Oracle8*i*

ユーティリティ・ガイド

リリース 8.1

ORACLE

Oracle8*i* ユーティリティ・ガイド リリース 8.1

部品番号 : A62770-1

第 1 版 : 1999 年 5 月（第 1 刷）

原本名 : Oracle8*i* Utilities, Release 8.1.5

原本部品番号 : A67792-01

原本著者 : Jason Durbin

原本協力者 : Karleen Aghevli、Lee Barton、Allen Brumm、George Claborn、William Fisher、Paul Lane、Tracy Lee、Vishnu Narayana、Visar Nimani、Joan Pearson、Mike Sakayeda、James Stenois、Chao Wang、Gail Ymanaka、Hiro Yoshioka

グラフィック・デザイナ : Valarie Moore

目次

2 Import

第 II 部　SQL*Loader

3　SQL*Loader の概念

4 SQL*Loader の事例研究

5 SQL*Loader 制御ファイル・リファレンス

6 SQL*Loader コマンド行リファレンス

7 SQL*Loader: ログ・ファイル参照

8 SQL*Loader: 従来型パス・ロードとダイレクト・パス・ロード

第 III 部　オフライン・データベース検査ユーティリティ

9 オフライン・データベース検査ユーティリティ

A SQL*Loader の予約

B DB2/DXT ユーザーに対する注意事項

索引

はじめに

このマニュアルでは、Oracle8*i* ユーティリティを使用してデータ転送、メンテナンス、データベース管理を行う方法について説明します。

『Oracle8*i* ユーティリティ・ガイド』には、Oracle8*i* および Oracle8*i* Enterprise Edition 製品の機能および機能性に関する情報を収めてあります。Oracle8*i* および Oracle8*i* Enterprise Edition の基本的な機能は同じです。ただし、Enterprise Edition のみで使用できる高度な機能もあり、そのうちの一部はオプションです。

Oracle ユーティリティ

このマニュアルでは、前述の各ユーティリティに関して基本概念を説明し、ユーティリティの使用例を示します。

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するほか、次の作業を行う Oracle ユーザーを対象としています。

- Export/Import ユーティリティによる、データ・アーカイブ、Oracle データベースのバックアップ、Oracle データベース間のデータ移動
- SQL*Loader による、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
- ユーザー定義キャラクタ・セット（NLS ユーティリティ）およびその他の Oracle NLS データの作成、メンテナンス

このマニュアルを使用するにあたって、読者は『Oracle8*i* 概要』で説明されている SQL および Oracle の基礎的な実務知識を習得しておく必要があります。また、SQL*Loader を使用する際には、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あらかじめ理解しておく必要があります。

注意：このマニュアルには、ユーティリティのインストレーション手順の説明はありません。ユーティリティのインストール手順は、オペレーティング・システムによって異なります。ユーティリティのインストールに関しては、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。

このマニュアルの構成

このマニュアルは 3 部構成になっています。内容は次のとおりです。

第 I 部：Export および Import

第 1 章「Export」

Export 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。この章では、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行方式、パラメータの指定、エクスポート・オブジェクトのサポートについて解説します。エクスポート・セッションの例も示します。

第 2 章「Import」

Import 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。この章では、インポートの概要、対話方式とコマンド行方式、パラメータの指定、インポート・オブジェクトのサポートについて解説します。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」

SQL*Loader を紹介し、その機能について説明します。また、データのロードの概念（オブジェクト・サポートも含む）も紹介します。さらに、SQL*Loader への入力、データベースの事前準備、SQL*Loader からの出力についても説明します。

第 4 章「SQL*Loader の事例研究」

さまざまな事例から SQL*Loader の機能について説明します。可変長データ、固定形式レコード、自由形式ファイルの各ロード方法、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、複数の表へデータをロードする方法、ダイレクト・パスを使用したロード方法、オブジェクトのロード方法、コレクション、REF 列などについて解説します。

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 フォーマットにマップする方法を SQL*Loader に対して記述する方法について説明しています。詳細な構文図およびデータ・ファイル、表および列の、データの位置、ロードするデータの型およびフォーマット、その他の指定に関する情報もあります。

第 6 章「SQL*Loader コマンド行リファレンス」

SQL*Loader で使用するコマンド行の構文について解説します。コマンド行引数、SQL*Loader のメッセージを抑止する方法やバインド配列のサイズ指定などについて説明します。

第 7 章「SQL*Loader: ログ・ファイル参照」

ログ・ファイルに記述されている情報について説明します。

第 8 章「SQL*Loader: 従来型パス・ロードと ダイレクト・パス・ロード」

従来型パス・ロード方法とダイレクト・パス・ロード方法の違いについて説明します。ダイレクト・パス・ロードは、大量のデータを従来よりも高速でロードするための高パフォーマンス・オプションです。

第 III 部：オフライン・データベース検査ユーティリティ

第 9 章「オフライン・データベース 検査ユーティリティ」

オフライン・データベース検査ユーティリティ、DBVERIVY の使用方法について説明します。

付録 A「SQL*Loader の予約」

SQL*Loader によって予約されている語のリストを示します。

付録 B「DB2/DXT ユーザーに対する注意事項」

SQL*Loader と DB2 ロード・ユーティリティの制御ファイル用のデータ定義言語の構文の違いについて説明します。この付録では、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。

表記法

このマニュアルの表記は、次の項で説明する字体および記号の使用規則に基づいています。

マニュアル本文

このマニュアルのテキストには、次の表記規則を使用します。

大文字	大文字のテキストは、コマンド・キーワード、オブジェクト名、パラメータ、ファイル名、その他を示します。たとえば、次のように使用します。 「プライベート・ロールバック・セグメントを作成する場合、その名前を ROLLBACK_SEGMENTS パラメータとして PARAMETER ファイルに記録しておく必要があります。」
イタリック体	イタリック体は、SQL 文のパラメータを示します。

PL / SQL および SQL、SQL*Plus のコマンドや文は、下記の規則に基づいて、一定幅のフォントで示されます。また、通常のテキストとは、次の例のようにして区切られます。

```
ALTER TABLESPACE users   ADD DATAFILE 'users2.ora' SIZE 50K;
```

句読点： ,'"	例文の中には、カンマや引用符などの句読点が含まれていますが、これらの句読点はすべて必須です。また、例文の最後には必ずセミコロンがついています。使用するアプリケーションによって、文の最後にセミコロンまたは他の終了記号が必要な場合と不要な場合があります。
大文字 : INSERT, SIZE	例文中の大文字は、Oracle SQL のキーワードを示します ただし実際に文を記述する場合は、キーワードでは大文字と小文字は区別されません。
小文字：emp, users2.ora	例文中の小文字は、その事例のみで使用される語を表します。たとえば、表や列、ファイルの名前などに小文字を使用します。オペレーティング・システムによっては大文字と小文字を区別するものもありますので、使用しているインストレーションおよびユーザーズ・マニュアルを参照の上、大文字と小文字を区別する必要があるかどうかを確認してください。

第 I 部

エクスポートおよびインポート

1

Export

この章では、Export 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、データベースの外に格納したり、Import ユーティリティ（第 2 章「Import」において説明）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。この章では、次のトピックについて説明します。

- Export ユーティリティとは
- エクスポート・モード
- Export ユーティリティの使用方法
- 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
- ダイレクト・パス・エクスポート
- 増分、累積および全エクスポート
- ネットワークに関する考慮事項
- キャラクタ・セットおよび NLS に関する考慮事項
- 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事項
- トランスポータブル表領域
- バージョンの異なる Export ユーティリティの使用方法
- Oracle8*i* データベースからの Oracle リリース 7 のエクスポート・ファイルの作成

Export ユーティリティとは

Export 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。Export を実行すると、Oracle データベースからオブジェクト定義と表データが抽出され、通常ディスクまたはテープにあるバイナリ形式の Oracle エクスポート・ダンプ・ファイルに書き込まれます。

次に、このエクスポート・ファイルは別サイトにファイル転送、または物理的に移送（テープの場合）され、Import ユーティリティによって、ネットワーク接続していないマシン上のデータベース間でのデータ転送や、標準のバックアップ手順以外のバックアップとして使用されます。

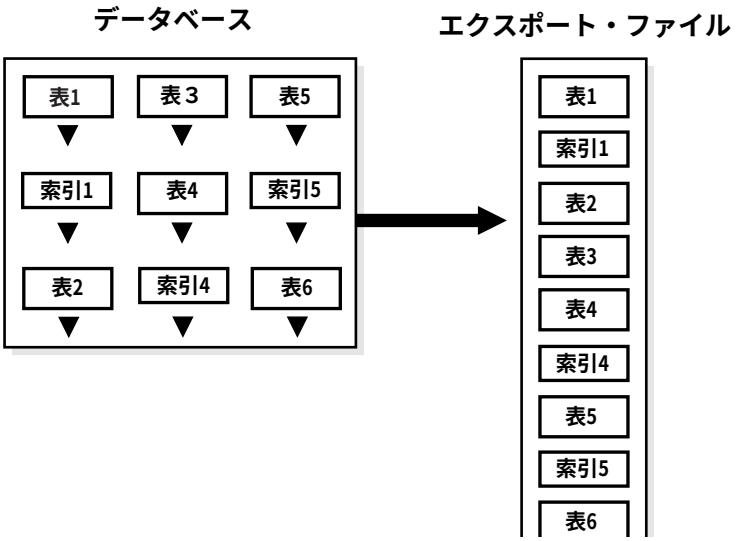
また、Export および Import ユーティリティは、オフライン・インスタンシエーションなど、Oracle アドバンスト・レプリケーションの機能面でも役立ちます。詳細は、『Oracle8*i* レプリケーション・ガイド』を参照してください。

エクスポート・ダンプ・ファイルが読み込めるのは、Oracle ユーティリティの Import を使用した場合のみです（第 2 章「Import」参照）。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場合は、このマニュアルの第 II 部「SQL*Loader」の情報を参照してください。

Oracle データベースに対して Export を実行すると、まずオブジェクト（表など）が抽出され、続いてそれに関連するオブジェクト（索引やコメント、権限など）が抽出されて、これらがエクスポート・ファイルに書き込まれます。図 1-1 を参照してください。

> **注意 :** 現在、アドバンスト・レプリケーション・オプションを使用している場合は、『Oracle8*i* レプリケーション・ガイド』の、移行と互換性に関する説明を参照してください。

図 1-1 データベースのエクスポート



新機能

Oracle8*i* の今回のリリースでは、次のエクスポート機能が追加されました。

- サブパーティションのエクスポート。詳細は、1-8 ページ「表レベル・エクスポートとパーティション・レベル・エクスポートの違い」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、1-19 ページ「FILE」および 1-19 ページ「FILESIZE」を参照してください。
- Export が表のアンロードに使用する SELECT 文に対して、問合せが指定可能。1-21 ページ「QUERY」を参照してください。
- 各テープ媒体上のエクスポート・ファイルに指定できる最大バイト数の増加。1-27 ページ「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（コマンド行でダイレクト・パスが指定されている場合を含みます）。1-55 ページ「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事項」を参照してください。
- エクスポートおよびインポートのオプティマイザ統計情報を、インポート時に再計算するかわりに、事前に計算可能。（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利用できます。）1-23 ページ「STATISTICS」を参照してください。

- ドメイン索引の開発者は、ODCIIndex インタフェース上で、新しい ODCIIndexGetMetadata 方式を使用することにより、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、『Oracle8*i* データ・カートリッジ開発者ガイド』を参照してください。
- プロシージャ・オブジェクトのエクスポート アドバンスト・キューおよびリソース・スケジューラ・オブジェクトなどのプロシージャ・オブジェクトを作成するデータ定義言語には、現在、SQL ではなく PL/SQL が使用されています。
- 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。1-26 ページ「TRANSPORT_TABLESPACE」を参照してください。

エクスポート・ファイルの内容の読込み

エクスポート・ファイルは Oracle バイナリ形式で格納されます。Export ユーティリティを使用して作成されたエクスポート・ファイルを読み込めるのは、Import ユーティリティのみです。Export 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、旧バージョンの Import ユーティリティでは読み込めません。ただし Import ユーティリティは、現行および前回リリースの Export ユーティリティで書き出されたファイルを読み込むことが可能で、その他のフォーマットについては読み込めません。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場合は、このマニュアルの第 II 部「SQL*Loader」の情報を参照してください。

ただし、第 2 章「Import」で説明する Import ユーティリティの SHOW パラメータを使用すると、エクスポート・ファイルの内容を表示できます。詳細は、2-28 ページ「SHOW」を参照してください。

アクセス権限

Export ユーティリティを使用するには、Oracle データベースに関する CREATE SESSION 権限が必要です。他のユーザーが所有してい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いてください。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と、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。シノニムを作成したとしても、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みであるため、Export で処理されません。

- ORDSYS
- MDSYS
- CTXSYS
- ORDPLUGINS

エクスポート・モード

Export ユーティリティには、エクスポートのモードが 4 種類用意されています。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、表モード、ユーザー・モード、トランスポータブル表領域モードおよび全データベース・モードのどれでも使用できます。エクスポートされるオブジェクトは、選択したモードによって異なります。トランスポータブル表領域モードは、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。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については、1-57 ページ「トランスポータブル表領域」および『Oracle8*i* 管理者ガイド』を参照してください。トランスポータブル表領域の機能の詳細は、『Oracle8*i* 概要』を参照してください。

各モードの指定の詳細は、1-14 ページ「エクスポート・パラメータ」を参照してください。

最初の 3 つのモードのどれでも、従来型パス・エクスポートまたはダイレクト・パス・エクスポートを使用できます。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、1-41 ページ「ダイレクト・パス・エクスポート」で説明します。

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。

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

表モード	ユーザー・モード	全データベース・モード	トランスポータブル表領域モード
TABLES リストの各表について、エクスポートおよびインポートできるオブジェクト	**所有者リストの各ユーザーについて、エクスポートおよびインポートできるオブジェクト**	**特権ユーザーがエクスポートおよびインポートできるすべてのデータベース・オブジェクト（SYS が所有するオブジェクト、および ORDSYS、CTXSYS、MDSYS および ORDPLUGINS スキーマを除く）**	**TABLESPACES リストの各表領域について、特権ユーザーがエクスポートおよびインポートできる、次のオブジェクトに対する DDL**
表の事前プロシージャ処理	外部関数ライブラリ	表領域定義	クラスタ定義
表で使用されるオブジェクト型定義	オブジェクト型	プロファイル	
表定義	データベース・リンク	ユーザー定義	**現行の表領域内の各表について、次のオブジェクトの DDL が含まれる。**
表の事前処理	順序番号	ロール	
パーティションごとの表データ	クラスタ定義	システム権限	表の事前プロシージャ処理

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）

表モード	ユーザー・モード	全データベース・モード	トランスポータブル表領域モード
ネストした表データ	**その他、特定のユーザーが所有する各表について、エクスポートおよびインポートできるオブジェクト**	ロール権限 デフォルト・ロール 表領域割当て制限	表で使用される オブジェクト型定義
表所有者権限 所有者表索引 表制約（主キー制約、一意制約、チェック制約）	表の事前プロシージャ処理	リソース・コスト	表定義（表の行は除く）
分析表	表で使用される オブジェクト型定義	ロールバック・セグメント定義	表の事前処理
列コメントおよび 表コメント	表定義	データベース・リンク	表権限
監査情報	表の事前処理	順序番号	表索引
表のセキュリティ・ ポリシー	パーティションごとの表データ	すべてのディレクトリ別名	表制約（主キー制約、一意制約、チェック制約）
表参照制約	ネストした表データ	アプリケーション ・コンテキスト	列コメントおよび 表コメント
所有者表トリガー	表所有者権限	すべての外部関数 ライブラリ	参照整合性制約
表の事後処理	所有者表索引 **(1)**	すべてのオブジェクト型 すべてのクラスタ定義	ビットマップ索引 （**注意：**ファンクションまたはドメイン索引以外）
表の事後処理プロシージャおよびオブジェクト	表制約（主キー制約、一意制約、チェック制約）	デフォルト監査およびシステム監査	表の事後処理
	分析表		トリガー
上記の他に、特権ユーザーがエクスポートおよびインポートできるオブジェクト	列コメントおよび 表コメント	**各表について、特権ユーザーがエクスポートおよびインポートできるオブジェクト**	表の事後処理プロシージャおよびオブジェクト
他のユーザーが 所有するトリガー	監査情報	表の事前プロシージャ処理	
他のユーザーが 所有する索引	表のセキュリティ・ ポリシー	表で使用されるオブジェクト型定義	
	表参照制約	表定義	
	プライベート・シノニム	表の事前処理	
	ユーザー・ビュー	パーティションごとの 表データ	

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）

表モード	ユーザー・モード	全データベース・モード	トランスポータブル表領域モード
	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、ユーザー・ストアド・ファンクション	ネストした表データ	
	参照整合性制約	表権限	
	演算子	表索引	
	トリガー（2）	表制約（主キー制約、一意制約、チェック制約）	
	表の事後処理	分析表	
	索引タイプ	列コメントおよび表コメント	
	スナップショットおよびマテリアライズド・ビュー	監査情報	
	スナップショット・ログ	すべての参照整合性制約	
	ジョブ・キュー	すべてのシノニム	
	リフレッシュ・グループ	すべてのビュー	
	ディメンション	すべてのストアド・プロシージャ、ストアド・パッケージ、ストアド・ファンクション	
	プロシージャ・オブジェクト	表の事後処理	
	表の事後処理プロシージャおよびオブジェクト	演算子	
	スキーマの事後処理プロシージャおよびオブジェクト	索引タイプ	
		表の事後処理	
		すべてのトリガー	
		分析クラスタ	
		すべてのスナップショットおよびマテリアライズド・ビュー	
		すべてのスナップショット・ログ	
		すべてのジョブ・キュー	

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）

表モード	ユーザー・モード	全データベース・モード	トランスポータブル表領域モード
		すべてのリフレッシュ・グループおよび子	
		ディメンション	
		パスワード履歴	
		システム監査	
		表の事後処理プロシージャおよびオブジェクト	
		スキーマの事後処理プロシージャおよびオブジェクト	

1. 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限り、その索引はエクスポートされません。
2. 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であっても）、エクスポートおよびインポートできます。

表レベル・エクスポートとパーティション・レベル・エクスポートの違い

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、索引その他の表の依存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全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。（ダイレクト・パス・エクスポートでも従来型パス・エクスポートでも、この点は同じです。）表レベル・エクスポートは、すべてのエクスポート・モード（全、ユーザー、表、トランスポータブル表領域）でサポートされています。

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。全データベース・モード、ユーザー・モードおよびトランスポータブル表領域モードでのエクスポートでは、パーティション・レベル・エクスポートは実行できません。パーティション・レベル・エクスポートを実行できるのは、表モードのエクスポートのみです。増分エクスポート（増分、累積、全）は全データベース・モードでしか実行できません。したがって、増分エクスポートではパーティション・レベル・エクスポートを指定できません。

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。

パーティション・レベル・エクスポートの指定方法は、1-24 ページ「TABLES」を参照してください。

Export ユーティリティの使用方法

この項では、エクスポートの事前準備や Export ユーティリティの起動方法など Export ユーティリティの使用方法について説明します。

Export ユーティリティを使用する前に

Export 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 CATALOG.SQL（CATEXP.SQL を実行します）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・ファイルの名前およびそれらを実行する方法については、ご使用のオペレーティング・システム固有の Oracle ドキュメント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。エクスポートの実行前にこれらのスクリプトを再度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。

- 必要なエクスポート・ビューを作成する。
- EXP_FULL_DATABASE ロールに、すべての必要な権限を割り当てる。
- EXP_FULL_DATABASE を DBA ロールに割り当てる。

Export ユーティリティを実行する前に、エクスポート・ファイルの書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。十分な領域がないと、書込み失敗というエラーで Export ユーティリティ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 ビューで見ることができます。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のデータで使用されているディスク領域は含まれません。

ディクショナリ・ビューの詳細は、『Oracle8*i* リファレンス・マニュアル』を参照してください。

Export ユーティリティの起動

Export ユーティリティは、次の方法で起動できます。

- 次のコマンドを入力します。

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

  PARFILE は、通常使用するエクスポート・パラメータが格納されているファイルです。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。

- 次のコマンドを入力します。

  ```
  exp username/password
  ```

  このコマンドの後に、使用する各種パラメータを指定します。

  注意 : システムのコマンド行の最大長を超える数のパラメータは指定できません。

- 対話型セッションを開始するにはコマンド「`exp` ユーザー名 / パスワード」のみを入力します。そうすると Export ユーティリティによって必要な情報を入力するよう要求されます。対話方式は、パラメータ指定による方式よりも機能面で劣ります。対話方式は、下位互換性のために用意されています。

最初のオプションと 2 番目のオプションは組み合せて使用できます。つまり、パラメータ・ファイルとコマンド行の両方にパラメータを指定できます。実際には、パラメータ・ファイルとコマンド行の両方に同じパラメータを指定することもできます。コマンド行での PARFILE パラメータと他のパラメータの位置によって、どのパラメータが優先されるかが決まります。たとえば、パラメータ・ファイル `params.dat` に、パラメータ INDEXES=Y が指定されていると、Export ユーティリティは次のコマンド行によって起動されます。

```
exp system/manager PARFILE=params.dat INDEXES=N
```

この場合、`INDEXES=N` は `PARFILE=params.dat` の後にあるので、PARFILE に指定されている INDEXES パラメータは `INDEXES=N` によって上書きされます。

ユーザー名とパスワードは、パラメータ・ファイルでも指定できますが、セキュリティ上の理由のため、この方法は使用しないでください。「ユーザー名 / パスワード」を指定しないと、入力するよう要求されます。

詳細は、1-14 ページ「エクスポート・パラメータ」を参照してください。

デフォルトのデータベース以外のデータベースからのエクスポートの指定方法は、1-52 ページ「Net8 を利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 としての Export ユーティリティの起動

通常は、Export を SYSDBA として起動する必要はありません。ただし、オラクル社カスタマ・サポートの要求があれば、SYSDBA で起動する場合もあります。Export を SYSDBA として起動するには、次の構文を使用します。

```
exp username/password AS SYSDBA
```

または、任意で次の構文を使用します。

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

> **注意 :** 文字列 "AS SYSDBA" にはブランクが含まれるため、ほとんどのオペレーティング・システムでは、'username/password AS SYSDBA' のように文字列全体を引用符で囲むか、なんらかの方法でリテラルとしてマーク設定することが必要です。オペレーティング・システムによっては、コマンド行に含まれる引用符も同様に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。ユーザー名またはパスワードを指定しないと、入力するよう要求されます。

Export ユーティリティの対話形式モードを使用する場合の詳細は、1-36 ページ「SYSDBA としての対話形式による Export ユーティリティの起動」を参照してください。

オンライン・ヘルプの利用

Export ユーティリティには、オンライン・ヘルプ画面が用意されています。コマンド行に `exp help=y` と入力すると、次のようなヘルプ画面が表示されます。

```
> exp help=y

Export: Release 8.1.5.0.0 - Production on Wed Oct 28 15:00:10 1998

(c) Copyright 1998 Oracle Corporation.  All rights reserved.



You can let Export prompt you for parameters by entering the EXP
command followed by your username/password:

     Example: EXP SCOTT/TIGER

Or, you can control how Export runs by entering the EXP command followed
by various arguments. To specify parameters, you use keywords:

     Format:  EXP KEYWORD=value or KEYWORD=(value1,value2,...,valueN)
     Example: EXP SCOTT/TIGER GRANTS=Y TABLES=(EMP,DEPT,MGR)
              or TABLES=(T1:P1,T1:P2), if T1 is partitioned table
```

```
USERID must be the first parameter on the command line.

Keyword  Description (Default)        Keyword      Description (Default)
--------------------------------------------------------------------------
USERID   username/password            FULL         export entire file (N)
BUFFER   size of data buffer          OWNER        list of owner usernames
FILE     output files (EXPDAT.DMP)    TABLES       list of table names
COMPRESS import into one extent (Y)   RECORDLENGTH length of IO record
GRANTS   export grants (Y)            INCTYPE      incremental export type
INDEXES  export indexes (Y)           RECORD       track incr. export (Y)
ROWS     export data rows (Y)         PARFILE      parameter filename
CONSTRAINTS export constraints (Y)    CONSISTENT   cross-table consistency
LOG      log file of screen output    STATISTICS   analyze objects (ESTIMATE)
DIRECT   direct path (N)              TRIGGERS     export triggers (Y)
FEEDBACK display progress every x rows (0)
FILESIZE maximum size of each dump file
QUERY    select clause used to export a subset of a table
VOLSIZE  number of bytes to write to each tape volume

The following keywords only apply to transportable tablespaces
TRANSPORT_TABLESPACE export transportable tablespace metadata (N)
TABLESPACES list of tablespaces to transport

Export terminated successfully without warnings.
```

コマンド行またはパラメータ・ファイルでのエクスポート・パラメータ指定

エクスポート・パラメータは、次の 3 つの方法で指定できます。コマンド行から入力する方法、Export ユーティリティにパラメータ値のプロンプトを表示させる方法、またはパラメータ・ファイル内で指定する方法です。

コマンド行パラメータの入力

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行から指定できます。

```
exp KEYWORD=value
```

または

```
exp KEYWORD=(value1,value2,...,valuen)
```

エクスポート・パラメータのプロンプト

Export ユーティリティによって、各パラメータ値に対するプロンプトを表示させるには、次の構文を使用します。

```
exp username/password
```

Export ユーティリティは、値の入力を要求するとともに各パラメータを表示します。

パラメータ・ファイル

パラメータ・ファイルの中にエクスポート・パラメータを指定しておくと、パラメータを容易に変更および再利用できます。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・ファイルを作成してください。コマンド行オプション PARFILE=*filename* は、コマンド行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むように Export ユーティリティに通知します。たとえば、次のように表示されます。

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

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。

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

パラメータ・ファイル内のリストの一部の例を次に示します。

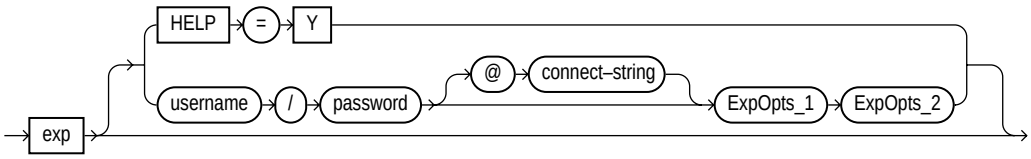
```
FULL=Y
FILE=DBA.DMP
GRANTS=Y
INDEXES=Y
CONSISTENT=Y
```

> **追加情報 :** 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・システムの命名規則に従います。詳細は、ご使用のオペレーティング・システム固有の Oracle ドキュメントを参照してください。

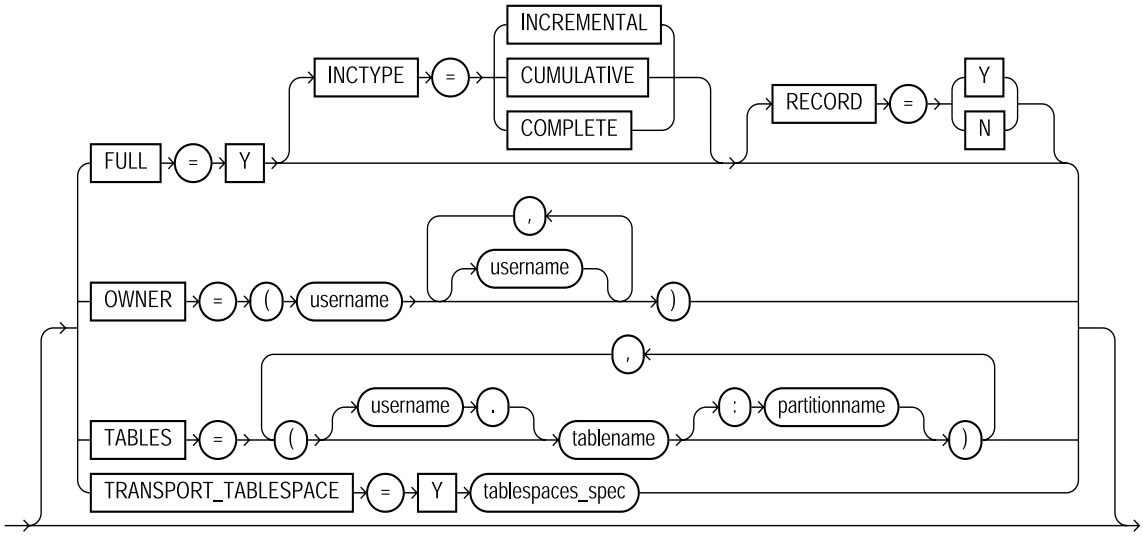
シャープ（#）記号を使用して、パラメータ・ファイルにコメントを追加できます。シャープ（#）の右側にある文字はすべて無視されます。

エクスポート・パラメータ

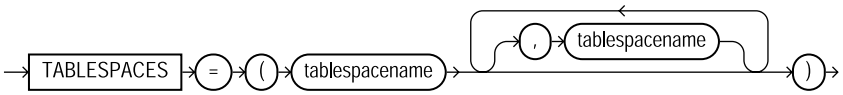
パラメータ・ファイルまたはコマンド行で指定できるパラメータの構文を次の 3 つのダイアグラムに示します。その後に、各パラメータについて説明します。



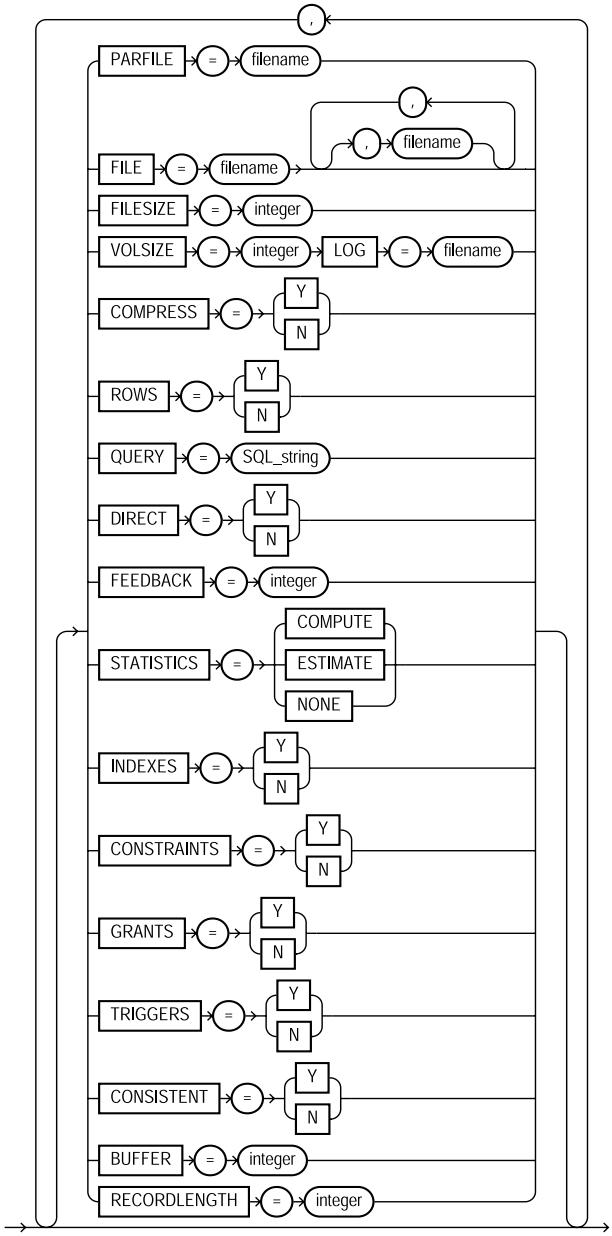
expopts_1



tablespaces_spec



Expopts_2


,
PARFILE
=
filename
,
,
filename
FILE
=
filename
FILESIZE
=
integer
VOLSIZE
=
integer
LOG
=
filename
COMPRESS
=
Y
N
ROWS
=
Y
N
QUERY
=
SQL_string
DIRECT
=
Y
N
FEEDBACK
=
integer
STATISTICS
=
COMPUTE
ESTIMATE
NONE
INDEXES
=
Y
N
CONSTRAINTS
=
Y
N
GRANTS
=
Y
N
TRIGGERS
=
Y
N
CONSISTENT
=
Y
N
BUFFER
=
integer
RECORDLENGTH
=
integer

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を決定するときは、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。

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、Export 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。

```
buffer_size = rows_in_array * maximum_row_size
```

0（ゼロ）を指定すると、一度に 1 行ずつしかフェッチされません。

LONG、LOB、BFILE、REF、ROWID、LOGICAL ROWID、DATE、またはオブジェクト型の列が含まれている表は 1 度に 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。

COMPRESS

デフォルト : Y

Export ユーティリティと Import ユーティリティによる、表データの初期エクステントの管理方法を指定します。

デフォルトの COMPRESS=Y では、インポート時に表データを 1 つの初期エクステントに整理統合するためのフラグが付きます。エクステント・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とき）、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 と指定すると、Export ユーティリティは、初期エクステントのサイズおよび第 2 エクステントのサイズなどが指定されている現行の記憶領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、またはデータベース・システムによって変更された値になります。たとえば、PCTINCREASE パラメータに 0（ゼロ）以外の値が指定されている場合、表が大きくなったときに第 2 エクステント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。記憶領域パラメータなどのデータ定義は、Import ユーティリティではなく、Export ユーティリティによって生成されるからです。したがって、エクスポート時に COMPRESS=Y を指定すると、そのデータは整理統合形式でしかインポートできません。

注意 : LOB データは圧縮されません。LOB データに関しては、初期エクステントのサイズと第 2 エクステントのサイズには元の値が使用されます。

CONSISTENT

デフォルト : N

Export 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、Export 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（デフォルト値）と指定すると、1 つのトランザクションでそれぞれの表がエクスポートされます。ただし、表の内側にネストした表がある場合には、外側の表と内側の各表は別のトランザクションとしてエクスポートされます。パーティション表の場合には、パーティションごとに別のトランザクションとして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。

ユーザー 1 とユーザー 2 という 2 人のユーザーがいて、ユーザー 1 がある表のパーティションをエクスポートし、ユーザー 2 が同じ表のデータを更新する場合のイベントの発生順序を次のチャートに示します。

時系列	ユーザー 1	ユーザー 2
1	TAB: P1 のエクスポートを開始	
2		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	
4	TAB: P2 をエクスポート	

エクスポートで CONSISTENT=Y を指定すると、ユーザー 2 に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 と指定すると、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ので、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、ユーザー 2 のトランザクションは部分的にしかエクスポート・ファイルに書き込まれないので、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 と指定しているときに更新量が多いと、ロールバック・セグメントが大きくなります。また、ロールバック・セグメントを走査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、次の点に注意してください。

- エクスポートに必要な時間と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、EMP 表と DEPT 表を CONSISTENT=Y と指定してまとめてエクスポートし、その後に残りの表をエクスポートします。

- 一貫性の維持が要求されるオブジェクトの数を最小にして、これらをまとめてエクスポートすると「スナップショットが古すぎます」というエラーが少なくなります。

  この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、読込み一貫性のあるイメージ（ビュー）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てしまった場合、データベースの読込み一貫性ビューを維持するために上書きによって消失した情報が必要となったときに、「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。

  このエラーを防ぐために、読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たり、可能ならばデータベースのトランザクション・レートを小さくします。）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。

  注意 : 増分エクスポートでは、CONSISTENT=Y を指定できません。

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、Export 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。

ダイレクト・パス・エクスポートの詳細は、1-41 ページ「ダイレクト・パス・エクスポート」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 つのドットで示す進捗メーターの表示を指定します。たとえば、FEEDBACK=10 と指定すると、10 行分のエクスポートが終了するたびにドットが 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。Export 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（1-19 ページ「FILESIZE」パラメータを参照）、複数のファイル名を指定できます。

エクスポートが FILESIZE に指定した最大値まで実行されると、現行ファイルへの書込みは中止され、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、エクスポートの完了または FILESIZE の最大値に再び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、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

Export ユーティリティは複数のエクスポート・ファイルへの書込みをサポートしており、Import ユーティリティは複数のエクスポート・ファイルから読取りができます。FILESIZE パラメータの値（バイト制限）を指定すると、エクスポートでは、それぞれのダンプ・ファイルに指定したバイト数が書き込まれます。

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ている場合、FILE パラメータから次のエクスポート・ファイルの名前が決められるか（1-19 ページ「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 を指定していないことになります）、FILE パラメータで指定したファイルの数にかかわらず、1 つのファイルにのみ書き込まれます。

> **注意 :** エクスポート・ファイルのサイズがディスク領域を超えたためにエクスポートが中止された場合は、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。

注意 : ファイルに格納可能な最大サイズは、オペレーティング・システムによって異なります。この最大サイズについて、FILESIZE を指定する前に、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。また、Export で指定するファイル・サイズが、Import 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 K（キロバイトの数）をつけて指定できます。たとえば、FILESIZE=2K は、FILESIZE=2048 と同じです。同様に、M はメガバイト（1024 × 1024）を、G はギガバイト（1024^3）を表します。B はバイトの省略です。この場合、本来のファイル・サイズの算出に乗算は不要です（FILESIZE=2048b は、FILESIZE=2048 と同じです）。

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を示します（データベース全体をエクスポートします）。全データベース・モードでエクスポートするには、FULL=Y と指定します。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 : N

エクスポート・パラメータの記述とともにヘルプ・メッセージが表示されます。

INCTYPE

デフォルト : なし

増分エクスポートのタイプを指定します。オプションは、COMPLETE および CUMULATIVE、INCREMENTAL です。詳細は、1-44 ページ「増分、累積および全エクスポート」を参照してください。

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

ファイル名を指定すると、情報メッセージおよびエラー・メッセージがこのファイルに記録されます。たとえば、次のように指定します。

```
exp system/manager 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。

OWNER

デフォルト : 定義なし

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 : 定義なし

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・ファイルの詳細は、1-12 ページ「コマンド行またはパラメータ・ファイルでのエクスポート・パラメータ指定」を参照してください。

QUERY

デフォルト : なし

表モード・エクスポートを実行するとき、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 SELECT 文に対する、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 より小さい従業員のみをエクスポートするには、次のように指定します（この例は UNIX ベースの場合です）。

```
exp scott/tiger tables=emp query=\"where job=\'SALESMAN\' and sal\<1600\"
```

> **注意 :** QUERY パラメータの値にはブランクが含まれるため、ほとんどのオペレーティング・システムでは、全部の文字列 `where job=\'SALESMAN\'` および `sal\<1600` を二重引用符で囲むか、何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。また、オペレーティング・システムの予約文字は、単一引用符、二重引用符および前述の UNIX の例では '<' を '\' で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。

このコマンドの実行時、Export によって、次のように SELECT 文が構築されています。

```
SELECT * FROM EMP where job='SALESMAN' and sal <1600;
```

QUERY 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。次に例を示します。

```
exp scott/tiger tables=emp,dept query=\"where job=\'SALESMAN\' and sal\<1600\"
```

この例では、問合せに一致した、EMP および DEPT の両方の行がアンロードされます。

また、Export 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 where job='SALESMAN' and sal <1600;

SELECT * FROM DEPT where where job='SALESMAN' and sal <1600;
```

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・モードまたはトランスポータブル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で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、内側にネストした表を持つ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、エクスポートの内容から判断することはできません。

RECORD

デフォルト : Y

増分エクスポートまたは累積エクスポートをシステム表 SYS.INCEXP および SYS.INCFIL、SYS.INCVID に記録するかどうかを指示します。これらの表の詳細は、1-50 ページ「システム表」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、エクスポート・ファイルを、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と、使用中のプラットフォーム固有の、BUFSIZ に関する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 の値と同等またはより大きい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、Export の I/O バッファのサイズ指定に使用できます。

追加情報 : 適切な値の決定や他のレコード・サイズでのファイルの作成の詳細は、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。

ROWS

デフォルト : Y

表データの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をインポートするときに生成されるデータベース・オプティマイザ統計のタイプを指定します。オプションは、ESTIMATE および COMPUTE、NONE です。オプティマイザおよびオプティマイザが使用する統計の詳細は、『Oracle8*i* 概要』を参照してください。Import のパラメータの詳細は、2-27 ページ「RECALCULATE_STATISTICS」および 2-63 ページ「統計情報のインポート」を参照してください。

場合によっては、Export によって、事前計算済みの統計情報がエクスポート・ファイルに書き込まれます。これは、ANALYZE コマンドが統計情報を再生成するのと同じです。

ただし、エクスポート時、次の場合は、計算済みのオプティマイザ統計は使用されません。

- 表に、システム生成の名前の索引がある場合（LOB 索引を含む）。
- 表に、システム生成の名前の列がある場合。
- エクスポート中、行にエラーがあった場合。
- クライアント・キャラクタ・セットまたは NCHARSET が、サーバー・キャラクタ・セットまたはサーバーの NCHARSET 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。
- 表に、分析された制約（チェック制約、UNIQUE および主キー制約）に基づく索引がある場合。
- 表に、システムによって生成された名前の索引があり、その索引が分析されている（IOT、ネストした表、特別な制約を課した索引のあるタイプ表）場合。

  注意 : ROWS=N を指定しても、計算済みの統計情報は、エキスポート・ファイルから除外されません。これによって、非本番データベースの問合せ生成プランを、本番データベースからの統計情報を使用して調整することができます。

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。

- スキーマには、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する。スキーマ名は、Export によって予約されている ORDSYS、MDSYS、CTXSYS および ORDPLUGINS です。
- 表名には、エクスポートされる表名を指定する。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の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ているときに、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。
- パーティションまたはサブパーティション名は、そのエクスポートが、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。

```
schema.tablename:partitionname
schema.tablename:subpartitionname
```

表を、表名とパーティション名の組合せで指定する場合、その表はパーティション化された表にしてください。また、パーティション名は、その表内のパーティションまたはサブパーティションの名前にしてください。

パーティション・レベル・エクスポートの例は、1-33 ページ「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。

> **追加情報 :** 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には、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

表名を引用符で囲まない限り、コマンド行で指定する表名の中にシャープ（#）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と、表名にシャープ（#）記号が使用されている場合、シャープ（#）記号より右側の文字はコメントと解釈されます。

たとえば、パラメータ・ファイルに次の行が記述されていると、EMP# の右側はすべてコメントと解釈されるので、表 DEPT および表 MYDATA はエクスポートされません。

```
TABLES=(EMP#, DEPT, MYDATA)
```

一方、次の例では、3 つの表はすべてエクスポートされます。

```
TABLES=("EMP#", DEPT, MYDATA)
```

> **注意 :** 表名を引用符で囲んで指定する場合、指定した表名の大文字と小文字は区別されます。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。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。

上の例では、EMP# という名前の表はエクスポートされますが、emp# という名前の表はエクスポートされません。表 DEPT と表 MYDATA は引用符で囲まれていないので、大文字および小文字の区別はされません。

> **追加情報 :** オペレーティング・システムによっては、二重引用符ではなく一重引用符を使用しなければならない場合と、逆に二重引用符を使用しなければならない場合があります。ご使用のオペレーティング・システム固有の Oracle ドキュメントで確認してください。表の命名方法に制限の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（#）（またはその他の特別な文字）には特別な意味があります。これらの文字を使用する場合には、シェルを通過して Export に移動できるように、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 を指定する場合、このパラメータを使用して、データベースからエクスポート・ファイルにエクスポートされる表領域をリストします。

詳細は、1-57 ページ「トランスポータブル表領域」を参照してください。

TRANSPORT_TABLESPACE

デフォルト : N

Y を指定すると、トランスポータブル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。詳細は、『Oracle8*i* 管理者ガイド』および『Oracle8*i* 概要』を参照してください。

USERID

デフォルト : なし

Export を実行するユーザーのユーザー名およびパスワード（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
```

詳細は、1-11 ページ「SYSDBA としての Export ユーティリティの起動」を参照してください。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（1-11 ページ参照）。

Net8 については、*@connect_string* 句を任意に指定できます。この句 *@connect_string* の正確な構文は、ご使用の Net8 プロトコルのユーザーズ・ガイドを参照してください。また、『Oracle8*i* 分散システム』も参照してください。

VOLSIZE

それぞれのテープ媒体のエクスポート・ファイルについて、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、オペレーティング・システムに固有のドキュメントを参照してください。

VOLSIZE の値は、数字に K（キロバイトの数）を付けて指定できます。たとえば、VOLSIZE=2K は、VOLSIZE=2048 と同じです。同様に、M はメガバイト（1024 × 1024）を、G はギガバイト（1024^3）を表します。B はバイトの省略です。この場合、本来のファイル・サイズの算出に乗算は不要です。（VOLSIZE=2048b は、VOLSIZE=2048 と同じです）

パラメータ間の相互作用

パラメータによっては、パラメータ間で矛盾することがあります。たとえば、TABLES と OWNER の両方を指定すると、矛盾が生じます。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/manager OWNER=jones TABLES=scott.emp
```

同様に、OWNER と FULL=Y や TABLE と FULL=Y も矛盾します。

ROWS=N と INCTYPE=INCREMENTAL の両方を指定することは可能ですが、ROWS=N（データなし）と指定すると、増分エクスポートの本来の目的が生かされません。増分エクスポートは、変更された表のバックアップ・コピーを作成するためのものです。

エクスポート・セッションの例

次に、全データベース、ユーザー、表の各モードでのコマンド行方式およびパラメータ・ファイル方式の使用例を示します。

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

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 GRANT（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/manager 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
```

コマンド行方式

```
> exp system/manager full=Y file=dba.dmp grants=Y rows=Y
```

エクスポート・メッセージ

```
Export: Release 8.1.5.0.0 - Production on Fri Oct 30 09:34:00 1998

(c) Copyright 1998 Oracle Corporation.  All rights reserved.

Connected to: Oracle8 Enterprise Edition Release 8.1.5.0.0 - Production
With the Partitioning option
PL/SQL Release 8.1.5.0.0 - Production
Export done in WE8DEC character set and WE8DEC 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 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                  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$LOB          0 rows exported
. . exporting table        SQLPLUS_PRODUCT_PROFILE          0 rows exported
. about to export OUTLN's tables via Conventional Path ...
. . exporting table                            OL$          0 rows exported
. . exporting table                        OL$HINTS          0 rows exported
. 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 referential integrity constraints
. exporting synonyms
. exporting views
. exporting stored procedures
. exporting operators
. exporting indextypes
. exporting bitmap, functional and extensible indexes
. exporting posttables actions
. exporting triggers
. exporting snapshots
. 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
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では、1 人以上のデータベース・ユーザーのバックアップが可能です。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と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とる場合や、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 が自分の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
```

コマンド行方式

```
> 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 8.1.5.0.0 - Production on Fri Oct 30 09:35:33 1998

(c) Copyright 1998 Oracle Corporation.  All rights reserved.

Connected to: Oracle8 Enterprise Edition Release 8.1.5.0.0 - Production
With the Partitioning option
PL/SQL Release 8.1.5.0.0 - Production
Export done in WE8DEC character set and WE8DEC NCHAR character set
. exporting pre-schema procedural objects and actions
. exporting foreign function library names for user SCOTT
. 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 snapshots
. exporting snapshot logs
. exporting job queues
. exporting refresh groups and children
. exporting dimensions
. exporting post-schema procedural objects and actions
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には、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の指定があれば、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=schema.table` と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。

`schema` を指定しないと、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、表 `a` の場合は SYSTEM、表 `c` 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/manager tables=(a, scott.b, c, mary.d)
```

EXP_FULL_DATABASE ロールを持たないユーザーがエクスポートできるのは、そのユーザー自身が所有する表のみです。EXP_FULL_DATABASE ロールを持つユーザーのみが他のユーザーに依存するオブジェクトのエクスポートを実行できます。非特権ユーザーは、そのユーザー自身が所有する、指定した表の依存オブジェクトしかエクスポートできません。

表モードの Export ユーティリティには、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、表の非クラスタ化に、表モードを使用できます。

例 1

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/manager 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
```

コマンド行方式

```
> exp system/manager tables=(scott.emp,blake.dept) grants=Y indexes=Y
```

エクスポート・メッセージ

```
Export: Release 8.1.5.0.0 - Production on Fri Oct 30 09:35:59 1998

(c) Copyright 1998 Oracle Corporation.  All rights reserved.

Connected to: Oracle8 Enterprise Edition Release 8.1.5.0.0 - Production
With the Partitioning option
PL/SQL Release 8.1.5.0.0 - Production
Export done in WE8DEC character set and WE8DEC 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

この例では、ユーザー BLAKE が自分の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
```

コマンド行方式

```
> 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 8.1.5.0.0 - Production on Fri Oct 30 09:36:08 1998

(c) Copyright 1998 Oracle Corporation.  All rights reserved.

Connected to: Oracle8 Enterprise Edition Release 8.1.5.0.0 - Production
With the Partitioning option
PL/SQL Release 8.1.5.0.0 - Production
Export done in WE8DEC character set and WE8DEC 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.
```

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

EMP というパーティション表があり、この表にパーティション M とパーティション Z があると仮定します（従業員名でパーティション化されている）。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
```

コマンド行方式

```
> exp scott/tiger tables=emp rows=Y
```

エクスポート・メッセージ

```
Export: Release 8.1.5.0.0 - Production on Fri Oct 30 09:36:23 1998

(c) Copyright 1998 Oracle Corporation.  All rights reserved.

Connected to: Oracle8 Enterprise Edition Release 8.1.5.0.0 - Production
With the Partitioning option
PL/SQL Release 8.1.5.0.0 - Production
Export done in WE8DEC character set and WE8DEC 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

EMP というパーティション表があり、この表にパーティション M とパーティション Z があると仮定します（従業員名でパーティション化されている）。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
```

コマンド行方式

```
> exp scott/tiger tables=emp:m rows=Y
```

エクスポート・メッセージ

```
Export: Release 8.1.5.0.0 - Production on Fri Oct 30 09:36:29 1998

(c) Copyright 1998 Oracle Corporation.  All rights reserved.

Connected to: Oracle8 Enterprise Edition Release 8.1.5.0.0 - Production
With the Partitioning option
PL/SQL Release 8.1.5.0.0 - Production
Export done in WE8DEC character set and WE8DEC 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

EMP というパーティション表があり、この表には M と Z という 2 つのパーティションがあるとします。表 EMP は、複合方式でパーティション化されています。M には、sp1 と sp2 というサブパーティションがあり、Z には sp3 と sp4 というサブパーティションがあります。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 と sp2）がエクスポートされます。表および指定したサブパーティション（sp4）をエクスポートする場合、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 partfile=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
```

コマンド行方式

```
> exp scott/tiger tables=(emp:m, emp:sp4) rows=Y
```

エクスポート・メッセージ

```
Export: Release 8.1.5.0.0 - Development on Fri Oct 30 09:36:29 1998

(c) Copyright 1998 Oracle Corporation.  All rights reserved.


Connected to: Oracle8i Enterprise Edition Release 8.1.5.0.0 - Development
With the Partitioning option
```

```
PL/SQL Release 8.1.5.0.0 - Production
Export done in WE8DEC character set and WE8DEC NCHAR character set

About to export specified tables via Conventional Path ...

