physical record  
Path used:        Conventional

Table EMP, loaded from every logical record.  
Insert option in effect for this table: REPLACE

Column Name	Position	Len	Term Encl	Datatype
EMPNO	1:4	4		CHARACTER
ENAME	6:15	10		CHARACTER
JOB	17:25	9		CHARACTER
MGR	27:30	4		CHARACTER
SAL	32:39	8		CHARACTER
COMM	41:48	8		CHARACTER
DEPTNO	50:51	2		CHARACTER
HIREDATE	52:60	9		CHARACTER

Record 8: Rejected - Error on table EMP.  
ORA-01400: ("SCOTT"."EMP"."EMPNO") には NULL は挿入できません。

Record 9: Rejected - Error on table EMP.  
ORA-00001: 一意制約 (SCOTT.EMPPIX) に反しています

Table EMP:  
  7 Rows successfully loaded.  
  2 Rows not loaded due to data errors.  
  0 Rows not loaded because all WHEN clauses were failed.  
  0 Rows not loaded because all fields were null.

Space allocated for bind array:                   4608 bytes (64 rows)  
Read   buffer bytes: 1048576

```

Total logical records skipped:      0
Total logical records read:         9
Total logical records rejected:     2
Total logical records discarded:    0

```

```

Run began on Wed Feb 27 14:28:53 2002
Run ended on Wed Feb 27 14:28:54 2002

```

```

Elapsed time was:      00:00:00.91
CPU time was:         00:00:00.13

```

## 事例 4 の不良ファイル

前述の理由のため、次に示すように、8 レコード目と 9 レコード目が不良ファイルに入っています（廃棄ファイルは作成されません）。

```

*      CHEN      ANALYST
      7566      3450.00      25 16-FEB-84
*7658 CHIN      ANALYST
      7566      3450.00      25 16-FEB-84

```

## 事例 5: 複数表へのデータのロード

ここでは、次のことを説明します。

- 複数の表へデータをロードします。詳細は、5-42 ページの「[複数表へのデータのロード](#)」を参照してください。
- SQL\*Loader を使用して、フラット・ファイル内の繰返しグループを分析し、そのデータを正規化した表にロードします。このとき、ファイルの 1 件のレコードからデータベースの行が複数生成される場合があります。
- 1 件の物理レコードからの複数の論理レコードを作成します。詳細は、5-38 ページの「[複数の INTO TABLE 句を使用するメリット](#)」を参照してください。
- WHEN 句を使用します。詳細は、5-34 ページの「[条件に基づいたレコードのロード](#)」を参照してください。
- 同一フィールド (empno) を複数の表へロードします。

## 事例 5 の制御ファイル

制御ファイルは ulcase5.ctl です。

```
-- Loads EMP records from first 23 characters
-- Creates and loads PROJ records for each PROJNO listed
-- for each employee
LOAD DATA
INFILE 'ulcase5.dat'
BADFILE 'ulcase5.bad'
DISCARDFILE 'ulcase5.dsc'
1)  REPLACE
2)  INTO TABLE emp
    (empno  POSITION(1:4)      INTEGER EXTERNAL,
     ename   POSITION(6:15)    CHAR,
     deptno  POSITION(17:18)   CHAR,
     mgr     POSITION(20:23)   INTEGER EXTERNAL)
2)  INTO TABLE proj
    -- PROJ has two columns, both not null: EMPNO and PROJNO
3)  WHEN projno != ' '
    (empno  POSITION(1:4)      INTEGER EXTERNAL,
3)  projno  POSITION(25:27)    INTEGER EXTERNAL)  -- 1st proj
2)  INTO TABLE proj
4)  WHEN projno != ' '
    (empno  POSITION(1:4)      INTEGER EXTERNAL,
4)  projno  POSITION(29:31)    INTEGER EXTERNAL)  -- 2nd proj

2)  INTO TABLE proj
5)  WHEN projno != ' '
    (empno  POSITION(1:4)      INTEGER EXTERNAL,
5)  projno  POSITION(33:35)    INTEGER EXTERNAL)  -- 3rd proj
```

### 注意：

1. REPLACE は、データのロード先の表（emp 表および proj 表）にすでにデータが含まれている場合に、新しい行のロード前に SQL\*Loader でそのデータを削除することを指定します。
2. INTO TABLE 句を複数指定して、2 つの表（emp および proj）にデータをロードします。proj 表のロードには、同じ一連のレコードが毎回異なる列の組合せで 3 回処理されます。
3. WHEN 句は、プロジェクト番号が空白ではない行のみをロードします。projno に 25 ～ 27 列と定義してあると、これらの列に値が存在する場合にのみ、行が proj に挿入されます。
4. projno に 29 ～ 31 列と定義してあると、これらの列に値が存在する場合にのみ、行が proj に挿入されます。

5. projno に 33 ～ 35 列と定義してあると、これらの列に値が存在する場合にのみ、行が proj に挿入されます。

## 事例 5 のデータ・ファイル

1234	BAKER	10	9999	101	102	103
1234	JOKER	10	9999	777	888	999
2664	YOUNG	20	2893	425	abc	102
5321	OTOOLE	10	9999	321	55	40
2134	FARMER	20	4555	236	456	
2414	LITTLE	20	5634	236	456	40
6542	LEE	10	4532	102	321	14
2849	EDDS	xx	4555		294	40
4532	PERKINS	10	9999	40		
1244	HUNT	11	3452	665	133	456
123	DOOLITTLE	12	9940			132
1453	MACDONALD	25	5532		200	

## 事例 5 の実行

次に示す手順で、事例を実行します。

1. システム・プロンプトで次のように入力して、SQL\*Plus を scott/tiger で起動します。

```
sqlplus scott/tiger
```

SQL プロンプトが表示されます。

2. SQL プロンプトで、この事例用の SQL スクリプトを次のように実行します。

```
SQL> @ulcase5
```

事例用の表が準備および移入されると、システム・プロンプトに戻ります。

3. システム・プロンプトで、次のように入力し、SQL\*Loader を起動して事例を実行します。

```
sqlldr USERID=scott/tiger CONTROL=ulcase5.ctl LOG=ulcase5.log
```

SQL\*Loader によって表にデータがロードされ、ログ・ファイルが作成されると、システム・プロンプトに戻ります。事例の実行結果を確認するには、ログ・ファイルを調べます。

## 事例 5 のログ・ファイル

次に、ログ・ファイルの一部を示します。

```
Control File:  ulcase5.ctl
Data File:     ulcase5.dat
  Bad File:    ulcase5.bad
  Discard File: ulcase5.dis
(Allow all discards)

Number to load: ALL
Number to skip: 0
Errors allowed: 50
Bind array:    64 rows, maximum of 256000 bytes
Continuation:  none specified
Path used:     Conventional
```

Table EMP, loaded from every logical record.  
Insert option in effect for this table: REPLACE

Column Name	Position	Len	Term	Encl	Datatype
EMPNO	1:4	4			CHARACTER
ENAME	6:15	10			CHARACTER
DEPTNO	17:18	2			CHARACTER
MGR	20:23	4			CHARACTER

Table PROJ, loaded when PROJNO != 0X202020(character ' ' )  
Insert option in effect for this table: REPLACE

Column Name	Position	Len	Term	Encl	Datatype
EMPNO	1:4	4			CHARACTER
PROJNO	25:27	3			CHARACTER

Table PROJ, loaded when PROJNO != 0X202020(character ' ' )  
Insert option in effect for this table: REPLACE

Column Name	Position	Len	Term	Encl	Datatype
EMPNO	1:4	4			CHARACTER
PROJNO	29:31	3			CHARACTER

Table PROJ, loaded when PROJNO != 0X202020(character ' ' )  
Insert option in effect for this table: REPLACE

Column Name	Position	Len	Term	Encl	Datatype

EMPNO	1:4	4	CHARACTER
PROJNO	33:35	3	CHARACTER

1) Record 2: Rejected - Error on table EMP.  
1)ORA-00001: 一意制約 (SCOTT.EMPX) に反しています

1) Record 8: Rejected - Error on table EMP, column DEPTNO.  
1)ORA-01722: 数値が無効です。

1) Record 3: Rejected - Error on table PROJ, column PROJNO.  
1)ORA-01722: 数値が無効です。

Table EMP:

2) 9 Rows successfully loaded.  
2) 3 Rows not loaded due to data errors.  
2) 0 Rows not loaded because all WHEN clauses were failed.  
2) 0 Rows not loaded because all fields were null.

Table PROJ:

3) 7 Rows successfully loaded.  
3) 2 Rows not loaded due to data errors.  
3) 3 Rows not loaded because all WHEN clauses were failed.  
3) 0 Rows not loaded because all fields were null.

Table PROJ:

4) 7 Rows successfully loaded.  
4) 3 Rows not loaded due to data errors.  
4) 2 Rows not loaded because all WHEN clauses were failed.  
4) 0 Rows not loaded because all fields were null.

Table PROJ:

5) 6 Rows successfully loaded.  
5) 3 Rows not loaded due to data errors.  
5) 3 Rows not loaded because all WHEN clauses were failed.  
5) 0 Rows not loaded because all fields were null.

Space allocated for bind array: 4096 bytes (64 rows)  
Read buffer bytes: 1048576

Total logical records skipped: 0  
Total logical records read: 12  
Total logical records rejected: 3

```
Total logical records discarded:      0

Run began on Wed Feb 27 14:34:33 2002
Run ended on Wed Feb 27 14:34:34 2002

Elapsed time was:      00:00:01.00
CPU time was:      00:00:00.22
```

**注意：**

- 1. バッファ方式（配列バッチ方式）を採用しているため、物理レコードの順序ではエラーは検出されません。不良ファイルおよび廃棄ファイルには、ログ・ファイルと同じ順序でレコードが書き込まれます。
- 2. 入力された総計 12 件の論理レコードのうち、3 つの行（joker、young、edds）は拒否されました。拒否レコード中のデータはロードされません。
- 3. 全レコードのうち 9 件は WHEN 句の条件を満たし、2 件（joker および young）はデータ・エラーのため拒否されました。
- 4. 全レコードのうち 10 件は WHEN 句の条件を満たし、3 件（joker、young および edds）はデータ・エラーのため拒否されました。
- 5. 全レコードのうち 9 件は WHEN 句の条件を満たし、3 件（joker、young および edds）はデータ・エラーのため拒否されました。

事例 5 のロードされた表

SQL 問合せおよびその結果の例を次に示します。

```
SQL> SELECT empno, ename, mgr, deptno FROM emp;
EMPNO      ENAME      MGR      DEPTNO
-----
1234      BAKER      9999      10
5321      OTOOLE      9999      10
2134      FARMER      4555      20
2414      LITTLE      5634      20
6542      LEE      4532      10
4532      PERKINS      9999      10
1244      HUNT      3452      11
123      DOOLITTLE      9940      12
1453      MACDONALD      5532      25
```

```
SQL> SELECT * from PROJ order by EMPNO;

EMPNO      PROJNO
-----
123      132
1234      101
```



1234	103
1234	102
1244	665
1244	456
1244	133
1453	200
2134	236
2134	456
2414	236
2414	456
2414	40
4532	40
5321	321
5321	40
5321	55
6542	102
6542	14
6542	321

## 事例 6: ダイレクト・パス・ロード方式を使用したデータのロード

この事例では、ダイレクト・パス・ロード方式を使用して emp 表にデータをロードしながら、同時にすべての索引も構築します。機能は次のとおりです。

- ロードおよび索引データ作成のために、ダイレクト・パス・ロード方式を使用します。詳細は、[第 9 章](#)を参照してください。
- データを事前ソートする索引を指定します。詳細は、9-17 ページの「[高速索引付けのためのデータの事前ソート](#)」を参照してください。
- NULLIF 句。詳細は、6-32 ページの「[WHEN、NULLIF および DEFAULTIF 句の使用](#)」を参照してください。
- すべてが空白の数値フィールドを NULL としてロードします。詳細は、6-40 ページの「[すべてが空白のフィールドのロード](#)」を参照してください。

この事例では、フィールド位置とデータ型を明示的に指定します。

## 事例 6 の制御ファイル

制御ファイルは ulcase6.ctl です。

```
LOAD DATA
INFILE 'ulcase6.dat'
REPLACE
INTO TABLE emp
1)   SORTED INDEXES (empix)
2)   (empno POSITION(01:04) INTEGER EXTERNAL NULLIF empno=BLANKS,
      ename  POSITION(06:15) CHAR,
      job    POSITION(17:25) CHAR,
      mgr     POSITION(27:30) INTEGER EXTERNAL NULLIF mgr=BLANKS,
      sal     POSITION(32:39) DECIMAL EXTERNAL NULLIF sal=BLANKS,
      comm    POSITION(41:48) DECIMAL EXTERNAL NULLIF comm=BLANKS,
      deptno  POSITION(50:51) INTEGER EXTERNAL NULLIF deptno=BLANKS)
```

**注意：**

- 1. SORTED INDEXES 文で、データをソートしている索引を指定します。この文を指定することで、索引 empix の列によってデータ・ファイルがあらかじめソートされていることを示します。SORTED INDEXES によって、ダイレクト・パス・ロード方法の使用時に SQL\*Loader でこのデータに対するソートが処理されなくなるため、索引作成を最適化できます。
- 2. NULLIF...BLANKS 句は、データ・ファイル中のフィールドの値がすべて空白の場合は、列が NULL としてロードされるように指定します。詳細は、6-32 ページの「WHEN、NULLIF および DEFAULTIF 句の使用」を参照してください。

## 事例 6 のデータ・ファイル

7499	ALLEN	SALESMAN	7698	1600.00	300.00	30
7566	JONES	MANAGER	7839	3123.75		20
7654	MARTIN	SALESMAN	7698	1312.50	1400.00	30
7658	CHAN	ANALYST	7566	3450.00		20
7782	CLARK	MANAGER	7839	2572.50		10
7839	KING	PRESIDENT		5500.00		10
7934	MILLER	CLERK	7782	920.00		10

## 事例 6 の実行

次に示す手順で、事例を実行します。

1. システム・プロンプトで次のように入力して、SQL\*Plus を scott/tiger で起動します。

```
sqlplus scott/tiger
```

SQL プロンプトが表示されます。

2. SQL プロンプトで、この事例用の SQL スクリプトを次のように実行します。

```
SQL> @ulcase6
```

事例用の表が準備および移入されると、システム・プロンプトに戻ります。

3. システム・プロンプトで、次のように入力し、SQL\*Loader を起動して事例を実行します。必ず DIRECT=true を指定してください。指定しない場合、従来型パスがデフォルトで使用され、事例は正常に実行されません。

```
sqlldr USERID=scott/tiger CONTROL=ulcase6.ctl LOG=ulcase6.log DIRECT=true
```

SQL\*Loader は、ダイレクト・パス・ロード方式を使用して emp 表をロードし、ログ・ファイルを作成すると、システム・プロンプトに戻ります。事例の実行結果を確認するには、ログ・ファイルを調べます。

## 事例 6 のログ・ファイル

次に、ログ・ファイルの一部を示します。

```
Control File:   ulcase6.ctl
Data File:     ulcase6.dat
Bad File:      ulcase6.bad
Discard File:  none specified
```

```
(Allow all discards)
```

```
Number to load: ALL
Number to skip: 0
Errors allowed: 50
Continuation:   none specified
Path used:      Direct
```

```
Table EMP, loaded from every logical record.
Insert option in effect for this table: REPLACE
```

Column Name	Position	Len	Term	Encl	Datatype
-----					

## 事例 6: ダイレクト・パス・ロード方式を使用したデータのロード

---

EMPNO	1:4	4	CHARACTER
ENAME	6:15	10	CHARACTER
JOB	17:25	9	CHARACTER
MGR	27:30	4	CHARACTER
NULL if MGR = BLANKS			
SAL	32:39	8	CHARACTER
NULL if SAL = BLANKS			
COMM	41:48	8	CHARACTER
NULL if COMM = BLANKS			
DEPTNO	50:51	2	CHARACTER
NULL if EMPNO = BLANKS			

The following index(es) on table EMP were processed:  
index SCOTT.EMPPIX loaded successfully with 7 keys

Table EMP:

7 Rows successfully loaded.  
0 Rows not loaded due to data errors.  
0 Rows not loaded because all WHEN clauses were failed.  
0 Rows not loaded because all fields were null.

Bind array size not used in direct path.

Column array rows : 5000

Stream buffer bytes: 256000

Read buffer bytes: 1048576

Total logical records skipped:	0
Total logical records read:	7
Total logical records rejected:	0
Total logical records discarded:	0
Total stream buffers loaded by SQL*Loader main thread:	2
Total stream buffers loaded by SQL*Loader load thread:	0

Run began on Wed Feb 27 13:21:29 2002

Run ended on Wed Feb 27 13:21:32 2002

Elapsed time was: 00:00:02.96

CPU time was: 00:00:00.22

## 事例 7: 書式化されたレポートからのデータの抽出

この事例では、SQL\*Loader の文字列処理機能が書式化されたレポートからデータを抽出します。この事例では、未定義のフィールドの最後の値を使用するトリガーを作成します。ここでは、次のことを説明します。

- SQL\*Loader の INSERT トリガーを使用します。データベース・トリガーの詳細は、『Oracle9i アプリケーション開発者ガイド - 基礎編』を参照してください。
- データの処理のために SQL 文字列を使用します。6-48 ページの「[フィールドへの SQL 演算子の適用](#)」を参照してください。
- 最初と最後で異なるデリミタを使用します。詳細は、6-24 ページの「[デリミタの指定](#)」を参照してください。
- SYSDATE を使用します。6-54 ページの「[列への現在の日付の設定](#)」を参照してください。
- TRAILING NULLCOLS 句を使用します。5-37 ページの「[TRAILING NULLCOLS 句](#)」を参照してください。
- あいまいなフィールド長に対する警告を発行します。6-21 ページの「[システム固有のデータ型フィールド長の競合](#)」および 6-27 ページの「[文字データ型フィールド長の競合](#)」を参照してください。
- 廃棄ファイルを使用します。詳細は、5-14 ページの「[制御ファイルでの廃棄ファイルの指定](#)」を参照してください。

### BEFORE INSERT トリガーの作成

この事例では、データ行に部門番号、ジョブ名およびマネージャ番号のフィールドがない場合、それらを書き込むための BEFORE INSERT トリガーが必要です。値が存在する場合は、グローバル変数に保存されます。値が存在しない場合は、そのグローバル変数が使用されます。

ulcase7s.sql スクリプトを実行すると、INSERT トリガーおよびグローバル変数パッケージが作成されます。

次に、グローバル変数で定義されるパッケージを示します。

```
CREATE OR REPLACE PACKAGE uldemo7 AS    -- Global Package Variables
    last_deptno    NUMBER(2);
    last_job       VARCHAR2(9);
    last_mgr       NUMBER(4);
END uldemo7;
/
```

次に、INSERT トリガーの定義を示します。

```
CREATE OR REPLACE TRIGGER uldemo7_emp_insert
```

```
BEFORE INSERT ON emp
FOR EACH ROW
BEGIN
  IF :new.deptno IS NOT NULL THEN
    uldemo7.last_deptno := :new.deptno; -- save value for later
  ELSE
    :new.deptno := uldemo7.last_deptno; -- use last valid value
  END IF;
  IF :new.job IS NOT NULL THEN
    uldemo7.last_job := :new.job;
  ELSE
    :new.job := uldemo7.last_job;
  END IF;
  IF :new.mgr IS NOT NULL THEN
    uldemo7.last_mgr := :new.mgr;
  ELSE
    :new.mgr := uldemo7.last_mgr;
  END IF;
END;
/
```

---

---

**注意：** FOR EACH ROW 句は重要です。SQL\*Loader は配列インタフェースを使用するため、この句を指定しなかった場合、INSERT トリガーは挿入配列ごとに 1 回のみ実行します。

---

---

残りの事例の実行を続ける前に、必ず `ulcase7e.sql` スクリプトを実行して、INSERT トリガーおよびグローバル変数パッケージを削除してください。詳細は、10-32 ページの「[事例 7 の実行](#)」を参照してください。

## 事例 7 の制御ファイル

制御ファイルは `ulcase7.ct1` です。

```
LOAD DATA
  INFILE 'ulcase7.dat'
  DISCARDFILE 'ulcase7.dis'
  APPEND
  INTO TABLE emp
1)   WHEN (57) = '.'
2)   TRAILING NULLCOLS
3)   (hiredate SYSDATE,
4)     deptno POSITION(1:2)  INTEGER EXTERNAL(3)
5)     NULLIF deptno=BLANKS,
      job   POSITION(7:14)  CHAR   TERMINATED BY WHITESPACE
6)     NULLIF job=BLANKS   "UPPER(:job)",
```

```

7)   mgr      POSITION(28:31) INTEGER EXTERNAL
      TERMINATED BY WHITESPACE, NULLIF mgr=BLANKS,
      ename    POSITION(34:41) CHAR
      TERMINATED BY WHITESPACE  "UPPER(:ename)",
      empno    POSITION(45) INTEGER EXTERNAL
      TERMINATED BY WHITESPACE,
      sal      POSITION(51) CHAR  TERMINATED BY WHITESPACE
8)   "TO_NUMBER(:sal, '$99,999.99')",
9)   comm     INTEGER EXTERNAL  ENCLOSED BY '(' AND '%'
      ":comm * 100"
)

```

**注意：**

1. 列 57（給与フィールド）の小数点は、データが入っている行を識別します。レポート内の他の行はすべて廃棄されます。
2. TRAILING NULLCOLS 句を指定すると、SQL\*Loader ではレコードの最後で欠落しているフィールドが NULL として扱われます。すべてのレコードにコミッション・フィールドが存在するわけではありません。そのため、8 フィールド検出されるはずの間合せに対して 7 フィールドのみが検出された場合でも、この句はレコードを拒否しないで NULL コミッションをロードします。
3. 従業員の雇用日が、現在のシステム日付を使用して記入されます。
4. この指定は、指定した長さがフィールド位置で決定された長さとは一致していない場合に警告メッセージを生成します。指定した長さ（3）が使用されます。詳細は、10-33 ページの「[事例 7 のログ・ファイル](#)」を参照してください。デフォルトのバイト長セマンティクスによって、長さはバイト単位になります。文字長セマンティクスが使用された場合、長さは文字単位になります。
5. レポートは値が変更されたときにのみ部門番号、ジョブおよびマネージャを表示するため、これらのフィールドは、空白になる可能性があります。この制御ファイルでは、それらを NULL としてロードして、挿入トリガーで最新の有効な値を書き込みます。
6. SQL 文字列はジョブ名を大文字に変更します。
7. ここで開始位置を指定する必要があります。ジョブ・フィールドおよびマネージャ・フィールドが両方とも空白の場合、ジョブ・フィールドの TERMINATED BY WHITESPACE 句によって、SQL\*Loader は従業員名フィールドに進みます。POSITION 句がない場合、従業員名フィールドは誤ってマネージャ・フィールドと解釈されます。
8. SQL 文字列はフィールドを書式化文字列から数値に変換します。数値は領域が少なくて済むため、様々な書式化オプションで印刷できます。
9. この事例では、最初と最後の異なるデリミタが、書式化フィールドから数値を選択します。その後、SQL 文字列は格納形式に値を変換します。

## 事例 7 のデータ・ファイル

次に、ロードされるデータを示します。

Today's Newly Hired Employees						
Dept	Job	Manager	MgrNo	Emp Name	EmpNo	Salary/Commission
----	-----	-----	-----	-----	-----	-----
20	Salesman	Blake	7698	Shepard	8061	\$1,600.00 (3%)
				Falstaff	8066	\$1,250.00 (5%)
				Major	8064	\$1,250.00 (14%)
30	Clerk	Scott	7788	Conrad	8062	\$1,100.00
		Ford	7369	DeSilva	8063	\$800.00
	Manager	King	7839	Provo	8065	\$2,975.00

## 事例 7 の実行

次に示す手順で、事例を実行します。

1. システム・プロンプトで次のように入力して、SQL\*Plus を scott/tiger で起動します。

```
sqlplus scott/tiger
```

SQL プロンプトが表示されます。

2. SQL プロンプトで、この事例用の SQL スクリプトを次のように実行します。

```
SQL> @ulcase7s
```

事例用の表が準備および移入されると、システム・プロンプトに戻ります。

3. システム・プロンプトで、次のように入力し、SQL\*Loader を起動して事例を実行します。

```
sqlldr USERID=scott/tiger CONTROL=ulcase7.ctl LOG=ulcase7.log
```

SQL\*Loader によってレポートからデータが抽出され、ログ・ファイルが作成されると、システム・プロンプトに戻ります。事例の実行結果を確認するには、ログ・ファイルを調べます。

4. この事例の実行後、残りの事例の実行を続ける前に、挿入トリガーおよびグローバル変数パッケージを削除する必要があります。削除するには、ulcase7.sql スクリプトを次のように実行します。

```
SQL> @ulcase7e
```



## 事例 7 のログ・ファイル

次に、ログ・ファイルの一部を示します。

```
1) SQL*Loader-307: Warning: conflicting lengths 2 and 3 specified for column
DEPTNO
table EMP
Control File: ulcase7.ctl
Data File:      ulcase7.dat
Bad File:       ulcase7.bad
Discard File:   ulcase7.dis
(Allow all discards)
```

```
Number to load: ALL
Number to skip: 0
Errors allowed: 50
Bind array:     64 rows, maximum of 256000 bytes
Continuation:   none specified
Path used:      Conventional
```

```
Table EMP, loaded when 57:57 = 0X2e(character '.')
Insert option in effect for this table: APPEND
TRAILING NULLCOLS option in effect
```

Column Name	Position	Len	Term	Encl	Datatype
HIREDATE					SYSDATE
DEPTNO	1:2	3			CHARACTER
NULL if DEPTNO = BLANKS					
JOB	7:14	8	WHT		CHARACTER
NULL if JOB = BLANKS					
SQL string for column : "UPPER(:job)"					
MGR	28:31	4	WHT		CHARACTER
NULL if MGR = BLANKS					
ENAME	34:41	8	WHT		CHARACTER
SQL string for column : "UPPER(:ename)"					
EMPNO	NEXT	*	WHT		CHARACTER
SAL	51	*	WHT		CHARACTER
SQL string for column : "TO_NUMBER(:sal, '\$99,999.99')"					
COMM	NEXT	*		(	CHARACTER
				%	
SQL string for column : ":comm * 100"					

```
2) Record 1: Discarded - failed all WHEN clauses.
Record 2: Discarded - failed all WHEN clauses.
Record 3: Discarded - failed all WHEN clauses.
Record 4: Discarded - failed all WHEN clauses.
Record 5: Discarded - failed all WHEN clauses.
```

## 事例 8: パーティション化された表のロード

---

```
Record 6: Discarded - failed all WHEN clauses.
Record 10: Discarded - failed all WHEN clauses.

Table EMP:
  6 Rows successfully loaded.
  0 Rows not loaded due to data errors.
2) 7 Rows not loaded because all WHEN clauses were failed.
  0 Rows not loaded because all fields were null.

Space allocated for bind array:          51584 bytes (64 rows)
Read  buffer bytes: 1048576

Total logical records skipped:          0
Total logical records read:             13
Total logical records rejected:         0
2) Total logical records discarded:      7

Run began on Wed Feb 27 14:54:03 2002
Run ended on Wed Feb 27 14:54:04 2002

Elapsed time was:      00:00:00.99
CPU time was:         00:00:00.21
```

### 注意：

1. 指定した長さと、位置指定から導出された長さが違うため警告が生成されます。
2. レポートの先頭には 6 行のヘッダーがあります。3 行にテキストが含まれ、残りの 3 行は空白です。レポート内の空白のセパレータ行同様に、6 行すべてのヘッダーが拒否されます。

## 事例 8: パーティション化された表のロード

ここでは、次のことを説明します。

- データをパーティション化します。パーティション化データの概念については、『Oracle9i データベース概要』を参照してください。
- フィールド位置およびデータ型を明示的に定義します。
- 固定レコード長オプションを使用したロードを行います。詳細は、3-4 ページの「[入力データおよびデータ・ファイル](#)」を参照してください。

## 事例 8 の制御ファイル

制御ファイルは `ulcase8.ctl` です。出荷日ごとにデータをパーティション化した `lineitem` 表に対して、固定長レコードをロードします。

```
LOAD DATA
1) INFILE 'ulcase8.dat' "fix 129"
BADFILE 'ulcase8.bad'
TRUNCATE
INTO TABLE lineitem
PARTITION (ship_q1)
2) (l_orderkey      position (1:6) char,
    l_partkey       position (7:11) char,
    l_suppkey       position (12:15) char,
    l_linenumbers   position (16:16) char,
    l_quantity      position (17:18) char,
    l_extendedprice position (19:26) char,
    l_discount      position (27:29) char,
    l_tax           position (30:32) char,
    l_returnflag    position (33:33) char,
    l_linestatus    position (34:34) char,
    l_shipdate      position (35:43) char,
    l_commitdate    position (44:52) char,
    l_receiptdate   position (53:61) char,
    l_shipinstruct  position (62:78) char,
    l_shipmode      position (79:85) char,
    l_comment       position (86:128) char)
```

### 注意：

1. データ・ファイルの各レコードが固定長であることを示します（この例では 129 文字です）。
2. 各列にロードするデータ・ファイルの列名およびデータの位置を示します。

## 表の作成

データをパーティション化するには、出荷日ごとに 4 つのパーティションを使用して、`lineitem` 表を作成します。

```
create table lineitem
(l_orderkey      number,
l_partkey       number,
l_suppkey       number,
l_linenumbers   number,
l_quantity      number,
l_extendedprice number,
l_discount      number,
```

```

l_tax          number,
l_returnflag   char,
l_linestatus   char,
l_shipdate     date,
l_commitdate   date,
l_receiptdate  date,
l_shipinstruct char(17),
l_shipmode     char(7),
l_comment      char(43))
partition by range (l_shipdate)
(
partition ship_q1 values less than (TO_DATE('01-APR-1996', 'DD-MON-YYYY'))
tablespace p01,
partition ship_q2 values less than (TO_DATE('01-JUL-1996', 'DD-MON-YYYY'))
tablespace p02,
partition ship_q3 values less than (TO_DATE('01-OCT-1996', 'DD-MON-YYYY'))
tablespace p03,
partition ship_q4 values less than (TO_DATE('01-JAN-1997', 'DD-MON-YYYY'))
tablespace p04
)

```

## 事例 8 のデータ・ファイル

次に、この事例のデータ・ファイル `ulcase8.dat` を示します。各レコード長は 128 文字です。ファイル内の各レコードの前には、空白が 5 つあります。

```

1 151978511724386.60 7.04.0NO09-SEP-6412-FEB-9622-MAR-96DELIVER IN PERSONTRUCK
iPBw4mMm7w7kQ zNPL i261OPP
1 2731 73223658958.28.09.06NO12-FEB-9628-FEB-9620-APR-96TAKE BACK RETURN MAIL
5wM04SNyl0AnghCP2nx lAi
1 3370 3713 810210.96 .1.02NO29-MAR-9605-MAR-9631-JAN-96TAKE BACK RETURN REG
AIRSQC2C 5PNCy4mM
1 5214 46542831197.88.09.06NO21-APR-9630-MAR-9616-MAY-96NONE AIR
Om0L65CSAwSj5k6k
1 6564 6763246897.92.07.02NO30-MAY-9607-FEB-9603-FEB-96DELIVER IN PERSONMAIL
CB0SnyOL PQ32B70wB75k 6Aw10m0wh
1 7403 160524 31329.6 .1.04NO30-JUN-9614-MAR-9601 APR-96NONE FOB
C2gOQj OB6RLk1BS15 igN
2 8819 82012441659.44 0.08NO05-AUG-9609-FEB-9711-MAR-97COLLECT COD AIR
O52M70MRgRNnm476mNm
3 9451 721230 41113.5.05.01AF05-SEP-9629-DEC-9318-FEB-94TAKE BACK RETURN FOB
6wQn00Llg6y
3 9717 1834440788.44.07.03RF09-NOV-9623-DEC-9315-FEB-94TAKE BACK RETURN SHIP
LhiA7wygz0k4g4zRhMLBAM
3 9844 1955 6 8066.64.04.01RF28-DEC-9615-DEC-9314-FEB-94TAKE BACK RETURN REG
AIR6nmBmjQkgiCyzCQBkxPPOx5j4hB 0lRywgniP1297

```

## 事例 8 の実行

次に示す手順で、事例を実行します。

1. システム・プロンプトで次のように入力して、SQL\*Plus を scott/tiger で起動します。

```
sqlplus scott/tiger
```

SQL プロンプトが表示されます。

2. SQL プロンプトで、この事例用の SQL スクリプトを次のように実行します。

```
SQL> @ulcase8
```

事例用の表が準備および移入されると、システム・プロンプトに戻ります。

3. システム・プロンプトで、次のように入力し、SQL\*Loader を起動して事例を実行します。

```
sqlldr USERID=scott/tiger CONTROL=ulcase8.ctl LOG=ulcase8.log
```

SQL\*Loader によってデータをパーティション化およびロードされ、ログ・ファイルが作成されると、システム・プロンプトに戻ります。事例の実行結果を確認するには、ログ・ファイルを調べます。

## 事例 8 のログ・ファイル

次に、ログ・ファイルの一部を示します。

```
Control File:   ulcase8.ctl
Data File:     ulcase8.dat
File processing option string: "fix 129"
Bad File:      ulcase8.bad
Discard File:  none specified
```

```
(Allow all discards)
```

```
Number to load: ALL
Number to skip: 0
Errors allowed: 50
Bind array:     64 rows, maximum of 256000 bytes
Continuation:   none specified
Path used:      Conventional
```

```
Table LINEITEM, partition SHIP_Q1, loaded from every logical record.
Insert option in effect for this partition: TRUNCATE
```

Column Name	Position	Len	Term	Encl	Datatype
-------------	----------	-----	------	------	----------

L_ORDERKEY	1:6	6	CHARACTER
L_PARTKEY	7:11	5	CHARACTER
L_SUPPKEY	12:15	4	CHARACTER
L_LINENUMBER	16:16	1	CHARACTER
L_QUANTITY	17:18	2	CHARACTER
L_EXTENDEDPRICE	19:26	8	CHARACTER
L_DISCOUNT	27:29	3	CHARACTER
L_TAX	30:32	3	CHARACTER
L_RETURNFLAG	33:33	1	CHARACTER
L_LINESTATUS	34:34	1	CHARACTER
L_SHIPDATE	35:43	9	CHARACTER
L_COMMITDATE	44:52	9	CHARACTER
L_RECEIPTDATE	53:61	9	CHARACTER
L_SHIPINSTRUCT	62:78	17	CHARACTER
L_SHIPMODE	79:85	7	CHARACTER
L_COMMENT	86:128	43	CHARACTER

Record 4: Rejected - Error on table LINEITEM, partition SHIP\_Q1.  
ORA-14401: 挿入されたパーティション・キーが指定されたパーティションの範囲外です。

Record 5: Rejected - Error on table LINEITEM, partition SHIP\_Q1.  
ORA-14401: 挿入されたパーティション・キーが指定されたパーティションの範囲外です。

Record 6: Rejected - Error on table LINEITEM, partition SHIP\_Q1.  
ORA-14401: 挿入されたパーティション・キーが指定されたパーティションの範囲外です。

Record 7: Rejected - Error on table LINEITEM, partition SHIP\_Q1.  
ORA-14401: 挿入されたパーティション・キーが指定されたパーティションの範囲外です。

Record 8: Rejected - Error on table LINEITEM, partition SHIP\_Q1.  
ORA-14401: 挿入されたパーティション・キーが指定されたパーティションの範囲外です。

Record 9: Rejected - Error on table LINEITEM, partition SHIP\_Q1.  
ORA-14401: 挿入されたパーティション・キーが指定されたパーティションの範囲外です。

Record 10: Rejected - Error on table LINEITEM, partition SHIP\_Q1.  
ORA-14401: 挿入されたパーティション・キーが指定されたパーティションの範囲外です。

Table LINEITEM, partition SHIP\_Q1:  
3 Rows successfully loaded.  
7 Rows not loaded due to data errors.  
0 Rows not loaded because all WHEN clauses were failed.  
0 Rows not loaded because all fields were null.

```

Space allocated for bind array:          11008 bytes (64 rows)
Read  buffer bytes: 1048576

Total logical records skipped:          0
Total logical records read:             10
Total logical records rejected:         7
Total logical records discarded:        0

Run began on Wed Feb 27 15:02:28 2002
Run ended on Wed Feb 27 15:02:29 2002

Elapsed time was:      00:00:01.37
CPU time was:          00:00:00.20

```

## 事例 9: LOBFILE のロード (CLOB)

ここでは、次のことを説明します。

- resume と呼ばれる CLOB 列を emp 表に追加します。
- FILLER フィールド (res\_file) を使用します。
- 複数の LOBFILE を emp 表にロードします。

### 事例 9 の制御ファイル

制御ファイルは ulcase9.ctl です。各従業員の職務経歴とともに、新しいレコードを emp 表にロードします。それぞれの履歴は、別々のファイルに定義されています。

```

LOAD DATA
INFILE *
INTO TABLE emp
REPLACE
FIELDS TERMINATED BY ','
( empno    INTEGER EXTERNAL,
  ename     CHAR,
  job       CHAR,
  mgr       INTEGER EXTERNAL,
  sal       DECIMAL EXTERNAL,
  comm      DECIMAL EXTERNAL,
  deptno    INTEGER EXTERNAL,
1) res_file FILLER CHAR,
2) "RESUME" LOBFILE (res_file) TERMINATED BY EOF NULLIF res_file = 'NONE'
)
BEGINDATA
7782,CLARK,MANAGER,7839,2572.50,,10,ulcase91.dat
7839,KING,PRESIDENT,,5500.00,,10,ulcase92.dat

```

```
7934,MILLER,CLERK,7782,920.00,,10,ulcase93.dat
7566,JONES,MANAGER,7839,3123.75,,20,ulcase94.dat
7499,ALLEN,SALESMAN,7698,1600.00,300.00,30,ulcase95.dat
7654,MARTIN,SALESMAN,7698,1312.50,1400.00,30,ulcase96.dat
7658,CHAN,ANALYST,7566,3450.00,,20,NONE
```

### 注意：

1. FILLER フィールドです。FILLER フィールドがマップされているデータ・フィールドの値が割り当てられます。詳細は、6-6 ページの「[FILLER フィールドの指定](#)」を参照してください。
2. RESUME 列は、CLOB としてロードされます。LOBFILE の関数は、LOB フィールドのデータを含むファイルの名前を指定するフィールド名を指定します。詳細は、7-23 ページの「[LOBFILE からの LOB データのロード](#)」を参照してください。

## 事例 9 のデータ・ファイル

```
>>ulcase91.dat<<
```

Resume for Mary Clark

Career Objective: Manage a sales team with consistent record-breaking performance.