. . exporting table                          EMP
. . exporting composite partition                  M
. . exporting subpartition                         SP1         4 rows exported
. . exporting subpartition                         SP2         0 rows exported
. . exporting composite partition                  Z
. . exporting subpartition                         SP4         1 rows exported
Export terminated successfully without warnings.
```

対話方式の使用

コマンド行から Export ユーティリティを起動する場合、引数を指定しないと、Export ユーティリティは対話方式で起動します。対話方式では、Export ユーティリティのすべての機能について、入力を求めるメッセージが表示されるわけではありません。対話方式は下位互換性のためにのみ提供されています。

コマンド行でユーザー名 / パスワードを指定しないと、この情報を入力するように Export ユーティリティから要求されます。

SYSDBA としての対話形式による Export ユーティリティの起動

通常は、Export を SYSDBA として起動する必要はありません。ただし特定の状況下で、オラクル社カスタマ・サポートの要求によって SYSDBA で起動する場合もあります。

Export の対話形式モードを使う場合は、SYSDBA として接続するか、@instance として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" はユーザー名として入力する必要があります。

Export の対話形式モードでのユーザー名プロンプトに対する入力例を示します。

```
username/password@instance as sysdba
username/password@instance
username/password as sysdba
username/password
username@instance as sysdba  (prompts for password)
username@instance            (prompts for password)
username                     (prompts for password)
username AS sysdba           (prompts for password)
/    as sysdba               (no prompt for password, OS authentication
                             is used)
```

```
/                           (no prompt for password, OS authentication
                            is used)
/@instance as sysdba        (no prompt for password, OS authentication
                            is used)
/@instance                  (no prompt for password, OS authentication
                            is used)
```

注意 : 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 はユーザー名としてのみ指定できます。

次に、Export ユーティリティは次のプロンプトを表示します。

```
Enter array fetch buffer size: 30720 >
Export file: expdat.dmp >
(1)E(ntire database), (2)U(sers), or (3)T(ables): (1)E >
Export grants (yes/no): yes >
Export table data (yes/no): yes >
Compress extents (yes/no): yes >
Export done in WE8DEC character set and WE8DEC 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 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                    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        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
. 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 referential integrity constraints
. exporting synonyms
. exporting views
. exporting stored procedures
. exporting operators
. exporting indextypes
. exporting bitmap, functional and extensible indexes
. exporting posttables actions
. exporting triggers
. exporting snapshots
. 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
Export terminated successfully without warnings.
```

一部のプロンプトは、別のプロンプトに対してユーザーが入力した応答に対して表示されるため、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限りません。一部のプロンプトでは、デフォルト値が表示されます。このデフォルト値を受け入れる場合は [Return] を押します。

制限事項

対話方式を使用する場合は、次の点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よう Export ユーティリティによって求められる。ユーザー名のリストを入力し終わったら、[Return] を押して現行のエクスポート・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そ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る。

  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」の入力要求に対して表を指定しないと、Export ユーティリティは終了する。

警告、エラーおよび完了メッセージ

この項では、特定の状況で Export ユーティリティによって発行されるメッセージについて説明します。

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 番目は、LOG パラメータを使う方法です（1-21 ページ「LOG」を参照）。2 番目は、システムでサポートされている場合に限りますが、Export 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 ドキュメントを参照してください。

警告メッセージ

Export ユーティリティは、発生したエラーが致命的なものでない限り、処理を続行します。たとえば、表のエクスポート中にエラーが発生した場合、Export ユーティリティはエラー・メッセージを表示し（またはログを記録し）、次の表にスキップして処理を続けます。致命的でないエラーは、警告と呼ばれます。

Export ユーティリティは、無効なオブジェクトを見つけると、警告を発します。

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、Export 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、次に示すように、警告を発して処理を正常に終了します。

```
> exp scott/tiger tables=xxx,emp

Export: Release 8.1.5.0.0 - Production on Fri Oct 30 09:38:11 1998

(c) Copyright 1998 Oracle Corporation.  All rights reserved.

Connected to: Oracle8 Enterprise Edition Release 8.1.5.0.0 - Production
With the Partitioning option
PL/SQL Release 8.1.5.0.0 - Production
Export done in WE8DEC character set and WE8DEC NCHAR character set

About to export specified tables via Conventional Path ...
EXP-00011: SCOTT.XXX does not exist
. . exporting table                          EMP         14 rows exported
Export terminated successfully with warnings.
```

致命的エラー・メッセージ

エラーの中には致命的なものもあり、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、またはメモリーなどのリソースが使用できないか、リソースを使い尽くしてしまっ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と、Export ユーティリティは次に示すように、致命的エラー・メッセージを発行します。

```
EXP-00024: Export views not installed, please notify your DBA
```

追加情報 : メッセージの詳細は、『Oracle8*i* エラー・メッセージ』および、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。

完了メッセージ

問題なくエクスポートが完了した場合、「Export terminated successfully without warnings.」というメッセージが表示されます。致命的でないエラーが 1 つ以上発生しても、エクスポートはそのまま続行され、処理が完了した場合は「Export terminated successfully with warnings.」というメッセージが表示されます。致命的なエラーが起きた場合、Export ユーティリティは即時終了し、「Export terminated unsuccessfully.」というメッセージが表示されます。

ダイレクト・パス・エクスポート

Export ユーティリティでは次の 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

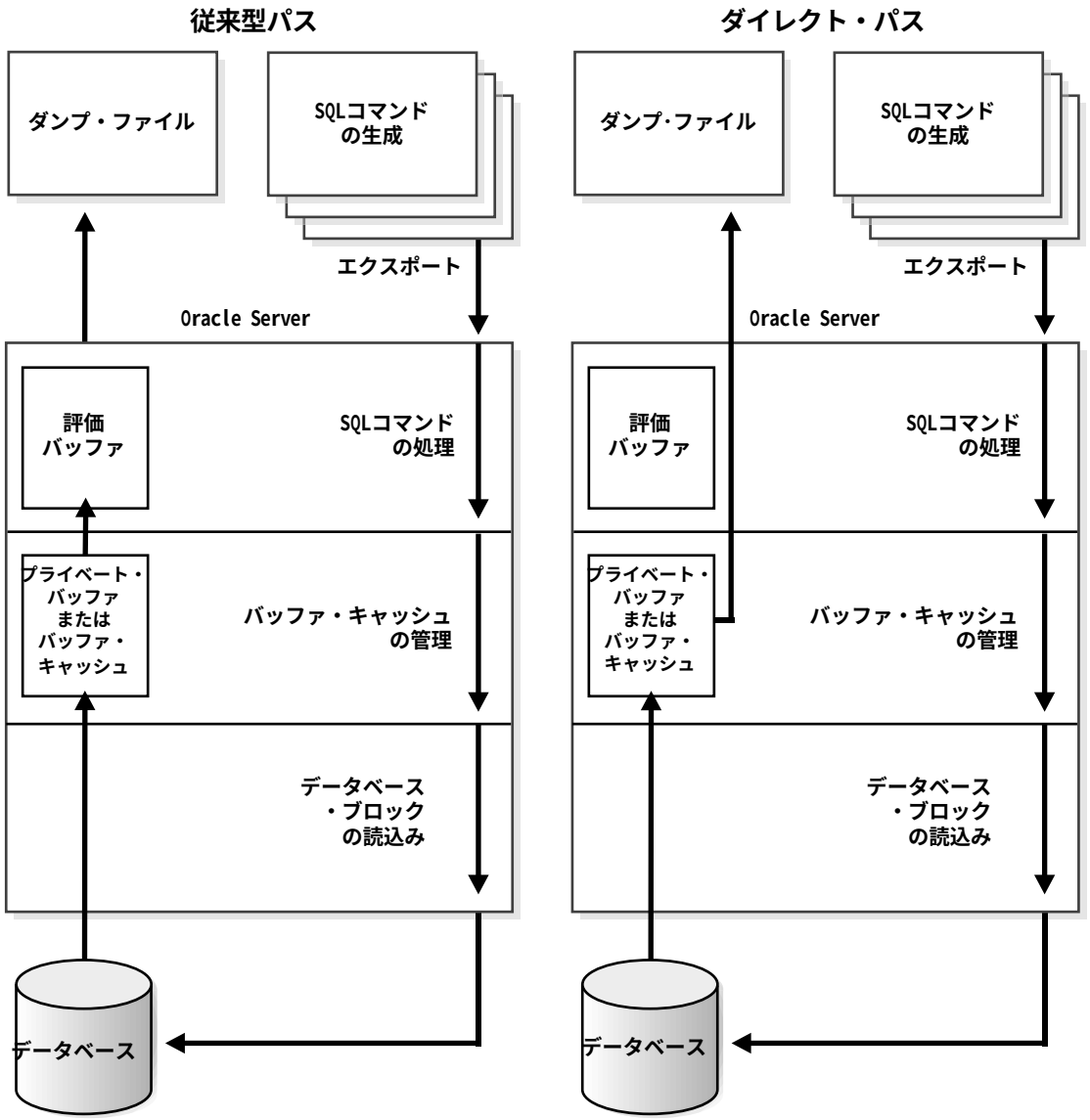
従来型パス・エクスポートでは、SQL SELECT 文によって、データベースの表からデータが抽出されます。データはディスクからバッファ・キャッシュに読み込まれ、行は評価バッファに転送されます。式の評価が終了すると、そのデータはエクスポートを実行するクライアントへ転送され、そこでエクスポート・ファイルに書き込まれます。

ダイレクト・パス・エクスポートは、従来型パスによるエクスポートに比べて非常に高速にデータを抽出できる機能です。ダイレクト・パス・エクスポートでは、SQL コマンド処理レイヤーをバイパスして直接データを読み込み、可能な限りデータのコピーを省くことによって、パフォーマンスの向上を実現します。

1-42 ページ図 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抽出方法の違いを示します。

ダイレクト・パス・エクスポートでは、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、行がエクスポートを行うクライアントに直接転送されます。評価バッファはバイパスします。すでにデータは、Export ユーティリティが要求する形式になっているので、不必要なデータ変換をしなくて済みます。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 従来型パスおよびダイレクト・パスでのデータベースの読込み



ダイレクト・パス・エクスポートの起動

ダイレクト・パス・エクスポートを使用するには、コマンド行またはパラメータ・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。この場合は従来型パスで表が抽出されます。

注意 : Export ユーティリティの BUFFER パラメータを使用できるのは、従来型パス・エクスポートのみです。ダイレクト・パス・エクスポートでは、エクスポート・ファイルへの書込みに使用するバッファのサイズはパラメータ RECORDLENGTH で指定します。

制限事項 : ダイレクト・パスを使用した一部の表はエクスポートできません。たとえば、LOB にオブジェクト機能を使用した表はエクスポートできません。エクスポートにダイレクト・パスを指定する場合、オブジェクトおよび LOB を含む表は、従来型パスを使用してエクスポートされます。

キャラクタ・セット変換

ダイレクト・パス・エクスポートでは、データベース・サーバーのキャラクタ・セットのみが使用されます。エクスポート・セッションのキャラクタ・セットがデータベースのキャラクタ・セットと異なる場合、エクスポートを開始すると、警告が表示されて異常終了します。その場合は、NLS_LANG パラメータで、そのセッションで使用されるキャラクタ・セットがデータベースのキャラクタ・セットと同じになるように設定してから、エクスポートを再実行してください。

パフォーマンスについて

ダイレクト・パス・エクスポートを起動するときに、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・ファイルの常駐するディスク・ドライブとは別にします。）

ダイレクト・パス・エクスポートを実行するときは、RECORDLENGTH パラメータを DB_BLOCK_SIZE データベース・パラメータと等しくして、それぞれの表に対する走査によって、データとして使用できる完全なデータベース・ブロックが戻されるようにします。このサイズがエクスポートの I/O バッファに合わない場合、それぞれのデータベース・ブロックを数回に分けてデータがエクスポート・ファイルに書き込まれます。

RECORDLENGTH には、一般的に次の値を推奨します。

- 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。

制限事項 : ダイレクト・パス・エクスポートの起動には対話方式を使用できません。

増分、累積および全エクスポート

重要 : 増分、累積および全エクスポートは、次のリリースから段階的に廃止される機能です。今後は、Oracle の Backup および Recovery Manager を使用してのデータベース・バックアップに移行してください。詳細は、『Oracle8*i* 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。

制限事項

- 増分エクスポート、累積エクスポートおよび全エクスポートは、全データベース・モード（FULL=Y）でのみ実行できます。ユーザーについては、EXP_FULL_DATABASE ロールを持つユーザーのみが、この 3 種類のエクスポートをすべて実行できます。EXP_FULL_DATABASE ロールには、増分エクスポートを記録するシステム表の修正に必要な権限が含まれています。システム表については、1-50 ページ「システム表」を参照してください。
- 増分エクスポートでは読込み一貫性を指定できません。

基本バックアップ

累積エクスポートおよび増分エクスポートを使用する場合は、定期的に全エクスポートを実行して基本バックアップを作成してください。全エクスポートの実行後に、増分エクスポートを頻繁に実施し、累積エクスポートをそれより少ない頻度で実施します。一定の期間が経過したら、再び全エクスポートを実行してこの手順を繰り返します。

増分エクスポート

増分エクスポートでは、前回のエクスポート（増分エクスポートまたは累積エクスポート、全エクスポートのいずれか）実施後に変更された表のバックアップのみが作成されます。増分エクスポートでは、変更された行のみではなく、表定義およびそのすべてのデータがエクスポートされます。通常、増分エクスポートは、累積または全エクスポートよりも頻繁に実行します。

時刻 1 で全エクスポートを実行したと仮定します。1-45 ページ図 1-3 に、3 つの表が変更された後に、時刻 2 の増分エクスポートを実行した場合の例を示します。この場合、変更された表および対応付けられた索引のみがエクスポートされます。

図 1-3 時刻 2 の増分エクスポート

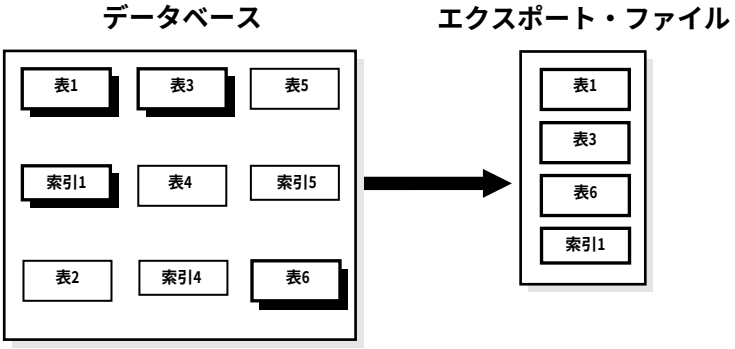
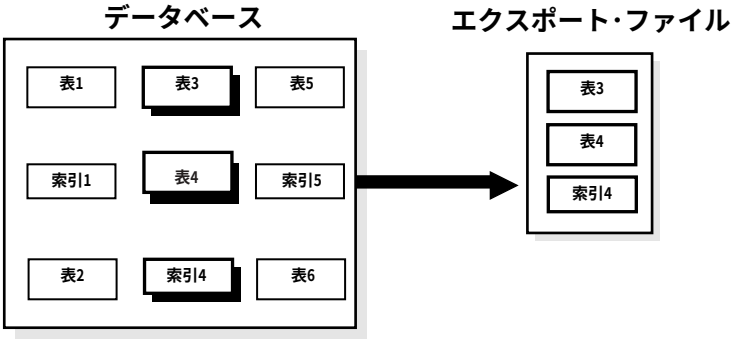


図 1-4 には、時刻 2 で 2 つの表が変更された後に実行された時刻 3 の別の増分エクスポートを示します。表 3 は再度変更されたため、時刻 2 と同様時刻 3 でもエクスポートされます。

図 1-4 時刻 3 の増分エクスポート

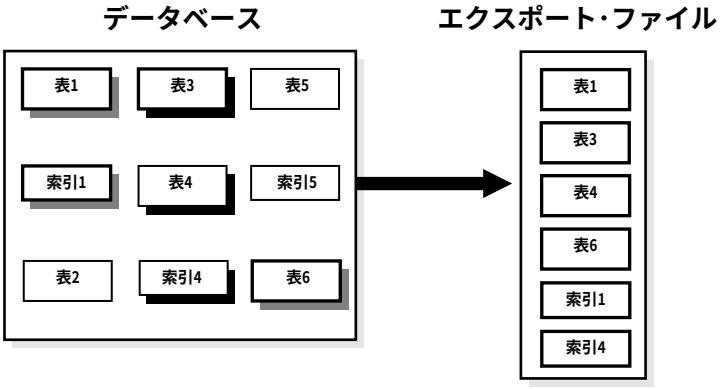


累積エクスポート

累積エクスポートは、前回の累積エクスポートまたは全エクスポート実施後に変更された表のバックアップを作成します。つまり、累積エクスポートはいくつかの増分エクスポートを圧縮して 1 つの累積エクスポート・ファイルに入れます。累積エクスポートを実施すると、前に作成した増分エクスポート・ファイルが累積エクスポート・ファイルで置き換えられるので、増分エクスポート・ファイルを保存する必要がなくなります。

図 1-5 に、時刻 4 の累積エクスポートを示します。表 1 および表 6 は時刻 3 以降に変更されているので、時刻 1 の全エクスポート以降に変更されたすべての表がエクスポートされます。

図 1-5 時刻 4 の累積エクスポート



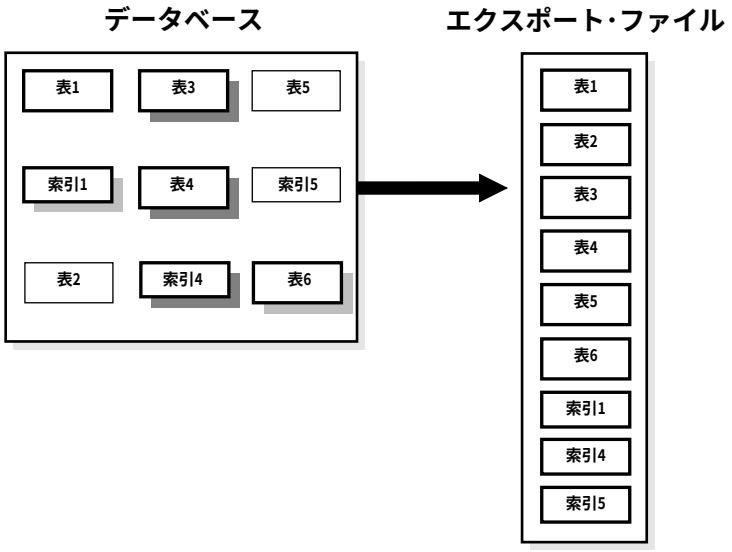
この累積エクスポート・ファイルには、時刻 2 および時刻 3 の増分エクスポートが含まれます。ただし、どちらの時刻でも修正された表 3 はエクスポート・ファイルに 1 度のみ現れています。このように、累積エクスポートでは、増分エクスポートを複数回実行する場合よりも領域を節約できます。

全エクスポート

全エクスポートは、増分エクスポートおよび累積エクスポートに対する基礎を確立します。全エクスポートの機能は全データベース・エクスポートとほぼ同じですが、全エクスポートでは、増分エクスポートおよび累積エクスポートの実施記録用の表も更新します。

1-47 ページ図 1-6 に、時刻 5 の全エクスポートを示します。全エクスポートでは、データベースのオブジェクトの変更時刻（または変更の有無）にかかわらず、すべてのオブジェクトがエクスポートされます。

図 1-6 時刻 5 の全エクスポート



運用方法

次に、累積エクスポートおよび増分エクスポートの運用方法の例を示します。

データ・センターの管理者として、次のタスクを実行するとします。

- 3 週間ごとに全エクスポート（X）
- 週末ごとに累積エクスポート（C）
- 毎晩の増分エクスポート（I）

エクスポート・スケジュールは次のとおりです。

```
DAY: 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22
     X  I  I  I  I  I  I  C  I  I  I  I  I  I  C  I  I  I  I  I  I  X
    Sun                 Sun                 Sun                 Sun
```

18 日目のデータベースを復元する場合、まず 18 日目に実施した増分エクスポートからシステム情報をインポートします。その後で次のものからデータをインポートします。

1. 1 日目に実施した全エクスポート
2. 8 日目に実施した累積エクスポート
3. 15 日目に実施した累積エクスポート
4. 16 日目から 18 日目まで実施した 3 回分の増分エクスポート

2 日目から 7 日目までの 5 日分の増分エクスポートは、8 日目の累積エクスポート実行の際に廃棄できます。これは、8 日目の累積エクスポートによって、それまでの増分エクスポートの内容がすべて取り込まれるためです。同様に、9 日目から 14 日目までの増分エクスポートも、15 日目の累積エクスポート時に廃棄できます。

注意 : 増分エクスポートおよび累積エクスポート、全エクスポートを指定する構文は、1-20 ページ INCTYPE で説明しています。

エクスポートの対象となるデータ

増分エクスポートおよび累積エクスポートの目的は、前回のエクスポート実施後に変更されたデータベース・オブジェクト（クラスタ、表、ビュー、シノニムなど）のみを識別し、エクスポートすることにあります。それぞれの表には、データ、索引、権限、監査、トリガーおよびコメントなどその他のオブジェクトが対応付けられています。

表またはビューに対する権限体系全体は、基礎になる実表とともにエクスポートされます。索引は、誰が作成したかにかかわりなく、実表とともにエクスポートされます。基礎ビューが取得される場合、実表ではなく、ビューのトリガーがエクスポートされます。

表に対してなんらかの変更（UPDATE または INSERT、DELETE）がなされると、その表は自動的に増分エクスポートの対象となります。表をエクスポートすると、内側のネストした表や LOB 列もすべてエクスポートされます。内側のネストした表を変更すると、外側の表がエクスポートされます。LOB 列を変更すると、LOB データが格納されている表全体がエクスポートされます。

また、次の処理によってデータベース構造が変更されている場合には、基礎となる実表およびデータがエクスポートされます。

- 表の作成
- ALTER TABLE 文による表定義の変更
- コメントの追加または編集
- 監査オプションの更新
- 権限の変更（すべてのレベル）

- 索引の追加または削除
- ALTER INDEX 文による索引記憶領域パラメータの変更

実表とデータの他に、次のデータがエクスポートされます。

- すべてのシステム・オブジェクト（表領域定義、ロールバック・セグメント定義、ユーザー権限を含む。ただし、一時セグメントを除く）
- 削除されたオブジェクトに関する情報
- 最後のエクスポート実行後に作成されたクラスタ、表、ビュー、プロシージャ、ファンクション、ディメンションおよびシノニム
- すべての型定義

  注意 : Export ユーティリティでは、セキュリティ上の理由から、Import ユーティリティに影響が及ばないよう、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、アクセス権が変更され、ユーザーがそれに気付かない場合があります。また、権限が強制的にインポートされなければ、ユーザーは適切な権限の設定を、インポートの際に必要に応じて行うことができます。

増分エクスポート・セッションの例

次に、表 SCOTT.EMP および表 SCOTT.DEPT が変更された後に実施した増分エクスポート・セッションの例を示します。

```
> exp system/manager full=y inctype=incremental

Export: Release 8.1.5.0.0 - Production on Fri Oct 30 09:40:11 1998

(c) Copyright 1998 Oracle Corporation.  All rights reserved.

Connected to: Oracle8 Enterprise Edition Release 8.1.5.0.0 - Production
With the Partitioning option
PL/SQL Release 8.1.5.0.0 - Production
Export done in WE8DEC character set and WE8DEC 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 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 ...
. about to export OUTLN's tables via Conventional Path ...
. about to export DBSNMP's tables via Conventional Path ...
. about to export SCOTT's tables via Conventional Path ...
. . exporting table                          DEPT          8 rows exported
. . exporting table                           EMP         23 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 referential integrity constraints
. exporting synonyms
. exporting views
. exporting stored procedures
. exporting operators
. exporting indextypes
. exporting bitmap, functional and extensible indexes
. exporting posttables actions
. exporting triggers
. exporting snapshots
. 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 information about dropped objects
Export terminated successfully without warnings.
```

システム表

ユーザー SYS は、Export ユーティリティによって管理される 3 つの表（INCEXP、INCFIL、INCVID）を所有しています。RECORD=Y（デフォルト値）と指定すると、これらの表が更新されます。これらの表は、一切変更しないでください。

SYS.INCEXP

表 SYS.INCEXP は、どのエクスポートでどのオブジェクトがエクスポートされたかを記録します。

この表には、次の列があります。

OWNER# 表を含むスキーマのユーザー ID。

NAME オブジェクト名。OWNER#、NAME および TYPE の組合せが表の主キーとなります。

TYPE オブジェクトの型。（INDEX、TABLE、CLUSTER、VIEW、SYNONYM、SEQUENCE、PROCEDURE、FUNCTION、PACKAGE、TRIGGER、DIMENSION、OPERATOR、INDEXTYPE、SNAPSHOT、SNAPSHOT LOG または PACKAGE BODY を指定するコード）。

CTIME このオブジェクトを含む累積エクスポートを、前回実行した日付および時刻。

ITIME このオブジェクトを含む増分エクスポートを、前回実行した日付および時刻。

EXPID 増分または累積エクスポートの ID で、SYS.INCFIL 表にも記録されます。

さまざまな方法でこの情報を利用できます。たとえば、エクスポート・ファイルを記録するために各エクスポートの後で、SYS.INCEXP からのレポートを生成することができます。増分エクスポートに関する情報を表示するには、ビュー DBA_EXP_OBJECTS および DBA_EXP_VERSION、DBA_EXP_FILES を使用します。

SYS.INCFIL

SYS.INCFIL 表には、実施された増分エクスポートおよび累積エクスポートが記録され、各エクスポートに対して固有の識別子が割り当てられます。

この表には、次の列があります。

EXPID 増分または累積エクスポートの ID で、SYS.INCEXP 表にも記録されます。

EXPTYPE エクスポートの種類（増分または累積）。

EXPFILE エクスポート・ファイル名。

EXPDATE エクスポートの日付。

EXPUSER エクスポートを実施したユーザーのユーザー名。

INCTYPE=COMPLETE パラメータを使用してエクスポートすると、以前のすべての入力が SYS.INCFIL から削除され、列 EXPTYPE に "x" を指定した新しい行が追加されます。

SYS.INCVID

表 SYS.INCVID には、最後に実施された有効なエクスポートの EXPID が記録される列が含まれています。この情報により、次のエクスポートの EXPID が決定されます。

ネットワークに関する考慮事項

この項では、ネットワークを介した Export および Import を実行する際の考慮事項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、ファイルの破損を防止してください。たとえば、FTP または類似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。

Net8 を利用したエクスポートおよびインポート

Net8（および SQL*Net バージョン 2）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、Export 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、Import 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。

Export ユーティリティで Net8 を使用するには、次の例に示すように、`exp` コマンドの入力時に `username/password` の後に `@connect_string` を指定します。

```
exp scott/tiger@SUN2 FILE=export.dmp FULL=Y
```

> **追加情報 :** この句の構文の詳細は、ご使用の Net8 または SQL*Net プロトコルのユーザーズ・ガイドを参照してください。Net8 または Oracle Names の詳細は、『Oracle8*i* Net8 管理者ガイド』を参照してください。

キャラクタ・セットおよび NLS に関する考慮事項

この項では、各国語サポート（NLS）に関連した Export ユーティリティおよび Import 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

従来型パス・エクスポートの場合、ユーザー・セッション用として指定されたキャラクタ・セット（7 ビットの ASCII や IBM コード・ページ 500（EBCDIC）など）、または JA16EUC などの、データベース・サーバー・キャラクタ・セットから必要に応じて変換した Oracle NLS キャラクタ・セットでエクスポート・ファイルが書き出されます。キャラクタ・セットがエクスポート・ファイルで使用されたものと異なっている場合、Import ではユーザー・セッション用として指定されたキャラクタ・セットに文字データが変換されます。

エクスポート・ファイルでは、ファイル内で文字データ用に使用される文字コード体系が識別されます。キャラクタ・セットがシングルバイト・キャラクタ・セットで（たとえば、EBCDIC または USASCII7）、ターゲット・データベースに使用されるキャラクタ・セットもシングルバイト・キャラクタ・セットのときは、NLS_LANG 環境変数の指定に従い、インポート中に、ユーザー・セッションに指定された文字コード体系にデータが自動的に変換されます。データはセッションのキャラクタ・セットに変換された後、さらにデータベース・キャラクタ・セットに変換されます。

変換時に、ターゲット・キャラクタ・セットに等しい文字がないエクスポート・ファイル中の文字は、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・セットにより定義されます。）100% 完全に変換されるようにするには、ターゲット・キャラクタ・セットを、ソース・キャラクタ・セットのスーパーセットか、またはソース・キャラクタ・セットと等しくなるようにしてください。

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失することがあります（つまり、等価な 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マシン固有のキャラクタ・セットが 7 ビットであるか、NLS_LANG オペレーティング・システム環境変数に 7 ビット・キャラクタ・セットが設定されている場合です。アクセント付きの文字からアクセント記号が消失するのが最もよく見られる例です。

エクスポートおよびインポートでは、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。

ダイレクト・パス・エクスポートを実行する場合は、ユーザー・セッションのキャラクタ・セットはデータベースのキャラクタ・セットと同じである必要があります。

詳細は、『Oracle8*i* NLS ガイド』を参照してください。

エクスポートおよびインポート中の NCHAR 変換

Export ユーティリティでは、NCHAR データはエクスポート・サーバーに対応する各国文字キャラクタ・セットでエクスポートされます。（各国文字キャラクタ・セットは、データベース作成時に NATIONAL character set 文で指定します。）

Import ユーティリティによって、インポート・サーバーの各国文字キャラクタ・セットに自動的にデータが変換されます。

マルチバイト・キャラクタ・セットと Export および Import

マルチバイト・キャラクタ・セット（たとえば、中国語や日本語）で作成されたエクスポート・ファイルは、キャラクタ・セットの文字長が同じ、またはインポート・キャラクタ・セットの文字の最大幅とエクスポート・キャラクタ・セットの文字の最小幅の割合が 1 であるシステムにインポートする必要があります。この割合が 1 でないと、Import ユーティリティにおいて文字データがインポート・キャラクタ・セットに変換されません。

> **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、Export 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 Export

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ジョブをインスタンスに関連付ける場合、Import および Export ユーティリティでインスタンス親和性を使用する方法については、『Oracle8*i* 管理者ガイド』、『Oracle8*i* リファレンス・マニュアル』および『Oracle8*i* Parallel Server 概要および管理』を参照してください。なお、リリース 8.0 および 8.1 の両方を使用している場合は、発生しそうな互換性の問題について、『Oracle8*i* 移行ガイド』を参照してください。

ファイン・グレイン・アクセスのサポート

ファイン・グレイン・アクセス・ポリシーを使用可能にして、表をエクスポートできます。

ただし、そのような表を含むエクスポート・ファイルからインポートするユーザーには、適切な権限が必要です（特に、表のセキュリティ・ポリシーを回復させるために DBMS_RLS パッケージに対する実行権限）。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、読込み権限のある行のみがエクスポートされます。

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事項

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場合、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最善の方法は、エクスポート中に順序番号にアクセスしないようにすることです。

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れば、現行のデータベースでこれらの順序番号を使用することができます。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。キャッシュされたが使用されていない順序番号は、順序番号がインポートされたときに失われます。

LONG データ型および LOB データ型のエクスポート

エクスポート時には、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。

注意 : LOB 列のデータをすべて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位置に移したら、データベース管理者は、ライブラリを移動させて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することができず、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、あらかじめオフライン・ローカル管理表領域を作成しておき、DDL コマンドが不完全な表領域を参照することによってエラーになるのを防ぐ必要があります。

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す場合、データベース管理者は、その新しい位置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、ディレクトリ別名の使用前に、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 列または属性で参照される外部ファイルの内容は格納されません。エクスポート時には、ファイルの名前とディレクトリ別名のみがコピーされ、インポート時に復元されます。データベースの場所の変更によって、旧ディレクトリではファイルにアクセスできなくなったときには、データベース管理者は、ファイルへアクセスできる新しい場所に、指定のファイルが格納されているディレクトリを移動させる必要があります。

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および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・モードおよびトランスポータブル表領域モードにおいて、オブジェクト型に対する実行権のないユーザーが Export 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限りません。この場合、インポートのターゲット・システムに存在する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によって、型の存在を検証するために必要な情報しか書き込まれません。

DBA に協力を求めて作成するか、全データベース・モードまたは DBA 権限でのユーザー・モードでエクスポートを実行して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ために、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。または、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、適切なユーザー・グループに関して DBA がユーザー・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、SCOTT 所有の表 1 に BLAKE のオブジェクト型である型 1 が存在する場合、この表に必要な型定義を保持するためには、DBA はユーザー・モードで BLAKE と SCOTT の両方を指定してエクスポートを実行します。

ネストした表のエクスポート

ネストした表については、外側の表がエクスポートされる場合は必ず、内側の表データもエクスポートされます。内側の複数の表を指定することはできますが、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

キューは表にインプリメントされています。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ので、キュー表がインポートされたときには、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。

詳細は、『Oracle8*i* 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。

トランスポータブル表領域

トランスポータブル表領域機能は、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、表領域を読取り専用にし、表領域のデータ・ファイルをコピーしてから、Export および Import 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移送は、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 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。トランスポータブル表領域のメタデータを含むエクスポート・ファイルからのインポートの詳細は、2-63 ページ「トランスポータブル表領域」を参照してください。

次のパラメータ・キーワードで、トランスポータブル表領域メタデータのエクスポートを使用可能にできます。

- TRANSPORT_TABLESPACE
- TABLESPACES

詳細は、「TRANSPORT_TABLESPACE」および 1-26 ページ「TABLESPACES」を参照してください。

追加情報 : トランスポータブル表領域の管理の詳細は、『Oracle8*i* 管理者ガイド』を参照してください。トランスポータブル表領域機能の詳細は、『Oracle8*i* 概要』を参照してください。

バージョンの異なる Export ユーティリティの使用方法

この項では、Oracle8*i* とは異なるエクスポート・バージョンを実行する際の一般的な動作と制限について説明します。

下位バージョンの Export ユーティリティの使用

一般に、Oracle7 のどのリリース・レベルでも、Export ユーティリティを使用して Oracle8*i* Server からのエクスポートを実行して、Oracle のリリース 7 のエクスポート・ファイルを作成できます。（この手順については、1-60 ページ「Oracle8*i* データベースからの Oracle リリース 7 のエクスポート・ファイルの作成」を参照してください。）

Oracle6 以前のバージョンの Export ユーティリティは、Oracle8*i* のデータベースに対して使用できません。

下位バージョンの Export ユーティリティを上位バージョンの Oracle Server 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（Oracle7 の Export から除外される Oracle8*i* のオブジェクトのリストは、1-60 ページ「除外されるオブジェクト」を参照してください。）

注意 : 下位互換性の問題があるときには、Oracle8*i* のデータベースに対して以前のバージョンの Export ユーティリティを使用し、従来型パス・エクスポートを使用してください。

注意 : ダイレクト・パスの場合も従来型パスの場合も、Oracle8*i* の Export 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 Import ユーティリティとは互換性がないので、インポートには Oracle8*i* の Import ユーティリティしか使用できません。

上位バージョンの Export ユーティリティの使用

上位バージョンの Export ユーティリティを下位の Oracle Server で使用すると、次のようなエラーが発生することがあります。

```
EXP-37: Database export views not compatible with Export utility
EXP-0:  Export terminated unsuccessfully
```

エラーは、上位レベル・バージョンの Export ユーティリティが要求したビューが存在しないために発生します。このエラーを防ぐため、Oracle サーバーと一致するバージョンの Export ユーティリティを使用してください。

Oracle8*i* データベースからの Oracle リリース 8.0 のエクスポート・ファイルの作成

Oracle8*i* データベースから Oracle リリース 8.0 のエクスポート・ファイルを作成する場合は、特別な手順は必要ありませんが、一部の機能がサポートされません。

- Oracle8*i* データベース上でリリース 8.0 の Export を使用し、ダイレクト・パス・ロード（DIRECT=Y）を指定する場合、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。
- Oracle8*i* データベースで、リリース 8.0 の Export を使用する場合、ディメンションはエクスポートされません。
- Oracle8*i* データベース上で Export のリリース 8.0 を使用する場合、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。
- Oracle8*i* データベース上で Export のリリース 8.0 を使用する場合、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。
- Oracle8*i* データベース上で Export のリリース 8.0 を使用する場合、ビュー、プロシージャ、ファンクション、パッケージ、タイプ本体および新しいリリース 8.1 の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。
- Oracle8*i*（またはそれ以前の）データベース上で Export のリリース 8.0 を使用する場合、DDL が、SQL ではなくストアド・プロシージャとしてインプリメントされているオブジェクトは、エクスポートされません。
- Oracle8*i* データベース上で Export のリリース 8.0 を使用する場合、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。
- Oracle8*i* データベース上で Export のリリース 8.0 を使用する場合、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。
- Oracle8*i* データベース上で Export のリリース 8.0 を使用する場合、一時表はエクスポートされません。
- Oracle8*i* データベース上で Export のリリース 8.0 を使用する場合、索引構成表（IOT）は圧縮前の状態に戻ります。
- Oracle8*i* データベース上で Export のリリース 8.0 を使用する場合、パーティション化された IOT は、パーティション情報が消失します。
- Oracle8*i* データベース上で Export のリリース 8.0 を使用する場合、索引タイプおよび演算子は、エクスポートされません。
- Oracle8*i* データベース上で Export のリリース 8.0 を使用する場合、ローカル管理表領域および一時表領域は、エクスポートされません。

- Oracle8*i* データベース上で Export のリリース 8.0 を使用する場合、Java のソース、クラスおよびリソースはエクスポートされません。
- Oracle8*i* データベース上で Export のリリース 8.0 を使用する場合、VARRAY 列の、可変幅 CLOB、コレクション拡張および LOB-storage_clause、またはネストした表の拡張はエクスポートされません。
- Oracle8*i* データベース上で Export のリリース 8.0 を使用する場合、ファイン・グレイン・アクセス・セキュリティ・ポリシーは、保持されません。

Oracle8*i* データベースからの Oracle リリース 7 のエクスポート・ファイルの作成

Oracle8*i* のデータベースから Oracle のエクスポート・ファイルを作成するには、Oracle7 の Export ユーティリティを Oracle8*i* Server に対して実行します。このためには、まずユーザー SYS で CATEXP7.SQL スクリプトを実行して、Export ユーティリティがデータベースを Oracle7 のデータベースと認識するようなエクスポート・ビューを作成する必要があります。

注意 : Oracle8*i* の Export ユーティリティでは、Export の実行前にデータベースに対して CATEXP.SQL スクリプトを実行する必要があります。必要なビューを作成するためにユーザー SYS で CATALOG.SQL を実行すると、自動的に CATEXP.SQL が実行されます。ただし、CATEXP7.SQL は自動的には実行されないので、手動で実行する必要があります。CATEXP7.SQL と CATEXP.SQL はどちらを先に実行してもよく、これらのスクリプトは一度実行すると、再実行する必要はありません。

除外されるオブジェクト

Oracle7 の Export ユーティリティは、CATEXP7.SQL によって作成されたビューに対して問合せを発行することにより、Oracle7 のエクスポート・ファイルを生成します。これらのビューは、Oracle リリース 7 と完全に互換性があるので、1-59 ページ「Oracle8*i* データベースからの Oracle リリース 8.0 のエクスポート・ファイルの作成」にリストされている新しい Oracle8*i* オブジェクト、または次に示す Oracle8 オブジェクトは含まれていません。

- ディレクトリ別名
- 外部関数ライブラリ
- オブジェクト型
- Oracle8 の新規オブジェクト（LOB 列および REF 列、BFILE 列、ネストした表など）が含まれている表
- パーティション表
- 索引編成表（IOT）
- 列の数が 254 列を超える表

- NCHAR 列が含まれている表
- 2,000 文字を超える長さの VARCHAR 列が含まれている表
- 逆索引
- パスワード履歴
- システムおよびスキーマのイベント・トリガー
- 一般的な ROWID 列のある表
- ビットマップ索引

Enterprise Manager および Oracle7 の Export ユーティリティ 7.3.2 のデータベースのエクスポートに Enterprise Manager を使用するには、Enterprise Manager のバージョン 1.4.0 以降が必要です。

2

Import

この章では、Import 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。

Import ユーティリティは、Export ユーティリティによって作成されたエクスポート・ファイルのみを読み込むことができます。データベースのエクスポート方法の詳細は、第 1 章「Export」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、このマニュアルの第 II 部「SQL*Loader」を参照してください。

この章では、次のトピックについて説明します。

- Import ユーティリティとは
- インポート・モード
- Import ユーティリティの使用方法
- オブジェクトの Import に必要な権限
- 既存の表へのインポート
- インポート・パラメータ
- 表レベルおよびパーティション・レベルの Export と Import の使用
- インポート・セッションの例
- 対話方式の使用
- 増分、累積および全エクスポート・ファイルのインポート
- 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
- 警告、エラーおよび完了メッセージ
- エラーの処理
- ネットワークに関する考慮事項

Import ユーティリティとは

Import の基本的な考え方は非常に単純です。Import とは、ある Oracle データベースから Export ユーティリティによって抽出された（そして、エクスポート・ダンプ・ファイルに書き込まれた）データ・オブジェクトを別の Oracle データベースに挿入することです。エクスポート・ダンプ・ファイルを読み込めるのは、Import ユーティリティを使用した場合のみです。Oracle の Export ユーティリティの詳細は、第 1 章「Export」を参照してください。

Import を実行すると、Export ユーティリティによって Oracle データベースから抽出され、通常、ディスクまたはテープにあるバイナリ形式の Oracle エクスポート・ダンプ・ファイルに書き込まれた、オブジェクト定義および表データが読み込まれます。

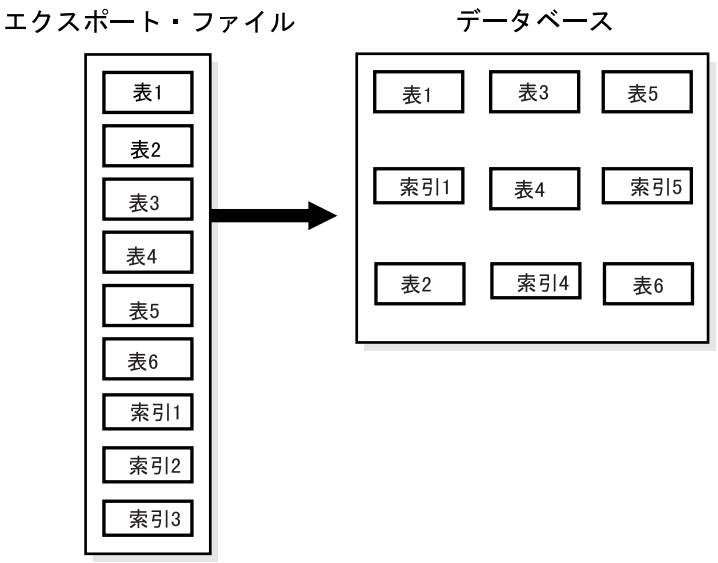
このファイルは通常、別サイトにファイル転送、または物理的に移送（テープの場合）され、Import ユーティリティにより、ネットワーク接続していないマシン上のデータベース間でのデータ転送や、標準のバックアップ手順以外のバックアップとして使用されます。

> **注意 :** エクスポート・ダンプ・ファイルが読み込めるのは、Oracle ユーティリティの Import を使用した場合のみです。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場合は、このマニュアルの第 II 部「SQL*Loader」を参照してください。

また、Export および Import ユーティリティは、オフライン・インスタンシエーションなど、Oracle アドバンスト・レプリケーションの機能面でも役立ちます。詳細は、『Oracle8*i* レプリケーション・ガイド』を参照してください。

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



新機能

Oracle の今回のリリースでは、次のインポート機能が追加されました。

- サブパーティションのインポート。2-33 ページ「表レベルおよびパーティション・レベルの Export と Import の使用」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。2-22 ページ「FILE」および 2-22 ページ「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レーションによって作成された型である場合）の妥当性チェックを省略。2-30 ページ「TOID_NOVALIDATE」を参照してください。
- 各テープ媒体のエクスポート・ファイルに指定できる最大バイト数が増加。2-32 ページ「VOLSIZE」を参照してください。
- ファイン・グレイン・アクセスのサポート。2-52 ページ「ファイン・グレイン・アクセスのサポート」を参照してください。
- エクスポートおよびインポートのオプティマイザ統計情報を、インポート時に再計算するかわりに、事前に計算可能。（この機能は、一部のエクスポートおよび一部の表に対してのみ利用できます。）2-27 ページ「RECALCULATE_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。2-31 ページ「TRANSPORT_TABLESPACE」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（あるいはその両方）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）起動することもなくなります。

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れば、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。上記のような状況では、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に、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 であり、DRAKE の行がまだロードされていない場合、SCOTT の行はロードされません（インポートが終了した場合には有効になるとしても）。

> **提案 :** 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。インポートが完了してから再び制約を使用可能にすることができます。

互換性

Import ユーティリティは、バージョン 5.1.22 以降の Export で作成されたエクスポート・ファイルを読み込むことができます。

インポート・モード

Import には 4 種類のモードがあります。インポートされるオブジェクトは、選択したインポート・モードと、使用したエクスポートのモードによって決まります。すべてのユーザーが選択できるインポート・モードは 2 つあります。IMP_FULL_DATABASE ロールを持つユーザー（特権ユーザー）は 4 つのモードから選択できます。

表モード	このモードでは、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。
ユーザー・モード	このモードでは、所有するすべてのオブジェクト（表、権限、索引およびプロシージャなど）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。
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、このモードを使用して全データベース・エクスポート・ダンプ・ファイルをインポートできます。
トランスポータブル表領域モード	このモードでは、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。

各モードの指定の詳細は、2-16 ページ「インポート・パラメータ」を参照してください。

IMP_FULL_DATABASE ロールを持つユーザーはこれらのモードのどれか、または増分インポートを指定する必要があります。指定がないと、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オプションをいずれも指定しない場合、ユーザー・レベルのインポートが実行されます。

1-5 ページ表 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。

表レベル・インポートとパーティション・レベル・インポートの違い

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・ファイル内のすべてのデータをインポートする。
- **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インポートする。

既存の表にデータをロードするときは、パラメータ IGNORE=Y と指定します。詳細は、2-24 ページ「IGNORE」を参照してください。

表レベル・インポート

表レベル・インポートでは、指定したそれぞれの表に関して表の全行がインポートされます。表レベル・インポートの特徴は次のとおりです。

- どのエクスポート・モード（全、ユーザー、表）でエクスポートされた場合でも、表レベル・インポートを使用すれば、エクスポートされたすべての表をインポートできる。
- 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）にインポートできる。

表が存在しない場合に、エクスポートされた表がパーティション表であると、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

パーティション・レベル・インポートでは、パーティションまたはサブパーティションのグループがソース表からターゲット表にインポートされます。次の点に注意してください。

- Import 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る。
- 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る。
- 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る。
- 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も上に入る行は、すべて拒否される。
- 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。

詳細は、2-33 ページ「表レベルおよびパーティション・レベルの Export と Import の使用」を参照してください。

Import ユーティリティの使用方法

この項では、インポートの事前準備や Import ユーティリティの起動方法および使用方法を説明します。

Import ユーティリティを使用する前に

Import ユーティリティを使用するには、データベースを作成、またはリリース 8.1 に移行した後で、スクリプト CATEXP.SQL または CATALOG.SQL（CATEXP.SQL を実行します）を実行する必要があります。

> **追加情報 :** スクリプト・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。スクリプト・ファイルの名前およびそれらを実行する方法の詳細は、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。

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。
CATEXP.SQL または CATALOG.SQL によって次の処理が行われ、データベースは Import に備えて調整されます。

- IMP_FULL_DATABASE ロールに、必要なすべての権限を割り当てる。
- IMP_FULL_DATABASE を DBA ロールに割り当てる。
- データ・ディクショナリに必要なビューを作成する。

Import ユーティリティの起動

Import ユーティリティは、次の 3 通りの方法で起動できます。

- 次のコマンドを入力する。

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

  PARFILE は、通常使用するインポート・パラメータが入っているファイルです。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。パラメータ・ファイルの使用については、2-10 ページ「パラメータ・ファイル」を参照してください。

- 次のコマンドを入力する。

  ```
  imp username/password <parameters>
  ```

  <parameters> には使用予定の各種パラメータを指定します。ご使用のシステムのコマンド行の最大長を超える数のパラメータは指定できないことに注意してください。

- 次のコマンドを入力する。

  ```
  imp username/password
  ```

  対話型セッションが開始されます。必要な情報の入力は Import 時に要求されます。対話方式では、パラメータ指定方式ほどの機能性は提供されません。対話方式は、下位互換性のために用意されています。

最初のオプションと 2 番目のオプションは組み合せて使用できます。つまり、パラメータ・ファイルとコマンド行の両方にパラメータを指定できます。実際には、パラメータ・ファイルとコマンド行の両方に同じパラメータを指定することもできます。コマンド行での PARFILE パラメータと他のパラメータの位置によって、どのパラメータが優先されるかが決まります。たとえば、パラメータ・ファイル `params.dat` に、パラメータ INDEXES=Y が指定されていると、Import ユーティリティは次のコマンド行によって起動されます。

```
imp system/manager PARFILE=params.dat INDEXES=N
```

この場合、`INDEXES=N` は `PARFILE=params.dat` の後にあるので、PARFILE に指定されている INDEXES パラメータは `INDEXES=N` によって上書きされます。

ユーザー名とパスワードは、パラメータ・ファイルでも指定できますが、セキュリティ上の理由のため、この方法は使用しないでください。

ユーザー名とパスワードを省略すると、入力が要求されます。

各パラメータの説明は、2-16 ページ「インポート・パラメータ」を参照してください。

SYSDBA としての Import ユーティリティの起動

通常は、Import を SYSDBA として起動する必要はありません。ただし、オラクル社カスタマ・サポートの要求によって、SYSDBA として起動する場合もあります。

Import を SYSDBA として起動するには、次の構文を使用します。

```
imp username/password AS SYSDBA
```

または、任意で次の構文を使用します。

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

注意 : 文字列「AS SYSDBA」にはブランクが含まれるため、ほとんどのオペレーティング・システムでは、'username/password AS SYSDBA' のように文字列全体を引用符で囲むか、何らかの方法でリテラルとしてマーク設定することが必要です。オペレーティング・システムによっては、コマンド行に含まれる引用符も同様に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 Oracle ドキュメントを参照してください。

ユーザー名またはパスワードの指定がないと、入力するよう要求されます。

Import の対話形式モードを使用する場合は、SYSDBA として接続するか、@instance として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、ユーザー名として入力する必要があります。

オンライン・ヘルプの利用

Import 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。コマンド行に `imp help=y` と入力すると、次のようなヘルプ画面が表示されます。

```
> imp help=y

Import: Release 8.1.5.0.0 - Production on Wed Oct 28 15:00:44 1998

(c) Copyright 1998 Oracle Corporation.  All rights reserved.


You can let Import prompt you for parameters by entering the IMP
command followed by your username/password:

     Example: IMP SCOTT/TIGER

Or, you can control how Import runs by entering the IMP command followed
by various arguments. To specify parameters, you use keywords:

     Format:  IMP KEYWORD=value or KEYWORD=(value1,value2,...,valueN)
     Example: IMP SCOTT/TIGER IGNORE=Y TABLES=(EMP,DEPT) FULL=N
               or TABLES=(T1:P1,T1:P2), if T1 is partitioned table

USERID must be the first parameter on the command line.

Keyword  Description (Default)       Keyword      Description (Default)
------------------------------------------------------------------------
USERID   username/password           FULL         import entire file (N)
BUFFER   size of data buffer         FROMUSER     list of owner usernames
FILE     input files (EXPDAT.DMP)    TOUSER       list of usernames
SHOW     just list file contents (N) TABLES       list of table names
IGNORE   ignore create errors (N)    RECORDLENGTH length of IO record
GRANTS   import grants (Y)           INCTYPE      incremental import type
INDEXES  import indexes (Y)          COMMIT       commit array insert (N)
ROWS     import data rows (Y)        PARFILE      parameter filename
LOG      log file of screen output   CONSTRAINTS  import constraints (Y)
DESTROY  overwrite tablespace data file (N)
INDEXFILE write table/index info to specified file
SKIP_UNUSABLE_INDEXES  skip maintenance of unusable indexes (N)
ANALYZE  execute ANALYZE statements in dump file (Y)
```

```
FEEDBACK display progress every x rows(0)
TOID_NOVALIDATE  skip validation of specified type ids
FILESIZE maximum size of each dump file
RECALCULATE_STATISTICS recalculate statistics (N)
VOLSIZE  number of bytes in file on each volume of a file on tape

The following keywords only apply to transportable tablespaces
TRANSPORT_TABLESPACE import transportable tablespace metadata (N)
TABLESPACES tablespaces to be transported into database
DATAFILES datafiles to be transported into database
TTS_OWNERS users that own data in the transportable tablespace set

Import terminated successfully without warnings.
```

パラメータ・ファイル

パラメータ・ファイルの中にインポート・パラメータを指定しておくと、パラメータを容易に修正および再利用できます。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・ファイルを作成してください。コマンド行オプション PARFILE=<*filename*> は、コマンド行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むように Import ユーティリティに通知します。たとえば、次のように指定します。

```
imp parfile=filename
```

または

```
imp username/password parfile=filename
```

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。

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

シャープ（#）記号を使用してパラメータ・ファイルにコメントを追加できます。# の右側にある文字はすべて無視されます。パラメータ・ファイル内のリストの一部の例を次に示します。

```
FULL=y
FILE=DBA.DMP
GRANTS=Y
INDEXES=Y # import all indexes
```

それぞれのパラメータの説明は、2-16 ページ「インポート・パラメータ」を参照してください。

オブジェクトの Import に必要な権限

この項では、Import 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。

アクセス権限

Import ユーティリティを使用するには、Oracle Server にログインするための権限 CREATE SESSION 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 ロールに属しています。

他のユーザーが作成したエクスポート・ファイルもインポートできます。ただし、他のユーザーが作成したエクスポート・ファイルの場合は、IMP_FULL_DATABASE ロールを持っている場合のみインポートできます。

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。これらの権限すべては、初期には RESOURCE ロールに属しています。

表 2-1 自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	権限		権限タイプ
クラスタ		CREATE CLUSTER	システム
	および :	表領域割当て制限、または	
		UNLIMITED TABLESPACE	システム
データベース・リンク		CREATE DATABASE LINK	システム
	および :	リモート DB の場合 CREATE SESSION	システム
表のトリガー		CREATE TRIGGER	システム
スキーマのトリガー		CREATE ANY TRIGGER	システム
索引		CREATE INDEX	システム
	および :	表領域割当て制限、または	
		UNLIMITED TABLESPACE	システム
整合性制約		ALTER TABLE	オブジェクト

表 2-1 自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	権限		権限タイプ
ライブラリ		CREATE ANY LIBRARY	システム
パッケージ		CREATE PROCEDURE	システム
プライベート・シノニム		CREATE SYNONYM	システム
順序		CREATE SEQUENCE	システム
スナップショット		CREATE SNAPSHOT	システム
ストアド・ファンクション		CREATE PROCEDURE	システム
ストアド・プロシージャ		CREATE PROCEDURE	システム
表データ		INSERT TABLE	オブジェクト
表定義		CREATE TABLE	システム
（コメントおよび監査オプションを含む）	および :	表領域割当て制限、または UNLIMITED TABLESPACE	システム
ビュー		CREATE VIEW	システム
	および :	実表の場合 SELECT、または	オブジェクト
		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 の付いたオブジェクト権限を持っ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、またはユーザーが WITH GRANT OPTION を伴ったオブジェクト権限を持っている必要がある。
システム権限	ユーザーは WITH ADMIN OPTION およびシステム権限を持っている必要がある。

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いてください。

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しておいてください。エクスポート・ファイルが全エクスポートのときにパラメータ FULL を指定すると、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

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 コマンドによって作成されます。したがって、旧バージョンの Export ユーティリティ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。

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際の考慮事項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、列幅を増やしたり列の順序を変更することはできますが、次のことはできません。

- NOT NULL 列の追加
- 列のデータ型の、互換性のないデータ型への変更（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

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。

ただし、データが既存の表にロードされる場合には、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、失敗した行を飛ばして、引き続き他の行が表にインポートされます。制約を手作業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、エクスポート・ファイル内で EMP 表の順序が DEPT 表より先であるときに EMP 表が DEPT 表を参照チェックしていると、参照制約違反のために 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。

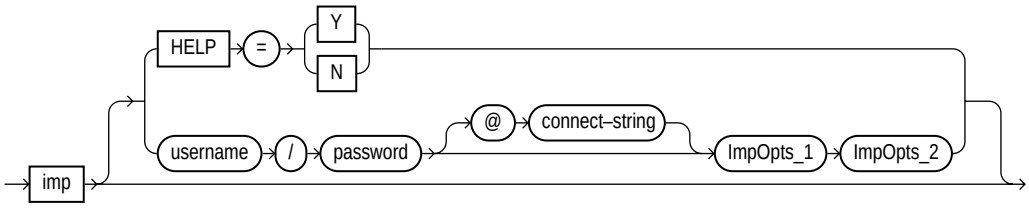
インポートを手動で順序付ける方法

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。表のチェックに要す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がよいこともあります。

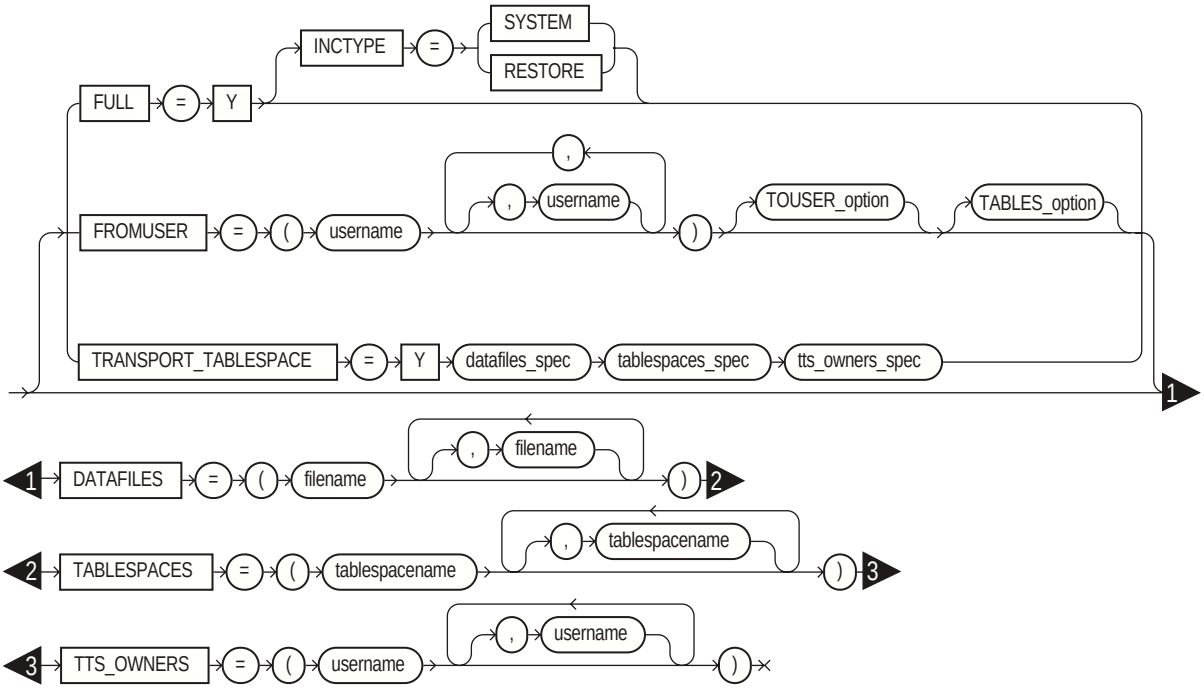
そのためには、エクスポート・ファイルからのインポートを 1 回ではなく複数回に分けて実行します。まず参照チェックのターゲットである表をインポートし、次にこれらの表を参照する表をインポートします。表が循環的に相互参照している場合と、表がその表自体を参照している場合を除き、この方法は有効です。

インポート・パラメータ

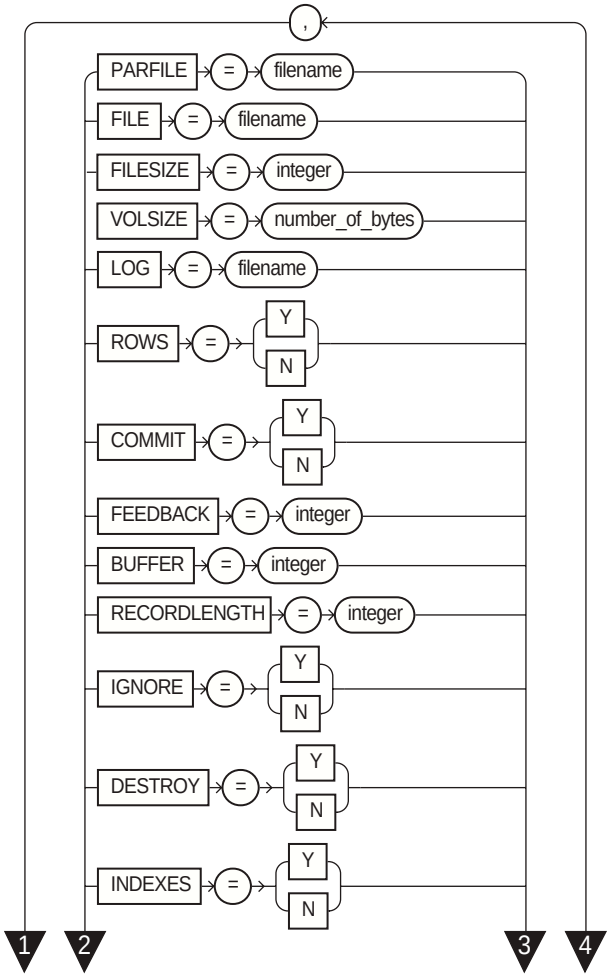
パラメータ・ファイルまたはコマンド行で指定できるパラメータの構文を次のダイアグラムに示します。



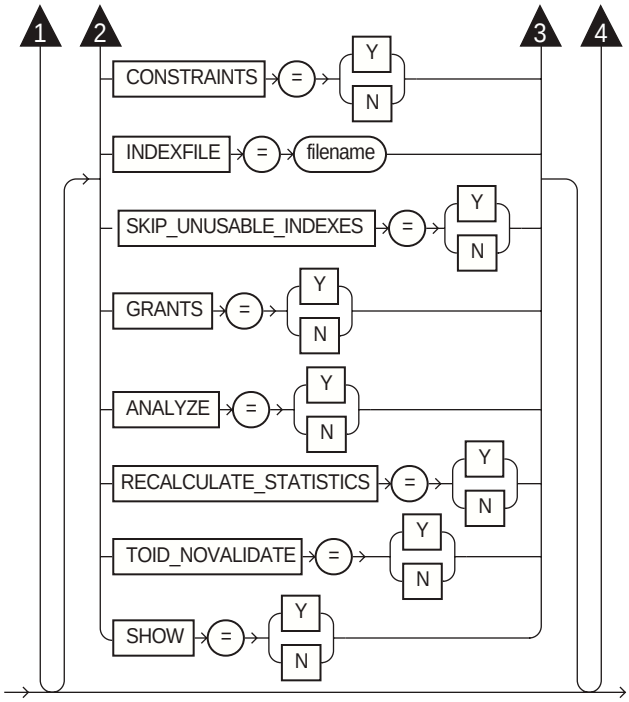
impopts_1



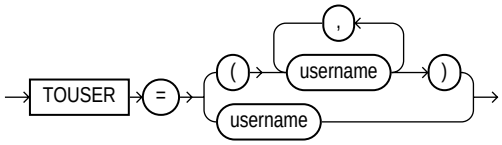
impopts_2


,
PARFILE
=
filename
FILE
=
filename
FILESIZE
=
integer
VOLSIZE
=
number_of_bytes
LOG
=
filename
ROWS
=
Y
N
COMMIT
=
Y
N
FEEDBACK
=
integer
BUFFER
=
integer
RECORDLENGTH
=
integer
IGNORE
=
Y
N
DESTROY
=
Y
N
INDEXES
=
Y
N
1
2
3
4

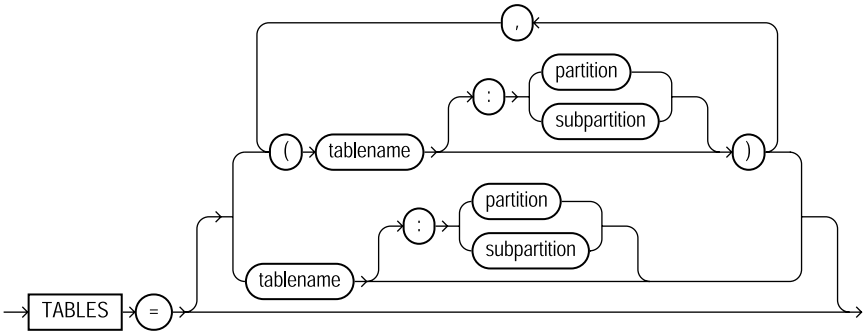
impopts_2（続き）


1
2
3
4
CONSTRAINTS
=
Y
N
INDEXFILE
=
filename
SKIP_UNUSABLE_INDEXES
=
Y
N
GRANTS
=
Y
N
ANALYZE
=
Y
N
RECALCULATE_STATISTICS
=
Y
N
TOID_NOVALIDATE
=
Y
N
SHOW
=
Y
N

TOUSER_option



TABLES_option



次の項では、パラメータの機能およびデフォルト値を説明します。

ANALYZE

デフォルト : Y

Import ユーティリティによってエクスポート・ファイル中の SQL ANALYZE 文が実行されるかどうか、またはエクスポート・システムで計算済みの表、索引および列のオプティマイザ統計がロードされるかどうかを指定します。Import のパラメータの詳細は、2-27 ページ「RECALCULATE_STATISTICS」および 2-63 ページ「統計情報のインポート」を参照してください。

BUFFER

デフォルト : オペレーティング・システムに依存

buffer-size は、取り出したデータ行を格納するバッファのバイト数です。

パラメータ BUFFER（バッファ・サイズ）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。所定の行配列を挿入するために必要なバッファ・サイズは、次のように計算できます。

```
buffer_size = rows_in_array * maximum_row_size
```

LONG、LOB、BFILE、REF、ROWID、DATE 型の列が含まれている表は 1 度に 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ときは、Import ユーティリティはさらにサイズの大きいバッファを割り当てようとします。

追加情報 : このパラメータのデフォルト値の決定については、ご使用のオペレーティング・システム固有の Oracle ドキュメントを参照してください。

CHARSET

注意 : このパラメータは Oracle バージョン 5 および 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方法の詳細は、2-66 ページ「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、外側の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 と指定されていると、それぞれのパーティションおよびサブパーティションに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がなくなり、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 と指定してください。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、致命的でないエラーとして通知されます。

表に一意制約がない場合、再度インポートを実行したときに、すでにインポート済み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 型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。

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。

DATAFILES

デフォルト : なし

TRANSPORT_TABLESPACE に Y を指定した場合、データベースに移送するファイルを、このパラメータを使用してリスト表示します。

詳細は、2-63 ページ「トランスポータブル表領域」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。つまり、DESTROY=Y を指定すると、CREATE TABLESPACE コマンドの datafile 句に REUSE オプションを付け、これによって、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）同一マシン上に 2 番目のデータベースを作成しようとすると、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。このような場合は、デフォルトの DESTROY=N を指定してください。そうすれば、表領域作成時にすでにデータ・ファイルがある場合、エラーになります。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するようにします。

警告 : データ・ファイルがロー・デバイスに格納されている場合は、DESTROY=N と指定しても、ファイルが上書きされます。

FEEDBACK

デフォルト : 0（ゼロ）

n 行分のインポートを 1 つのドットで示す進捗メーターの表示を指定します。たとえば、FEEDBACK=10 と指定すると、10 行分のインポートが終了するたびにドットが 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。Export 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（次のパラメータ FILESIZE 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。

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取り権限が必要です。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

Export は複数のエクスポート・ファイルへの書込みをサポートしており、Import では複数のエクスポート・ファイルから読取りができます。エクスポートで、エクスポートの FILESIZE パラメータの値（バイト制限）を指定した場合は、それぞれのダンプ・ファイルに指定したバイト数が書き込まれます。インポートでは、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、インポートの 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。

FILESIZE の値は、数字に K（キロバイトの数）をつけて指定できます。たとえば、FILESIZE=2K は、FILESIZE=2048 と同じです。同様に、M はメガバイト（1024 × 1024）を、G はギガバイト（1024^3）を表します。B はバイトの省略です。この場合、本来のファイル・サイズの算出に乗算は不要です（FILESIZE=2048b は、FILESIZE=2048 と同じです）。

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）からスキーマのサブセットがインポートできます。

通常は、インポート・パラメータ TOUSER と FROMUSER を組み合せて使用し、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（2-31 ページ「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・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、インポートするユーザーのスキーマにオブジェクトを作成します（インポートの FROMUSER スキーマに存在するかどうかにかかわらず）。

  注意 : FROMUSER=SYSTEM と指定しても、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 オプションを使用して、権限が与えら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 と指定すると、オブジェクト権限はインポートされません。（GRANTS=N と指定しても、システム権限はインポートされます。）

注意 : Export ユーティリティでは、セキュリティ上の理由から、Import に影響が及ばないよう、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、アクセス権が変更され、インポートしたユーザーがそれに気付かない場合があります。

HELP

デフォルト : N

インポート・パラメータの説明を表示します。

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。IGNORE=Y と指定すると、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ます。IGNORE=Y を指定すると、インポートは、エラーを通知せずに続行されます。IGNORE=Y を指定しても既存のオブジェクトは置換されず、スキップされます。

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されるか、または表示され（あるいはその両方が実行され）、インポートは続行します。

表については、IGNORE=Y を指定すると、行が既存の表にインポートされます。このとき、メッセージは表示されません。表がすでに存在する場合、IGNORE=N が指定されていれば、エラーがレポートされ、表には行が挿入されずにスキップされます。また、表に依存するオブジェクトは（索引、権限、制約など）、表がすでに存在する場合や、IGNORE=N の場合は作成されません。

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、一部のオブジェクトは何回も作成されます（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、この問題を防ぐことができます。CONSTRAINTS パラメータを NO に設定して全インポートを実行すると、表の制約は一切インポートされません。

既存の表にデータをインポートする場合もあります。たとえば、新たな記憶領域パラメータを使用する場合や、クラスタ上にあらかじめ表を作成している場合などです）。このようなときには、IGNORE=Y と指定すれば、既存の表にデータ行がインポートされます。

警告 : 既存の表にインポートするときに、表の列の索引が一意でないと、すでに表に存在する行までもがインポートされるので、行データが重複することがあります。（この警告は、増分インポート以外にのみ適用されます。増分インポートでは、最後の全エクスポートによって表を置き換え、一連の累積エクスポートおよび増分エクスポートによって最後にバックアップを作成したときの状態に表を再構築します）。

INCTYPE

デフォルト : 定義なし

増分インポートのタイプを指定します。

メニュー項目は次のとおりです。

SYSTEM	最新バージョンのシステム・オブジェクトをインポートします。このオプションを使用するときは、最新の増分エクスポートを指定してください。SYSTEM インポートでは、外部関数ライブラリやオブジェクト型定義などのシステム・オブジェクトはインポートされますが、ユーザー・データやオブジェクトはインポートされません。
RESTORE	エクスポート・ファイル内の、システム・オブジェクト以外のユーザーのデータベース・オブジェクトおよびデータを、すべてインポートします。

INCTYPE パラメータの詳細は、2-43 ページ「増分、累積および全エクスポート・ファイルのインポート」を参照してください。

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 索引または 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、Import ユーティリティによって自動的に再作成されます。

INDEXES=N と指定すれば、すべてのユーザー作成索引を Import 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

索引作成コマンド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コマンドは、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（記憶領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容易に識別するために、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、次のステップを実行します。

1. INDEXFILE パラメータを使用してインポートを行い、索引作成コマンド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 文字列に必ず追加します。
3. INDEXES=N を指定して Import を再実行します。

＜このステップではデータベース・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. SQL スクリプトとして索引作成コマンドのファイルを実行し、索引を作成します。

INDEXFILE パラメータを指定できるのは、FULL=Y または FROMUSER、TOUSER、TABLES パラメータを指定したときのみです。

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、端末画面とログ・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 : 定義なし

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・ファイルの詳細は、2-10 ページ「パラメータ・ファイル」を参照してください。

RECALCULATE_STATISTICS

デフォルト : N

このパラメータに Y を指定すると、エクスポート・データがインポートされたとき、データベース・オプティマイザ統計が生成されます。オプティマイザおよびオプティマイザが使用する統計の詳細は、『Oracle8*i* 概要』を参照してください。また、エクスポート・パラメータの詳細は、1-23 ページ「STATISTICS」を、インポート・パラメータの詳細は、2-19 ページ「ANALYZE」および 2-63 ページ「統計情報のインポート」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と、ご使用のプラットフォーム固有の、BUFSIZ に関する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 の値と同等またはより大きい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。

注意 : Import の I/O バッファのサイズ指定に、このパラメータを使用できます。

追加情報 : 適切な値の決定や他のレコード・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 ドキュメントを参照してください。

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

このパラメータを指定すると、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、インポートがその文を実行す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 または FROMUSER、TOUSER、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によって）索引使用禁止に設定されている索引またはパーティションのキーは作成されません。詳細は、『Oracle8*i* SQL リファレンス』の「ALTER SESSION SET SKIP_UNUSABLE_INDEXES=TRUE」を参照してください。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、行が挿入されるとき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。

INDEXES = N と指定して INDEXFILE パラメータを使用すると、索引を再作成する SQL スクリプトができます。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がメンテナンスされる場合に、その行挿入によってエラーになります。

TABLES

デフォルト : なし

インポートの対象とする表名のリストを指定します。すべての表を指定するには、アスタリスク（*）を使用します。このパラメータを指定すると、表モード・インポートが開始されます。表モード・インポートでは、インポート対象は、表とその関連オブジェクトに限定されます（1-5 ページ表 1-1 を参照）。1 度に指定できる表の数は、コマンド行の制限によって決まります。

エクスポート時には表名をスキーマ名（SCOTT.EMP のような）で修飾できますが、インポート時にはできません。次の例では、TABLES パラメータは間違って指定されています。

```
imp system/manager 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/manager FROMUSER=jones TABLES=(accts)
imp system/manager FROMUSER=scott TABLES=(emp,dept)
```

追加情報 : 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には、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

コマンド行またはパラメータ・ファイルで指定する表名の中にシャープ記号（#）を使用する場合は、表名を引用符で囲む必要があります。

たとえば、パラメータ・ファイルに次の行が含まれていると、EMP# の右側はすべてコメントと解釈されます。その結果 DEPT および MYDATA はインポートされません。

```
TABLES=(EMP#, DEPT, MYDATA)
```

一方、パラメータ・ファイルに次の行が含まれている場合には、3 つの表がすべてインポートされます。

```
TABLES=("EMP#", DEPT, MYDATA)
```

注意 : 表名を引用符で囲むと、大文字と小文字を区別して認識されます。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。デフォルトでは、データベース名は大文字で格納されます。

追加情報 : オペレーティング・システムによっては、二重引用符ではなく、一重引用符を使用するものがあります。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。

TABLESPACES

デフォルト : なし

TRANSPORT_TABLESPACE に Y を指定した場合、データベースに移送する表領域を、このパラメータを使用してリスト表示します。

詳細は、2-63 ページ「トランスポータブル表領域」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をインポートしようとしているが、データベースにはその名前の型がすでに存在している場合、先に存在する型が、実際にその表で使用されているかどうか（実際は異なる型であり、単に同じ名前であるだけではないか）検証されます。

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とが比較され、TOID が一致しない場合は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すべきではない型もあります（たとえば、その型がカートリッジのインストレーション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。

構文は次のとおりです。

```
toid_novalidate=([schema-name.]type-name [, ...])
```

たとえば、次のように指定します。

```
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。

通常のインポートにおいて、除外される型が含まれていた場合、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。たとえば、次のように指定します。

```
imp system/manager 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 のオブジェクトは JOE のスキーマにインポートされ、FRED のオブジェクトは TED のスキーマにインポートされます。

```
imp system/manager FROMUSER=scott,fred TOUSER=joe,ted
```

> **注意 :** 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。余分のオブジェクトが 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/manager FROMUSER=scott,adams TOUSER=ted,ted
> ```
>
> ユーザー Ted は 2 度指定されています。

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スポータブル表領域メタデータがインポートされます。

詳細は、2-63 ページ「トランスポータブル表領域」を参照してください。

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、一連のトランスポータブル表領域のデータの所有者ユーザーをリスト表示できます。

詳細は、2-63 ページ「トランスポータブル表領域」を参照してください。

USERID

デフォルト : 定義なし

インポートを実行するユーザーのユーザー名とパスワード（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 AS SYSDBA
```

詳細は、2-8 ページ「SYSDBA としての Import ユーティリティの起動」を参照してください。オペレーティング・システムによっては、AS SYSDBA が特殊文字列とみなされるため、文字列全体を引用符で囲む必要があります（2-8 ページを参照）。

Net8 については、*@connect_string* 句を任意に指定できます。この句 *@connect_string* の構文の詳細は、ご使用の Net8 プロトコルのユーザーズ・ガイドを参照してください。

VOLSIZE

それぞれのテープ媒体のエクスポート・ファイルについて、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、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。

VOLSIZE の値は、数字に K（キロバイトの数）をつけて指定できます。たとえば、VOLSIZE=2K は、VOLSIZE=2048 と同じです。同様に、M はメガバイト（1024 × 1024）を、G はギガバイト（1024^3）を表します。B はバイトの省略です。この場合、本来のファイル・サイズの算出に乗算は不要です（VOLSIZE=2048b は、VOLSIZE=2048 と同じです）。

表レベルおよびパーティション・レベルの Export と Import の使用

表レベル・エクスポートやパーティション・レベル・エクスポートでは、表、パーティションおよびサブパーティションの間でデータを移行させることができます。

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

この項では、パーティション・レベル・インポートについて詳しく説明します。一般的な情報は、2-5 ページ「表レベル・インポートとパーティション・レベル・インポートの違い」を参照してください。

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れば、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（エクスポート時に表名に指定された表）がパーティション表であり、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、警告が発せられる。
- 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を指定する。エクスポート・ファイルには、エクスポート・ソース・システムの表全体のデータのうち、一部のデータしか含まれていない場合があります。

ROWS=Y（デフォルト）を指定していて、表がインポート先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）を指定していて、対象となる表がインポート前に存在している場合は、指定し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。

Import のターゲット表がパーティション表の場合は、そのターゲット表の最上位パーティションよりも上に入る行はすべて拒否されます。拒否された行はレポートされます。

ROWS=N と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・ファイル中の表と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の処理を続けます。

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 と指定します。

パーティションと表の間のデータ移行

TABLES パラメータに表名 : パーティション名を指定すると、パーティション名に指定されたソース・パーティションまたはサブパーティションのデータ行のみが、エクスポート・ファイルから読み込まれます。パーティション名またはサブパーティション名を指定しないと、表全体がソースとして使用されます。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と、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。

エクスポート・ファイルに存在しないパーティションまたはサブパーティションを指定した場合、警告が発せられます。

1 つ以上のパーティションまたはサブパーティションからエクスポートされたデータは、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できます。行は、ターゲット表のパーティション化基準に基づいて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。

次の例では、パーティション名に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はインポートされます。

```
imp system/manager FILE = export.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 = export.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 を作成後、同じパーティションにデータが挿入されます。Import 前にターゲット・システムに表 e が存在する場合は、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。

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行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・セッション例を示します。

- 管理者が、エクスポート元のスキーマに表をインポートする場合
- ユーザーが、他のスキーマの表を自分のスキーマにインポートする場合
- 管理者が、表をエクスポート元のスキーマとは異なるスキーマにインポートする場合
- パーティション・レベル・インポートで表をインポートする場合

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/manager 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)
```

コマンド行方式

```
> imp system/manager file=dba.dmp fromuser=scott tables=(dept,emp)
```

インポート・メッセージ

```
Import: Release 8.1.5.0.0 - Production on Fri Oct 30 09:41:18 1998

(c) Copyright 1998 Oracle Corporation.  All rights reserved.

Connected to: Oracle8 Enterprise Edition Release 8.1.5.0.0 - Production
With the Partitioning option
PL/SQL Release 8.1.5.0.0 - Production

Export file created by EXPORT:V08.01.05 via conventional path
import done in WE8DEC character set and WE8DEC 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.
```

他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する方法を示します。

パラメータ・ファイル方式

```
> imp system/manager 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)
```

コマンド行方式

```
> imp system/manager fromuser=blake touser=scott file=blake.dmp tables=(unit,manager)
```

インポート・メッセージ

```
Import: Release 8.1.5.0.0 - Production on Fri Oct 30 09:41:34 1998

(c) Copyright 1998 Oracle Corporation.  All rights reserved.

Connected to: Oracle8 Enterprise Edition Release 8.1.5.0.0 - Production
With the Partitioning option
PL/SQL Release 8.1.5.0.0 - Production

Export file created by EXPORT:V08.01.05 via conventional path

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

import done in WE8DEC character set and WE8DEC 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/manager 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=(*)
```

コマンド行方式

```
> imp system/manager file=scott.dmp fromuser=scott touser=blake tables=(*)
```

インポート・メッセージ

```
Import: Release 8.1.5.0.0 - Production on Fri Oct 30 09:41:36 1998

(c) Copyright 1998 Oracle Corporation.  All rights reserved.

Connected to: Oracle8 Enterprise Edition Release 8.1.5.0.0 - Production
With the Partitioning option
PL/SQL Release 8.1.5.0.0 - Production

Export file created by EXPORT:V08.01.05 via conventional path

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

import done in WE8DEC character set and WE8DEC NCHAR character set
. . 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、パーティションおよび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 からな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

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・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、表 emp の p1 および p3 を指定しています。

```
> imp scott/tiger tables=(emp:p1,emp:p3) file=exmpexp.dat rows=y

Export file created by EXPORT:V08.01.05 via direct path
import done in WE8DEC character set and WE8DEC 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る例を示します。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、p1_sp3 があり、p2 には 2 つのサブパーティション p2_sp1 と 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

About to export specified tables via Conventional Path --
. . exporting table                        EMP
. . exporting partition                         P1
```

```
. . exporting subpartition                    P1_SP1         11 rows exported
. . exporting subpartition                    P1_SP2         17 rows exported
. . exporting subpartition                    P1_SP3          3 rows exported
. . exporting partition                       P2
. . exporting subpartition                    P2_SP1          5 rows exported
. . exporting subpartition                    P2_SP2         12 rows exported

Export terminated successfully without warnings.
```

次のインポート・コマンドでは、表 emp 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、表 emp 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

Export file created by EXPORT:V08.01.05 via conventional path
import done in WE8DEC character set and WE8DEC NCHAR character set
. importing SCOTT's objects into SCOTT
. . importing table              EMP
. . importing subpartition       "EMP":"P1_SP2"              17 rows imported
. . importing subpartition       "EMP":"P1_SP3"               3 rows imported
. . importing subpartition       "EMP":"P2_SP1"               5 rows imported
. . importing subpartition       "EMP":"P2_SP2"              12 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ると仮定します。EMP 表を DEPTNO 列で再パーティション化します。

表の別の列で再パーティション化するには、次のステップ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。

次の例に、表を別の列で再パーティション化する方法を示します。

```
> exp scott/tiger tables=emp file=empexp.dat
```

```
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> drop table emp cascade constraints;
Table dropped.
SQL>
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_d1,
  16     partition dept_mid values less than (25)
  17       tablespace tbs_d2,
  18     partition dept_high values less than (35)
  19       tablespace tbs_d3
  20     );
Table created.
SQL> exit
> imp scott/tiger tables=emp file=empexp.dat ignore=y
```

```
Export file created by EXPORT:V08.01.05 via conventional path
. importing SCOTT's objects into SCOTT
. . importing table           EMP
. . 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ています。

```
SQL> select empno, deptno from emp partition (dept_low);
     EMPNO     DEPTNO
---------- ----------
      7934         10
      7782         10
      7839         10
3 rows selected.
SQL> select empno, deptno from emp partition (dept_mid);
     EMPNO     DEPTNO
---------- ----------
      7369         20
      7566         20
      7902         20
      7788         20
      7876         20
5 rows selected.

SQL> select empno, deptno from emp partition (dept_high);

     EMPNO     DEPTNO
---------- ----------
      7499         30
      7521         30
      7900         30
      7654         30
      7698         30
      7844         30
6 rows selected.
```

対話方式の使用

コマンド行から Import を開始するときにパラメータを指定しないと、Import ユーティリティは対話方式で開始されます。対話方式では、Import 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性のためにのみ提供されています。

コマンド行でユーザー名とパスワードを指定してしないと、この情報の入力を求めるプロンプトが表示されます。次に、対話方式の例を示します。

```
> imp system/manager

Import: Release 8.1.5.0.0 - Production on Fri Oct 30 09:42:54 1998

(c) Copyright 1998 Oracle Corporation.  All rights reserved.
```

```
Connected to: Oracle8 Enterprise Edition Release 8.1.5.0.0 - Production
With the Partitioning option
PL/SQL Release 8.1.5.0.0 - Production

Import file: expdat.dmp >
Enter insert buffer size (minimum is 8192) 30720>
Export file created by EXPORT:V08.01.05 via conventional path

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

import done in WE8DEC character set and WE8DEC 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): yes >
. importing BLAKE's objects into SYSTEM
. . importing table                          "DEPT"          4 rows imported
. . importing table                       "MANAGER"          3 rows imported
Import terminated successfully without warnings.
```

インポート・セッションによっては、プロンプトが他のプロンプトに対する応答に依存していることがあるので、すべてのプロンプトが表示されるとは限りません。また、デフォルト値が表示されるプロンプトもあります。このデフォルト値を受け入れる場合は、[RETURN] を押します。

注意 : 直前のプロンプトで N を指定している場合、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の入力が要求されます。

```
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。対話方式では、一度に 1 つのスキーマしか指定できません。

増分、累積および全エクスポート・ファイルのインポート

増分エクスポートは、最後に実行された増分または累積、全エクスポート以降に変更された表のみを抽出します。増分エクスポート・ファイルからのインポートでは、変更された行のみではなくその表の定義とすべてのデータがインポートされます。

増分エクスポート・ファイルからのインポートはそのデータがどの方法でエクスポートされたかによって異なるので、1-44 ページ「増分、累積および全エクスポート」も参照してください。

増分エクスポート・ファイルをインポートすると既存オブジェクトの新しいバージョンがインポートされるため、新しいオブジェクトをインポートする前に既存のオブジェクトが削除されます。この処理は、標準インポートとは異なっています。標準インポートでは、オブジェクトは削除されないので、オブジェクトがすでに存在するとエラーになります。

オブジェクト・セットの復元

増分エクスポートおよび累積エクスポート、全エクスポートの実行順序は重要です。オブジェクト・セットは、データベースに対して全エクスポートが実行されるまでは復元できません。全エクスポートの実行後、次に示す手順でオブジェクトを復元します。

1. 最新の増分エクスポート・ファイル（インポートで INCTYPE=SYSTEM を指定）をインポートします。増分エクスポートを実行していない場合は、累積エクスポート・ファイルをインポートします。このステップでは、データベースに、現行のシステム・オブジェクト（たとえば、ユーザー、オブジェクト型、その他）がインポートされます。
2. 最新の全エクスポート・ファイルをインポートします。（インポートで、INCTYPE=RESTORE を指定）
3. 最新の全エクスポート以降のすべての累積エクスポート・ファイルをインポートします。（インポートで、INCTYPE=RESTORE を指定）
4. 最新の累積エクスポート以降のすべての増分エクスポート・ファイルをインポートします。（インポートで、INCTYPE=RESTORE を指定）

たとえば、次のエクスポート・ファイルがあるとします。

- X1 という全エクスポート・ファイル
- C1 および C2 という 2 つの累積エクスポート・ファイル
- I1、I2、I3 という 3 つの増分エクスポート・ファイル

この場合、次の手順でインポートします。

```
imp system/manager INCTYPE=SYSTEM  FULL=Y FILE=I3
imp system/manager INCTYPE=RESTORE FULL=Y FILE=X1
imp system/manager INCTYPE=RESTORE FULL=Y FILE=C1
imp system/manager INCTYPE=RESTORE FULL=Y FILE=C2
imp system/manager INCTYPE=RESTORE FULL=Y FILE=I1
```

```
imp system/manager INCTYPE=RESTORE FULL=Y FILE=I2
imp system/manager INCTYPE=RESTORE FULL=Y FILE=I3
```

注意 :

- 前回実施した増分エクスポート・ファイルを、最初と最後に 2 度インポートしていることに注意してください。つまり、最初は、最新バージョンのシステム・オブジェクトをインポートするためにこのファイルをインポートし、最後は、ユーザー・データおよびオブジェクトに対して最後に実施した変更を適用するためにインポートしています。
- この方法で表を復元するときは、インポートは、ユーザー表が含まれていない新しいデータベースで開始する必要があります。

増分エクスポート・ファイルからオブジェクト型および外部関数ライブラリをインポートする方法

増分インポートの場合に限り、オブジェクト型と外部関数ライブラリはシステム・オブジェクトとして処理されます。つまり、INCTYPE=SYSTEM と指定すると、これらの定義は、他のシステム・オブジェクトとともにインポートされます。この場合、最新のオブジェクト型定義（オブジェクト識別子を含む）と最新のライブラリ仕様定義がインポートされます。

次に、古い増分エクスポート・ファイルから表をインポートする場合、INCTYPE=RESTORE と指定すると、インポートの際に、その表に必要なオブジェクト型が存在するか、またそのオブジェクト型が同じオブジェクト識別子を持っているかが検証されます。オブジェクト型が存在しなかったり、存在するけれどもオブジェクト識別子が一致しない場合、表はインポートされません。

これは、そのオブジェクト型が後続の増分エクスポートで削除されたか、置換されたことを表しています。この場合必ず、そのオブジェクトに依存する表もすべて削除されていなければなりません。

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した Import ユーティリティの動作について説明します。

索引作成およびメンテナンスの制御

SKIP_UNUSABLE_INDEXES=Y と指定すると、Import 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、行が挿入されるとき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 Import する間、索引の更新がセーブされます。

索引のメンテナンスが延期されると、その索引でサポートされている既存の一意整合性制約に対して違反が生じることがあります。表に一意整合性制約が存在しても、INDEXES=N と指定してインポートされた表内でキーの重複を避けることはできません。このため、サポートしている索引は、重複キーが削除されて索引が再構築されるまでは、使用禁止の状態となります。

索引の作成延期

Import 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（再）作成やメンテナンスを実行すると、その処理時間は一般に、Import で各行が挿入される度にメンテナンスを実行するよりも短くて済みます。

索引の作成には時間がかかるので、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が上がります。Import では、インポートが完了するまで、索引の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 Import 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。この指定がない場合、索引作成コマンドは Import 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は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、通常は、CONNECT 文にパスワードを指定した後、SQL スクリプトとして INDEXFILE で指定したファイルの内容を使用します。

Import 後の索引再作成時よりも、データ挿入時の方がメンテナンスする索引のデータの総量が小さい場合は、表データ挿入時にこれらの索引をメンテナンスするよう選択できます。その場合は、INDEXES=Y と指定します。

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・システムにあるとします。また、パーティション p1 にローカル索引 p1_ind と、パーティション p2 にローカル索引 p2_ind があるものとします。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も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ます。

p1_ind を索引メンテナンスする場合は、表データ挿入時に実行する方が、パーティション索引の再作成時に実行する場合よりも、必然的に、処理効率は高くなります。p2_ind については、この逆になります。

また、p2_ind については、Import 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、次に示すステップを実行します。

1. Import 前に、次の SQL 文を発行します。

```
ALTER TABLE t MODIFY PARTITION p2 UNUSABLE LOCAL INDEXES;
```

2. 次の Import コマンドを発行します。

```
imp scott/tiger FILE=export.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. Import 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、Import 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、Import 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 をシャットダウンします。
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 ドキュメントを参照してください。

4. CREATE DATABASE コマンドを使用して、データベースを作成し直します。
5. データベース全体を復元するために、全データベース・インポート（FULL=Y）を実行します。

データベース作成の詳細は、『Oracle8*i* 管理者ガイド』を参照してください。

警告、エラーおよび完了メッセージ

デフォルトでは、すべてのエラー・メッセージが表示されます。LOG パラメータを使用してログ・ファイルを指定すると、エラー・メッセージが端末画面のみでなくログ・ファイルにも書き込まれます。インポートを実行するときには、必ずログ・ファイルを指定するようにしてください。（I/O リダイレクトの可能なシステムでは、インポートの出力先をファイルに変更できます。）

追加情報 : 2-26 ページ「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・システム固有のドキュメントを参照してください。

インポートが問題なく終了すると、「Import terminated successfully without warnings.」というメッセージが表示されます。致命的でないエラーが 1 つ以上発生したが Import ユーティリティは続行し完了できた場合は「Import terminated successfully with warnings.」というメッセージが表示されます。致命的なエラーが起きた場合、Import ユーティリティは即時終了し、「Import terminated unsuccessfully.」というメッセージが表示されます。

追加情報 : メッセージの詳細は、『Oracle8*i* エラー・メッセージ』および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。

エラーの処理

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絶されると、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、後続の行すべて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。

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 かつ一意）制約
- 参照整合性制約
- チェック制約

整合性制約の詳細は、『Oracle8*i* アプリケーション開発者ガイド 基礎編』および『Oracle8*i* 概要』を参照してください。

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、新しい表の列よりも長いデータの挿入、無効なデータ型、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。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。そして、Import ユーティリティは、エクスポート・ファイルの次のデータベース・オブジェクトの処理を継続します。

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、Import ユーティリティは次のデータベース・オブジェクトを処理します。現データベース・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、データベース、SQL 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。

警告 : IGNORE=Y を指定した場合、表の 1 つ以上の列に対して一意整合性制約が指定されていないと、その表に対して、重複した行が挿入されてしまいます。たとえば、誤って 2 度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直す必要があるときには、順序を削除してください。Import ユーティリティでは、既存の順序の削除と再作成は行われません。そのため、インポートの前に削除しなかった場合、順序はエクスポート・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、内部的な問題やメモリーなどのリソースの不足によりリソース・エラーが発生することがあります。

行のインポート中にリソース・エラーが発生すると、現在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、現在の表のインポート済みの部分がコミットされます。

COMMIT=Y を指定していない場合は、現在の表の処理がロールバックされてから、インポートが続行されます（COMMIT パラメータの詳細は、2-20 ページ「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 文よりも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してみなされるため、関連付けられた索引は作成されません。

致命的エラー

致命的なエラーが発生すると、Import は終了します。たとえば、無効なユーザー名およびパスワードの組合せを入力したり、CATEXP.SQL や CATALOG.SQL スクリプトの実行によるデータベースの準備をせずにエクスポートまたはインポートを実行しようとすると、致命的なエラーが発生して、インポートは終了します。

ネットワークに関する考慮事項

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を実行する際の考慮事項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

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場合には、必ずファイルの整合性を保持するプロトコルを使用してください。たとえば、FTP または類似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。

Net8 を利用したエクスポートおよびインポート

Net8 によって、ネットワーク環境で Export ユーティリティおよび Import ユーティリティを使用できるようになります。たとえば、Import 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。

Net8 で Import ユーティリティを使用するには、exp コマンドまたは imp コマンドにユーザー名 / パスワードを入力するときに接続修飾文字列の *@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 Net8 プロトコルのユーザーズ・ガイドを参照してください。Net8 の詳細は、『Oracle8*i* Net8 管理者ガイド』を参照してください。また、『Oracle8*i* 分散システム』も参照してください。

インポートとスナップショット

注意 : 特定の状況、特にデータ・ウェアハウジングに関係している場合、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。ここでは、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、マスター表およびオプションのスナップショット・ログ、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（マスター表およびスナップショット・ログ表定義、スナップショット表）は個別にエクスポートすることができます。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限り、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 Export とユーザー・モード Export 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がインポートされるときに、高速リフレッシュが受ける影響について説明します。スナップショットとスナップショット・ログの詳細は、『Oracle8*i* レプリケーション・ガイド』を参照してください。また、インポート固有の情報については、『Oracle8*i* レプリケーション・ガイド』の付録 B「移行および互換性」も参照してください。

マスター表

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、インポートされたデータはスナップショット・ログに記録されます。

スナップショット・ログ

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 はインポート時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の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防ぐには、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、後続の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・ログのエクスポートでは、キーの値はそのままの意味をインポート時に保持します。したがって、主キーのスナップショットは、インポート後に高速リフレッシュを実行できます。主キーのスナップショットについては、『Oracle8*i* レプリケーション・ガイド』を参照してください。

スナップショットとマテリアライズド・ビュー

エクスポート・ファイルから復元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定めるログ・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復元すると、問題が起こることがあります。

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きに、破損などの問題が発生した場合、スナップショットを削除してインポートし直すことによって復元する必要があります。新たに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となるログ・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

が存在しない場合は、高速リフレッシュは失敗し、完全リフレッシュが必要とされ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、およびスナップショット・ログ、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/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と、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。

インポートおよびインスタンス親和性

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場合、Import および Export ユーティリティでインスタンス親和性を使用する方法の詳細は、『Oracle8*i* 管理者ガイド』、『Oracle8*i* リファレンス・マニュアル』および『Oracle8*i* Parallel Server 概要および管理』を参照してください。

ファイン・グレイン・アクセスのサポート

使用可能なファイン・グレイン・アクセス・ポリシーで、表をエクスポートできます。

ただし、ポリシーを回復するために、そのような表を含むエクスポート・ファイルからインポートするユーザーには、適切な権限が必要です（特に、表のセキュリティ・ポリシーを回復させるために DBMS_RLS パッケージに対する実行権限）。ファイン・グレイン・アクセス・ポリシーを使用した表を含むエクスポートファイルからインポートするための、正しい権限が付与されていない場合、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のため、そのような表をエクスポートおよびインポートを実行するユーザーは、DBA である必要があります。

記憶領域パラメータ

デフォルトでは、表は、元の表領域にインポートされます。

その表領域がもう存在しない場合、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は、そのユーザーに対してはデフォルトの表領域が割り当てられます。

- 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
- オーバフロー・セグメントがある索引構成表を含む表

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については、2-54 ページ「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領域パラメータ 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 INDEX と LOB 列の記憶領域

表は、その表の現行の記憶領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 が作成される際、OIDINDEX の現行の記憶領域パラメータと名前が設定されていれば、それらを使用して作成します。LOB 列または VARRAY 列が含まれている表に関しては、LOB データまたは VARRAY データは、それらの現行の記憶領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、既存の表の記憶領域パラメータを変更する場合もありますが、このような場合、表は、変更された記憶領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 データの記憶領域パラメータは、エクスポートの前には変更できません（たとえば、LOB 列のサイズ、LOB 列が CACHE か 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、インポート時に読込み / 書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ったり、あるい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ので、インポートの際にエラーが発生しても、Import ユーティリティはどの表領域句に起因するエラーかを特定できません。

記憶領域パラメータの上書き

インポートの前に、別の記憶領域パラメータで、事前に大きな表を作成した方が良い場合があります。その場合は、コマンド行またはパラメータ・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、初期エクステント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、記憶領域パラメータが調整されます。初期エクステント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トが整理統合されないように（COMPRESS=N と）指定します。COMPRESS パラメータの説明については、1-16 ページ「COMPRESS」を参照してください。

読取り専用表領域

読取り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・データベース内にもう存在しない場合、表領域は読取り / 書込み表領域として作成されます。読取り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取り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、読取り専用である場合、まずインポート前にこの表領域を読取り / 書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定義し直すと、表領域を削除できます。Import 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 コマンド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。

その表領域のオブジェクトはすべて（ただし、パーティション表、特定の型の表、LOB 列、VARRAY 列、オーバーフロー・セグメントのある索引構成表が含まれている表を除く）、その所有者のデフォルトの表領域にインポートされます。Import ユーティリティは、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、ユーザー自身が表を事前作成した後、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。

表領域がすでに存在しない場合、または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であれば、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。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。ただし次の場合は適用されません。表がパーティション表ではない、LOB 列または VARRAY 列が含まれている、特定の型の表である、オーバーフロー・セグメントのある索引構成表、などの場合です。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。

   注意 : 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、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。）

キャラクタ・セットおよび NLS に関する考慮事項

ここでは、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

CHAR データ

エクスポートおよびインポート操作時に、文字データに対して最高 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、NLS_LANG 環境変数でユーザー・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。ユーザー・セッション用キャラクタ・セットの文字の最大幅と、エクスポート・ファイルのキャラクタ・セットの文字の最小幅の割合が 1 のときのみ、インポート・データはユーザー・セッションのキャラクタ・セットに変換されます。
3. 最後は、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・セッション用キャラクタ・セットと異なる場合に、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ためには、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。

NCHAR データ

データ型が NCHAR、NVARCHAR2 および NCLOB のデータは、ソース・データベースの各国キャラクタ・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・データベースの各国キャラクタ・セットが、インポート・データベースの各国キャラクタ・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。

インポートとシングルバイト・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失することがあります（つまり、等価な 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マシンに固有の 7 ビット・キャラクタ・セットがあるか、NLS_LANG オペレーティング・システム環境変数に 7 ビット・キャラクタ・セットが設定されている場合です。アクセント付きの文字からアクセントが消失するのが最もよく見られる例です。

このような状況を避けるために、NLS_LANG オペレーティング・システム環境変数にエクスポート・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 6 のエクスポート・ファイルをネイティブのオペレーティング・システムのキャラクタ・セットとは異なるキャラクタ・セットや、NLS_LANG の設定と異なるキャラクタ・セット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。

1-53 ページ「キャラクタ・セット変換」を参照してください。

インポートとマルチバイト・キャラクタ・セット

マルチバイト・キャラクタ・セットの場合、インポート・キャラクタ・セットで使用されている文字の最大幅と、エクスポート・キャラクタ・セットで使用されている文字の最小幅の割合が 1 のときのみ、インポート・データはユーザー・セッションのキャラクタ・セットに変換されます。この割合が 1 でない場合は、ユーザー・セッションのキャラクタ・セットはエクスポート・ファイルのキャラクタ・セットと一致するように設定して、変換が行われないようにします。

変換時に、ターゲット・キャラクタ・セットに等しい文字がないエクスポート・ファイル中の文字は、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% 完全に変換されるようにするためには、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・セットと等しくなければなりません。

詳細は、『Oracle8*i* NLS ガイド』を参照してください。

データベース・オブジェクトをインポートする場合の考慮事項

この項では、各種データベース・オブジェクトのインポート中の動作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 Server は、オブジェクト型およびオブジェクト表、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を割り当てます。オブジェクト識別子はインポートで保持されます。

型を参照している表をインポートするときに、データベースにはその名前の型がすでに存在しているという場合、先に存在する型が、実際にその表で使用されているかどうか（実際は異なる型であり、単に同じ名前であるだけではないか）検証されます。

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とが比較され、TOID が一致しない場合は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すべきではない型もあります（たとえば、その型がカートリッジのインストレーションによって作成された場合）。TOID_NOVALIDATE を使用して、TOID と比較しない型を指定できます。詳細は、2-30 ページ「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に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がどのように処理されるかが決められます。

オブジェクト型に関して、IGNORE=Y と指定されていて、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が一致する場合は、エラーはレポートされません。オブジェクト識別子が一致しない場合で、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場合は、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 と指定されていて、そのオブジェクト型がすでに存在する場合は、エラーがレポートされます。オブジェクト識別子が一致しない場合で、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場合は、エラーがレポートされ、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 と指定されていて、オブジェクト表がすでに存在し、そのオブジェクト識別子が一致する場合は、エラーはレポートされません。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が一致しない場合で、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場合は、エラーがレポートされ、その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 と指定されていて、オブジェクト表がすでに存在する場合は、エラーがレポートされ、オブジェクト表はインポートされません。

Import では、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ので、FROMUSER パラメータと 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ときには、次の点を考慮してください。

- FROMUSER のオブジェクト型とオブジェクト表がターゲット表にすでに存在する場合には、TOUSER のオブジェクト型とオブジェクト表の識別子がすでに使用されているのでエラーが発生する。インポート開始前に、FROMUSER のオブジェクト型と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には、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ない。1 つ目の表がインポートできても、同じオブジェクト識別子がすでに使用されているので、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

実際の運用の中では、表領域の使用方法を変えたり、表の記憶領域パラメータを変更するため、インポート前に表を作成しなければならないことがよくあります。表を作成する場合、以前（記憶領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。

オブジェクト型の列を含む表の場合、同じオブジェクト型が指定されなければならず、そのオブジェクト型のオブジェクト識別子も、元のオブジェクト型と同じでなければなりません。パラメータ TOID_NOVALIDATE にオブジェクト型を無視するよう設定した場合、オブジェクト ID は一致しなくてもかまいません。

エクスポートでは、表に使用されているオブジェクト型についての情報がエクスポート・ファイルに書き込まれます。その際、別のスキーマのオブジェクト型の情報も書き込まれます。別のスキーマのオブジェクト型で、一番上のレベルの列に使用されているものについては、インポート時に、名前およびオブジェクト識別子が一致するかどうかが検証されます。別のスキーマのオブジェクト型で、他のオブジェクト型内でネストしているものについては、検証されません。

オブジェクト型がすでに存在する場合は、そのオブジェクト識別子が検証されます。パラメータ TOID_NOVALIDATE にオブジェクト型を無視するよう設定した場合、オブジェクト ID は一致しなくてもかまいません。どのオブジェクト型が作成されたかを示す情報は、インポート時にそのまま保持されるので、あるオブジェクト型が複数の表で使用されていても、そのオブジェクト型は 1 度しか作成されません。

注意 : どのようなケースでも、オブジェクト型は、記憶領域に使用する内部形式に関して互換性がなければなりません。Import ユーティリティは、オブジェクト型の内部形式の互換性についての検証は行いません。エクスポートされたデータに互換性がない場合、インポートの結果は保証できません。

ネストした表のインポート

内側のネストした表は外側の表とは別にエクスポートされます。したがって、内側のネストした表が正しくインポートされない場合、いろいろな状況が予想されます。

- 内側のネストした表を持つ表がインポートされ、インポートの際には、表の削除も表内の行の削除も行われなかったとします。IGNORE=Y パラメータが指定されていると、外側の表に各行を挿入するときに、制約違反が起こります。ところが、内側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、内側の表の行データが重複します。
- 外側の表へデータを挿入中にエラーが発生すると、外側の表の残りのデータはスキップされますが、対応する内側の表の行はスキップされません。その結果、内側の表の行は外側の表のどの行からも参照されないことになります。
- 致命的でないエラーの後で内側の表へのインポートがエラーになっても、外側の表の行はすでに外側の表にインポートされています。また、同じ外側の表内の内側の表や他の内側の表へのデータのインポートが続行されます。その結果、不完全な論理行が作成されます。
- 内側の表へのデータの挿入中に致命的なエラーが起こると、その内側の表の残りのデータはスキップされますが、外側の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・ファイルを細かく調べて、外側の表や内側の表にエラーがないか確かめるようにしてください。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となることがあります。