Education: BA Business University of Iowa 1992

Experience: 1992-1994 - Sales Support at MicroSales Inc.  
Won "Best Sales Support" award in 1993 and 1994  
1994-Present - Sales Manager at MicroSales Inc.  
Most sales in mid-South division for 2 years

```
>>ulcase92.dat<<
```

Resume for Monica King

Career Objective: President of large computer services company

Education: BA English Literature Bennington, 1985

Experience: 1985-1986 - Mailroom at New World Services  
1986-1987 - Secretary for sales management at  
New World Services  
1988-1989 - Sales support at New World Services  
1990-1992 - Salesman at New World Services  
1993-1994 - Sales Manager at New World Services  
1995 - Vice President of Sales and Marketing at  
New World Services  
1996-Present - President of New World Services

```
>>ulcase93.dat<<
```

Resume for Dan Miller



Career Objective: Work as a sales support specialist for a services  
company  
Education: Plainview High School, 1996  
Experience: 1996 - Present: Mail room clerk at New World Services

>>ulcase94.dat<<

Resume for Alyson Jones

Career Objective: Work in senior sales management for a vibrant and  
growing company  
Education: BA Philosophy Howard Univerity 1993  
Experience: 1993 - Sales Support for New World Services  
1994-1995 - Salesman for New World Services. Led in  
US sales in both 1994 and 1995.  
1996 - present - Sales Manager New World Services. My  
sales team has beat its quota by at least 15% each  
year.

>>ulcase95.dat<<

Resume for David Allen

Career Objective: Senior Sales man for aggressive Services company  
Education: BS Business Administration, Weber State 1994  
Experience: 1993-1994 - Sales Support New World Services  
1994-present - Salesman at New World Service. Won sales  
award for exceeding sales quota by over 20%  
in 1995, 1996.

>>ulcase96.dat<<

Resume for Tom Martin

Career Objective: Salesman for a computing service company  
Education: 1988 - BA Mathematics, University of the North  
Experience: 1988-1992 Sales Support, New World Services  
1993-present Salesman New World Services

## 事例 9 の実行

次に示す手順で、事例を実行します。

- 1. システム・プロンプトで次のように入力して、SQL\*Plus を scott/tiger で起動します。

```
sqlplus scott/tiger
```

SQL プロンプトが表示されます。

- 2. SQL プロンプトで、この事例用の SQL スクリプトを次のように実行します。

```
SQL> @ulcase9
```

事例用の表が準備および移入されると、システム・プロンプトに戻ります。

- 3. システム・プロンプトで、次のように入力し、SQL\*Loader を起動して事例を実行します。

```
sqlldr USERID=scott/tiger CONTROL=ulcase9.ctl LOG=ulcase9.log
```

SQL\*Loader によって emp 表にデータがロードされ、ログ・ファイルが作成されると、システム・プロンプトに戻ります。事例の実行結果を確認するには、ログ・ファイルを調べます。

## 事例 9 のログ・ファイル

次に、ログ・ファイルの一部を示します。

```
Control File:   ulcase9.ctl
Data File:      ulcase9.ctl
Bad File:       ulcase9.bad
Discard File:   none specified

(Allow all discards)

Number to load: ALL
Number to skip: 0
Errors allowed: 50
Bind array:     64 rows, maximum of 256000 bytes
Continuation:   none specified
Path used:      Conventional

Table EMP, loaded from every logical record.
Insert option in effect for this table: REPLACE

Column Name          Position  Len  Term Encl Datatype
-----
```

```

EMPNO                FIRST      *      ,      CHARACTER
ENAME                NEXT       *      ,      CHARACTER
JOB                   NEXT       *      ,      CHARACTER
MGR                   NEXT       *      ,      CHARACTER
SAL                   NEXT       *      ,      CHARACTER
COMM                  NEXT       *      ,      CHARACTER
DEPTNO                NEXT       *      ,      CHARACTER
RES_FILE              NEXT       *      ,      CHARACTER
(FILLER FIELD)
"RESUME"              DERIVED    *      EOF      CHARACTER
      Dynamic LOBFILE.  Filename in field RES_FILE
      NULL if RES_FILE = 0X4e4f4e45(character 'NONE')

```

Table EMP:

```

7 Rows successfully loaded.
0 Rows not loaded due to data errors.
0 Rows not loaded because all WHEN clauses were failed.
0 Rows not loaded because all fields were null.

```

```

Space allocated for bind array:          132096 bytes (64 rows)
Read   buffer bytes: 1048576

```

```

Total logical records skipped:           0
Total logical records read:              7
Total logical records rejected:          0
Total logical records discarded:         0

```

```

Run began on Wed Feb 27 15:06:49 2002
Run ended on Wed Feb 27 15:06:50 2002

```

```

Elapsed time was:      00:00:01.01
CPU time was:          00:00:00.20

```

## 事例 10: REF フィールドおよび VARRAY のロード

ここでは、次のことを説明します。

- 主キーを OID として利用し、注文した商品を VARRAY に格納するカスタマ表をロードします。
- カスタマ表および VARRAY 内の注文した商品への参照を持つ注文表をロードします。

---

**注意：** 事例 10 では、初期化パラメータ・ファイルの COMPATIBILITY パラメータを 8.1.0 以上に設定しておく必要があります。設定していない場合、表が正しく作成されずにエラー・メッセージが返されます。COMPATIBILITY パラメータの設定方法の詳細は、『Oracle9i データベース移行ガイド』を参照してください。

---

## 事例 10 の制御ファイル

```
LOAD DATA
INFILE *
CONTINUEIF THIS (1) = '*'
INTO TABLE customers
REPLACE
FIELDS TERMINATED BY ","
(
  CUST_NO              CHAR,
  NAME                 CHAR,
  ADDR                 CHAR
)
INTO TABLE orders
REPLACE
FIELDS TERMINATED BY ","
(
  order_no             CHAR,
1) cust_no             FILLER  CHAR,
2) cust                REF (CONSTANT 'CUSTOMERS', cust_no),
1) item_list_count     FILLER  CHAR,
3) item_list           VARRAY COUNT (item_list_count)
(
4) item_list           COLUMN OBJECT
(
5)  item               CHAR,
    cnt                CHAR,
    price              CHAR
  )
)
)
6) BEGINDATA
*00001,Spacely Sprockets,15 Space Way,
*00101,00001,2,
*Sprocket clips, 10000, .01,
*Sprocket cleaner, 10, 14.00
*00002,Cogswell Cogs,12 Cogswell Lane,
*00100,00002,4,
*one quarter inch cogs,1000,.02,
```

```
*one half inch cog, 150, .04,
*one inch cog, 75, .10,
*Custom coffee mugs, 10, 2.50
```

#### 注意：

1. FILLER フィールドです。FILLER フィールドがマップされているデータ・フィールドの値が割り当てられます。詳細は、6-6 ページの「[FILLER フィールドの指定](#)」を参照してください。
2. REF フィールドとして作成されます。詳細は、7-15 ページの「[REF 列のロード](#)」を参照してください。
3. item\_list は、VARRAY に格納されます。
4. item\_list が 2 回目に現れると、VARRAY の各要素のデータ型を識別できます。この場合、データ型は COLUMN OBJECT です。
5. VARRAY に格納されているすべての列オブジェクトの属性が示されます。このリストは、カッコで囲まれています。詳細は、7-2 ページの「[列オブジェクトのロード](#)」を参照してください。
6. データは、制御ファイルに含まれており、データの前に BEGINDATA パラメータが付きます。

## 事例 10 の実行

次に示す手順で、事例を実行します。

1. システム・プロンプトで次のように入力して、SQL\*Plus を scott/tiger で起動します。

```
sqlplus scott/tiger
```

SQL プロンプトが表示されます。

2. SQL プロンプトで、この事例用の SQL スクリプトを次のように実行します。

```
SQL> @ulcase10
```

事例用の表が準備および移入されると、システム・プロンプトに戻ります。

3. システム・プロンプトで、次のように入力し、SQL\*Loader を起動して事例を実行します。

```
sqlldr USERID=scott/tiger CONTROL=ulcase10.ctl LOG=ulcase10.log
```

SQL\*Loader によってデータがロードされ、ログ・ファイルが作成されると、システム・プロンプトに戻ります。事例の実行結果を確認するには、ログ・ファイルを調べます。

事例 10 のログ・ファイル

次に、ログ・ファイルの一部を示します。

```
Control File:  ulcase10ctl
Data File:     ulcase10ctl
  Bad File:    ulcase10.bad
  Discard File: none specified

(Allow all discards)

Number to load: ALL
Number to skip: 0
Errors allowed: 50
Bind array:    64 rows, maximum of 256000 bytes
Continuation:  1:1 = 0X2a(character '*'), in current physical record
Path used:     Conventional
```

Table CUSTOMERS, loaded from every logical record.  
Insert option in effect for this table: REPLACE

Column Name	Position	Len	Term	Encl	Datatype
CUST_NO	FIRST	*	,		CHARACTER
NAME	NEXT	*	,		CHARACTER
ADDR	NEXT	*	,		CHARACTER

Table ORDERS, loaded from every logical record.  
Insert option in effect for this table: REPLACE

Column Name	Position	Len	Term	Encl	Datatype
ORDER_NO	NEXT	*	,		CHARACTER
CUST_NO	NEXT	*	,		CHARACTER
(FILLER FIELD)					
CUST	DERIVED				REF
Arguments are:					
CONSTANT 'CUSTOMERS'					
CUST_NO					
ITEM_LIST_COUNT	NEXT	*	,		CHARACTER
(FILLER FIELD)					
ITEM_LIST	DERIVED	*			VARRAY
Count for VARRAY					
ITEM_LIST_COUNT					
*** Fields in ITEM_LIST					
ITEM_LIST	DERIVED	*			COLUMN OBJECT

```
*** Fields in ITEM_LIST.ITEM_LIST
ITEM          FIRST      *      ,      CHARACTER
CNT           NEXT       *      ,      CHARACTER
PRICE        NEXT       *      ,      CHARACTER
*** End of fields in ITEM_LIST.ITEM_LIST
```

```
*** End of fields in ITEM_LIST
```

Table CUSTOMERS:

```
  2 Rows successfully loaded.
  0 Rows not loaded due to data errors.
  0 Rows not loaded because all WHEN clauses were failed.
  0 Rows not loaded because all fields were null.
```

Table ORDERS:

```
  2 Rows successfully loaded.
  0 Rows not loaded due to data errors.
  0 Rows not loaded because all WHEN clauses were failed.
  0 Rows not loaded because all fields were null.
```

```
Space allocated for bind array:          149120 bytes (64 rows)
Read  buffer bytes: 1048576
```

```
Total logical records skipped:          0
Total logical records read:             2
Total logical records rejected:          0
Total logical records discarded:         0
```

```
Run began on Wed Feb 27 14:05:29 2002
Run ended on Wed Feb 27 14:05:31 2002
```

```
Elapsed time was:      00:00:02.07
CPU time was:          00:00:00.20
```

## 事例 11: Unicode キャラクタ・セットのデータのロード

この事例では、SQL\*Loader で、Unicode キャラクタ・セットのデータ・ファイルからデータをロードします。この事例は、事例 3 と対応します。ただし、ここでは UTF16 キャラクタ・セットを使用し、empno フィールドおよび deptno フィールドに最大長が指定されています。データは、CHARACTERSET キーワードが指定されているため、別々のデータ・ファイルにある必要があります。ここでは、次のことを説明します。

- SQL\*Loader を使用して、Unicode キャラクタ・セット UTF16 のデータをロードします。
- SQL\*Loader を使用して、固定幅マルチバイト・キャラクタ・セットのデータをロードします。
- 文字長セマンティクスを使用します。
- SQL\*Loader を使用して、リトル・エンディアンバイト順序でデータをロードします。SQL\*Loader を実行しているシステムのバイト順序が確認されます。必要に応じて、SQL\*Loader を使用して、バイト順序依存のデータが正しくロードされるように、データのバイト順序を入れ替えます。

### 事例 11 の制御ファイル

制御ファイルは ulcase11.ctl です。

```
LOAD DATA
1) CHARACTERSET UTF16
2) BYTEORDER LITTLE
INFILE ulcase11.dat
REPLACE

INTO TABLE emp
3) FIELDS TERMINATED BY X'002c' OPTIONALLY ENCLOSED BY X'0022'
4) (empno INTEGER EXTERNAL (5), ename, job, mgr,
   hiredate DATE(20) "DD-Month-YYYY",
   sal, comm,
5) deptno CHAR(5) TERMINATED BY ":",
   projno,
   loadseq SEQUENCE(MAX,1) )
```

#### 注意：

1. CHARACTERSET キーワードで指定されたキャラクタ・セットは UTF16 です。SQL\*Loader で、UTF16 キャラクタ・セットのデータをデータ・ファイルのキャラクタ・セットに変換します。この行によって、文字長セマンティクスを使用してロードすることが SQL\*Loader に指定されます。



2. BYTEORDER LITTLE は、データ・ファイルのデータがリトル・エンディアンバイト順序であることが SQL\*Loader に指定されます。SQL\*Loader で、バイト・スワップが必要かどうかを判断するために、実行しているシステムのバイト順序を確認します。この例では、UTF16 のすべての文字データはバイト順序依存です。
3. TERMINATED BY 句および OPTIONALLY ENCLOSED BY 句は、両方とも 16 進文字列を指定します。ビッグ・エンディアン形式の UTF-16 では、X'002c' はカンマ (,) のエンコーディングです。X'0022' は、ビッグ・エンディアン形式での二重引用符 (") のエンコーディングです。データ・ファイルがリトル・エンディアン形式のため、SQL\*Loader で、一致するかどうかを確認する前にバイト・スワップを行います。  
これらの句が、16 進文字列のかわりに文字列として指定された場合、SQL\*Loader でデータ・ファイルのキャラクタ・セット (UTF16) に文字列を変換して、一致するかどうかを確認する前に必要なバイト・スワップを行います。
4. 文字長セマンティクスが使用されるため、empno、hiredate および deptno フィールドの最大長は、バイトでなく文字として解釈されます。
5. deptno フィールドの TERMINATED BY 句は、文字列「:」を使用して指定します。SQL\*Loader で、文字列をデータ・ファイルのキャラクタ・セット (UTF16) に変換して、一致するかどうかを確認する前に必要なバイト・スワップを行います。

**参照：** 次の項を参照してください。

- 5-16 ページの「異なる文字コード体系の処理」
- 6-36 ページの「バイト順序」

## 事例 11 のデータ・ファイル

```
7782, "Clark", "Manager", 7839, 09-June-1981, 2572.50,, 10:101
7839, "King", "President", , 17-November-1981, 5500.00,, 10:102
7934, "Miller", "Clerk", 7782, 23-January-1982, 920.00,, 10:102
7566, "Jones", "Manager", 7839, 02-April-1981, 3123.75,, 20:101
7499, "Allen", "Salesman", 7698, 20-February-1981, 1600.00, 300.00, 30:103
7654, "Martin", "Salesman", 7698, 28-September-1981, 1312.50, 1400.00, 30:103
7658, "Chan", "Analyst", 7566, 03-May-1982, 3450,, 20:101
```

## 事例 11 の実行

次に示す手順で、事例を実行します。

1. システム・プロンプトで次のように入力して、SQL\*Plus を scott/tiger で起動します。

```
sqlplus scott/tiger
```

SQL プロンプトが表示されます。

2. SQL プロンプトで、この事例用の SQL スクリプトを次のように実行します。

```
SQL> @ulcase11
```

事例用の emp 表が準備されると、システム・プロンプトに戻ります。

3. システム・プロンプトで、次のように入力し、SQL\*Loader を起動して事例を実行します。

```
sqlldr USERID=scott/tiger CONTROL=ulcase11.ctl LOG=ulcase11.log
```

SQL\*Loader によって emp 表にデータがロードされ、ログ・ファイルが作成されると、システム・プロンプトに戻ります。事例の実行結果を確認するには、ログ・ファイルを調べます。

## 事例 11 のログ・ファイル

次に、ログ・ファイルの一部を示します。

```
Control File:   ulcase11.ctl
Character Set utf16 specified for all input.
1) Using character length semantics.
2) Byteorder little endian specified.
Processing datafile as little endian.
3) SQL*Loader running on a big endian platform. Swapping bytes where needed.
```

```
Data File:      ulcase11.dat
Bad File:       ulcase11.bad
Discard File:   none specified
```

(Allow all discards)

```
Number to load: ALL
Number to skip: 0
Errors allowed: 50
Bind array:     64 rows, maximum of 256000 bytes
Continuation:   none specified
Path used:      Conventional
```

Table EMP, loaded from every logical record.

Insert option in effect for this table: REPLACE

Column Name	Position	Len	Term	Encl	Datatype
4) EMPNO	FIRST	10	,	O(")	CHARACTER
ENAME	NEXT	*	,	O(")	CHARACTER
JOB	NEXT	*	,	O(")	CHARACTER
MGR	NEXT	*	,	O(")	CHARACTER
4) HIREDATE	NEXT	40	,	O(")	DATE DD-Month-YYYY
SAL	NEXT	*	,	O(")	CHARACTER
COMM	NEXT	*	,	O(")	CHARACTER
DEPTNO	NEXT	10	:	O(")	CHARACTER
4) PROJNO	NEXT	*	,	O(")	CHARACTER
LOADSEQ					SEQUENCE (MAX, 1)

Table EMP:

7 Rows successfully loaded.  
 0 Rows not loaded due to data errors.  
 0 Rows not loaded because all WHEN clauses were failed.  
 0 Rows not loaded because all fields were null.

Space allocated for bind array: 104768 bytes (64 rows)  
 Read buffer bytes: 1048576

Total logical records skipped: 0  
 Total logical records read: 7  
 Total logical records rejected: 0  
 Total logical records discarded: 0

Run began on Wed Feb 27 16:33:47 2002  
 Run ended on Wed Feb 27 16:33:49 2002

Elapsed time was: 00:00:01.74  
 CPU time was: 00:00:00.20

注意：

- 1. SQL\*Loader では、このロードに文字長セマンティクスを使用します。キャラクタ・セットが UTF16 の場合は、デフォルトです。最大サイズの長さの確認は、文字単位で行われることを意味しています（4 の説明を参照）。
  - 2. BYTEORDER LITTLE は、制御ファイルで指定されます。これによって、データ・ファイルの UTF16 文字データのバイト順序がリトル・エンディアンであることが SQL\*Loader に指定されます。
  - 3. このメッセージは、SQL\*Loader がデータ・ファイルのバイト順序とは逆のバイト順序（この場合、ビグ・エンディアン）のシステムで実行されている場合のみに表示されます。このメッセージは、SQL\*Loader を実行しているシステムのバイト順序とデータ・ファイルのバイト順序が逆であることが、SQL\*Loader で検出されたことを示しています。そのため、SQL\*Loader で、バイト順序に依存しているデータ（この場合、すべての UTF16 文字データ）のバイト・スワップを行う必要があります。
  - 4. len の下にある最大長は、文字長セマンティクスが使用されていてもバイト単位です。ただし、最大長は UTF16 文字の最大サイズに基づいてバイト単位で調整されます。すべての UTF16 文字は、2 バイトです。したがって、empno および projno (5) に指定されたサイズを 2 倍して、最大サイズは 10 バイトになります。
- 同様に、hiredate の最大サイズ (20) を 2 倍して、最大サイズは 40 バイトになります。

事例 11 のロードされた表

SQL\*Loader のこの実行結果を参照するには、SQL プロンプトで次の問合せを実行します。

```
SQL> SELECT * FROM emp;
```

問合せの結果は、次のように表示されます（表示形式は、ディスプレイによって多少異なる場合があります）。

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO	PROJNO	LOADSEQ
7782	Clark	Manager	7839	09-JUN-81	2572.50		10	101	1
7839	King	President		17-NOV-81	5500.00		10	102	2
7934	Miller	Clerk	7782	23-JAN-82	920.00		10	102	3
7566	Jones	Manager	7839	02-APR-81	3123.75		20	101	4

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO	PROJNO	LOADSEQ
-------	-------	-----	-----	----------	-----	------	--------	--------	---------

7499	Allen	Salesman	7698	20-FEB-81	1600.00	300	30	103	5
7654	Martin	Salesman	7698	28-SEP-81	1312.50	1400	30	103	6
7658	Chan	Analyst	7566	03-MAY-82	3450.00		20	101	7

7 rows selected.

表の出力は、キャラクタ・セット **US7ASCII** で表示されます。これは、通常、**NLS\_LANG** パラメータが指定されていない場合のデフォルトのキャラクタ・セットです。**SQL\*Loader** で、データベース・キャラクタ・セット（通常は、デフォルトの **WE8DEC**）の出力を、**NLS\_LANG** パラメータによってセッションに指定されたキャラクタ・セットに変換します。



# 第III部

---

## 外部表

第 III 部では、外部表の使用方法について説明します。この部に含まれる章は、次のとおりです。

### [第 11 章「外部表の概要」](#)

この章では、外部表の基本概念について説明します。

### [第 12 章「外部表アクセス・パラメータ」](#)

この章では、外部表 API でのインタフェースに使用する接続パラメータについて説明します。





---

## 外部表の概要

Oracle9i の外部表機能は、既存の SQL\*Loader 機能を補足する機能です。この機能を使用して、外部ソースにデータがある場合でも、そのデータがデータベースの表にあるかのようにアクセスできます。

外部表は読み込み専用です。外部表に対しては、データ操作言語（DML）操作も索引作成もできません。したがって、データ・ロード時にステージング表の追加の索引付けが必要な場合、SQL\*Loader を使用の方が有効です。SQL\*Loader と外部表との処理内容の違いについては、11-7 ページの「[SQL\\*Loader と外部表との処理内容の違い](#)」を参照してください。

外部表機能を使用するには、ご使用のプラットフォーム上のデータ・ファイルのファイル形式およびレコード形式の知識が必要です。また、外部表を作成し、その外部表に問合せを実行するための SQL の知識も必要です。

この章の内容は、次のとおりです。

- [アクセス・ドライバ](#)
- [外部表の制限事項](#)
- [データ・ファイルおよび出力ファイルの位置](#)
- [外部表を使用したデータのロード](#)
- [外部表へのパラレル・アクセス](#)
- [外部表使用時のパフォーマンスのヒント](#)
- [SQL\\*Loader と外部表との処理内容の違い](#)

## アクセス・ドライバ

外部表には、外部表レイヤーからサーバーへのデータの提供方法が記述されています。アクセス・ドライバおよび外部表レイヤーを使用して、データ・ファイルのデータを変換して外部表の定義と一致させます。

特定の型の外部表を作成する場合は、外部データ・ソースを記述するアクセス・パラメータを指定します。外部表に型を指定しない場合は、デフォルトとして ORACLE\_LOADER 型が使用されます。ORACLE\_LOADER 型のアクセス・パラメータの詳細は、[第 12 章](#)を参照してください。

データ・ソースのデータの記述は外部表の定義とは別です。これは、次のことを意味します。

- ソース・ファイルに含まれるフィールドの数は、外部表の列数と異なる場合があります。
- データ・ソースのフィールドのデータ型は、外部表の列のデータ型と異なる場合があります。

アクセス・ドライバによって、データ・ソースのデータが、外部表の定義と一致するように処理されます。

次の例では、emp という従来の表と emp\_load という外部表が定義されます。

```
CREATE TABLE emp (emp_no CHAR(6), last_name CHAR(25), first_name CHAR(20),
                  middle_initial CHAR(1));

CREATE TABLE emp_load (employee_number CHAR(5), employee_last_name CHAR(20),
                       employee_first_name CHAR(15), employee_middle_name CHAR(15))
ORGANIZATION EXTERNAL (TYPE ORACLE_LOADER DEFAULT DIRECTORY ext_tab_dir
                       ACCESS PARAMETERS (RECORDS FIXED 62 FIELDS (employee_number INTEGER(2),
                                                                    employee_dob CHAR(20),
                                                                    employee_last_name CHAR(18),
                                                                    employee_first_name CHAR(11),
                                                                    employee_middle_name CHAR(11)))
                       LOCATION ('foo.dat')));

INSERT INTO emp (emp_no, first_name, middle_initial, last_name)
(SELECT employee_number, employee_first_name,
        substr(employee_middle_name, 1, 1),
        employee_last_name
FROM emp_load);
```

この例では、次のことに注意してください。

- データ・ファイルの employee\_number フィールドは、外部表の employee\_number フィールドの文字列に変換されます。

- データ・ファイルには、表のいずれのフィールドにもロードされない `employee_dob` フィールドが含まれています。
- 外部表の `employee_middle_name` 列で使用されている `substr` 関数によって、`emp` 表の `middle_initial` の値が生成されます。

## 外部表の制限事項

この項では、外部表で行われない処理および外部表処理上の制限事項について説明します。

- 外部表には、データベースに格納されているデータは記述されません。
- 外部表には、外部ソースでのデータの格納方法は記述されません。これは、アクセス・パラメータの機能です。
- 外部表は読み込み専用ソースです。外部表に対して、挿入操作もレコードの更新もできません。
- 列の処理。外部表では、問合せが実行される列に応じて、返される行数が異なります。問合せの実行に必要なデータ変換およびデータ処理の数を最小限するには、外部表のアクセス・ドライバを使用して、問合せで参照される列のみを処理します。これによってデータ型変換エラーが発生した列を含んでいたために拒否された行も、その列が参照されないかぎり、別の問合せでは拒否されません。
- ロード時におけるバイト順序マークの処理。データ・ファイル・キャラクタ・セットが UTF8 または UTF16 の外部表ロードでは、バイト順序マークの確認は抑止できません。バイト順序マークの確認は、データ・ファイルの先頭にバイト順序マークのエンコーディングと一致するバイナリ・データが含まれている場合にのみ抑止する必要があります（SQL\*Loader を使用したロードでは、バイト順序マークの確認を抑止できます）。バイト順序マークを確認するということが、必ずしもバイト順序マークがデータ・ファイル内に存在するということではないことに注意してください。バイト順序マークがない場合は、サーバー・プラットフォームのバイト順序が使用されます。

## データ・ファイルおよび出力ファイルの位置

アクセス・ドライバは、データベース・サーバー内で実行されます。これは、SQL\*Loader が、ロードするデータをサーバーに送信するクライアント・プログラムであるという点で、SQL\*Loader とは異なります。この違いは、次のことを意味しています。

- サーバーでは、アクセス・ドライバによってロードされるファイルにアクセスする必要があります。
- サーバーでは、アクセス・ドライバによって作成されたファイル（ログ・ファイル、不良ファイルおよび廃棄ファイル）の作成および書込みを行う必要があります。

アクセス・ドライバでは、ファイルにランダムな名前を指定できません。これは、ユーザーがアクセスできないファイルに、サーバーがアクセスする場合があります、ユーザーがデータを

読み込むことができると、セキュリティに影響するためです。同様に、通常、ユーザーが削除権限を持たないファイルがサーバーによって上書きされる場合があるため、ユーザーは出力ファイルの位置を指定することもできません。

かわりに、ファイルの読み込み元および書き込み元の位置としてディレクトリ・オブジェクトを指定する必要があります。ディレクトリ・オブジェクトは、ファイル・システムのディレクトリ名に名前をマップします。たとえば、次の文で `load_src` という名前のディレクトリ・オブジェクトを作成します。

```
create directory load_src as '/usr/apps/datafiles';
```

ディレクトリ・オブジェクトは、DBA または `CREATE ANY DIRECTORY` 権限を持つすべてのユーザーが作成できます。ディレクトリの作成後、ディレクトリ・オブジェクトを作成するユーザーは、そのディレクトリの読み込み権限または書き込み権限を他のユーザーに付与する必要があります。たとえば、`load_src` で指定されたディレクトリのユーザー `scott` のかわりに、サーバーがファイルを読み込むことができるようにするには、ディレクトリ・オブジェクトを作成したユーザーが、次のコマンドを実行する必要があります。

```
GRANT READ ON DIRECTORY load_src TO scott;
```

ディレクトリ・オブジェクトの名前は、`CREATE TABLE...ORGANIZATION EXTERNAL` 文の次の位置に示すことができます。

- デフォルトのディレクトリ句。この句には、明示的にディレクトリ・オブジェクトに名前を付けないすべての入出力ファイルに対して使用するデフォルトのディレクトリを指定します。
- 外部表のすべてのデータ・ファイルをリストする `LOCATION` 句。ファイル名は、`directory:file` という書式です。`directory` の部分はオプションです。この部分を指定しないと、デフォルトのディレクトリがファイルのディレクトリとして使用されます。
- 出力ファイルに名前を付けるアクセス・パラメータ。ファイル名は、`directory:file` という書式です。`directory` の部分はオプションです。この部分を指定しないと、デフォルトのディレクトリがファイルのディレクトリとして使用されます。アクセス・パラメータの構文で、特定の出力ファイルを作成しないように指定できます。この構文が有効なのは、出力ファイルを必要としない場合、またはいずれのディレクトリ・オブジェクトにも書き込みアクセスをしない場合です。

`SYS` ユーザーのみがディレクトリ・オブジェクトを所有できます。ただし、`SYS` ユーザーは、ディレクトリ・オブジェクトを作成する権限を他のユーザーに付与できます。ディレクトリ・オブジェクトへの読み込み権限または書き込み権限は、Oracle データベース・サーバーによるファイルの読み込みまたは書き込みのみを意味します。適切なオペレーティング・システム権限がないかぎり、Oracle データベース・サーバーの外部にあるファイルには直接アクセスできません。同様に、Oracle データベース・サーバーには、ディレクトリのファイルに対して読み込みおよび書き込みを行うオペレーティング・システム権限が必要です。

## 外部表を使用したデータのロード

外部表は、主に、データベースの実際の表にデータをロードするための行ソースとして使用されます。外部表を作成した後、その外部表を `SELECT` 句のソースとして使用して、`CREATE TABLE AS SELECT` 文または `INSERT INTO... AS SELECT` 文を発行できます。外部表は読み込み専用であるため、外部表へのデータの挿入、および外部表のレコードの更新はできません。

SQL 文を介して外部表にアクセスすると、外部表のフィールドは、通常の表の他のフィールドと同様に使用できます。特に、SQL の組込み関数、PL/SQL ファンクションまたは Java ファンクションの引数として使用できます。これによって、外部ソースのデータを操作できます。

外部表に列オブジェクトは含まれませんが、コンストラクタ・ファンクションを使用して外部表の属性から列オブジェクトを作成できます。たとえば、データベースの表が次のように定義されているとします。

```
CREATE TYPE student_type AS object (  
    student_no CHAR(5),  
    name CHAR(20))  
/
```

```
CREATE TABLE roster (  
    student student_type,  
    grade CHAR(2));
```

また、次のように定義された外部表があるとします。

```
CREATE TABLE roster_data (  
    student_no CHAR(5),  
    name CHAR(20),  
    grade CHAR(2))  
    ORGANIZATION EXTERNAL (TYPE ORACLE_LOADER DEFAULT DIRECTORY ext_tab_dir  
        ACCESS PARAMETERS (FIELDS TERMINATED BY ',')  
        LOCATION ('foo.dat'));
```

表 `roster` を `roster_data` からロードするには、次のとおり指定します。

```
INSERT INTO roster (student, grade)  
    (SELECT student_type(student_no, name), grade FROM roster_data);
```

## 外部表へのパラレル・アクセス

データ・ファイルでのパラレル処理を外部表でサポートするには、外部表の作成時に `PARALLEL` 句を使用します。アクセス・ドライバで、大きいデータ・ファイルを個別に処理できるチャンクに分割します。

次のファイル、レコードおよびデータ特性によって、ファイルのパラレル処理が禁止されます。

- 順次データ・ソース（テープ・ドライブ、パイプなど）
- 文字の境界が文字列中の任意のバイトで始まり、境界を判断できないマルチバイト・キャラクタ・セットのデータ

この制限事項は、1 レコード当たりのバイト数が固定のデータ・ファイルには適用されません。

- `VAR` 形式のレコード

## 外部表使用時のパフォーマンスのヒント

パフォーマンスを監視する場合、最も重要なことは、ロードの経過時間の測定です。また、CPU 使用量、メモリー使用量および I/O 率の測定も重要です。

並列度を増減することによって、パフォーマンスを変更できます。並列度は、データ・ファイルの処理に起動できるアクセス・ドライバの数を示します。並列度によって、リソース使用率を低くした遅いロードと、すべてのリソースを使用した速いロードを選択できます。アクセス・ドライバは、アクセス・ドライバ専用使用するリソース量を判断できないため、自動的にチューニングされません。

パフォーマンスは、日付キャッシュ機能を使用して向上させることができる場合もあります。日付キャッシュを使用して、ロード中に予測される一意の日付の数を指定する、入力データ内に多数の重複する日付またはタイムスタンプ値が存在する場合と、日付変換が実行される回数を減らすことができます。外部表で提供される日付キャッシュ機能は、`SQL*Loader` で提供されるものと同じです。詳細は、9-21 ページの「[日付キャッシュの値の指定](#)」を参照してください。

パフォーマンスを向上させるには、並列度の変更および日付キャッシュの使用に加えて、次のことを考慮してください。

- 固定長レコードは、文字列で終了しているレコードより速く処理される。
- 固定長フィールドは、デリミタ付きフィールドより速く処理される。
- シングルバイト・キャラクタ・セットは、最も速く処理される。
- 固定幅キャラクタ・セットは、可変幅キャラクタ・セットより速く処理される。
- 可変幅キャラクタ・セットのバイト長セマンティクスは、文字長セマンティクスより早く処理される。

- 1 文字のレコード終了デリミタおよびフィールド・デリミタは、複数文字のデリミタより速く処理される。
- キャラクタ・セットを変換するより、データ・ファイルのキャラクタ・セットをデータベースのキャラクタ・セットに一致させる方が速く処理される。
- データ型を変換するより、データ・ファイルのデータ型をデータベースのデータ型に一致させる方が速く処理できる。
- 拒否された行を拒否ファイルに書き込まない場合は、行の書込みオーバーヘッドが削減されるため、処理速度が速くなる。
- 条件句（WHEN、NULLIF および DEFAULTIF を含む）を使用すると、処理速度が遅くなる。

アクセス・ドライバは、マルチスレッドを使用して作業をできるだけ簡素化します。

## SQL\*Loader と外部表との処理内容の違い

この項では、外部表を使用したデータのロード方法と、SQL\*Loader の従来型パス・ロードおよびダイレクト・パス・ロードを使用したデータのロード方法の重要な違いについて説明します。

### 複数のプライマリ入力データ・ファイル

SQL\*Loader のロードを使用したプライマリ入力データ・ファイルが複数存在する場合は、入力データ・ファイルごとに不良ファイルおよび廃棄ファイルが作成されます。外部表ロードでは、すべての入力データ・ファイルに対する不良ファイルおよび廃棄ファイルは、1 つずつのみです。外部表ロードでパラレル・アクセス・ドライバが使用される場合は、各アクセス・ドライバに不良ファイルおよび廃棄ファイルが含まれます。

### 構文およびデータ型

次の操作は、外部表ロードではサポートされていません。

- CONTINUEIF または CONCATENATE を使用した、1 つの論理レコードへの複数の物理レコードの結合
- SQL\*Loader データ型（GRAPHIC、GRAPHIC EXTERNAL および VARGRAPHIC）のロード
- データベースの列型（CLOB、NCLOB、BLOB、LONG、ネストした表、VARRAY、REF、主キー、REF および SID）の使用

## 拒否された行

SQL\*Loader では、SEQUENCE パラメータが使用された場合に拒否された行があっても、この行の順序番号値は更新されます。外部表では、SEQUENCE パラメータが使用されると、拒否された行の順序番号値は更新されません。たとえば、順序番号が 1 から始まり、1 ずつ増える 5 つの行をロードするとします。SQL\*Loader では、行 2 および 4 が拒否された場合、正常にロードされた行に順序番号 1、3 および 5 が割り当てられます。外部表ロードでは、正常にロードされた行に順序番号 1、2 および 3 が割り当てられます。

## BOM

SQL\*Loader では、プライマリ・データ・ファイルに Unicode キャラクタ・セット (UTF8 または UTF16) が使用され、バイト順序マーク (BOM) が含まれている場合、バイト順序マークは対応する不良ファイルおよび廃棄ファイルの先頭に書き込まれます。外部表ロードでは、バイト順序マークは不良ファイルおよび廃棄ファイルの先頭に書き込まれません。

## デフォルトのキャラクタ・セットおよび日付マスク

データ・ファイルのフィールドでは、クライアントの NLS 環境によってデフォルトのキャラクタ・セットおよび日付マスクが決定されます。外部表のフィールドでは、サーバーの NLS 環境によってデフォルトのキャラクタ・セットおよび日付マスクが決定されます。



---

## 外部表アクセス・パラメータ

この章で説明するアクセス・パラメータによって、外部表アクセス・ドライバへのインタフェースが提供されます。アクセス・パラメータは、外部表の作成時に指定します。この章では、デフォルトのアクセス・ドライバに対するアクセス・パラメータの構文について説明します。

この章で説明する情報を使用するには、ご使用のプラットフォームのデータ・ファイルのファイル形式およびレコード形式（キャラクタ・セット、フィールドのデータ型など）についての知識が必要です。また、外部表を作成し、その外部表に問合せを実行するための SQL の知識も必要です。

SQL\*Loader で `EXTERNAL_TABLE=GENERATE_ONLY` パラメータを使用すると、任意の SQL\*Loader 制御ファイルに適正なアクセス・パラメータを取得できます。GENERATE\_ONLY を指定すると、制御ファイルに記述されているとおり、SQL\*Loader ログ・ファイル内の外部表を使用してロードを行うために必要なすべての SQL 文が書き込まれます。これらの SQL 文は、編集およびカスタマイズできます。実際のロードは、SQL\*Loader を使用せずに、SQL\*Plus でこれらの文を実行して、後で行うことができます。

**参照：** 次の項を参照してください。

- 4-7 ページの「[EXTERNAL\\_TABLE](#)」
- 8-8 ページの「[EXTERNAL\\_TABLE=GENERATE\\_ONLY](#) の使用時に作成されるログ・ファイル」

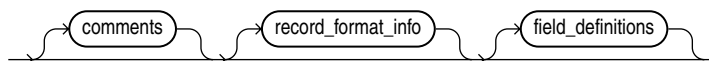
---

**注意：**

- 章の後半に記載されている他の構文が使用されているため、構文の説明がわかりにくい場合があります。構文によって行われる処理が明確でない場合は、先に進み、その説明を参照してください。
  - この章では、外部表の場合の CREATE TABLE...ORGANIZATION EXTERNAL 文の例をデータ・ファイルの内容のサンプルとともに示します。これらの内容は、CREATE TABLE 文の一部ではなく、完全な例を示します。
- 

## access\_parameters 句

access\_parameters 句には、コメント、レコード形式およびフィールド形式の情報が含まれています。access\_parameters 句の構文は次のとおりです。



### コメント

コメントは、2つのハイフンで始まり、その後にテキストが続く行です。コメントは、次の例のように、アクセス・パラメータより前に位置する必要があります。

```
--This is a comment
--This is another comment
RECORDS DELIMITED BY NEWLINE
```

二重ハイフンの右側のすべてのテキストは行末まで無視されます。

### record\_format\_info

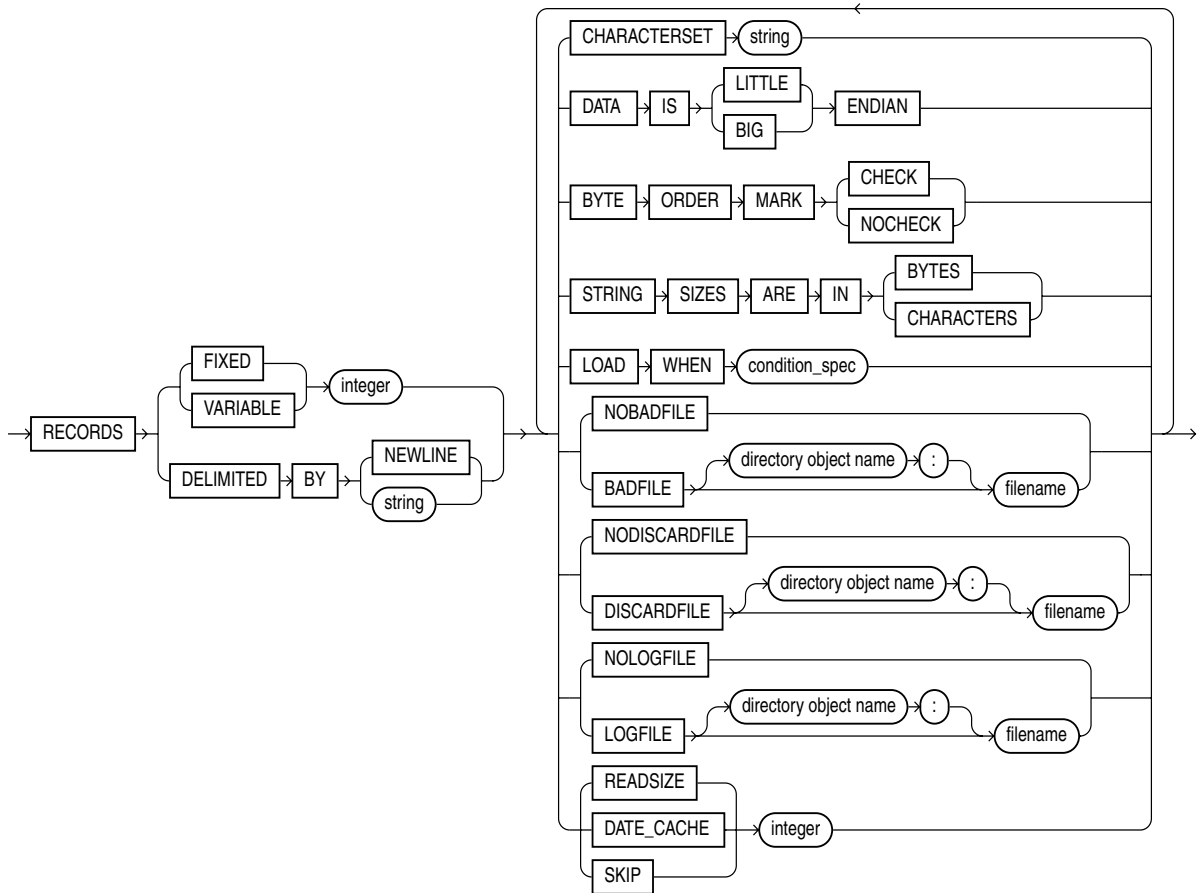
record\_format\_info 句には、レコード（形式など）、データのキャラクタ・セットおよびレコードをロード対象とする規則についての情報が含まれます。record\_format\_info 句はオプションです。構文の詳細は、12-3 ページの「[record\\_format\\_info 句](#)」を参照してください。

### field\_definitions

field\_definitions 句を使用して、データ・ファイルのフィールドを指定します。データ・ファイルのフィールドが外部表の列と同じ名前の場合、フィールドのデータはその列に使用されます。構文の詳細は、12-14 ページの「[field\\_definitions 句](#)」を参照してください。

## record\_format\_info 句

record\_format\_info 句には、レコード（形式など）、データのキャラクタ・セットおよびレコードをロード対象とする規則についての情報が含まれます。record\_format\_info 句はオプションです。句を指定しない場合、デフォルトの値は RECORDS DELIMITED BY NEWLINE です。record\_format\_info 句の構文は次のとおりです。



## FIXED

FIXED 句を使用して、すべてのレコードをバイト単位の固定長として識別します。FIXED レコードに対して指定したサイズには、改行などのレコード終了文字を含める必要があります。他のレコード型と比較して、固定長レコードの固定長フィールドは、アクセス・ドライバを最も簡単に処理できるフィールドおよびレコード形式です。

次に、FIXED レコードの使用例を示します。データ・ファイルの各レコードの終わりに 1 バイトの改行文字があるとしします。その後に、ロードが可能なデータ・ファイルのサンプルを示します。

```
CREATE TABLE emp_load (first_name CHAR(15), last_name CHAR(20), year_of_birth CHAR(4))
  ORGANIZATION EXTERNAL (TYPE ORACLE_LOADER DEFAULT DIRECTORY ext_tab_dir
    ACCESS PARAMETERS (RECORDS FIXED 20 FIELDS (first_name CHAR(7),
                                                    last_name CHAR(8),
                                                    year_of_birth CHAR(4)))
    LOCATION ('foo.dat'));
```

```
Alvin  Tolliver1976
KennethBaer    1963
Mary   Dube    1973
```

## VARIABLE

VARIABLE 句を使用して、レコードを可変長として識別します。各レコードの先頭に、レコードのバイト数を示す文字列が付きます。カウント・フィールドを含む文字列の長さは、VARIABLE パラメータの後に続くサイズ引数となります。サイズは、文字数ではなく、バイト数で表されることに注意してください。レコードの先頭の数値にレコード終了文字の分が含まれる必要があります。ただし、カウント・フィールド自身のサイズは含まれません。レコード終了文字のバイト数は、ファイルの作成方法および作成時のプラットフォームによって異なります。

次に、VARIABLE レコードの使用例を示します。データ・ファイルの各レコードの終わりに 1 バイトの改行文字があるとしします。その後に、ロードが可能なデータ・ファイルのサンプルを示します。

```
CREATE TABLE emp_load (first_name CHAR(15), last_name CHAR(20), year_of_birth CHAR(4))
  ORGANIZATION EXTERNAL (TYPE ORACLE_LOADER DEFAULT DIRECTORY ext_tab_dir
    ACCESS PARAMETERS (RECORDS VARIABLE 2 FIELDS TERMINATED BY ','
                        (first_name CHAR(7),
                          last_name CHAR(8),
                          year_of_birth CHAR(4)))
    LOCATION ('foo.dat'));
```

```
21Alvin,Tolliver,1976,
19Kenneth,Baer,1963,
16Mary,Dube,1973,
```

## DELIMITED BY

DELIMITED BY 句を使用して、レコードの終わりを識別する文字を指定します。

DELIMITED BY NEWLINE を指定する場合、実際に使用される値はプラットフォームに依存します。UNIX プラットフォームでは、終了文字は「\n」です。Windows NT では、終了文字は「\r\n」です。

DELIMITED BY *string* を指定する場合、*string* は、テキストまたは一連の 16 進数字のいずれかになります。テキストの場合は、データ・ファイルのキャラクタ・セットに変換され、その結果がレコードの境界の識別に使用されます。詳細は、12-10 ページの「[string](#)」を参照してください。

次の条件を満たす場合は、デリミタの識別には 16 進数字を使用する必要があります。

- アクセス・パラメータのキャラクタ・セットがデータ・ファイルのキャラクタ・セットとは異なる場合
- デリミタ文字列中にデータ・ファイルのキャラクタ・セットに変換できない文字がある場合

16 進数字はバイトに変換されます。16 進文字列ではキャラクタ・セットの変換は実行されません。

ファイルの終わりがレコード終了記号の前で検出された場合、アクセス・ドライバは、終了記号が検出された場合と同様に、ファイルの終わりまでの処理されていないすべてのデータをレコードの部分とみなします。

---

**注意：** デリミタ付きのレコードには、VARCHAR および VARRAW のバイナリ数値を含むバイナリ・データを含めないでください。バイナリ・データを含めると、そのバイナリ・データがデリミタの検索中に文字として解釈されるため、エラーまたは破損が発生します。

---

次に、DELIMITED BY レコードの使用例を示します。

```
CREATE TABLE emp_load (first_name CHAR(15), last_name CHAR(20), year_of_birth CHAR(4))
  ORGANIZATION EXTERNAL (TYPE ORACLE_LOADER DEFAULT DIRECTORY ext_tab_dir
    ACCESS PARAMETERS (RECORDS DELIMITED BY '|' FIELDS TERMINATED BY ','
      (first_name CHAR(7),
        last_name CHAR(8),
        year_of_birth CHAR(4)))
    LOCATION ('foo.dat'));
```

Alvin,Tolliver,1976|Kenneth,Baer,1963|Mary,Dube,1973

## CHARACTERSET

CHARACTERSET *string* 句を使用して、データ・ファイルのキャラクタ・セットを識別します。キャラクタ・セットを指定しない場合、データベースのデフォルトのキャラクタ・セットが使用されます。詳細は、12-10 ページの「[string](#)」を参照してください。

---

---

**注意：** クライアント側の NLS 設定は、データベースに使用されるキャラクタ・セットには影響しません。

---

---

**参照：** Oracle でサポートされるキャラクタ・セットのリストは、『Oracle9i Database グローバリゼーション・サポート・ガイド』を参照してください。

## DATA IS...ENDIAN

DATA IS...ENDIAN 句を使用して、バイト順序がデータ・ファイルを生成したプラットフォームによって異なるデータのエンディアンを指定します。次の型のフィールドは、この句の影響を受けます。

- INTEGER
- UNSIGNED INTEGER
- FLOAT
- DOUBLE
- VARCHAR (数値のみ)
- VARRAW (数値のみ)
- UTF16 キャラクタ・セットの文字データ型
- RECORDS DELIMITED BY *string* によって指定する UTF16 キャラクタ・セット文字列

リトル・エンディアン・データを生成する一般的なプラットフォームには、Windows98 および Windows NT があります。ビッグ・エンディアン・プラットフォームには、Sun Solaris および IBM MVS があります。DATA IS...ENDIAN 句を指定しない場合、データは、アクセス・ドライバが実行されているプラットフォームと同じエンディアンになります。UTF16 データ・ファイルには、ファイルの先頭にデータのエンディアンを示すマークがあります。このマークは、DATA IS...ENDIAN 句に優先します。

## BYTE ORDER MARK (CHECK | NOCHECK)

BYTE ORDER MARK 句を使用して、データ・ファイルにバイト順序マーク (BOM) があるかどうかを確認するかどうかを指定します

BYTE ORDER MARK NOCHECK を指定すると、データ・ファイルに BOM が存在するかどうかを確認されず、データ・ファイルのすべてのデータがデータとして読み込まれます。

BYTE ORDER MARK CHECK を指定すると、データ・ファイルに BOM が存在するかどうかを確認されます。これは、Unicode キャラクタ・セットのデータ・ファイルのデフォルトの動作です。

次に、使用例をいくつか示します。

- データをリトル・エンディアンまたはビッグ・エンディアンとして指定し、CHECK を指定したときにそのエンディアンがデータ・ファイルと一致していないと判断された場合は、エラーが返されます。たとえば、次のパラメータを指定したとします。

```
DATA IS LITTLE ENDIAN  
BYTE ORDER MARK CHECK
```

Unicode データ・ファイル内に BOM が存在するかどうかを確認され、そのデータが実際にはビッグ・エンディアンであった場合は、リトル・エンディアンを指定していたため、エラーが返されます。

- BOM が存在せず、DATA IS...ENDIAN パラメータを使用してエンディアンを指定しない場合は、プラットフォームのエンディアンが使用されます。
- BYTE ORDER MARK NOCHECK を指定し、DATA IS...ENDIAN パラメータを使用してエンディアンを指定した場合は、その値が使用されます。それ以外の場合は、プラットフォームのエンディアンが使用されます。

**参照：** 6-36 ページの「[バイト順序](#)」を参照してください。

## STRING SIZES ARE IN

STRING SIZES ARE IN 句を使用して、文字列の長さがバイト単位であるか、または文字単位であるかを指定します。この句を指定しない場合、アクセス・ドライバは、データベースが使用するモードを使用します。長さが埋め込まれた文字型 (VARCHAR など) も、この句の影響を受けます。この句を指定すると、埋め込まれた長さは、バイト数ではなく、文字数となります。UTF16 などのマルチバイト・キャラクタ・セットのロード時には、STRING SIZES ARE IN CHARACTERS を指定する必要があります。

## LOAD WHEN

LOAD WHEN *condition\_spec* 句を使用して、データベースに渡すレコードを識別します。評価の方法は様々です。

- *condition\_spec* 句がレコードのフィールドを参照する場合、この句は、すべてのフィールドがレコードから解析された後で、NULLIF 句または DEFAULTIF 句の評価が行われる前にのみ評価されます。
- 条件指定が範囲のみを参照する（フィールド名は参照しない）場合、フィールドが解析される前に句が評価されます。これは、ファイル中のロードできないレコードを、エラーなしで現行のレコード定義に解析できない場合に有効です。

詳細は、12-11 ページの「[condition\\_spec](#)」を参照してください。

次に、LOAD WHEN の使用例を示します。

```
LOAD WHEN (empid != BLANKS)
LOAD WHEN ((dept_id = "SPORTING GOODS" OR dept_id = "SHOES") AND total_sales != 0)
```

## BADFILE | NOBADFILE

BADFILE 句を使用して、エラーのためにロードできない場合にレコードが書き込まれるファイルを指定します。たとえば、データ・ファイルのフィールドは外部表の列のデータ型に変換できないため、不良ファイルにレコードが書き込まれます。LOAD WHEN 句が正常に実行されない場合、レコードは不良ファイルには書き込まれず、かわりに、廃棄ファイルに書き込まれます。また、外部表のレコードを使用中にエラーが発生する場合は（外部表に対して INSERT INTO...AS SELECT... を使用した場合の制約違反など）、レコードは不良ファイルに書き込まれません。

不良ファイルの目的は、すべての拒否されたデータを調査および修正して、ファイルをロードできるようにすることです。不良レコードがあってもデータを修正しない場合は、NOBADFILE オプションを使用して不良ファイルの作成を回避できます。

BADFILE を指定する場合は、ファイル名を指定する必要があります。指定しない場合は、エラーが返されます。

BADFILE または NOBADFILE のいずれも指定しない場合、デフォルトでは 1 つ以上のレコードが拒否されると、不良ファイルが作成されます。このファイルの名前は、表名の後に `_%p` が付いたものになります。

詳細は、12-12 ページの「[\[directory object name:\] filename](#)」を参照してください。



## DISCARDFILE | NODISCARDFILE

DISCARDFILE 句を使用して、レコードが LOAD WHEN 句の条件を満たすことができないことが書き込まれるファイルを指定します。この廃棄ファイルは、廃棄される最初のレコードが検出されると作成されます。同じ外部表が複数回アクセスされる場合、廃棄ファイルはそのたびに再度書き込まれます。廃棄レコードを個別のファイルに保存する必要がない場合、NODISCARDFILE を使用します。

DISCARDFILE を指定する場合は、ファイル名を指定する必要があります。指定しない場合は、エラーが返されます。

DISCARDFILE または NODISCARDFILE のいずれも指定しない場合、デフォルトでは 1 つ以上のレコードで LOAD WHEN 句が失敗すると、廃棄ファイルが作成されます。このファイルの名前は、表名の後に `_%p` が付いたものになります。

詳細は、12-12 ページの「[\[directory object name:\] filename](#)」を参照してください。

## LOG FILE | NOLOGFILE

LOGFILE 句を使用して、データ・ファイルのデータへのアクセス中に外部表のユーティリティによって生成されたメッセージを含むファイルを指定します。ログ・ファイルがすでに同じ名前で存在する場合は、アクセス・ドライバによってそのログ・ファイルが再びオープンされ、新しいログ情報がファイルの終わりに追加されます。この点では、既存のファイルを上書きする不良ファイルおよび廃棄ファイルとは異なります。NOLOGFILE を使用してログ・ファイルの作成を回避できます。

LOGFILE を指定する場合は、ファイル名を指定する必要があります。指定しない場合は、エラーが返されます。

LOGFILE または NOLOGFILE のいずれも指定しない場合、デフォルトではログ・ファイルが作成されます。このファイルの名前は、表名の後に `_%p` が付いたものになります。

詳細は、12-12 ページの「[\[directory object name:\] filename](#)」を参照してください。

## SKIP

ロードの前に、データ・ファイルに含まれる、指定した件数のレコードをスキップします。SKIP は、データにパラレルにアクセスしない場合にのみ指定できます。

## READSIZE

READSIZE パラメータを使用して、読み込みバッファのサイズを指定します。読み込みバッファのサイズによって、アクセス・ドライバで処理可能な最大レコード・サイズが制限されます。サイズは、整数のバイト数で指定します。デフォルト値は 512KB (524288 バイト) です。データ・ファイル内に 512KB より大きいレコードがある場合は、デフォルト値より大きい値を指定する必要があります。READSIZE のサイズに上限はありませんが、アクセス・ドライバで割当て可能なメモリの最大量が事実上の上限となります。また、複数のバッファが割り当てられているため、割当てに使用可能なメモリの量によっても制限されます。

## DATE\_CACHE

デフォルトでは、(1000 要素に対して) 日付キャッシュ機能が使用できます。日付キャッシュ機能を完全に使用禁止にするには、値を 0 に設定します。

DATE\_CACHE を使用して、日付キャッシュ・サイズ (エントリ数) を指定します。たとえば、DATE\_CACHE=5000 を指定すると、作成された日付キャッシュごとに最大 5000 の一意の日付エントリが含まれます。必要に応じて、すべての表に固有の日付キャッシュが作成されます。日付キャッシュは、表への格納のためにデータ型変換が必要な日付値またはタイムスタンプ値が 1 つ以上ロードされた場合にのみ作成されます。

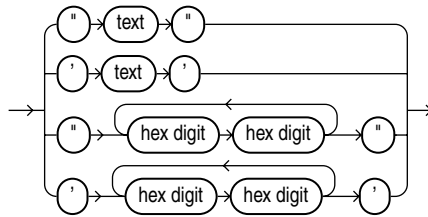
日付キャッシュ機能は、ダイレクト・パス・ロードでのみ使用できます。デフォルトでは、この機能は使用可能です。デフォルトの日付キャッシュ・サイズは 1000 要素です。デフォルトのサイズを使用し、1000 を超える一意の入力値をロードすると、日付キャッシュ機能は、この表に対して自動的に使用禁止となります。ただし、デフォルトを変更して 0 以外の日付キャッシュ・サイズを指定し、キャッシュ量がこのサイズを超えた場合、キャッシュは使用禁止になりません。

ログ・ファイルに含まれている日付キャッシュ統計 (エントリ数、ヒット数、ミス数) を使用して、将来、同様のロードを行うためのキャッシュのサイズを調整できます。

**参照：** 9-21 ページの「[日付キャッシュの値の指定](#)」を参照してください。

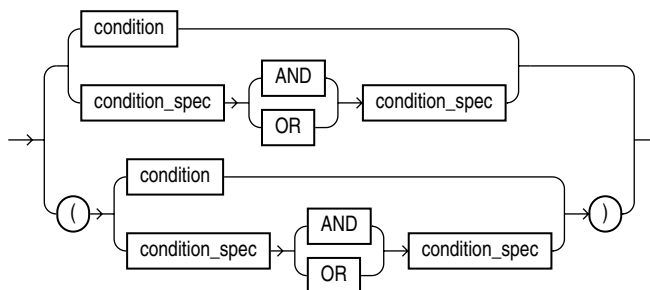
## string

string は、引用符で囲まれた一連の文字または 16 進数字です。16 進数字は、偶数にする必要があります。すべてのテキストが、データ・ファイルのキャラクタ・セットに変換されます。16 進数字は、バイナリに翻訳されたものに変換され、その翻訳結果は文字列として処理されます。アクセス・ドライバでは、その文字列は翻訳されませんが、データ・ファイルのキャラクタ・セットが使用されるとみなされます。string の構文は次のとおりです。



## condition\_spec

condition\_spec は、真または偽のいずれかに評価される式です。ブール演算子によって結合される 1 つ以上の条件を指定します。条件およびブール演算子は、左から右へと評価されます（ブール演算子は、条件が評価された後に適用されます）。カッコを使用して、ブール演算子を実行するデフォルトの順序を変更できます。condition\_spec 句の評価にはより多くのレコード処理時間が必要であるため、多くの句を使用しないようにする必要があります。condition\_spec の構文は次のとおりです。



条件指定にフィールド名を参照する条件が含まれている場合、条件指定は、すべてのフィールドがレコードで検出され、空白の切捨てが行われた後のみに評価されます。空白がフィールドから切り捨てられている場合、フィールドと BLANKS の比較は有効ではありません。

次に、condition\_spec の使用例を示します。

```
empid = BLANKS OR last_name = BLANKS
(dept_id = SPORTING GOODS OR dept_id = SHOES) AND total_sales != 0
```

**参照：** 12-13 ページの「[condition](#)」を参照してください。

## [directory object name:] filename

この句を使用して、出力ファイル（BADFILE、DISCARDFILE または LOGFILE）の名前を指定します。そのディレクトリ・オブジェクト名は、外部表にアクセスしているユーザーが書き込み権限を所有しているディレクトリ・オブジェクトの名前です。このディレクトリ・オブジェクト名を指定しない場合、CREATE TABLE AS EXTERNAL 文の DEFAULT DIRECTORY 句に対して指定した値が使用されます。

filename パラメータは、ディレクトリ・オブジェクト内に作成するファイルの名前です。パラレル・ロードでファイル名を一意にするには、アクセス・ドライバで記号置換を行います。UNIX および Windows NT でサポートされる記号置換は、次のとおりです（その他のプラットフォームでは、別の記号が使用される場合があります）。

- %p は、現行のプロセスのプロセス ID に置換されます。たとえば、アクセス・ドライバのプロセス ID が 12345 の場合、exttab\_%p.log は、exttab\_12345.log となります。
- %a は、現行のプロセスのエージェント番号に置換されます。エージェント番号は、外部表にアクセスしている各パラレル・プロセスに割り当てられた一意の番号です。この番号には、3 文字になるように、左側に 0 が埋められます。たとえば、3 番目のパラレル・エージェントがファイルを作成する場合、bad\_data\_%a.bad をファイル名として指定した場合、エージェントは bad\_data\_003.bad というファイルを作成します。
- %% は、% に置換されます。ファイル名にパーセント符号が必要な場合、この記号置換が使用されます。

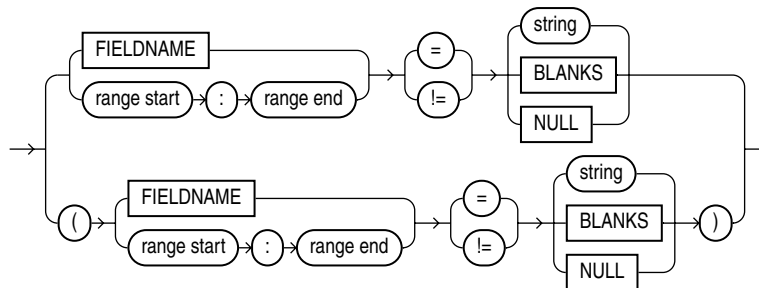
% 文字が検出され、前述の文字以外の文字がその後続く場合、エラーが返されます。

%p または %a を使用しないで出力ファイルに対して一意のファイル名を作成し、外部表にパラレルでアクセス中の場合、出力ファイルが破損するか、エージェントがファイルに書き込みをできないという問題が発生する場合があります。

BADFILE（または DISCARDFILE か LOGFILE）を指定する場合は、ファイル名を指定する必要があります。指定しない場合は、エラーが返されます。ただし、BADFILE（または DISCARDFILE か LOGFILE）を指定しない場合、アクセス・ドライバでは、表の名前に %p を付いたものがファイル名として使用されます。ファイルに対して拡張子がない場合は、デフォルトの拡張子を使用されます。デフォルトの拡張子は、不良ファイルでは .bad、廃棄ファイルでは .dsc、ログ・ファイルでは .log となります。

## condition

`condition` を使用して、定数文字列とレコードのバイト範囲またはフィールドを比較します。比較のソースは、レコードのフィールドまたはレコードのバイト範囲のいずれかです。比較は、バイト単位で実行されます。文字列は、比較のターゲットとして指定すると、データ・ファイルのキャラクタ・セットに変換されます。フィールドに文字以外のデータ型が含まれる場合、データ型変換はフィールド値および文字列のいずれでも実行されません。`condition` の構文は次のとおりです。



### range start : range end

この句を使用してレコードのバイト範囲または文字範囲を記述して条件を指定します。`STRING SIZES ARE` 句に使用する値で、`range` がバイトを示すか、文字を示すかを決定します。`range start` および `range end` は、レコードへのバイト・オフセットまたは文字オフセットです。`range start` は、`range end` 以下である必要があります。文字範囲の検索は、可変幅キャラクタ・セットのデータに対してより固定幅キャラクタ・セットのデータに対しての方が速く処理されます。範囲が、存在しないレコードの一部を指す場合、その範囲を参照しようとするレコードは拒否されます。

---

**注意：** データ・ファイルには、バイナリ・データ（`VARCHAR` などの 2 進数を持つデータ型を含む）および文字データ（可変幅キャラクタ・セットが使用されているか、または文字幅が 1 バイトより大きいデータ）が混在しないようにする必要があります。この場合、アクセス・ドライバは、開始位置の検索時にバイナリ・データを文字データとして処理するため、フィールドの適切な開始位置を検索できない場合があります。

---

フィールドが `NULL` の場合、そのフィールドを `NULL` 以外の値と比較すると、`FALSE` が返されます。

次に、`condition` の使用例を示します。

```
empid != BLANKS
10:13 = 0x00000830
```

```
PRODUCT_COUNT = "MISSING"
```

## field\_definitions 句

field\_definitions 句を使用して、データ・ファイルのフィールドを指定し、レコード内で検索する方法を指定します。

field\_definitions 句を指定しない場合は、次のようになります。

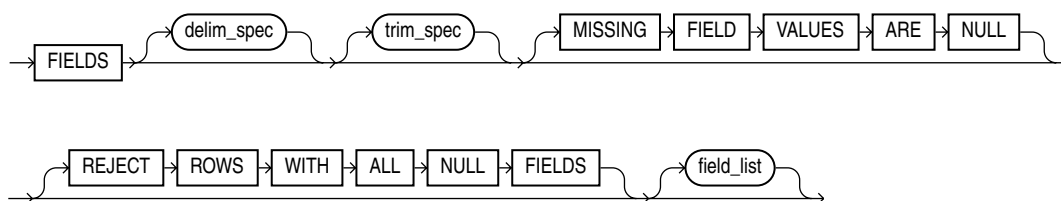
- フィールドは、「,」で区切られる。
- フィールドは、文字型である。
- フィールドの最大長は 255 である。
- データ・ファイルのフィールドの順序は、外部表で定義されたフィールドの順序となる。
- 空白はフィールドから切り捨てられない。

次に、アクセス・パラメータを含まずに作成する外部表の例を示します。その後に、ロード可能なデータ・ファイルのサンプルを示します。

```
CREATE TABLE emp_load (first_name CHAR(15), last_name CHAR(20), year_of_birth CHAR(4))
  ORGANIZATION EXTERNAL (TYPE ORACLE_LOADER DEFAULT DIRECTORY ext_tab_dir LOCATION
                        ('foo.dat'));
```

```
Alvin,Tolliver,1976
Kenneth,Baer,1963
```

field\_definitions 句の構文は次のとおりです。



### delim\_spec 句

delim\_spec 句を使用して、レコード内のすべてのフィールドの終了位置を識別します。すべてのフィールドに指定される delim\_spec は、特定のフィールドに対して field\_list 句の一部として上書きできます。構文の詳細は、12-15 ページの「[delim\\_spec](#)」を参照してください。

## trim\_spec 句

trim\_spec 句を使用して、すべての文字フィールドでデフォルトとして実行される空白の切捨てタイプを指定します。すべてのフィールドに指定される trim\_spec 句は、個々のフィールドに対して trim\_spec 句を指定して上書きできます。構文の詳細は、12-18 ページの「[trim\\_spec](#)」を参照してください。

## MISSING FIELD VALUES ARE NULL

MISSING FIELD VALUES ARE NULL は、レコードのすべてのフィールドに十分なデータがない場合、データ値が欠落しているフィールドが NULL に設定されることを示します。構文の詳細は、12-19 ページの「[MISSING FIELD VALUES ARE NULL](#)」を参照してください。

## REJECT ROWS WITH ALL NULL FIELDS

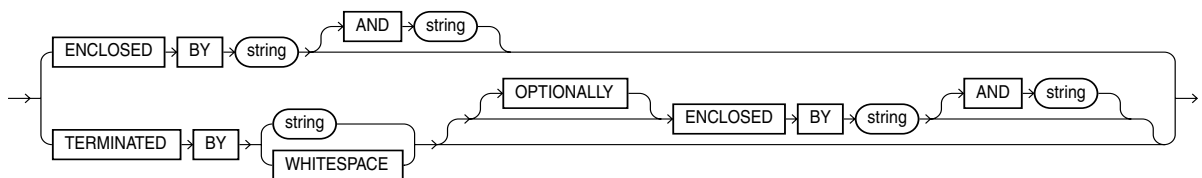
REJECT ROWS WITH ALL NULL FIELDS は、行内で参照されるすべてのフィールドが NULL の場合、その行が外部表にロードされないことを示します。このパラメータを指定しない場合、デフォルト値が使用され、すべてのフィールドが NULL の行が外部表にロードされます。このパラメータの設定は、「reject rows with all null fields」または「rows with all null fields are accepted」としてログ・ファイルに書き込まれます。

## field\_list 句

field\_list 句を使用して、データ・ファイルのフィールドおよびそのデータ型を識別します。構文の詳細は、12-19 ページの「[field\\_list](#)」を参照してください。

## delim\_spec

delim\_spec 句を使用して、フィールドの終了位置（ENCLOSED BY を指定する場合は、開始位置）を検索します。構文は次のとおりです。



ENCLOSED BY を指定すると、アクセス・ドライバで、レコードの現在の位置から最初のデリミタまでの間のすべての空白がスキップされます。現在の位置と最初のデリミタの間のすべての空白が無視されます。次に、アクセス・ドライバでは、2 番目の囲みデリミタが検索されます（または、2 番目のデリミタが指定されていない場合は、最初のデリミタがもう 1 度検索されます）。これら 2 つのデリミタの間にあるすべての文字がフィールド部分とみなされます。

TERMINATED BY *string* を ENCLOSED BY 句で指定する場合、終了記号文字列は、2 番目の囲みデリミタの直後に置く必要があります。2 番目の囲みデリミタと終了デリミタの間の空白はスキップされます。2 つのデリミタの間で空白以外の文字が検索される場合、正しく書式化されていないため行が拒否されます。

ENCLOSED BY 句を使用せずに TERMINATED BY を指定する場合、レコードの現在の位置と次に検索される終了記号文字列の間にあるすべての文字がフィールド部分とみなされます。

OPTIONALLY を指定する場合は、TERMINATED BY も指定する必要があります。OPTIONALLY パラメータによって、ENCLOSED BY デリミタは、両方存在するかまたは両方存在しないかのいずれかであることが示されます。終了デリミタは、ENCLOSED BY デリミタの有無にかかわらず存在する必要があります。OPTIONALLY を指定する場合、アクセス・ドライバは、最初の空白以外の文字までのすべての空白をスキップします。最初の空白以外の文字が検索されると、アクセス・ドライバは、現在の位置に最初の囲みデリミタが含まれているかどうかを確認します。含まれている場合は、アクセス・ドライバによって 2 番目の囲み文字列が検索され、最初の囲みデリミタと 2 番目の囲みデリミタの間のすべての文字がフィールド部分とみなされます。終了デリミタは、2 番目の囲みデリミタの直後に置く必要があります (2 番目の囲みデリミタと終了デリミタの間にオプションで空白を置くことも可能)。最初の空白以外の文字が最初の囲み文字列ではない場合、アクセス・ドライバは終了デリミタを検索します。この場合、先頭 (前にある空白も含めて) から終了デリミタまでのすべての文字がフィールド部分とみなされます。

デリミタが検出された後、レコードの現在の位置は、フィールドの最後のデリミタの後に設定されます。TERMINATED BY WHITESPACE を指定した場合、レコードの現在の位置は、フィールドの後に続くすべての空白の後に設定されます。

レコードの最後のフィールドで終了記号が欠落している場合は、エラーではありません。アクセス・ドライバは、終了記号が検出された場合と同様に処理を行います。2 番目の囲みデリミタが欠落している場合は、エラーとなります。

2 番目の囲みに使用される文字列は、2 番目の囲みを 2 回続けることによって、データ・フィールドに含むことができます。たとえば、フィールドが一重引用符で囲まれる場合、次のような方法で、データ・ファイルに一重引用符を含むことができます。

```
'I don't like green eggs and ham'
```

囲みデリミタを使用せずに、データ・フィールドの終了文字列を引用符で囲む方法はありません。フィールドに終了デリミタを含むことができるのは、フィールド・パーサーが囲みデリミタを検出するまで終了デリミタを検索しないためです。

通常、1 文字の文字列は、複数文字の文字列より速く指定できます。また、固定幅キャラクタ・セットのデータは、可変幅のキャラクタ・セットよりも速く検索できます。

次に、delim\_spec の使用例を示します。

```
TERMINATED BY "|"
ENCLOSED BY "\" TERMINATED BY ","
ENCLOSED BY "START MESSAGE" AND "END MESSAGE"
```



## 例：終了デリミタを含む外部表

次に、終了デリミタが使用されている外部表の例を示します。その後に、ロードが可能なデータ・ファイルのサンプルを示します。

```
CREATE TABLE emp_load (first_name CHAR(15), last_name CHAR(20), year_of_birth CHAR(4))
  ORGANIZATION EXTERNAL (TYPE ORACLE_LOADER DEFAULT DIRECTORY ext_tab_dir
    ACCESS PARAMETERS (FIELDS TERMINATED BY WHITESPACE)
    LOCATION ('foo.dat'));
```

Alvin Tolliver 1976

Kenneth Baer 1963

Mary Dube 1973

## 例：囲みデリミタおよび終了デリミタを含む外部表

次に、囲みデリミタと終了デリミタの両方を使用する外部表の例を示します。2 番目の囲みデリミタと終了記号の間のすべての空白が無視されるのと同様に、終了文字列と最初の囲み文字列の間のすべての空白も無視されます。この例の後に、ロードが可能なデータ・ファイルのサンプルを示します。

```
CREATE TABLE emp_load (first_name CHAR(15), last_name CHAR(20), year_of_birth CHAR(4))
  ORGANIZATION EXTERNAL (TYPE ORACLE_LOADER DEFAULT DIRECTORY ext_tab_dir
    ACCESS PARAMETERS (FIELDS TERMINATED BY "," ENCLOSED BY "(" AND ")")
    LOCATION ('foo.dat'));
```

(Alvin) , (Tolliver), (1976)

(Kenneth), (Baer) , (1963)

(Mary), (Dube) , (1973)

## 例：オプションの囲みデリミタを含む外部表

次に、オプションの囲みデリミタを使用する外部表の例を示します。フィールドの先頭および後続の空白を切り捨てるために、LRTRIMを使用していることに注意してください。この例の後に、ロードが可能なデータ・ファイルのサンプルを示します。

```
CREATE TABLE emp_load (first_name CHAR(15), last_name CHAR(20), year_of_birth CHAR(4))
  ORGANIZATION EXTERNAL (TYPE ORACLE_LOADER DEFAULT DIRECTORY ext_tab_dir
    ACCESS PARAMETERS (FIELDS TERMINATED BY ','
    OPTIONALLY ENCLOSED BY '(' and ') '
    LRTRIM)
    LOCATION ('foo.dat'));
```

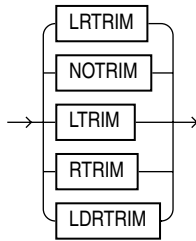
Alvin , Tolliver , 1976

(Kenneth), (Baer), (1963)

( Mary ), Dube , (1973)

## trim\_spec

trim\_spec 句を使用して、空白をテキスト・フィールドの始めから切り捨てるか、終わりに切り捨てるか、またはその両方から切り捨てるかを指定します。空白には、空白文字およびその他の印字されない文字（タブ、LF、改行など）が含まれます。trim\_spec 句の構文は次のとおりです。



フィールドから文字を切り捨てない場合は、NOTRIMを使用します。

フィールドから文字を切り捨てる場合は、LRTRIM、LTRIMおよびRTRIMを使用します。LRTRIMを使用すると、先頭と後続の空白の両方が切り捨てられます。先頭の空白を切り捨てるには、LTRIMを使用します。後続の空白を切り捨てるには、RTRIMを使用します。

SQL\*Loader の切捨て機能との互換性を保つには、LDRTRIMを使用します。次の場合を除いて、NOTRIMと同様です。

- フィールドがデリミタ付きのフィールドではない場合、空白は右から切り捨てられる。
- フィールドが `OPTIONALLY ENCLOSED BY` で指定されたデリミタ付きフィールドで、オプションの囲みが特定のインスタンスで欠落している場合、空白は左から切り捨てられる。

デフォルトは、LDRTRIM です。NOTRIM を指定すると、パフォーマンスが向上します。

trim\_spec 句をフィールド・リストの前に指定して、デフォルトの切捨てをすべてのフィールドに設定できます。trim\_spec がフィールド・リストの前で指定されない場合、LDRTRIM が、デフォルトの切捨て設定となります。デフォルトの切捨ては、個々のフィールドに対して datatype\_spec の一部として上書きできます。