内側のネストした表は、外側の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、内側の表の行がインポートされる前に、外側の表の行にアクセスすると、ユーザーには不完全な行が戻されます。

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。Import 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 を自動的には再設定しません。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
```

ANALYZE TABLE コマンドの詳細は、『Oracle8*i* SQL リファレンス』を参照してください。

BFILE 列およびディレクトリ別名のインポート

BFILE 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・データベースにコピーされません。BFILE 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波及されるだけです。BFILE 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させてください。

BFILE 列を含む表データをインポートする場合、BFILE ロケータは、エクスポート時のディレクトリ別名とファイル名でインポートされます。Import ユーティリティは、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検証は行いません。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 BFILE データにアクセスするとエラーが発生します。

オペレーティング・システムの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レポートされません。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戻ります。

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

Import ユーティリティでは、外部関数ライブラリの参照先の場所が正しいかどうかは検証されません。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レポートされません。この場合、インポート後にその関数を呼び出そうとすると、エラーが戻ります。

DBA またはユーザーが手動でライブラリを移動させて、ライブラリの指定がインポート・システムで有効となるようにしてください。

ストアド・プロシージャおよびファンクション、パッケージのインポート

ローカルのストアド・プロシージャまたはストアド・ファンクション、パッケージがインポートされるときは、元の指定タイムスタンプが保持されます。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・プロシージャによりアクセスしてもエラーは発生しません。

プロシージャは表およびビュー、シノニムの後でエクスポートされます。したがって、すべての依存関係がすでに存在することから、通常、プロシージャのコンパイルは正常に行われます。ただし、プロシージャおよびファンクション、パッケージは依存関係の順にはエクスポートされません。プロシージャまたはファンクション、パッケージが、後でエクスポート・ダンプ・ファイルに格納されるプロシージャまたはファンクション、パッケージに依存する場合は、正常にコンパイルされません。後にプロシージャまたはファンクション、パッケージを使用すると、それらは自動的に再コンパイルされます。コンパイルが成功すると、タイムスタンプが変更されます。この場合、リモート・プロシージャによるコールでエラーが発生することがあります。

Java オブジェクトのインポート

Java のソースまたはクラスがインポートされる場合、元のリゾルバ（Java のフル・ネームの解決に使用したスキーマのリスト）は、保持されます。オブジェクトが異なるスキーマにインポートされた場合、リゾルバは無効になります。たとえば、SCOTT のスキーマにある Java オブジェクトのデフォルトのリゾルバは、((* SCOTT)（* PUBLIC))です。オブジェクトが BLAKE のスキーマにインポートされる場合、BLAKE のスキーマを参照するようにそのオブジェクトを変更する必要があります。

アドバンスト・キュー（AQ）表のインポート

キューをインポートすると、そのキューの基礎となっている表や関連するディクショナリ表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しか実行できません。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。

詳細は、『Oracle8*i* 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なければなりません。ただし、LONG データはセクション単位でロードされるので、LONG 列を格納するメモリーは連続していなくてもかまいません。

ビューのインポート

ビューのエクスポートには、順序関係があります。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限りません。順序が正しくないと、ビューのインポート時にコンパイル上の警告が出されます。この場合、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、Export ユーティリティは、ビュー VIEWA と VIEWC を正しく順序付けることはできません。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、ビューがコンパイル時のエラーを持ってい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、表またはプロシージャ、他のビューなど）が存在していないと、ビューのコンパイル・エラーが起こるからです。実表が存在しないと、実表に対する権限を付与したユーザー自身が GRANT オプションで適正な権限を持っているかどうかを、サーバーは検証できません。

したがって、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には、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、エラーになります。

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。この非特権ユーザーの場合、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だけでは、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。

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_P*nnn* 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場合、インポート時に非パーティション索引または索引パーティションが（索引使用禁止に設定されていたり、その他の不適合が理由で）メンテナンスできないと、エクスポート・データはターゲット表に挿入できません。

トランスポータブル表領域

トランスポータブル表領域機能は、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。

そのためには、表領域を読取り専用にし、表領域のデータ・ファイルをコピーしてから、Export および Import 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トランスポートは、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 への出力などのような、バイナリ・ファイルの全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。

トランスポータブル表領域を含むエクスポート・ファイルの作成の詳細は、1-57 ページ「トランスポータブル表領域」を参照してください。

次のパラメータ・キーワードで、トランスポータブル表領域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、2-31 ページ「TRANSPORT_TABLESPACE」、2-29 ページ「TABLESPACES」、2-21 ページ「DATAFILES」および 2-31 ページ「TTS_OWNERS」を参照してください。

追加情報 : 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については、『Oracle8*i* 管理者ガイド』を参照してください。トランスポータブル表領域機能の詳細は、『Oracle8*i* 概要』を参照してください。

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あり、表にアナライザ統計が利用できるとき、Export によって ANALYZE コマンドが発行され、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。特定の状況では、表、索引および列に対する事前計算済みのオプティマイザ統計情報が、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、1-23 ページ「STATISTICS」を、インポート・パラメータの詳細は、2-27 ページ「RECALCULATE_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、Export によって保存される ANALYZE 文を計算するよりは、なるべく表（およびその索引や列）の計算済みオプティマイザ統計情報を使用してください。ただし次の場合は、信頼性が低いため、計算済み統計情報は無視されます。

- 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（計算済み統計情報で暗黙に照合順番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 Import 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、計算済み統計は使用できます。これによって、本番データベースから統計情報を使用して、非本番データベースで、問合せの生成プランを調整することができます。

特定の状況では、インポート時、計算済み統計よりも常に ANALYZE コマンドを使用すべき場合もあります。たとえば、分散したデータベースから集めた統計情報で、データが圧縮形式でインポートされたとき、その情報が適切でない場合があります。このような場合は、インポート時に RECALCULATE_STATISTICS=Y を指定して、統計情報を再計算するようにしてください。

インポート時に統計情報を確定しない場合、ANALYZE=N を指定することができ、この場合の RECALCULATE_STATISTICS パラメータは無視されます。2-19 ページ「ANALYZE」を参照してください。

前回リリースの Oracle のエクスポート・ファイルの使用方法

Oracle バージョン 7 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 7 のデータベースのデータを、Oracle8*i* Server 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。詳細は、『Oracle8*i* 移行ガイド』を参照してください。

DATE 列に関するチェック制約

Oracle8*i* では、DATE 列に関するチェック制約を有効にするには、TO_DATE 関数を使用して、日付形式を指定する必要があります。Oracle の旧バージョンでは、この関数は必要なかったので、旧バージョンの Oracle データベースからデータをインポートした場合、TO_DATE 関数が使用されていないことがあります。このような場合には、Oracle8*i* データベースに制約は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 および USER_CONSTRAINTS、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、Oracle8*i* Server 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。詳細は、『Oracle8*i* 移行ガイド』を参照してください。

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。

整合性制約の構文

Oracle バージョン 6 の整合性制約の SQL 構文は、Oracle7 Server および Oracle8*i* Server の構文とは異なりますが、整合性制約は Oracle8*i* Server に適切にインポートされます。

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。

DEFAULT 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、Oracle8*i* へのインポートを実行したときに次のエラーが発生します。

```
ORA-1401: inserted value too large for column
```

Oracle バージョン 6 では、CREATE TABLE 文で列はチェックされず、列の長さがデフォルト値を保持するのに十分かどうかが確認されませんでした。このため、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、Oracle8*i* Server では、CREATE TABLE 文で列がチェックされます。このため、バージョン 6 のデータベースにインポートできた表が、Oracle8*i* にはインポートできないことがあります。

関数によって戻される値が DEFAULT の場合、その関数によって戻される可能性がある最大値を保持するのに十分な列の長さが必要になります。長さが足りないと、エクスポート・ファイルに記録されている CREATE TABLE 文により、インポート時にエラーが発生します。

注意 : Oracle7 では、ユーザー・ファンクションの最大値が大きくなったため、ユーザー・ファンクション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。ユーザー・ファンクションによって戻される最大サイズを判別するには、次の SQL コマンドを実行してください。

```
DESCRIBE user_sys_privs
```

表示される USERNAME 列の長さが、ユーザー・ファンクションによって戻される列の最大長です。

Oracle バージョン 5 のエクスポート・ファイルの使用方法

Oracle8*i* の Import ユーティリティでは、Oracle リリース 5.1.22 以降に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。次の点に注意してください。

- CHAR 列は自動的に VARCHAR2 に変換される
- NOT NULL 制約は ENABLED としてインポートされる
- Import ユーティリティにより、インポートするクラスタの索引が自動的に作成される

CHARSET パラメータ

デフォルト : なし

注意 : このパラメータは Oracle バージョン 5 および 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 6 のエクスポート・ファイルには、NLS キャラクタ・セット識別子がありません。ただし、バージョン 5 または 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 EBCDIC かということが認識されます。

このパラメータは、エクスポート時に実際に使用されたキャラクタ・セットを示すために使用します。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 かどうかが検証されます。

CHARSET パラメータの値の指定を省略すると、ユーザー・セッションのキャラクタ・セットは、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 ASCII、EBCDIC のときは EBCDIC と検証されます。

Oracle7 または Oracle8*i* の Export ユーティリティを使用している場合、キャラクタ・セットはエクスポート・ファイルの中に指定され、現行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チェックするためだけのものです。キャラクタ・セットが一致しないとエラーになります。

第 II 部

SQL*Loader

3

SQL*Loader の概念

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礎的な概念を説明します。この章では、次のトピックについて説明します。

- SQL*Loader の基礎
- SQL*Loader 制御ファイル
- 入力データおよびデータ・ファイル
- データ変換とデータ型仕様
- 廃棄されたレコードと拒否されたレコード
- ログ・ファイルおよびログ情報
- 従来型パス・ロードとダイレクト・パス・ロード
- オブジェクト、コレクションおよび LOB のロード
- パーティション化およびサブパーティション化された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 の基礎

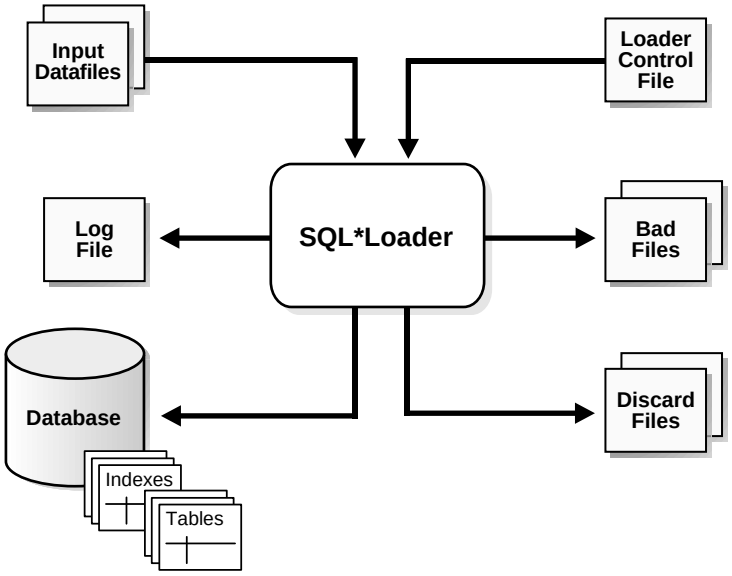
SQL*Loader を使用して、外部ファイルのデータを Oracle データベースの表にロードします。

SQL*Loader には、次の機能があります。

- 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。
- 同一のロード・セッションで、複数のデータ・ファイルからデータをロード。
- 同一のロード・セッションで、複数の表にデータをロード。
- キャラクタ・セットの識別（データのキャラクタ・セットが指定可能）。
- ロード・データの選択が可能（レコード値に基づいたロードが可能）。
- ロード前に、SQL 関数を使用したデータ処理が可能。
- 指定した列に対し、一意の順序キーを生成可能。
- 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が可能。
- ディスク、テープまたは名前付きパイプからのデータのロードが可能。
- 高度なエラー報告機能で、トラブルシューティングを支援。
- 2 つのローディング・パスをサポート－従来型とダイレクト。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（第 8 章「SQL*Loader: 従来型パス・ロードと ダイレクト・パス・ロード」を参照）。
- 複合オブジェクト・リレーショナル・データを任意にロード可能。
- セカンダリ・データ・ファイルセカンダリ・データ・ファイルのサポートによって、LOB およびコレクションをロード。
- IBM 社 DB2 ロード・ユーティリティとの高い互換性によって、DB2 ロード・ユーティリティ制御ファイルを、ほとんど変更せず SQL*Loader 制御ファイルとして使用可能。付録 B「DB2 / DXT ユーザーに対する注意事項」を参照してください。

図 3-1「SQL*Loader の概要」に、SQL*Loader セッションの基本的な構成要素を示します。

図 3-1 SQL*Loader の概要



SQL*Loader では、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、Oracle データベース（データがロードされる）、ログファイル、不良ファイル、および場合によって必要な廃棄ファイルです。

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。制御ファイルには、SQL*Loader が実行するタスクが記述されています。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。制御ファイルの例は、第 4 章「SQL*Loader の事例研究」を参照してください。

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。

1. 第 1 のセクションには、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
   - データのキャラクタ・セット指定

2. 2 番目のセクションは、1 つ以上の INTO TABLE ブロックから構成されます。これらのそれぞれのブロックには、表名および表の列などのような、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。
3. 3 番目の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ついては、次の注意事項があります。

- 構文は、自由形式で記述できる（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ない。
- 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る。ハイフンから行の終わりまでがコメントになります。ただし、オプションである 3 番目のセクションでは、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。
- SQL*Loader に対して特別な意味のある、予約語が存在する（予約語の全リストについては、付録 A「SQL*Loader の予約」を参照）。特殊なリテラルまたはデータベース・オブジェクト名（列名、表名など）に予約語（キーワード）を使用する場合、一重または二重引用符で囲む必要があります。

制御ファイルの構文およびその記述方法の詳細は、第 5 章「SQL*Loader 制御ファイル・リファレンス」を参照してください。

入力データおよびデータ・ファイル

SQL*Loader への入力には、制御ファイル以外に、データがあります。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。「INFILE: データ・ファイルの指定」（5-22 ページ）を参照してください。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。

重要 : 制御ファイル内部でデータが指定されている場合（つまり、INFILE * が制御ファイルに指定された場合）、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。固定形式は、指定が簡単です。次に例を示します。

```
INFILE <datafile_name> "fix n"
```

ここでは SQL*Loader が特殊なデータ・ファイルを、全レコード *n* バイト長の固定レコード形式で解釈す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。第 1 の物理レコードは [001,cd,] で、ちょうど 11 バイト（シングルバイト・キャラクタ・セットと仮定）です。第 2 のレコードは [0002,fghi,] で、11 バイト目に改行文字が続きます。

例 3-1 固定レコード形式へのデータのロード

```
load data
infile 'example.dat'  "fix 11"
into table example
fields terminated by ',' optionally enclosed by '"'
(col1 char(5),
 col2 char(7))

example.dat:
001,   cd, 0002,fghi,
00003,lmn,
1, "pqrs",
0005,uvwx,
```

可変レコード形式

データ・ファイルを可変レコード形式で指定する場合、データ・ファイルの各レコードの開始位置で、文字フィールドの各レコード長が確認されます。この形式は、固定レコード形式よりも柔軟性があり、ストリームレコード形式よりも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。n を指定しない場合、デフォルトは 5 になります。また *n* に、2^{32} - 1 よりも大きい値を指定すると、エラーになります。

例 3-2 に、データ・ファイル example.dat でデータを検索し、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にする指定を示します。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おり、第 1 のレコードは 009（つまり 9）バイト長、第 2 のレコードは 010 バイト長、第 3 のレコードは 012 バイト長です。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。

例 3-2 可変レコード形式へのデータのロード

```
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（SRF）

ストリーム・レコード形式は、最も柔軟性のある形式です。そのため、多少パフォーマンスに影響があります。ストリーム・レコード形式では、レコードをサイズで指定するのではなく、SQL*Loader にレコード終了記号を読み取らせることによって、レコードが認識されます。

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' は、英数字で指定します。ただし、次の場合は、terminator_string を 16 進文字列で指定してください（データ・ファイルのキャラクタ・セットでエンコードされている文字の場合に、指定したとおりの terminator_string にするため）。

- terminator_string に、特別な（印刷不可能な）文字が含まれる場合。
- terminator_string に改行またはキャリッジ・リターン文字が含まれる場合。
- キャラクタ・セットが、クライアント（の制御ファイル）と異なるデータ・ファイルに、terminator_string を指定する場合

terminator_string を指定しない場合、デフォルトは改行（行末）文字（UNIX ベースのプラットフォームでの行送り、Microsoft プラットフォームでは行送りを行うキャリッジ・リターンなど）になります。

例 3-3 に、terminator_string が 16 進文字列で指定されている箇所を、ストリーム・レコード形式でロードする方法を示します。キャラクタセットが ASCII であると仮定すると、文字列 X'7c0a' は、後に改行文字 '\n' が付く '|' に変換されます。例のデータ・ファイルは、2 件のレコードからなり、両方とも正しく '|\n' 文字列（つまり X'7c0a'）で終了しています。

例 3-3 ストリーム・レコード形式へのデータのロード

```
load data
infile 'example.dat'  "str X'7c0a'"
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せる。
- 一定の条件に合致した物理レコードを論理レコードに組み合せる。

4-15 ページ「事例 4: 結合された物理レコードのロード」では、継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例を説明しています。

詳細は、5-36 ページ「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。

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に基づいて、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 によって判断されます。2 つ以上のフィールド指定に同じデータを使用することも、制御ファイルのフィールドに指定されていないデータを論理レコードに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。この指定形式には、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。5-48 ページ「データ・フィールドの位置指定」を参照してください。
- 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。区切られた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。5-69 ページ「デリミタの指定」を参照してください。

- 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。5-48 ページ「データ・フィールドの位置指定」を参照してください。
- 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 X 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。5-57 ページ「SQL*Loader のデータ型」を参照してください。

データ変換とデータ型仕様

図 3-2 は従来型パス・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る段階を示しています（ダイレクト・パス・ロードは概念的には似ていますが、インプリメントは異なります）。図の一番上は、1 つ以上のデータフィールドを含むデータ・レコードを表しています。図の一番下は宛先のデータベースの列を表しています。SQL*Loader を使用するときは、これらの段階を理解しておく必要があります。

図 3-2 は、SQL*Loader と Oracle Server の「役割分担」を表しています。SQL*Loader は、フィールド仕様によりデータ・ファイル形式の解釈方法を判別します。次に Oracle Server で、列のデータ型をもとにデータが変換され、データベースの列に格納されます。データ・ファイルにおけるフィールドとデータベースにおける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・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。

SQL*Loader は、制御ファイルにあるフィールド仕様を使用して入力データを解析し、そのデータを使用する SQL INSERT 文に対応するバインド配列を移入します。次に INSERT 文は、表に格納するために Oracle Server によって実行されます。Oracle Server 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。変換には、次の 2 つのステップがあります。

1. SQL*Loader がデータ・ファイルのフィールドを識別し、データを解釈し、バインド・バッファ経由で Oracle Server へ渡す。
2. Oracle Server がデータを受け取り、データベースにデータを格納する。

図 3-2 入力データ・フィールドの Oracle データベースの列への変換

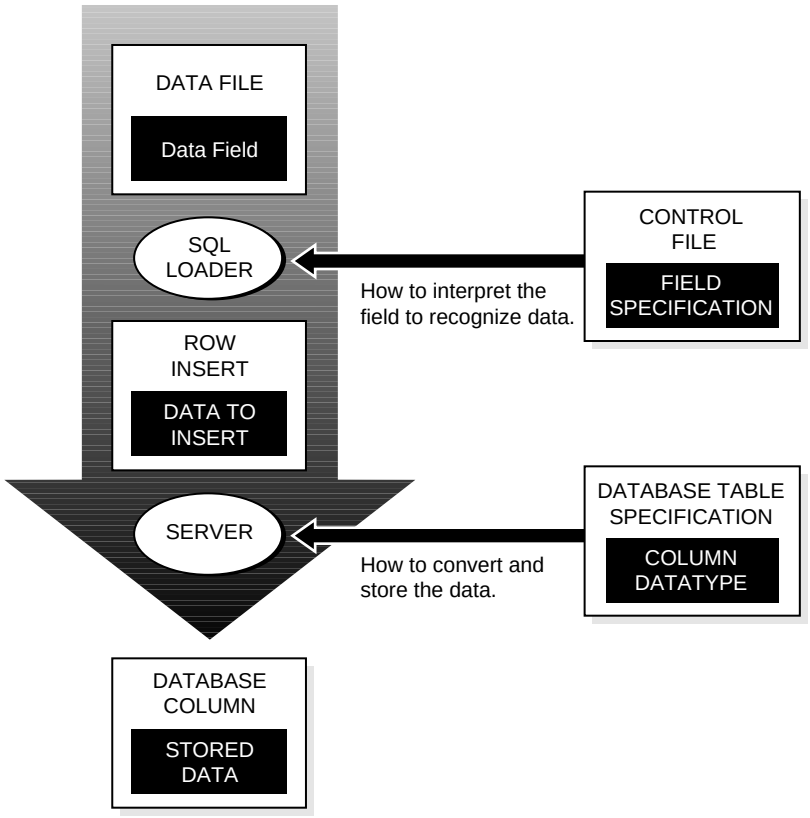
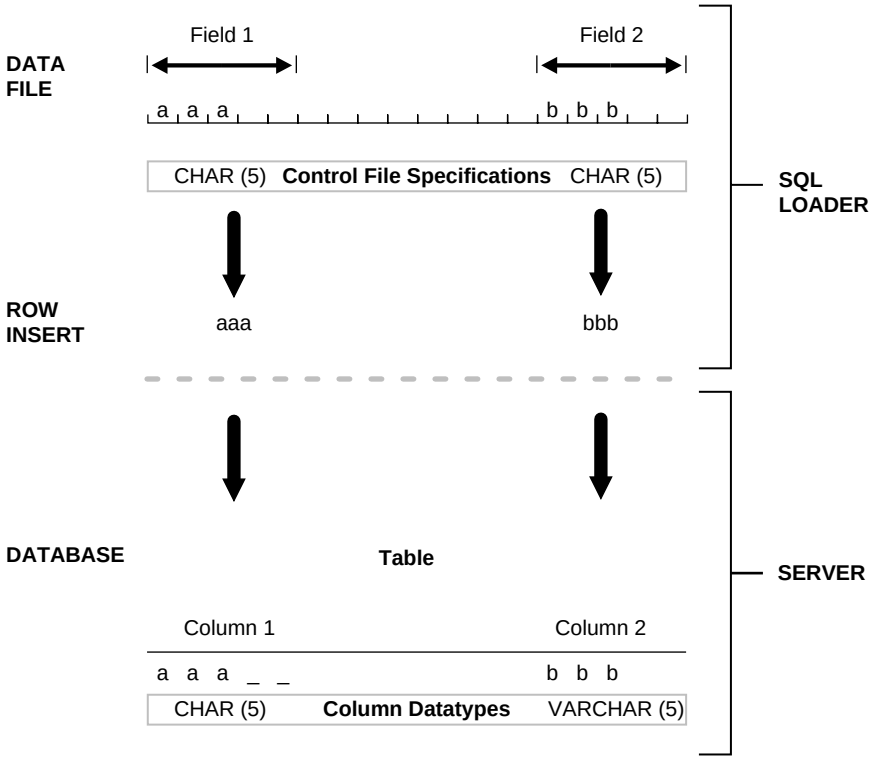


図 3-3 では、2 つの CHAR フィールドがデータ・レコードに定義されています。フィールド指定は制御ファイルに含まれています。制御ファイルの CHAR 指定がデータベースの CHAR 指定と同じではないことに注意してください。制御ファイル内で CHAR として定義されたデータ・フィールドは SQL*Loader に行挿入の作成方法を通知するだけです。Oracle8*i* Server が必要な変換を行い、データベースの CHAR または VARCHAR2、NCHAR、NVARCHAR、NUMBER の列にデータを挿入できるようになります。

デフォルトでは、SQL*Loader は CHAR データから後続ブランクを削除してから、このデータをデータベースに渡します。次に、図 3-3 に示すように、フィールド A とフィールド B はデータベースに 3 列のフィールドとして渡されます。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 3-3 フィールド変換の例



列 A は長さ 5 の固定長 CHAR 列としてデータベースに定義されています。そのため、データ（aaa）は 5 文字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 B は最大長 5 文字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 文字のままです。

フィールドの名前により、SQL*Loader はデータを挿入する列を認識します。最初のデータ・フィールドは制御ファイルで名前 "A" として指定されているため、SQL*Loader はデータの挿入先はターゲット・データベースの表の列 A であると認識します。

次の事項を覚えておくと便利です。

- データ・フィールドの名前は、データのロード先の表の列の名前と対応している。
- SQL*Loader は、フィールドのデータ型によりデータ・ファイルのデータの取扱い方法（バインド型など）を判別する。ただし、このデータ型は列のデータ型とは異なります。SQL*Loader の入力データ型は、列のデータ型には依存しません。
- データは、制御ファイル内で指定されたデータ型から、データベース内の列のデータ型に変換される。
- VARRAY に格納されているデータは、格納前に変換される。
- 論理レコードと物理レコードの違い。

廃棄されたレコードと拒否されたレコード

入力ファイルから読み込まれたレコードがすべてデータベースに挿入されるわけではありません。図 3-4 では、レコードがどの段階で拒否または廃棄されるかについて示します。

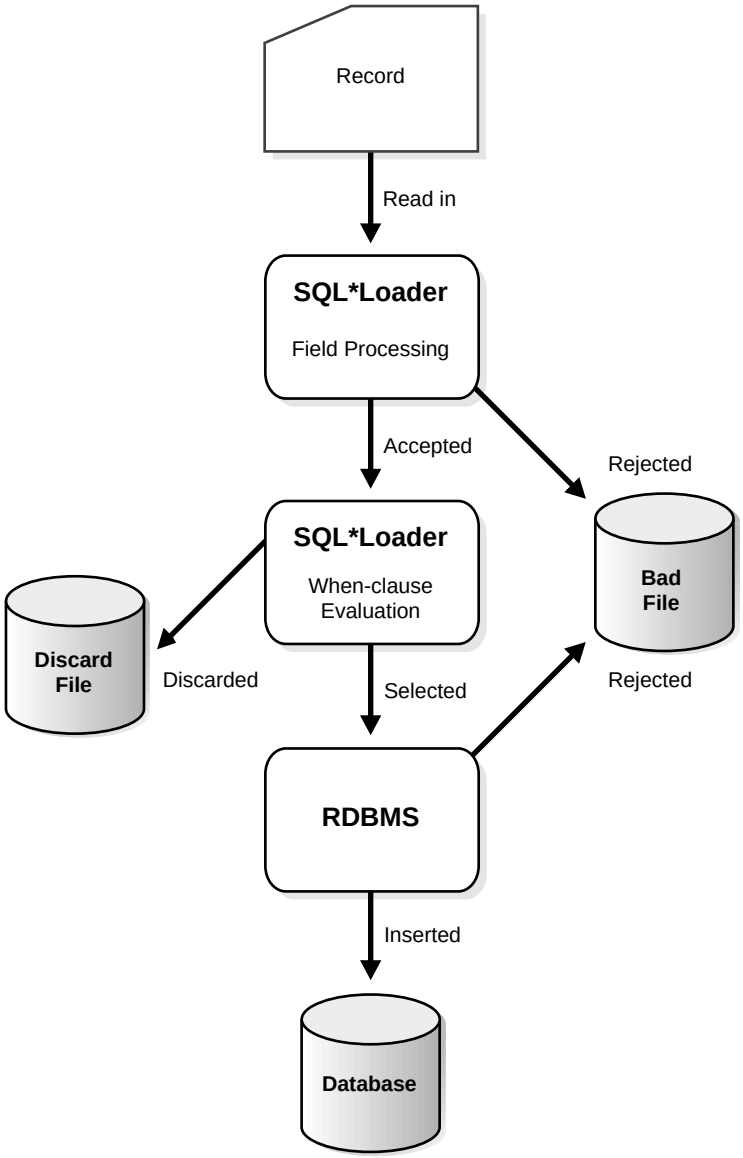
不良ファイル

「不良ファイル」には SQL*Loader または Oracle により受付けが拒否されたレコードが入ります。その理由については次の項で説明します。

SQL*Loader による拒否

入力形式が不適切なレコードは、SQL*Loader により拒否されます。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には、SQL*Loader はレコードを拒否します。拒否されたレコードは不良ファイルに書き込まれます。不良ファイルの指定方法の詳細は、5-25 ページ「BADFILE: 不良ファイルの指定」を参照してください。

図 3-4 レコードのフィルタ処理



Oracle による拒否

レコードが SQL*Loader によって受け付けられた後、各行は挿入のために Oracle に送られます。Oracle により有効であると判別された行は、データベースに挿入されます。行が有効であると判別されなかった場合、レコードは拒否され、不良ファイルに書き込まれます。行が拒否される例としては、キーが重複している場合や必須入力フィールドに対応するデータが NULL 値の場合、フィールドに許可されるデータ型と矛盾するデータ型が指定された場合などが考えられます。

不良ファイルは、データ・ファイルと同じ形式で書き込まれます。したがって、拒否されたデータは適切な修正を行った後で、既存の制御ファイルを使用してロードできます。

4-15 ページ「事例 4: 結合された物理レコードのロード」に、不良ファイルの使用例を示します。

SQL*Loader による廃棄

SQL*Loader の実行により廃棄ファイルが作成されることがあります。廃棄ファイルが作成されるのは、廃棄ファイルが必要な場合で、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限ります（5-27 ページ「廃棄ファイルの指定」を参照）。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まったく合致しなかったために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。

廃棄ファイルは、データ・ファイルと同じ形式で書き込まれます。したがって、廃棄されたデータは適切な編集や修正を行った後で、既存の制御ファイルを使用してロードできます。

4-15 ページ「事例 4: 結合された物理レコードのロード」では、廃棄ファイルがどのように使用されるかについて説明します。詳細は、5-27 ページ「廃棄ファイルの指定」を参照してください。

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ときは、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。ログ・ファイルに記録される情報の詳細は、第 7 章「SQL*Loader: ログ・ファイル参照」を参照してください。また、第 4 章に記載されている各事例にも、ログ・ファイルの例が示されています。

従来型パス・ロードとダイレクト・パス・ロード

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。従来型パスは、バインド配列で SQL INSERT 文を使用し、ダイレクト・パスは、データを直接データベースにロードします。次にこれらのモードについて説明します。また、詳細は、第 8 章「SQL*Loader: 従来型パス・ロードと ダイレクト・パス・ロード」を参照してください。SQL*Loader は、既存の表にロードするのであって、表は作成しません。そのため、ロードする表はデータベースに存在する表でなければなりません。ロードする表にデータがすでに存在していても、または空であっても問題は生じません。

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規にデータをロードする場合には、その表についての DELETE 権限。

追加情報 : Trusted Oracle では、前述の権限に加えて、Trusted Oracle データベースにロードするデータについて、すべてのラベルの書込み権限が必要です。詳細は、ご使用の Trusted Oracle のドキュメントを参照してください。

従来型パス

従来型パス・ロードでは、入力レコードがフィールド仕様をもとに解析され、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への挿入が実行されます。従来型パス・ロードの詳細は、8-2 ページ「データのロード方法」を参照してください。バインド配列の詳細は、5-74 ページ「バインド配列サイズの決定」を参照してください。

SQL*Loader は、LOB フィールドをバインド配列に挿入した後で格納します。そのため、LOB フィールドの処理にエラーがある場合（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。

従来型パスを経由してロードする表については特に必要な条件はありません。

ダイレクト・パス

ダイレクト・パス・ロードでは、入力レコードがフィールド仕様をもとに解析され、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・フォーマッタに渡されます。新規に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。ダイレクト・パスの詳細は、8-2 ページ「データのロード方法」を参照してください。

注意 : LOB、VARRAY、オブジェクトまたはネストした表を含む表には、ダイレクト・パスは使用できません。

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。パラレル・ダイレクト・パスの詳細は、8-2 ページデータのロード方法を参照してください。

オブジェクト、コレクションおよび LOB のロード

SQL*Loader を使用すると、大量のオブジェクト、コレクションおよび LOB をロードできます。オブジェクトの概念およびサポートしているオブジェクト・インプリメンテーションの詳細は、『Oracle8*i* 概要』および『Oracle8*i* 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

表の列が何らかのオブジェクト型であるとき、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、参照できません。

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。そのオブジェクト表には、さらに、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。

SQL*Loader 制御ファイルのデータ定義言語を使用してこれらのオブジェクト型をロードする方法の詳細は 5-90 ページ「列オブジェクトのロード」、および 5-95 ページ「オブジェクト表のロード」を参照してください。

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、列の中にもう 1 つ表があるように見えます。他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、要素という、一連のビルトイン・データ型またはオブジェクトの順番です。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な番号（index）を持ちます。

VARRAY 型を作成するときは、最大数を指定してください。いったん、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。

SQL*Loader 制御ファイルのデータ定義言語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、5-107 ページ「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。このリリースの 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:** シングルバイト文字データを含む LOB。
- **NCLOB:** 各国キャラクタ・セットで、固定サイズの文字を含む LOB。
- **BFILE:** データベースの表領域ではなくサーバー側の OS 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。

SQL*Loader 制御ファイルのデータ定義言語を使用してこれらの LOB 型をロードする方法の詳細は、5-98 ページ「LOB のロード」を参照してください。

新しい SQL*Loader DDL の動作および制限事項

オブジェクトをサポートするため、特定の DDL 句の動作および特定の制限事項が、前回のリリースから変更になっています。これらの変更は、オブジェクト、コレクションまたは LOB をロードする場合のみでなく、すべての場合に対して適用されます。たとえば、次の変更があります。

- レコード :
  - LOBFILE からの LOB をメモリーに合わせる必要はありません。SQL*Loader は、64K 単位で LOBFILE を読み込みます。64K を超える物理レコードをロードするには、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。3-20 ページ「セカンダリ・データ・ファイル（SDF）および LOBFILES」、6-7 ページ「READSIZE（読込みバッファ）」および 5-14 ページ「SDF_spec」を参照してください。
  - 論理レコードは、クライアントが利用可能なメモリーに完全に合わせてください。これによって、特殊なレコードの一部は除外されますが、その部分はセカンダリ・データ・ファイルから読み込まれます。この論理レコード・サイズの制限は、SDF 内のサブ・レコードにも適用されます。3-20 ページ「セカンダリ・データ・ファイル（SDF）および LOBFILES」を参照してください。
- レコード形式 :
  - ストリーム・レコード形式

    ストリーム・レコード形式では、改行文字は、物理レコードの最後を表します。リリース 8.1 からは、OS ファイル処理の文字列に、カスタム・レコード・セパレータを指定できます。詳細は、3-20 ページ「新しい SQL*Loader DDL がサポートするオブジェクト、コレクションおよび LOB」を参照してください。
  - 可変レコード形式

    通常の構文で、INFILE 指示句に続けて "var" 文字列を使用する場合（『Oracle8*i* 概要』を参照）は、指定したレコード長に解釈されるように、各レコードの開始位置で文字列の数を含めるように拡張されています。構文の詳細は、第 5 章を参照してください。

    値が指定されない場合、デフォルトでは、5 文字になります。また、可変レコードの最大サイズは、2^{32} - 1 で、それ以上の値を指定してもエラーになります。

- DEFAULTIF および NULLIF:

  field_condition が真の場合、DEFAULTIF 句は、LOB およびコレクションを空（NULL ではない）に初期化します。

  field_condition が真の場合、NULLIF 句は、他のデータ型と同様に、LOB およびコレクションも NULL に初期化します。

  また、AND 論理演算子を使用して、field_conditions 引数を連結することもできます。構文の詳細は、第 5 章を参照してください。

  注意 :

  - NULLIF および DEFAULTIF 句は、同一のセカンダリ・データ・ファイルのフィールドに存在するフィールド以外は、セカンダリ・データ・ファイル（SDF）にあるフィールドを参照できません。
  - NULLIF および DEFAULTIF のフィールド条件は、LOBFILE からの読取りフィールドに基づくことはできません。

- フィールド・デリミタ

  前バージョンの SQL*Loader では、文字で区切られた（終了したまたは囲まれた）フィールドをロードすることができるようになりました。今回のリリースから、デリミタは、1 文字以上の長さにできます。デリミタ・フィールドを指定する構文は、デリミタ文字として文字列全体を指定できることを除き、前回リリースと同じです。

  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、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、デリミタを 16 進文字列で指定できます（つまり X'<16 進文字列 >'）。デリミタを実際に 16 進文字列で指定する場合、入力データ・ファイルのキャラクタ・セットにおいて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（つまり制御ファイル）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、データ・ファイルのデリミタを検索する前に、データ・ファイルのキャラクタ・セットに変換されます。

  次の点に注意してください。

  - 1 文字デリミタと同様に、文字列デリミタを使用して区切る構文がサポートされています（つまり、囲みデリミタを終了すると区切られます）。
  - 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。

- SQL 文字列

  SQL 文字列は、LOB、BFILE、オブジェクト列、ネストした表、VARRAY をサポートしていないため、SQL 文字列を FILLER フィールドの一部として指定することはできません。

- FILLER フィールド

  ロードを円滑にするため、新たに FILLER キーワードを使用できます。このキーワードは、FILLER フィールドの指定に使用します。FILLER フィールドは、データ・ファイルではマップされているが、対応するデータベースの列が存在しないフィールドです。

  データ・ファイルから、マップされた値が FILLER フィールドに割り当てられます。FILLER フィールドは、多数のファンクションに対する引数として使用できます。たとえば NULLIF です。FILLER フィールドを引数として使用する場合、ファンクションの構文を指定する方法の詳細は、5-3 ページ「SQL*Loader のデータ定義言語（DDL）構文図」を参照してください。

  FILLER フィールドの構文は、フィールド名の後にキーワード 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 フィールドは、NULLIF、DEFAULTIF 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 および DEFAULTIF 句を含めることはできません。FILLER フィールドの構文の詳細は、第 5 章「SQL*Loader 制御ファイル・リファレンス」を参照してください。

  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。

新しい SQL*Loader DDL がサポートするオブジェクト、コレクションおよび LOB

次の項では、SQL*Loader を使用してオブジェクト、コレクション、LOB をロードする方法に関する新しい概念について説明します。

セカンダリ・データ・ファイル（SDF）および LOBFILES

LOB やコレクションのような、一部の新しいデータ型にロードされるデータは、もともと非常に長いデータであるため、そのようなデータのインスタンスを、残りのデータとは別に作成する場合があります。LOBFILES およびセカンダリ・データ・ファイル（SDF）によって、冗長なデータを分割する方法があります。

LOBFILES LOBFILES は、LOB 型へのロードを円滑にする、比較的単純なデータ・ファイルです。LOBFILE とプライマリ・データ・ファイルとの違いは、LOBFILE には、レコードという概念がない点です。LOBFILE のデータ・フィールドは、次のいずれかの型です。

- 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（つまり TERMINATED BY または ENCLOSED BY）

  注意 : 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 または 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。
- ファイルの内容全体を読み込む単一の LOB フィールド

LOBFILE の構文の詳細は、第 5 章を参照してください。

注意 : LOBFILE から読み込むフィールドは、句に対する引数としては使用できません（たとえば、NULLIF 句）。

セカンダリ・データ・ファイル（SDF） セカンダリ・データ・ファイルの概念は、プライマリ・データ・ファイルと同様です。プライマリ・データ・ファイルと同じく、SDF は、レコードおよびフィールドによって形成された各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 キーワードは、SDF の指定に使用します。SDF キーワードに続けて、ファイル指定文字列（5-18 ページ「ファイル名とオブジェクト名の指定」を参照）か、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（3-20 ページ「セカンダリ・データ・ファイル（SDF）および LOBFILES」を参照）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）。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（3-20 ページ「セカンダリ・データ・ファイル（SDF）および LOBFILES」を参照）。
- RECORDSIZE。
- CHARACTERSET 句を使用して、SDF のキャラクタ・セットが指定可能（5-30 ページ「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・フィールドおよび属性、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（デリミタ指定を使用）

- 64K 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。コマンド行からでも、OPTIONS 指示句の一部としてでも READSIZE パラメータを指定できます（5-18 ページ「OPTIONS」を参照）。6-7 ページ「READSIZE（読込みバッファ）」および 5-14 ページ「SDF_spec」も参照してください。

SDF 構文の詳細は、第 5 章を参照してください。

全フィールド名

SQL*Loader を使用して、列オブジェクトのような複合データ型をサポートする場合は、制御ファイル内に、列と列オブジェクトの属性に対する、2 つの同じフィールド名が存在する可能性があることを覚えておいてください。制御ファイルに同名のフィールドが存在する場合、句がフィールド（たとえば、WHEN、NULLIF、DEFAULTIF、SID、OID、REF、BFILE など）を参照すると、名前の重複が発生する場合があります。

したがって、フィールドを参照する句を使用する場合は、フルネーム（たとえば、フィールド fld1 が COLUMN OBJECT に指定されており、そこにはフィールド fld2 が含まれる場合、NULLIF のような句の中で fld2 を指定する場合は、フルネーム fld1.fld2 を指定）で指定してください。

LOBFILE または SDF を指定する場合

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必要があるとします。非常に長い従業員の履歴を LOBFILE から読み込む間に、従業員名および従業員 ID を、メイン・データ・ファイルから読むことができます。

静的・動的な LOBFILE および SDF 指定

SDF および LOBFILE の両方を静的に指定するか（実際のファイル名を指定）、または動的に（FILLER フィールドをファイル名のソースとして使用）指定できます。いずれの場合においても、SDF および LOBFILE の EOF に到達した場合、ファイルはクローズされ、さらに、そのファイルからのソース・データは、空のフィールドからのソース・データと等しい結果を生成します。

動的セカンダリ・ファイルを指定する場合、やや異なる動作になります。参照ファイルの指定が変更になるたびに、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、現行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度オープンし、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 および 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 つのフィールドは、データを別々に読み込みます。

制限事項

- 存在しない SDF または 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るか、または、そのフィールドが空にできない場合は、NULL で初期化されます。
- POSITION 指示句は、LOBFILE からデータを読み込むフィールドでは使用できません。
- 表レベル・デリミタは、SDF および LOBFILE から読み込まれるフィールドに指定できません。

パーティション化およびサブパーティション化されたオブジェクトのサポート

Oracle8*i* 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 SQL*Loader によってサポートしています。Oracle 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。パーティションは一般に、共通の論理属性によってグループ化されます。たとえば、1997 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。

Oracle8*i* で SQL*Loader がパーティション・オブジェクトをサポートしたことによって、SQL*Loader でのロードが可能になったものを、次に示します。

- 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

Oracle8*i* の SQL*Loader では、次に示す 3 つのパス（モード）のどれを使用しても、パーティション・オブジェクトをロードできます。

- 従来型パス : Oracle7 から変更されたのは、行とパーティションとのマッピングが、SQL によって透過的に処理されるようになった点のみです。
- ダイレクト・パス : Oracle7 から大幅に変更されたのは、行と表およびコンポジット・パーティションのパーティションとのマッピング処理において、ローカル索引、ファンクション索引およびグローバル索引（いずれもパーティション化可能）がサポートされるようになった点です。ダイレクト・パスでは、SQL はバイパスされ、ブロックはデータベースに直接ロードされます。

- パラレル・ダイレクト・パス : Oracle7 から変更されたのは、個別パーティションとパーティション表の同時ロードがサポートされた点です。パラレル・ダイレクト・パスでは、複数のダイレクト・パス・ロード・セッションにおいて、同じセグメントまたは同じ一連のセグメントを同時にロードできます。

  パラレル・ダイレクト・パス・ロードはセグメント内での並列処理のために使用します。セグメント内の並列処理は、1 つの表の異なるパーティションをロードする各ロード・セッションが同時に 1 つのパーティションをダイレクト・パス・ロードすることにより実現されることに注意してください。

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、『Oracle8*i* コール・インタフェース・プログラマーズ・ガイド』を参照してください。

4

SQL*Loader の事例研究

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。

この章では次の事柄について説明します。

- 事例研究
- 事例研究ファイル
- 各事例で使用する表
- 参照および注意
- 事例研究 SQL スクリプトの実行
- 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
- 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 フィールドと VARRAY のロード

事例研究

この章には、次の事例があります。

事例 1: 可変長データのロード 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、カンマで区切られているか、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 すべてのレコード長が同一である、固定長フィールドのストリーム形式レコードのデータ・ファイルをロードします。

事例 3: 自由区分形式ファイルのロード 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 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 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ロード 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 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 パーティション表をロードします。

事例 9: LOBFILE のロード（CLOB） 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 フィールドと VARRAY のロード 主キーを OID として利用するカスタマ表をロードします。また、カスタマ表への REF を含む VARRAY を持つオーダー表をロードします。

事例研究ファイル

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（ULCASE1.CTL など）
- データ・ファイル（ULCASE2.DAT など）
- セットアップ・ファイル（ULCASE3.SQL など）

事例研究用のサンプル・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。

事例研究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 4-1 に、各事例に関連のあるファイルを示します。

表 4-1 事例研究および関連ファイル

事例	.CTL	.DAT	.SQL
1	x		x
2	x	x	
3	x		x
4	x	x	x
5	x	x	x
6	x	x	x
7	x	x	x S, E
8	x	x	x
9	x	x	x
10	x		x

追加情報 : 事例研究ファイルの実際の名前は、オペレーティング・システムにより異なります。ご使用のオペレーティング・システム固有の Oracle ドキュメントで、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、SCOTT/TIGER のユーザー ID を持つユーザーが、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、さらに列を追加することもあります。）

EMP 表の内容

```
(empno          NUMBER(4) NOT NULL,
 ename          VARCHAR2(10),
 job            VARCHAR2(9),
 mgr            NUMBER(4),
 hiredate       DATE,
 sal            NUMBER(7,2),
 comm           NUMBER(7,2),
 deptno         NUMBER(2))
```

DEPT 表の内容

```
(deptno         NUMBER(2) NOT NULL,
 dname          VARCHAR2(14),
 loc            VARCHAR2(13))
```

参照および注意

各事例の最初に示す要約に、ページ番号が記されています。このページ番号の項で、各事例が対象としている SQL*Loader の機能を詳しく説明しています。

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、ファイルの後の番号付き注意書きと対応しています。制御ファイルを記述する場合、この番号を付加しないでください。

事例研究 SQL スクリプトの実行

表を作成してデータをロードするために、SQL スクリプトの ULCASE1.SQL と ULCASE3.SQL ～ ULCASE10.SQL を実行する必要があります。事例 2 は ULCASE1.SQL で処理されるので、ULCASE2.SQL はありません。

事例 1: 可変長データのロード

次の項目が対象です。

- ロードする 1 つの表と 3 つの列を示す簡単な制御ファイル。5-21 ページ「BEGINDATA を使用した、制御ファイルのデータの識別」を参照してください。
- 制御ファイル内に組み込まれたデータのロード（したがって個別のデータ・ファイルは存在しません）。5-21 ページ「BEGINDATA を使用した、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、すなわちカンマで終了するフィールドと引用符で囲まれたフィールドを使ったストリーム形式のデータのロード。5-73 ページ「デリミタ付きフィールド」を参照してください。

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) BEGINDATA
   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、データが外部の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 が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。データ型は指定されないので、デフォルトの 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。

SQL*Loader の起動

この事例を実行するには、次のコマンドを入力して SQL*Loader を起動します。

```
sqlldr userid=scott/tiger control=ulcase1.ctl log=ulcase1.log
```

SQL*Loader が DEPT 表をロードしてログ・ファイルを作成します。

追加情報 : コマンド sqlldr は UNIX の場合の起動方法です。SQL*Loader を使用しているオペレーティング・システムで起動するには、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。

ログ・ファイル

次にログ・ファイルの一部を示します。

```
Control File:   ulcase1.ctl
Data File:      ulcase1.ctl
  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 65536 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:                  65016 bytes(84 rows)
Space allocated for memory besides bind array:        0 bytes

Total logical records skipped:          0
Total logical records read:             7
Total logical records rejected:         0
Total logical records discarded:        0

Run began on Sun Nov 08 11:08:19 1998
Run ended on Sun Nov 08 11:08:20 1998

Elapsed time was:     00:00:01.16
CPU time was:         00:00:00.10
```

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囲んでもデータを区切っていることを示しています。

事例 2: 固定形式フィールド

次の項目が対象です。

- 別個のデータ・ファイル。5-22 ページ「INFILE: データ・ファイルの指定」を参照してください。
- データ変換。5-68 ページ「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。

制御ファイル

制御ファイルは ULCASE2.CTL 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、キーワード INFILE 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、表 EMP の列名です。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. 一連の列を指定する場合は、カッコで囲みます。

データ・ファイル

サンプルとして、ファイル ULCASE2.DAT のデータ行の一部を示します。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
```

SQL*Loader の起動

次のコマンドを入力し SQL*Loader を起動してください。

```
sqlldr userid=scott/tiger control=ulcase2.ctl log=ulcase2.log
```

この例でロードされた EMP 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ないと、参照整合性検査によってそれらのレコードは拒否されます（参照整合性の制約が EMP 表で使用可能の場合）。

追加情報 : コマンド sqlldr は UNIX の場合の起動方法です。SQL*Loader を使用しているオペレーティング・システムで起動するには、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。

ログ・ファイル

次にログ・ファイルの一部を示します。

```
Control File:   ulcase2.ctl
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 65536 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:                  65520 bytes(1092 rows)
Space allocated for memory besides bind array:        0 bytes

Total logical records skipped:          0
Total logical records read:             7
Total logical records rejected:         0
Total logical records discarded:        0

Run began on Sun Nov 08 11:09:31 1998
Run ended on Sun Nov 08 11:09:32 1998

Elapsed time was:     00:00:00.63
CPU time was:         00:00:00.16
```

事例 3: 自由区分形式ファイルのロード

次の項目が対象です。

- ストリーム形式のデータ（デリミタで囲まれているデータ、およびデリミタで終端を示されているデータ）のロード。5-73 ページ「デリミタ付きフィールド」を参照してください。
- データ型 DATE を使った日付のロード。5-64 ページ「DATE」を参照してください。
- ロードしたデータに一意キーを生成するための SEQUENCE 番号の使用。5-55 ページ「列への一意の順序番号の設定」を参照してください。
- 新規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示するための APPEND の使用。5-32 ページ「空および空でない表へのデータのロード」を参照してください。
- 制御ファイルでのコメントの使用。コメントは先頭に二重ハイフンを付けます。5-17 ページ「制御ファイルの基礎」を参照してください。
- 個々のフィールドの宣言による、一般の指定の上書き。5-44 ページ「フィールド条件の指定」を参照してください。

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 列と LOADSEQ 列が含まれていません。ユーザーがこの制御ファイルをテストする場合には、次のコマンドを入力して EMP 表に 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、カンマで終了さ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行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。コメントは二重ダッシュの後に記述します。二重ダッシュはコマンド行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. 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形式は、DD-Month-YYYY です。日付フィールドの長さは、最大で 20 です。長さが指定されないときは、フィールドの長さは最大の 20 となります。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、行の COMM が NULL 値となることがあるので注意してください。

SQL*Loader の起動

次のコマンドを入力し SQL*Loader を起動してください。

```
sqlldr userid=scott/tiger control=ulcase3.ctl log=ulcase3.log
```

追加情報 : コマンド sqlldr は UNIX の場合の起動方法です。SQL*Loader を使用しているオペレーティング・システムで起動するには、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。

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 65536 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:                  65379 bytes(31 rows)
Space allocated for memory besides bind array:        0 bytes

Total logical records skipped:          0
Total logical records read:             7
Total logical records rejected:         0
Total logical records discarded:        0

Run began on Sun Nov 08 11:13:41 1998
Run ended on Sun Nov 08 11:13:46 1998

Elapsed time was:     00:00:04.83
CPU time was:         00:00:00.09
```

事例 4: 結合された物理レコードのロード

次の項目が対象です。

- CONTINUEIF を使用して、複数の物理レコードを結合し 1 件の論理レコードを作成する。5-36 ページ「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。
- 負数を挿入する。
- 新しいデータを挿入する前に、REPLACE を使用して表を空にしておく必要があることを指定する。5-32 ページ「空および空でない表へのデータのロード」を参照してください。
- DISCARDFILE を使用して、制御ファイル内に廃棄ファイルを指定する。5-27 ページ「廃棄ファイルの指定」を参照してください。
- DISCARDMAX を使用して、廃棄レコード件数の最大値を指定する。5-27 ページ「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理由にレコードを拒否する。5-26 ページ「拒否レコード」を参照してください。

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. 実行終了までに処理できる廃棄レコード件数の最大値を、999 に設定します（事実上すべてのレコードを廃棄できます）。
3. REPLACE を指定すると、データをロードする先の表上にすでにデータが含まれている場合、SQL*Loader はそのデータを削除してから新しいデータをロードします。
4. CONTINUEIF THIS を指定すると、現行レコードの 1 列目にアスタリスクがある場合、その次の物理レコードはこの現行レコードに付加され、論理レコードが形成されます。したがって、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るので注意してください。

データ・ファイル

この事例のデータ・ファイル ULCASE4.DAT を次に示します。1 列目のアスタリスク、および表示されませんが 20 列目の復帰改行標識に注目してください。また、CLARK のコミッションが -10 である場合、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るので注意してください。

```
*7782 CLARK
MANAGER   7839 2572.50    -10    2512-NOV-85
*7839 KING
PRESIDENT      5500.00           2505-APR-83
*7934 MILLER
CLERK     7782 920.00            2508-MAY-80
*7566 JONES
MANAGER   7839 3123.75           2517-JUL-85
*7499 ALLEN
SALESMAN  7698 1600.00   300.00  25 3-JUN-84
*7654 MARTIN
SALESMAN  7698 1312.50  1400.00  2521-DEC-85
*7658 CHAN
ANALYST   7566 3450.00           2516-FEB-84
*     CHEN
ANALYST   7566 3450.00           2516-FEB-84
*7658 CHIN
ANALYST   7566 3450.00           2516-FEB-84
```

拒否レコード

最後の 2 件のレコードは、次の理由で受付けが拒否されます。EMPNO 列に重複を許さない索引が作成されている場合には、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には、EMPNO に値が存在しないため CHEN のレコードが受付けを拒否されます。

SQL*Loader の起動

次のコマンドを入力し SQL*Loader を起動してください。

```
sqlldr userid=scott/tiger control=ulcase4.ctl log=ulcase4.log
```

追加情報 : コマンド sqlldr は UNIX の場合の起動方法です。SQL*Loader を使用しているオペレーティング・システムで起動するには、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。

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 65536 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: cannot insert NULL into ("SCOTT"."EMP"."EMPNO")

Record 9: Rejected - Error on table EMP.
ORA-00001: unique constraint (SCOTT.EMPIX) violated
```

```
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:                  65520 bytes(910 rows)
Space allocated for memory besides bind array:        0 bytes

Total logical records skipped:          0
Total logical records read:             9
Total logical records rejected:         2
Total logical records discarded:        0

Run began on Sun Nov 08 11:49:42 1998
Run ended on Sun Nov 08 11:49:42 1998

Elapsed time was:     00:00:00.69
CPU time was:         00:00:00.13
```

不良ファイル

前述した理由のため、次に示すように、レコード 8 とレコード 9 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           2516-FEB-84
*     CHIN         ANALYST
      7566         3450.00           2516-FEB-84
```

事例 5: 複数表へのデータのロード

次の項目が対象です。

- 複数の表へのロード。5-52 ページ「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解除し、そのデータを正規化した表にロードする（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の作成。5-50 ページ「複数の INTO TABLE 文の使用」を参照してください。
- WHEN 句の使用。5-40 ページ「ロードする行の選択」を参照してください。
- 同一フィールド（EMPNO）の複数の表へのロード。

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
3) 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 を指定すると、データをロードする先の表（EMP 表および PROJ 表）上にすでにデータが含まれている場合、SQL*Loader はそのデータを削除してから新しい行をロードします。
2. INTO 句を複数指定して、2 つの表、EMP と PROJ にデータをロードします。PROJ 表のロードのために、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限り、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限り、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限り、行が PROJ に挿入されます。

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
```

SQL*Loader の起動

次のコマンドを入力し SQL*Loader を起動してください。

```
sqlldr userid=scott/tiger control=ulcase5.ctl log=ulcase5.log
```

追加情報 : コマンド sqlldr は UNIX の場合の起動方法です。SQL*Loader を使用しているオペレーティング・システムで起動するには、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。

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 65536 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, column DEPTNO.
1) ORA-00001: unique constraint (SCOTT.EMPIX) violated

1) Record 8: Rejected - Error on table EMP, column DEPTNO.
1) ORA-01722: invalid number

1) Record 3: Rejected - Error on table PROJ, column PROJNO.
1) ORA-01722: invalid number


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:                  65536 bytes(1024 rows)
Space allocated for memory besides bind array:        0 bytes

Total logical records skipped:          0
```

```
Total logical records read:           12
Total logical records rejected:        3
Total logical records discarded:       0

Run began on Sun Nov 08 11:54:39 1998
Run ended on Sun Nov 08 11:54:40 1998

Elapsed time was:     00:00:00.67
CPU time was:         00:00:00.16
```

注意 :

1. バッファ方式（配列バッチ方式）を採っているため、物理レコードの順序でエラーが検出されるわけではありません。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 の行）は受付けを拒否されました。受付けを拒否された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・エラーのため受付けを拒否されました。
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、EDDS）はデータ・エラーのため受付けを拒否されました。
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、EDDS）はデータ・エラーのため受付けを拒否されました。

ロード結果

SQL*Loader の実行結果は次のとおりです。

```
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       ALBERT          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の、ダイレクト・パス・ロード方式の使用。第 8 章の「SQL*Loader: 従来型パス・ロードと ダイレクト・パス・ロード」を参照してください。
- データを事前ソートする索引の指定方法。8-16 ページ「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。
- ブランクの数値フィールドを NULL としてロードする。5-81 ページ「ブランク・フィールドのロード」を参照してください。
- NULLIF 句。5-80 ページ「NULLIF キーワード」を参照してください。

  注意 : データをロードする表名を指定してください。指定しないと、LDR-927 が表示されます。コマンド行パラメータとしての DIRECT=TRUE の指定は、表のシノニムへロードするときのオプションではありません。

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。

制御ファイル

制御ファイルは ULCASE6.CTL です。

```
   LOAD DATA
   INFILE 'ulcase6.dat'
   INSERT
   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 句で、データをソートする索引を指定します。この句を指定すると、索引 EMPIX の列によってデータ・ファイルがソートされます。したがって、SQL*Loader がダイレクト・パス・ロード方法を使用するときにこのデータに対するソートの処理がなくなるため、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（ブランク）のときに、NULL として列をロードするために使用されます。詳細は、5-81 ページ「ブランク・フィールドのロード」を参照してください。

SQL*Loader の起動

スクリプト ULCASE6.SQL を SCOTT/TIGER として実行し、次のコマンド行を入力してください。

```
sqlldr scott/tiger ulcase6.ctl direct=true log=ulcase6.log
```

追加情報 : コマンド sqlldr は UNIX の場合の起動方法です。SQL*Loader を使用しているオペレーティング・システムで起動するには、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。

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
------------------------------ ---------- ----- ---- ---- ---------------------
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.EMPIX 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.
Space allocated for memory besides bind array:        0 bytes

Total logical records skipped:          0
Total logical records read:             7
Total logical records rejected:         0
Total logical records discarded:        0

Run began on Sun Nov 08 11:15:28 1998
Run ended on Sun Nov 08 11:15:31 1998

Elapsed time was:     00:00:03.22
CPU time was:         00:00:00.10
```

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。機能は次のとおりです。

- SQL*Loader の使用における INSERT トリガーの定義（『Oracle8*i* アプリケーション開発者ガイド 基礎編』のデータベース・トリガーに関する章を参照してください。）
- データを処理するための SQL 文字列の使用。5-87 ページ「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。5-69 ページ「デリミタの指定」を参照してください。
- SYSDATE の使用。5-55 ページ「列への現在の日付の設定」を参照してください。
- TRAILING NULLCOLS 句の使用。5-42 ページ「TRAILING NULLCOLS」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。5-68 ページ「システム固有なデータ型フィールド長の衝突」および 5-72 ページ「文字データ型フィールド長の矛盾」を参照してください。

注意 : この事例は未指定のフィールドの最終値を使用するトリガーを作成します。

データ・ファイル

次のレポートのリストはロードされるデータを示しています。

```
          Today's Newly Hired Employees
Dept  Job       Manager   MgrNo  Emp Name  EmpNo  Salary     (Comm)
----  --------  --------  -----  --------  -----  --------- ------
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
```

挿入トリガー

この事例では、部門番号、ジョブ名およびマネージャ番号フィールドがデータ行にない場合にそれらを書き込むために、BEFORE INSERT トリガーが必要です。値が存在する場合、グローバル変数に保存されます。値が存在しない場合、グローバル変数が使用されます。

INSERT トリガーおよびグローバル変数を定義するパッケージを次に示します。

```
CREATE OR REPLACE PACKAGE uldemo7 AS   -- Global Package Variables
    last_deptno   NUMBER(2);
    last_job      VARCHAR2(9);
    last_mgr      NUMBER(4);
    END uldemo7;
/
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。この句を指定しないと、SQL*Loader は配列インタフェースを使用するため、INSERT トリガーは各挿入配列ごとに 1 度のみ起動します。

制御ファイル

制御ファイルは ULCASE7.CTL です。

```
   LOAD DATA
   INFILE 'ULCASE7.DAT'
   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（給与のフィールド）の小数点はデータの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 はレコードの最後で欠落しているフィールドを NULL として扱います。コミッション・フィールドは各レコードごとに存在するわけではありません。そのため、この句は 7 フィールド見つかるはずの問合せに対して 6 フィールドしか見つからない場合でも、レコードを拒否しないで NULL コミッションをロードすることを指示します。
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. この指定は警告メッセージを生成します。指定された長さがフィールドの位置によって決定された長さと一致していないからです。指定された長さ (3) が使用されます。
5. レポートは値が変更されたときのみ部門番号およびジョブ、マネージャを表示するため、これらのフィールドはブランクになる可能性があります。この制御ファイルはそれらを NULL としてロードします。そして RDBMS 挿入トリガーが最新の有効値を書き込みます。
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ブランクの場合、ジョブ・フィールドが TERMINATED BY BLANKS 句であると、SQL*Loader は、従業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字に変換します。数値はスペースが少なくてすみ、さまざま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ぶために使用されます。その後、SQL 文字列は値を格納される様式に変換されます。

SQL*Loader の起動

次のコマンドを入力し SQL*Loader を起動してください。

```
sqlldr scott/tiger ulcase7.ctl ulcase7.log
```

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:  none specified

 (Allow all discards)

Number to load: ALL
Number to skip: 0
Errors allowed: 50
Bind array:     64 rows, maximum of 65536 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.
  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:                  65286 bytes(81 rows)
Space allocated for memory besides bind array:        0 bytes

   Total logical records skipped:          0
   Total logical records read:            13
   Total logical records rejected:         0
2) Total logical records discarded:        7

Run began on Sun Nov 08 11:16:30 1998
Run ended on Sun Nov 08 11:16:31 1998

Elapsed time was:     00:00:00.75
CPU time was:         00:00:00.09
```

注意 :

1. 指定された長さと、位置指定から導出された長さが違うために生成された警告です。
2. レポートの先頭の 6 行のヘッダー行は拒否され、また同様にレポート内の空白のセパレータ行も拒否されます。

挿入トリガーおよびグローバル変数パッケージの削除

この事例を実行後、ULCASE7E.SQL を使用して挿入トリガーおよびグローバル変数パッケージを削除します。

事例 8: パーティション化された表のロード

次の項目が対象です。

- データのパーティション化。パーティション化データの概念の詳細は、『Oracle8*i* 概要』を参照してください。
- 明示的に定義されたフィールド位置およびデータ型。
- 固定レコード長オプションを使用してのロード。

制御ファイル

制御ファイルは ULCASE8.CTL です。出荷日に従ってデータをパーティション化し、固定長レコードの行項目表をロードします。

```
LOAD DATA
1)  INFILE 'ulcase10.dat' "fix 129"
BADFILE 'ulcase10.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_linenumber    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）。3-5 ページ「入力データおよびデータ・ファイル」を参照してください。
2. 各列にロードするデータ・ファイル中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ために、出荷日に従って分けられた 4 つのパーティションを使用して行項目表が作成されます。

```
create table lineitem
(l_orderkey     number,
l_partkey       number,
l_suppkey       number,
l_linenumber    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
)
```

データ・ファイルの入力

この事例のデータ・ファイル ULCASE8.DAT を次に示します。各レコード長は 129 文字です。ファイル内の各レコードの前には、ブランクが 5 つあることに注意してください。

```
     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
O52M70MRgRNnmm476mNm
     3 9451 721230 41113.5.05.01AF05-SEP-9629-DEC-9318-FEB-94TAKE BACK RETURN FOB
6wQnO0Llg6y
     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
```

SQL*Loader の起動

次のコマンドを入力し SQL*Loader を起動してください。

```
sqlldr scott/tiger control=ulcase8.ctl data=ulcase8.dat
```

追加情報 : コマンド sqlldr は UNIX の場合の起動方法です。SQL*Loader を起動するには、Oracle 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。

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:     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 65536 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: inserted partition key is outside specified partition

Record 5: Rejected - Error on table LINEITEM, partition SHIP_Q1.
ORA-14401: inserted partition key is outside specified partition

Record 6: Rejected - Error on table LINEITEM, partition SHIP_Q1.
ORA-14401: inserted partition key is outside specified partition

Record 7: Rejected - Error on table LINEITEM, partition SHIP_Q1.
ORA-14401: inserted partition key is outside specified partition

Record 8: Rejected - Error on table LINEITEM, partition SHIP_Q1.
ORA-14401: inserted partition key is outside specified partition

Record 9: Rejected - Error on table LINEITEM, partition SHIP_Q1.
ORA-14401: inserted partition key is outside specified partition

Record 10: Rejected - Error on table LINEITEM, partition SHIP_Q1.
```

```
ORA-14401: inserted partition key is outside specified partition


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:                  65532 bytes(381 rows)
Space allocated for memory besides bind array:        0 bytes

Total logical records skipped:          0
Total logical records read:            10
Total logical records rejected:         7
Total logical records discarded:        0

Run began on Sun Nov 08 11:30:49 1998
Run ended on Sun Nov 08 11:30:50 1998

Elapsed time was:     00:00:01.11
CPU time was:         00:00:00.14
```

事例 9: LOBFILE のロード（CLOB）

次の項目が対象です。

- RESUME と呼ばれる CLOB 列を EMP 表に追加する。
- FILLER フィールド（RES_FILE）を使用する。
- 複数の LOBFILE を EMP 表にロードする。

制御ファイル

制御ファイルは ULCASE9.CTL です。RESUME を使用して、新しい EMP レコードを異なる表のそれぞれの従業員にロードします。

```
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. これは FILLER フィールドです。データ・ファイルから、マップされた値が FILLER フィールドに割り当てられます。詳細は、3-20 ページ「セカンダリ・データ・ファイル（SDF）および LOBFILES」を参照してください。

2. RESUME 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する場合に使用されます。詳細は、5-101 ページ「LOBFILE を使用した LOB データのロード」を参照してください。

データ・ファイルの入力

```
>>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 - Saleman 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 agressive 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
```

SQL*Loader の起動

次のコマンドを入力し SQL*Loader を起動してください。

```
sqlldr sqlldr/test control=ulcase9.ctl data=ulcase9.dat
```

追加情報 : コマンド sqlldr は UNIX の場合の起動方法です。SQL*Loader を起動するには、Oracle 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。

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 65536 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     *  WHT      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:                  63984 bytes(31 rows)
Space allocated for memory besides bind array:       0 bytes

Total logical records skipped:          0
```

```
Total logical records read:             7
Total logical records rejected:         0
Total logical records discarded:        0

Run began on Sun Nov 08 11:31:11 1998
Run ended on Sun Nov 08 11:31:19 1998

Elapsed time was:     00:00:08.14
CPU time was:         00:00:00.09
```

事例 10: REF フィールドと VARRAY のロード

次の項目が対象です。

- 主キーを OID として利用するカスタマ表のロード。
- カスタマ表への REF を含む VARRAY を持つオーダー表のロード。

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. これは FILLER フィールドです。データ・ファイルから、マップされた値が FILLER フィールドに割り当てられます。詳細は、3-20 ページ「セカンダリ・データ・ファイル（SDF）および LOBFILES」を参照してください。
2. このフィールドは、REF フィールドとして作成されます。
3. VARRAY に格納される `item_list` を参照してください。
4. item_list の 2 番目のオカレンスで、VARRAY の各要素のデータ型を識別します。そこでは、データ型は列オブジェクトです。
5. このリストでは、VARRAY に格納されているすべての列オブジェクトの属性が分かります。リストは、カッコで囲まれています。列オブジェクトのロードの詳細は、5-90 ページ「列オブジェクトのロード」の項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、データの前にキーワード BEGINDATA が付いています。

SQL*Loader の起動

次のコマンドを入力し SQL*Loader を起動してください。

```
sqlldr sqlldr/test control=ulcase10.ctl
```

追加情報 : コマンド sqlldr は UNIX の場合の起動方法です。SQL*Loader を起動するには、Oracle 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。

ログ・ファイル

次にログ・ファイルの一部を示します。

```
Control File:   ulcase10.ctl
Data File:      ulcase10.ctl
  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 65536 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:                  65240 bytes(28 rows)
Space allocated for memory besides bind array:        0 bytes

Total logical records skipped:          0
Total logical records read:             2
Total logical records rejected:         0
Total logical records discarded:        0

Run began on Sun Nov 08 11:46:13 1998
Run ended on Sun Nov 08 11:46:14 1998

Elapsed time was:     00:00:00.65
CPU time was:         00:00:00.16
```

5

SQL*Loader 制御ファイル・リファレンス

この章では、SQL*Loader 制御ファイルの構文について説明します。この章では、次のトピックについて説明します。

SQL*Loader のデータ定義言語（DDL）

SQL*Loader の制御ファイル : ロードの設定

SQL*Loader の制御ファイル : データのロード

SQL*Loader の制御ファイル : オブジェクト、LOB およびコレクションのロード

SQL*Loader のデータ定義言語（DDL）構文図

SQL*Loader のデータ定義言語（DDL）を使用して、データベースにデータをロードする方法を制御します。また、DDL を使用して、ロードしているデータを処理できます。

SQL*Loader 制御ファイル

SQL*Loader 制御ファイルは、ロードされるデータの位置、データ書式設定の方法、データがロードされるときの 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）、データがロードされたときの処理方法、などについて DDL 命令を記述するためにユーザーが作成するリポジトリです。SQL*Loader 制御ファイルを作成し、内容は vi や xemax のような簡易テキスト・エディタを使用して編集します。

この章の残りの部分では、DDL を使用して必要なデータをロードする方法について説明します。

SQL*Loader DDL 構文図の表記法

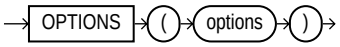
この章で、構文の説明に使用されている SQL*Loader DDL 構文図（線路図ともいいます）は、標準 SQL 構文の表記法で記述されています。この章で使用している構文の表記法の詳細は、『PL/SQL ユーザーズ・ガイドおよびリファレンス』または『Oracle8*i* SQL リファレンス』の「はじめに」を参照してください。

詳細は、5-17 ページ「制御ファイルの基礎」を参照してください。

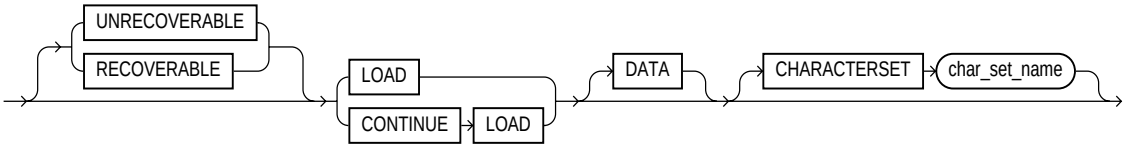
高水準の構文図

次に示す DDL 構文図は、特定の句が省略された形で示されています（position_spec 句や into_table 句など）。5-15 ページ「拡張された DDL 構文」で、これらの構文図を展開して詳しく説明します。

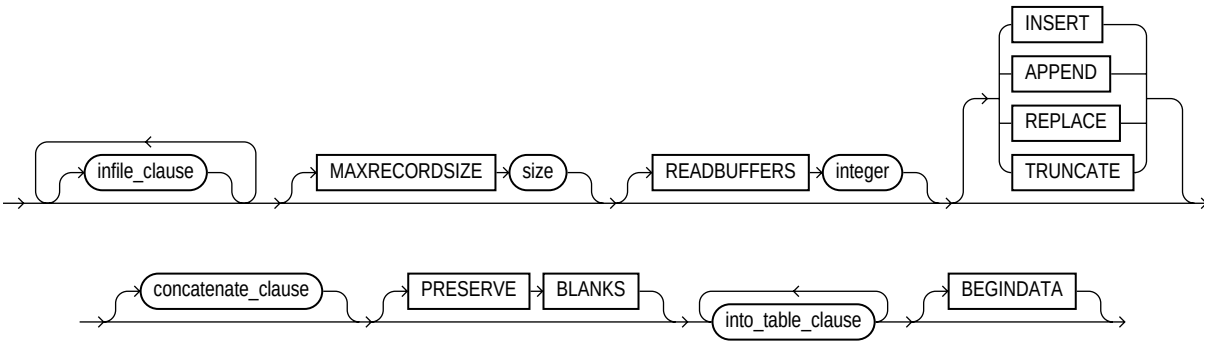
OPTIONS 句



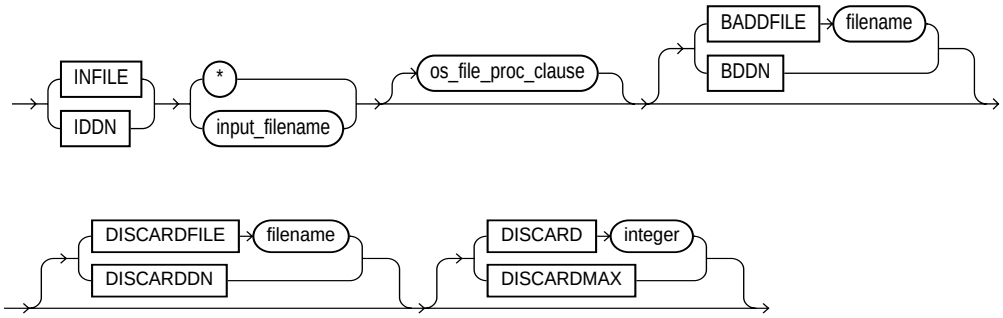
Load 文



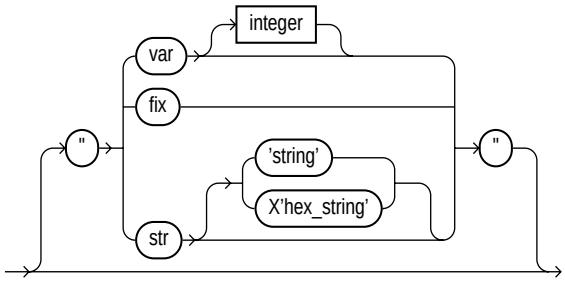
注意：キャラクタセットの指定は、制御ファイル内のデータには適用されません。



infile_clause

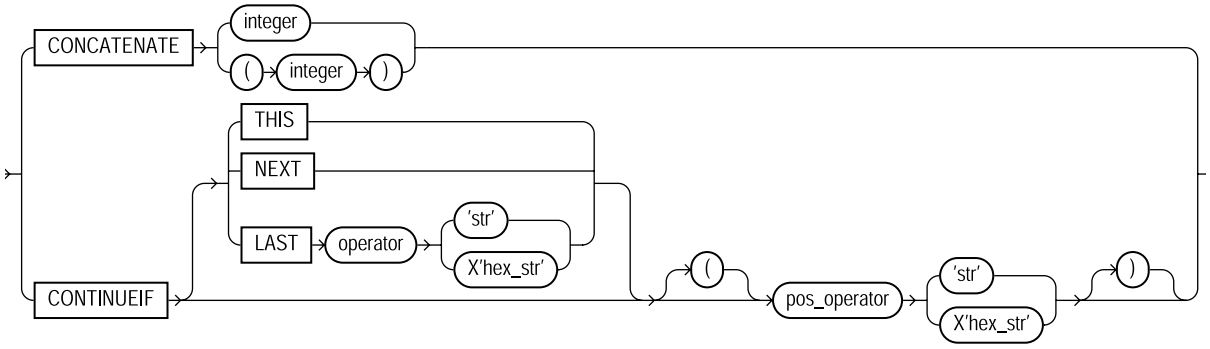


os_file_proc_clause

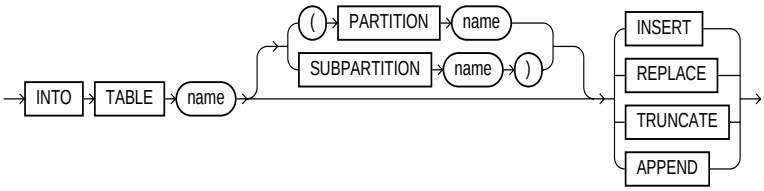


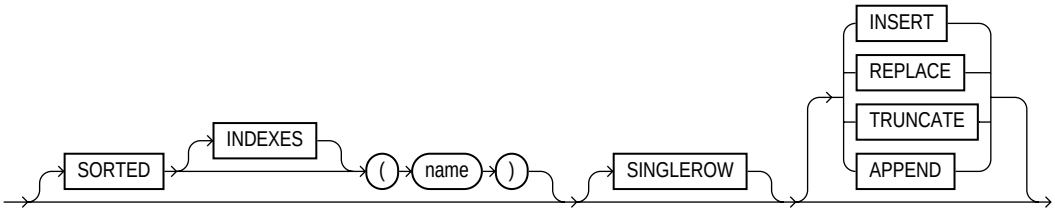
重要 : 前述の構文は、UNIX プラットフォームに固有です。ご使用のプラットフォームで必要な構文の詳細は、ご使用の Oracle オペレーティング・システム固有のドキュメントを参照してください。

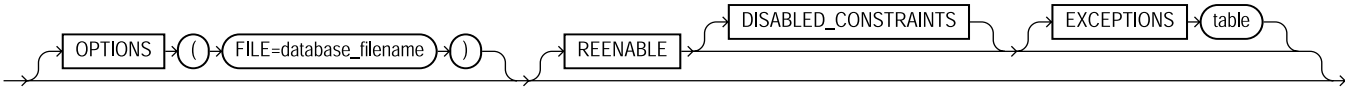
concatenate_clause

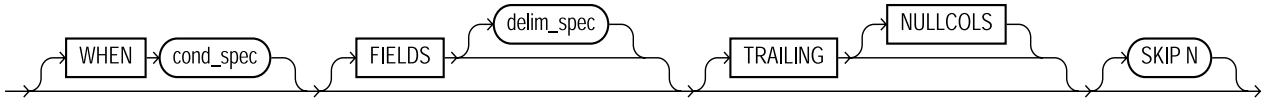


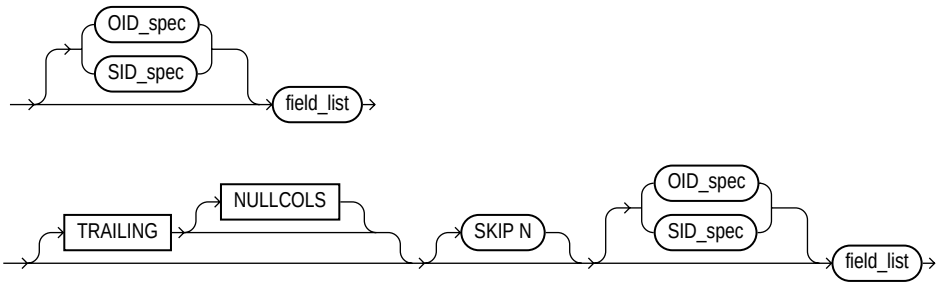
into_table_clause



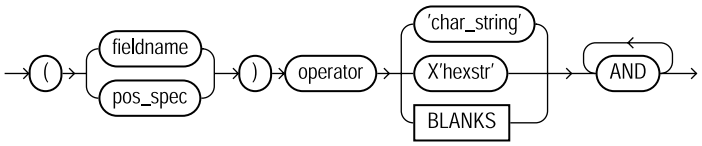




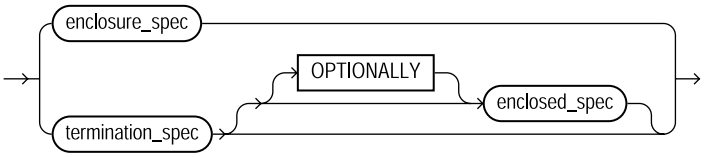




cond_spec



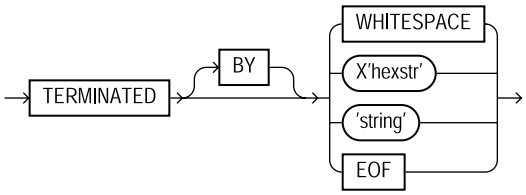
delim_spec



full_fieldname

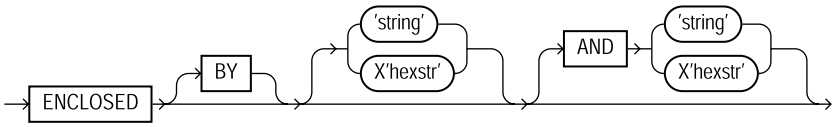
注意：`full_fieldname` には、ドット表記法を使用してフィールドのフルネームを指定してください。フィールド `col2` が列オブジェクト `col1` の属性の場合、指示句の中で `col2` を参照するときは、`col1.col2` と表記してください。同じエンティティを参照および命名している `column_name` および `full_fieldname` があっても、`column_name` には、エンティティのフルネームを指定できない（ドット表記法がない）ので、別のものとして認識されます。

termination_spec



注意 : LOBFILE からロードしたフィールドのみが、EOF によって終了できます。

enclosure_spec

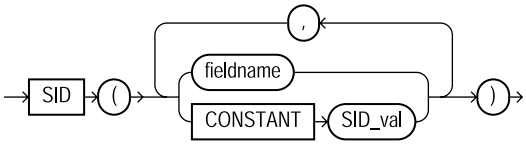


注意 : EOF によって終了するフィールドは、囲めません。

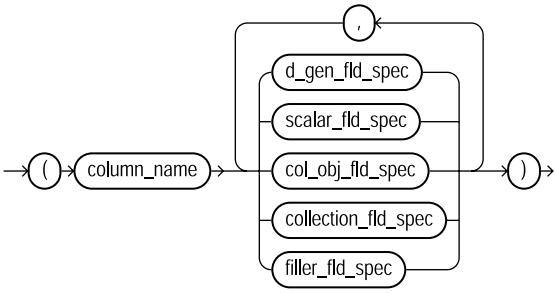
OID_spec

注意 : システム生成 OID のかわりに、主キー OID を使用している表では、OID 句を指定できません。

SID_spec

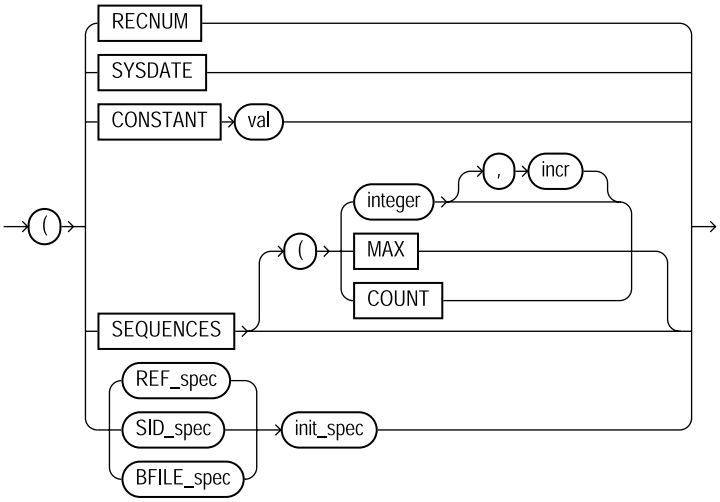


field_list

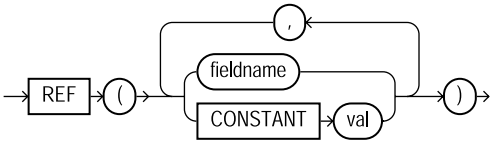


注意 : 同じエンティティを参照および命名している `column_name` および `fieldname` があっても、`column_name` には、エンティティのフルネームを指定できない（ドット表記法がない）ので、別のものとして認識されます。

d_gen_fld_spec



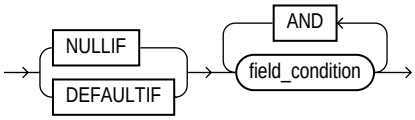
REF_spec



注意 :

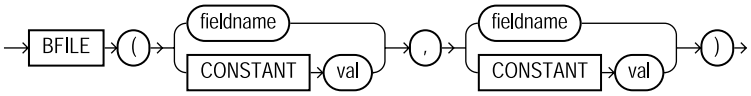
- REF 指示句の第 1 引数を表名とします。
- REF 列が主キー REF の場合、REF 指示句の引数の相対順序は、主キー REF を構成している、列の相対順序と一致させてください（つまり、オブジェクト表の主キー OID を構成している列の相対順序）。

init_spec



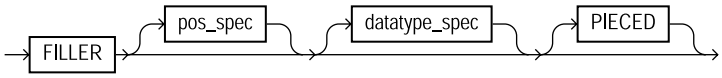
注意 : field_condition は、セカンダリ・データ・ファイル（SDF）のフィールドに基づくことができません。

BFILE_spec



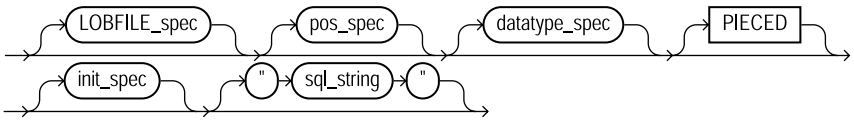
注意 : BFILE 指示句の第 1 引数には、DIRECTORY OBJECT（server_directory の別名）が含まれます。第 2 引数には、ファイル名が含まれます。

filler_fld_spec



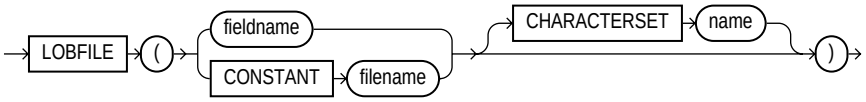
注意 : 従来型パス・ロードでは、必要に応じて分割されます。ダイレクト・パス・ロードでは、データは自動的に分割されてロードされるため、PIECED キーワードを指定する必要はありません。

scalar_fld_spec



注意 : 従来型パス・ロードでは、必要に応じて分割されます。ダイレクト・パス・ロードでは、データは自動的に分割されてロードされるため、PIECED キーワードを指定する必要はありません。また、LOB フィールド（LOBFILE_spec が指定されているかどうかにかかわらず）に `sql_string` は指定できません。

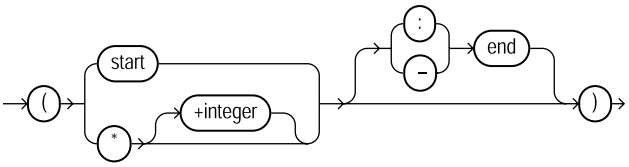
LOBFILE_spec



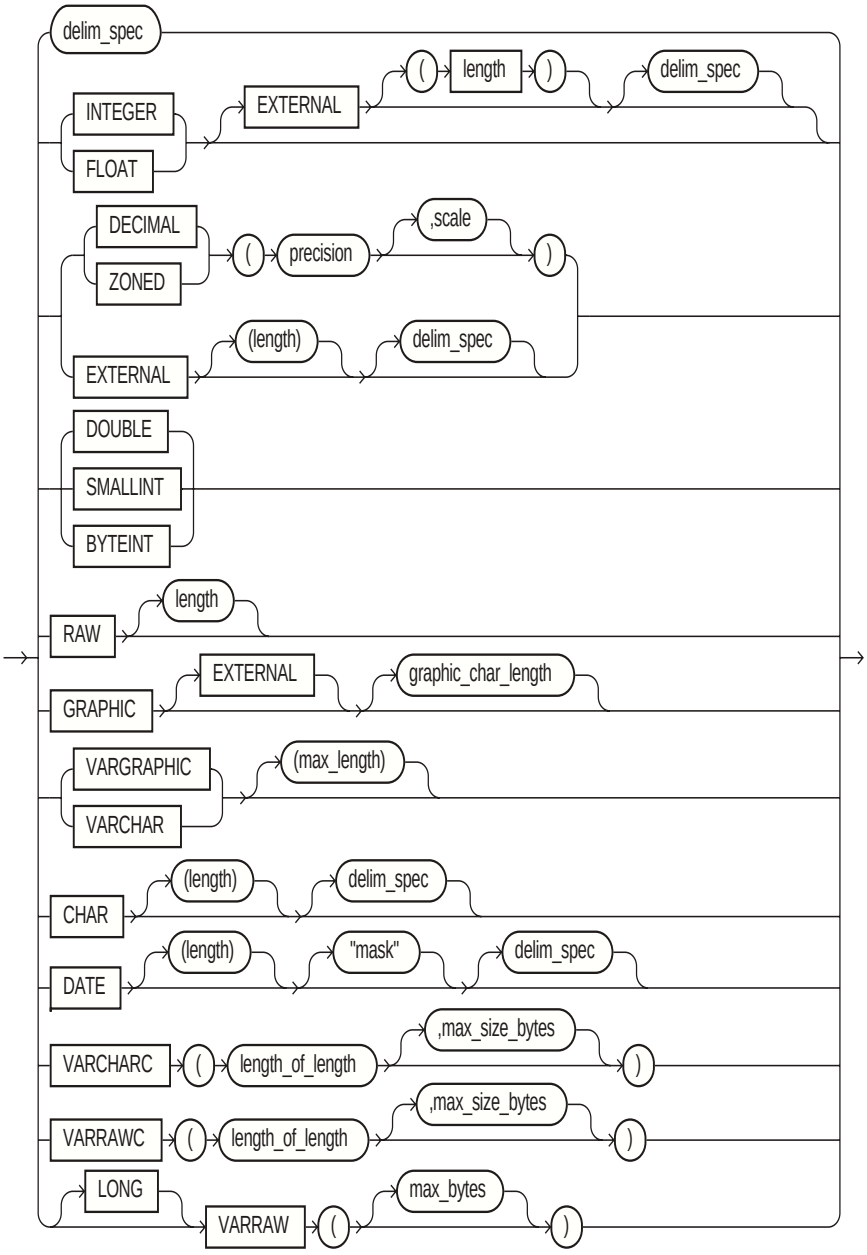
注意 :

- LOBFILE からデータをロードする場合、pos_spec は使用できません。
- LOBFILE からロードできるのは、LOB のみです。

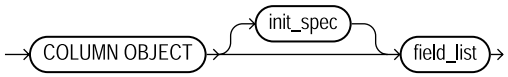
pos_spec



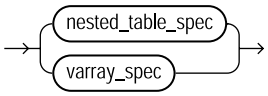
datatype_spec



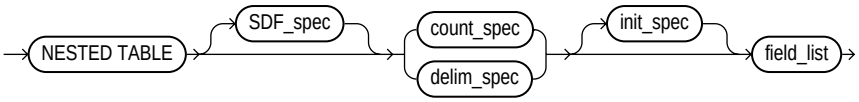
col_obj_fld_spec



collection_fld_spec

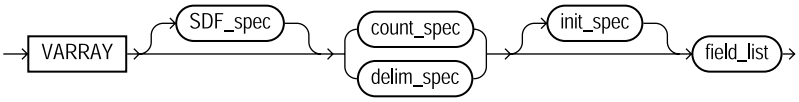


nested_table_spec



注意 : `field_list` には、`collection_fld_spec` を含めることはできません。

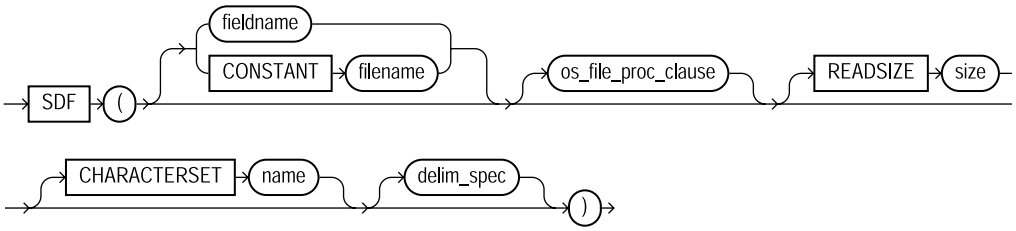
VARRAY_spec



注意 : VARRAY 内でネストする `col_obj_spec` 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、キーワード VARRAY を前に付けた `<column_name>` と同一である必要があります。

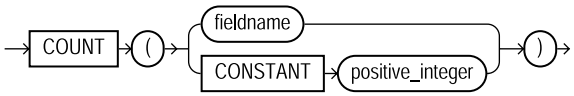
SDF_spec



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。
- delim_spec は、collection_fld_spec の field_list の一部として記述したすべてのフィールドにおいて、デフォルトのデリミタとして使用されます。

count_spec



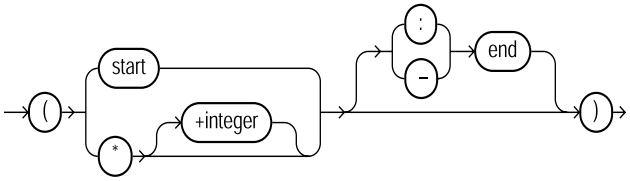
注意 : COUNT 句の引数としてフィールドを指定する場合、そのフィールドには、整数に変換可能なデータ・ファイルのデータをマップしてください（たとえば文字列 124 など）。

拡張された DDL 構文

位置指定

pos_spec

位置指定（*pos_spec*）はフィールドの開始位置を指定します。また、オプションとして終了位置も指定します。*pos_spec* の構文は次のとおりです。

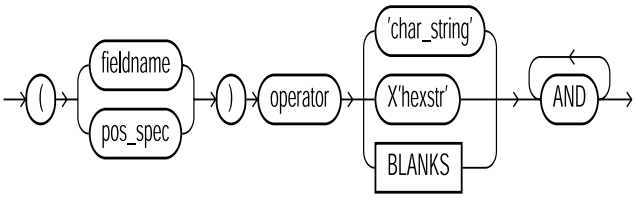


位置は小カッコで囲んでください。開始位置の指定は、列番号、*（次の列）または *+n（次の列にオフセット分を加算）の形で指定できます。*start* 位置と *end* 位置は、コロン（:）またはダッシュ（-）で区切ります。

フィールド条件

field_condition

フィールド条件は名前で指定されたフィールド、またはレコードの領域を、指定した値と比較します。条件が真として評価されると、指定された関数が実行されます。たとえば、条件が真の場合、NULLIF 関数が指定されていれば NULL データ値が挿入され、DEFAULTIF 関数が指定されていればデフォルト値が挿入されます。*field_condition* 構文は、次のとおりです。



char_string および *hex_string* の部分は、一重引用符で囲んでも、二重引用符で囲んでもかまいません。*hex_string* は 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。また、BLANKS キーワードを指定すると、そのフィールドがすべてブランクかどうかをテストできます。BLANKS の指定は、デリミタ付きのデータをロードする際にフィールド長が事前にわかっていないとき、またはブランク文字が複数あるマルチバイト・キャラクタ・セットを使用するときなどに必要となります。

演算子とその前後のオペランドの間には空白を入れないでください。たとえば次の例では、

```
(1)='x'
```

は正しい記述ですが、

```
(1) = 'x'
```

と記述するとエラーになります。

列名
column_name

フィールド条件に指定する列名には、入力レコード中に定義されている列名を指定してください。列名が予約語と同じ場合は、列名を二重引用符で囲む必要があります。詳細は、5-18 ページ「ファイル名とオブジェクト名の指定」を参照してください。

精度と長さ
precision
length

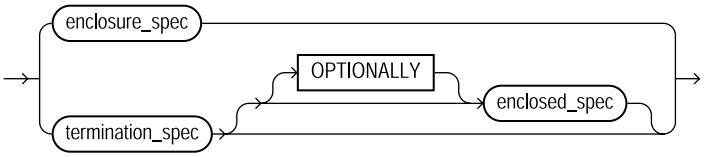
数値型フィールドにおける精度とは、そのフィールドに含まれる数字の桁数のことです。また、数値型フィールドの長さとは、レコード上でそのフィールドが占めるバイト数のことです。ゾーン 10 進数型フィールドの場合、精度とバイト長が同じ数字になります。これに対し、（packed）DECIMAL フィールドの場合は、精度を *p* とするとバイト長は（*p*+1）/ 2 を切り上げた値になります。これは、1 バイトに数字 2 桁（または数字と符号）が含まれるためです。

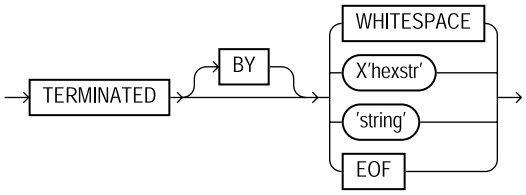
日付マスク

日付マスクは日付の値の形式を指定します。詳細は、5-64 ページ「DATE」の「データ型」を参照してください。

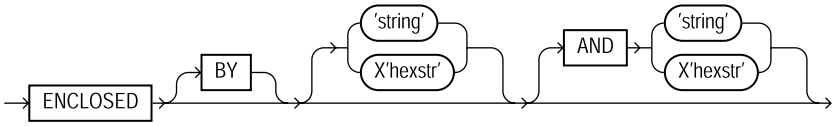
デリミタの指定
delimiter_spec

delimiter_spec は終了デリミタまたは囲みデリミタ、またはその 2 つの組合せを指定します。次に示すとおりです。





注意 : LOB ファイルからロードしたフィールドのみが、EOF によって終了できます。



注意 : EOF によって終了するフィールドは、囲めません。

詳細は、5-69 ページ「デリミタの指定」を参照してください。

制御ファイルの基礎

次の項では、SQL*Loader 制御ファイルの各種 DDL エントリおよびその機能について説明します。すべての文は、前の項に記載されているデータ定義言語の構文によって記述されます。制御ファイルの文については、制御ファイルに出現する順序で説明します。

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。たとえば、

```
--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。4-11 ページ付録「事例 3: 自由区分形式ファイルのロード」に、制御ファイルのコメントについて説明があります。

制御ファイル中でのコマンド行パラメータの指定

OPTIONS 文は、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 文は、LOAD DATA 文の前に置きます。

OPTIONS

OPTIONS パラメータを使用することにより、実行時の引数をコマンド行ではなく制御ファイル中で指定することができます。OPTIONS パラメータでは、次の引数を指定できます。これらの引数の詳細は、第 6 章の「SQL*Loader コマンド行リファレンス」を参照してください。

```
SKIP = n
LOAD = n
ERRORS = n
ROWS = n
BINDSIZE = n
SILENT = {FEEDBACK | ERRORS | DISCARDS | ALL}
DIRECT = {TRUE | FALSE}
PARALLEL = {TRUE | FALSE}
```

たとえば、次のように表示されます。

```
OPTIONS (BINDSIZE=100000, SILENT=(ERRORS, FEEDBACK) )
```

注意 : コマンド行で指定された値は、制御ファイルの OPTIONS 文で指定された値よりも優先されます。

ファイル名とオブジェクト名の指定

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、それらの標準規則に対する一部の例外と、特別な措置が必要な場合の SQL*Loader 制御ファイルのデータベース・オブジェクトおよびファイル名の指定方法について説明します。また、引用符で囲まれた文字列の中でのエスケープ文字の使用方法も説明します。

SQL および SQL*Loader の予約語と競合するファイル名

SQL および SQL*Loader の予約語は、二重引用符で囲んでください。列名の場合に、名前として使用されやすい予約語は次のとおりです。

```
COUNT     DATA     DATE     FORMAT
OPTIONS   PART     POSITION
```

PART、COUNT、DATA という列名を持つ在庫（inventory）システムを例にとった場合、これらの列名は SQL*Loader の制御ファイル中では、二重引用符の中で指定します。たとえば、次のように表示されます。

```
INTO TABLE inventory
(partnum   INTEGER,
"PART"     CHAR(15),
"COUNT"    INTEGER,
"DATA"     VARCHAR2(30))
```

予約語のリストは、付録 A の「SQL*Loader の予約」を参照してください。

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や、オブジェクト名に大文字と小文字の区別が必要な場合にも使用します。

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 文字列を使用すれば、SQL 演算子をデータ・フィールドに適用できます。詳細は、5-87 ページ「フィールドへの SQL 演算子の適用」を参照してください。

制限事項 制御ファイルのエントリは、BFILE、SID、OID または REF 指示句を使用する制御ファイルのフィールドに対して、SQL 文字列を指定できません。

SQL 文字列は、列オブジェクトまたはコレクションで、または列オブジェクトまたはコレクションの属性で使用できません。

オペレーティング・システムに関する考慮事項

完全パスの指定

特殊文字の使用によってオペレーティング・システム固有の非互換が起き、完全パス名を指定しようとしたときに問題が発生する場合があります。多くの場合、パス名を一重引用符で囲んで指定すれば、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。

バックスラッシュの使用

DDL 構文の場合、二重引用符の前にエスケープ文字であるバックスラッシュ（\）を付けることにより、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することができます（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）。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ときも、その前にバックスラッシュを付けます。

たとえば、`homedir\data"norm\myfile` 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つけることによって、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、バックスラッシュそれ自体を文字列中に表示できます。

たとえば、次のように使用します。

```
"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ないので、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。ファイル名およびファイル処理オプションの文字列です。したがって、通常は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、変換後に手作業で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・システム上でパス名の中のディレクトリを区切る文字としてバックスラッシュ（\）が使用されていて、かつそのオペレーティング・システム上で使用している Oracle バージョンでファイル名やその他の移植不能文字列の中でバックスラッシュをエスケープ文字として使用できる場合は、パス名の中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。

追加情報 : 必要な、または使用可能なエスケープ文字の詳細は、ご使用の Oracle 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。

エスケープ文字が使用できない場合

現在使用している Oracle バージョンでは、移植不能文字列に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です）。このとき前述の例は次のようになります。

```
INFILE 'topdir\mydir\myfile'
```

パス名は通常どおり指定します。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ので、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。

BEGINDATA を使用した、制御ファイルのデータの識別

ユーザーのデータが制御ファイル自体に含まれていて、個別のデータ・ファイルにはない場合は、次のロード構成設定を含めてください。

最初のデータ・レコードの前に、BEGINDATA キーワードを指定します。構文は次のとおりです。

```
BEGINDATA data
```

BEGINDATA は、INFILE * に指定することによって、5-22 ページで説明する INFILE キーワードとともに使用します。4-5 ページ「事例 1: 可変長データのロード」に例を示します。

注意 :

- 制御ファイルにデータが含まれているのに、BEGINDATA キーワードを省略すると、SQL*Loader が制御情報としてデータを解釈しようとするためにエラー・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 キーワードは使用できません。
- BEGINDATA を含む行がデータの先頭行と解釈されてしまうため、その同じ行に BEGINDATA 句として、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 の後にコメントを続けて記述しないでください。

INFILE: データ・ファイルの指定

INFILE キーワードを使用して、データ・ファイルまたはファイル処理オプション文字列が続くデータ・ファイルを指定します。指定するファイルが複数あるときは、INFILE キーワードを複数使用できます。コマンド行からデータ・ファイルを指定する場合は、6-3 ページ「コマンド行キーワード」で説明している DATA パラメータを使用します。

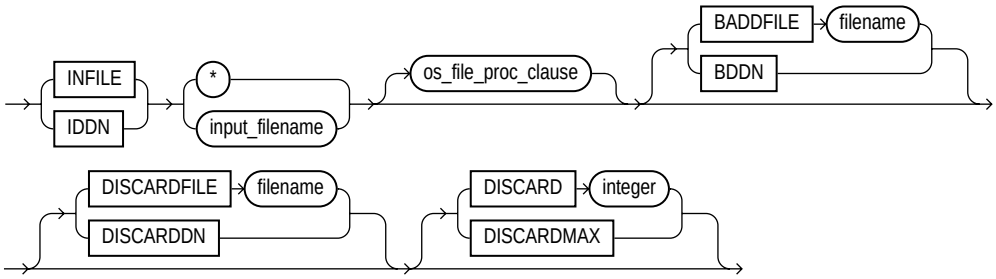
注意 : コマンド行パラメータ DATA は、制御ファイル内の INFILE キーワードを上書きします。

ファイルの命名

ロードするデータ・ファイルを指定するには、INFILE キーワードにファイル名を続け、必要なときはその後にファイル処理オプションのための記述を続けます。コマンド行でファイル名を指定すると、制御ファイルの最初の INFILE キーワードが上書きされることを覚えておいてください。ファイル名が指定されない場合、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。この指定については、5-21 ページ「BEGINDATA を使用した、制御ファイルのデータの識別」を参照してください。

注意 : IDDN は、DB2 との互換性を保持しています。



INFILE または INDDN	（DB2 互換が必要な場合は、INDDN を使用します。）このキーワードは、次に続くデータ・ファイルを指定します。
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には、一重引用符で囲む必要があります。5-18 ページ「ファイル名とオブジェクト名の指定」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。制御ファイルとデータ・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。
processing_options	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。5-24 ページ「データ・ファイル形式およびバッファリングの指定」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ったファイル処理オプション文字列で指定されていたり、3 番目のファイルが制御ファイル中のデータから構成されていても実行することができます。

データ・ファイルごとに、廃棄ファイルと不良ファイルを指定することもできます。ただし、個別の不良ファイルおよび廃棄ファイルは、各データ・ファイル名の次に宣言する必要があります。次に、4 つのデータ・ファイルをそれぞれの不良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、引用して示します。