すべてが空白のフィールドに対して切捨てを指定する場合、そのフィールドは NULL に設定されます。

次の例では、すべてのデータが固定長です。ただし、先頭に空白がある文字データはロードできません。この例の後に、ロードが可能なデータ・ファイルのサンプルを示します。

```
CREATE TABLE emp_load (first_name CHAR(15), last_name CHAR(20),
year_of_birth CHAR(4))
  ORGANIZATION EXTERNAL (TYPE ORACLE_LOADER DEFAULT DIRECTORY ext_tab_dir
                        ACCESS PARAMETERS (FIELDS LTRIM)
                        LOCATION ('foo.dat'));
```

Alvin,	Tolliver,1976
Kenneth,	Baer, 1963
Mary,	Dube, 1973

## MISSING FIELD VALUES ARE NULL

MISSING FIELD VALUES ARE NULL は、レコードのすべてのフィールドに十分なデータがない場合、データ値が欠落しているフィールドが NULL に設定されることを示します。MISSING FIELD VALUES ARE NULL を指定せず、レコードのすべてのフィールドに十分なデータがない場合、行は拒否されます。

次の例で、2 番目のレコードは、生まれた年のデータがデータ・ファイルから欠落していても、year\_of\_birth 列に対し NULL に設定されて格納されます。MISSING FIELD VALUES ARE NULL 句をアクセス・パラメータで指定しない場合、year\_of\_birth 列の値が含まれていない 2 番目のレコードが拒否されます。この例の後に、ロードが可能なデータ・ファイルのサンプルを示します。

```
CREATE TABLE emp_load (first_name CHAR(15), last_name CHAR(20), year_of_birth INT)
  ORGANIZATION EXTERNAL (TYPE ORACLE_LOADER DEFAULT DIRECTORY ext_tab_dir
    ACCESS PARAMETERS (FIELDS TERMINATED BY ","
      MISSING FIELD VALUES ARE NULL)
    LOCATION ('foo.dat'));
```

Alvin,Tolliver,1976
Baer,Kenneth
Mary,Dube,1973

## field\_list

field\_list 句を使用して、データ・ファイルのフィールドおよびそのデータ型を識別します。field\_list 句では、次のように評価します。

- フィールドにいずれのデータ型も指定されない場合、データ型は、デリミタなしフィールドでは CHAR(1)、デリミタ付きフィールドでは CHAR(255) である。
- いずれのフィールド・リストも指定されない場合、データ・ファイルのフィールドは外部表と同じ順序である。すべてのフィールドのデータ型は、CHAR(255)。
- いずれのフィールド・リストも指定されず、delim\_spec 句も指定されない場合、データ・ファイルのフィールドは外部表と同じ順序である。すべてのフィールドは、CHAR(255) であり、カンマで終了する。

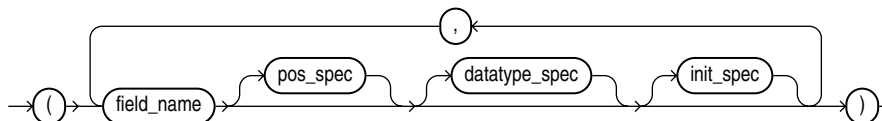
次の例では、field\_list および delim\_spec を含まない外部表の定義を示します。その後に、ロードが可能なデータ・ファイルのサンプルを示します。

```
CREATE TABLE emp_load (first_name CHAR(15), last_name CHAR(20), year_of_birth INT)
  ORGANIZATION EXTERNAL (TYPE ORACLE_LOADER DEFAULT DIRECTORY ext_tab_dir
```

```
ACCESS PARAMETERS (FIELDS TERMINATED BY "|" )  
LOCATION ('foo.dat');
```

```
Alvin|Tolliver|1976  
Kenneth|Baer|1963  
Mary|Dube|1973
```

field\_list 句の構文は次のとおりです。



### field\_name

field\_name は、データ・ファイルのフィールド名を識別する文字列です。文字列が引用符内にない場合、フィールド名は外部表の列名に一致され、大文字になります。

field\_name が問合せで参照される外部表の列名と一致する場合は、このフィールド値が外部表列の値に使用されます。名前が外部表で参照されたいずれの名前にも一致しない場合、フィールドはロードされません。ただし、このフィールドは句の評価（たとえば、WHEN または NULLIF）には使用できます。

### pos\_spec

pos\_spec 句を使用して、レコード内の列の位置を指定します。構文の詳細は、12-21 ページの「[pos\\_spec 句](#)」を参照してください。

### datatype\_spec

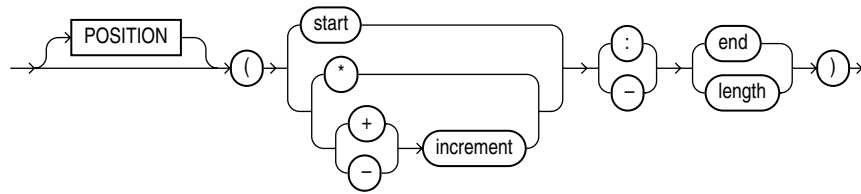
datatype\_spec 句を使用して、フィールドのデータ型を指定します。datatype\_spec が指定されない場合、アクセス・ドライバは、データ型は CHAR(255) であると想定します。構文の詳細は、12-22 ページの「[datatype\\_spec 句](#)」を参照してください。

### init\_spec

init\_spec 句で、フィールドが NULL になる、またはデフォルトの値を設定されるタイミングを指定します。構文の詳細は、12-29 ページの「[init\\_spec 句](#)」を参照してください。

## pos\_spec 句

pos\_spec 句を使用して、レコード内の列の位置を指定します。STRING SIZES ARE IN 句を設定して、pos\_spec がバイト位置と文字位置のどちらを参照するかを決定します。可変幅キャラクタ・セットで文字位置を使用すると、固定幅キャラクタ・セットで文字位置を使用するより大幅に時間がかかります。pos\_spec が文字位置に使用されると、バイナリ文字データとマルチバイト文字データは、同じデータ・ファイルには指定できません。指定した場合の結果は保証されません。pos\_spec 句の構文は次のとおりです。



### start

start パラメータは、レコードの開始位置からフィールドの開始位置までのバイト数または文字数です。前のフィールド位置からの相対ではなく、レコードの絶対位置でフィールドの開始位置を設定します。

★

★ パラメータで、フィールドが前のフィールドの直後のバイトまたは文字から始まることを指定します。これは、可変長フィールドの後に固定長フィールドが続く場合に有効です。このオプションは、レコードの最初のフィールドには使用できません。

### increment

increment パラメータを使用して、フィールドの開始位置を前のフィールドの終了位置からの固定のバイト数または固定の文字数で設定します。\*-increment を使用して、フィールドの開始位置をレコードの現在の位置の前に指定します。\*+increment を使用して、開始位置を現在の位置の後に移動します。

### end

end パラメータを使用して、フィールドの終了バイトをレコード内の絶対バイトまたは絶対文字オフセットで指定します。start を end とともに指定する場合、end は、start より小さくできません。★ または increment を end とともに指定し、開始位置が特定のレコードの end より大きいオフセットと評価される場合、レコードは拒否されます。

## length

length パラメータで、フィールドの終了位置を開始位置からの固定のバイト数または文字数で指定します。開始位置を \* で指定すると、固定長フィールドに有効です。

次に、pos\_spec の使用例を示します。その後、ロードが可能なデータ・ファイルのサンプルを示します。

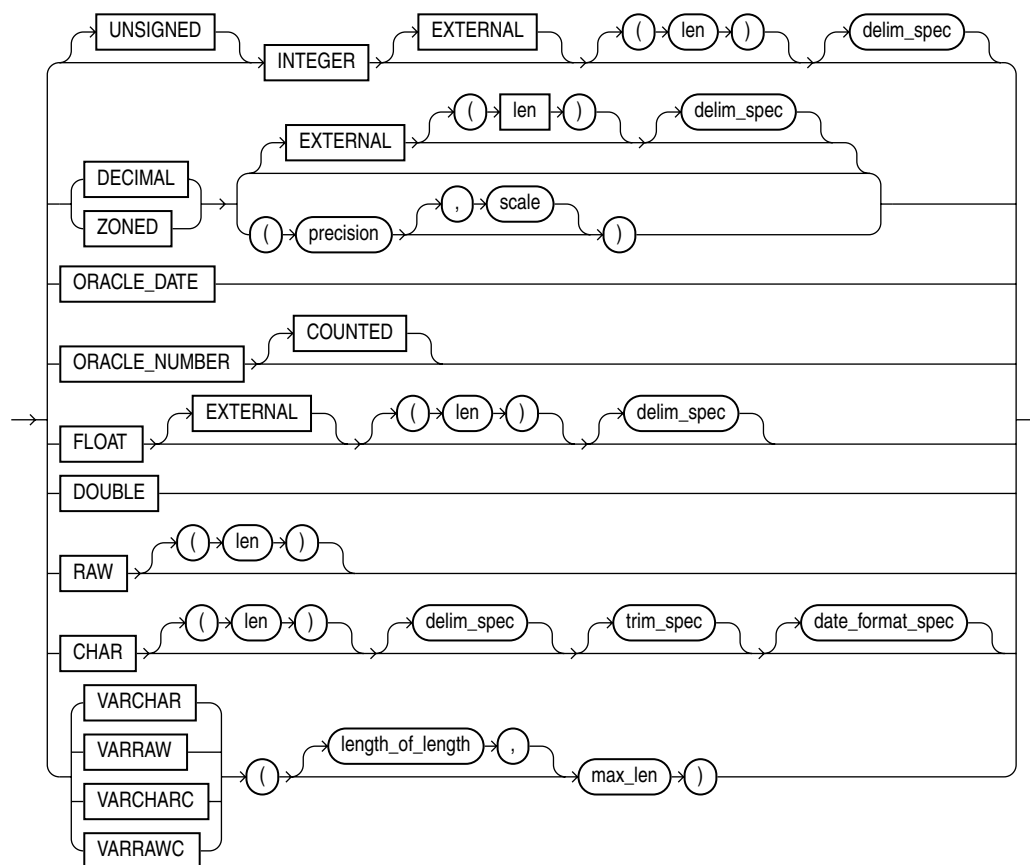
```
CREATE TABLE emp_load (first_name CHAR(15),
                        last_name CHAR(20),
                        year_of_birth INT,
                        phone CHAR(12),
                        area_code CHAR(3),
                        exchange CHAR(3),
                        extension CHAR(4))
ORGANIZATION EXTERNAL
(TYPE ORACLE LOADER
DEFAULT DIRECTORY ext_tab_dir
ACCESS PARAMETERS
(FIELDS RTRIM
 (first_name (1:15) CHAR(15),
  last_name (*:+20),
  year_of_birth (36:39),
  phone (40:52),
  area_code (*-12: +3),
  exchange (*+1: +3),
  extension (*+1: +4)))
LOCATION ('foo.dat'));
```

Alvin	Tolliver	1976415-922-1982
Kenneth	Baer	1963212-341-7912
Mary	Dube	1973309-672-2341

## datatype\_spec 句

データ型がデフォルトと異なる場合、datatype\_spec 句を使用して、データ・ファイルのフィールドのデータ型を指定します。フィールドのデータ型は、外部表内の対応する列のデータ型と異なる場合があります。アクセス・ドライバで必要な変更が行われます。

datatype\_spec 句の構文は次のとおりです。



フィールドのバイト数または文字数が 0 の場合、フィールドは NULL であると想定されます。オプションの DEFAULTIF 句を使用して、フィールドをデフォルトの値に設定するタイミングを指定します。また、オプションの NULLIF 句で、フィールドに対応付けられた列を NULL に設定するタイミングに関するその他の条件を指定します。DEFAULTIF 句または NULLIF 句が真の場合、これらの句を使用すると、データ・ファイルから読み込まれるすべての値が上書きされます。

**参照：** NULLIF および DEFAULTIF の詳細は、12-29 ページの「[init\\_spec 句](#)」を参照してください。

## [UNSIGNED] INTEGER [EXTERNAL] [(len)]

この句を使用して、フィールドを整数として定義します。EXTERNAL を指定する場合、数値は文字列で指定します。EXTERNAL を指定しない場合、数値はバイナリ・フィールドです。2 進整数フィールドの len に対する有効な値は、1、2、4 および 8 です。len が 2 進整数で指定されていない場合、デフォルトの値は、アクセス・ドライバが実行されているプラットフォーム上の `sizeof(int)` の値です。DATA IS {BIG | LITTLE} ENDIAN 句を使用すると、データは格納される前にバイト・スワップされます。

EXTERNAL を指定する場合、len の値は、(STRING SIZES ARE IN BYTES 句または CHARACTERS 句の設定に応じて) バイト数または文字数を数値で指定します。長さを指定しない場合、デフォルト値は 255 になります。

## DECIMAL [EXTERNAL] および ZONED [EXTERNAL]

DECIMAL 句を使用して、フィールドが PACKED 型の 10 進数であることを指定します。

ZONED 句を使用して、フィールドが ZONED 型の 10 進数であることを指定します。

precision フィールドで、数値の桁数を指定します。scale フィールドで、数値の小数点の位置を指定します。つまり、小数点の右側にくる桁数を指定します。scale を指定しない場合、値は 0 となります。

使用中のキャラクタ・セットが EBCDIC ベースか ASCII ベースかによって、ZONED 型の 10 進数には異なるエンコーディング形式があることに注意してください。ソース・データの言語が EBCDIC の場合、そのファイルの ZONED 型の 10 進数は、EBCDIC エンコーディングと一致する必要があります。言語が ASCII ベースの場合、その数値は ASCII エンコーディングと一致する必要があります。

EXTERNAL パラメータを指定する場合、データ・フィールドは、その長さがフィールドの精度と一致する文字列です。

## ORACLE\_DATE

ORACLE\_DATE は、Oracle バイナリ・データ・フォーマットの日付を含むフィールドであることを指定します。これは、Oracle Call Interface (OCI) プログラムでは、DTYDAT データ型として使用される形式です。固定長 7 のフィールドです。

## ORACLE\_NUMBER

ORACLE\_NUMBER は、Oracle 数値書式の数値を含むフィールドであることを指定します。COUNTED を指定しないかぎり、フィールドは固定長 (Oracle 数値フィールドの最大サイズ) です。その場合、フィールドの最初のバイトには残りのフィールドのバイト数が含まれません。

ORACLE\_NUMBER は、固定長 22 バイトのフィールドです。ORACLE\_NUMBER COUNTED フィールドの長さは、カウント・バイト用の 1 バイトに、カウント・バイトで指定されたバイト数を加えた長さです。



## DOUBLE [EXTERNAL]

DOUBLE 句を使用して、フィールドが、アクセス・ドライバが実行されているプラットフォーム上の C 言語の DOUBLE データ型と同じ形式であることを指定します。DATA IS {BIG | LITTLE} ENDIAN 句を使用すると、データは格納される前にバイト・スワップされます。このデータ型は特定のプラットフォーム間では移植できません。

EXTERNAL パラメータを指定する場合、フィールドは、最大長 255 の文字列です。

## FLOAT [EXTERNAL]

FLOAT 句を使用して、フィールドが、アクセス・ドライバが実行されているプラットフォーム上の C 言語の FLOAT データ型と同じ形式であることを指定します。DATA IS {BIG | LITTLE} ENDIAN 句を使用すると、データは格納される前にバイト・スワップされます。このデータ型は特定のプラットフォーム間では移植できません。

EXTERNAL パラメータを指定する場合、フィールドは、最大長 255 の文字列です。

## RAW

RAW 句を使用して、ソース・データがバイナリ・データであることを指定します。RAW フィールドに対する len は常にバイト単位です。RAW フィールドがキャラクタ列にロードされると、列に書き込まれるデータは、RAW フィールドのバイトの 16 進表現となります。

## CHAR

CHAR 句を使用して、フィールドが文字データ型であることを指定します。CHAR フィールドの長さ (len) で、フィールドの最大バイト数または最大文字数を指定します。len は、STRING SIZESAREIN 句の設定に応じて、バイト単位または文字単位になります。

CHAR データ型のフィールドに長さを指定しない場合、フィールドが区切られていないかぎり、フィールド・サイズは 1 になります。

- デリミタ付き CHAR フィールドでは、長さが指定されている場合、その長さが最大長として使用されます。
- 長さが指定されていないデリミタ付き CHAR フィールドでは、デフォルトの 255 バイトが使用されます。
- デリミタ付きで 255 バイトを超える CHAR フィールドには、最大長を指定する必要があります。そうしない場合、データ・ファイルのフィールドが最大長を超えているというエラーを受信します。

date\_format\_spec 句を使用して、指定された形式の日付または時刻がフィールドに含むことを指定します。

次に、CHAR 句の使用例を示します。その後、ロードが可能なデータ・ファイルのサンプルを示します。

```
CREATE TABLE emp_load
```

```

(first_name CHAR(15),
last_name CHAR(20),
hire_date CHAR(10),
resume_file CHAR(500))
ORGANIZATION EXTERNAL
(TYPE ORACLE_LOADER
DEFAULT DIRECTORY ext_tab_dir
ACCESS PARAMETERS (FIELDS TERMINATED BY ","
                    (first_name,
                     last_name,
                     hire_date CHAR(10) DATE_FORMAT DATE MASK "mm/dd/yyyy",
                     resume_file))
LOCATION ('foo.dat'));

```

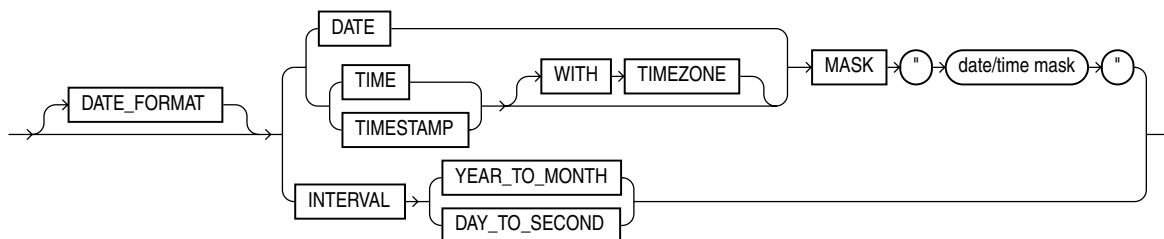
Alvin,Tolliver,12/2/1995,tolliver\_resume.ps

Kenneth,Baer,6/6/1997,KB\_resume.ps

Mary,Dube,1/18/2000,dube\_resume.ps

## date\_format\_spec

date\_format\_spec 句を使用して、特定の形式の日付データまたは時刻データ（あるいはその両方）が文字列フィールドに含まれることを指定します。この情報は、文字フィールドが日付データ型または時刻データ型に変換される場合、および文字列フィールドが日付列にマップされる場合のみに使用されます。date\_format\_spec 句の構文は次のとおりです。



**DATE** DATE 句を使用して、文字列に日付が含まれることを指定します。

**MASK** MASK 句を使用して、データ型に対するデフォルトのグローバリゼーション書式マスクを上書きします。日付マスクを指定しない場合は、データ型に対する適切なグローバリゼーション・パラメータの NLS セッション設定（クライアントの設定ではない）が使用されます。

- DATE データ型の NLS\_DATE\_FORMAT
- TIME データ型の NLS\_TIME\_FORMAT
- TIMESTAMP データ型の NLS\_TIMESTAMP\_FORMAT

- TIME WITH TIME ZONE データ型の NLS\_TIME\_WITH\_TIMEZONE\_FORMAT
- TIMESTAMP WITH TIME ZONE データ型の NLS\_TIMESTAMP\_WITH\_TIMEZONE\_FORMAT

**TIME** TIME 句を使用して、フィールドに書式化された時刻を表す文字列が含まれることを指定します。

**TIMESTAMP** TIMESTAMP 句を使用して、書式化されたタイムスタンプがフィールドに含まれることを指定します。

**INTERVAL** INTERVAL 句を使用して、フィールドに書式化された期間が含まれることを指定します。期間の型は、YEAR TO MONTH または DAY TO SECOND のいずれかです。

## VARCHAR および VARRAW

VARCHAR データ型には、文字データが後に続くバイナリ・カウント・フィールドが含まれます。バイナリ・カウント・フィールドの値は、フィールドのバイト数または文字数のいずれかです。数値が、文字数とバイト数のどちらで解釈されるかを指定する方法の詳細は、12-7 ページの「[STRING SIZES ARE IN](#)」を参照してください。

VARRAW データ型には、バイナリ・データが後に続くバイナリ・カウント・フィールドが含まれます。バイナリ・カウント・フィールドの値は、バイナリ・データのバイト数です。

VARRAW フィールドのデータは、DATA IS...ENDIAN 句の影響を受けません。

オプションの `length_of_length` フィールドは、カウント・フィールドのバイト数です。VARCHAR に対する `length_of_length` の有効な値は、1、2、4 および 8 です。`length_of_length` を指定しない場合、値に 2 が使用されます。カウント・フィールドは、DATA IS...ENDIAN 句で指定するエンディアンと同じエンディアンです。

`max_len` フィールドを使用して、データ・ファイルのフィールドのインスタンスの最大サイズを指定します。VARRAW フィールドでは、`max_len` はバイト数です。VARCHAR フィールドでは、`max_len` は、STRING SIZES ARE IN 句の設定に応じて、文字数またはバイト数のいずれかになります。

次に、VARCHAR および VARRAW の使用例を示します。カウント・バイトのバイナリ値および RAW データの値は、データ・ファイルに各バイナリ・バイトにつき 2 文字がイタリック体で示されています。

```
CREATE TABLE emp_load
    (first_name CHAR(15),
     last_name CHAR(20),
     resume CHAR(2000),
     picture RAW(2000))
ORGANIZATION EXTERNAL
(TYPE ORACLE_LOADER
 DEFAULT DIRECTORY ext_tab_dir
 ACCESS PARAMETERS
  (FIELDS (first_name VARCHAR(2,12),
```

```

        last_name VARCHAR(2,20),
        resume VARCHAR(4,10000),
        picture VARRAW(4,100000)))
LOCATION ('foo.dat'));

```

```

0005Alvin0008Tolliver0000001DAlvin Tolliver's Resume etc.
0000001013f4690a30bc29d7e40023ab4599ffff

```

## VARCHARC および VARRAWC

VARCHARC データ型には、文字データが後に続く文字カウント・フィールドが含まれます。カウント・フィールドの値は、フィールドのバイト数または文字数のいずれかです。数値が、文字数とバイト数のどちらで解釈されるかを指定する方法の詳細は、12-7 ページの「[STRING SIZES ARE IN](#)」を参照してください。オプションの `length_of_length` は、長さが文字とバイトのどちらで解釈されるかに応じて、VARCHARC に対するカウント・フィールドのバイト数または文字数のいずれかになります。

VARCHARC に対する `length_of_lengths` の最大値は、文字列のサイズが文字単位の場合は 10 で、文字列のサイズがバイト単位の場合は 20 です。`length_of_length` のデフォルトの値は 5 です。

VARRAWC データ型には、バイナリ・データが後に続くキャラクタ・カウント・フィールドが含まれます。カウント・フィールドの値は、バイナリ・データのバイト数です。`length_of_length` は、カウント・フィールドのバイト数です。

`max_len` フィールドを使用して、データ・ファイルのフィールドのインスタンスの最大サイズを指定します。VARRAWC の場合は、`max_len` はバイト数です。VARCHARC フィールドの場合は、`max_len` は、STRING SIZES ARE IN 句の設定に応じて、文字数またはバイト数のいずれかになります。

次に、VARCHARC および VARRAWC の使用例を示します。`picture` フィールドの長さは 0 です。これは、このフィールドが NULL に設定されていることを意味します。

```

CREATE TABLE emp_load
    (first_name CHAR(15),
     last_name CHAR(20),
     resume CHAR(2000),
     picture RAW (2000))
ORGANIZATION EXTERNAL
(TYPE ORACLE_LOADER
 DEFAULT DIRECTORY ext_tab_dir
 ACCESS PARAMETERS
    (FIELDS (first_name VARCHAR(5,12),
             last_name VARCHAR(2,20),
             resume VARCHAR(4,10000),
             picture VARRAWC(4,100000)))
LOCATION ('foo.dat'));

```

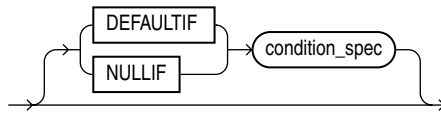
```

00007William05Ricca0035Resume for William Ricca is missing0000

```

## init\_spec 句

init\_spec 句を使用して、フィールドを NULL またはデフォルト値に設定するタイミングを指定します。init\_spec 句の構文は次のとおりです。



NULLIF 句および DEFAULTIF 句は、フィールドに各 1 回のみ指定できます。これらの句を使用して、次の処理を実行できます。

- NULLIF condition\_spec を指定し、真と評価された場合、フィールドは NULL に設定されます。
- DEFAULTIF condition\_spec を指定し、真と評価された場合、フィールドの値はデフォルトの値に設定されます。デフォルトの値は、フィールドのデータ型によって次のように異なります。
  - 文字データ型の場合は、デフォルトの値は空の文字列
  - 数値データ型の場合は、デフォルトの値は 0
  - 日付データ型の場合は、デフォルトの値は NULL
- NULLIF 句および DEFAULTIF 句の両方をフィールドに指定する場合、まず NULLIF 句が評価され、DEFAULTIF 句は、NULLIF 句が偽と評価された場合のみに評価されます。



# 第Ⅳ部

---

## その他のユーティリティ

第Ⅳ部には、次の章が含まれます。

### [第13章「DBVERIFY: オフライン・データベース検査ユーティリティ」](#)

この章では、オフライン・データベース検査ユーティリティである DBVERIFY の使用方法について説明します。

### [第14章「DBNEWID ユーティリティ」](#)

この章では、DBNEWID ユーティリティを使用してデータベースの名前または ID（あるいはその両方）を変更する方法について説明します。

### [第15章「メタデータ API の使用」](#)

この章では、データベース・オブジェクトに対するメタデータの完全な表現の抽出および処理が可能なメタデータ API について説明します。





---

## DBVERIFY: オフライン・データベース検査ユーティリティ

DBVERIFY は外部コマンド・ユーティリティで、オフライン・データベース上で物理データ構造に対する整合性チェックを実行します。チェックの対象となるのは、バックアップ・ファイルおよびオンライン・ファイル（またはファイルの一部）です。DBVERIFY を使用するのには、バックアップ・データベース（またはデータ・ファイル）をリストアする前にそれが有効であることを確認する場合です。また、データ破損の問題が発生した場合に診断機能としても使用します。

DBVERIFY はオフライン・データベースに対して実行できるため、整合性チェックが非常に高速に行えます。

DBVERIFY によるチェックは、キャッシュ管理（データ・ブロック）ブロックのみに制限されています。DBVERIFY は、データ・ファイルでのみ使用します。制御ファイルまたは REDO ログに対しては機能しません。

DBVERIFY には、2 つのコマンドライン・インタフェースがあります。最初のインタフェースでは、確認の対象として単一データ・ファイルのディスク・ブロックを指定します。次のインタフェースでは、確認の対象としてセグメントを指定します。次の項では、これらのインタフェースの記述について説明します。

- [DBVERIFY を使用した単一データ・ファイルのディスク・ブロックの検査](#)
- [DBVERIFY を使用したセグメントの検査](#)

---

**注意：** DBVERIFY の起動に使用するコマンドはご使用のオペレーティング・システムによって異なります（たとえば、Sun/Sequent システムでは dbv）。詳細は、ご使用のオペレーティング・システム固有の Oracle ドキュメントを参照してください。

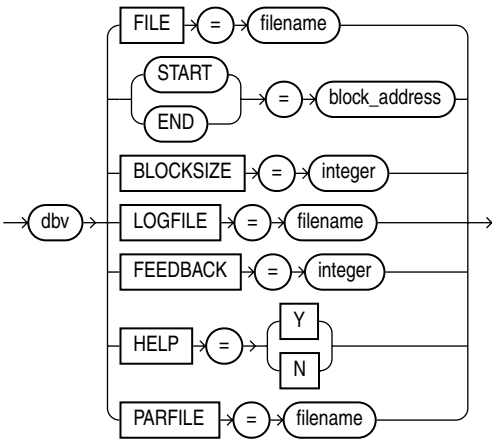
---

# DBVERIFY を使用した単一データ・ファイルのディスク・ブロックの検査

このモードでは、DBVERIFY によって、単一データ・ファイルの 1 つ以上のディスク・ブロックがスキャンされ、ページのチェックが実行されます。

## 構文

単一データ・ファイルのディスク・ブロックを検査するときに使用する DBVERIFY の構文は次のとおりです。



## パラメータ

パラメータの詳細は次のとおりです。

パラメータ	説明
FILE	検査するデータベース・ファイル名。
START	検査する最初のブロック・アドレス。ブロック・アドレスは、Oracle ブロックで指定します (オペレーティング・システム・ブロックではありません)。START を指定しないと、ファイル内の最初のブロックが DBVERIFY によってデフォルト設定されます。
END	検査する最後のブロック。END を指定しないと、ファイル内の最後のブロックが DBVERIFY によってデフォルト設定されます。

パラメータ	説明
BLOCKSIZE	BLOCKSIZE は、検査するファイルのブロック・サイズが 2KB でない場合にのみ指定します。ファイルのブロック・サイズが 2KB でない場合、BLOCKSIZE を指定しないと DBV-00103 のエラーが発生します。
LOGFILE	ログ情報を書き込むファイルを指定します。デフォルトでは、端末画面への出力となります。
FEEDBACK	進捗画面が端末に表示され、DBVERIFY の実行中に検証されたページ数 $n$ が 1 つのピリオド (.) で示されます。 $n=0$ と設定すると、進捗画面は表示されません。
HELP	オンライン・ヘルプを表示します。
PARFILE	使用するパラメータ・ファイル名を指定します。フラット・ファイルの DBVERIFY パラメータには、複数の値を格納できます。これによって、パラメータ・ファイルをカスタマイズして、異なるタイプのデータ・ファイルの処理、およびデータ・ファイルに固有の整合性チェックの実行ができます。

## コマンドライン・インタフェース

次に、DBVERIFY をこのモードで起動するコマンドライン・インタフェースの使用例を示します。

```
% dbv FILE=t_db1.dbf FEEDBACK=100
```

## DBVERIFY のサンプル出力

次に示すのは、ファイル `t_db1.f` の検査の出力です。フィードバック・パラメータに 100 が指定されているため、100 ページの処理が行われるたびにピリオドが 1 つ表示されます。

```
% dbv FILE=t_db1.dbf FEEDBACK=100
```

```
DBVERIFY: Release 9.2.0.1.0 - Production on Wed Feb 27 13:55:26 2002
```

```
(c) Copyright 2002 Oracle Corporation. All rights reserved.
```

```
DBVERIFY - Verification starting : FILE = t_db1.dbf
```

```
.....
```

```
DBVERIFY - Verification complete
```

```
Total Pages Examined          : 9216
```

```
Total Pages Processed (Data) : 2044
```

```
Total Pages Failing   (Data) : 0
Total Pages Processed (Index): 733
Total Pages Failing   (Index): 0
Total Pages Empty      : 5686
Total Pages Marked Corrupt : 0

Total Pages Influx      : 0
```

### 注意：

- Pages = ブロック。
- Total Pages Examined = ファイルのブロック数。
- Total Pages Processed = 検査されたブロック数（書式化されたブロック）。
- Total Pages Failing (Data) = データ・ブロック・チェック・ルーチンで違反したブロック数。
- Total Pages Failing (Index) = 索引ブロック・チェック・ルーチンで違反したブロック数。
- Total Pages Marked Corrupt = キャッシュ・ヘッダーが無効で、DBVERIFY がブロック型を識別できないブロック数。
- Total Pages Influx = 読み込みおよび書き込みが同時に行われるブロック数。DBVERIFY の実行時にデータベースがオープンされている場合は、一貫性を維持したイメージの取得のために DBVERIFY によってブロックが複数回読み込まれます。ただし、データベースがオープンされているため、同時に読み込みおよび書き込みが行われるブロックがあります (INFLUX)。DBVERIFY では、絶え間なく変化するページの一貫性のあるイメージは取得できません。

## DBVERIFY を使用したセグメントの検査

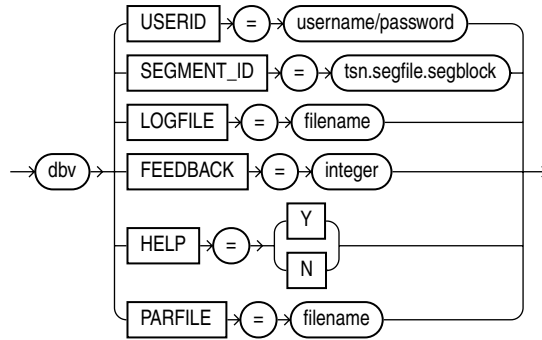
このモードでは、DBVERIFY を使用して表セグメントまたは索引セグメントの検査を指定できます。確認中のセグメントに行連鎖ポインタがあることを確認します。

このモードでは、検査するセグメント（データ・セグメントまたは索引セグメント）を指定する必要があります。また、セグメントについての情報をデータベースから取得するため、SYSDBA 権限でデータベースにログインする必要があります。

このモードでは、セグメントはロックされます。1 つの索引セグメントを指定する場合、親表はロックされます。IOT など、一部の索引には親表はありません。

## 構文

セグメントを検査するときの DBVERIFY の構文は次のとおりです。



## パラメータ

パラメータの詳細は次のとおりです。

パラメータ	説明
USERID	ユーザー名およびパスワードを指定します。
SEGMENT_ID	検査するセグメントを指定します。tsn、segfile および segblock を識別するには、USER_TABLES、USER_SEGMENTS などの適切なデータ・ディクショナリ表を結合して問合せを実行します。
LOGFILE	ログ情報を書き込むファイルを指定します。デフォルトでは、端末画面への出力となります。
FEEDBACK	進捗画面が端末に表示され、DBVERIFY の実行中に検証されたページ数 <i>n</i> が 1 つのピリオド (.) で示されます。 <i>n</i> = 0 と設定すると、進捗画面は表示されません。
HELP	オンライン・ヘルプを表示します。
PARFILE	使用するパラメータ・ファイル名を指定します。フラット・ファイルの DBVERIFY パラメータには、複数の値を格納できます。これによって、パラメータ・ファイルをカスタマイズして、異なるタイプのデータ・ファイルの処理、およびデータ・ファイルに固有の整合性チェックの実行ができます。

## コマンドライン・インタフェース

次に、DBVERIFY をこのモードで起動するコマンドライン・インタフェースの使用例を示します。

```
dbv USERID=username/password SEGMENT_ID=tsn.segfile.segblock
```

---

## DBNEWID ユーティリティ

DBNEWID ユーティリティは、オペレーショナル・データベースの DBID および DBNAME を変更できるデータベース・ユーティリティです。

この章の内容は、次のとおりです。

- DBNEWID ユーティリティとは
- DBID および DBNAME の変更による影響
- データベースの DBID および DBNAME の変更
- DBNEWID ユーティリティの構文

## DBNEWID ユーティリティとは

DBNEWID ユーティリティの導入以前は、データベースのコピーを手動で作成し、制御ファイルを再作成してデータベースに新しいデータベース名 (DBNAME) を指定することができました。ただし、データベースに新しい識別子 (DBID) を指定することはできません。DBID は、データベース用の内部的な一意の識別子です。Recovery Manager では DBID によってデータベースが区別されるため、シード・データベースおよび手動でコピーされたデータベースは、同じ Recovery Manager リポジトリに登録できません。DBNEWID ユーティリティを使用すると、次のいずれかを変更することによって、この問題を解決できます。

- データベースの DBID のみ
- データベースの DBNAME のみ
- データベースの DBNAME および DBID の両方

## DBID および DBNAME の変更による影響

データベースの DBID の変更は、重要な処理です。データベースの DBID を変更すると、そのデータベースのすべての古いバックアップおよびアーカイブ・ログが使用できなくなります。DBID の変更後、RESETLOGS オプションを指定してデータベースをオープンする必要があります。これによって、オンライン REDO ログが再作成され、この順序番号が 1 にリセットされます (詳細は、『Oracle9i データベース管理者ガイド』を参照)。このため、DBID の変更直後に、データベース全体のバックアップを取る必要があります。

DBID を変更せずに DBNAME を変更する場合は、RESETLOGS オプションを指定してデータベースをオープンする必要がないため、データベースのバックアップおよびアーカイブ・ログは無効になりません。ただし、DBNAME を変更すると、これらが無効になります。データベース名の変更後、新しい名前を反映させるには DB\_NAME 初期化パラメータを変更する必要があります。また、Oracle パスワード・ファイルの再作成が必要な場合もあります。制御ファイルの古い (データベース名を変更する前の) バックアップをリストアする場合、データベース名を変更する前の初期化パラメータ・ファイルおよびパスワード・ファイルを使用する必要があります。

## データベースの DBID および DBNAME の変更

この項の内容は、次のとおりです。

- [DBID およびデータベース名の変更](#)
- [データベース名のみの変更](#)
- [DBID の変更操作のトラブルシューティング](#)
- [データベース名の変更操作のトラブルシューティング](#)



## DBID およびデータベース名の変更

この項では、データベースの DBID を変更する方法について説明します。オプションで、データベース名も変更できます。

1. データベース全体のリカバリ可能なバックアップがあることを確認します。
2. ターゲット・データベースがマウント状態であること、またそのデータベースがマウント前に一貫性を維持した状態で停止されていることを確認します。次に例を示します。

```
SHUTDOWN IMMEDIATE
STARTUP MOUNT
```

3. コマンドラインで DBNEWID ユーティリティを起動し、SYSDBA 権限を持つ有効なユーザーを指定します。次に例を示します。

```
% nid TARGET=SYS/oracle@test_db
```

DBID に加えてデータベース名も変更するには、DBNAME パラメータを指定します。この例では、データベース名を test\_db2 に変更します。

```
% nid TARGET=SYS/oracle@test DBNAME=test_db2
```

DBNEWID ユーティリティを使用して、データ・ファイルおよび制御ファイルへの入出力を試行する前に、これらのファイルのヘッダーで妥当性チェックを実行します。妥当性チェックが正常に行われると、操作の確認が求められ（ログ・ファイルを指定した場合、確認は求められません）、各データ・ファイル（NORMAL モードでオフラインされたデータ・ファイルおよび読み込み専用データ・ファイルを含む）の DBID が変更されます。これらの処理の完了後、DBNEWID ユーティリティが終了します。データベースはマウントされた状態のままですが、使用できません。次に例を示します。

```
DBNEWID: Release 9.2.0.1.0
```

```
(c) Copyright 2002 Oracle Corporation. All rights reserved.
```

```
Connected to database TEST_DB (DBID=3942195360)
```

```
Control Files in database:
```

```
  /oracle/dbs/cf1.f
  /oracle/dbs/cf2.f
```

```
Change database id of database SOLARIS? (Y/[N]) => y
```

```
Proceeding with operation
```

```
Datafile /oracle/dbs/tbs_01.f - changed
Datafile /oracle/dbs/tbs_02.f - changed
Datafile /oracle/dbs/tbs_11.f - changed
Datafile /oracle/dbs/tbs_12.f - changed
Datafile /oracle/dbs/tbs_21.f - changed
```

```
New DBID for database TEST_DB is 3942196782.  
All previous backups and archived redo logs for this database are unusable  
Proceed to shutdown database and open with RESETLOGS option.  
DBNEWID - Database changed.
```

妥当性チェックが正常に行われなかった場合、DBNEWID ユーティリティは終了し、ターゲット・データベースは変更されません。データベースをオープンし、エラーを修正した後で、DBNEWID ユーティリティの操作を再開するか、または DBID を変更せずに継続してデータベースを使用することができます。

4. DBNEWID ユーティリティによって DBID が正常に変更された後で、データベースを停止します。次に例を示します。

```
SHUTDOWN IMMEDIATE
```

5. データベースをマウントします。次に例を示します。

```
STARTUP MOUNT
```

6. RESETLOGS モードでデータベースをオープンし、通常の使用を再開します。次に例を示します。

```
ALTER DATABASE OPEN RESETLOGS;
```

データベースの新しいバックアップを取ります。オンライン REDO ログをリセットしたため、現行のデータベースでは古いバックアップとアーカイブ・ログは使用できません。

## データベース名のみの変更

この項では、DBID を変更せずにデータベース名を変更する方法について説明します。

1. データベース全体のリカバリ可能なバックアップがあることを確認します。
2. ターゲット・データベースがマウント状態であること、またそのデータベースがマウント前に一貫性を維持した状態で停止されていることを確認します。次に例を示します。

```
SHUTDOWN IMMEDIATE
```

```
STARTUP MOUNT
```

3. コマンドラインでユーティリティを起動し、SYSDBA 権限を持つ有効なユーザーを指定します。DBNAME パラメータと SETNAME パラメータの両方を指定する必要があります。この例では、データベース名を test\_db2 に変更します。

```
% nid TARGET=SYS/oracle@test_db DBNAME=test_db2 SETNAME=YES
```

DBNEWID ユーティリティを使用して、制御ファイル（データ・ファイルではなく）への入出力を試行する前に、ファイルのヘッダーで妥当性チェックを実行します。妥当性

チェックが正常に行われると、操作の確認が求められ、制御ファイル内のデータベース名が変更されます。これらの処理の完了後、DBNEWID ユーティリティが終了します。DBNEWID ユーティリティが正常に終了した後、データベースはマウントされた状態のままですが、使用できません。

DBNEWID: Release 9.2.0.1.0

(c) Copyright 2002 Oracle Corporation. All rights reserved.