```
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、必要があれば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、そのファイル名と拡張子はデフォルトが使用されます。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。
- MYDAT4.DAT では、必要な場合に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、必要な場合にデフォルトの不良ファイルが作成されます。さらに、DISCARDMAX オプションが指定されているので、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、必要に応じて廃棄ファイルがデフォルト名（`mydat4.dsc`）で作成されます。

例

データが制御ファイル中にある場合

```
INFILE  *
```

データがデフォルト拡張子が .dat である WHIRL ファイルにある場合

```
INFILE  WHIRL
```

データが、datafile.dat にある場合 : フルパス指定

```
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には、ファイル名を一重引用符で囲みます。ファイル名指定の詳細は、5-18 ページ「ファイル名とオブジェクト名の指定」を参照してください。

READBUFFERS の指定

READBUFFERS キーワードはメモリー使用量を制限します。このキーワードは、ダイレクト・パス・ロードでのみ使用できます。詳細は、8-16 ページ「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。

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、制御ファイルの処理に対して、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を制御ファイルに指定できます。この文字列を使用して、ファイル形式およびバッファリングを指定します。

> **追加情報 :** 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 ドキュメントを参照してください。

ファイル処理の例

たとえば、使用しているオペレーティング・システムではオプション文字列の構文が次のように定められていると仮定します。

ここで 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、SQL*Loader が 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"
```

> **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。詳細は、5-18 ページ「ファイル名とオブジェクト名の指定」を参照してください。

BADFILE: 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。

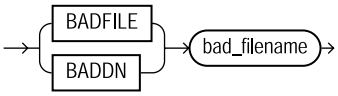
- 1 つ以上のレコードが拒否された場合、不良ファイルが作成される。
- 拒否されたレコードがない場合、不良ファイルは作成されない。この場合は、次の実行のために不良ファイルを再初期化する必要があります。
- 不良ファイルが作成される場合、同じ名前の既存のファイルは上書きされる。そのため、残しておきたいファイルが上書きされないようにする必要があります。

追加情報 : 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 ドキュメントで確認してください。

不良ファイルの名前を指定するときは、BADFILE キーワード（または DB2 互換性のために BADDN キーワード）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。コマンド行から不良ファイルを指定する場合は、6-3 ページ「コマンド行キーワード」で説明している BAD パラメータを使用します。

コマンド行から指定した不良ファイル名は、制御ファイル中で最初に現れる 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、コマンド行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、データ修正をした後そのまま再ロードすることができます。構文は次のとおりです。



BADFILE または BADDN	（DB2 互換が必要な場合は、BADDN を使用します。）このキーワードは、その後に不良ファイルのファイル名が指定されることを示します。
bad_filename	使用するプラットフォームに有効な任意のファイル名。 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には、一重引用符で囲む必要があります。5-18 ページ「ファイル名とオブジェクト名の指定」を参照してください。

例

不良ファイルの名前を UGH とし、ファイル拡張子（またはファイル・タイプ）をデフォルトの .bad とする場合、次のように指定します。

```
BADFILE UGH
```

次に、不良ファイルの名前を BAD0001 とし、ファイル拡張子（またはファイル・タイプ）を .rej とする場合の指定例を示します。

```
BADFILE BAD0001.REJ
BADFILE '/REJECT_DIR/BAD0001.REJ'
```

拒否レコード

次の条件のいずれかが満たされると、レコードはロードを拒否されます。

- レコードの挿入時に Oracle エラーが発生する（指定されたデータ型と実際のデータが適合しない場合など）。
- レコード中のデータがロード可能かどうかを SQL*Loader で判断できない。レコード中のフィールドに終了デリミタが欠落している場合などのように、WHEN 句の指定条件を満たすかどうかを判断できない場合です。

デリミタの指定が不正であっても、WHEN 句の指定条件に基づいてデータを評価できる場合は、条件判断を行ってデータを挿入または拒否します。

挿入時にレコードが拒否されると、そのレコード中のデータは、どの表に対してもいっさい挿入されません。たとえば、1 レコード中のデータが複数の表に挿入される場合、ほとんどの表についてデータが正常に挿入されたが 1 つの表への挿入が失敗したとします。その時点でそのレコードから挿入されたデータはすべてロールバックされ、挿入前の状態に戻ります。このレコードは不良ファイルに出力されるので、データを修正した後で再びロードすることができます。エラーが発生する前のレコードは、ロールバックの影響を受けません。

ログ・ファイルには、拒否された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。4-15 ページ「事例 4: 結合された物理レコードのロード」の、拒否されたレコードの例を参照してください。

注意 : 複数表のロードで、1 つの表から拒否された 1 行がある場合、他のすべての表からも拒否されます。これによって、不良ファイルで行が修復された後、すべての表に再ロードしたときの一貫性が保証されます。また、表に行をロードするときは、まだロードしていない他のすべての表に対してもロードします。そうしないと、不良ファイルで修復した行を再ロードしたときに、一部の表に対してデータが 2 回ロードされることになります。

このため、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、行のロードが続行され、すでに表にロードされた有効な行が確実にすべての表にロードされるようになります。また、拒否された行がすべての表から取り除かれるようになります。

LOB ファイルとセカンダリ・データ・ファイル LOB ファイルまたはセカンダリ・データ・ファイルのデータは、拒否された行があっても、不良ファイルには書き出されません。LOB のロードにエラーがある場合、その行は、拒否されるのではなく、LOB フィールドが空のまま（NULL ではなく長さが 0）になります。

廃棄ファイルの指定

SQL*Loader を実行すると、ロード条件をまったく満たさないレコードがあるときに、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 句の指定をまったく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、拒否されたレコードとは異なります。廃棄レコードには必ず不良データがあるとは限りません。また、廃棄レコードに関しては、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 句の条件に満たない場合。（廃棄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。）
- 廃棄レコードがない場合、廃棄ファイルは作成されない。

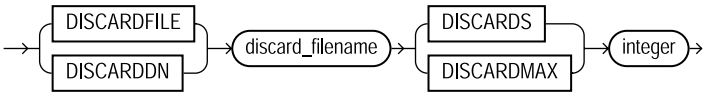
廃棄ファイルを生成するには、次のいずれかの構文を使用します。

制御ファイルで	コマンド行で
DISCARDFILE *filename*	DISCARD
DISCARDDN *filename*	DISCARDMAX
DISCARDS	
DISCARDMAX	

このように、名前を指定するパラメータを使用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。

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 DISCARDDN（DB2 互換用）キーワードの後にファイル名を指定します。



DISCARDFILE または DISCARDDN	（DB2 互換が必要な場合は、DISCARDDN を使用します。）このキーワードは、その後に廃棄ファイルのファイル名が指定されることを示します。
discard_filename	使用するプラットフォームに有効な任意のファイル名。 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。5-18 ページ「ファイル名とオブジェクト名の指定」を参照してください。

デフォルトのファイル名は、データ・ファイル名にファイル拡張子（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行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、コマンド行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。

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。したがって、WHEN 句の変更やデータの修正を行った後で、既存の制御ファイルを使用して廃棄ファイル内のレコードをそのまま再ロードできます。

例

廃棄ファイルの名前を CIRCULAR とし、ファイル拡張子（またはファイル・タイプ）をデフォルトの .dsc とする場合、次のように指定します。

```
DISCARDFILE  CIRCULAR
```

廃棄ファイル名が notappl で、ファイル拡張子（またはファイル・タイプ）を .may とする場合、次のように指定します。

```
DISCARDFILE NOTAPPL.MAY
```

次の例では、廃棄ファイル forget.me をフルパスで指定しています。

```
DISCARDFILE  '/DISCARD_DIR/FORGET.ME'
```

廃棄レコード

レコードに対して INTO TABLE キーワード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 INTO TABLE キーワードがすべて WHEN 句を持っていて、レコードがどの条件にも該当しない、またはすべてのフィールドが NULL であるときに発生します。

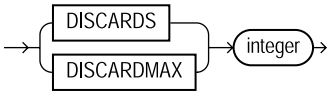
INTO TABLE キーワードが WHEN 句なしで指定されているときは、レコードは廃棄されません。すべてのレコードに関して、指定の表への挿入が試みられます。このときに、レコードが受付けを拒否されることはありますが、廃棄されることはありません。

4-15 ページ「事例 4: 結合された物理レコードのロード」にある廃棄ファイルの使用例を参照してください。

LOB ファイルとセカンダリ・データ・ファイル LOB ファイルまたはセカンダリ・データ・ファイルのデータは、廃棄された行があっても、廃棄ファイルには書き出されません。

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指定できます。このためには、次のような句を使用します。



ここで *n* は整数です。廃棄レコード数がこの数に達すると、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った数の上限を設定できます。ただし、廃棄数を 1 回しか指定しないと、指定した廃棄上限値はすべてのファイルについて共通となります。

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。4-15 ページ「事例 4: 結合された物理レコードのロード」にある例を参照してください。

コマンド行パラメータの使用方法

コマンド行から廃棄ファイルを指定できます。その場合、6-3 ページ「コマンド行キーワード」で説明しているパラメータ DISCARDFILE を使用します。

コマンド行で指定したファイル名で、制御ファイルに指定した不良ファイルが上書きされます。

異なる文字コード体系の処理

SQL*Loader では、異なる文字コード体系（キャラクタ・セットまたはコード・ページと呼ばれる）がサポートされます。SQL*Loader では、Oracle の NLS（各国語サポート）機能を使用して、さまざまな文字コード体系（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。サポートしている文字コード体系の詳細は、『Oracle8*i* NLS ガイド』を参照してください。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・セットはアジアの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（フィールド、表など）をマルチバイト文字で指定できます。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。

入力文字変換

SQL*Loader 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あります。

従来型パス・ロードの場合、データは、そのセッションに対して NLS_LANG 初期化パラメータで指定されているセッション・キャラクタ・セットに変換されます。変換されたデータは、SQL INSERT 文に従ってロードされます。ここで、セッション・キャラクタ・セットとは、使用する端末でサポートされているキャラクタ・セットです。

ダイレクト・パス・ロードの場合、データは、データベース・キャラクタ・セットに直接変換されます。したがって、ダイレクト・パスを使用すれば、使用する端末でサポートされていないキャラクタ・セットのデータをロードできます。

注意 : データ変換を行う際、ターゲット・キャラクタ・セットには、データ中に存在するすべての文字に相当する文字が含まれている必要があります。そうしないと、ターゲット・キャラクタ・セット中に等しい文字を持たない文字はデフォルト文字に変換され、結果としてデータの損失となります。

ダイレクト・パスを使用する際、データベース・キャラクタ・セットはデータ・ファイル・キャラクタ・セットと等しいか、そのスーパーセットである必要があります。同様に、従来型パスを使用する際、セッション・キャラクタ・セットはデータ・ファイル・キャラクタ・セットと等しいか、そのスーパーセットである必要があります。

各入力ファイルで使用されるキャラクタ・セットは、CHARACTERSET キーワードで指定します。

CHARACTERSET キーワード

CHARACTERSET キーワードを使用して SQL*Loader に、それぞれのデータ・ファイルでどのキャラクタ・セットを使用するかを指定します。データ・ファイルごとに、異なるキャラクタ・セットを指定できます。ただし、各データ・ファイルに指定できるのは、1 つのキャラクタ・セットのみです。

CHARACTERSET キーワードを指定すると、文字データはデータベースへのロードの際に自動的に変換されます。この文字変換の対象となるのは CHAR 型、DATE 型および numeric EXTERNAL 型のフィールドのみです。CHARACTERSET キーワードが指定されない場合、変換は行われません。

CHARACTERSET の構文は次のとおりです。

```
CHARACTERSET character_set_spec
```

ここで、*character_set_spec* には、使用する特定のコード体系を Oracle 専用の文字列で指定します。

> **追加情報 :** サポートされているキャラクタ・セット、コード・ページおよび NLS_LANG パラメータの詳細は、『Oracle8*i* NLS ガイド』を参照してください。

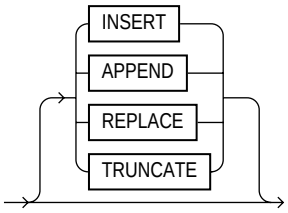
制御ファイル・キャラクタ・セット

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。しかし、デリミタや比較句の値は、データ・ファイルで使用しているキャラクタ・セットに一致するように指定する必要があります。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。

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となっています。異なるキャラクタ・セットでデータを記述する場合は、データを別のファイルに入れてください。

空および空でない表へのデータのロード

表をロードするには、次の方法のうちの 1 つを指定します。



空の表へのロード

ロード先の表が空の場合は、INSERT オプションを使用します。

空でない表へのロード

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

警告 : REPLACE または TRUNCATE キーワードが指定される場合、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やそれに相当するユーティリティで保存しておかない限り、回復することは不可能です。

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「DB2 / DXT ユーザーに対する注意事項」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 は新しい行を表に追加します。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。4-11 ページ「事例 3: 自由区分形式ファイルのロード」にある例を参照してください。

REPLACE

表の既存の行はすべて削除され、新規に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であるか、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。4-15 ページ「事例 4: 結合された物理レコードのロード」にある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あれば起動します。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、『Oracle8*i* 概要』の「データ整合性」の章を参照してください。

既存の行の更新

REPLACE は、個々の行を置換するの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する。
2. 相関副問合せを指定した、SQL 言語の UPDATE 文を使用する。
3. 作業表を削除する。

詳細は、『Oracle8*i* SQL リファレンス』の「UPDATE 文」を参照してください。

TRUNCATE

この方法が指定された場合、SQL*Loader は SQL TRUNCATE コマンドを使用して最大限の処理パフォーマンスを実現します。TRUNCATE コマンドを機能させるには、まず表の参照整合性制約を使用禁止にしてください。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してください。

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 DELETE ANY TABLE 権限を持っている必要があります。

注意 :

SQL TRUNCATE オプションとは異なり、この方法では、表のエクステントが再利用されます。

INSERT 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、実行が終了します。4-5 ページ「事例 1: 可変長データのロード」にある例を参照してください。

ロード中断後の継続処理

ロード実行中にデータ行や索引エントリの領域が足りなくなると、ロード処理は中断されます（表が最大エクステントの値に達したときなど）。表に新たに領域を作成すると、再びロード可能な状態になります。

表と索引の状態

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ダイレクト・ロード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。これ以外の索引は、他にエラーが発生していない限り、有効です。（索引がダイレクト・ロード状態のままとなるその他の理由については、8-11 ページ索引使用禁止状態（Index Unusable）のままの索引を参照してください。）

ログ・ファイルの使用

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ています。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。

索引の削除

ダイレクト・パス・ロードを継続する前に、SQL*Loader ログ・ファイルを調べて、ダイレクト・ロード状態になっている索引がないことを確認してください。ダイレクト・ロード状態のままの索引は、ロードを継続する前に削除する必要があります。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。

単一表へのロード継続

1 つの表のみをロードする場合、中断されたロード（ダイレクト・パス・ロードまたは従来型パス・ロード）を継続するには、コマンド行パラメータ SKIP を使用して、スキップする論理レコード数を指定します。SQL*Loader ログ・ファイルに、すでに読み込まれたレコード数が 345 件と記録されている場合、次のようなコマンドを入力して処理を継続します。

```
SQLLDR USERID=scott/tiger CONTROL=FAST1.CTL DIRECT=TRUE SKIP=345
```

複数表へのロード継続（従来型ロード）

複数の表に対して従来型パス・ロードを実行した場合、それぞれの表について同期がとれなくなることはありません。したがって、複数表に対するロード処理を継続する際にも、コマンド行から SKIP パラメータを指定できます。前の段落で説明した、単表へのロードを継続する場合と同じ手順でパラメータを指定してください。

複数表へのロード継続（ダイレクト・ロード）

ダイレクト・パスによる複数表へのロードが中断された場合、それまでに処理された論理レコードの件数が、表によって異なることがあります。このような場合、表の同期がとられていないので、ロードの継続操作は多少複雑になります。

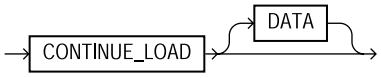
複数表への中断されたダイレクト・パス・ロードを再開するには、まず SQL*Loader ログ・ファイルを調べて、各表にそれぞれ何件のレコードがロードされたかを確認します。ロードされたレコード数がすべて等しい場合は、前述の簡単な指定により処理を継続します。

CONTINUE_LOAD ロードされたレコード数が表ごとに異なっている場合は、CONTINUE_LOAD キーワードを使用してロード・レベルではなく表レベルで SKIP を指定します。これらの文は、中断されて同期がとれていないロードを再開する場合に指定します。

通常は、制御ファイルの先頭で、

```
LOAD DATA...
```

と指定しますが、この文のかわりに、次のように記述します。



SKIP 次に、SKIP キーワードを使用して、各表に関してスキップする論理レコード数を、それぞれ対応する INTO TABLE 句に指定します。

```
...
INTO TABLE emp
SKIP 2345
...
INTO TABLE dept
SKIP 514
...
```

SKIP と CONTINUE_LOAD の組合せ

CONTINUE_LOAD キーワードを指定する必要があるのは、ダイレクト・ロードが失敗した場合のみです。従来型パスの場合、複数の表へロードするときに、同期がとれなくなることはありません。

CONTINUE_LOAD を指定した場合、コマンド行からは SKIP パラメータを指定できません。この場合は、表レベルの SKIP 句を使用する必要があります。LOAD を指定した場合は、コマンド行からでも SKIP パラメータを指定できますが、このときは表レベルの SKIP 句は使用できません。

物理レコードからの論理レコードの作成

Oracle8*i* は、64K より大きい（6-7 ページ「READSIZE（読込みバッファ）」を参照）ユーザー定義レコードをサポートしているので、物理レコードにする際に論理レコードを断片化する必要性が削減されます。ただし、断片化が完全になくなるわけではありません。

複数の物理レコードから 1 つの論理レコードを作成するには、次の 2 つの句のうちのいずれかを指定します。どちらを指定するかは、対象とするデータによって異なります。

```
CONCATENATE
CONTINUEIF
```

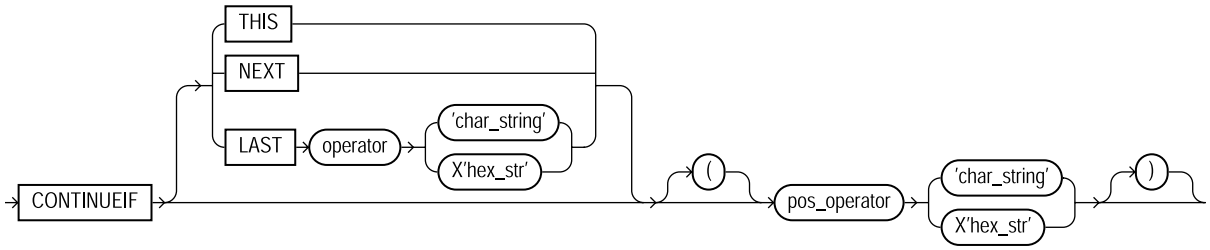
CONCATENATE は、1 つの論理レコードを構成するために結合する物理レコードの数が、常に一定の場合に使用します。

構文は次のとおりです。

```
CONCATENATE  n
```

ここで *n* は 1 つの論理レコードを構成するために結合される物理レコード数です。

結合される物理レコード数が処理中に変化する場合は、CONTINUEIF を使用する必要があります。CONTINUEIF キーワードを使用する場合は、後ろ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 文字目にシャープ記号（#）がある場合に、2 つのレコードを結合するとします。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。下記に CONTINUEIF の完全な構文を示します。この構文にあるように、CONTINUEIF ではさらに柔軟な処理を実行できます。



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。
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、それにより決まる継続フィールドの長さと指定されたバイト列または文字列の長さが異なる場合は、短い方に不足分を埋めるための文字が追加されます。文字列の場合はブランクが追加され、16 進数のバイト列の場合はゼロが追加されます。
LAST	レコードの最後の文字（ブランク以外）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（ブランク以外）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。
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。
char_string	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-string'	16 進形式で指定された文字列で、用途は上記の char_string 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。

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。物理レコード中の文字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。

CONTINUEIF THIS および CONTINUEIF NEXT が指定された場合は、すべての物理レコードから継続フィールドを取り除いたものを結合して、論理レコードを構成します。このため、データの値を構成するレコードとレコード間に、余分な文字（継続フィールドの文字）が残りません。例として、次のように CONTINUEIF THIS と CONTINUE NEXT が指定された場合を考えます。

```
CONTINUEIF THIS
CONTINUEIF NEXT
(1:2) = '%%'  (1:2) ='%%'
```

ここで、物理データ・レコードは下記のような 12 文字のレコードとします。また、ピリオドは空白を表します。

```
%%aaaaaaaa......aaaaaaaa....
%%bbbbbbbb....%%bbbbbbbb....
..cccccccc....%%cccccccc....
%%dddddddddd....dddddddddd..
%%eeeeeeeeee..%%eeeeeeeeee..
..ffffffffff..%%ffffffffff..
```

この場合、構成される論理レコードはどちらも同じになります。

```
aaaaaaaa....bbbbbbbb....cccccccc....
dddddddddd..eeeeeeeeee..ffffffffff..
```

注意：

- CONTINUEIF LAST は、CONTINUEIF THIS や CONTINUEIF NEXT とは異なります。CONTINUEIF LAST では、継続文字は物理レコードから除去されません。論理レコードが構成されるときにもこの文字が含まれます。
- 物理レコードにおける後続の空白は、論理レコードの一部になります。
- セカンダリ・データ・ファイル（SDF）のレコードを断片化して複数の物理レコードに入れることはできません。

CONTINUEIF の使用

最初の例では、現在の物理レコード（レコード 1）の 1 列目がアスタリスクのときについて指定します。この場合、次の物理レコード（レコード 2）をレコード 1 に連結します。レコード 2 の 1 列目もアスタリスクのときは、同様にレコード 3 がレコード 2 の後に連結され、以降同様の処理が繰り返されます。

レコード 2 の 1 列目がアスタリスクでないときは、レコード 2 はレコード 1 の後に連結されますが、レコード 3 は次の新しい論理レコードの先頭となります。

```
CONTINUEIF THIS (1) = "*"
```

次の例では、現在の物理レコード（レコード 1）のブランクでない最後の列がカンマかどうかを評価する CONTINUEIF 文を指定します。この条件が真であれば、次の物理レコード（レコード 2）はレコード 1 に連結されます。レコードの最後の文字がカンマでなければ、そのレコードが現在の論理レコードの最後の物理レコードとなります。

```
CONTINUEIF LAST = ","
```

最後の例では、次の物理レコード（レコード 2）の 7 列目と 8 列目が "10" であるときについて指定します。この場合、レコード 2 を直前の物理レコード（レコード 1）に連結します。レコードの 7 列目と 8 列目が "10" でない場合は、レコード 2 が新しい論理レコードの先頭となります。

```
CONTINUEIF NEXT (7:8) = '10'
```

4-15 ページ「事例 4: 結合された物理レコードのロード」にある CINTINUEIF 句の使用例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
- レコード中の列のデフォルト特性の指定方法

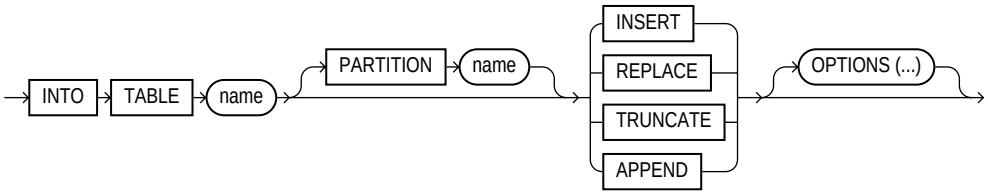
表名の指定

LOAD DATA 文の INTO TABLE キーワードを使用して表、フィールドおよびデータ型を識別できます。INTO TABLE キーワードは、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

INTO TABLE キーワードが持つさまざまな機能の一つに、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ときは、それぞれについて 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、INTO TABLE キーワードの次に、データを受け取る Oracle の表名を指定します。



ここでは、すでに存在する表を指定してください。表名が SQL や SQL*Loader のキーワードと同じ名前の場合、または表名に特殊文字が含まれる場合、表名の中の大文字と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."COMMENT"
INTO TABLE SCOTT."Comment"
INTO TABLE SCOTT."-COMMENT"
```

SQL*Loader を実行するユーザーは、その表に対する INSERT 権限が必要です。この権限がない場合は、この表の所有者のユーザー名を、次のように表名の前に付けて表を指定します。

```
INTO TABLE SOPHIA.EMP
```

表固有のロード方法

INTO TABLE 句では、表ごとにロード方法（INSERT、APPEND、REPLACE、TRUNCATE）を指定できます。表ごとに指定されたロード方法は、それぞれの表にのみ適用されます。INTO TABLE 句の中で指定されたロード方法は、グローバルに指定された表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ければ、グローバルの表ロード方法は、デフォルトで INSERT となります。これらのオプションの詳細は、5-32 ページ「空および空でない表へのデータのロード」を参照してください。

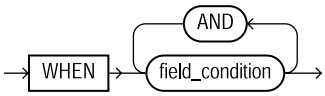
表固有の OPTIONS キーワード

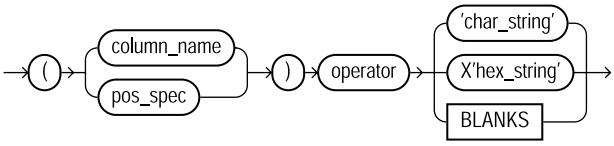
パラレル・ロードでは、個々の表に対して OPTIONS キーワードを指定できます。（OPTIONS キーワードは、パラレル・ロードの場合のみ有効です。）詳細は、8-26 ページ「パラレル・データ・ロード・モデル」を参照してください。

ロードする行の選択

WHEN 句を使用すれば、レコード中の条件によってその論理レコードをロードするか廃棄するかを決めることができます。

WHEN 句は表名の後に記述します。WHEN の後に、フィールド条件を 1 つ以上指定します。





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 を使用することにより、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、複数の条件を設定しているときはあいまいさを避けるために必ず使用してください。例を示します。

```
WHEN (DEPTNO = '10') AND (JOB = 'SALES')
```

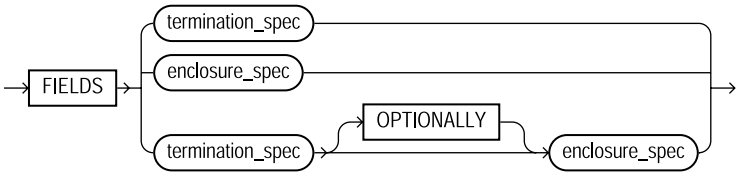
WHEN 句を評価する際、SQL*Loader によって、まず比較対象レコードのすべてのフィールドの値が調べられます。次に、WHEN 句が評価されます。WHEN 句の条件が真となる場合のみ、表に行が挿入されます。

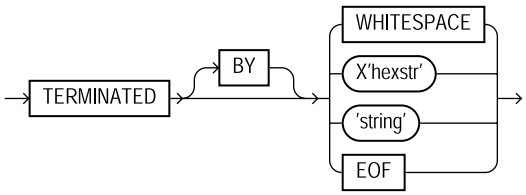
フィールド条件の詳細は、5-44 ページ「フィールド条件の指定」を参照してください。4-19 ページ「事例 5: 複数表へのデータのロード」にある WHEN 句の使用例を参照してください。

LOB ファイルとセカンダリ・データ・ファイルでの WHEN 句の使用 WHEN 指示句でレコードがエラーになった場合、そのレコードは廃棄（スキップ）されます。また、この場合、スキップされたレコードはメイン・データ・ファイルに含まれていると想定されるため、セカンダリ・データ・ファイルには影響はありません。

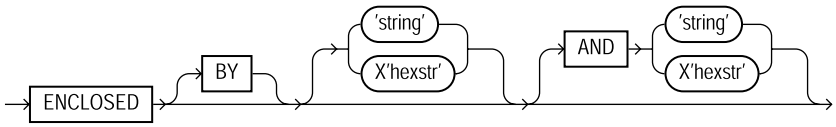
デフォルトのデータ・デリミタ（区切り記号）の指定

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。構文は次のとおりです。





注意 : 終了記号は、シングル・キャラクタに限らない文字列です。また、TERMINATED BY EOF は、LOBFILE からの LOB のロードにのみ適用されます。



注意 : 囲み文字列には、シングル・キャラクタはありません。

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。4-11 ページ「事例 3: 自由区分形式ファイルのロード」にある例を参照してください。デリミタの指定の詳細は、5-69 ページ「デリミタの指定」を参照してください。

ショート・レコードによるデータの欠落についての処理

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を 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 や LOC のように）相対位置で定義されていて、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てしまった場合は、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 句を指定しているかどうかによって決まります。

TRAILING NULLCOLS

TRAILING NULLCOLS 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ときに、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
```

これを次の制御ファイルで読み込みます。

```
INTO TABLE dept
    TRAILING NULLCOLS
( deptno CHAR TERMINATED BY " ",
  dname  CHAR TERMINATED BY WHITESPACE,
  loc    CHAR TERMINATED BY WHITESPACE
)
```

このレコードは DNAME の後で終了しています。したがって、残りの LOC フィールドには NULL 値が設定されます。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。

4-28 ページ「事例 7: 書式化されたレポートからのデータの抽出」の TRAILING NULLCOLS の例を参照してください。

索引オプション

この項では、索引エントリを作成する方法を制御する SQL*Loader オプションについて説明します。

SORTED INDEXES オプション

SORTED INDEXES オプションはダイレクト・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 により認識されるので、ロード時には SQL*Loader のパフォーマンスが最適化されます。この機能の構文は、5-4 ページ「高水準の構文図」で説明しています。詳細は、8-17 ページ「SORTED INDEXES 文」を参照してください。

SINGLEROW オプション

SINGLEROW オプションは、限られたメモリーでダイレクト・パス・ロードを APPEND オプションで実行する場合、または少数の行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 行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領域に置かれ、ロード終了時点で元の索引とマージされます。この方法をとると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、記憶領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用領域が同時に記憶領域を占有します。

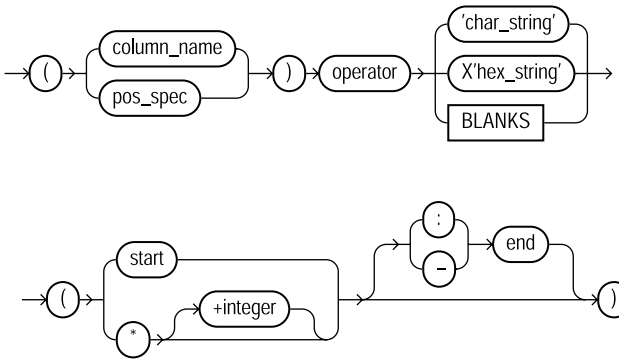
SINGLEROW オプションの場合、新しい索引エントリや新しい索引のための記憶領域は必要ありません。結果としてできる索引は、新規にソートされたものほど最適化されていない可能性がありますが、作成するための記憶領域は少なくて済みます。ただし SINGLEROW を指定すると、各索引が挿入されるたびに UNDO 情報が追加で作成されるので、そのための時間がかかります。このオプションは、次の場合に使用してください。

- 使用可能な記憶領域が限られている。
- ロードされる行数が表のサイズに比べて小さい（比率が 1:20 以下かどうかがおおよその目安になります）。

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似ていますが、CONTINUEIF 句の条件とは次の 2 つの点で異なります。第一に、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。第二に、指定する位置として、論理レコードの位置またはロードするフィールド名のどちらでも指定できます。



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。start-end と表記することも、start:end と表記することもできます。end を省略した場合、このフィールドの長さには、比較文字列の長さが採用されます。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。文字列の場合はブランクが追加され、16 進数のバイト列の場合はゼロが追加されます。
full_field_name	full_field_name 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 col2 が列オブジェクト col1 の属性であるとき、ディレクティブで col2 を参照する場合は、col1.col2 という表記法を使用してください。同じエンティティを参照および命名している column_name および full_fieldname があっても、column_name 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）ので、別のものとして認識されます。
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	比較対象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。比較の結果が真であるときに、現在の行が表に挿入されます。
X' hex_string '	16 進数のバイト列で、用途は上記の char_string と同じです。
BLANKS	任意の数のブランクを示すキーワードです。詳細は次の項を参照してください。

BLANKS フィールドと BLANKS の比較

BLANKS キーワードを使用すると、長さが不明なフィールドのデータがブランクかどうかを知ることができます。

次の指定を実行すると、ブランク・フィールドに NULL 値を設定することができます。

```
full_field_name ... NULLIF column_name=BLANKS
```

BLANKS キーワードが認識できるのはブランクのみです。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。列の値がすべてブランクのときにのみ条件が真となります。

BLANKS キーワードは、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ブランク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意味です。

```
fixed_field CHAR(2) NULLIF (fixed_field)=BLANKS
fixed_field CHAR(2) NULLIF (fixed_field)="  "
```

注意 : マルチバイト・キャラクタ・セットには複数のブランク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、ブランク文字列を指定するかわりに BLANKS キーワードを使用するとよいでしょう。

文字列は特定のブランク文字の組合せにしか一致しませんが、BLANKS キーワードの場合はさまざまなブランク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、5-30 ページ「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・フィールドと比較文字列との比較において、比較文字列の方がデータ長が短い場合、比較のために文字列に特定の値が埋め込まれます。文字データ型の文字列の場合は、ブランクが埋め込まれます。例を示します。

```
NULLIF (1:4)="_"
```

このように指定すると、位置 1:4 にあるデータが 4 つのブランクと比較されます。位置 1:4 のデータがブランク 4 つであれば、この句は真となります。

一方、16 進文字列には 16 進数のゼロが埋め込まれます。次の句

```
NULLIF (1:4)=X'FF'
```

はデータ位置 1:4 を 16 進数のバイト列 'FF000000' と比較します。

列とフィールドの指定

表の列はいくつでもロードすることができます。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（列に NULL 値を割り当てる場合は、通常このようにします）。

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たり、大文字と小文字を区別する必要があるときは、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、列指定の中でキーワード RECNUM または CONSTANT、または SEQUENCE 関数を指定します。5-53 ページ「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ときは、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*）とデータ・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。

列オブジェクトをロードするとき、必ずしもすべての属性を指定する必要はありません。欠落した属性には、NULL が設定されます。

FILLER フィールドの指定

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。ただし、FILLER フィールドは、指示句（たとえば、SID、OID、REF、BFILE）と同様に、引数として init_specs（たとえば、NULLIF および DEFAULTIF）に使用できます。また、FILLER フィールドは、データ・ファイルのどこにでも指定できます。オブジェクトのフィールド・リスト内部、または VARRAY の定義の内部に指定できます。FILLER フィールドとその使用方法の詳細は、3-18 ページ「新しい SQL*Loader DDL の動作および制限事項」を参照してください。

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り、SQL*Loader がフィールドのデータを処理する方法が決まります。たとえば、データ型 INTEGER 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。ただし、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。

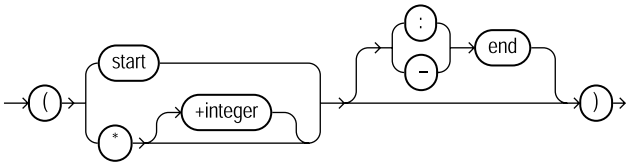
各フィールドには、1 つのデータ型のみ指定できます。指定しない場合は、CHAR とみなされます。

SQL*Loader データ型から Oracle データ型への変換処理や、SQL*Loader の各データ型については、5-57 ページ「SQL*Loader のデータ型」で説明します。

データ型を指定する前に、そのフィールドの位置を指定する必要があります。

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 キーワード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 の引数は、下記のように、小カッコで囲みます。



start	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文字位置は 1 となります。
end	論理レコード中のデータ・フィールドの終了位置です。start-end と表記することも、start:end と表記することもできます。end 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かれます。CHAR 型については、start と end の両方を省略すると、データ長が 1 として扱われます。データ型から長さを導くことができない場合は、エラー・メッセージが出力されます。
*	対象となるデータ・フィールドが直前のフィールドのすぐ次の位置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アスタリスクを指定した場合、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。位置指定にアスタリスクを使用した場合、フィールド長はデータ型から導かれます。
+n	現フィールドの位置を前フィールドからのオフセットで表すには、+n と指定してオフセットをオンにします。この場合、現フィールドの値は、n 文字分スキップした後で読み込まれます。

POSITION キーワード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・フィールドの位置は、POSITION（*）と指定した場合と同じになります。

例を示します。

```
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 の終わりから +2 の位置、すなわち列 28 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。

タブを含むデータでの POSITION の使用

フィールド位置の指定において、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をロードする際に SQL*Loader の拡張 SQL 文字列機能を使用する場合、次のような失敗をすることがよくあります。

- 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て、制御ファイルを作成する。
- これをロードしたところ、「無効数字」や「欠落フィールド」エラーが多数発生した。

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拡がります。しかし、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・ファイルを読み込むとき、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。

デリミタを使用してフィールドの相対位置を指定する方法については、5-69 ページ「デリミタの指定」を参照してください。特に、この項の WHITESPACE デリミタの使用方法に関する説明を参照してください。

複数表へのロードにおける POSITION の使用

複数の表にロードするときは複数の 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 句の処理を開始する段階で、位置が自動的に論理レコードの先頭にセットされるわけではありません。これによって、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することができます。この指定については、5-50 ページ「複数の論理レコードの抽出」の 2 番目の例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、位置指定を省略したり INTO TABLE 句で先頭フィールドを POSITION（*+*n*）と指定したりするかわりに、POSITION（1）や POSITION（*n*）と指定します。

例

```
SITEID  POSITION (*) SMALLINT
SITELOC POSITION (*) INTEGER
```

これが、最初の 2 つの列の指定の場合は、SITEID が 1 列目から始まり SITELOC がそのすぐ次の列から始まることになります。

```
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 の終わりから +2 の位置、すなわち位置 28 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。

複数の INTO TABLE 文の使用

複数の INTO TABLE 文を指定すると、次の処理が可能になります。

- 異なる表にデータをロードする。
- 1 つの入力レコードから複数の論理レコードを抽出する。
- 異なる入力レコード形式を区別する。

上記の中で、最初の場合は複数の INTO TABLE 文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、これらのそれぞれの場合における複数の INTO TABLE 文の指定方法を、例を使用して示します。また、その際の POSITION キーワードの指定方法について説明します。

注意 : INTO TABLE 文を複数指定した場合、次の INTO TABLE 文の処理に移ったときのフィールド走査は、前回フィールド走査を停止した位置から続行されます。この項の以降の部分では、このような走査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 キーワードを用い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

データストレージ / 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することができます。

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 つ指定して EMP 表にデータをロードする方法について説明します。データが次のような場合、

```
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)
```

相対的な位置指定

この同じレコードを、別の指定方法でロードすることもできます。次の制御ファイルでは、絶対位置を指定するかわりに相対的な位置指定を使用しています。ここでは、各フィールドがブランク（""）1 文字、またはいくつかのブランクやタブ（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 ENAME の直後にありますが、それぞれ別個の INTO TABLE 句に入れられていることに注意してください。つまり 2 番目の INTO TABLE 句に対してフィールドを先頭から走査し直しているのではなく、前に行われた処理の状態がそのまま残されて処理が継続されます。強制的にレコードの途中から走査を開始するには POSITION キーワードを使用します。

レコードの走査を特定の位置から強制的に開始するには、POSITION キーワードを使用します。詳細は、次の例で説明します。

異なる入力レコード形式の区別

通常、データ・ファイルにはさまざま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 表と 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。部門レコードは最初の列は "1"、従業員レコードは "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  POSITION(1:1)  INTEGER EXTERNAL,
    deptno POSITION(3:4)  INTEGER EXTERNAL,
    ename  POSITION(8:21) CHAR)
INTO TABLE emp
   WHEN recid <> 1
   (recid  POSITION(1:1)   INTEGER EXTERNAL,
    empno  POSITION(3:6)   INTEGER EXTERNAL,
    ename  POSITION(8:17)  CHAR,
    deptno POSITION(19:20) INTEGER EXTERNAL)
```

相対的な位置指定

前記の例のレコードを、デリミタ付きのデータとしてロードすることもできます。ただし、その場合は POSITION キーワード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  INTEGER EXTERNAL TERMINATED BY WHITESPACE,
    deptno INTEGER EXTERNAL TERMINATED BY WHITESPACE,
    dname  CHAR TERMINATED BY WHITESPACE)
INTO TABLE emp
   WHEN recid <> 1
   (recid  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 句における POSITION キーワードは、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ことで、2 つ目の書式に一致するデータをチェックするときのフィールド走査は、列 1 から開始されます。この POSITION 指定がないと、SQL*Loader は、RECID フィールドが DNAME フィールドの後にあるものとして走査します。

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。4-19 ページ「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することができます。

デリミタ付きデータの場合は、期待する結果が得られるように POSITION キーワードを正しく指定してください。

複数の INTO TABLE 句の指定の中で POSITION キーワードを指定しない場合は、1 つの（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 句の指定の中で POSITION キーワードを指定した場合は、同一のレコードを異なる方法で処理できます。つまり、1 つの入力ファイルにおいて複数の形式を識別できます。

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行に格納する関数について説明します。ここで説明する関数は次のとおりです。

- CONSTANT
- RECNUM
- SYSDATE
- SEQUENCE

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、システム日付および定数のみを指定して SQL*Loader でデータを生成できます。

SQL*Loader は、LOAD キーワードで指定された数の行を挿入します。SQL*Loader でデータを生成する場合は、必ず LOAD キーワードを指定してください。SKIP キーワードは指定できません。

SQL*Loader は、このような場合に対して最適化されています。生成されるデータの指定のみ使用されていることが検出された場合、データ・ファイルを指定していても SQL*Loader によって無視されます。I/O の読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、SQL*Loader はデータの評価が必要と判断して、入力レコードを読み込みます。

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。ロード実行中であっても、何回ロードしても、このデータは不変です。

CONSTANT

列に定数値を設定するには、CONSTANT キーワードを指定して、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ときは、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。指定した値が無効であると、すべての行が拒否されてしまいます。

2^{32} - 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 キーワードを使用して列に NULL を設定することはできません。列を NULL に設定したい場合は、その列については何も指定しないでください。そうすれば、Oracle で行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 と値の組合せを指定すれば、列指定は完結します。

列へのデータ・ファイルのレコード番号の設定

RECNUM キーワードを列名の後に指定すると、行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。したがって、廃棄、スキップ、拒否またはロードされたレコードについても、RECNUM のカウントに含まれます。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

列名と RECNUM キーワードの組合せを指定すれば、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 関数で定義されているものと同様のシステム日付が列に設定されます。詳細は、『Oracle8*i* SQL リファレンス』の「DATE データ型」の項を参照してください。

SYSDATE

列名と SYSDATE キーワードの組合せを指定すれば、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 型または 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アクセスができます。システム日付を DATE 列にロードすれば、そのシステム日付は、時間と日付を含むさまざまな書式でアクセスできます。

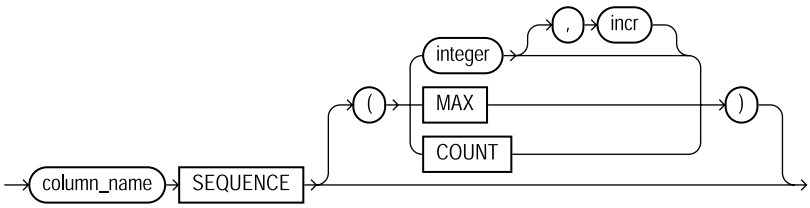
新しいシステム日付 / 時間は、従来型パス・ロードで挿入されたレコードの各配列や、ダイレクト・パス・ロード中にロードされた各レコード・ブロックで使用されます。

列への一意の順序番号の設定

SEQUENCE キーワードは、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

列名と SEQUENCE キーワードの組合せを指定すれば、列指定は完結します。



column_name	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 キーワードを使用します。
n	先頭となる特定の順序番号を指定します。
COUNT	順序番号は表中にすでにある行数で始まり、以降は増分値（increment）分ずつ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（increment）分ずつ加えた値が設定されます。
increment	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。

行が拒否された場合（書式エラーがあるかまたは Oracle エラーを発生させた場合）でも、その欠落を埋めるために生成された順序番号が振り直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、16 が割り当てられたところ、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、16 であって、10、12、14 ではありません。このような処理を行うことにより、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。

4-11 ページ「事例 3: 自由区分形式ファイルのロード」にある SEQUENCE の使用例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、各表へ挿入するたびに付けられるのではなく、各論理入力レコードに対して付けられます。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、多くの場合に役立つ処理方法といえます。

ただし、各 INTO TABLE 句ごとに別の順序番号を生成する場合もあります。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) を使うと、各表の最大値が採用されるため、順序番号が矛盾する場合があります。

このレコードの順序番号を生成する場合、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。そのためには、単純に次のように指定します。1 レコードあ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

下記の部門名を 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 番目の INTO TABLE 句では 2 が、3 番目の INTO TABLE 句では 3 が生成されます。INTO TABLE 句に指定してある増分値はすべて 3 になっています（増分値は各レコードに含まれる部門名の数と一致させます）。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。

SQL*Loader のデータ型

SQL*Loader では、幅広いデータ型がサポートされています。これらのデータ型は、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。これら 2 つのグループ内で、Length-value データ型と Value データ型に分類されます。

移植可能なデータ型と移植不能なデータ型は主に、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。これは、異なるプラットフォームのバイト順（ビッグ・エンディアンとリトル・エンディアン）の違い、特殊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）などのように、プラットフォームの仕様が複数あることによって起こります。これらの問題が、すべての移植不能データ型に適用されるわけではありません。

次の分類の、Value と Length-Value には、別の問題があります。Value データ型はデータフィールドの 1 つの部分であると想定しているのに対し、Length-Value データ型は、2 つのサブフィールドからなるデータフィールドが必要です。それは、2 番目の（値）サブフィールドと、そのサブフィールドの長さを指定する長さサブフィールドです。

移植不能データ型

VALUE データ型	Length-Value データ型
INTEGER	
SMALLINT	VARCHAR
FLOAT	VARGRAPHIC
DOUBLE	VARRAW
BYTEINT	LONG VARRAW
ZONED	
(packed)DECIMAL	

INTEGER

データはフルワードのバイナリ整数（符号なし）で表現されます。POSITION 句で *start: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フルワード整数（C においての LONG INT データ型）の長さが取られます。このデータ型のサイズは、制御ファイルを使用して変更することはできません。

```
INTEGER
```

SMALLINT

データはハーフワードのバイナリ整数（符号なし）で表現されます。POSITION 句で *start:end* を指定すると *end* は無視されます。フィールド長には、ご使用のシステムのハーフワード整数の長さが取られます。

```
SMALLINT
```

追加情報 : このデータ型のフィールド長は、C プログラミング言語の 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り、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することはできません。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バイナリ数で表現されます。POSITION 句で *end* を指定しても無視されます。このフィールド長には、ご使用のシステムの単精度浮動小数点バイナリ数の長さ（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変更することはできません。

DOUBLE

データは倍精度浮動小数点バイナリ数で表現されます。POSITION 句で *end* を指定しても無視されます。このフィールド長には、ご使用のシステムの倍精度浮動小数点バイナリ数の長さ（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変更することはできません。

```
DOUBLE
```

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 フィールド長は常に 1 バイトです。POSITION（*start*:*end*）を指定すると、*end* は無視されます。（C においてのデータ型 UNSIGNED CHAR。）

このデータ型の構文は次のとおりです。

```
BYTEINT
```

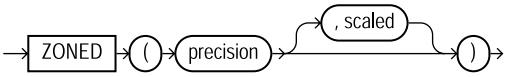
次に例を示します。

```
(column1 position(1) BYTEINT,
column2 BYTEINT,
...
)
```

ZONED

ZONED 型のデータは、ゾーン 10 進数形式で表現されます。つまり、10 進数の各 1 桁が 1 バイトで表され、最後のバイトに符号が入ります。（COBOL で言えば、SIGN TRAILING フィールドに相当します。）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。

このデータ型の構文は次のとおりです。



ここで *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。たとえば、

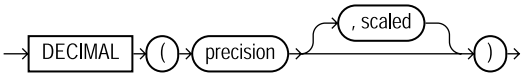
```
sal    POSITION(32)    ZONED(8),
```

と指定すると、位置 32 から始まる 8 桁の整数を表します。

DECIMAL

DECIMAL 型のデータは、パック化された 10 進数の形式で表現されます。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ので、分数の値を表すこともできます。

このデータ型の構文は次のとおりです。



precision	数値の桁数です。DECIMAL フィールドの文字長は、桁数から計算します。つまり、（桁数 +2/2）を求め、その小数点以下を切り上げた値が文字長となります。
scale	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。デフォルト値はゼロです（つまり整数となります）。全体の桁数より大きい数は指定できますが、負数は指定できません。

たとえば、

```
sal DECIMAL (7,2)
```

と指定すると、+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。）

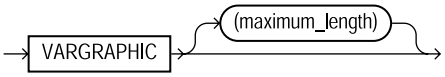
VARGRAPHIC

このデータは、可変長の 2 バイト文字列です。長さを示すサブフィールドと、2 バイト文字（DBCS）の文字列で構成されます。

> **追加情報 :** 長さを示すサブフィールドのサイズには、使用しているシステムにおける 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、5-58 ページ「SMALLINT」を参照してください。

現在のフィールドの長さは、先頭の 2 バイトで示されます。ここで示される長さは、グラフィック（2 バイト）文字の文字数を表しています。したがって、この長さを 2 倍すれば読み込むバイト数が求められます。

このデータ型の構文は次のとおりです。



VARGRAPHIC キーワードの後に指定する最大長（maximum_length）には、長さを示す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（2 バイト）文字の文字数を指定します。したがって、この maximum_length の値を 2 倍すれば、フィールドの最大長（バイト）が求められます。

フィールド最大長のデフォルトは、グラフィック文字で 4KB、つまり 8KB（2 × 4Kb）です。このような可変フィールドに対しては、必要となるメモリーを最小限にするため、できるかぎり最大長を指定してください。詳細は、5-74 ページ「バインド配列サイズの決定」を参照してください。

POSITION 句を使用する場合、指定する位置は、グラフィック文字の先頭ではなく、長さを示すサブフィールドの位置になります。POSITION（*start*:*end*）と指定すると、end の位置によりそのフィールドの最大長が決まります。ここで、*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* の値を引くことにより、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 句から計算された最大長よりも優先されます。

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ので、そのフィールド長の方が正確であるとみなされます。

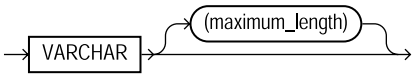
VARGRAPHIC データに対してはデリミタを使用できません。

VARCHAR

VARCHAR フィールドは、Length-value データ型です。VARCHAR は、バイナリの長さサブフィールドと、その長さを持った文字列で構成されます。

> **追加情報 :** 長さサブフィールドのサイズには、ご使用のシステムにおける 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、5-58 ページ「SMALLINT」を参照してください。

この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、長さサブフィールドのサイズは含めません。VARCHAR キーワードの後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割り当てられます。

デフォルトのバッファ・サイズは 4KB です。ロードするの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 が使用す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ときに有効です。詳細は、5-74 ページ「バインド配列サイズの決定」を参照してください。

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、長さサブフィールドの位置になります。POSITION（*start*:*end*）と指定すると、end の位置により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 の値を引くことにより、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ので、そのフィールド長の方が正確であるとみなされます。

VARCHAR 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長さサブフィールドと、その後に続く RAW 文字列の値サブフィールドから成ります。

このデータ型の構文は 5-12 ページ「datatype_spec」のダイアグラムを参照してください。

デフォルトでは、VARRAW は、長さサブフィールドが 2 バイトで、最大サイズが 4KB の VARRAW になります。VARRAW(65000) は、長さサブフィールドが 2 バイトで、最大サイズが 65000 バイトの VARRAW になります。

LONG VARRAW

LONG VARRAW は、2 バイトの長さサブフィールドではなく、4 バイトの長さサブフィールドの VARRAW です。

このデータ型の構文は 5-12 ページ「datatype_spec」のダイアグラムを参照してください。

デフォルトでは、LONG VARRAW は、長さサブフィールドが 4 バイトで、最大サイズが 4KB の VARRAW になります。VARRAW(300000) は、長さサブフィールドが 4 バイトで、最大サイズが 300000 バイトの VARRAW になります。

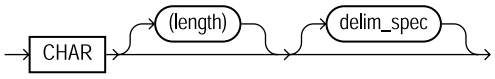
移植可能なデータ型

VALUE データ型	Length-Value データ型
CHAR	VARCHARC
DATE	VARRAWC
INTEGER EXTERNAL	
RAW	
GRAPHIC	
GRAPHIC EXTERNAL	

文字データ型には、CHAR 型および DATE 型、numeric 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。データ長の指定はオプションであり、指定しなかった場合は POSITION 句から長さが求められます。データ長を指定した場合は、POSITION 句から求めれられる長さよりも優先されます。データ長がまったく指定されない場合は、CHAR のデータ長は 1 文字とみなされます。構文は次のとおりです。

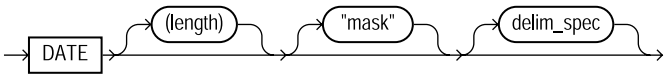


データ型 CHAR のフィールドは、デリミタで区切ったり囲んだりすることにより可変長にもできます。5-69 ページ「デリミタの指定」を参照してください。

注意 : データベース表中の列が LONG 型または VARCHAR2 型と定義されている場合は、その CHAR キーワードの長さ指定子または POSITION キーワードを使用して、フィールドの最大長（LONG は最大で 2GB）を明示的に指定する必要があります。最大長を指定すると、その値に対して十分なサイズのバッファが割り当てられます。データがデリミタで区切られるか囲まれていても、この指定は必要です。

DATE

このデータ・フィールドには、文字データが入ります。DATE 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。構文は次のとおりです。



たとえば、次のように表示されます。

```
LOAD DATA
INTO TABLE DATES (COL_A POSITION (1:15) DATE "DD-Mon-YYYY")
BEGINDATA
1-Jan-1991
1-Apr-1991 28-Feb-1991
```

注意：デリミタがない場合は、空白は無視され、日付は左から右に構文解析されます。

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。前述の例では、日付マスクは 11 文字の固定長日付形式を指定しています。この日付マスクの文字数が 11 文字のため、SQL*Loader はこのフィールドの最大文字数を 11 文字とみなします。したがって、上述の指定は正しく処理されます。しかし、次のように指定すると、

```
DATE "Month dd, YYYY"
```

日付マスクは 14 文字になります。一方、次のようなフィールドの場合、

```
September 30, 1991
```

フィールドの最大長は 18 文字になります。このような場合は、長さを指定する必要があります。同様にユリウス暦日付（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（マスク内の文字数）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に確実におさまる場合を除いて、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 句から求められる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。4-11 ページ「事例 3: 自由区分形式ファイルのロード」にある DATE データ型の使用例を参照してください。

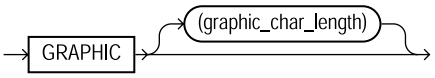
DATE 型のフィールドでは、デリミタを使用することもできます。詳細は、5-69 ページ「デリミタの指定」を参照してください。

日付フィールドのデータがすべて空白文字の場合、NULLIF BLANKS が指定されていない限り、そのフィールドはエラーとなります。詳細は、5-81 ページ「ブランク・フィールドのロード」を参照してください。

GRAPHIC

このデータは、2 バイト文字（DBCS）の文字列データです。Oracle 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 フィールドは何も変更されずにそのまま指定の列に格納されます。

このデータ型の構文は次のとおりです。

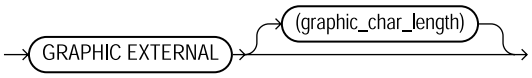


GRAPHIC 型および GRAPHIC EXTERNAL 型では、POSITION(*start*:*end*) を指定すると、論理レコードにおけるフィールドの正確な位置が決まります。

ただし、GRAPHIC(EXTERNAL) キーワードの後にデータ長を指定するときは、2 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、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 GRAPHIC 型とほぼ同じですが、GRAPHIC 型と違って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）はロードされません。このデータ型の構文は次のとおりです。



GRAPHIC	データは 2 バイト文字です。
EXTERNAL	先頭と最後の文字が無視されます。
graphic_char_length	DBCS におけるデータ長です（上記の GRAPHIC 型参照）。

たとえば、[ ] をシフトイン / シフトアウト文字とし、# を任意の 2 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。

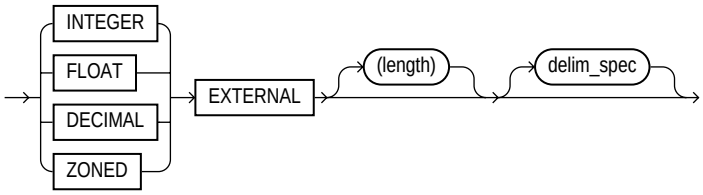
[####] を表現する場合は、"POSITION(1:6) GRAPHIC EXTERNAL" または、"POSITION(1) GRAPHIC EXTERNAL(2)" と指定します。

numeric EXTERNAL データ型

numeric EXTERNAL データ型は、数値データ型（INTEGER、FLOAT、DECIMAL、ZONED）に EXTERNAL キーワード、オプション・データ長およびデリミタ仕様を指定したものです。

このデータ型は、判読可能な文字形式の数値データです。numeric EXTERNAL データ型を指定するときは、CHAR 型と同様にデータ長とデリミタも指定できます。データ長の指定はオプションですが、指定した場合は POSITION から計算された値よりも優先されます。

このデータ型の構文は次のとおりです。



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、これらのデータ型は、その性質も処理方法も CHAR と同じです。ただし DEFAULTIF を使用する場合には注意が必要です。デフォルトを NULL にする場合は CHAR を使用します。デフォルトをゼロにする場合は EXTERNAL を使用します。5-80 ページ「列への NULL またはゼロの設定」および「DEFAULTIF 句」も参照してください。

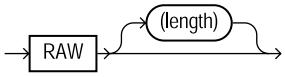
FLOAT EXTERNAL データ値

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

データは、そのままバイナリ形式でロードされます。キャラクタ・セットの変換は行われません。RAW 型のデータベース列にロードした場合、Oracle によるデータ変換は行われません。CHAR 型の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。

この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とメモリー・リソースの許容範囲内であれば自由に指定できます。RAW データ・ファイルは区切ることができません。

VARCHARC

データ型 VARCHARC は、文字の長さサブフィールドと、その後に続く文字列の値サブフィールドで構成されます。

このデータ型の構文は 5-12 ページ「datatype_spec」のダイアグラムを参照してください。

たとえば、次のようになります。

- VARCHARC はエラーになります。
- VARCHARC（7）は、長さサブフィールドが 7 バイトで、最大サイズが 4KB の VARCHARC になります（つまりデフォルト）。
- VARCHARC（3,500）は、長さサブフィールドが 3 バイトで、最大サイズが 500 バイトの VARCHARC になります。

VARRAWC

データ型 VARRAWC は、RAW 文字列の値サブフィールドから構成されます。

このデータ型の構文は 5-12 ページ「datatype_spec」のダイアグラムを参照してください。

たとえば、次のようになります。

- VARRAWC はエラーになります。
- VARRAWC（7）は、長さサブフィールドが 7 バイトで、最大サイズが 4KB の VARRAWC になります（つまりデフォルト）。

- VARRAWC（3,500）は、長さサブフィールドが 3 バイトで、最大サイズが 500 バイトの VARRAWC になります。

システム固有なデータ型フィールド長の衝突

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った値を指定して、値が矛盾する場合は、そのうちの 1 つの値が優先されます。警告は矛盾が発生した時点で出されます。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. INTEGER、SMALLINT、FLOAT および DOUBLE のデータ・サイズは固定長です。制御ファイルでこれらのデータ型の長さを指定することはできません。開始位置と終了位置が指定された場合は、開始位置のみが有効となり、終了位置は無視されます。
2. DECIMAL、ZONED、GRAPHIC、GRAPHIC EXTERNAL または RAW において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. VARCHAR や 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
```

これを実行すると警告が出力され、正しいフィールド長である 4 バイトが採用されます。この場合、ログ・ファイルには実際に使用されたフィールド長が列表の "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
```

データ型の変換

制御ファイルに指定されたデータ型は、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 に対して指示します。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づける手がかりとなるのが、制御ファイル中に指定されている列名です。

SQL*Loader は、入力ファイル中のフィールドからデータを抽出します。その際、制御ファイルに指定されたデータ型に基づいて抽出処理が行われます。次に SQL*Loader は、そのフィールドをサーバーに送信します。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。

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。クライアントでは、コレクション列の（VARRAY およびネストした表）フィールドに対して、データ型の変換が行われます。ネストした表を親から分割した表としてロードする場合、データ型の変換は行われません。

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ければ、Oracle で自動的に変換を行います。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 や VARCHAR2 などの Oracle 内部データ型に関するデータ型仕様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）と、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。しかし、SQL*Loader は NUMBER や VARCHAR2 のようなデータ型を認識しませんが、Oracle で変換可能なデータであれば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。

デリミタの指定

CHAR、DATE、numeric 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。データ型の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示します。

データを区切る方法には、TERMINATED（終端を示す）または ENCLOSED（囲む）があります。

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。

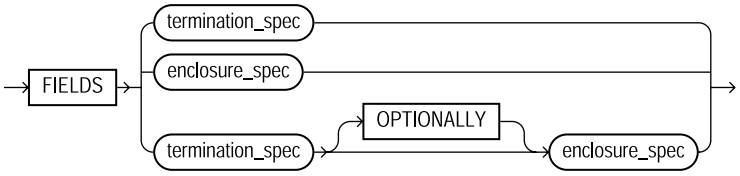
TERMINATED BY WHITESPACE

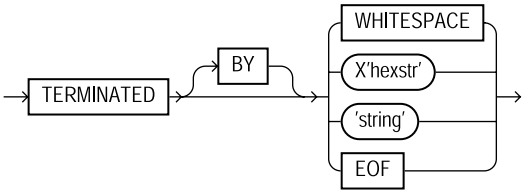
TERMINATED BY WHITESPACE を指定すると、最初に空白文字（空白、タブ、改行）が現れるまでデータが読み込まれます。空白文字が現れると、次に空白以外の文字が現れるまで連続する空白文字列は読み込まれません。したがって、フィールド値の間に入る空白は、いくつあってもかまいません。

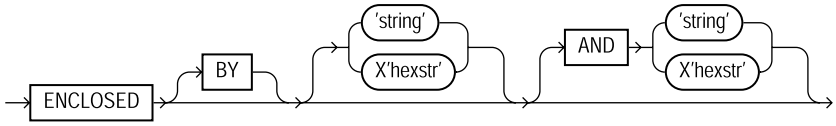
囲まれたフィールド

囲まれたフィールドの読込みでは、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であれば、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、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 つのデリミタしか含まれていない場合は NULL 値となります。デリミタの指定には、TERMINATED BY 句または ENCLOSED BY 句、あるいはその両方を使用することもできます。両方とも指定するときは TERMINATED BY 句を先に指定してください。デリミタの指定の構文は次のとおりです。







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	読みやすくする場合に、このキーワードをオプションで指定します。
WHITESPACE	デリミタには改行、改ページまたはキャリッジ・リターン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	ここで指定する文字でデータを囲むこともできます。SQL*Loader 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を読み込みます。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。
char	char で指定する 1 文字がデリミタとなります。
X'hex_byte'	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 16 進（hex-byte）表記の値で、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" は大文字にしてください。
AND	このキーワードは、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。

各指定方法によるデリミタの指定例と、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・マーク

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場合があります。このような場合、デリミタ文字を 2 つ続けて記述すれば、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 paren's, (, and right paren's, ).
```

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、問題が起こります。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、フィールド 1 とフィールド 2 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、上記のデータ全体が、中央に 1 つの "/" のみを持つ単一の文字列とみなされ、フィールド 1 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をバインド配列に格納する際に、記憶領域が大量に使用されることがあります。このため、最大長にはできるかぎり小さな値を指定してください。詳細は、5-74 ページ「バインド配列サイズの決定」を参照してください。

デリミタによる後続の空白のロード

後続の空白は、データ型がデリミタ付きで指定されている場合のみロードできます。たとえば、データ・フィールド長が 9 文字で、DANIEL*bbb* という値のデータがあるとします。ここで、*bbb* は 3 つの空白文字を示します。このとき、CHAR(9) と宣言されていると、Oracle には "DANIEL" がロードされます。この例において後続の空白文字も必要な場合は、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 DANIEL*bbb*: とします。こうすれば、後続の空白もロードされて、"DANIEL " となります。フィールド中の空白の詳細は、5-81 ページ「ブランクとタブの切捨て」を参照してください。

文字データ型フィールド長の矛盾

CHAR 型、DATE 型、numeric EXTERNAL 型の文字データ型の場合、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っていて値が矛盾する場合は、そのうちの 1 つが優先されます。矛盾が発生すると、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ていて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 / 終了位置と長さの両方が指定されていてその長さが異なる場合は、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。たとえば、次のように指定します。

```
position(1:10) char(15)
```

この場合、このフィールドの長さは 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の指定の中で長さが指定された場合、または開始位置と終了位置から長さが計算できる場合は、その長さがフィールドの最大長となります。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。フィールドの開始位置と終了位置の両方が指定され、さらにフィールド長も指定された場合は、指定されたフィールド長の値の方が開始 / 終了位置から計算された長さよりも優先されます。

デリミタが見つからず、最大長も指定されなかった場合は、レコードの終わりがフィールドの終端となります。このとき、TRAILING NULLCOLS が指定されていると、残りのフィールドには NULL 値が設定されます。また、デリミタやレコードの終端で区切った結果、フィールドの長さが指定された最大長よりも大きくなる場合は、エラーが出力されます。

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 はその形式に基づいてレコード中のデータを解釈します。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" はレコード中で 11 文字を占有し、"January 31, 1992" は 16 文字を占有することになります。

しかし、開始位置と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。

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 バイトを使用するのに対し、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、Net8 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。上記のような問題が生じた場合は、できるだけこの方法を使用してください。

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よって発生します。場合によっては、フィールドにゼロを追加してフィールド長を伸ばしたり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ことで、問題を回避できることもあります。(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や、その逆のパターンの場合がこれに相当します)。しかし、バイト順やデータ型の表現に互換性がない場合は、この方法で問題を解決することはできません。

Net8 データベース・リンクを使用しない場合は、できれば CHAR 型、DATE 型、VARCHARC 型、numeric EXTERNAL データ型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、システム固有なデータ型を使用して作成されたデータ・ファイルよりもサイズが大きくなります。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。しかし、バイト順の互換性の問題があるときは、特別なフィルタを通してデータのバイト順を変更してから転送する必要があります。

バインド配列サイズの決定

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・パスによるロードの場合は、必要ありません。ダイレクト・パス・ロードでは、Oracle SQL インタフェースを介さずにデータベース・ブロックの形式を直接構成するので、バインド配列は使用しません。

SQL*Loader は、データをデータベースに転送するときに SQL 配列インタフェース・オプションを使用します。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 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。

最低条件

バインド配列には、少なくとも 1 行は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 はエラーを出力します。通常の場合は、バインド配列内に入るかぎりの行が格納されます。この場合の読込み行数の上限は、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、6-3 ページ「コマンド行キーワード」で説明します。

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。

パフォーマンスに関する考慮点

Oracle へのコール回数を最小にして、パフォーマンスを最大にするには、バインド配列のサイズを大きくしてください。一般に、バインド配列の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格段に向上します。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。この項ではこれ以降、この配列サイズの決定方法について解説します。

一般に、適切な大きさのサイズであれば SQL*Loader は効果的に処理を行います。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考にしてください。

行数と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、コマンド行パラメータ BINDSIZE（6-4 ページ「BINDSIZE（最大サイズ）」参照）または制御ファイル中の OPTIONS 句（5-18 ページ「OPTIONS」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 は 1 行をロードするのに必要な領域を決定します。このサイズが指定された最大値を超えるときは、エラーを出力して実行を終了します。

次に SQL*Loader は、このサイズとロードする行数を掛け合せます。このとき、ロードする行数はコマンド行パラメータ ROWS（6-7 ページ「ROWS（1 回にコミットする行数）」参照）で指定しても、制御ファイル中の OPTIONS 句（5-18 ページ「OPTIONS」参照）を使用して指定してもかまいません。

このサイズがバインド配列の最大値を超えていなければ、ロードは継続されます。SQL*Loader はバインド配列の最大サイズの限界まで行数を拡張することはありません。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 はこれらの値のうち小さい方をバインド配列に適用します。

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を採用します。

計算方法

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、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) * (maximum row length)

(maximum row length) = SUM(fixed field lengths) +
SUM(maximum varying field lengths) +
SUM(overhead for varying length fields)
```

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、5-57 ページ「SQL*Loader のデータ型」で説明したように、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。

```
VARCHAR                 VARGRAPHIC
CHAR                    DATE
numeric EXTERNAL
```

これらのデータ型の最大長については、5-57 ページ「SQL*Loader のデータ型」で説明しています。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、バイト数または文字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 型、DATE 型、numeric EXTERNAL 型）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における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。

バインド配列には、これらの各フィールドごとに、長さの情報を保持する標識が入っています。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、行ごとに長さ標識で示されます。

要約すると次のようになります。

```
bind array size =
    (number of rows) * (  SUM(fixed field lengths)
                        + SUM(maximum varying field lengths)
                        + ( (number of varying length fields)
                            * (size of length-indicator) )
                       )
```

長さ標識のサイズの決定

ほとんどのシステムでは、長さ標識のサイズは 2 バイトです。まれに 3 バイトのシステムもあります。長さ標識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)
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 文字のフィールドをロードします（ただし実際は、データはロードされません。これは、"a" を数字としてロードするときに、数値型への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、フィールド長の標識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れば、バインド配列サイズが求められます。

フィールド・バッファ・サイズの計算

次の表に、各データ型のメモリー所要量を示します。"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" はフィールド長標識のサイズとなります。これらの値の詳細は、5-57 ページ「SQL*Loader のデータ型」を参照してください。

表 5-1 固定長フィールド

データ型	サイズ
INTEGER	OS によって異なる
SMALLINT	
FLOAT	
DOUBLE	

表 5-2 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	なし	(P+1) / 2 切上げ
ZONED	なし	P
RAW	なし	L
CHAR 型（デリミタなし）	1	L+S
DATE 型（デリミタなし）	なし	
numeric EXTERNAL （デリミタなし）	なし	

表 5-3 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 5-4 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（デリミタ付き） DATE（デリミタ付き） numeric EXTERNAL（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 型や VARGRAPHIC 型フィールド、およびデリミタ付きの CHAR 型、DATE 型、numeric 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。たとえば、次のように表示されます。

```
CHAR(10) TERMINATED BY ","
```

この場合、バインド配列では、(10+2) × 64=768 バイトのメモリーを使用します（ここでは、長さ標識を 2 バイトとして計算しています）。

```
CHAR TERMINATED BY ","
```

このように指定すると、（255+2）× 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 文の使用

制御ファイルに複数の INTO TABLE 文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 文が指定されていない場合と同じように行います。言い換えれば、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 行のデータ構造体としてとらえます。

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たびに、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。

生成されたデータ

CONSTANT、RECNUM、SYSDATE および SEQUENCE の各関数を指定すると、SQL*Loader により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。

列への NULL またはゼロの設定

指定された列に対する挿入値すべてを NULL にする場合は、列指定全体を省略します。また、論理レコードにおいて特定の条件判断を行い、それに基づいて列の値を NULL に設定する場合は、NULLIF 句を指定します。詳細は、5-80 ページ「NULLIF キーワード」を参照してください。数値型の列に、NULL 値ではなくゼロを設定するときは、DEFAULTIF 句を使用します。DEFAULTIF 句については、次に説明します。

DEFAULTIF 句

数値型データに対して DEFAULTIF 句を使用すると、指定されたフィールド条件が真のとき、列にゼロが設定されます。文字型（CHAR または DATE）データに対して DEFAULTIF 句を使用すると、列に NULL が設定されます（5-66 ページ「numeric EXTERNAL データ型」と比較）。条件判断の詳細は、5-44 ページ「フィールド条件の指定」も参照してください。

```
DEFAULTIF  field_condition
```

1 つの列に NULLIF 句と DEFAULTIF 句の両方を指定することもできますが、多くの場合には内容が重複することになります。

> **注意 :** SQL の文字列と DECODE 関数を使用すれば、これと同様の処理を行うことができます。5-87 ページ「フィールドへの SQL 演算子の適用」を参照してください。

NULLIF キーワード

NULLIF キーワードは、データ型とオプションのデリミタの指定の後に指定します。NULLIF の後には条件を記述します（条件の形式は、WHEN 句で指定する場合と同じです）。この条件が真の場合、列の値は NULL に設定されます。真でない場合は、列の値はそのままになります。

```
NULLIF field_condition
```

NULLIF 句は、次の例のようにそれ自体を格納する列も参照できます。

```
COLUMN1  POSITION(11:17)  CHAR  NULLIF  (COLUMN1 = "unknown")
```

上記の指定は、特定のデータ値を NULL に置き換える場合に有効です。列の値は、最初はデータ・ファイルから取得した値となります。次に、その値が NULL に設定され、挿入されます。4-25 ページ「事例 6: ダイレクト・パス・ロード方式を使用したロード」にある NULLIF 句の使用例を参照してください。

注意 : SQL の文字列と NVL 関数を使用すれば、これと同様の処理を行うことができます。5-87 ページ「フィールドへの SQL 演算子の適用」を参照してください。

レコードの終わりの NULL 列

制御ファイルに指定された 1 レコード当たりのフィールド数が、実際にレコードに存在するフィールド数よりも多い場合、SQL*Loader は（指定された）残りの列を NULL 値に設定するか、またはエラーを出力します。この場合にどちらの処理を行うかを指定するには、TRAILING NULLCOLS 句（5-42 ページ「TRAILING NULLCOLS」参照）を使用します。

ブランク・フィールドのロード

数値フィールドまたは DATE フィールドでフィールドがすべてブランクであると、レコードの受付けは拒否されます。このようなフィールドのうち、特定のフィールドを NULL としてロードするには、NULLIF 句で BLANKS キーワードを指定します。その指定方法は、5-45 ページ「BLANKS フィールドと BLANKS の比較」で説明しています。NULLIF 句を使用してブランク・フィールドを NULL としてロードする例は、4-25 ページ「事例 6: ダイレクト・パス・ロード方式を使用したロード」を参照してください。

ブランク・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ブランク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。文字型フィールドにおける空白の切捨て処理については、次の項で説明します。

ブランクとタブの切捨て

ブランクやタブは、空白文字に分類されます。フィールドの指定方法にもよりますが、フィールド開始位置の空白（先頭の空白）とフィールド終了位置の空白（後続の空白）は、フィールドをデータベースに挿入する際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。この項では、文字データ・フィールドがどのように認識され、ロードされるかについて説明します。特に、空白文字をフィールドから切り捨てる条件に重点をおきます。

注意 : PRESERVE BLANKS を指定すると、逆に、空白文字をデータの一部としてロードできます。詳細は、5-86 ページ「空白文字の保存」を参照してください。

データ型

ここで説明する内容は、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
- DATE データ型
- numeric EXTERNAL 型
  - INTEGER EXTERNAL
  - FLOAT EXTERNAL
  - （packed）DECIMAL EXTERNAL
  - ZONED（10 進）EXTERNAL

VARCHAR 型フィールド

VARCHAR 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。VARCHAR 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。

フィールド長の指定

フィールド長の指定には 2 通りの方法があります。制御ファイルにフィールド長を固定長で定義した場合、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標識でフィールド長を決める場合は、そのフィールドをデリミタで区切ります。

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、フィールドの前後に指定します。次の例の中の引用符がこれに当たります。

```
"__aa__"
```

ここでアンダースコアはブランクまたはタブを意味します。一方、終了デリミタは、フィールドの終わりを示します。次の例の中のカンマがこれに当たります。

```
__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。それぞれの指定例を示します。

```
loc POSITION(19) TERMINATED BY ","
loc POSITION(19) ENCLOSED BY '"'
loc TERMINATED BY "." OPTIONALLY ENCLOSED BY '|'
```

デリミタと事前に決定されたサイズの併用

デリミタ付きのフィールドに対して事前にサイズが指定されていて、指定されたサイズに相当するフィールド境界までにデリミタが現れない場合は、エラーが出力されます。たとえば、次のように指定したとします。

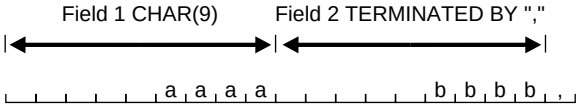
```
loc POSITION(19:31) CHAR TERMINATED BY ","
```

このとき、入力レコードの位置 19 から 31 の間にカンマがないと、このレコードは拒否されます。カンマがあれば、そのフィールドはカンマの位置で区切られます。

フィールドの相対位置指定

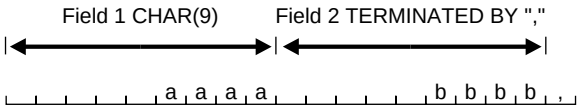
フィールドの開始位置が指定されていない場合は、前のフィールドの終了位置の直後の位置が、そのフィールドの開始位置となります。図 5-1 では、前のフィールドのサイズが事前に決定されている場合を示しています。

図 5-1 固定長フィールドの後の相対位置指定



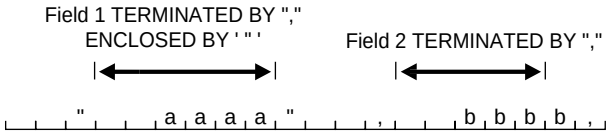
前のフィールドの終端がデリミタで指定されている場合、次のフィールドはそのデリミタの直後から開始します。この例を、図 5-2 に示します。

図 5-2 デリミタ付きフィールドの後の相対位置指定



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 5-3 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 5-3 囲みデリミタの後の相対位置指定



先頭の空白

図 5-3 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、現在のフィールドの開始位置が指定されていないとき。
- 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とき。

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 5-4 に示します。

図 5-4 空白で区切られたフィールド

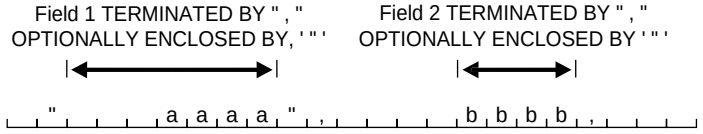
このようなケースは、上記の例のように前のフィールドが TERMINATED BY WHITESPACE 句で明示的に指定された場合に生じます。グローバルに FIELDS TERMINATED BY WHITESPACE 句が指定された場合も、このケースに該当します。

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、先頭の空白文字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 は前方方向に走査してデリミタを検索します。これが見つからないときは最初に見つかった空白以外の文字をフィールドの開始と判断します。SQL*Loader は空白文字をスキップして、フィールドから除外します。この例を図 5-5 で示します。

図 5-5 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 で指定された場合と異なり、上記のように指定された場合は、現在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空白文字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 5-5 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

後続の空白が切り捨てられるのは、その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のみです。このようなフィールドであれば、後続の空白は必ず切り捨てられます。

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、または、図 5-5 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は、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。

空白の切捨て : 要約

表 5-5 は、PRESERVE BLANKS が指定されていない場合の、入力データ・フィールドの空白が切り捨てられるケースとその処理内容をまとめたものです。切捨てを行わないようにする方法の詳細は、5-86 ページ「空白文字の保存」を参照してください。

表 5-5 空白の切捨て

指定	データ	結果	先頭の空白 **(1)**	後続の空白 **(1)**
サイズ指定あり	__aa__	__aa	Y	N
終了デリミタ	__aa__,	__aa__	Y	Y **(2)**
囲みデリミタ	"__aa__"	__aa__	Y	Y
終了と囲み	"__aa__",	__aa__	Y	Y
オプションの囲み（あり）	"__aa__",	__aa__	Y	Y
オプションの囲み（なし）	__aa__,	aa__	N	Y
前のフィールドが空白で区切られている場合	__aa__	aa **(3)**	N	(3)

(1) ブランク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。

(2) TERMINATED BY WHITESPACE と指定されたフィールドを除きます。

(3) 後続の空白があるかどうかは、表の他のエントリによって示されるとおり、現行のフィールド指定に依存します。

空白文字の保存

CHAR 型および DATE 型、numeric EXTERNAL 型のすべてのフィールドの中の空白文字を切り捨てないようにするには、制御ファイルに PRESERVE BLANKS を指定します。空白文字の切捨ての詳細は、5-81 ページ「ブランクとタブの切捨て」を参照してください。

PRESERVE BLANKS キーワード

PRESERVE BLANKS を指定すると、オプションの囲みデリミタがない場合には先頭の空白はそのまま残ります。また、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。このキーワードはタブやブランクに対しても有効です。たとえば、次のようなフィールドがあるとします。

```
__aa__,
```

（ここで、アンダースコアはブランクを意味します。）このフィールドをロードするために、次の制御句を指定します。

```
TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
```

この場合、PRESERVE BLANKS を指定すると、先頭と後続の空白はそのままデータとして残ります。PRESERVE BLANKS を指定しなければ、先頭の空白が切り捨てられます。

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、現在のフィールドに対して先頭の空白を含めた範囲が POSITION 句で指定されている場合は、先頭の空白は残ります。このような POSITION 指定がない場合、SQL*Loader は前フィールド末尾の空白をすべて読み込まずに走査して、次にブランク以外の文字またはタブ以外の文字が現れた位置を次のフィールドの開始位置と認識します。

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、さまざまな 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 により INSERT 文中の VALUES 句に対して有効であると認識されたものに限ります。通常は、1 つの値のみ返す SQL 関数ならどれでも使用できます。詳細は、『Oracle8*i* SQL リファレンス』の「式、条件および問合せ」の章の「式」の項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。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
```

SQL 文字列は、二重引用符で囲んで記述します。上記の例では LAST は SQL*Loader のキーワードなので、引用符で囲む必要があります。一方、FIRST は SQL*Loader のキーワードではないので、引用符は不要です。SQL 文字列の中で列名を指定する場合は、エスケープする必要があります。

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 文字列の評価は、NULLIF 句または DEFAULTIF 句の後、DATE マスクよりも前に行われます。SQL 文字列は、RECNUM、SEQUENCE、CONSTANT または SYSDATE のフィールドには使用できません。RDBMS が 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処理は終了します。文字列が認識されても、データベース・エラーが発生すれば、エラーの発生した行は拒否されます。

フィールドの参照

レコード中のフィールドを参照するときは、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。次の例は現在のフィールドの参照方法を示しています。

```
field1 POSITION(1:6) CHAR "LOWER(:field1)"
field1 CHAR TERMINATED BY ','
       NULLIF ((1) = 'a') DEFAULTIF ((1)= 'b')
       "RTRIM(:field1)"
field1 CHAR(7) "TRANSLATE(:field1, ':field1', ':1')"
```

上記の例の最後の文で、*:field1* のみが一重引用符で囲まれていないので、列名として解釈されます。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、5-18 ページ「ファイル名とオブジェクト名の指定」を参照してください。

```
field1 POSITION(1:4) INTEGER EXTERNAL
       "decode(:field2, '22', '34', :field1)"
```

注意 : SQL 文字列は、列オブジェクトのフィールドあるいは、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 フィールドを参照することもできません。

SQL*Loader キーワードと同名のフィールドの参照

同一レコードにおける別のフィールドを参照することもできます。たとえば、次のように指定します。

```
field1 POSITION(1:4)  INTEGER EXTERNAL
            "decode(:field2, '22', '34',  :field1)"
```

一般的な使用方法

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)"
```

演算子の組合せ

複数の演算子を次の例のように組み合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 CHARACTER(10)
       "NVL( LTRIM(RTRIM(:field1)), 'unknown' )"
```

日付マスクの併用

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。たとえば次のようなフィールドを指定したとします。

```
field1 DATE 'dd-mon-yy' "RTRIM(:field1)"
```

このフィールドは、次のように挿入されます。

```
TO_DATE(RTRIM(<field1_value>), 'dd-mon-yyyy')
```

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。たとえば、次のように表示されます。

```
field1 ... "TO_CHAR(:field1, '$09999.99')"
```

この指定によって、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 CHAR 型の列です。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。

しかし、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。

4-28 ページ「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。

列オブジェクトのロード

列オブジェクトは、その属性の用語で制御ファイルに記述されています。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、単純なリレーショナル列に対応するデータフィールドと同じような形式でデータ・ファイルに記述されています。

列オブジェクトのロードに関する例を次に示します。最初は、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、次はデリミタ付きフィールドにデータがある例です。

ストリーム・レコード形式への列オブジェクトのロード

例 5-1 ストリーム・レコード・フォーム（位置を指定したフィールド）へのロード

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、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 5-2 可変レコード・フォーム（終了または囲みフィールド、あるいはその両方）へのロード；

制御ファイルの内容

```
LOAD DATA
INFILE 'sample.dat' "var 6"
INTO TABLE departments
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
1  (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）

```
2  000034101,Mathematics,Johny Q.,30,1024,
   000039237,Physics,"Albert Einstein",65,0000,
```

注意：

1. 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。
2. 最初の 6 文字（斜体）に、次のレコードの長さを指定します。3-18 ページ「新しい SQL*Loader DDL の動作および制限事項」を参照してください。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ています。

ネストした列オブジェクトのロード

例 5-3 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 5-3 ストリーム・レコード・フォーム（終了または囲みフィールド、あるいはその両方）へのロード

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、列オブジェクト内でネストした列オブジェクトを指定します。

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 オブジェクトに限ります）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 5-4 に、例を示します。

例 5-4 ストリーム・レコード・フォーム（位置を指定したフィールド）へのロード

制御ファイル

```
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            Johny 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、マップされた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、5-92 ページ「オブジェクトに対する NULL 値の指定」では、指定されたマップ・フィールドは、dept_no、dept_name、name、age、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）ので、指定されたマップ・フィールドではありません。）

オブジェクトが NULL 値を取る条件が、マップされたフィールドから独立している場合は、前述のとおりに指定してもうまくいかない場合があります。このような場合は、FILLER フィールドを使用できます（3-20 ページ「セカンダリ・データ・ファイル（SDF）および LOBFILES」を参照）。

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。

たとえば、次のように指定します。

例 5-5 ストリーム・レコード・フォーム（終了または囲みフィールド、あるいはその両方）へのロード

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,Johny 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 がブランクかまたは emp_id の属性がブランクの場合、値は NULL（アトミック NULL）になります。

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とんど同じです。例 5-6 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 5-6 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することができません。

すでに実 OID を含むデータをロードする場合でかつ、そのデータを利用する場合は、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 を使用してください。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>)
```

<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 は、その指定された OID が、現行のフォーマットで、グローバルな独自性を保持した OID であるとみなします。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 5-7 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 5-7 OID のロード

制御ファイル

```
   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  Johny Quest, 18, 007, 21E978406D3E41FCE03400400B403BC3,
   Speed Racer, 16, 000, 21E978406D4441FCE03400400B403BC3,
```

注意：

1. OID 句では、s_oid のロード・フィールドが OID を含むように指定しています。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。

REF 列のロード

SQL*Loader では、主キー REF 列と同様に、実 REF 列（参照しているオブジェクトの実 OID を含む REF）もロードできます。

実 REF 列

実 REF をロードする場合、SQL*Loader は、実 OID から 構築される REF 列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します。REF 列に対応するフィールドの記述は、列名の後に REF 指示句を記述することによって行います。

REF 指示句には、引数に、表名と OID が必要です。その引数は、定数として、または動的に（FILLER フィールドを使用して）指定できます。正確な構文の詳細は、5-10 ページ「REF_spec」を参照してください。例 5-8 に、実 REF のロード例を示します。

例 5-8 実 REF 列のロード

制御ファイル

```
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 フィールド自身には、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 指示句を記述する必要があります。REF 指示句には、カンマで区切ったフィールドの名前および定数値のリストが引数として必要です。最初の引数には、表名の後にロードする REF 列がベースとしている主キー OID を指定する引数を記述します。適切な構文については、5-10 ページ「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 5-9 に、主キー REF のロード例を示します。

例 5-9 主キー REF 列のロード

制御ファイル

```
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,
```

LOB のロード

次の項では、SQL*Loader を使用して内部 LOB（BLOB、CLOB、NCLOB）および外部 LOB（BFILE）をロードする方法について説明します。

内部 LOB（BLOB、CLOB、NCLOB）

LOB は非常に大きなデータなので、SQL*Loader では、LOB データをメイン・データ・ファイル（残りのデータを持つインライン）からでも、LOBFILE からでもロードすることができます。5-101 ページ「LOBFILE を使用した LOB データのロード」を参照してください。

LOB データをメイン・データ・ファイルからロードする場合、標準 SQL*Loader のフォーマットを使用できます。LOB データのインスタンスは、あらかじめ決められたサイズのフィールド、デリミタ付きフィールド、または Length-Value Pair フィールドに記述できます。次にこれらのインスタンスの例を示します。

あらかじめ決められたサイズのフィールドの LOB データ

これは LOB データをロードする際、最も高速で、概念的に単純なフォーマットです。

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。後続の空白を切り捨てる場合の詳細は、5-86 ページ「空白の切捨て : 要約」を参照してください。

このフォーマットを使用して LOB をロードする場合、CHAR または RAW をロード時のデータ型として使用する必要があります。

例 5-10 あらかじめ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 が空のデータフィールドが含まれている場合、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、逆のことが発生します（3-19 ページ「DEFAULTIF および NULLIF:」を参照）。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

このフォーマットでは、同じ列（データ・ファイルのフィールド）内の異なるサイズの 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 が区切り文字列を探してデータを走査する必要があるため、パフォーマンスに影響が出ます。3-20 ページ「セカンダリ・データ・ファイル（SDF）および LOBFILES」を参照してください。

例 5-11 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

```
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。CHAR（507）を使用した読取り可能な LOB の最大長は、507 バイトです。
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 `X'7C0A'`）。

Length-Value Pair フィールドの LOB データ

VARCHAR（5-61 ページ「VARCHAR」を参照）、VARCHARC または VARRAW データ型（3-12 ページ「廃棄されたレコードと拒否されたレコード」を参照）を使用して、編成された LOB データを Length-Value Pair フィールドにロードできます。このロード方法では、柔軟性は犠牲になりますが（たとえば、各 LOB をロードする前に、LOB の長さを確認する必要があります）、デリミタ付きフィールドを使用するよりも、高いパフォーマンスを得ることができます。例 5-12 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 5-12 Length-Value Pair フィールドへの LOB データのロード

制御ファイル

```
LOAD DATA
INFILE 'sample.dat' "str '<endrec>\n'"
INTO TABLE person_table
FIELDS TERMINATED BY ','
   (name       CHAR(25),
1  "RESUME"    VARCHARC(3,500))
```

データ・ファイル（sample.dat）

```
Johny Quest,479                Johny Quest
                           500 Oracle Parkway
                           jquest@us.oracle.com
                                 ... <endrec>
2 3  Speed Racer,000<endrec>
```

注意：

1. "\" でのエスケープ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。制御ファイルにおいては、VARCHARC がそのフィールドで、フィールド長が 3 文字、最大サイズが 500 バイトです。
3. VARCHARC の長さサブフィールドは、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。

LOBFILE を使用した 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、LOBFILE からロードすると便利です。LOBFILE では、LOB データのインスタンスは、フィールド（あらかじめ決められたサイズ、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。

ファイルあたり 1 つの LOB 例 5-13 では、それぞれの LOBFILE は、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 データ型の仕様に従った列およびフィールド名を使用してください。たとえば、次のように記述します。

例 5-13 単一の LOB の LOBFILE を使用した LOB データのロード

制御ファイル

```
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',
```

セカンダリ・データ・ファイル（jqresume.txt）

```
        Johny Quest
    500 Oracle Parkway
       ...
```

セカンダリ・データ・ファイル（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取られる、40 バイト長のデータフィールドにマップされています。
2. SQL*Loader は、ext_fname FILLER フィールドの LOBFILE 名を使用します。（CHAR データ型を使用する）LOBFILE の最初のバイトから最初の EOF 文字までのデータをロードします。実在する 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。また、3-22 ページ「静的・動的な LOBFILE および SDF 指定」も参照してください。

あらかじめ決められたサイズの LOB

例 5-14 では、制御ファイルに、列にロードする LOB のサイズを指定しています。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。あらかじめ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す。困難な点は、すべての LOB データが必ずしも同じサイズであるとは限らないことです。

例 5-14 あらかじめ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

```
LOAD DATA
INFILE 'sample.dat'
INTO TABLE person_table
FIELDS TERMINATED BY ','
   (name     CHAR(20),
   ext_fname FILLER CHAR(40),
1  "RESUME"    LOBFILE(CONSTANT '/usr/private/jquest/jqresume')
               CHAR(2000))
```

データ・ファイル（sample.dat）

```
Johny Quest,
Speed Racer,
```

セカンダリ・データ・ファイル（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。

デリミタ付きフィールドの LOB

例 5-15 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。このフォーマットでは、サイズの異なる LOB を同じ列にロードすることは問題になりません。ただし、このような柔軟性によって、SQL*Loader が区切り文字列を探してデータを走査する必要があるため、パフォーマンスに影響が出ます。

例 5-15 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,
```

セカンダリ・データ・ファイル（jqresume.txt）

```
    Johny Quest
500 Oracle Parkway
   ... <endlob>
    Speed Racer
400 Oracle Parkway
   ... <endlob>
```

注意：

1. `max-length` (2000) で、フィールドの最大長を指定すると、メモリー使用量を最適化できます。`max-length` を指定する場合、過少な値は指定しないでください。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 句を使用できます。ENCLOSED BY 句は、LOBFILE（LOBFILE 内の LOB には順序が不要）での LOB の相対位置指定について多少柔軟性があります。

Length-Value Pair で指定した LOB

この例では、LOBFILE のそれぞれの LOB は、フィールドの先頭でデータ長を定義します。VARCHAR（5-61 ページ「VARCHAR」を参照）、VARCHARC または VARRAW データ型（3-12 ページ「廃棄されたレコードと拒否されたレコード」を参照）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。

このロード方法では、柔軟性は犠牲になりますが（たとえば、各 LOB をロードする前に、LOB の長さを確認する必要があります）、デリミタ付きフィールドを使用するよりも、高いパフォーマンスを得ることができます。

例 5-16 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

```
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,
```

セカンダリ・データ・ファイル（jqresume.txt）

```
2       0501Johny Quest
        500 Oracle Parkway
           ...
3       0000
```

注意：

1. VARCHARC(4, 2000) のエントリでは、LOBFILE の LOB が Length-Value Pair フォーマットであり、最初の 4 バイトが長さを示している、という指定になります。max_length は、フィールドの最大サイズが 2000 であることを示します。
2. Johnny 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ています。
3. このエントリは、空の（NULL ではない）LOB を示しています。

LOBFILE から LOB をロードする場合の考慮事項

LOBFILE から LOB をロードする場合は、次の点に注意してください。

- 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。
- 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、max_length を指定すると、メモリー使用量の最適化のヒントに使用されます。max_length には、本来の最大長よりも小さい値を指定しないでください。

外部 LOB（BFILE）

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・ファイルに非構造化バイナリ・データを格納します。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ています。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、後で作成することができます。必要なオブジェクトは、すでに作成されているとみなされます（サーバー・ファイルシステム上の物理ディレクトリの論理エイリアス名）。詳細は、『Oracle8*i* 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 句を記述することによって行います。BFILE 句には、引数として DIRECTORY OBJECT 名の後に BFILE 名が必要です。いずれも文字列定数として指定されるか、または動的に他のフィールドを介して使用されます。詳細は、『Oracle8*i* SQL リファレンス』を参照してください。

次の 2 つの BFILE のロード例のうち、例 5-17 では、1 つのファイル名のみ動的に指定されています。例 5-18 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 5-17 BFILE を使用したデータのロード：ファイル名のみ動的に指定

制御ファイル

```
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、そのまま使用されるので、文字列は、大文字にせず、そのまま指定します。

例 5-18 BFILE を使用したデータのロード : ファイル名および OBJECT_DIRECTORY を動的に指定

制御ファイル

```
LOAD DATA
INFILE sample.dat
INTO TABLE planets
FIELDS TERMINATED 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。

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、メイン・データ・ファイル（データ・インライン）またはセカンダリ・データ・ファイル（データ・アウトライン）のいずれからでもロードできます。3-21 ページ「セカンダリ・データ・ファイル（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および要素の数を、DDL 構文の COUNT を使用して指定できます。COUNT のパラメータとして使用したそのフィールドは、制御ファイルで COUNT 句自身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 句固有のものです。また、COUNT(0) または COUNT(x) (x==0) で、NULLIF 指示句によって上書きされる場合を除き、空のコレクション（NULL ではない）になります。5-14 ページ「count_spec」を参照してください。

- TERMINATED BY および ENCLOSED BY 指示句を使用することによって、独自のデリミタを指定できます。

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（5-90 ページ「列オブジェクトのロード」を参照）。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、前述の機能を使用します。
- コレクションの記述には、セカンダリ・データ・ファイル（SDF）の指定ができます。
- フィールド名を引数として使用する句または指示句は、同じコレクションのフィールドに対する DDL 指定を除き、コレクション内のフィールド名を使用できません。例 5-19 において、name、age および empid は、dept_no、dname、emp_cnt、emps または projects に対する NULLIF または DEFAULTIF 句のフィールド条件指定で使用できません。
- フィールド・リストには、1 つの非 FILLER フィールドおよびいくつかの 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 の両方の構文図の詳細は、5-3 ページ「SQL*Loader のデータ定義言語（DDL）構文図」を参照してください。

例 5-19 に、VARRAY およびネストした表のロード例を示します。

例 5-19 VARRAY およびネストした表のロード

制御ファイル

```
   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 ,
1) emp_cnt    FILLER INTEGER EXTERNAL(5),
2) emps       VARRAY COUNT(emp_cnt)
3)    (name)  FILLER CHAR(10),
      emps    COLUMN OBJECT NULLIF emps.emps.name=BLANKS
         (name)  CHAR(30),
         age     INTEGER EXTERNAL(3),
         emp_id  CHAR(7) NULLIF emps.emps.emp_id=BLANKS)),
   mysid      FILLER CHAR(32),
4) projects   NESTED TABLE SDF(CONSTANT 'pr.txt' "fix 71")
              SID(mysid) TERMINATED BY ";"
       (project_id  POSITION(1:5) INTEGER EXTERNAL(5),
       project_name POSITION(7:30) CHAR
                    NULLIF projects.project_name=BLANKS,
```

```
        p_desc_src   FILLER POSITION(35:70) CHAR,
5)      proj_desc    LOBFILE( projects.p_desc_src) CHAR(2000)
                     TERMINATED BY "<>\n"))
```

データ・ファイル（sample.dat）

```
  101,Math,2, ,"J. H.",28,2828, ,"Cy",123,9999,21E978407D4441FCE03400400B403BC3|
6) 210,"Topologic Transforms", ,21E978408D4441FCE03400400B403BC3|
```

セカンダリ・データ・ファイル（SDF）（pr.txt）

```
   21034 Topological Transforms      '/mydir/projdesc.txt';
7) 77777 Impossible Proof;
```

セカンダリ・データ・ファイル（LOBFILE）（'/mydir/projdesc.txt'）

```
8) Topological Transforms equate ...........<>
   If there is more then one LOB in the file, it starts here .....<>
```

注意：

1. emp_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。
2. 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 句を使用することです。5-80 ページ「DEFAULTIF 句」を参照してください。VARRAY 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。
3. フルネームによるフィールド参照（ドット表記）は、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を解決します。
4. このエントリでは、'pr.txt' と呼ばれる SDF をデータのソースとして指定しています。また、SDF 内で固定レコード形式を指定します。TERMINATED BY 句では、ネストした表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。SID 句では、ネストした表に対して SID を含むフィールドを指定します。また、SID 句が、データ・ファイルから損失したレコードの SID を除いて指定されているとき、そのレコードの SID は、システムによって生成されます。データ・ファイルでの SID の指定は、オプションであり、特にパフォーマンスを向上させるものではありません。
5. p_desc_src が NULL の場合、DEFAULTIF 句は proj_desc LOB を空に初期化します。5-80 ページ「DEFAULTIF 句」を参照してください。

6. emp_cnt フィールドは NULL で、DEFAULTIF 句の下にあり、0（空の VARRAY）に変換されます。"mysid" 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。SQL*Loader では、SID に対してこれ以外の妥当性チェックは実行されません。
7. TRAILING NULLCOLS 句の存在によって、p_desc_src に対応するフィールドは存在しませんが、p_desc_src は、NULL に初期化されます。
8. LOB 終了記号 <> の後に、改行文字が続きます。

親表を子表から分割してロード

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 がロード時に分かっていれば（つまり、SID がデータとともにデータ・ファイルにある場合）親表と子表を個別にロードできます。

例 5-20 親表をユーザー定義 SID とともにロードする

制御ファイル

```
   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 フィールドであり、SID 句に対する引数として指定できます。

VARRAY 列ロード時のメモリーの問題

- VARRAY 列をロードする場合、VARRAY は、データベースにロードされる前に、クライアントのメモリーで作成されます。VARRAY の各要素は、データベースにロードする前に、クライアント・メモリーの 4 バイトを必要とします。そのため、1000 個の要素とともに VARRAY をロードしている場合、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、少なくとも 4000 バイトのクライアント・メモリーが必要です。多くの場合、SQL*Loader では、そのような 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 パラメータは、SQL*Loader によって、レコードのロードに対して割り当てられるメモリーの上限（デフォルトでは 64K）を指定します。表にロードされる各フィールドのサイズに基づき、1 回のトランザクションでロードできる行数に対して BINDSIZE で指定するバイト数が、SQL*Loader によって決定されます。ROWS パラメータを使用して、計算結果よりも小さい行数を、強制的に使用させることができます。ROW に対して大きな値を使用すれば、それだけトランザクションが少なくなり、パフォーマンスが向上します。
- 非常に大きな VARRAY、または多数の小さな VARRAY によって、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 に対して、より小さな値を指定し、再度ロードしてください。

6

SQL*Loader コマンド行リファレンス

この章では、コマンド行キーワードを使用した SQL*Loader の実行方法を示します。ここで示すコマンド行キーワードの詳細は、第 5 章「SQL*Loader 制御ファイル・リファレンス」を参照してください。

この章では次の事項を解説します。

- SQL*Loader コマンド行
- コマンド行キーワード
- 索引メンテナンス・オプション
- 終了コードによる結果の検査と表示

SQL*Loader コマンド行

コマンド行からキーワードを使用して SQL*Loader を起動する方法を示します。

追加情報 : 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。以降の例では、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、Oracle 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。キーワードを何も指定しないで SQL*Loader を起動すると、使用可能なキーワードとデフォルト値を示すヘルプ画面が表示されます。次の表示例は、各オペレーティング・システムに共通するデフォルト値を示したものです。

```
sqlldr
...
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
                (System-dependent default)
       silent — Suppress messages during run
                (header, feedback, errors, discards, partitions, all)
       direct — Use direct path
                (Default FALSE)
      parfile — Parameter file: name of file that contains
                parameter specifications
     parallel — Perform parallel load
                (Default FALSE)
     readsize — Size (in bytes) of the read buffer
         file — File to allocate extents from
```

コマンド行キーワードの使用方法

オプションとして、キーワードをカンマで区切ることができます。キーワードは任意の順序で入力できます。キーワードの後には、有効な引数を指定します。

たとえば、次のように記述します。

```
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
```

制御ファイル内でのキーワードの指定

コマンド行の長さが、ご使用のシステムの許容最大長を超える場合は、制御ファイルのキーワードである OPTIONS を使用して、コマンド行キーワードを制御ファイル中で指定できます。5-18 ページ「OPTIONS」を参照してください。

キーワード PARFILE で指定した別のファイルに指定することもできます（6-6 ページ「PARFILE（パラメータ・ファイル）」を参照）。ほとんど変更しないキーワード・エントリを使用する場合は、上記のいずれかのファイルで指定してください。このようにして指定されたキーワードは、コマンド行から別の指定を行うことにより、取り消すこともできます。

コマンド行キーワード

この項では、使用可能な SQL*Loader の各コマンド行キーワードについて説明します。

BAD（不良ファイル）

BAD は、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、デフォルトとして制御ファイル名に拡張子 .BAD の付いた名前が適用されます。不良ファイルは入力データ・ファイルと同じ形式で作成されるので、更新または修正を行った後、同じ制御ファイルを使用してロードできます。

コマンド行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、コマンド行で指定した不良ファイル名の方が優先されます。