Connected to database TEST\_DB (DBID=3942196782)

Control Files in database:

/oracle/dbs/cf1.f

/oracle/dbs/cf2.f

Change database name of database TEST\_DB to TEST\_DB2? (Y/[N]) => Y

Proceeding with operation

Database name changed from TEST\_DB to TEST\_DB2 - database needs to be shutdown.  
Modify parameter file and generate a new password file before restarting.

DBNEWID - Successfully changed database name

妥当性チェックが正常に行われなかった場合、DBNEWID ユーティリティは終了し、ターゲット・データベースは変更されません。データベースをオープンし、エラーを修正した後で、DBNEWID ユーティリティの操作を再開するか、またはデータベース名を変更せずに継続してデータベースを使用することができます。

4. データベースを停止します。次に例を示します。

SHUTDOWN IMMEDIATE

5. 初期化パラメータ・ファイルの DB\_NAME 初期化パラメータを新しいデータベース名に設定します。
6. 新しいパスワード・ファイルを作成します。
7. データベースを起動し、通常の使用を再開します。次に例を示します。

STARTUP

## DBID の変更操作のトラブルシューティング

DBNEWID ユーティリティによって妥当性チェックが正常に行われたが、DBID の変更中にエラーが検出された場合、DBNEWID ユーティリティは停止し、データベースは変更中のままになります。この場合、DBNEWID ユーティリティの操作を完了するか、または元に戻さないかぎり、データベースをオープンできません。DBNEWID ユーティリティによって操作の状態を示すメッセージが表示されます。

操作を続行するか元に戻す前に、エラーの原因を解決します。データベース全体を最新のバックアップからリストアして、DBNEWID ユーティリティの起動前の時点までリカバリを実行する以外に解決策がない場合もあります。そのため、DBNEWID ユーティリティの実行前に最新のバックアップを使用可能にしておくことが重要です。

DBID の変更操作を元に戻さずに続行する場合は、元のコマンドを再実行します。DBNEWID ユーティリティが再開され、すべてのデータ・ファイルおよび制御ファイルに新しい DBID が指定されるまで変更操作が続行されます。この時点では、データベースはマウントされたままです。RESETLOGS オプションを使用してデータベースをオープンする前に、データベースを停止し、再度マウントする必要があります。

DBNEWID ユーティリティの操作を元に戻した場合、DBNEWID ユーティリティによって実行されたすべての変更が元に戻され、データベースはマウントされた状態のままになります。

停止した DBID の変更操作を元に戻すには、REVERT キーワードを指定して DBNEWID ユーティリティを再実行します。次に例を示します。

```
% nid TARGET=SYS/oracle REVERT=YES LOGFILE=$HOME/nid.log
```

## データベース名の変更操作のトラブルシューティング

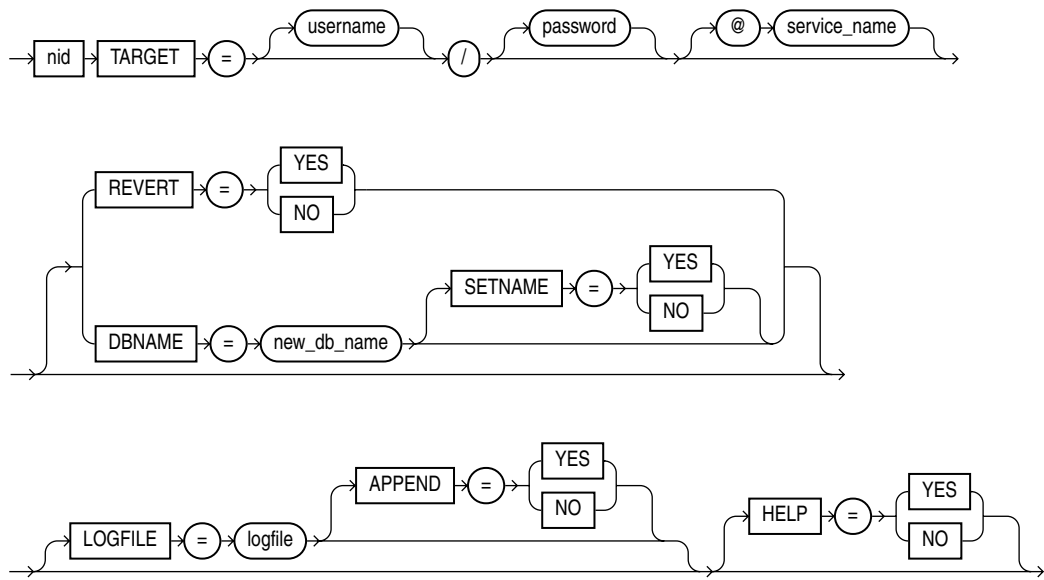
データベース名の変更（DBID は変更しない）を指定した場合、妥当性チェックのプロセスは DBID の変更と同じです。ただし、制御ファイルのみが DBNEWID ユーティリティによってチェックされます。データ・ファイルは読み込まれません。妥当性チェックで問題が発生した場合、データベースはマウントされたままになります。

妥当性チェックを正常に終了することはできますが、実際のデータベース名は変更されません。発生の可能性がある障害の例は、データベース内に存在する制御ファイルの数によって、次のとおり異なります。

- 1 つ以上の制御ファイルが存在し、最初の制御ファイルで DBNEWID ユーティリティに障害が発生した場合、制御ファイルのデータベース名は変更されません。操作を再試行するか、またはデータベースをオープンして通常のデータベースの使用を再開します。
- 複数の制御ファイルが存在し、2 つ目以降の制御ファイルで DBNEWID ユーティリティに障害が発生した場合、古い DBNAME を含む制御ファイルと新しい DBNAME を含む制御ファイルが混在します。この場合、すべての CONTROL\_FILES の場所に最初に変更された制御ファイルを手動でコピーするか、またはすべての CONTROL\_FILES の場所に変更されていない制御ファイルをコピーして変更を元に戻す必要があります。

# DBNEWID ユーティリティの構文

次の図に、DBNEWID ユーティリティの構文を示します。



## パラメータ

表 14-1 に、DBNEWID ユーティリティの構文のパラメータを示します。

表 14-1 DBNEWID ユーティリティのパラメータ

パラメータ	説明
TARGET	データベースへの接続に使用するユーザー名およびパスワードを指定します。ユーザーには、SYSDBA 権限が必要です。オペレーティング・システム認証を使用している場合、スラッシュ (/) を使用して続けることができます。環境変数 \$ORACLE_HOME および \$ORACLE_SID が正しく設定されていない場合、ターゲット・データベースへの接続にセキュリティで保護されたサービス (IPC または BEQ) を指定できます。DBNEWID ユーティリティを起動する場合は、常にターゲット・データベースを指定する必要があります。

表 14-1 DBNEWID ユーティリティのパラメータ（続き）

パラメータ	説明
REVERT	YES（デフォルトは NO）を指定すると、失敗した DBID の変更が元に戻ります。ターゲット・データベース上で DBID の変更操作が進行していない場合は、ユーティリティによってエラーが出力されます。DBID の変更が正常に完了した場合は、元に戻せません。REVERT=YES は、DBID の変更が正常に行われなかった場合にのみ有効です。
DBNAME=new_db_name	データベース名を変更します。データベースの DBID および DBNAME は同時に変更できます。DBNAME のみを変更するには、SETNAME パラメータも指定します。
SETNAME	YES（デフォルトは NO）を指定すると、DBNEWID ユーティリティによってデータベースの名前のみが変更され、DBID は変更されません。SETNAME=YES を指定すると、ユーティリティによってターゲット・データベースの制御ファイルのみに書き込まれます。
LOGFILE=logfile	DBNEWID ユーティリティによってメッセージが書き込まれるファイルを指定します。デフォルトでは、古いログに上書きされます。ログ・ファイルを指定すると、DBNEWID ユーティリティによって確認が求められません。
APPEND	YES（デフォルトは NO）を指定すると、既存のログ・ファイルにログの出力が追加されます。
HELP	YES（デフォルトは NO）を指定すると、DBNEWID ユーティリティの構文オプションのリストが出力されます。

制限事項および使用上の注意

DBNEWID ユーティリティには次の制限事項があります。

- このユーティリティは、UNIX および Windows NT のオペレーティングシステムでのみ使用可能です。
- nid 実行可能ファイルを実行するにはデータ・ファイルおよび制御ファイルに直接アクセスする必要があるため、この実行可能ファイルは Oracle 所有者が所有し、実行する必要があります。他のユーザーがこのユーティリティを実行する場合は、そのユーザー ID をデータ・ファイルおよび制御ファイルの所有者に設定します。
- DBNEWID ユーティリティを使用するには、ローカル接続を介してデータベースのデータ・ファイルに直接アクセスする必要があります。DBNEWID ユーティリティでは、ネット・サービス名は指定できますが、非ローカル・データベースの DBID は変更できません。
- データベースの DBID を変更するには、データベースがマウントされており、マウント前に一貫性を維持した状態で停止されている必要があります。Oracle Real Application

Clusters データベースの場合は、データベースを NOPARALLEL モードでマウントする必要があります。

- DBID の変更後は、RESETLOGS オプションを使用してデータベースをオープンする必要があります。データベース名のみの変更後は、RESETLOGS オプションを使用してオープンする必要はありません。
- DBNEWID ユーティリティの実行中に、データベースに対して他のプロセスを実行することはできません。他のセッションがデータベースを停止して起動した場合、DBNEWID ユーティリティは異常終了します。
- すべてのオンライン・データ・ファイルは、リカバリが不要な、一貫性を維持した状態である必要があります。
- NORMAL モードでオフラインされたデータ・ファイルは、アクセスおよび書込みが可能である必要があります。そうでない場合は、DBNEWID ユーティリティを起動する前に、これらのファイルを削除する必要があります。
- すべての読み込み専用表領域は、DBNEWID ユーティリティを起動する前に、オペレーティング・システム・レベルでアクセスおよび書込みが可能である必要があります。これらの表領域を書込み可能にできない場合（たとえば、表領域が CD-ROM にある場合）は、トランスポータブル表領域機能を使用して、DBNEWID ユーティリティを起動する前に、表領域を書込み可能な領域にトランスポートする必要があります（詳細は、『Oracle9i データベース管理者ガイド』を参照）。
- DBID のみを変更する場合は、REVERT のみ指定できます。

## DBNEWID ユーティリティの使用例

### DBID のみの変更

次の例では、オペレーティング・システム認証を使用して接続し、DBID のみを変更します。

```
% nid TARGET=/  

```

### DBID およびデータベース名の変更

次の例では、ユーザー SYS として接続し、DBID を変更して、データベース名を test2 に変更します。

```
% nid TARGET=SYS/oracle@test1 DBNAME=test2  

```

### データベース名のみの変更

次の例では、ユーザー SYSTEM として接続し、データベース名のみを変更して、出力用のログ・ファイルを指定します。

```
% nid TARGET=SYSTEM/manager@test2 DBNAME=test3 SETNAME=YES LOGFILE=dbid.out  

```





---

## メタデータ API の使用

この章では、データベース・オブジェクトに対するメタデータの完全な表現の抽出および処理が可能なメタデータ Application Programming Interface (API) について説明します。この章の内容は、次のとおりです。

- [メタデータ API の概要](#)
- [メタデータ API の実装方法](#)
- [DBMS\\_METADATA プログラム・インタフェース](#)
- [DBMS\\_METADATA ブラウザ・インタフェース](#)
- [メタデータ API の例](#)

## メタデータ API の概要

メタデータ API によって、次の作業を集中的、かつ単純で柔軟な方法で実行できます。

- XML または作成 DDL のいずれかとしての、データベース・オブジェクト（メタデータ）の完全な定義の抽出
- 業界標準の Extensible Stylesheet Language Transformation（XSLT）を介したメタデータの変換
- SQL DDL の生成によるデータベース・オブジェクトの再作成

Oracle9i リリース 1（9.0.1）以降では、インスタンスが動作中の場合常にメタデータ API が使用可能です。メタデータ API は、Oracle Lite では使用できません。

## 以前の方法によるメタデータの抽出

オブジェクトのメタデータは、正規化された状態でデータベース・ディクショナリ内に分散されます。以前のリリースでは、まずディクショナリ内でオブジェクトのメタデータが表現されている方法および位置を理解し、次に、オブジェクトの完全な表現の抽出のために、複数の問合せを発行する必要がありました。メタデータが抽出されると、通常、次のタスクを実行します。

1. オブジェクト表領域の変更、列データ型の変更、オブジェクト所有者の変更など、メタデータをなんらかの方法で変換します。
2. ソース・データベースまたは他のデータベースで実行するために、メタデータを SQL DDL テキストに変換します。

Oracle9i リリース 1（9.0.1）以前は、これらの手順を支援する方法はありませんでした。

## メタデータ API の構成要素

メタデータ API の基礎となっているのは、一連のユーザー定義型（UDT）および対応するオブジェクト・ビューで構成される Oracle データベース・ディクショナリのオブジェクト・モデルです。UDT によって各オブジェクト・クラスのメタデータの集計操作が実行され、オブジェクト・ビューによって UDT の属性がディクショナリ内の適切なベース・リレーショナル表にマップされます。メタデータ API によって、これらのオブジェクト・ビューに対する問合せが生成され、集計済データベース・オブジェクト定義が取得されます。

これらの問合せの結果は、XML SQL Utility（XSU）によって XML 文書に変換されます。この機能も、Oracle9i リリース 1（9.0.1）で導入されました。コール元が DDL の出力を要求すると、メタデータ API は、Oracle サーバーに統合された XML Parser および XSL Processors を使用して、XML 文書を作成用 DDL に変換します。

## メタデータ API の機能

メタデータ API の機能は、次のとおりです。

- 詳細なプログラム制御または簡単な参照を実現するための、強力な PL/SQL インタフェースを提供します。
- SQL の CREATE 文で作成可能なすべてのディレクトリ・オブジェクトに対して、完全な集計済データベース定義の取得をサポートします。オブジェクト型の完全なリストについては、『Oracle9i PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』の DBMS\_METADATA の章を参照してください。
- オブジェクトの完全な表現のみを提供します。

---

**注意：** XSLT 変換を介した場合以外のオブジェクト属性のサブセット化は、このリリースではサポートされていません。

---

- データベース・オブジェクトのメタデータを XML 形式で提供します。この形式は、ダウンストリーム・プロセスによって XSLT を介して簡単に変換できます。
- サポートされているすべてのオブジェクトに対して、完全な Oracle 固有の作成用 DDL を提供します。
- 柔軟性のあるオブジェクト選択を提供します。問合せごとに複数のオブジェクトを返すことができます。
- デイジー・チェーン変換をサポートします。この変換では、最初の出力が 2 番目の入力になり、2 番目の出力が 3 番目の入力になるように続きます。
- オブジェクト型固有の変換パラメータを使用した DDL 出力のカスタマイズをサポートします。

## インターネットによる計算

メタデータ API は、オブジェクト・メタデータのエンコーディングおよび変換に、XML および XSLT という 2 つのインターネット標準を使用します。メタデータのエンコーディングに（独自の形式ではなく）業界標準の形式を使用すると、標準ツールを使用して出力の解析および変換ができます。

現在、データベース・メタデータには業界標準の XML モデルがないため、メタデータ API では Oracle DDL の生成用に最適化されたモデルが使用されます。文書の要素名は、Oracle データベース・ディレクトリ・モデルの UDT の属性から直接導出されます。標準モデルが開発された場合、メタデータ API にはそれらをプラグインする機能がサポートされています。古い文書は、XSLT を使用して代替モデルに変換できます。

## メタデータ API の実装方法

メタデータ API は、PL/SQL の DBMS\_METADATA パッケージを使用して実装されます。DBMS\_METADATA パッケージを使用すると、データベース・ディクショナリからメタデータを取得できます。このパッケージによって、柔軟性および拡張性のある方法でオブジェクトを選択できます。DBMS\_METADATA を使用して、XML および DDL のデータベース・オブジェクト・メタデータを抽出できます。

DBMS\_METADATA パッケージには、次の 2 種類のインタフェースがあります。

- [DBMS\\_METADATA プログラム・インタフェース](#)
- [DBMS\\_METADATA ブラウザ・インタフェース](#)

---

---

**注意：** メタデータ API で定義された型およびパブリック・インタフェースの詳細は、次の場所を参照してください。

```
$ORACLE_HOME/rdbms/admin/dbmsmeta.sql
```

---

---

## DBMS\_METADATA とセキュリティ

Oracle メタデータ・モデルのオブジェクト・ビューでのセキュリティは、次のとおり実装されています。

- 権限を持たないユーザーが表示できるのは、各自のオブジェクトのメタデータのみです。
- SYS および SELECT\_CATALOG\_ROLE を持つユーザーは、すべてのオブジェクトを表示できます。
- 権限を持たないユーザーも、ユーザー自身に付与されたオブジェクトおよびシステム権限、またはユーザーが他のユーザーに付与したオブジェクトおよびシステム権限は取得できます。これには、PUBLIC に付与された権限も含まれます。
- コール元が取得権限のないオブジェクトを要求した場合、例外は発生しません。オブジェクトが取得されないのみです。
- 権限を持たないユーザーに他のユーザーのスキーマ内にあるオブジェクトに対するなんらかのアクセス権限が付与された場合、メタデータ API を介して付与指定は取得できませんが、オブジェクトの実際のメタデータは取得できません。

# DBMS\_METADATA プログラム・インタフェース

DBMS\_METADATA プログラム・インタフェースは、詳細なファイングレイイン制御に使用されます。

- OPEN、SET\_FILTER、SET\_PARSE\_ITEM、SET\_COUNT、ADD\_TRANSFORM、SET\_TRANSFORM\_PARAM、FETCH\_xxx、CLOSE のプロシージャが提供されています。
- メタデータは XML として表現されます。このため、XSLT を使用して業界標準のメタデータへの変換を実行できます。
- DBMS\_METADATA では、メタデータを DDL として返すように指定できます。API では内部で XSL スクリプトが使用され、変換が透過的に実行されます。
- Oracle XML Parser またはサード・パーティ製のツールのいずれかを使用して XSL スクリプトを起動すると、XML 表現をオフラインで変換できます。

表 15-1 では、DBMS\_METADATA プログラム・インタフェースで提供されるプロシージャを簡単に説明します。構文も含めた詳細は、『Oracle9i PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

表 15-1 DBMS\_METADATA プログラム・インタフェースのプロシージャ

PL/SQL プロシージャ	説明
DBMS_METADATA.OPEN()	取得するオブジェクトの型、メタデータのバージョンおよびオブジェクト・モデルを指定します。返される値は、後続のコールで使用されるオブジェクト・セット用の不透明なコンテキスト・ハンドルです。
BMS_METADATA.SET_FILTER()	取得するオブジェクトに、オブジェクト名、スキーマなどの制限を指定します。ベース・オブジェクトを、索引、トリガーなどの依存オブジェクトに対して指定できます。
DBMS_METADATA.SET_COUNT()	1 回の FETCH_xxx コールで取得するオブジェクトの数を指定します。デフォルトでは、FETCH_xxx へのコールごとに 1 つのオブジェクトが返されます。
DBMS_METADATA.GET_QUERY()	FETCH_xxx で使用される問合せ（複数の場合もあります）のテキストを返します。このテキストは、デバッグに有効です。
DBMS_METADATA.SET_PARSE_ITEM()	出力を解析可能にし、解析オブジェクト属性および返すオブジェクト属性を指定します。これによって、コール元は SQL の DDL のキー属性を解析する必要がなくなります。

表 15-1 DBMS\_METADATA プログラム・インタフェースのプロシージャ（続き）

PL/SQL プロシージャ	説明
DBMS_METADATA.ADD_TRANSFORM()	<p>取得したオブジェクトの XML 表現に、FETCH_xxx によって適用される変換を指定します。複数の変換を追加できます。デフォルトでは（変換を追加しない状態で）、オブジェクトが XML 文書として返されます。</p> <p>ADD_TRANSFORM プロシージャをコールして、返された文書を変換する XSLT スクリプトを指定します。DDL が指定されている場合、後続の FETCH_xxx コールからオブジェクトの作成用 DDL が返されます。ADD_TRANSFORM プロシージャでは、OPEN で返されるものとは異なる不透明な変換ハンドルが返されます。</p>
DBMS_METADATA.SET_TRANSFORM_PARAM()	<p>ADD_TRANSFORM プロシージャから返される <i>transform_handle</i> で識別される XSLT スタイルシートにパラメータを指定します。</p> <p>DDL 変換の場合、これらのパラメータによって DDL の形式が変更されます。たとえば、制約が列制約または ALTER TABLE 文として要求される場合があります。</p>
DBMS_METADATA.FETCH_xxx()	<p>FETCH_xxx ルーチンでは、OPEN、SET_FILTER、SET_COUNT および ADD_TRANSFORM プロシージャによって確立された条件を満たすオブジェクトのメタデータが返されます。</p> <p>FETCH_XML および FETCH_DDL では、メタデータがそれぞれ XML および SQL DDL として返されます。FETCH_CLOB ルーチンでは、指定された変換で示されたとおり、XML または DDL のいずれかが返されます。</p> <p>これらのルーチンで使用される型の詳細は、『Oracle9i PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。</p>
DBMS_METADATA.CLOSE()	<p>OPEN プロシージャによって返されたハンドルを無効にし、関連付けられた状態をクリーンアップします。</p>

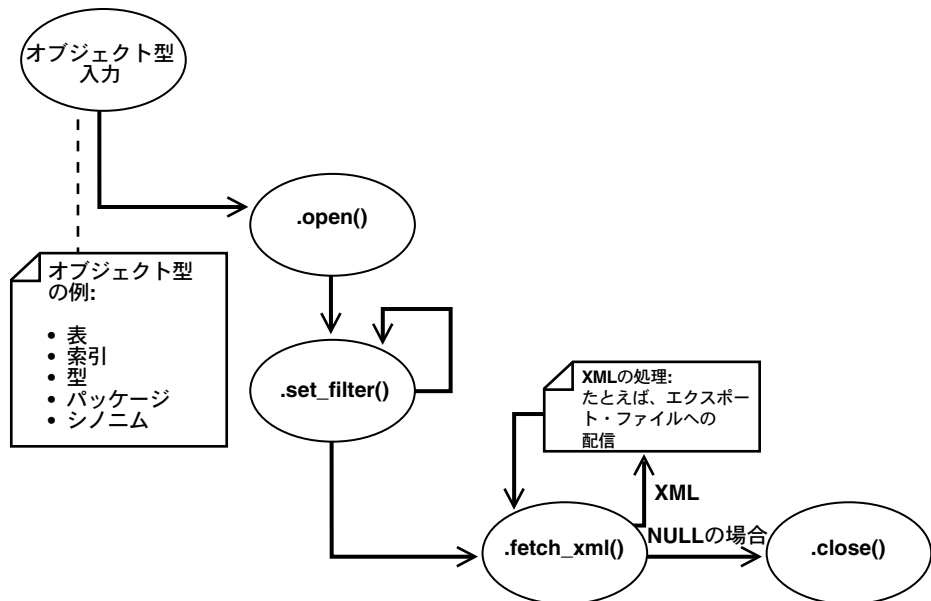
## DBMS\_METADATA.FETCH\_XML プロシージャの使用

図 15-1 に、DBMS\_METADATA.FETCH\_XML() を使用する手順を示します。

1. DBMS\_METADATA.OPEN() プロシージャを使用して、オブジェクト型をオープンします。オープン可能なオブジェクト型には、表、索引、型、パッケージ、シノニムなどがあります。
2. DBMS\_METADATA.SET\_FILTER() プロシージャを使用して、取得するオブジェクトを指定します。
3. DBMS\_METADATA.FETCH\_XML() プロシージャを使用して、各修飾オブジェクトのメタデータを XML 文書としてフェッチします。たとえば、XML は処理されてエクスポート・ファイルに配信されます。
4. この操作の結果が NULL の場合は、DBMS\_METADATA.CLOSE() プロシージャをコールします。

図 15-1 DBMS\_METADATA.FETCH\_XML() の使用

### DBMS\_METADATA: fetch\_xml()



## DBMS\_METADATA.FETCH\_DDL プロシージャの使用

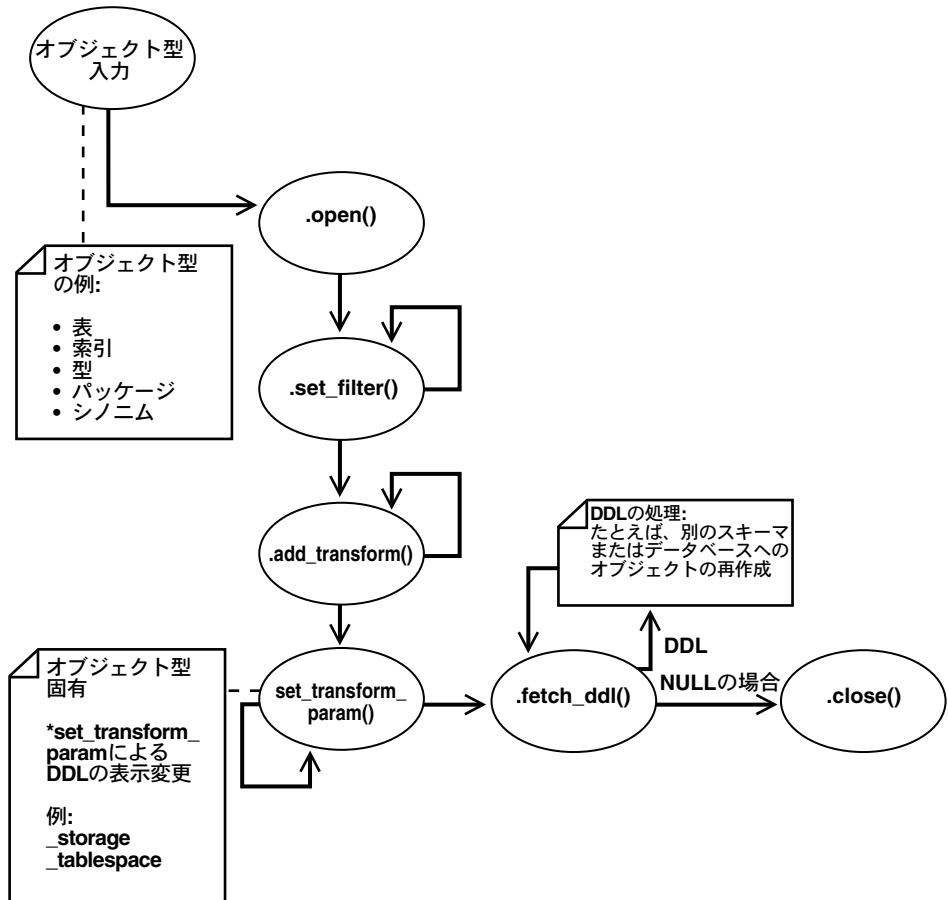
図 15-2 に、DBMS\_METADATA.FETCH\_DDL() を使用する手順を示します。

1. DBMS\_METADATA.OPEN() プロシージャを使用して、オブジェクト型をオープンします。オープン可能なオブジェクト型には、表、索引、型、パッケージ、シノニムなどがあります。
2. DBMS\_METADATA.SET\_FILTER() プロシージャを使用して、取得するオブジェクトを指定します。
3. 出力に対して起動する変換の種類を指定します。  
DBMS\_METADATA.ADD\_TRANSFORM() プロシージャを使用して、変換を追加します。  
最後に追加する変換は、「DDL」変換にする必要があります。
4. DBMS\_METADATA.SET\_TRANSFORM\_PARAM() プロシージャを使用して、DDL をカスタマイズします。たとえば、このプロシージャを使用して、表定義の STORAGE 句を排除できます。変換パラメータは、選択したオブジェクト型に固有です。
5. DBMS\_METADATA.FETCH\_DDL() プロシージャを使用して、DDL をフェッチします。  
DDL 処理の例には、他のスキーマまたはデータベースでのオブジェクトの再作成があります。
6. この操作の結果が NULL の場合は、DBMS\_METADATA.CLOSE() プロシージャをコールします。



図 15-2 DBMS\_METADATA.FETCH\_DDL() の使用

## DBMS\_METADATA: fetch\_ddl()



## メタデータ API のプログラム・インタフェースに関するパフォーマンスのヒント

この項では、メタデータ API のプログラム・インタフェース使用時にパフォーマンスを向上する方法について説明します。

1. ある型のすべてのオブジェクトを、次の型のオブジェクトをフェッチする前にフェッチします。たとえば、スキーマ内に含まれるすべてのオブジェクトの定義を取得する場合は、まずすべての表をフェッチし、次にすべての索引をフェッチし、その次にすべてのトリガーをフェッチします。この方法は、OPEN コンテキストをネストする方法（表をフェッチした後でそのすべての索引、権限およびトリガーをフェッチし、その後、次の表をフェッチした後でそのすべての索引、権限およびトリガーをフェッチするという方法）より非常に高速です。15-11 ページの「[メタデータ API の例](#)」では、この非効率な後者の方法が示されていますが、その例の目的は多くのプログラム・コールを示すことであり、その目的には、この方法が最適であるためです。
2. SET\_COUNT プロシージャを使用して、複数のオブジェクトを一度に取得します。これによってサーバーのラウンドトリップが最小化され、多くの冗長なファンクション・コールが削減されます。
3. FETCH\_CLOB のファンクション形式ではなく、プロシージャを使用します。プロシージャ形式では、IN OUT NOCOPY 指定子を介した参照によって出力 CLOB が返されます。ファンクション形式では、追加の LOB コピーを必要とする値によって出力 CLOB が返されます。
4. メタデータ API をコールする PL/SQL パッケージを記述する場合、LOB 変数および LOB を含むオブジェクト（SYS.KU\$\_DDL など）は、個々のファンクション内ではなく、パッケージ・スコープで宣言します。これによって、ファンクションの開始および終了時に、負荷の高い操作である、LOB の存続時間構造の作成および削除が実行されなくなります。

**参照：**『Oracle9i アプリケーション開発者ガイド - ラージ・オブジェクト』を参照してください。

## DBMS\_METADATA ブラウザ・インタフェース

DBMS\_METADATA ブラウザ・インタフェースは、SQL\*Plus などの SQL クライアント内部で簡単に使用するためのインタフェースです。このインタフェースは、GET\_XXX、GET\_DEPENDENT\_XXX および GET\_GRANTED\_XXX ファンクションで提供されます。

- GET\_DDL ファンクションおよび GET\_XML ファンクションでは、1 つの名前付きオブジェクトに対してメタデータが返されます。たとえば、次の問合せでは、現在のユーザーのスキーマに含まれるすべての表に対する DDL が表示されます。

```
SQL> SELECT dbms_metadata.get_ddl('TABLE', table_name) FROM user_tables;
```

- GET\_DEPENDENT\_XML、GET\_DEPENDENT\_DDL、GET\_GRANTED\_XML および GET\_GRANTED\_DDL ファンクションでは、1 つ以上の依存オブジェクトまたは権限が付与されたオブジェクトに対してメタデータが返されます。

表 15-2 では、DBMS\_METADATA ブラウザ・インタフェースで提供されるプロシージャを簡単に説明します。構文も含めた詳細は、『Oracle9i PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

表 15-2 DBMS\_METADATA ブラウザ・インタフェースのプロシージャ

PL/SQL プロシージャ名	説明
DBMS_METADATA.GET_XXX()	シングル・オブジェクトに対してメタデータを返す方法を提供します。各 GET_XXX コールは、OPEN プロシージャ、1 つか 2 つの SET_FILTER コール、ADD_TRANSFORM プロシージャ（オプション）、FETCH_XXX コールおよび CLOSE プロシージャで構成されます。  object_type パラメータには、OPEN プロシージャと同じセマンティクスが含まれます。schema および name はフィルタ処理に使用されます。  変換を指定した場合、セッション・レベルの変換フラグが継承されます。
DBMS_METADATA.GET_DEPENDENT_XXX()	XML または DDL で指定したとおり、1 つ以上の依存オブジェクトに対してメタデータが返されます。
DBMS_METADATA.GET_GRANTED_XXX()	XML または DDL で指定したとおり、権限が付与された 1 つ以上のオブジェクトに対してメタデータが返されます。

例 : DBMS\_METADATA ブラウザ・インタフェースの使用

次の SQL\*Plus の問合せでは、現在のユーザーのスキーマに含まれるすべての表に対する作成用 DDL が表示されます。完全で中断のない出力を生成するには、問合せの実行前に、次のとおり、PAGESIZE を 0（ゼロ）にし、また LONG を大きい数に設定します。

```
SQL> SET PAGESIZE 0
SQL> SET LONG 90000
SQL> SELECT dbms_metadata.get_ddl('TABLE', table_name) FROM user_tables;
```

メタデータ API の例

この項では、メタデータ API の詳細なプログラム例 PAYROLL DEMO を示します。このプログラムでは、「PAYROLL」から始まる MDDMO スキーマに含まれるすべての表に対して、DDL が取得されます。その後、それらの表で定義された権限、索引およびトリガーに対して DDL がフェッチされます。このスクリプトは、Oracle ホーム・ディレクトリの rdbms/demo/mddemo.sql ファイルに含まれています。

## mddemo.sql

```
-- This script demonstrates how to use the Metadata API. It first
-- establishes a schema (MDDemo) and some payroll users, then creates three
-- payroll-like tables within it along with associated indexes, triggers
-- and grants.

-- It then creates a package PAYROLL_DEMO that shows common usage of the
-- Metadata API. The procedure GET_PAYROLL_TABLES retrieves the DDL for the
-- two tables in this schema that start with 'PAYROLL' then for each one,
-- retrieves the DDL for its associate dependent objects; indexes, grants
-- and triggers. All the DDL is written to a table named "MDDemo"."DDL".

-- First, Install the demo. cd to rdbms/demo:
-- > sqlplus system/manager
-- SQL> @mddemo

-- Then, run it.
-- > sqlplus mddemo/mddemo
-- SQL> set long 40000
-- SQL> set pages 0
-- SQL> call payroll_demo.get_payroll_tables();
-- SQL> select ddl from DDL order by seqno;

Rem Set up schema for demo pkg. PAYROLL_DEMO.

connect system/manager
drop user mddemo cascade;
drop user mddemo_clerk cascade;
drop user mddemo_mgr cascade;

create user mddemo identified by mddemo;
GRANT resource, connect, create session
    , create table
    , create procedure
    , create sequence
    , create trigger
    , create view
    , create synonym
    , alter session
TO mddemo;

create user mddemo_clerk identified by clerk;
create user mddemo_mgr identified by mgr;

connect mddemo/mddemo
```

```
Rem Create some payroll-like tables...

create table payroll_emps
( lastname varchar2(60) not null,
  firstname varchar2(20) not null,
  mi varchar2(2),
  suffix varchar2(10),
  DOB date not null,
  badge_no number(6) primary key,
  exempt varchar(1) not null,
  salary number (9,2),
  hourly_rate number (7,2)
)
/
create table payroll_timecards
  badge_no number(6) references payroll_emps (badge_no),
  week number(2),
  job_id number(5),
  hours_worked number(4,2)
)
/
-- This is a dummy table used only to show that tables NOT starting with
-- 'PAYROLL' are NOT retrieved by payroll_demo.get_payroll_tables

create table audit_trail
(action_time DATE,
lastname VARCHAR2(60),
action LONG
)
/

Rem Then, create some grants...

grant update (salary, hourly_rate) on payroll_emps to mddemo_clerk;
grant ALL on payroll_emps to mddemo_mgr with grant option;

grant insert, update on payroll_timecards to mddemo_clerk;
grant ALL on payroll_timecards to mddemo_mgr with grant option;

Rem Then, create some indexes...

create index i_payroll_emps_name on payroll_emps(lastname);
create index i_payroll_emps_dob on payroll_emps(DOB);

create index i_payroll_timecards_badge on payroll_timecards(badge_no);

Rem Then, create some triggers (and required procedure)...
```

```
create or replace procedure check_sal( salary in number) as
begin
    return; -- Fairly loose security here...
end;
/

create or replace trigger salary_trigger before insert or update of salary on
payroll_emps
for each row when (new.salary > 150000)
call check_sal(:new.salary)
/

create or replace trigger hourly_trigger before update of hourly_rate on payroll_
emps
for each row
begin :new.hourly_rate:=:old.hourly_rate;end;
/

--
-- Set up a table to hold the generated DDL
--
CREATE TABLE ddl (ddl CLOB, seqno NUMBER);

Rem Finally, create the PAYROLL_DEMO package itself.

CREATE OR REPLACE PACKAGE payroll_demo AS

    PROCEDURE get_payroll_tables;
END;
/
CREATE OR REPLACE PACKAGE BODY payroll_demo AS

-- GET_PAYROLL_TABLES: Fetch DDL for payroll tables and their dependent objects

PROCEDURE get_payroll_tables IS

tableOpenHandle NUMBER;
depObjOpenHandle NUMBER;
tableTransHandle NUMBER;
indexTransHandle NUMBER;
schemaName VARCHAR2(30);
tableName VARCHAR2(30);
tableDDLs sys.ku$_ddl;
tableDDL sys.ku$_ddl;
parsedItems sys.ku$_parsed_items;
depObjDDL CLOB;
```

```

seqNo NUMBER := 1;

TYPE obj_array_t IS VARRAY(3) OF VARCHAR2(30);

-- Load this array with the dependent object classes to be retrieved...
obj_array obj_array_t := obj_array_t('OBJECT_GRANT', 'INDEX', 'TRIGGER');

BEGIN

-- Open a handle for tables in the current schema.
tableOpenHandle := dbms_metadata.open('TABLE');

-- Tell mdAPI to retrieve one table at a time. This call is not actually
-- necessary since 1 is the default... just showing the call.
dbms_metadata.set_count(tableOpenHandle, 1);

-- Retrieve tables whose name starts with 'PAYROLL'. When the filter is
-- 'NAME_EXPR', the filter value string must include the SQL operator. This
-- gives the caller flexibility to use LIKE, IN, NOT IN, subqueries, etc.
dbms_metadata.set_filter(tableOpenHandle, 'NAME_EXPR', 'LIKE 'PAYROLL%');

-- Tell the mdAPI to parse out each table's schema and name separately so we
-- can use them to set up the calls to retrieve its dependent objects.
dbms_metadata.set_parse_item(tableOpenHandle, 'SCHEMA');
dbms_metadata.set_parse_item(tableOpenHandle, 'NAME');

-- Add the DDL transform so we get SQL creation DDL
tableTransHandle := dbms_metadata.add_transform(tableOpenHandle, 'DDL');

-- Tell the XSL stylesheet we don't want physical storage information (storage,
-- tablespace, etc), and that we want a SQL terminator on each DDL. Notice that
-- these calls use the transform handle, not the open handle.
dbms_metadata.set_transform_param(tableTransHandle,
    'SEGMENT_ATTRIBUTES', FALSE);
dbms_metadata.set_transform_param(tableTransHandle,
    'SQLTERMINATOR', TRUE);

-- Ready to start fetching tables. We use the FETCH_DDL interface (rather than
-- FETCH_XML or FETCH_CLOB). This interface returns a SYS.KU$_DDL; a table of
-- SYS.KU$_DDL objects. This is a table because some object types return
-- multiple DDL statements (like types / pkgs which have create header and
-- body statements). Each KU$_DDL has a CLOB containing the 'CREATE foo'
-- statement plus a nested table of the parse items specified. In our case,
-- we asked for two parse items; Schema and Name. (NOTE: See admin/dbmsmeta.sql
-- for a more detailed description of these types)

LOOP

```

```
tableDDLs := dbms_metadata.fetch_ddl(tableOpenHandle);
EXIT WHEN tableDDLs IS NULL; -- Get out when no more payroll tables

-- In our case, we know there is only one row in tableDDLs (a KU$_DDLs tbl obj)
-- for the current table. Sometimes tables have multiple DDL statements;
-- eg, if constraints are applied as ALTER TABLE statements, but we didn't ask
-- for that option. So, rather than writing code to loop through tableDDLs,
-- we'll just work with the 1st row.
--
-- First, write the CREATE TABLE text to our output table then retrieve the
-- parsed schema and table names.
tableDDL := tableDDLs(1);
INSERT INTO ddl VALUES(tableDDL.ddltext, seqNo);
seqNo := seqNo + 1;
parsedItems := tableDDL.parsedItems;

-- Must check the name of the returned parse items as ordering isn't guaranteed
FOR i IN 1..2 LOOP
    IF parsedItems(i).item = 'SCHEMA'
    THEN
        schemaName := parsedItems(i).value;
    ELSE
        tableName := parsedItems(i).value;
    END IF;
END LOOP;

-- Now, we want to retrieve all the dependent objects defined on the current
-- table: indexes, triggers and grants. Since all 'dependent' object types
-- have BASE_OBJECT_NAME and BASE_OBJECT_SCHEMA in common as filter criteria,
-- we'll set up a loop to get all objects of the 3 types, just changing the
-- OPEN context in each pass through the loop. Transform parameters are
-- different for each object type, so we'll only use one that's common to all;
-- SQLTERMINATOR.

FOR i IN 1..3 LOOP
    depObjOpenHandle := dbms_metadata.open(obj_array(i));
    dbms_metadata.set_filter(depObjOpenHandle, 'BASE_OBJECT_SCHEMA',
        schemaName);
    dbms_metadata.set_filter(depObjOpenHandle, 'BASE_OBJECT_NAME', tableName);

-- Add the DDL transform and say we want a SQL terminator
    indexTransHandle := dbms_metadata.add_transform(depObjOpenHandle, 'DDL');
    dbms_metadata.set_transform_param(indexTransHandle,
        'SQLTERMINATOR', TRUE);

-- Retrieve dependent object DDLs as CLOBs and write them to table DDL.
    LOOP
```



```

        depObjDDL := dbms_metadata.fetch_clob(depObjOpenHandle);
        EXIT WHEN depObjDDL IS NULL;
        INSERT INTO ddl VALUES(depObjDDL, seqNo);
        seqNo := seqNo + 1;
    END LOOP;

    -- Free resources allocated for current dependent object stream.
    dbms_metadata.close(depObjOpenHandle);

    END LOOP; -- End of fetch dependent objects loop

    END LOOP; -- End of fetch table loop

    -- Free resources allocated for table stream and close output file.
    dbms_metadata.close(tableOpenHandle);
    RETURN;

END; -- of procedure get_payroll_tables

END payroll_demo;
/

```

## PAYROLL\_DEMO の出力

次に、プロシージャ mddemo.payroll\_demo.get\_payroll\_tables の実行によって得られる出力を示します。この出力は、次の問合せをユーザー mddemo として実行することによって得られます。

```
SQL> SELECT ddl FROM ddl ORDER BY seqno;
```

```

CREATE TABLE "MDDemo"."PAYROLL_EMPS"
(
    "LASTNAME" VARCHAR2(60) NOT NULL ENABLE,
    "FIRSTNAME" VARCHAR2(20) NOT NULL ENABLE,
    "MI" VARCHAR2(2),
    "SUFFIX" VARCHAR2(10),
    "DOB" DATE NOT NULL ENABLE,
    "BADGE_NO" NUMBER(6,0),
    "EXEMPT" VARCHAR2(1) NOT NULL ENABLE,
    "SALARY" NUMBER(9,2),
    "HOURLY_RATE" NUMBER(7,2),
    PRIMARY KEY ("BADGE_NO") ENABLE
) ;

GRANT UPDATE ("SALARY") ON "MDDemo"."PAYROLL_EMPS" TO "MDDemo_CLERK";
GRANT UPDATE ("HOURLY_RATE") ON "MDDemo"."PAYROLL_EMPS" TO "MDDemo_CLERK";
GRANT ALTER ON "MDDemo"."PAYROLL_EMPS" TO "MDDemo_MGR" WITH GRANT OPTION;
GRANT DELETE ON "MDDemo"."PAYROLL_EMPS" TO "MDDemo_MGR" WITH GRANT OPTION;

```

```
GRANT INDEX ON "MDDemo"."PAYROLL_EMPS" TO "MDDemo_MGR" WITH GRANT OPTION;
GRANT INSERT ON "MDDemo"."PAYROLL_EMPS" TO "MDDemo_MGR" WITH GRANT OPTION;
GRANT SELECT ON "MDDemo"."PAYROLL_EMPS" TO "MDDemo_MGR" WITH GRANT OPTION;
GRANT UPDATE ON "MDDemo"."PAYROLL_EMPS" TO "MDDemo_MGR" WITH GRANT OPTION;
GRANT REFERENCES ON "MDDemo"."PAYROLL_EMPS" TO "MDDemo_MGR" WITH GRANT OPTION;
GRANT ON COMMIT REFRESH ON "MDDemo"."PAYROLL_EMPS" TO "MDDemo_MGR" WITH GRANT OPTION;
GRANT QUERY REWRITE ON "MDDemo"."PAYROLL_EMPS" TO "MDDemo_MGR" WITH GRANT OPTION;

CREATE INDEX "MDDemo"."I_PAYROLL_EMPS_DOB" ON "MDDemo"."PAYROLL_EMPS" ("DOB")
PCTFREE 10 INITRANS 2 MAXTRANS 255
STORAGE (INITIAL 10240 NEXT 10240 MINEXTENTS 1 MAXEXTENTS 121 PCTINCREASE 50
FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT) TABLESPACE "SYSTEM" ;

CREATE INDEX "MDDemo"."I_PAYROLL_EMPS_NAME" ON "MDDemo"."PAYROLL_EMPS" ("LASTNAME")
PCTFREE 10 INITRANS 2 MAXTRANS 255
STORAGE (INITIAL 10240 NEXT 10240 MINEXTENTS 1 MAXEXTENTS 121 PCTINCREASE 50
FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT) TABLESPACE "SYSTEM" ;

CREATE OR REPLACE TRIGGER hourly_trigger before update of hourly_rate on payroll_emps
for each row
begin :new.hourly_rate:=:old.hourly_rate;end;
/
ALTER TRIGGER "MDDemo"."HOURLY_TRIGGER" ENABLE;

CREATE OR REPLACE TRIGGER salary_trigger before insert or update of salary on payroll_emps
for each row
WHEN (new.salary > 150000) CALL check_sal(:new.salary)
/
ALTER TRIGGER "MDDemo"."SALARY_TRIGGER" ENABLE;

CREATE TABLE "MDDemo"."PAYROLL_TIMECARDS"
(
    "BADGE_NO" NUMBER(6,0),
    "WEEK" NUMBER(2,0),
    "JOB_ID" NUMBER(5,0),
    "HOURS_WORKED" NUMBER(4,2),
    FOREIGN KEY ("BADGE_NO")
    REFERENCES "MDDemo"."PAYROLL_EMPS" ("BADGE_NO") ENABLE
) ;

GRANT INSERT ON "MDDemo"."PAYROLL_TIMECARDS" TO "MDDemo_CLERK";
GRANT UPDATE ON "MDDemo"."PAYROLL_TIMECARDS" TO "MDDemo_CLERK";
GRANT ALTER ON "MDDemo"."PAYROLL_TIMECARDS" TO "MDDemo_MGR" WITH GRANT OPTION;
GRANT DELETE ON "MDDemo"."PAYROLL_TIMECARDS" TO "MDDemo_MGR" WITH GRANT OPTION;
GRANT INDEX ON "MDDemo"."PAYROLL_TIMECARDS" TO "MDDemo_MGR" WITH GRANT OPTION;
GRANT INSERT ON "MDDemo"."PAYROLL_TIMECARDS" TO "MDDemo_MGR" WITH GRANT OPTION;
```

```
GRANT SELECT ON "MDDMO"."PAYROLL_TIMECARDS" TO "MDDMO_MGR" WITH GRANT OPTION;  
GRANT UPDATE ON "MDDMO"."PAYROLL_TIMECARDS" TO "MDDMO_MGR" WITH GRANT OPTION;  
GRANT REFERENCES ON "MDDMO"."PAYROLL_TIMECARDS" TO "MDDMO_MGR" WITH GRANT OPTION;  
GRANT ON COMMIT REFRESH ON "MDDMO"."PAYROLL_TIMECARDS" TO "MDDMO_MGR" WITH GRANT OPTION;  
GRANT QUERY REWRITE ON "MDDMO"."PAYROLL_TIMECARDS" TO "MDDMO_MGR" WITH GRANT OPTION;  
  
CREATE INDEX "MDDMO"."I_PAYROLL_TIMECARDS_BADGE" ON "MDDMO"."PAYROLL_TIMECARDS" ("BADGE_NO")  
PCTFREE 10 INITRANS 2 MAXTRANS 255  
STORAGE(INITIAL 10240 NEXT 10240 MINEXTENTS 1 MAXEXTENTS 121 PCTINCREASE 50  
FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT) TABLESPACE "SYSTEM" ;
```



# 第 V 部

---

## 付録

第 V 部には、次の付録が含まれます。

### [付録 A 「SQL\\*Loader の構文図」](#)

この付録では、SQL\*Loader の構文図について説明します。

### [付録 B 「DB2/DXT ユーザーに対する注意事項」](#)

この付録では、SQL\*Loader で使用する DDL の構文と、DB2 ロード・ユーティリティの制御ファイルで使用する DDL の構文の違いについて説明します。

### [付録 C 「バックス正規形構文」](#)

この付録では、テキストによる構文図の説明に使用される、バックス正規形の変形に関する記号および表記規則について説明します。

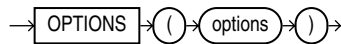


## SQL\*Loader の構文図

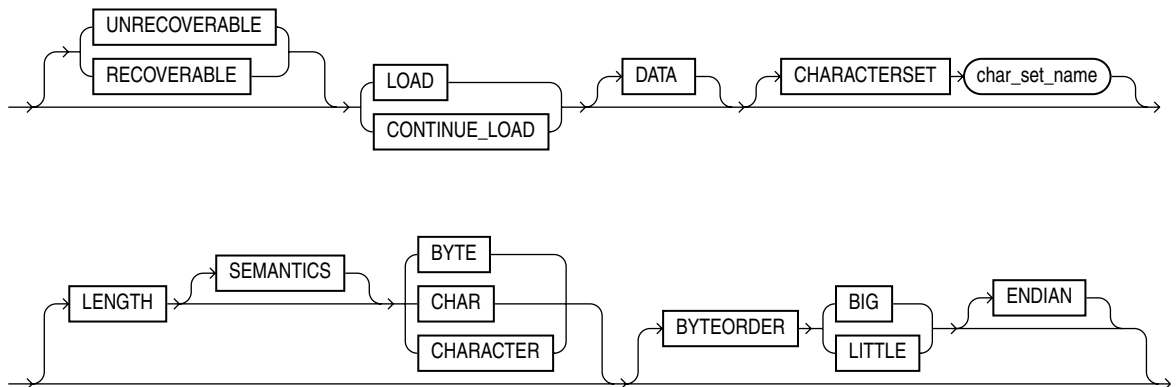
SQL\*Loader の DDL 構文図（線路図ともいいます）は、標準 SQL 構文の表記法を使用しています。この付録で使用している構文の表記法の詳細は、『PL/SQL ユーザーズ・ガイドおよびリファレンス』および『Oracle9i SQL リファレンス』を参照してください。

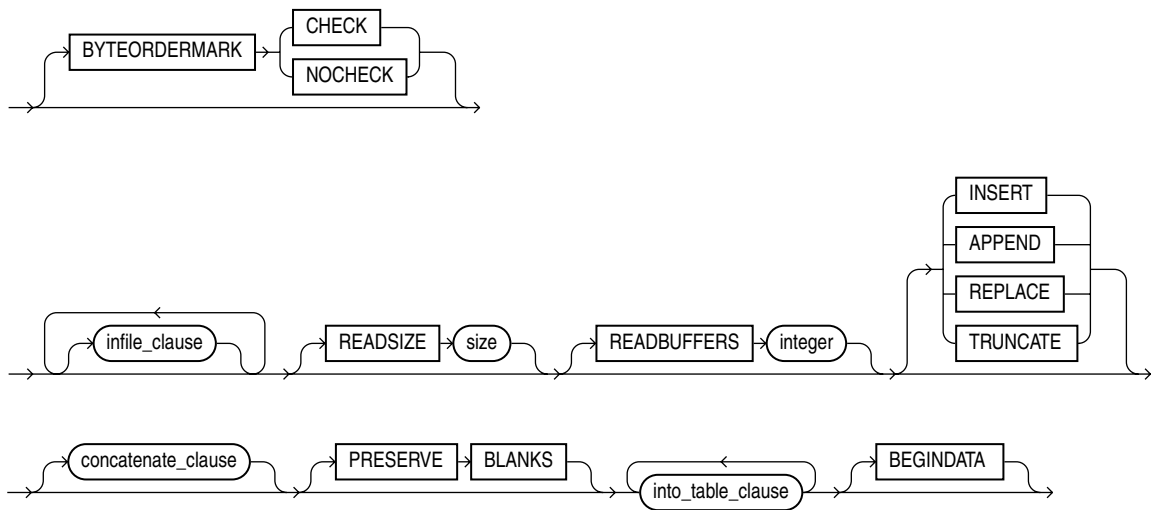
次に示す DDL 構文図は、特定の句（pos\_spec 句など）が省略された形で示されています。この付録では、これらの構文図を拡張して詳しく説明します。

### OPTIONS 句

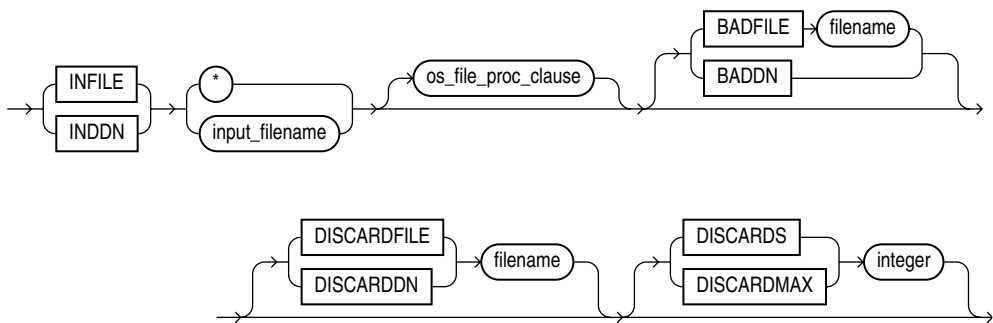


### Load 文



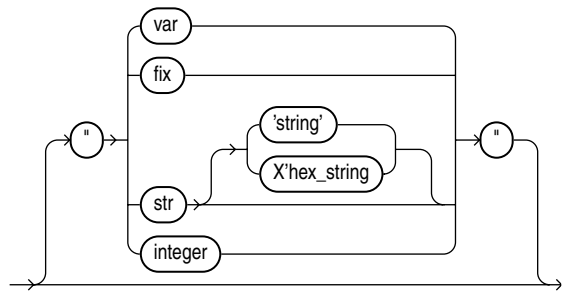


### infile\_clause

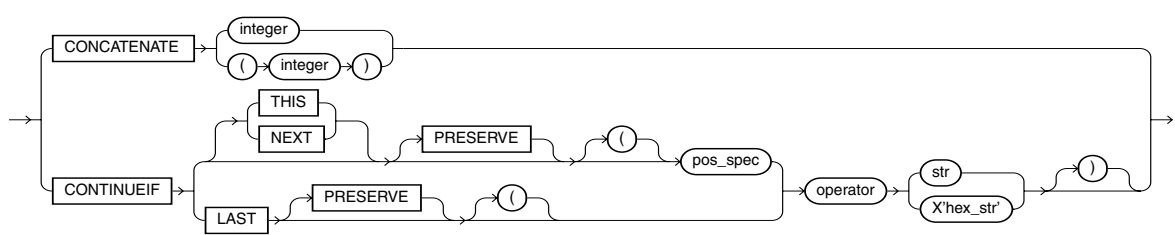




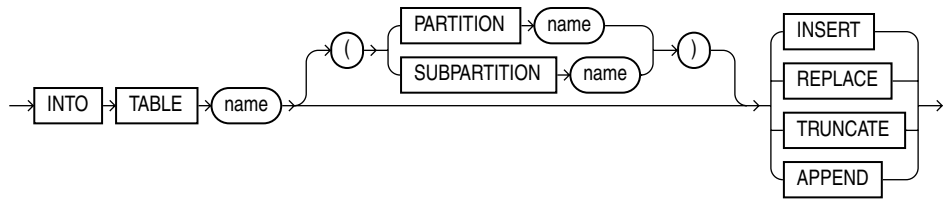
**os\_file\_proc\_clause**

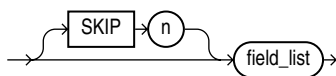
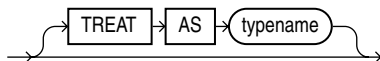
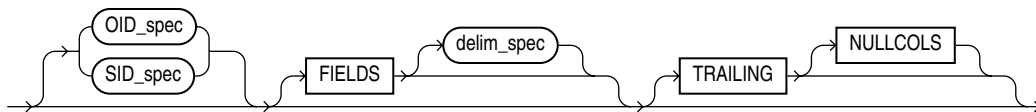
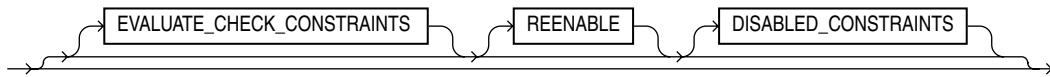
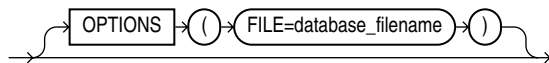
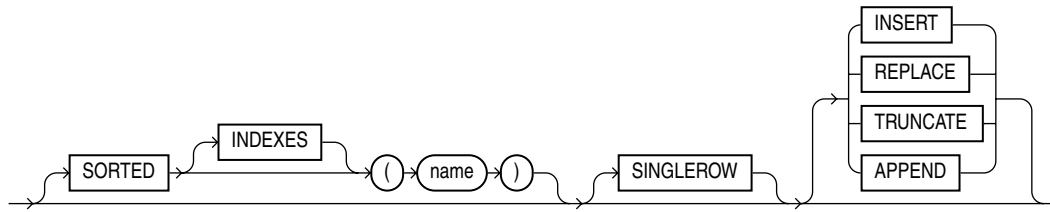


**concatenate\_clause**

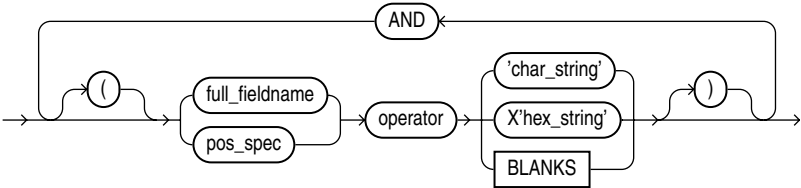


**into\_table\_clause**

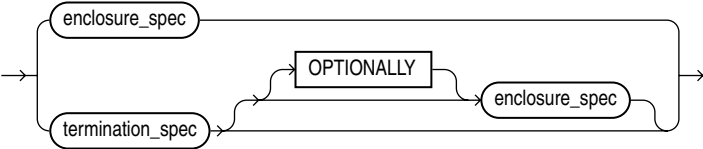




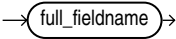
**field\_condition**



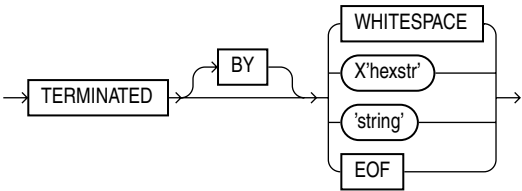
**delim\_spec**



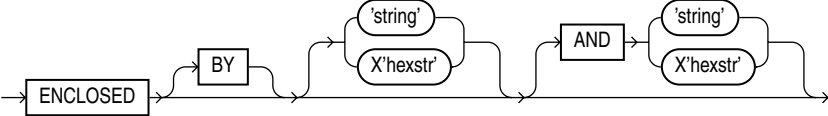
**full\_fieldname**



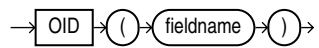
**termination\_spec**



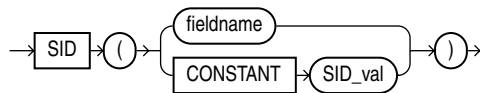
**enclosure\_spec**



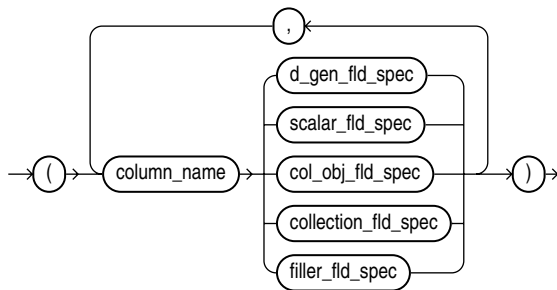
### oid\_spec



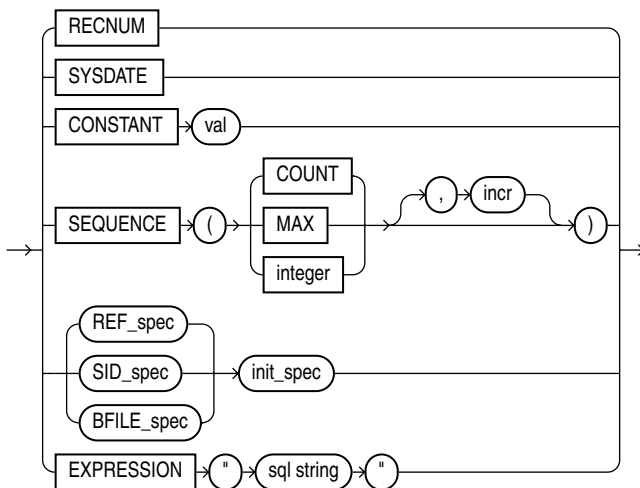
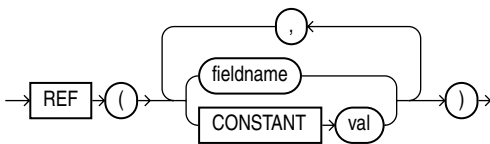
### sid\_spec



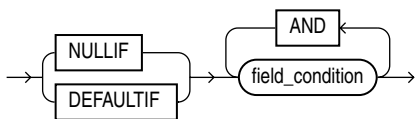
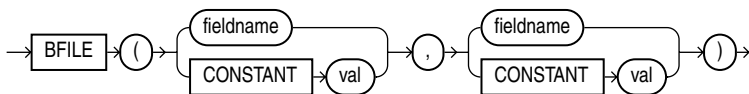
### field\_list



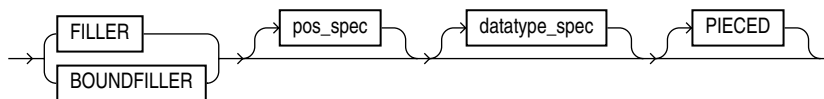
## d\_gen\_fld\_spec

**ref\_spec**

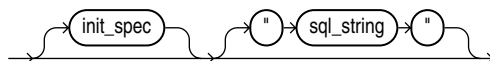
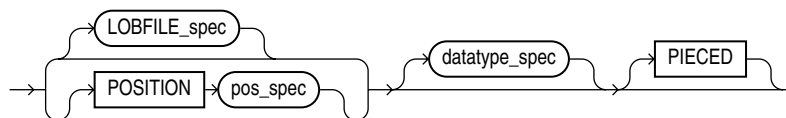
## init\_spec

**bfile\_spec**

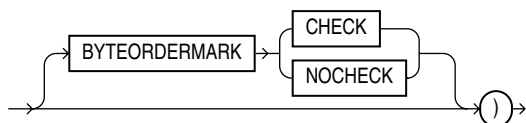
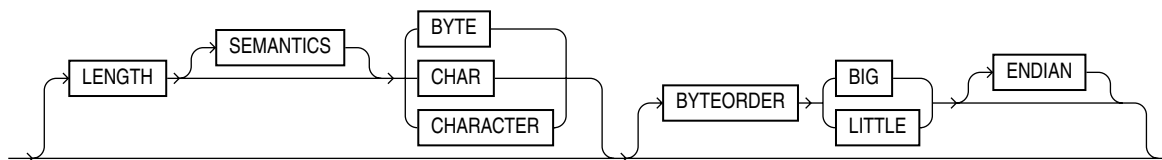
### filler\_fld\_spec



### scalar\_fld\_spec

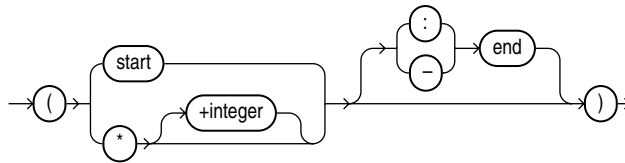


### lobfile\_spec



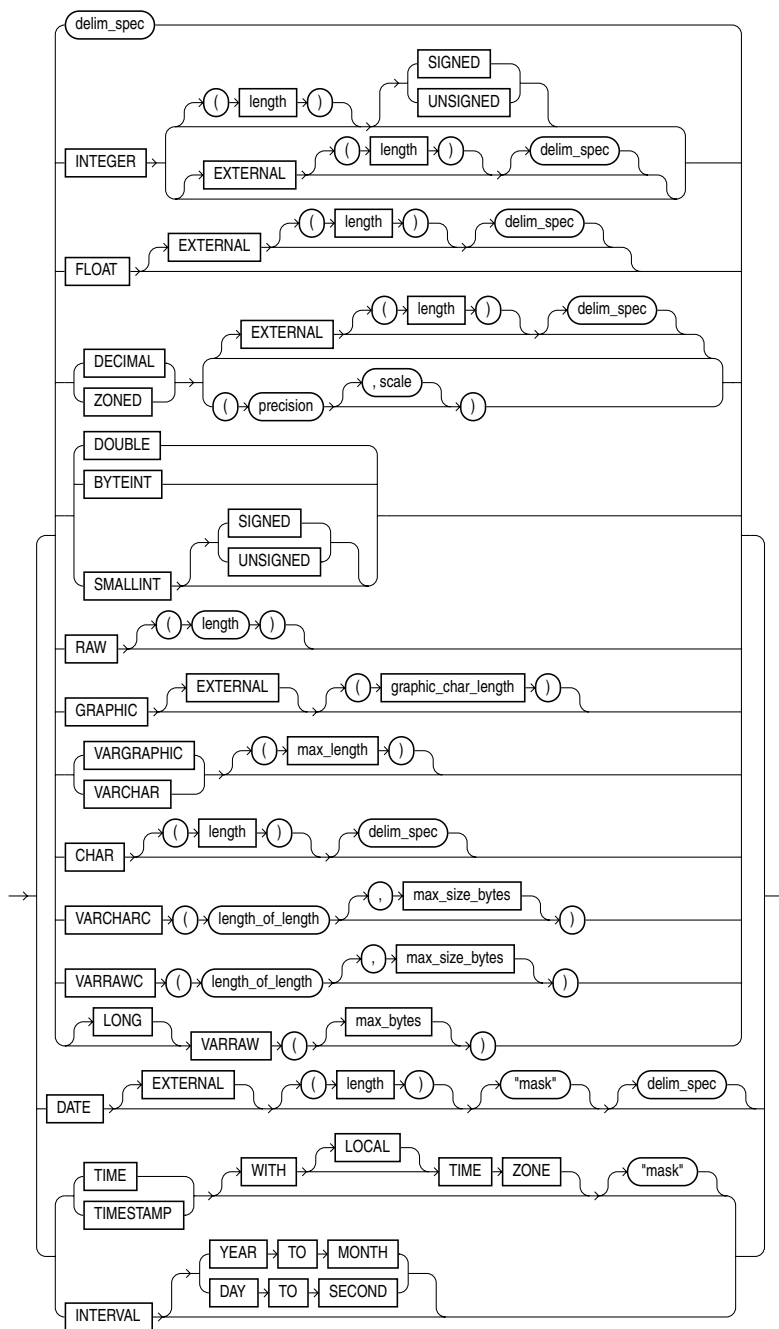
---

## pos\_spec



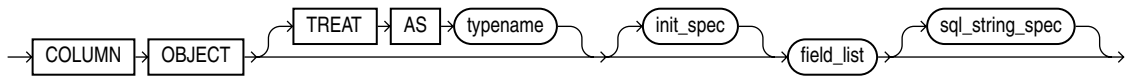
## datatype\_spec

datatype\_spec の構文は次のとおりです。

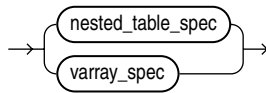




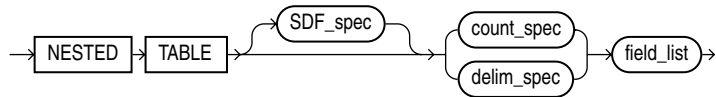
## col\_obj\_fld\_spec



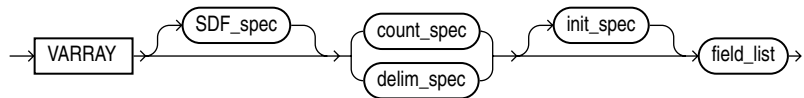
## collection\_fld\_spec



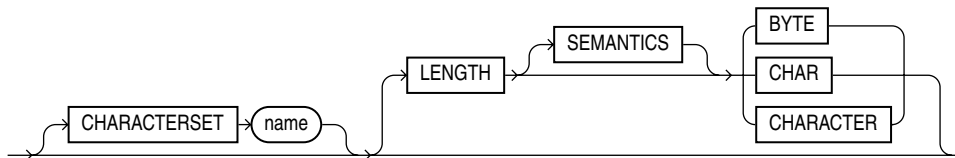
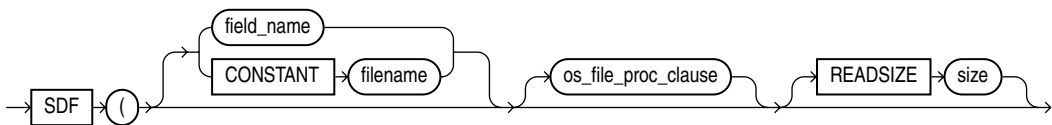
## nested\_table\_spec

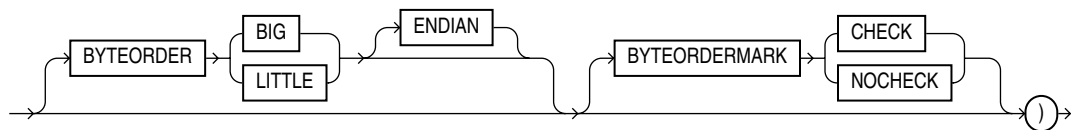


## varray\_spec

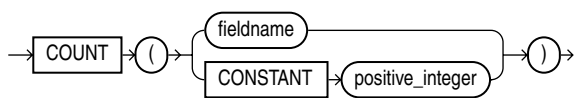


## sdf\_spec





### **count\_spec**



---

## DB2/DXT ユーザーに対する注意事項

この付録では、SQL\*Loader の DDL 構文と DB2 ロード・ユーティリティ /DXT 制御ファイルの構文の違いについて説明します。この付録の内容は、次のとおりです。

- [DB2 RESUME オプションの使用方法](#)
- [互換性維持のための機能](#)
- [制限事項](#)
- [DB2 と互換性のある文を含む SQL\\*Loader の構文](#)

# DB2 RESUME オプションの使用方法

ロード中の表にデータがすでに含まれている場合、ユーザーはそのデータの処理方法を 3 つのオプション（表 B-1 を参照）の中から選択します。

表 B-1 DB2 の関数とそれに相当する SQL\*Loader のオプション

DB2	SQL*Loader オプション	結果
RESUME NO または RESUME 句なし	INSERT	表が空のときのみデータがロードされます。その他の場合には、エラーが通知されます。
RESUME YES	APPEND	表内に既存のデータがある場合、これに新しいデータが付加されます。
RESUME NO REPLACE	REPLACE	表内に既存のデータがある場合、これと新しいデータが置き換えられます。

RESUME 句の DB2 構文は次のとおりです。

```
RESUME { YES | NO [ REPLACE ] }
```

DB2 の RESUME 構文のかわりに、相当する SQL\*Loader オプションを指定することもできます。

SQL\*Loader では、すべての INTO TABLE 句の指定の前に RESUME 句を 1 回指定すると、ロードする表すべてにその RESUME 句が適用されます。また、INTO TABLE 指定の後に RESUME 句を指定すると、RESUME オプションを表単位で指定できます。表名の後に RESUME オプションを指定した場合は、同一ファイル中でそれより前に指定された RESUME オプションより優先されます。INTO TABLE 句の前に指定した RESUME は、個別に RESUME 句が指定されていない表すべてに適用されます。

## 互換性維持のための機能

IBM 社の DB2 ロード・ユーティリティには、SQL\*Loader では使用しない要素も含まれています。たとえば、DB2 では、外部ファイルを使用してソート済索引を作成しますが、この外部ファイルの指定はロード文の中で行うことができます。DB2 のローダーとの互換性を保つため、SQL\* Loader ではこれらのオプションが解析されますが、Oracle データベース・サーバーにとって意味がない場合、そのオプションは無視されます。次の項に示す構文要素は使用可能ですが、SQL\*Loader では無視されます。

## LOG 文

DB2 との互換性を保つために用意されています。SQL\*Loader ではこの文の解析は行われますが、無視されます（この LOG オプションは、SQL\*Loader が書き込むログ・ファイルとは関係がありません）。DB2 ではエラー・リカバリのためにログ・ファイルが使用されますが、必ずログが記録されるわけではありません。

SQL\*Loader では、Oracle の自動ログ機能が利用されるため、ウォーム・スタート・オプションの設定によっては使用できない場合もあります。

```
[ LOG { YES | NO } ]
```

## WORKDDN 文

DB2 との互換性を保つために用意されています。SQL\*Loader ではこの文の解析は行われますが、無視されます。DB2 では、この文は、ソート用の一時ファイルの指定に使用されます。

```
[ WORKDDN filename ]
```

## SORTDEVT 文および SORTNUM 文

SORTDEVT 文および SORTNUM 文は、DB2 との互換性を保つために用意されています。これらの文は、SQL\*Loader によって解析はされますが、無視されます。DB2 では、これらの文は、ソート用の一時データ・セットの番号と型の指定のために使用されます。

```
[ SORTDEVT device_type ]
```

```
[ SORTNUM n ]
```

## DISCARD の指定

複数のファイルを処理する場合は、DISCARD 句（DISCARDN および DISCARDS）を制御ファイル内の別の位置、つまりデータ・ファイル指定の直後に置く必要があります。ただし、DB2 互換のファイルを 1 つのみロードする場合は、これらの句は元の位置（RESUME 句と RECLN 句の間）に置くことができます。なお、DB2 ロード・ユーティリティでは、DISCARDS オプションが 0（ゼロ）の場合は廃棄レコード件数の最大値が設定されていないことを意味します。一方 SQL\*Loader では、このオプションが 0（ゼロ）の場合、レコードが 1 件廃棄されると処理が停止します。

## 制限事項

DB2 ローダーの機能の中には、SQL\*Loader には提供されていないものもあります。たとえば、SQL\*Loader では、SQL/DS ファイルおよび DB2 UNLOAD ファイルからはデータはロードされません。SQL\*Loader では、次の項に示す DB2 ロード・ユーティリティ・コマンドが検出されると、エラーが通知されます。

## FORMAT 文

SQL\*Loader でロードする場合は、制御ファイルに DB2 の FORMAT 文を指定しないでください。DB2 ローダーでは、DB2 UNLOAD 形式、SQL/DS 形式および DB2 ロード・ユーティリティ形式のファイルがロードされます。SQL\*Loader ではこれらの形式はサポートされていません。FORMAT 文がコマンド・ファイル内に指定されていると、SQL\*Loader での処理が停止し、エラーが通知されます (IBM ではこれらのファイルの形式が情報として記録されないため、SQL\*Loader ではファイルの読み込みができません)。

```
FORMAT { UNLOAD | SQL/DS }
```

## PART 文

PART 文は DB2 との互換性を保つために用意されています。Oracle には、DB2 のパーティション表に対応する概念はありません。

DB2 のパーティション表をロードする場合でも、SQL\*Loader では、表全体が読み込まれます。このとき、警告メッセージにより、パーティション表はサポートされていないため表全体がロードされたことが示されます。

```
[ PART n ]
```

## SQL/DS オプション

オプションの SQL/DS=tablename は、WHEN 句では指定できません。これは、SQL\*Loader では SQL/DS 内部形式がサポートされていないためです。WHEN 句を含む文に SQL/DS オプションが指定されていると、SQL\*Loader での処理が停止し、エラーが通知されます。

## DBCS GRAPHIC 型文字列

Oracle データベース・サーバーでは、ダブルバイト・キャラクタ・セット (DBCS) はサポートされていないため、GRAPHIC データ型の文字列 (G'\*\*\*' の形式) は指定できません。

## DB2 と互換性のある文を含む SQL\*Loader の構文

次に示す SQL\*Loader の構文では、DB2 と互換性のある文を太字で示します。

```

OPTIONS (options)
{ LOAD | CONTINUE_LOAD } [DATA]
[ CHARACTERSET character_set_name ]
[ { INFILE | INDDN } { filename | * } ]
[ "OS-dependent file processing options string" ]
[ { BADFILE | BADDN } filename ]
[ { DISCARDFILE | DISCARDN } filename ]
[ { DISCARDS | DISCARDMAX } n ]
[ { INFILE | INDDN } ] ...
[ APPEND | REPLACE | INSERT |
RESUME [( { YES | NO [REPLACE] } []) ]
[ LOG { YES | NO } ]
[ WORKDDN filename ]
[ SORTDEV device_type ]
[ SORTNUM n ]
[ { CONCATENATE [( n []) ] |
CONTINUEIF { [ THIS | NEXT ]
[( ( start [ { : | - } end ) | LAST ]
operator { 'char_str' | X'hex_str' } []) } ]
[ PRESERVE BLANKS ]
INTO TABLE tablename
[ CHARACTERSET character_set_name ]
[ SORTED [ INDEXES ] ( index_name [ , index_name... ] ) ]
[ PART n ]
[ APPEND | REPLACE | INSERT |
RESUME [( { YES | NO [REPLACE] } []) ]
[ REENABLE [DISABLED_CONSTRAINTS] [EXCEPTIONS table_name] ]
[ WHEN field_condition [ AND field_condition ... ] ]
[ FIELDS [ delimiter_spec ] ]
[ TRAILING [ NULLCOLS ] ]
[ SKIP n ]
(.column_name
{ [ RECNUM
| SYSDATE | CONSTANT value
| SEQUENCE ( { n | MAX | COUNT } [ , increment ] )
| [ POSITION ( { start [ { : | - } end ] | * [+n] } ) ]
[ datatype_spec ]
[ NULLIF field_condition ]
[ DEFAULTIF field_condition ]
[ "sql string" ] ] ) }
[ , column_name ] ...)
[ INTO TABLE ] ... [ BEGINDATA ]
[ BEGINDATA]

```





# バックス正規形構文

このマニュアルの各構文図の後には、テキストによる図の説明へのリンクが続きます。テキストによる説明は、[表 C-1](#) に示す記号および表記規則を含む、バックス正規形（BNF）構文の単純な変形です。

**表 C-1 バックス正規形構文の記号および表記規則**

記号または表記規則	意味
[ ]	大カッコは、任意に選択する 1 つ以上の項目を囲みます。
{ }	中カッコは、2 つ以上の項目を囲み、そのうちの 1 つの項目は必須です。
	縦線は、大カッコまたは中カッコ内の選択項目を区切ります。
...	省略記号は、前述の構文要素を繰り返すことができることを示します。
デリミタ	大カッコ、中カッコ、縦線および省略記号以外のデリミタは、表示されているとおりに入力する必要があります。
太字	太字で示されているワードはキーワードです。キーワードは、表示されているとおりに入力する必要があります（オペレーティング・システムによっては、キーワードは大文字と小文字が区別されます）。太字以外のワードは、プレースホルダであり、名前または値を代入する必要があります。



## 数字

16 進文字列

SQL\*Loader, 6-31

8 ビット・キャラクタ・セットのサポート, 1-55, 2-58

## A

ANYDATA 型

表モードのインポート時の影響, 2-14

APPEND パラメータ

SQL\*Loader ユーティリティ, 5-38

## B

BADDN パラメータ

SQL\*Loader ユーティリティ, 5-11

BADFILE パラメータ

SQL\*Loader ユーティリティ, 5-11

BAD パラメータ

SQL\*Loader コマンドライン, 4-4

BEGINDATA パラメータ

SQL\*Loader の制御ファイル, 5-10

BFILE データ型, 7-22

BFILE 列

インポート, 2-62

エクスポート, 1-58

BINDSIZE パラメータ

SQL\*Loader コマンドライン, 4-4, 5-44

BLANKS パラメータ

SQL\*Loader ユーティリティ, 6-31

BUFFER パラメータ

インポート・ユーティリティ, 2-19

エクスポート・ユーティリティ, 1-18

BYTEINT データ型, 6-10

BYTEORDERMARK パラメータ

SQL\*Loader ユーティリティ, 6-39

BYTEORDER パラメータ

SQL\*Loader ユーティリティ, 6-37

## C

catalog.sql スクリプト

インポートのためのデータベースの準備, 2-5

エクスポートのためのデータベースの準備, 1-4

catexp.sql スクリプト

インポートのためのデータベースの準備, 2-5

エクスポートのためのデータベースの準備, 1-4

catldr.sql スクリプト

ダイレクト・パス・ロードの準備, 9-10

CHARACTERSET パラメータ

SQL\*Loader ユーティリティ, 5-20

CHARSET パラメータ

インポート・ユーティリティ, 2-19

CHAR データ型

参照

SQL\*Loader, 6-14

デリミタ付き形式および SQL\*Loader, 6-24

CHAR 列

バージョン 6 のエクスポート・ファイル, 2-73

CHECK 制約

使用禁止の変更, 9-25

CLOB

例, 10-39

COLUMNARRAYROWS パラメータ

SQL\*Loader コマンドライン, 4-4

COMMIT パラメータ

インポート・ユーティリティ, 2-20

COMPILE パラメータ

インポート・ユーティリティ, 2-20

COMPRESS パラメータ  
    エクスポート・ユーティリティ, 1-18  
CONCATENATE パラメータ  
    SQL\*Loader ユーティリティ, 5-26  
CONSISTENT パラメータ  
    エクスポート・ユーティリティ, 1-19  
        ネストした表, 1-19  
        パーティション表, 1-19  
CONSTANT パラメータ  
    SQL\*Loader, 6-53  
CONSTRAINTS パラメータ  
    インポート・ユーティリティ, 2-21  
    エクスポート・ユーティリティ, 1-21  
CONTINUEIF パラメータ  
    SQL\*Loader ユーティリティ, 5-26  
    例, 10-15  
CONTROL パラメータ  
    SQL\*Loader コマンドライン, 4-5  
CREATE SESSION 権限  
    インポート・ユーティリティ, 2-6  
    エクスポート・ユーティリティ, 1-5  
CREATE USER コマンド  
    インポート・ユーティリティ, 2-73

## D

---

DATAFILES パラメータ  
    インポート・ユーティリティ, 2-21  
DATA パラメータ  
    SQL\*Loader コマンドライン, 4-5  
DATE\_CACHE パラメータ  
    SQL\*Loader ユーティリティ, 4-5  
DATE データ型  
    SQL\*Loader, 6-16  
    デリミタ付き形式および SQL\*Loader, 6-24  
    長さの決定, 6-28  
    マスク  
        SQL\*Loader, 6-28  
DB2 の FORMAT 文  
    SQL\*Loader では使用不可, B-4  
DB2 の PART 文  
    SQL\*Loader では使用不可, B-4  
DB2 ロード・ユーティリティ  
    RESUME パラメータ, 5-32  
    SQL\*Loader 機能の制限, B-3  
    SQL\*Loader の互換性  
        処理されない文, B-2

    文の配置  
        DISCARD DDN, B-3  
        DISCARDS, B-3  
DBA ロール  
    EXP\_FULL\_DATABASE ロール, 1-5  
DBCS (DB2 ダブル・バイト・キャラクタ・セット)  
    Oracle では未サポート, B-4  
DBID (データベース識別子)  
    変更, 14-3  
DBMS\_METADATA パッケージ, 15-4  
    セキュリティ, 15-4  
    ブラウザ・インタフェース, 15-10  
    プログラム・インタフェース, 15-5  
    メタデータ API の実装に対する使用, 15-4  
DBNAME  
    変更, 14-4  
DBNEWID ユーティリティ, 14-1  
    構文, 14-7  
    制限事項, 14-8  
    停止した変更操作の回復, 14-6  
    データベース ID の変更, 14-3  
    データベース ID の変更のトラブルシューティング,  
        14-6  
    データベース名の変更, 14-4  
    データベース名の変更のトラブルシューティング,  
        14-6  
    例, 14-9  
DBVERIFY ユーティリティ  
    構文, 13-2  
    出力例, 13-3  
    制限事項, 13-1  
    セグメントの検査, 13-4  
    ディスク・ブロックの検査, 13-2  
DECIMAL データ型, 6-11  
    EXTERNAL 形式  
        SQL\*Loader, 6-19  
DEFAULTIF パラメータ  
    SQL\*Loader, 6-29  
DELETE CASCADE  
    SQL\*Loader, 5-33  
    空でない表へのデータのロードの影響, 5-32  
DELETE 権限  
    SQL\*Loader, 5-32  
DESTROY パラメータ  
    インポート・ユーティリティ, 2-21  
DIRECT パラメータ  
    エクスポート・ユーティリティ, 1-21

DISCARDDN パラメータ  
DB2 の制御ファイル, B-3  
DISCARDMAX パラメータ  
SQL\*Loader コマンドライン, 4-6  
DISCARDS パラメータ  
DB2 の制御ファイル, B-3  
DISCARD パラメータ  
SQL\*Loader コマンドライン, 4-6  
DOUBLE データ型, 6-10  
DROP ANY TABLE 権限  
SQL\*Loader, 5-33

## E

---

EBCDIC キャラクタ・セット  
インポート・ユーティリティ, 2-58  
ERRORS パラメータ  
SQL\*Loader コマンドライン, 4-7  
EVALUATE\_CHECK\_CONSTRAINTS 句, 9-25  
EXP\_FULL\_DATABASE ロール  
エクスポートでの割当て, 1-5  
expdat.dmp  
エクスポート出力ファイル, 1-22  
EXPRESSION パラメータ  
SQL\*Loader, 6-53  
EXTERNAL SQL\*Loader データ型, 6-19  
DECIMAL, 6-19  
FLOAT, 6-19  
GRAPHIC, 6-18  
ZONED, 6-19  
数値型, 6-19  
長さの決定, 6-27  
EXTERNAL パラメータ  
SQL\*Loader, 6-19

## F

---

FEEDBACK パラメータ  
インポート・ユーティリティ, 2-22  
エクスポート・ユーティリティ, 1-21  
FILESIZE パラメータ  
インポート・ユーティリティ, 2-22  
エクスポート・ユーティリティ, 1-22  
FILE パラメータ  
SQL\*Loader ユーティリティ, 9-32  
インポート・ユーティリティ, 2-22  
エクスポート・ユーティリティ, 1-22

FILLER フィールド  
引数としての init\_spec の使用, 6-6  
例, 10-39  
FLASHBACK\_SCN パラメータ  
エクスポート・ユーティリティ, 1-23  
FLASHBACK\_TIME パラメータ  
エクスポート・ユーティリティ, 1-24  
FLOAT EXTERNAL データ値  
SQL\*Loader, 6-19  
FLOAT EXTERNAL の科学表記法, 6-19  
FLOAT データ型, 6-9  
EXTERNAL 形式  
SQL\*Loader, 6-19

FROMUSER パラメータ  
インポート・ユーティリティ, 2-23  
FTP  
エクスポート・ファイルの転送, 1-54  
FULL パラメータ  
インポート・ユーティリティ, 2-23  
エクスポート・ユーティリティ, 1-24

## G

---

GRAPHIC データ型  
SQL\*Loader の EXTERNAL 形式, 6-18  
GRANTS パラメータ  
インポート・ユーティリティ, 2-24  
エクスポート・ユーティリティ, 1-24

## H

---

HELP パラメータ  
インポート・ユーティリティ, 2-24  
エクスポート・ユーティリティ, 1-14, 1-24

## I

---

IGNORE パラメータ  
インポート・ユーティリティ, 2-24  
IMP\_FULL\_DATABASE ロール, 2-5  
INDEXES パラメータ  
インポート・ユーティリティ, 2-25  
エクスポート・ユーティリティ, 1-24  
INDEXFILE パラメータ  
インポート・ユーティリティ, 2-25  
INFILE パラメータ  
SQL\*Loader ユーティリティ, 5-7

INTEGER データ型, 6-8  
EXTERNAL 形式, 6-19  
INTO TABLE 文  
SQL\*Loader, 5-30  
廃棄, 5-15  
列名, 6-5  
SQL\*Loader を使用した複数の文, 5-38  
バインド配列サイズへの影響, 5-49

## L

---

Length-Value Pair で指定した LOB, 7-27  
LENGTH-VALUE データ型, 6-7  
length サブフィールド  
VARCHARDATA  
SQL\*Loader, 6-12  
LOAD パラメータ  
SQL\*Loader コマンドライン, 4-9  
LOB  
ロード, 7-18  
LOBFILE, 3-8, 7-18, 7-23  
例, 10-39  
LOB データ, 3-8  
Length-Value Pair フィールド, 7-21  
エクスポート中の圧縮, 1-19  
エクスポート・ユーティリティ, 1-57  
事前にサイズが決められたフィールド, 7-19  
デリミタ付きフィールド, 7-20  
LOB 読み込みバッファ  
サイズ, 4-11  
LOG パラメータ  
SQL\*Loader コマンドライン, 4-9  
インポート・ユーティリティ, 2-26  
エクスポート・ユーティリティ, 1-25  
LONG VARRAW データ型, 6-14  
LONG データ  
C 言語データ型 LONG FLOAT, 6-10  
インポート, 2-63  
エクスポート, 1-57

## M

---

MULTITHREADING パラメータ  
SQL\*Loader コマンドライン, 4-9

## N

---

NOLOG 属性, 9-20  
NOT NULL 制約  
ロード方法, 9-10  
NULL  
アトミック, 7-7  
属性, 7-6  
NULLIF...BLANKS 句  
SQL\*Loader, 6-31  
例, 10-26  
NULLIF 句  
SQL\*Loader, 6-29, 6-40  
NULL 値  
オブジェクト, 7-6  
NULL データ  
指定されていない列および SQL\*Loader, 6-5  
ロード中のレコードの終わりの桁の欠落, 5-36  
NUMBER データ型  
SQL\*Loader, 6-22, 6-23

## O

---

OBJECT\_CONSISTENT パラメータ  
エクスポート・ユーティリティ, 1-25  
OID  
「オブジェクト識別子」を参照  
OPTIMAL 記憶域パラメータ  
インポート・ユーティリティ, 2-68  
OPTIONALLY ENCLOSED BY 句  
SQL\*Loader, 6-44  
OPTIONS パラメータ  
SQL\*Loader ユーティリティ, 5-4  
パラレル・ロード, 5-33  
Oracle Net  
ネットワークを介したエクスポート, 1-54  
Oracle アドバンスド・キューイング  
「アドバンスド・キューイング」を参照  
Oracle バージョン 6  
データベース・オブジェクトのエクスポート, 2-73  
OWNER パラメータ  
エクスポート・ユーティリティ, 1-25

## P

---

PARALLEL パラメータ, 9-31  
SQL\*Loader コマンドライン, 4-10

PARFILE パラメータ

SQL\*Loader コマンドライン, 4-10

インポート・コマンドライン, 2-26

エクスポート・コマンドライン, 1-25

PIECED パラメータ

SQL\*Loader, 9-16

POSITION パラメータ

タブを含むデータでの使用, 6-4

複数の SQL\*Loader の INTO TABLE 句, 5-40, 6-3, 6-4

PRESERVE BLANKS オプション

SQL\*Loader, 6-48

PRESERVE パラメータ, 5-28

## Q

---

QUERY パラメータ

エクスポート・ユーティリティ, 1-26

制限事項, 1-26

## R

---

RAW データ型

SQL\*Loader, 6-19

READSIZE パラメータ

SQL\*Loader コマンドライン, 4-11

LOB への影響, 4-11

最大サイズ, 4-11

RECNUM パラメータ

SQL\*Loader の SKIP パラメータでの使用, 6-54

RECORDLENGTH パラメータ

インポート・ユーティリティ, 2-26

エクスポート・ユーティリティ, 1-27

REDO ログ

インスタンスおよびメディア・リカバリ

SQL\*Loader, 9-15

ダイレクト・パス・ロード, 9-15

ダイレクト・パス・ロード中の使用の最小化, 9-19

領域の節約

ダイレクト・パス・ロード, 9-20

REF データ

インポート, 2-61

REF フィールド

例, 10-43

REF 列, 7-15

実, 7-15

主キー, 7-16

ロード, 7-15

REPLACE 表

SQL\*Loader を使用した表の置換え, 5-32  
例, 10-15

RESOURCE ロール, 2-6

RESUMABLE\_NAME パラメータ

SQL\*Loader ユーティリティ, 4-12

インポート・ユーティリティ, 2-27

エクスポート・ユーティリティ, 1-28

RESUMABLE\_TIMEOUT パラメータ

SQL\*Loader ユーティリティ, 4-12

インポート・ユーティリティ, 2-27

エクスポート・ユーティリティ, 1-28

RESUMABLE パラメータ

SQL\*Loader ユーティリティ, 4-11

インポート・ユーティリティ, 2-27

エクスポート・ユーティリティ, 1-27

RESUME パラメータ

DB2, 5-32, B-2

ROWS パラメータ

SQL\*Loader コマンドライン, 4-12

インポート・ユーティリティ, 2-28

エクスポート・ユーティリティ, 1-28

データ・セーブ時の指定での使用, 9-13

パフォーマンスの問題

SQL\*Loader, 9-19

## S

---

SDF

「セカンダリ・データ・ファイル」を参照

SEQUENCE パラメータ

SQL\*Loader, 6-54

SHORTINT データ型

C 言語, 6-9

SHOW パラメータ

インポート・ユーティリティ, 2-28

SILENT パラメータ

SQL\*Loader コマンドライン, 4-13

SINGLEROW パラメータ, 5-38, 9-24

SKIP\_INDEX\_MAINTENANCE パラメータ

SQL\*Loader コマンドライン, 4-14, 9-24

SKIP\_UNUSABLE\_INDEXES パラメータ

SQL\*Loader コマンドライン, 4-14, 9-24

インポート・ユーティリティ, 2-28

SKIP パラメータ

SQL\*Loader RECNUM 指定への影響, 6-54

SQL\*Loader コマンドライン, 4-14  
SMALLINT データ型, 6-9  
SORTED INDEXES 句  
    SQL\*Loader, 9-18  
    ダイレクト・パス・ロード, 5-37  
    例, 10-26  
SQL\*Loader  
    BADDN パラメータ, 5-11  
    BADFILE パラメータ, 5-11  
    BAD コマンドライン・パラメータ, 4-4  
    BINDSIZE コマンドライン・パラメータ, 4-4, 5-44  
    COLUMNARRAYROWS コマンドライン・パラメータ, 4-4  
    CONCATENATE パラメータ, 5-26  
    CONTINUEIF パラメータ, 5-26  
    CONTROL コマンドライン・パラメータ, 4-5  
    DATA コマンドライン・パラメータ, 4-5  
    DATE\_CACHE コマンドライン・パラメータ, 4-5  
    DIRECT コマンドライン・パラメータ, 9-11  
    DISCARDFILE パラメータ, 5-14  
    DISCARDMAX コマンドライン・パラメータ, 4-6  
    DISCARDMAX パラメータ, 5-16  
    DISCARDS パラメータ, 5-16  
    DISCARD コマンドライン・パラメータ, 4-6  
    ERRORS コマンドライン・パラメータ, 4-7  
    FILE コマンドライン・パラメータ, 4-9  
    INTO TABLE 文, 5-30  
    LOAD コマンドライン・パラメータ, 4-9  
    LOG コマンドライン・パラメータ, 4-9  
    MULTITHREADING コマンドライン・パラメータ, 4-9  
    PARFILE コマンドライン・パラメータ, 4-10  
    READSIZE コマンドライン・パラメータ, 4-11  
        最大サイズ, 4-11  
    RESUMABLE\_NAME パラメータ, 4-12  
    RESUMABLE\_TIMEOUT パラメータ, 4-12  
    RESUMABLE パラメータ, 4-11  
    ROWS コマンドライン・パラメータ, 4-12  
    SILENT コマンドライン・パラメータ, 4-13  
    SINGLEROW パラメータ, 5-38  
    SKIP\_INDEX\_MAINTENANCE コマンドライン・パラメータ, 4-14  
    SKIP\_UNUSABLE\_INDEXES コマンドライン・パラメータ, 4-14  
    STREAMSIZE コマンドライン・パラメータ, 4-15  
    USERID コマンドライン・パラメータ, 4-15  
    オブジェクト表のロード, 7-12

オブジェクト名, 5-5  
行の更新, 5-33  
拒否レコード, 3-9  
グローバリゼーション・テクノロジー, 5-16  
異なるプラットフォーム間でのデータのロード, 6-35  
    コマンドライン・パラメータ, 4-2  
    索引オプション, 5-37  
    従来型パス・ロード, 9-4  
    事例, 10-1  
        LOBFILE のロード (CLOB), 10-39  
        REF フィールドのロード, 10-43  
        Unicode キャラクタ・セットのデータのロード, 10-48  
        VARRAY のロード, 10-43  
        可変長データのロード, 10-6  
        結合した物理レコードのロード, 10-15  
        固定長データのロード, 10-9  
        自由区分形式ファイルのロード, 10-12  
        書式化されたレポートからのデータの抽出, 10-29  
        ダイレクト・パス・ロード, 10-25  
        パーティション表のロード, 10-34  
        複数表へのデータのロード, 10-19  
制御ファイルへのロード・データの組込み, 6-52  
セッションの例, 10-1  
ダイレクト・パスによる方法, 3-12  
    日付キャッシュ機能を使用したパフォーマンスの向上, 9-21  
ダイレクト・パス・ロードでの SORTED INDEXES, 5-37  
    タブが原因で発生するエラー, 6-4  
    単一表へのロードの継続, 5-26  
    データ型の指定, 3-9  
    データ定義言語  
        構文図, A-1  
    データ・ファイルの指定, 5-7  
    データ変換, 3-9  
    データをロードする方法, 3-11  
    廃棄レコード, 3-9  
    排他的アクセス, 9-29  
    バインド配列およびパフォーマンス, 5-44  
    パラレル・データ・ロード, 9-29, 9-30, 9-34  
    必要な権限, 9-2  
    表中の行の置換え, 5-32  
    表への行の挿入, 5-32  
    表への行の追加, 5-32



ファイル名, 5-5  
フィールド条件の指定, 6-29  
フィールドの指定, 6-5  
複数の INTO TABLE 文, 5-38  
複数のデータ・ファイルの指定, 5-9  
不良ファイル, 4-4  
メッセージの抑止, 4-13  
列オブジェクトのロード, 7-2  
列の指定, 6-5  
ロードする行の選択, 5-34  
ロード方法, 9-2  
ログ・ファイル, 3-11  
ログ・ファイル・エントリ, 8-1  
ログ・ファイルのグローバル情報, 8-2  
ログ・ファイルのサマリー統計, 8-6  
ログ・ファイルのデータ・ファイル情報, 8-5  
ログ・ファイルの表情報, 8-3  
ログ・ファイルの表ロード情報, 8-5  
ログ・ファイルのヘッダー情報, 8-2  
SQL/DS オプション (DB2 ファイル形式)  
SQL\*Loader では未サポート, B-4  
SQL 演算子  
フィールドへの適用, 6-48  
SQL 文字列  
SQL 演算子のフィールドへの適用, 6-48  
引用符, 5-5  
例, 10-29  
STATISTICS パラメータ  
インポート・ユーティリティ, 2-29  
エクスポート・ユーティリティ, 1-28  
STORAGE パラメータ, 9-33  
STREAMS\_CONFIGURATION パラメータ  
インポート・ユーティリティ, 2-29  
STREAMS\_INSTANTIATION パラメータ  
インポート・ユーティリティ, 2-30  
STREAMSIZE パラメータ  
SQL\*Loader コマンドライン, 4-15  
SYSDATE データ型  
例, 10-29  
SYSDATE パラメータ  
SQL\*Loader, 6-54

## T

---

TABLESPACES パラメータ  
インポート・ユーティリティ, 2-32  
エクスポート・ユーティリティ, 1-31

TABLES パラメータ  
インポート・ユーティリティ, 2-30  
エクスポート・ユーティリティ, 1-29  
TERMINATED BY  
SQL\*Loader, 6-24  
WHITESPACE  
SQL\*Loader, 6-24  
TERMINATED BY 句  
OPTIONALLY ENCLOSED BY 付き, 6-44  
TOID\_NOVALIDATE パラメータ  
インポート・ユーティリティ, 2-32  
TOUSER パラメータ  
インポート・ユーティリティ, 2-33  
TRAILING NULLCOLS パラメータ  
SQL\*Loader ユーティリティ, 5-3, 5-37  
例, 10-29  
TRANSPORT\_TABLESPACE パラメータ  
インポート・ユーティリティ, 2-34  
エクスポート・ユーティリティ, 1-31  
TRIGGERS パラメータ  
エクスポート・ユーティリティ, 1-32  
TTS\_FULL\_CHECK パラメータ  
エクスポート・ユーティリティ, 1-32  
TTS\_OWNERS パラメータ  
インポート・ユーティリティ, 2-34

## U

---

UNLOAD 文 (DB2 ファイル形式)  
SQL\*Loader では未サポート, B-4  
UNRECOVERABLE パラメータ  
SQL\*Loader, 9-20  
USER\_SEGMENTS ビュー  
エクスポート, 1-5  
USERID パラメータ  
SQL\*Loader コマンドライン, 4-15  
インポート・ユーティリティ, 2-34  
エクスポート・ユーティリティ, 1-32

## V

---

VALUE データ型, 6-7  
VARCHAR2 データ型, 2-73  
SQL\*Loader, 6-22  
VARRAWC データ型, 6-20  
VARCHARC データ型  
SQL\*Loader, 6-20

VARCHAR データ型

SQL\*Loader, 6-12

VARGRAPHIC データ型

SQL\*Loader, 6-11

VARRAW データ型, 6-13

VARRAY 列

ロード時のメモリーの問題, 7-34

VOLSIZE パラメータ

インポート・ユーティリティ, 2-35

エクスポート・ユーティリティ, 1-33

## W

---

WHEN 句

SQL\*Loader, 5-34, 6-29

SQL\*Loader の廃棄, 5-15

例, 10-19

WHITESPACE パラメータ

SQL\*Loader, 6-24

## X

---

XML 列

SQL\*Loader による処理, 7-18

ロード, 7-18

## Z

---

ZONED データ型, 6-10

EXTERNAL 形式

SQL\*Loader, 6-19

## あ

---

アーカイブ

使用禁止

ダイレクト・パス・ロードの影響, 9-20

アクセス権限

インポート・ユーティリティ, 2-6

エクスポート・ユーティリティ, 1-5

アドバンスト・キューイング

アドバンスト・キュー表のインポート, 2-63

アドバンスト・キュー表のエクスポート, 1-59

アトミック NULL, 7-7

アナライザ統計, 2-70

## い

---

一意制約

インポート中のエラー防止, 2-20

ダイレクト・パス・ロードの影響, 9-34

一意の値

SQL\*Loader を使用した生成, 6-54

一時セグメント, 9-32

FILE パラメータ

SQL\*Loader, 9-32

インスタンス親和性

エクスポート・ユーティリティ, 1-57

インスタンス・リカバリ, 9-15

インポート・ユーティリティ

BUFFER パラメータ, 2-19

catexp.sql スクリプト

データベースの準備, 2-5

CHARSET パラメータ, 2-19

COMMIT パラメータ, 2-20

COMPILE パラメータ, 2-20

CONSTRAINTS パラメータ, 2-21

DATAFILES パラメータ, 2-21

DESTROY パラメータ, 2-21

FEEDBACK パラメータ, 2-22

FILESIZE パラメータ, 2-22

FILE パラメータ, 2-22

FROMUSER パラメータ, 2-23

FULL パラメータ, 2-23

GRANTS パラメータ, 2-24

HELP パラメータ, 2-24

IGNORE パラメータ, 2-24

INDEXES パラメータ, 2-25

INDEXFILE パラメータ, 2-25

INSERT エラー, 2-49

LOG パラメータ, 2-26

LONG 列, 2-63

OPTIMAL 記憶域パラメータ, 2-68

Oracle バージョン 6 エクスポート・ファイルのデ

フォルト列の長さ, 2-73

Oracle バージョン 6 ファイルの使用, 2-73

PARFILE パラメータ, 2-26

RECORDLENGTH パラメータ, 2-26

RESUMABLE\_NAME パラメータ, 2-27

RESUMABLE\_TIMEOUT パラメータ, 2-27

RESUMABLE パラメータ, 2-27

ROWS パラメータ, 2-28

SHOW パラメータ, 2-28

- SKIP\_UNUSABLE\_INDEXES パラメータ, 2-28
- STATISTICS パラメータ, 2-29
- STREAMS\_CONFIGURATION パラメータ, 2-29
- STREAMS\_INSTANTIATION パラメータ, 2-30
- TABLESPACES パラメータ, 2-32
- TABLES パラメータ, 2-30
- TOID\_NOVALIDATE パラメータ, 2-32
- TOUSER パラメータ, 2-33
- TRANSPORT\_TABLESPACE パラメータ, 2-34
- TTS\_OWNER パラメータ, 2-34
- USERID パラメータ, 2-34
- VOLSIZE パラメータ, 2-35
- 一意制約
  - インポート・エラーの防止, 2-20
- インポート実行前の手動による表の作成, 2-9
- エクスポート・ファイル
  - インポート実行前の内容のリスト, 2-28
  - ファイル全体のインポート, 2-23
- エクスポート・ファイルの指定, 2-22
- エラーのタイプ, 2-49
- オブジェクト作成エラー, 2-24
- オンライン・ヘルプの表示, 2-24
- 環境変数 NLS\_LANG, 2-58
- 記憶域パラメータ
  - 上書き, 2-69
- 既存のデータ・ファイルの再利用, 2-21
- 起動, 2-11
  - コマンドライン, 2-11
  - 対話方式, 2-11
  - パラメータ・ファイル, 2-11
- キャラクタ・セット, 2-57
- キャラクタ・セット変換, 2-58
- 行
  - インポート対象にする場合の指定, 2-28
- 行のインポート, 2-28
- グローバルゼーションに関する考慮点, 2-57
- 警告メッセージ, 2-47
- 権限
  - インポート対象にする場合の指定, 2-24
- 権限のインポート, 2-24
- 索引作成 SQL スクリプトの作成, 2-25
- 索引作成コマンドの指定, 2-25
- 参照制約の使用禁止, 2-9
- システム・オブジェクト, 2-8
- 終了コード, 2-48
- 出力のログ・ファイルへのリダイレクト, 2-47
- 手動による表の順序付け, 2-10

- 順序, 2-50
- 初期エクステンツのサイズの保存, 2-69
- シングルバイト・キャラクタ・セット, 2-58
- スキーマ・オブジェクト, 2-7
- ストアド・パッケージ, 2-62
- ストアド・ファンクション, 2-62
- ストアド・プロシージャ, 2-62
- スナップショット, 2-65
  - 削除された場合のリストア, 2-66
- スナップショット・マスター表, 2-66
- 制限
  - 自分のスキーマへのインポート, 2-6
- 整合性制約違反, 2-49
- 整理統合されたエクステンツ, 2-69
- セッションの例, 2-35
  - あるユーザーから別のユーザーへのインポート, 2-38
  - 特定のユーザー用に選択された表, 2-36
  - パーティション・レベル・インポートの使用, 2-39
  - 別のユーザーによってエクスポートされた表, 2-37
- 対話方式, 2-45
- データベース・オブジェクトのインポートでのエラー, 2-50
- データベース・オブティマイザ統計, 2-29
- データベース断片化の解消, 2-55
- データベースの準備, 2-5
- バージョン 6 の CHAR 列から VARCHAR2 への変換, 2-73
- パーティション・レベル, 2-51
- 配列挿入後のコミット, 2-20
- パラメータ, 2-15
- パラメータ・ファイル, 2-26
  - 最大サイズ, 2-12
- 表オブジェクト
  - インポート順序, 2-4
- 表のインポート, 2-30
- 表名に関する制限, 2-32
- 表名のパターン一致, 2-30
- 表領域の再編成, 2-70
- 表領域の削除, 2-69
- 表レベル, 2-51
- 他のスキーマへのオブジェクトのインポート, 2-7
- 他のバージョンとの互換性, 2-3
- 無効なデータ, 2-49
- メッセージ・ログ・ファイル, 2-47

- モード, 2-14
- ユーザーごとの指定, 2-23
- ユーザー定義, 2-73
- 読込み専用表領域, 2-69
- リソース・エラー, 2-51
- リフレッシュ・エラー, 2-66
- レコード
  - 長さの指定, 2-26
- ロールバック・セグメントのサイズの制御, 2-20
- 引用符
  - SQL 文字列, 5-5
  - エスケープ, 5-6
  - データベース・オブジェクト名との使用, 5-5
- 表名, 1-30, 2-32
- ファイル名, 5-5

## え

---

- エクステンツ
  - 整理統合, 1-18
  - 整理統合された場合のインポート, 2-69
- エクスポート
  - 読込み専用データベース, 1-62
- エクスポート・ファイル
  - インポート実行前の内容のリスト, 2-28
  - 指定, 1-22
  - ファイル全体のインポート, 2-23
- エクスポート・メッセージ
  - 完了, 1-50
  - 警告, 1-49
  - リカバリ不能, 1-49
- エクスポート・ユーティリティ
  - 8 ビット・キャラクタ・セットおよび7 ビット・キャラクタ・セット, 1-55
  - BUFFER パラメータ, 1-18
  - COMPRESS パラメータ, 1-18
  - CONSISTENT パラメータ, 1-19
  - CONSTRAINTS パラメータ, 1-21
  - DIRECT パラメータ, 1-21
  - FEEDBACK パラメータ, 1-21
  - FILESIZE パラメータ, 1-22
  - FILE パラメータ, 1-22
  - FLASHBACK\_SCN パラメータ, 1-23
  - FLASHBACK\_TIME パラメータ, 1-24
  - FULL パラメータ, 1-24
  - GRANTS パラメータ, 1-24
  - HELP パラメータ, 1-24
  - INDEXES パラメータ, 1-24
  - LOG パラメータ, 1-25
  - LONG 列, 1-57
  - OBJECT\_CONSISTENT パラメータ, 1-25
  - Oracle7 のエクスポート・ファイルの作成, 1-64
  - OWNER パラメータ, 1-25
  - PARFILE パラメータ, 1-25
  - QUERY パラメータ, 1-26
  - RECORDLENGTH パラメータ, 1-27
  - RESUMABLE\_NAME パラメータ, 1-28
  - RESUMABLE\_TIMEOUT パラメータ, 1-28
  - RESUMABLE パラメータ, 1-27
  - ROWS パラメータ, 1-28
  - STATISTICS パラメータ, 1-28
  - TABLESPACES パラメータ, 1-31
  - TABLES パラメータ, 1-29
  - TRANSPORT\_TABLESPACE パラメータ, 1-31
  - TRIGGERS パラメータ, 1-32
  - TTS\_FULL\_CHECK パラメータ, 1-32
  - USERID パラメータ, 1-32
  - VOLSIZE パラメータ, 1-33
  - エクスポート順序番号, 1-57
  - エラー・メッセージのロギング, 1-25
  - オンライン・ヘルプ, 1-14
  - オンライン・ヘルプの表示, 1-24
  - 記憶要件, 1-5
  - 起動, 1-6
  - キャラクタ・セットの変換, 1-55
  - 権限に基づく制限事項, 1-5
  - 索引のエクスポート, 1-24
  - 作成
    - 必要な権限, 1-5
    - 必要なビュー, 1-5
  - シノニムのエクスポート, 1-60
  - 従来型パス, 1-51
  - 終了コード, 1-50
  - 出力のログ・ファイルへのリダイレクト, 1-49
  - 順序番号, 1-57
  - セッションの例, 1-33
    - 全データベース・モード, 1-34
    - パーティション・レベル, 1-41
    - 表モード, 1-38
    - ユーザー・モード, 1-37
  - 全データベース・モード
    - 例, 1-34
  - ダイレクト・パス, 1-51
  - 対話方式, 1-44

- データベース移行のパーティション化, 1-62
- データベース・オブティマイザ統計, 1-28
- データベース全体のエクスポート, 1-24
- ネットワークに関する問題, 1-54
- ネットワークを介したエクスポート・ファイルの転送, 1-54
- パラメータ, 1-14
- パラメータ競合, 1-33
- パラメータ・ファイル, 1-25
  - 最大サイズ, 1-7
- 表名に関する制限, 1-30
- 表モード
  - セッションの例, 1-38
- 別のオペレーティング・システムへのエクスポート, 1-27, 2-26
- モード
  - 各モードでエクスポートされるオブジェクト, 1-9
- ユーザー・アクセス権限, 1-5
- ユーザー・モード
  - 指定, 1-25
  - セッションの例, 1-37
- リモート操作, 1-54, 2-56
- ログ・ファイル
  - 指定, 1-25
- エスケープ文字
  - インポートでの使用方法, 2-31
  - 引用符付き文字列, 5-6
  - エクスポートでの使用方法, 1-30
- エラー
  - DB2 ロード・ユーティリティによる生成, B-3
  - LONG データ, 2-49
  - SQL\*Loader データ中のタブ文字が原因, 6-4
  - インポート中の行エラー, 2-49
  - インポート・ユーティリティ, 2-47
  - インポート・リソース・エラー, 2-51
  - オブジェクト作成, 2-50
    - インポート・パラメータ IGNORE, 2-24
  - 警告
    - インポート・ユーティリティ, 2-47
    - エクスポート・ユーティリティ, 1-49
  - リカバリ可能
    - インポート・ユーティリティ, 2-47
    - エクスポート・ユーティリティ, 1-49
  - リカバリ不能
    - インポート・ユーティリティ, 2-47
    - エクスポート・ユーティリティ, 1-49

- ログ・ファイルのエクスポート, 1-25

## お

---

- オブジェクト, 3-12
  - NULL 値, 7-6
  - インポート作成エラー, 2-24
  - インポート中の既存オブジェクトの無視, 2-24
  - インポートに関する考慮点, 2-58
  - 可変レコード形式, 7-3
  - 作成エラー, 2-50
  - ストリーム・レコード形式, 7-2
  - ネストした列オブジェクトのロード, 7-4
- オブジェクト型定義
  - エクスポート, 1-59
- オブジェクト・サポート, 3-14
- オブジェクト識別子, 7-12
  - インポート, 2-58
- オブジェクト表
  - サブタイプの使用
  - ロード, 7-14
  - ロード, 7-12
- オブジェクト名
  - SQL\*Loader, 5-5
- オブティマイザ統計, 2-70
- オフライン・ローカル管理表領域
  - エクスポート, 1-58
- オペレーティング・システム
  - SQL\*Loader の使用、異なるシステムへのデータの移動, 6-35
- オンライン・ヘルプ
  - インポート・ユーティリティ, 2-15
  - エクスポート・ユーティリティ, 1-14

## か

---

- 外部 LOB (BFILE), 7-22
- 外部関数ライブラリ
  - インポート, 2-62, 2-63
  - エクスポート, 1-58
- 外部表
  - field\_definitions 句, 12-2, 12-14
  - record\_format\_info 句, 12-2, 12-3
  - SQL\*Loader との処理内容の違い, 11-7
  - アクセス・パラメータ, 12-1
  - 可変長レコード, 12-4
  - キャラクタ・セットの識別, 12-6

- 空白の切捨て, 12-18
- 固定長レコード, 12-4
- コメントの使用, 12-2
- コンストラクタ・ファンクションの使用, 11-5
- 使用時のパフォーマンスの向上, 11-6
  - 日付キャッシュ機能, 11-6
- 制限事項, 11-3
- データ型, 12-22
- データ型の識別, 12-19
- データのロード時のレコードのスキップ, 12-9
- データのロードでの使用, 11-5
- デリミタ, 12-5
- デリミタの指定, 12-15
- ビッグ・エンディアン・データ, 12-6
- 日付キャッシュ機能, 11-6
- フィールドの NULL 設定, 12-29
- フィールドのデータ型の記述, 12-22
- フィールドのデフォルト値設定, 12-29
- リトル・エンディアン・データ, 12-6
- ロード条件の指定, 12-8
- 外部ファイル
  - エクスポート, 1-58
- 囲まれたフィールド
  - 囲みデリミタおよび SQL\*Loader による指定, 6-24
  - 空白, 6-47
- 囲みデリミタ, 6-24
  - SQL\*Loader, 6-24, 6-44
- 可変長レコード
  - 外部表, 12-4
- 可変レコード, 3-5
  - 形式, 7-3
- 環境変数 NLS\_LANG, 2-57
  - インポート・ユーティリティ, 2-58
  - エクスポート・ユーティリティ, 1-55
- 完了メッセージ
  - インポート・ユーティリティ, 2-48
  - エクスポート・ユーティリティ, 1-50

## き

---

- キー値
  - SQL\*Loader を使用した生成, 6-54
- 記憶域パラメータ, 2-68
  - OPTIMAL パラメータ, 2-68
- 上書き
  - インポート・ユーティリティ, 2-69
- エクスポート要件の見積り, 1-5

- 事前割当て
  - ダイレクト・パス・ロード, 9-17
  - ダイレクト・パス・ロード時の一時記憶域パラメータ, 9-12
- 期間データ型, 6-15
- 起動
  - インポート・ユーティリティ, 2-11
  - SYSDBA, 2-13
  - コマンドライン, 2-11
  - 対話方式, 2-11
  - パラメータ・ファイル, 2-11
- エクスポート・ユーティリティ, 1-6
  - SYSDBA, 1-8
  - コマンドライン, 1-6
  - ダイレクト・パス, 1-53
  - 対話方式, 1-6
  - パラメータ・ファイル, 1-6
- キャッシュされる順序番号
  - エクスポート・ユーティリティ, 1-57
- キャラクタ・セット
  - 8 ビットから 7 ビットへの変換
    - エクスポートおよびインポート, 1-55, 2-58
  - SQL\*Loader 制御ファイル, 5-21
  - SQL\*Loader での変換, 5-16
  - Unicode, 5-17, 10-48
  - 外部表の識別, 12-6
  - シングルバイト
    - エクスポートおよびインポート, 1-55, 2-58
  - バージョン 6 の変換
    - インポート / エクスポート, 2-58
  - 変換
    - エクスポートおよびインポート中, 1-55, 2-57
  - マルチバイト
    - SQL\*Loader, 5-17
- キャラクタ・セットの変換
  - エクスポートおよびインポート中, 1-55
  - キャラクタ・セットのソートの影響, 1-56
- 行
  - SQL\*Loader を使用した既存の行の更新, 5-32
  - SQL\*Loader を使用してロードする場合の選択, 5-34
  - インポート対象にする場合の指定, 2-28
  - エクスポート, 1-28
  - セーブ前に挿入する数の指定
    - SQL\*Loader, 9-13
- 行エラー
  - インポート・ユーティリティ, 2-49

拒否ファイル  
    SQL\*Loader の指定, 5-11  
拒否レコード  
    SQL\*Loader, 3-9, 5-11  
切捨て  
    後続の空白  
        SQL\*Loader, 6-47  
    サマリー, 6-42

## く

---

空白  
    切捨て, 6-41  
        外部表, 12-18  
    空白, 6-41  
    空白のフィールドのロード, 6-40  
    後続, 6-27  
    先頭, 6-43  
    そのまま残す, 6-47  
    フィールドに含まれた, 6-46  
    フィールドの終了, 6-24, 6-46  
    フィールド比較用の SQL\*Loader BLANKS パラメータ, 6-31  
グローバル化  
    SQL\*Loader, 5-16

## け

---

形式  
    SQL\*Loader 入力レコード, 5-40  
権限  
    EXEMPT ACCESS POLICY  
        ダイレクト・パス・エクスポートの影響, 1-53  
    SQL\*Loader に必要, 9-2  
    インポート, 2-7, 2-24  
    インポートに必要, 2-6  
    エクスポート, 1-24  
    エクスポートに必要, 1-5

## こ

---

更新  
    表の中の行  
        SQL\*Loader, 5-33  
後続の空白  
    切捨て, 6-47  
    デリミタを使用したロード, 6-27

構文図  
    BNF 変形で使用される記号, C-1  
    SQL\*Loader, A-1  
固定形式レコード, 3-4  
固定長レコード  
    外部表, 12-4  
固定レコード長  
    例, 10-34  
コマンドライン・パラメータ  
    SQL\*Loader, 4-2  
    SQL\*Loader の制御ファイルでの指定, 5-4  
    インポート・ユーティリティ, 2-15  
    エクスポート・ユーティリティ, 1-14  
コメント  
    SQL\*Loader の制御ファイル, 10-13  
    インポート・パラメータ・ファイル, 2-12  
    エクスポート・パラメータ・ファイル, 1-7  
    外部表, 12-2  
コレクション, 3-12  
    ロード, 7-29  
コンストラクタ  
    属性値, 7-8  
        上書き, 7-8  
    ユーザー定義, 7-8  
        列オブジェクトのロード, 7-8

## さ

---

再開可能な領域割当て  
    有効化および無効化, 1-27, 2-27, 4-11  
最適化  
    SQL\*Loader 入力ファイル処理, 5-11  
    ダイレクト・パス・ロード, 9-17  
索引  
    SQL\*Loader, 5-37  
        一意, 2-25  
        インポート, 2-25  
        エクスポート, 1-24  
        記憶域要件の計算, 9-12  
    索引作成コマンド  
        インポート・ユーティリティ, 2-25  
削除  
    SQL\*Loader, 9-24  
手動での作成, 2-25  
使用禁止状態, 5-25, 9-18  
使用禁止の場合のスキップ, 2-28, 4-14, 9-24

- ダイレクト・パス・ロード
  - ダイレクト・ロードの状態, 9-12
- データの事前ソート
  - SQL\*Loader, 9-17
- 複数列
  - SQL\*Loader, 9-18
- メンテナンスのスキップ, 4-14, 9-24
- ロード中断後の状態, 5-25
- 索引オプション
  - SQL\*Loader SINGLEROW パラメータ, 5-38
  - SQL\*Loader での SORTED INDEXES, 5-37
- 索引使用禁止状態
  - 索引使用禁止状態のままの索引, 5-25, 9-12
- 索引メンテナンスのスキップ, 4-14, 9-24
- 削除されたスナップショット
  - インポート・ユーティリティ, 2-66
- 作成
  - 表
    - 手動, 2-9
- サブパーティション表
  - ロード, 9-6
- 参照整合性制約
  - SQL\*Loader, 9-24
  - インポート時の使用禁止, 2-9

## し

---

- 時間
  - SQL\*Loader データ型, 6-15
- システム・オブジェクト
  - インポート, 2-8
- システム固有のデータ型
  - 長さ指定との競合
    - SQL\*Loader, 6-21
- システム・トリガー
  - インポート時の影響, 2-10
  - テスト, 2-10
- 事前ソート
  - ダイレクト・パス・ロード時のデータ
    - 例, 10-25
- 事前に決められたサイズ LOB, 7-25
- 事前に決められたサイズのフィールド
  - SQL\*Loader, 6-44
- 実 REF 列, 7-15
- シノニム
  - エクスポート, 1-60
  - ダイレクト・パス・ロード, 9-10

- 終端を指定されたフィールド
  - デリミタおよび SQL\*Loader による指定, 6-24
  - デリミタによる指定, 6-44
- 従来型パス・エクスポート
  - ダイレクト・パスとの比較, 1-51
- 従来型パスによる同時ロード, 9-29
- 従来型パス・ロード
  - SQL\*Loader バインド配列, 5-44
  - 使用する場合, 9-4
  - ダイレクト・パス・ロードとの比較, 9-9
  - 単一パーティション, 9-4
  - 中断された場合の動作, 5-24
  - 同時, 9-30
- 終了コード
  - SQL\*Loader, 4-16
  - インポート・ユーティリティ, 2-48
  - エクスポート・ユーティリティ, 1-50
- 主キー OID
  - 例, 7-12, 10-43
- 主キー REF 列, 7-16
- 主キー制約
  - ダイレクト・パス・ロードの影響, 9-34
- 順序番号
  - SQL\*Loader の SEQUENCE 句による生成, 6-54, 10-12
  - SQL\*Loader を使用した列への一意の番号の設定, 6-54
  - エクスポート, 1-57
  - キャッシュ, 1-57
  - 複数表へのロードおよび SQL\*Loader, 6-56
  - 読み込まれず生成された順序番号、SQL\*Loader, 6-5
- 使用禁止索引のスキップ, 4-14, 9-24
- 書式エラー
  - SQL\*Loader, 5-11
- 事例
  - SQL\*Loader, 10-1
  - SQL\*Loader ファイル名, 10-4
  - 「SQL\*Loader」を参照
- 新機能, xxxix
- シングルスバイト・キャラクタ・セット
  - インポート・ユーティリティ, 2-58

## す

---

- 数値型 EXTERNAL データ型
  - SQL\*Loader, 6-19



- デリミタ付き形式および SQL\*Loader, 6-24
- 長さの決定, 6-27
- スキーマ
  - エクスポートのための指定, 1-29
- スクリプト・ファイル
  - インポート前の実行, 2-5
  - エクスポート前の実行, 1-4
- ストアド・パッケージ
  - インポート, 2-62
  - COMPILE パラメータの影響, 2-62
- ストアド・ファンクション
  - インポート, 2-62
  - COMPILE パラメータの影響, 2-62
- ストアド・プロシージャ
  - インポート, 2-62
  - COMPILE パラメータの影響, 2-62
- ダイレクト・パス・ロード, 9-28
- ストリーム・バッファ
  - ダイレクト・パスのサイズ指定, 9-21
- ストリーム・レコード形式, 3-6
  - 列オブジェクトのロード, 7-2
- スナップショット, 2-66
  - インポート, 2-65
  - 削除された場合のリストア
    - インポート・ユーティリティ, 2-66
  - マスター表
    - インポート・ユーティリティ, 2-66
- スナップショット・ログ
  - インポート・ユーティリティ, 2-66

## せ

---

- 制御ファイル
  - SQL\*Loader 廃棄ファイルの指定, 5-14
  - キャラクタ・セット, 5-21
  - 作成
    - ガイドライン, 3-3
  - データ定義言語の構文, 5-2
  - データの指定, 5-10
- 制限
  - DB2 ロード・ユーティリティ, B-3
  - インポート・パラメータ・ファイル内の表名, 2-32
  - エクスポート・パラメータ・ファイル内の表名,  
1-30
  - 他のユーザーのスキーマへのインポート, 2-7
- 整合性制約
  - インポート時の違反, 2-49

- ダイレクト・パス・ロード時の使用可能, 9-25
- ダイレクト・パス・ロード時の使用禁止, 9-25
- ロード方法, 9-10
- 制約
  - 一意制約によるインポート・エラーの防止, 2-20
  - 違反
    - インポート・ユーティリティ, 2-49
  - 参照制約の使用禁止, 2-9
  - 自動整合性および SQL\*Loader, 9-27
  - 使用可能化
    - パラレル・ダイレクト・パス・ロード後, 9-33
  - ダイレクト・パス・ロード, 9-24
  - ダイレクト・ロードに対する施行, 9-25
  - ロード方法, 9-10
- 整理統合
  - エクステンツ, 1-18
- セカンダリ・データ・ファイル, 3-8, 7-31
- セキュリティに関する考慮点
  - ダイレクト・パス・エクスポート, 1-53
- セグメント
  - 一時
    - SQL\*Loader の FILE パラメータ, 9-32
- 接続文字列
  - Oracle Net, 1-54
- 前提条件
  - SQL\*Loader, 9-2
- 全データベース・モード
  - FULL での指定, 1-24
  - インポート・ユーティリティ, 2-23
  - エクスポート・ユーティリティ, 1-9
- 先頭の空白
  - 切捨ておよび SQL\*Loader, 6-46
  - 定義, 6-43

## そ

---

- 相対的なフィールド位置指定
  - フィールドの開始位置および SQL\*Loader, 6-45
  - 複数の SQL\*Loader の INTO TABLE 句, 5-39
- 挿入エラー
  - インポート・ユーティリティ, 2-49
  - 指定, 4-7
- ソート
  - SORTED INDEXES 句
    - SQL\*Loader, 9-18
  - 最適なソート順序
    - SQL\*Loader, 9-19

- ダイレクト・パス・ロード時の事前ソート, 9-17
- 複数列索引
  - SQL\*Loader, 9-18
- 属性
  - NULL, 7-6
- 属性値コンストラクタ, 7-8
  - 上書き, 7-8
- そのまま残す
  - 空白, 6-47

## た

---

- ダイレクト・パス・エクスポート, 1-51, 1-53
  - EXEMPT ACCESS POLICY 権限の影響, 1-53
  - セキュリティに関する考慮点, 1-53
  - ダイレクト・パスとの比較, 1-51
  - パフォーマンスの問題, 1-54
- ダイレクト・パス・ロード
  - DIRECT コマンドライン・パラメータ
    - SQL\*Loader, 9-11
  - ROWS コマンドライン・パラメータ, 9-13
  - SQL\*Loader でのデータ・ロード方法, 3-12
  - アーカイブ使用禁止の影響, 9-19
  - 一意制約の影響, 9-34
  - 一時セグメント記憶域要件, 9-12
  - インスタンス・リカバリ, 9-14
  - 記憶域の事前割当て, 9-17
  - 索引, 9-11
  - 索引の削除, 9-24
  - 指定, 9-11
  - シノニムへのロード, 9-10
  - 従来型パスによるロードとの比較, 9-9
  - 主キー制約の影響, 9-34
  - 使用, 9-9, 9-10
  - 使用の条件, 9-8
  - セグメント内同時処理, 9-30
  - セットアップ, 9-10
  - ソート順序の選択
    - SQL\*Loader, 9-19
  - 中断された場合の動作, 5-24
  - データ・セーブ, 9-13, 9-19
  - データの事前ソート, 9-17
  - データ変換の位置, 9-6
  - 同時, 9-30
  - トリガー, 9-24
  - バージョンの要件, 9-9

- パーティション・ロード
  - SQL\*Loader, 9-29
- パフォーマンス, 9-11, 9-17
- 表挿入トリガー, 9-26
- フィールド・デフォルト, 9-10
- 複数 CPU システムの最適化, 9-23
- 不適切なソート
  - SQL\*Loader, 9-18
- メディア・リカバリ, 9-15
- 読み込む行数の指定, 4-12
- リカバリ, 9-14
- 利点, 9-8
  - 例, 10-25
- ダイレクト・パス・ロード時の一時記憶, 9-12
- 対話方式
  - インポート・ユーティリティ, 2-45
  - エクスポート・ユーティリティ, 1-44
- タブ
  - 切捨て, 6-41
  - 空白, 6-41
  - タブを含むデータ・ファイルのロード, 6-4
- 単一表へのロード
  - 継続, 5-26
- ダンプ・ファイル
  - 最大サイズ, 1-22
- 断片化
  - 解消, 2-55

## ち

---

- 致命的エラー
  - 「リカバリ不能エラー」を参照
- 中断されたロード, 5-23
  - 継続, 5-26
  - 従来型パスの動作, 5-24
  - ダイレクト・パスの動作, 5-24

## て

---

- ディレクトリ別名
  - エクスポート, 1-58
  - インポート, 2-62
- データ
  - SQL\*Loader 制御ファイルへのロード・データの組込み, 6-52
  - SQL\*Loader のデリミタ付きデータの最大長, 6-27

- SQL\*Loader のパフォーマンスのために最適化された値, 6-52
- SQL\*Loader への異なる入力形式の区別, 5-38
- SQL\*Loader を使用した一意の値の生成, 6-54
- SQL\*Loader を使用したオペレーティング・システム間の移動, 6-35
- エクスポート, 1-28
- 行の保存
  - SQL\*Loader, 9-19
- 異なる入力行オブジェクトのサブタイプの区別, 5-38, 5-41
- 書式化されたデータと SQL\*Loader, 10-29
- 制御ファイルへの組み込み, 5-10
- セクションごとのロード
  - SQL\*Loader, 9-16
- ダイレクト・パス・ロード時の保存, 9-13
- デリミタで区切られたデータおよび SQL\*Loader, 6-26
- 複数表へのロード
  - SQL\*Loader, 5-38
- 未ソート
  - SQL\*Loader, 9-18
- データ型
  - BFILE
    - インポート・ユーティリティ, 2-62
    - エクスポート・ユーティリティ, 1-58
  - BYTEINT, 6-10
  - CHAR, 6-15
  - DATE, 6-16
  - DATE 長の決定, 6-28
  - DECIMAL, 6-11
  - DOUBLE, 6-10
  - FLOAT, 6-9
  - GRAPHIC, 6-18
  - GRAPHIC EXTERNAL, 6-18
  - INTEGER (n), 6-8
  - LENGTH-VALUE, 6-7
  - LONG
    - インポート・ユーティリティ, 2-63
    - エクスポート・ユーティリティ, 1-57
  - LONG VARRAW, 6-14
  - NUMBER
    - SQL\*Loader, 6-22, 6-23
  - RAW, 6-19
  - SMALLINT, 6-9
  - SQL\*Loader に対する文字フィールド長の設定, 6-27
  - SQL\*Loader のデフォルト, 6-7
  - SQL\*Loader の変換, 6-22
  - VALUE, 6-7
  - VARCHAR, 6-12
  - VARCHAR2
    - SQL\*Loader, 6-22
  - VARCHARC, 6-20
  - VARGRAPHIC, 6-11
  - VARRAW, 6-13
  - VARRAWC, 6-20
  - ZONED, 6-10
  - 移植可能, 6-14
  - 移植不能, 6-8
  - 外部表の識別, 12-19
  - 外部表フィールドの記述, 12-22
  - 期間, 6-15
  - システム固有
    - SQL\*Loader での長さ指定との競合, 6-21
  - 数値型 EXTERNAL, 6-19
  - データ・フィールドの SQL\*Loader データ型の指定, 6-7
  - 日時, 6-15
  - 非スカラー・データ型, 7-6
  - 文字データ型フィールドの競合, 6-27
- データが欠落しているショート・レコード
  - SQL\*Loader, 5-36
- データ・パス・ロード
  - ダイレクトおよび従来型, 9-2
- データ・ファイル
  - SQL\*Loader の形式の指定, 5-11
  - SQL\*Loader の指定, 5-7
  - SQL\*Loader のバッファの指定, 5-11
  - インポート時の再利用, 2-21
  - インポート中の上書き防止, 2-21
  - 指定, 4-5
- データ・フィールド
  - SQL\*Loader データ型の指定, 6-7
- データベース
  - エクスポート時の権限, 1-5
  - 既存のデータ・ファイルの再利用
    - インポート・ユーティリティ, 2-21
  - 全インポート, 2-23
  - 全体のエクスポート, 1-24
  - 断片化の解消, 2-55
  - データベース ID の変更, 14-3
  - 名前の変更, 14-4

- データベース ID (DBID)
  - 変更, 14-3
- データベース ID の変更, 14-3
- データベース移行のパーティション化, 1-62
  - インポート中の手順, 2-72
  - エクスポート中の手順, 1-63
  - デメリット, 1-62, 2-71, 2-72
  - メリット, 1-62, 2-71
- データベース・オブジェクト
  - LONG 列のエクスポート, 1-57
  - メタデータの抽出, 15-1
- データベース識別子
  - 変更, 14-3
- データベースの移行
  - パーティション化, 1-62
- データベース名 (DBNAME)
  - 変更, 14-4
- データベース名の変更, 14-4
- データ変換
  - ダイレクト・パス・ロード, 9-6
  - ダイレクト・パス・ロード時, 9-6
- データ・リカバリ
  - ダイレクト・パス・ロード
    - SQL\*Loader, 9-14
- データ列の欠落
  - SQL\*Loader, 5-36
- デフォルト列値
  - Oracle バージョン 6 のエクスポート・ファイル, 2-73
- デリミタ
  - SQL\*Loader の囲み, 6-44
  - SQL\*Loader の指定, 5-35, 6-24
  - SQL\*Loader のフィールド指定, 6-44
  - 外部表, 12-5
  - 外部表の指定, 12-15
  - 後続の空白のロード, 6-27
  - 最初と最後の例, 10-29
  - 終了, 6-44
  - データ中のマークおよび SQL\*Loader, 6-26
- デリミタ付きデータ
  - SQL\*Loader に対する最大長, 6-27
- デリミタ付きフィールド
  - フィールド長, 6-28
- デリミタ付きフィールドの LOB, 7-26

## と

---

### 統計

- アナライザ, 2-70
- インポート対象にする場合の指定, 2-29
- オブティマイザ, 2-70
- データベース・オブティマイザ
  - エクスポートのための指定, 1-28
- トランスポータブル表領域, 1-61, 2-67
- トリガー
  - 永続的に使用禁止, 9-29
- 更新
  - SQL\*Loader, 9-27
- システム
  - テスト, 2-10
- スキーマおよびトリガー
  - インポート時の影響, 2-10
- 整合性制約への置換, 9-27
- データベース挿入, 9-26

## な

---

### 内部 LOB

- ロード, 7-19

長さインジケータ

- サイズの決定, 5-46

## に

---

日時データ型, 6-15

入力文字の変換, 5-19

## ね

---

### ネストした表

- インポート, 2-61
- エクスポート, 1-59
  - 一貫性, 1-19

### ネストした列オブジェクト

- ロード, 7-4

### ネットワーク

- インポートおよび, 2-56
- エクスポート・ファイルの転送, 1-54

## は

---

### パーティション表

#### DB2

同等の Oracle 概念なし, B-4

インポート, 2-36, 2-51

エクスポート, 1-13

エクスポートの一貫性, 1-19

例, 10-34

ロード, 9-6

### パーティション・レベル・エクスポート

セッションの例, 1-41

### パーティション・レベル・インポート, 2-51

指定, 1-29

### パーティション・レベル・エクスポート, 1-13

### パーティション・ロード

SQL\*Loader, 9-29

従来型パスによる同時ロード, 9-29

### 廃棄された SQL\*Loader レコード, 3-9

原因, 5-15

上限, 5-16

廃棄ファイル, 5-14

### 廃棄ファイル

DB2 ロード・ユーティリティ, B-3

SQL\*Loader, 5-14

最大値の指定, 5-15

例, 10-15

### バイト順序, 6-36

SQL\*Loader の制御ファイルでの指定, 6-37

ビッグ・エンディアン, 6-36

リトル・エンディアン, 6-36

### バイト順序マーク, 6-38

確認の抑止, 6-39

優先順位

LOBFILE および SDF, 6-39

第 1 プライマリ・データ・ファイル, 6-38

### 配列

挿入後のコミット, 2-20

### 配列行の列

数の指定, 9-21

### バインド配列

SQL\*Loader に対するサイズの指定, 5-45

SQL\*Loader のメモリー所要量の最小化, 5-48

SQL\*Loader パフォーマンスとの関連, 5-44

行数の指定, 4-12

最大値の指定, 4-4

最低条件, 5-43

複数の SQL\*Loader の INTO TABLE 文のサイズ,  
5-49

バウンド FILLER, 6-6

パターン一致

インポート時の表名, 2-30

バックス正規形

「構文図」を参照

バックス正規形構文

「構文図」を参照

バックアップ

削除されたスナップショットのリストア

インポート・ユーティリティ, 2-66

バックスラッシュ・エスケープ文字, 5-6

バッファ

SQL\*Loader の BINDSIZE パラメータを使用した指  
定, 5-45

エクスポートのための計算, 1-18

必要な領域

SQL\*Loader の VARCHAR データ, 6-13

パフォーマンス

SQL\*Loader データ・ファイルの読み込みの最適化,  
5-11

インポート・ユーティリティ, 2-20

外部表使用時の問題, 11-6

整合性制約使用時の向上, 9-29

ダイレクト・パス・ロードの最適化, 9-17

パラメータ・ファイル

SQL\*Loader, 4-10

インポート・ユーティリティ, 2-26

コメント, 2-12

最大サイズ, 2-12

エクスポート・ユーティリティ, 1-25

コメント, 1-7

最大サイズ, 1-7

パラレル・ロード, 9-29

ダイレクト・パスの制限, 9-31

## ひ

---

比較文字列の埋込み

SQL\*Loader, 6-31

非スカラー・データ型, 7-6

ビッグ・エンディアン・データ

外部表, 12-6

日付キャッシュ

DATE\_CACHE パラメータ, 4-5

外部表, 11-6

日付キャッシュ機能

SQL\*Loader, 9-21

表

DB2 のパーティション

同等の Oracle 概念なし, B-4

SQL\*Loader を使用した既存の行の更新, 5-32

SQL\*Loader を使用した行の置換え, 5-32

SQL\*Loader を使用した行の挿入, 5-32

SQL\*Loader を使用した行の追加, 5-32

SQL\*Loader を使用した複数表へのデータのロード,  
5-38

アドバンスト・キューイング

インポート, 2-63

エクスポート, 1-59

インポート, 2-30

インポート前の定義, 2-9

エクスポート中の一貫性の維持, 1-19

エクスポートのための指定, 1-29

オブジェクト

インポートの順序, 2-4

オブジェクト表のロード, 7-12

外部, 11-1

切捨て

SQL\*Loader, 5-33

個々の表に対する SQL\*Loader の方法, 5-31

手動によるインポートの順序付け, 2-10

挿入トリガー

SQL\*Loader でのダイレクト・パス・ロード,  
9-26

ダイレクト・パス・ロード中の排他的アクセス

SQL\*Loader, 9-29

定義

インポート実行前の作成, 2-9

名前の制限

インポート・ユーティリティ, 2-30, 2-32

エクスポート・ユーティリティ, 1-30

ネストした

インポート, 2-61

エクスポート, 1-59

パーティション, 1-13

表モード・エクスポートの指定, 1-29

マスター表

インポート・ユーティリティ, 2-66

「外部表」を参照

表への INSERT

SQL\*Loader, 5-32

表への追加

SQL\*Loader, 5-32

例, 10-12

表名

大文字と小文字の区別の保存, 1-30

表モード・エクスポート, 1-9

指定, 1-29

表モードのインポート

例, 2-36

表領域

あるデータベースから別のデータベースへの移動,  
2-67

一連のエクスポート, 1-61

インポート中の削除, 2-69

再編成

インポート・ユーティリティ, 2-70

メタデータ

トランスポート, 2-34

読み込み専用

インポート・ユーティリティ, 2-69

表領域モード・エクスポート, 1-9

表レベル・インポート, 2-51

表レベル・エクスポート, 1-13

## ふ

ファイル名

SQL\*Loader, 5-5

SQL\*Loader 不良ファイル, 5-11

引用符, 5-5

複数の SQL\*Loader の指定, 5-9

ファイングレイン・アクセスのサポート

インポート・ユーティリティ, 2-58

エクスポート・ユーティリティ, 1-60

フィールド

SQL\*Loader の指定, 6-5

SQL\*Loader へのデフォルトのデリミタの指定,  
5-35

SQL\*Loader を使用した文字列との比較, 6-31

囲みおよび SQL\*Loader, 6-24

囲みデリミタおよび SQL\*Loader による指定, 6-24

事前のサイズ決定

SQL\*Loader, 6-44

長さ, 6-27

終了および SQL\*Loader, 6-24

終了デリミタおよび SQL\*Loader での指定, 6-24

すべての空白のロード, 6-40

相対的な位置指定および SQL\*Loader, 6-45

デリミタ付きフィールド

SQL\*Loader, 6-24

長さの決定, 6-28

デリミタ付きの SQL\*Loader

指定, 6-44

文字データ長および SQL\*Loader, 6-27

フィールド位置

SQL\*Loader, 6-3

FIELDS 句

SQL\*Loader, 5-35

空白での終了, 6-46

フィールド条件

SQL\*Loader の指定, 6-29

フィールド長

SQL\*Loader の指定, 6-44

複数 CPU システム

ダイレクト・パス・ロードの最適化, 9-23

複数表へのロード

SQL\*Loader 制御ファイルの指定, 5-38

SQL\*Loader を使用した一意の順序番号の生成,  
6-56

複数列索引

SQL\*Loader, 9-18

負数

ロード, 10-15

不良ファイル

SQL\*Loader の指定, 5-11

## へ

---

別名

ディレクトリ

インポート, 2-62

エクスポート, 1-58

## ま

---

マスター表

スナップショット

インポート・ユーティリティ, 2-66

マテリアライズド・ビュー, 2-65

マルチスレッド

複数 CPU システム, 9-23

マルチバイト・キャラクタ・セット

SQL\*Loader, 5-17

SQL\*Loader での空白, 6-31

## み

---

未ソート・データ

ダイレクト・パス・ロード

SQL\*Loader, 9-18

## む

---

無効なオブジェクト

インポート中の警告メッセージ, 2-47

無効なデータ

インポート・ユーティリティ, 2-49

## め

---

メタデータ API, 15-2

実装, 15-4

パフォーマンスの向上, 15-10

ブラウザ・インタフェース, 15-10

プログラム・インタフェース, 15-5

プログラム例, 15-11

メッセージ

インポート・ユーティリティ

完了, 2-48

警告, 2-47

リカバリ不能, 2-47

エクスポート・ユーティリティ

完了, 1-50

警告, 1-49

リカバリ不能, 1-49

メディア・リカバリ

ダイレクト・パス・ロード, 9-15

## も

---

文字長セマンティクス, 5-22

文字データ型

フィールドの競合, 6-27

文字フィールド

SQL\*Loader に対する長さの指定, 6-27

SQL\*Loader のデータ型, 6-14

デリミタと SQL\*Loader, 6-14, 6-24

文字列

SQL\*Loader, 6-31

外部表

バイトまたは文字の指定, 12-7

文字列の比較  
SQL\*Loader, 6-31

## ゆ

---

ユーザー定義  
インポート, 2-73  
ユーザー定義のコンストラクタ, 7-8  
列オブジェクトのロード, 7-8  
ユーザー・モード・エクスポート, 1-9  
指定, 1-25

## よ

---

読込み一貫性のあるエクスポート, 1-19  
読込み専用データベース  
エクスポート, 1-62  
読込み専用表領域  
インポート・ユーティリティ, 2-69

## ら

---

ライブラリ  
外部関数  
インポート, 2-62, 2-63  
エクスポート, 1-58

## り

---

リカバリ  
行の置換え, 5-32  
ダイレクト・パス・ロード  
SQL\*Loader, 9-14  
リカバリ可能なエラー  
インポートでの警告のフラグの付与, 2-47  
エクスポートでの警告のフラグの付与, 1-49  
リカバリ不能エラー  
インポート・ユーティリティ, 2-47  
エクスポート・ユーティリティ, 1-49  
リソース・エラー  
インポート・ユーティリティ, 2-51  
リトル・エンディアン・データ  
外部表, 12-6  
リフレッシュ・エラー  
スナップショット  
インポート・ユーティリティ, 2-66

リモート操作  
エクスポートおよびインポート, 1-54, 2-56

## れ

---

レコード  
1つの論理レコードの作成  
SQL\*Loader, 5-26  
DISCARDMAX コマンドライン・パラメータ, 4-6  
SQL\*Loader による拒否, 3-9, 3-10, 5-11  
SQL\*Loader による廃棄, 3-9, 5-14  
SQL\*Loader の異なる形式の区別, 5-40  
SQL\*Loader を使用した複数の論理レコードの抽出,  
5-38  
SQL\*Loader を使用した列のレコード番号の設定,  
6-54  
インポート時の長さの指定, 2-26  
エクスポート時の長さの指定, 1-27  
固定形式, 3-4  
ストリーム・レコード形式, 3-6  
ロード中のデータ列の欠落, 5-36  
ロード方法の指定, 4-9

列  
LONG データ型のエクスポート, 1-57  
PIECED としての指定  
SQL\*Loader, 9-16  
REF 列のロード, 7-15  
SQL\*Loader の使用, 6-53  
SQL\*Loader を使用した NULL の設定, 6-53  
SQL\*Loader を使用した一意の順序番号の設定,  
6-54  
SQL\*Loader を使用した現在の日付の設定, 6-54  
SQL\*Loader を使用した式の値の設定, 6-53  
SQL\*Loader を使用した定数値の設定, 6-53  
SQL\*Loader を使用したデータ・ファイルのレコー  
ド番号の設定, 6-54  
インポート実行前の順序変更, 2-9  
オブジェクト  
可変レコード形式, 7-3  
ストリーム・レコード形式, 7-2  
ネストした列オブジェクトのロード, 7-4  
指定  
SQL\*Loader, 6-5  
命名  
SQL\*Loader, 6-5  
列オブジェクト  
ロード, 7-2



ユーザー定義コンストラクタ, 7-8

## ろ

### ロード

- LOB, 7-18
- REF 列, 7-15
- XML 列, 7-18
- オブジェクト表, 7-12
- 外部表データ
  - 条件の指定, 12-6, 12-11
  - レコードのスキップ, 12-9
- 可変長データ, 10-6
- 結合した物理レコード, 10-15
- 固定長データ, 10-9
- コレクション, 7-29
- サブタイプを使用したオブジェクト表, 7-14
- サブパーティション表, 9-6
- 自由区分形式ファイル, 10-12
- タブを含むデータ・ファイル
  - SQL\*Loader, 6-4
- ネストした列オブジェクト, 7-4
- 表, 9-6
- 負数, 10-15
- 列オブジェクト, 7-2
  - 可変レコード形式, 7-3
  - ユーザー定義コンストラクタ, 7-8
  - 導出サブタイプの使用, 7-5

### ロード時のデータの正規化

SQL\*Loader, 10-19

### ロール

- EXP\_FULL\_DATABASE, 1-5
- IMP\_FULL\_DATABASE, 2-5
- RESOURCE, 2-6

### ロールバック・セグメント

- CONSISTENT エクスポート・パラメータの影響, 1-19
- インポート中のサイズの制御, 2-20

### ログ・ファイル

- SQL\*Loader, 3-11
- SQL\*Loader グローバル情報, 8-2
- SQL\*Loader サマリー統計, 8-6
- SQL\*Loader データ・ファイル情報, 8-5
- SQL\*Loader の指定, 4-9
- SQL\*Loader 表情報, 8-3
- SQL\*Loader 表ロード情報, 8-5
- SQL\*Loader ヘッダー情報, 8-2

インポート・ユーティリティ, 2-26, 2-47

エクスポート・ユーティリティ, 1-25, 1-49

例, 10-27, 10-33

ロード中断後, 5-25

### 論理レコード

SQL\*Loader を使用した複数の物理レコードの統合, 5-26