BINDSIZE（最大サイズ）

BINDSIZE には、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。バインド配列の詳細は、5-74 ページ「バインド配列サイズの決定」を参照してください。デフォルト値は 65536 バイトです。また、6-7 ページ「READSIZE（読込みバッファ）」も参照してください。

CONTROL（制御ファイル）

CONTROL には、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、デフォルト拡張子として CTL が適用されます。省略すると、SQL*Loader からファイル名の入力を要求されます。

注意 : 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。ご使用のオペレーティング・システムのドキュメントを参照してください。

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、次の点に注意してください。

- バックスラッシュ以外の文字の前にあるバックスラッシュは通常の文字として処理される。
- 連続する 2 つのバックスラッシュは 1 つのバックスラッシュとして処理される。
- 連続する 3 つのバックスラッシュは 2 つのバックスラッシュとして処理される。
- パスを引用符で囲む場合は、複数のバックスラッシュをエスケープする必要はない。ただし、オペレーティング・システムによっては、引用符自体をエスケープしなければならない場合があります。

DATA（データ・ファイル）

DATA 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。ファイル名を指定しない場合、デフォルトとして制御ファイル名が適用されます。ファイル拡張子またはファイル・タイプを指定していない場合は、デフォルトとして .DAT が適用されます。

注意 : ファイル処理オプションを指定する場合、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。

DIRECT（データ・パス）

DIRECT には、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 はダイレクト・パス・ロードを指定します。FALSE は従来型パス・ロードを指定します。デフォルトは FALSE です。ロード方法については、第 8 章「SQL*Loader: 従来型パス・ロードと ダイレクト・パス・ロード」を参照してください。

DISCARD（廃棄ファイル）

DISCARD には、SQL*Loader が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、表に挿入されなかったが拒否もされなかったレコードが保存されます。ファイル名を指定しない場合、デフォルトは DSC になります。

このファイルは、入力データ・ファイルと同じ形式になります。したがって、適切な更新や修正後、同じ制御ファイルを使用してロードできます。

コマンド行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、コマンド行で指定した廃棄ファイル名の方が優先されます。

DISCARDMAX（許容されない廃棄数）

DISCARDMAX では、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。デフォルトでは、すべての廃棄レコードが許可されます。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）

ERRORS では、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 パラメータの値を超えると、ロード処理が中止されます。デフォルト値は 50 です。エラーをいっさい認めない場合は、ERRORS=0 を設定します。エラー発生を無制限に認める場合には、非常に大きな数を指定します。

単表のロードの場合、このエラー上限を超える数のエラーが発生すると、ロード処理が中止されます。ただし、エラー上限に達する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、複数の表のロードの場合は、エラー上限を超えるエラーが発生しても、すぐにはロード処理は中止されません。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が確実にすべての表にロードされます。また、拒否された行がすべての表から取り除かれます。

どちらの場合でも 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。

FILE（ロード先ファイル）

FILE では、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 の各プロセスごとに 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るときのディスクの競合を最小限におさえることができます。詳細は、8-26 ページ「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）

LOAD では、（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。デフォルトでは、すべてのレコードがロードされます。実際のレコード件数が指定された最大数より少なくても、エラーは発生しません。

LOG（ログ・ファイル）

LOG では、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。ファイル名を指定しない場合、デフォルトとして制御ファイル名にデフォルトの拡張子（LOG）の付いた名前が適用されます。

PARFILE（パラメータ・ファイル）

PARFILE では、コマンド行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、コマンド行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（"="）の前後に空白を入れられないものもあります。

PARALLEL（パラレル・ロード）

PARALLEL では、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。PARALLEL ロードの詳細は、8-26 ページ「パラレル・データ・ロード・モデル」を参照してください。

READSIZE（読込みバッファ）

コマンド行パラメータ READSIZE で、読込みバッファのサイズを（バイトで）指定します。デフォルト値は 65536 バイトですが、指定できる読込みバッファのサイズは、ご使用のシステムによって異なります。

従来型パスの方式を使用する場合、バインド配列は、読込みバッファのサイズによって制限されるので、読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。

たとえば、次のように記述します。

```
sqlldr scott/tiger control=ulcas1.ctl readsize=1000000
```

この場合、SQL*Loader によって、外部のデータ・ファイルから 1000000 バイト単位で、コミットを実行する前に読込みを実行できます。

注意 : READSIZE および BINDSIZE パラメータ両方のデフォルト値は、65536 バイトです。READSIZE に指定したサイズよりも小さいサイズを BINDSIZE に指定した場合、BINDSIZE の値は、自動的に READSIZE の値に増やされます。

また、READSIZE に BINDSIZE よりも小さい値を指定した場合、READSIZE の値も増やされます。

また、このパラメータは、ダイレクト・パス・ロードで使用する READBUFFERS キーワードには影響しません。

また、6-4 ページ「BINDSIZE（最大サイズ）」も参照してください。

ROWS（1 回にコミットする行数）

従来型パス・ロードの場合 : バインド配列の行数を指定します。デフォルトは 64 です。（バインド配列の詳細は、5-74 ページ「バインド配列サイズの決定」を参照してください。）

ダイレクト・パス・ロードの場合 : 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、ロード終了時点でデータ・セーブが 1 回実行されます。詳細は、8-12 ページ「データ・セーブ」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。このバッファが満杯になった時点でのみデータがデータベースに書き込まれるので、指定した ROWS の値はおよその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:   Production on Wed Feb 24 15:07:23...
Copyright (c) Oracle Corporation...
```

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 が次のようなデータ・エラー・メッセージを表示する場合もあります。

```
Record 4: Rejected - Error on table EMP
ORA-00001: 一意制約：(string.string) に反しています。
```

SILENT キーワードを引数とともに指定して、これらのメッセージが出力されないようにできます。

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、コマンド行の引数で次のように指定します。

```
SILENT=(HEADER, FEEDBACK)
```

適切なキーワードを使用すると、それぞれ次の部分が抑制されます（複数も可）。

HEADER	画面に通常表示される SQL*Loader のヘッダー・メッセージを表示しません。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・メッセージを表示しません。
ERRORS	レコードに Oracle エラーが発生したためにそのレコードが不良ファイルに書き込まれた場合に、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、受付けを拒否された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・ロードにおいて、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	すべてのキーワードを有効にします。

SKIP（スキップされるレコード）

SKIP では、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。デフォルトではレコードは 1 件もスキップされません。

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す。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。詳細は、5-34 ページ「複数表へのロード継続（従来型ロード）」を参照してください。

USERID（ユーザー名 / パスワード）

USERID では、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。Net8 データベース・リンクを使用して、リモート・データベースに対して従来型パス・ロードを行うこともできます。Net8 の詳細は、『Oracle8*i* Net8 管理者ガイド』を参照してください。データベース・リンクの詳細は、『Oracle8*i* 分散システム』を参照してください。

索引メンテナンス・オプション

Oracle8*i* では、次に示す 2 つの新しい索引メンテナンス・オプションが使用できます（デフォルトは FALSE です）。

- SKIP_UNUSABLE_INDEXES={TRUE | FALSE}
- SKIP_INDEX_MAINTENANCE={TRUE | FALSE}

SKIP_UNUSABLE_INDEXES

SKIP_UNUSABLE_INDEXES オプションは、従来型ロードとダイレクト・パス・ロードのどちらにも適用できます。

SKIP_UNUSABLE_INDEXES=TRUE オプションを指定すると、表のロードとともに、ロード開始前の状態が Index Unusable（IU）である索引もロードされます。ロード時点で IU 状態でない索引には、SQL*Loader によってメンテナンスされます。一方、ロード時点で 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、UNIQUE 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 と指定してロードを実行すると、従来型ロードとダイレクト・パス・ロードとでは処理内容が多少異なります。

- 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あれば、そのレコードは挿入されずに拒否される。
- 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、ロード処理は終了する。

SKIP_INDEX_MAINTENANCE

SKIP_INDEX_MAINTENANCE={TRUE | FALSE} は、ダイレクト・パス・ロードにおける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 Index Unusable 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 Index Unusable 状態が保持されます。

SKIP_INDEX_MAINTENANCE オプションの機能は、次のとおりです。

- ローカル索引とグローバル索引の両方に適用できる。
- 索引を持つオブジェクトのパラレル・ロードを実行できる（PARALLEL オプションとともに指定します）。
- 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る（INTO TABLE 句で PARTITION キーワードを指定します）。
- ロードによって Index Unusable 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）作成する。

終了コードによる結果の検査と表示

Oracle SQL*Loader では、SQL*Loader の実行結果を完了後すぐに確認できます。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、コマンド行やスクリプトから SQL*Loader を起動したときにもその結果をチェックできます。次に、ロード結果と、通知される終了コードを示します。

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廃棄された	EX_WARN
ロードが中断された	EX_WARN
コマンド行 / 構文エラー	EX_FAIL
SQL*Loader に対する Oracle の回復不能エラー	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_FTL

UNIX の場合、終了コードは次のようになります。

```
EX_SUCC0
EX_FAIL1
EX_WARN2
EX_FTL3
```

シェルの終了コードをチェックして、ロードの結果を判断できます。たとえば、SQL*Loader コマンドをスクリプトで使用して、スクリプト内で終了コードをチェックできます。

```
#!/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 exectuion exited with EX_WARN, see logfile" ;;
3) echo "SQL*Loader execution encountered a fatal error" ;;
*) echo "unknown return code";;
esac
```

7

SQL*Loader: ログ・ファイル参照

SQL*Loader が実行を開始すると、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 が正常に実行したことの記録として作成されます。しかし、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。

この章では、次のログ・ファイル・エントリについて説明します。

- 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

ヘッダー情報

ヘッダー・セクションには、次のエントリがあります。

- 実行日
- ソフトウェアのバージョン番号

例を次に示します。

```
SQL*Loader: Version 8.0.2.0.0 - Production on Mon Nov 26...
Copyright (c) Oracle Corporation...
```

グローバル情報

グローバル情報セクションには、次のエントリがあります。

- すべての入出力ファイル名
- コマンド行引数のエコー
- 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。

例を次に示します。

```
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

表情報セクションでは、ロードされる各表について次のエントリがあります。

- 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 句の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 または APPEND、REPLACE の設定。
- 列情報。次のようなエントリが書き込まれます。
  - 列の位置または長さ、データ型、デリミタ（データ・ファイルに定義されている場合）
  - RECNUM または SEQUENCE、CONSTANT（指定されている場合）
  - DEFAULTIF または NULLIF（指定されている場合）

例を次に示します。

```
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
```

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、次のエントリがあります。

- SQL*Loader/Oracle のデータ・レコード・エラー
- 廃棄されたレコード

例を次に示します。

```
Record 2: Rejected - Error on table EMP.
ORA-00001:  unique constraint <name> violated
Record 8: Rejected - Error on table EMP, column DEPTNO.
ORA-01722:  invalid number
Record 3: Rejected - Error on table PROJ, column PROJNO.
ORA-01722:  invalid number
```

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あります。

- ロードされた行の数
- ロード対象であったが、データ・エラーのためロードを拒否された行の数
- どの WHEN 句条件にも合致せず、廃棄された行の数
- 関連するフィールドがすべて NULL であった行の数

例を次に示します。

```
The following indexes on table EMP were processed:
Index EMPIX was left in Direct Load State due to
ORA-01452: cannot CREATE UNIQUE INDEX; duplicate keys found

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.
```

サマリー統計

サマリー統計セクションでは、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、下記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
  - 拒否されたレコード
  - 廃棄されたレコード
- 実行開始 / 終了時刻
- 総経過時間
- 総 CPU 時間（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 時間は含みません）

例を次に示します。

```
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 Mon Nov 26 10:46:53 1990
Run ended on Mon Nov 26 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

Oracle のログ・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 以降統計レポートに関する変更はない。
- パーティション表をダイレクト・ロードした場合は、（Oracle7 の）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る。
- 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示される。

単一パーティション・ロード時の統計

- 表の列の説明の中にパーティション名が示される。
- エラー・メッセージの中にパーティション名が示される。
- 統計一覧の中にパーティション名が示される。

表ロード時の統計

- 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る。
- 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ない。
- 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 以降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 以降統計レポートに関する変更はありません。

メディア回復が使用可能でない場合、ロード時の情報はログに記録されません。したがって、メディア回復が使用禁止になっていると、ログを記録する操作の要求は受け付けられません。

silent=partitions|all 指定時の統計

パーティション表をダイレクト・ロードする場合、コマンド行オプション `silent=partitions` を指定すると、パーティション別の統計セクションはログ・ファイルには出力されません。

オプション `silent=all` を指定すると、`partitions` フラグも指定されたことになるので、パーティション別統計値が出力されません。

8

SQL*Loader: 従来型パス・ロードとダイレクト・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。この章では、次のトピックについて説明します。

- データのロード方法
- 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
- 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・ロードによるロードの例は、4-25 ページ「事例 6: ダイレクト・パス・ロード方式を使用したロード」を参照してください。その他の事例では、従来型パスによるロード方法を使用しています。

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。

- 従来型パスによるロード
- ダイレクト・パスによるロード

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ので、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。つまり、ダイレクト・パスによるロードでは、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生じないので、ディスク速度に近い速度でデータをロードできます。データベース・ファイルへのアクセス方法に特有の問題（制限、セキュリティ、バックアップなど）についても、この章で説明します。

従来型パスによるロード

従来型パスによるロード（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 Tools およびアプリケーションで使用されます。

SQL*Loader で従来型パスによる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であるとみなされ、競合が生じます。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ので、さらにオーバーヘッドが発生します。

挿入が発生するたびに、Oracle は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）を探し、そこにデータを書込みます。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。

単一パーティションの従来型パス・ロード

従来型パスでロードする場合、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、SQL*Loader は次のような形式の 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、従来型パスよりダイレクト・パスによるロードを使用してください。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、必然的に従来型パス・ロードを使用しなければならない場合もあります。次のような場合には、従来型パスを使用してください。

- 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 が、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。

- 異種プラットフォームから SQL*Net を使用してデータをロードする場合。

  Net8 を使用してダイレクト・パス・ロードでデータをロードするには、双方のシステムが同じコンピュータ系列に属し、同じキャラクタ・セットを使用している必要があります。これらの条件を満たしていても、ネットワークのオーバーヘッドによってロード・パフォーマンスがかなり損なわれることがあります。

- 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していません。

- 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、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と、この処理にかかる時間が節約した分より多くなる可能性があります。

- SQL 関数をデータ・フィールドに適用する場合。

  SQL 関数はダイレクト・パス・ロードでは使用できません。SQL 関数の詳細は、5-87 ページ「フィールドへの SQL 演算子の適用」を参照してください。

ダイレクト・パスによるロード

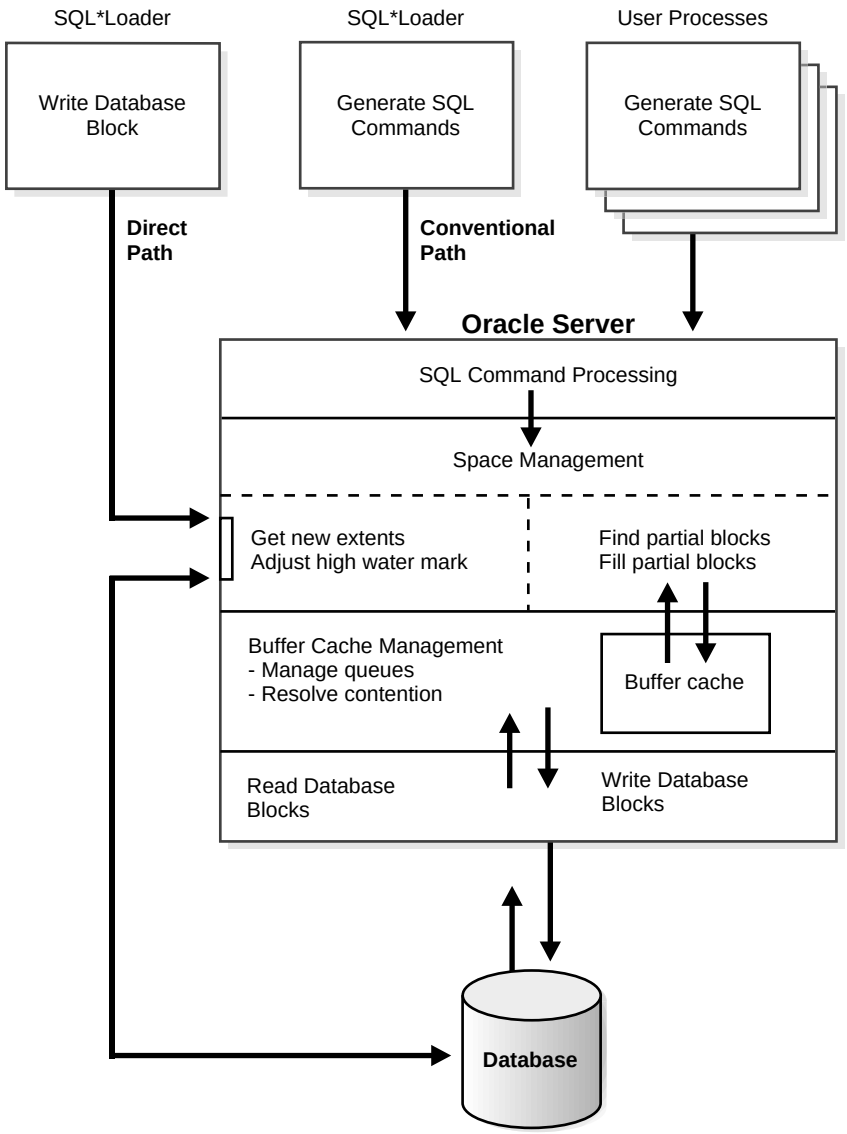
従来型パス・ロードでは、バインド配列バッファが一杯になるまでデータが書き込まれた後 SQL INSERT コマンドで Oracle に渡されるのに対して、ダイレクト・パス・ロードの場合、SQL*Loader の制御ファイルに指定された記述に従って入力データが解析され、各入力フィールドのデータが、対応する Oracle の列のデータ型に変換されて、列配列構造体（< 長さ , データ > の組合せの配列）が作成されます。

次に SQL*Loader によって、列配列構造体を使用して Oracle データ・ブロックがフォーマットされ、索引キーが作成されます。それから、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（ホストの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ので、ロード・パフォーマンスが向上します。

図 8-1 では、従来型パスとダイレクト・パスによるロードで、データベースへの書込みがどのように行われるかを示しています。

図 8-1 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込み



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、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。

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 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 ...
```

または

```
LOAD INTO TABLE T subpartition (P) VALUES ...
```

パーティション表の 1 つのパーティションまたはサブパーティションをロードしているときも、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 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の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・ポイントの上限を調整するといった動作も発生します。上限については、8-12 ページ「データ・セーブ」を参照してください。

ダイレクト・パスによるロードの利点

ダイレクト・パスによるロードは、従来型パス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ので、空きブロックを探すための読込み処理が不要で、書込みも少ない。

- SQL*Loader は SQL INSERT コマンドを実行しないので、Oracle データベースの処理負荷が減る。
- バインド配列バッファを使用せず、フォーマットされたデータベース・ブロックに直接書き込む。
- ダイレクト・パス・ロードでは、表と索引をロード開始時にロックするよう Oracle に要求を出し、ロード完了時にロック解除の要求を出す。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。
- ダイレクト・パス・ロードでは、マルチ・ブロックの非同期 I/O を使用してデータベース・ファイルに書き込む。
- ダイレクト・パス・ロードを使用するプロセスは、他の Oracle ユーザーと競合する 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込み I/O を実行する。
- 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レーション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できる。
- ロードする表が空である場合、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る。索引は、単にデータが挿入された時点で入力されます。
- ダイレクト・パスによるロードでは、インスタンス障害から回復するための REDO ログ・ファイル・エントリを必要としない。したがって、以下の場合は、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 のとき
  - ロードの UNRECOVERABLE オプションの値が Y のとき
  - ロードするオブジェクトの NOLOG 属性が設定されているとき

8-14 ページ「インスタンス回復とダイレクト・パス・ロード」を参照してください。

ダイレクト・パスを使用する場合

上記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。表が空であっても空でなくても、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 PARALLEL でロードする場合。8-26 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。
- 現行のセッションではサポートできないキャラクタ・セットでデータをロードする場合、またはデータベース・キャラクタ・セットへの変換を従来の方法で実行するとエラーが発生するおそれがある場合。

ダイレクト・パス・ロード使用上の制限

ダイレクト・パス・ロード方法を使用する場合、3-15 ページ「従来型パス・ロードとダイレクト・パス・ロード」の一般的なロード条件の他に、次に示す条件が満たされ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チェックするには、MONITOR TABLE という Enterprise Manager コマンドを使用して、ロードする表のオブジェクト ID を検索してください。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てください。
- 制御ファイルで SQL 文字列が使用されていないこと。

次の機能は、ダイレクト・パス・ロードでは使用できません。

- 列オブジェクトのロード
- LOB のロード
- VARRAY のロード
- ネストした表のロード
- システム生成 OID のあるオブジェクト表に対する OID の指定
- SID の指定
- REF 列のロード
- BFILE 列のロード
- 64K を超える物理レコード（コマンド行オプション READSIZE によって設定）

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、参照制約およびチェック制約が使用禁止（Disable）であること。
- トリガーが使用禁止（Disable）であること。

整合性制約

ダイレクト・パスによるロード時には、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限りません。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。

ロード中とロード後には、UNIQUE 制約が施行されます。UNIQUE 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。この章の 8-21 ページ「ダイレクト・ロード、整合性制約およびトリガー」の項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・パスでロードする場合、データベースに定義されている DEFAULT 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 句を使用して指定します。DEFAULTIF 句の詳細は、5-80 ページ「DEFAULTIF 句」を参照してください。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示していなければなりません。シノニムがビューのシノニムであったり、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。

バージョンの要件

SQL*Loader のダイレクト・ロードは、同じバージョンのデータベースに対してのみ実行できます。たとえば、SQL*Loader バージョン 7.1.2 のダイレクト・パス・ロード機能を使用して、Oracle バージョン 7.1.3 のデータベースにデータをロードすることはできません。

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レーションの時点でダイレクト・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・ロード・モードで起動するには、コマンド行または（使用している場合）パラメータ・ファイルにおいて、次の形式で DIRECT パラメータを TRUE に設定してください。

```
DIRECT=TRUE
```

4-25 ページ「事例 6: ダイレクト・パス・ロード方式を使用したロード」にある例を参照してください。

索引の作成

ダイレクト・パス・ロードでは、一時記憶領域を使用することによって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）セグメントに入れられます。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はすべてマージが完了するまで記憶領域を必要とします。最後に、古い索引と一時セグメントが削除されます。

従来型パスによるロードでは、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領域は不要ですが、処理に時間がかかります。

SINGLEROW オプション

メモリーに制限があるシステムでは、SINGLEROW オプションを使用することによっても、パフォーマンスは向上します。詳細は、5-43 ページ「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には、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、索引にキーが直接付けられます。詳細は、8-16 ページ「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。

索引記憶領域の要件

索引自体を格納するために必要な領域の大きさを計算する定式は、『Oracle8*i* 管理者ガイド』の「データベース・ファイルの管理」に記載されています。ロードが完了するまでは新しい索引と古い索引の 2 つの索引が存在するため、計算のときに考慮してください。

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）は、次の定式で計算することができます。

```
1.3 * key_storage

key_storage = (number_of_rows) *
     ( 10 + sum_of_column_sizes + number_of_columns )
```

上記の定式における列（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。データが正反対の順序に並んでいると、ソートには 2 倍のキー記憶領域が必要となるので、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があればよいので、そのときの定数の値は 1.0 に下がります。詳細は、8-16 ページ「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。

索引使用禁止状態（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も最新のものになると、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・オプションでのロード実行時に、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 が索引のための領域を使い果たし、索引が更新できない。
- データが SORTED INDEXES 句で指定した順序になっていない。
- インスタンス障害が起こったか、または索引作成中に Oracle シャドウ・プロセスが失敗した。
- 一意の索引内に重複キーがある。
- データ・セーブ・ポイントが使用中である。データ・セーブ・ポイント発生後にロードが失敗したか、またはキーボードからの中断によって終了された。

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';
```

ある索引パーティションが使用不可（*unusable*）状態かどうかを調べるには、次の問合せを行います。

```
SELECT INDEX_NAME,
       PARTITION_NAME,
       STATUS FROM USER_IND_PARTITIONS
       WHERE STATUS != 'VALID';
```

表の所有者でない場合、USER_INDEXES のかわりに、ALL_INDEXES または DBA_INDEXES を検索してください。パーティション索引については、USER_IND_PARTITIONS のかわりに、ALL_IND_PARTITIONS および DBA_IND_PARTITIONS を検索してください。

データ・セーブ

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喪失を防ぐことができます。前回のデータ・セーブ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 オプションを使用して処理済み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。ロードが終了したら、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障害の詳細は、8-13 ページ「回復」を参照してください。

注意 : 索引はデータ・セーブでは保護されません。SQL*Loader はデータのロードが完了するまで索引を作成しないからです（事前ソートされたデータを空の表にロードする場合に限り、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・ロードにおいてデータ・セーブをいつ行うかを設定します。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 が入力ファイルから読み込む行数です。

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・ブロックと同じフォーマットのデータ・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の数は、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 は、必ずデータベース・ブロックの充填に必要な行数を読み込みます。廃棄されたり、受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。したがって、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、破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、パフォーマンスが悪化します。

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似ていますが、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る。
- データ・セーブ後は、行をロールバックできない。

一方、従来型との主な相違点は、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。

回復

ダイレクト・パス・オプションを指定すると、SQL*Loader のデータ回復機能が完全にサポートされます。回復には大きく分けて 2 種類あります。

メディア回復　　メディア回復は、喪失したデータベース・ファイルを回復します。データベース・ファイルを喪失した場合に、それを回復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。

インスタンス回復	インスタンス回復は、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場合の、システム障害を回復します。Oracle 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回復できます。

回復の詳細は、『Oracle8*i* 管理者ガイド』を参照してください。

インスタンス回復とダイレクト・パス・ロード

SQL*Loader はデータベース・ファイルに直接書き込むようになっています。そのため、インスタンスを再度開始したときには、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は、自動的にデータベース・ファイルに存在しています。変更が REDO ログ・ファイルに記録されていなくても、インスタンス回復は可能です。

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、表または表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の詳細は、8-11 ページ「索引使用禁止状態（Index Unusable）のままの索引」を参照してください。

メディア回復とダイレクト・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回復が可能になります。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、メディア回復はできません。

ロード中に失われたデータベース・ファイルを回復するには、従来型パスでロードしたデータを回復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復元する。
2. RECOVER コマンドを使用して、表領域を回復する。（RECOVER コマンドの詳細は、『Oracle8*i* 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。）

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、PIECED オプションを指定するか、READBUFFERS の数を指定します。この項ではこれらの 2 つのオプションについて解説します。

データを PIECED としてロードする

データが論理レコードの最終列である場合、PIECED オプションを指定すると、データを分割してロードできます。この指定を行う構文については、5-4 ページ「高水準の構文図」を参照してください。

列を PIECED と宣言することにより、フィールドを分割した状態にして一度に 1 つのバッファで処理するように、ダイレクト・パス・ローダーに指示します。

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・パスでのみ有効である。
- 1 つの表につき 1 フィールドのみを PIECED にできる。
- PIECED フィールドは論理レコードの最終フィールドでなければならない。
- WHEN 句または NULLIF 句、DEFAULTIF 句では PIECED フィールドを使用できない。
- 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ならない。
- PIECED に対応するデータベースの列を索引に含むことはできない。
- 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ない。

  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 は、最初のレコードから PIECED フィールドの第一分割をロードし、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をロードします。その後、同じバッファを使用して同様に 3 番目のレコードをロードします。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ので、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。

READBUFFERS キーワードの使用

ロードするデータが個別のセクションに分割されていない場合や、最終列でない場合は、READBUFFERS を指定します。READBUFFERS を指定すると、一度に論理レコード全体を保持するのに十分な大きさのバッファ転送域を割り当てることができます。

READBUFFERS では、ダイレクト・パス・ロードで使用するバッファ数を指定します（LONG 型の場合は複数バッファにまたがることができます）。デフォルトのバッファ数は 4 です。読込みバッファの数が小さすぎると、次のようなエラーが発生します。

```
ORA-02374 ... バッファ・キュー読込み用のスロットがありません。
```

注意 : ORA-2374 にも示されているように、必要な時以外は READBUFFERS に値を指定できません。READBUFFERS の値を必要以上に大きくしても、パフォーマンスは向上しません。不必要に大きな値を指定すると、かえってシステムのオーバーヘッドが増加する結果になります。

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領域を制御できます。

時間を最小化するには、次の方法があります。

- 記憶領域を事前に割り当てる。
- データを事前にソートする。
- データ・セーブの回数を減らす。
- REDO ログ・ファイルのアーカイブを使用禁止にする。

領域を最小化するには、次の方法があります。

- ロード前にデータをソートするときに、最も多くの一時記憶領域を必要とする索引でデータをソートする。
- ロード中の索引メンテナンスを避ける。

高速ロードのための記憶域の事前割当て

SQL*Loader は、必要に応じて自動的に表にエクステントを追加しますが、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トをあらかじめ割り当ててください。

表に必要な領域を計算するには、『Oracle8*i* 管理者ガイド』の「データベース・ファイルの管理」の章を参照してください。必要な領域を割り当てるには、SQL コマンド CREATE TABLE で INITIAL または MINEXTENTS 句を使用してください。

別の方法として、エクステントの割当て回数が減るようにエクステントの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、ロード中に必要な一時記憶領域を最小にできます。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。必要な記憶領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。必要な記憶領域の見積りについては、8-11 ページ「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。ソート・ルーチンはまったく使用されず、索引作成のマージ段階も省略されます。単に、新しいキーが索引に挿入されます。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、一時記憶領域は不要であり、時間も短縮できます。

SORTED INDEXES 文

SORTED INDEXES 文では事前ソートするデータの索引を指定します。この文は、ダイレクト・パス・ロードでのみ使用できます。この文の構文については、第 5 章の「SQL*Loader 制御ファイル・リファレンス」を参照してください。また、具体例については、4-25 ページ「事例 6: ダイレクト・パス・ロード方式を使用したロード」を参照してください。

一般に、SORTED INDEXES 文では 1 つの索引のみ指定します。これは、ある索引でソートされたデータは、通常、別の索引にとって正しい順序ではないからです。しかし、複数の索引に関してデータの順序がまったく同じである場合は、索引すべてを同時に指定できます。

SORTED INDEXES 文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 文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 文で複数列の索引を指定する場合は、まず索引の最初の列に関して順番に並び、次に 2 番目の列について順番に並ぶように、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に、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・ロードの最高の性能を引き出すには、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 つのテキスト列からなる場合、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領域の両方を最小にできます。

最も大きな記憶領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、最大幅を持った索引を選択します。
3. 複数の表のロードの場合、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. 複数の表のロードにおいて、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、ステップ 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。各索引にロードされる行数とその索引の幅を掛けて、最も大きな値となった索引を選択します。

データ・セーブの回数を減らす

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ので、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、なるべく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のがよいでしょう。その値から、ROWS に設定すべき値が求められます。

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 ログの使用を最小限に抑える

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。

アーカイブを使用禁止にする

メディア回復が使用禁止の場合、ダイレクト・パス・ロードは全イメージの REDO ログを生成しません。

UNRECOVERABLE の指定

UNRECOVERABLE を使用すると、時間および REDO ログ・ファイルの領域を節約できます。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）を生成します。UNRECOVERABLE は、ロード・セッション中にロードされたオブジェクトすべて（データ・セグメントと索引セグメントの両方）に適用されるので注意してください。

このため、ロードされた表についてはメディア回復はできません。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ので、必要な場合はロード後にデータのバックアップをとってください。

UNRECOVERABLE 句を指定してロードしたデータについてメディア回復が必要になった場合、ロードしたデータ・ブロックには、論理的に破壊されたというマークが付けられます。

データを回復するには、データを削除して再作成します。データが回復不能にならないように、データのロード後ただちにバックアップをとってください。

デフォルトでは、ダイレクト・パス・ロードは RECOVERABLE です。RECOVERABLE と UNRECOVERABLE の詳細は、5-3 ページ「SQL*Loader のデータ定義言語（DDL）構文図」を参照してください。

NOLOG 属性

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。

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。

次のいずれかの方法を使用して、索引のメンテナンスを回避できます。

- ロードを始める前に索引を削除する。
- 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 オプションを使用する。
- SKIP_INDEX_MAINTENANCE オプションを使用する。（ダイレクト・パスの場合に限られるので、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ことにより、ダイレクト・ロード使用中の一時記憶域を節約できます。索引のメンテナンスを回避するとロード中に必要な領域を最小限にできます。その理由は次のとおりです。

- 一度に索引を作成できるので、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る。
- 索引を作成するとき、索引セグメントは 1 つしか存在しない（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する）。

表の全行数に対してロードする行数が多い場合は、索引のメンテナンスを避けることは非常に合理的です。しかし、比較的少数の行を大きな表に追加する場合には、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、従来型パスを使用するか、SINGLEROW オプションを使用してください。

ダイレクト・ロード、整合性制約およびトリガー

従来型パスでは、行の挿入には標準 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。しかし、ダイレクト・パスでデータをロードする場合は、一部の整合性制約とすべてのデータベース・トリガーが使用禁止になります。この項では、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、『Oracle8*i* アプリケーション開発者ガイド 基礎編』の「データ整合性の保持」の章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
- UNIQUE 制約
- PRIMARY KEY 制約（NOT NULL 列における UNIQUE 制約）

NOT NULL 制約は列配列の構築時にチェックされます。この制約に違反する行はすべて拒否されます。UNIQUE 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。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。8-11 ページ「索引使用禁止状態（Index Unusable）のままの索引」を参照してください。

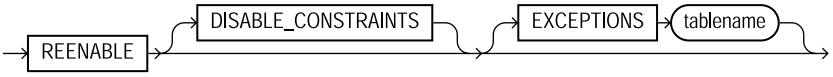
使用禁止の制約

次の制約は使用禁止になります。

- チェック制約
- 参照制約（外部キー）

制約を使用可能に戻す

REENABLE 句を指定しておくと、ロードが完了した時点で、整合性制約が自動的に使用可能に戻されます。この句の構文は次のとおりです。



キーワード DISABLED_CONSTRAINTS はオプションで、読みやすくするために使用します。EXCEPTIONS 句を使う場合は、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なければなりません。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行の ROWID が格納されます。また、違反があった制約名も入ります。この例外表の作成方法の詳細は、『Oracle8*i* SQL リファレンス』を参照してください。

REENABLE 句を使用しない場合は、制約を手動で使用可能に戻す必要があります。制約が使用可能になると、表内のすべての行が検証されます。ここで新しいデータにエラーが見つかると、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。ENABLE の詳細は、『Oracle8*i* 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。

> **注意 :** 表内に不良データが存在していると、整合性制約を使用可能に戻すことはできません。
>
> **提案 :** 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。行のロードと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、ロードの際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ひと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ることはありません。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。

挿入トリガーを整合性制約に置き換える

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ので、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とき

挿入トリガーを 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ルックアップ関数を使っ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、自動制約は現在の行における定数および列しか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

どちらの方法の場合も、表に対して事前に行うべき作業があります。その作業について、下記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 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からです。
3. ロード時に SQL*Loader の CONSTANT 句を使用して、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。

以上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ので、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であれば、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" のすべての箇所を "*old*.column_name" に変更してください。

2. 現在の更新トリガーがあれば、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れば、それを復元します。

   注意 : 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権限が必要となる場合もあります。この権限があれば、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。

例外処理を実現する

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。挿入トリガーにおける例外の呼出し箇所に相当するところで、更新トリガーは追加した列にフラグを設定します。これにより、その行が無効であることを示します。

要約 : 挿入トリガーで例外を呼び出している場合は、次の条件を満たすことができれば、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）。
- 表の排他的更新が可能である（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、そのインプリメンテーションはより複雑になります。このプロシージャは、挿入トリガーが例外を呼び出すときに使用できます。そのときに、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ので、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. 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。次に示す一般的な指針に従ってください（インプリメンテーションの詳細。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）。
   - 表から新しい行をすべて選択するように、カーソルを宣言する。
   - 処理ループの中でカーソルをオープンし、1 回に 1 行ずつフェッチする。
   - 挿入トリガーにおける操作を実行する。

- 操作が正常に終了した場合は、「新データ」のフラグを NULL に変更する。
- 操作が失敗した場合は、「新データ」のフラグを「不良データ」に変更する。

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. プロシージャを実行した後、表の中に「不良データ」フラグの付いた行がないかどうか調べます。
4. 不良の行を更新するか、削除します。
5. 挿入トリガーを使用可能に戻します。

永続的に使用禁止のトリガーおよび制約

SQL*Loader 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 が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 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 は、できるかぎり問題のないように処理しようとします。ロード終了前に、SQL*Loader は使用禁止のトリガーおよび制約を使用可能に戻そうと試みます。しかし、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に、SQL*Loader がトリガーや制約を使用可能にする処理も実行できてい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操作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はありますが、発生する可能性はあります。このような状況を回避する最善の方法は、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できるアプリケーションを実行しないことです。

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・ファイルを注意深く調べてください。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 が使用可能に戻せなかったトリガーや制約は、ENABLE 句を使用して、手動で使用可能にする必要があります。ENABLE 句の詳細は、『Oracle8*i* SQL リファレンス』を参照してください。

代替方法 : 従来型パスによる同時ロード

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したい場合は、従来型パスによる同時ロードの使用を考えてください。すなわち、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、次のような特徴があります。

- 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るが、ダイレクト・ロードほど速くはない。
- 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る。

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・ロード方法を使用したセグメント間同時処理
- ダイレクト・パス・ロード方法を使用したセグメント内同時処理

注意 : パラレル・ロードは Enterprise Edition でのみ実行できます。

同時従来型パス・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については、前の項で説明しています。この方法は、同一の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使用できます。

ダイレクト・パスを使用した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・ロードを行うときは、次の事項について考慮してください。

- ローカル索引は、ロードによってメンテナンスされる。
- グローバル索引は、ロードではメンテナンスできない。
- 参照整合性およびチェック制約は使用禁止にしなければならない。
- トリガーは使用禁止にしなければならない。
- 入力データはあらかじめパーティション化しなければならない。（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、パフォーマンスを悪化させます。）

ダイレクト・パスを使用した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。

データ・ロードのこの方法は、DIRECT および PARALLEL オプションをともに TRUE に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。

並列処理はユーザーにより管理されるものだということを理解してください。PARALLEL オプションを TRUE に設定した場合、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

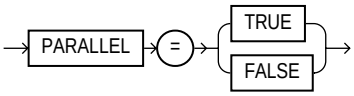
パラレル・ダイレクト・パス・ロードには次の制限があります。

- 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ない。
- 参照整合性およびチェック制約は使用禁止にしなければならない。
- トリガーは使用禁止にしなければならない。
- 行は追加（APPEND）のみできる。REPLACE および TRUNCATE、INSERT は使用できない。（これは、個別のロードにより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、データを事前にパーティション化しておいてください（そうしないと、パーティション不一致のためのレコード拒否のオーバーヘッドにより、ロード速度が遅くなりなす）。

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・ロードするセッションすべてに対して、PARALLEL 句を TRUE と設定する必要があります。構文は次のとおりです。



PARALLEL は、コマンド行またはパラメータ・ファイルに指定できます。また、OPTION 句を使っ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。

```
SQLLOAD USERID=SCOTT/TIGER CONTROL=LOAD1.CTL DIRECT=TRUE PARALLEL=TRUE
SQLLOAD USERID=SCOTT/TIGER CONTROL=LOAD2.CTL DIRECT=TRUE PARALLEL=TRUE
SQLLOAD USERID=SCOTT/TIGER CONTROL=LOAD3.CTL DIRECT=TRUE PARALLEL=TRUE
```

上記のコマンドは、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（次の制御ファイルの例を参照してください）。

> **注意 :** パラレル・ロード時には、索引は作成されません。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、セグメントの上限の上の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せることができます。

パラレル・ダイレクト・パス・ロードの Option キーワード

パラレル・ダイレクト・パス・ロードの使用中に、オプションを使用して 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できます。

一時セグメントの指定

同時実行のダイレクト・パス・ロード・セッションでは、最大の入出力スループットを得るために、それぞれの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 キーワードを使用して、ファイル名を指定できます。前述の例で SQL*Loader セッションに使用した制御ファイルの一部を次に示します。

```
LOAD DATA
INFILE 'load1.dat'
INSERT INTO TABLE emp
OPTIONS(FILE='/dat/data1.dat')
(empno POSITION(01:04) INTEGER EXTERNAL NULLIF empno=BLANKS
...
```

一時セグメントの割当て元となるデータベース・ファイルは、各オブジェクト（表またはパーティション）について OPTIONS 句の FILE キーワードを使用して制御ファイル内で指定します。同時実行する各 SQL*Loader セッションのコマンド行でも、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 キーワードの使用 Oracle の FILE キーワードは、パラレル・ダイレクト・パス・ロードでは次の制限があります。

1. **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しなければならない。
2. **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しなければならない。
3. **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しなければならない。つまり、すべてのパーティションは同じ表領域に存在しなければならない。

STORAGE キーワードの使用 STORAGE キーワードは、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領域属性を指定する場合に使用されます。STORAGE キーワードが使用されない場合、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領域属性が使用されます。

```
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 キーワードが使用できるのは制御ファイル内のみで、コマンド行では使用できません。PCTINCREASE を 0 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 キーワード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・ロードの後に制約を使用可能にする

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。

一般的なパフォーマンス改善のヒント

この項では、ロードのパフォーマンス改善に役立ついくつかのガイドラインを示します。データのロードに、ある機能の使用が必須の場合は、必ずそれを使用してください。しかし、ロードするデータの形式について制御が可能な場合は、ここ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。

1. 論理レコードの処理の効率化。
   - 物理レコードと論理レコードの 1 対 1 マップを使用する（continueif、concatenate を使用しない）。
   - ソフトウェアが物理レコードの境界を容易に判断できるようにする。ファイル処理オプション文字列 "FIX nnn" または "VAR" を使用する。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 は各物理レコードを走査してレコード終了記号（改行文字）を探さなければなりません。
2. フィールド設定の効率化。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。固定位置のフィールドを使用する。デリミタ付きフィールドを使用すると、SQL*Loader は入力データを走査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、空白を切り捨ててはならない。
3. 変換の効率化。キャラクタ・セット変換やデータ型変換など SQL*Loader はいくつかの変換を行います。こうした変換がなければ、処理は最も速くなります。
   - キャラクタ・セット変換はできるだけ避ける。SQL*Loader は 4 つのキャラクタ・セットをサポートしています。a) クライアント・キャラクタ・セット（クライアントの sqlldr プロセスの NLS_LANG）、b) 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・セットと同じですが、異なる場合もあります）、c) サーバー・キャラクタ・セット、d) サーバー各国文字キャラクタ・セット、の 4 つです。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・セットとサーバー・キャラクタ・セットが同じである場合、最もよい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。
   - 可能であればシングルバイト・キャラクタ・セットを使用する。
4. ダイレクト・パス・ロードを使用する。
5. 「SORTED INDEXES」句を使用する。

6. NULLIF 句および DEFAULTIF 句は必要な場合以外は避ける。これらの句は、ロードされるすべての行についてそれに関連する句を持つ各列を評価する必要があります。
7. 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可能な場合使用する。

第III部

オフライン・データベース
検査ユーティリティ

9

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、DBVERIFY の使用方法について説明します。この章では、次のトピックについて説明します。

- DBVERIFY
  - 構文
  - DBVERIFY のサンプル出力

DBVERIFY

DBVERIFY は外部コマンド・ユーティリティであり、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 を使用するのは、バックアップ・データベース（またはデータ・ファイル）を復元する前にそれが有効であることを確認する場合です。または、データ破壊の問題が発生した場合に診断援助機能として使用します。

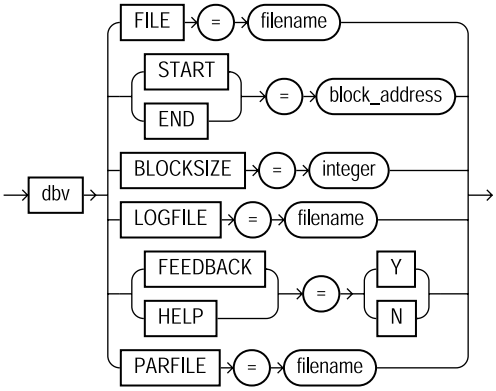
DBVERIFY はオフライン・データベースに対して実行できるので、整合性チェックが非常に高速に行えます。

> **追加情報 :** DBVERIFY の名前と場所は使用しているオペレーティング・システムに依存します（たとえば、Sun/Sequent システムでは **dbv**）。ご使用のシステムでの DBVERIFY の場所については、ご使用のオペレーティング・システム固有の Oracle ドキュメントを参照してください。

制限事項

DBVERIFY によるチェックはキャッシュ管理ブロックのみに制限されています。

構文



パラメータ

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。
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。2KB でないファイルに BLOCKSIZE を指定しないと、エラー DBV-00103 が返ります。
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	キーワード FEEDBACK を指定すると、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 がシングル・ドット "." 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	画面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、特定のタイプの整合性チェック、または異なるタイプのデータ・ファイル、あるいはその両方のためにパラメータ・ファイルをカスタマイズできます。

DBVERIFY のサンプル出力

次の例で、オンライン・ヘルプの表示方法を示します。

```
% dbv help=y

DBVERIFY: Release 8.1.5 - Wed Aug  2 09:14:36 1995

Copyright (c) Oracle Corporation 1979, 1994.  All rights reserved.

Keyword   Description        (Default)
----------------------------------------------
FILE      File to Verify     (NONE)
START     Start Block        (First Block of File)
END       End Block          (Last Block of File)
BLOCKSIZE Logical Block Size (2048)
LOGFILE   Output Log         (NONE)
```

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 が指定されているので、100 ページの処理が行われるたびに画面に点が 1 つ表示されます。

```
% dbv file=t_db1.f feedback=100

DBVERIFY: Release 8.1.5 - Wed Aug  2 09:15:04 1995

Copyright (c) Oracle Corporation 1979, 1994.  All rights reserved.

DBVERIFY - Verification starting : FILE = t_db1.f

.......................................................................


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
```

重要

- Pages= ブロック
- Total Pages Examined = ファイルのブロック数
- Total Pages Processed = 検証されたブロック（書式化されたブロック）

A

SQL*Loader の予約

Oracle ユーティリティで予約されている語のリストを示します。予約語を表および列の名前に使用すると問題が生じるため、通常は名前に予約語は使用しません。予約語の名前を付けるときは、下記の使用方法の指示に従って、問題が生じないようにしてください。

予約語リストおよび情報

通常は、表および列に名前を付けるとき、インストレーションで使用する可能性のある言語やユーティリティの予約語は使用しないようにします。各種の言語マニュアルおよびリファレンス・マニュアル、またはこの付録の予約語リストを参照してください。

SQL 予約語リストの詳細は、『Oracle8*i* SQL リファレンス』を参照してください。表および列の名前に SQL の予約語を使用する場合は、予約語の部分を二重引用符で囲んで指定します。

SQL*Loader で表および列を命名する際は、通常のルールに従ってください。表または列の名前を「予約語」、つまり SQL*Loader に関して特別な意味を持つ語に指定することはできません。次の語は、表または列の名前に使用する場合、二重引用符で囲むようにしてください。

AND	APPEND	BADDN
BADFILE	BEGINDATA	BFILE
BLANKS	BLOCKSIZE	BY
BYTEINT	CHAR 型	CHARACTERSET
COLUMN	CONCATENATE	CONSTANT
CONTINUE_LOAD	CONTINUEIF	COUNT
DATA	DATE 型	DECIMAL
DEFAULTIF	DELETE	DISABLED_CONSTRAINTS
DISCARDDN	DISCARDFILE	DISCARDMAX
DISCARDS	DOUBLE	ENCLOSED
EOF	EXCEPTIONS	EXTERNAL
FIELDS	FILLER	FIXED
FLOAT	FORMAT	GENERATED
GRAPHIC	INDDN	INDEXES
INFILE	INSERT	INTEGER
INTO	LAST	LOAD
LOBFILE	LOG	LONG
MAX	MLSLABEL	NESTED
NEXT	NO	NULLCOLS
NULLIF	OBJECT	OID

OPTIONALLY	OPTIONS	PART
PARTITION	PIECED	POSITION
PRESERVE	RAW	READBUFFERS
READSIZE	RECLEN	RECNUM
RECORD	RECOVERABLE	REENABLE
REF	REPLACE	RESUME
SDF	SEQUENCE	SID
SINGLEROW	SKIP	SMALLINT
SORTDEVT	SORTED	SORTNUM
SQL/DS	STORAGE	STREAM
SUBPARTITION	SYSDATE	TABLE
TERMINATED	THIS	TRAILING
TRUNCATE	UNLOAD	UNRECOVERABLE
USING	VARCHAR	VARCHARC
VARGRAPHIC	VARIABLE	VARRAW
VARRAWC	VARRAY	WHEN
WHITESPACE	WORKDDN	YES
ZONED		

B

DB2/DXT ユーザーに対する注意事項

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。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
- SQL*Loader の全構文 (DB2 と互換性を持つ部分も表示)

DB2 RESUME オプションの使用方法

ロード先の表に既存のデータがある場合の処理については、DB2 の RESUME 構文を指定することもできますが、RESUME 機能に相当する SQL*Loader キーワードを指定する方が便利です。次の表にまとめた SQL*Loader オプションの詳細は、5-32 ページ「空および空でない表へのデータのロード」を参照してください。

表 B-1 DB2 の関数とそれに相当する SQL*Loader の操作

DB2	SQL*Loader オプション	結果
RESUME NO または RESUME 句なし	INSERT	表が空のときだけデータがロードされる。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る。
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る。

次に、DB2 の構文について説明します。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 つのオプションの中から選択します。RESUME 句を使用して選択した処理を指示します。RESUME の引数は、大カッコで囲むことができます。

```
RESUME { YES | NO [ REPLACE ] }
```

- 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。または、個々の INTO TABLE 句の後に RESUME 句を指定すれば、RESUME オプションをそれぞれの表単位に対して指定できます。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も前に指定された RESUME オプションよりも優先されます。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み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 はこれらのオプションを解析しますが、Oracle では意味を成さない場合はそのオプションを無視します。次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 はこの文の解析は行いますが、無視します。（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 ではログ・ファイルをエラー回復のために使用しますが、必ずログが取られるわけではありません。

SQL*Loader は、Oracle の自動ログ機能を利用するので、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 はこの文の解析は行いますが、無視します。DB2 では、この文はソート用のテンポラリ・ファイルの指定に使用されます。

```
[ WORKDDN filename ]
```

SORTDEVT 文と SORTNUM 文

DB2 との互換性を保つために用意されています。SQL*Loader はこれらの文の解析は行いますが、無視します。DB2 では、これらの文はソート用の一時データ・セットの番号と型を指定する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（DISCARDDN 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 つだけロードする場合は、これらの句は元の位置 (RESUME 句と RECLEN 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 オプションがゼロ (0) 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 は、SQL / DS ファイルまたは DB2 UNLOAD ファイルからのデータ・ロードは行いません。SQL*Loader は、次に示す DB2 ロード・ユーティリティ・コマンドを検出すると、エラーを通知します。

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 ローダーは、DB2 UNLOAD 形式および SQL / DS 形式、DB2 ロード・ユーティリティ形式のファイルをロードします。SQL*Loader はこれらの形式をサポートしていません。このオプションがコマンド・ファイル内に指定されていると、SQL*Loader は処理を停止して、エラーを通知します (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ので、SQL*Loader はファイルを読み込むことができません)。

```
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 が SQL / DS 内部形式をサポートしていないためです。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 は処理を停止して、エラーを通知します。

DBCS GRAPHIC 型文字列

Oracle 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ので、GRAPHIC データ型の文字列 (G'**' の形式) は指定できません。

SQL*Loader の全構文 (DB2 と互換性を持つ部分も表示)

次のリストの中で、DB2 と互換性のある文は太字で表示します。

```
OPTIONS (options)
{ LOAD | CONTINUE_LOAD } [DATA]
[ CHARACTERSET character_set_name ]
[ { INFILE | INDDN } { filename | * }
[ "OS-dependent file processing options string" ]
[ { BADFILE | BADDN } filename ]
[ { DISCARDFILE | DISCARDDN } filename ]
[ { DISCARDS | DISCARDMAX } n ] ]
[ { INFILE | INDDN } ] ...
```

```
[ APPEND | REPLACE | INSERT |
RESUME [(] { YES | NO [REPLACE] } [)]  ]
[ LOG { YES | NO } ]
[ WORKDDN filename ]
[ SORTDEVT 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
{ [ RECNU
| 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]
```

索引

数字

A

B

C

D

E

F

G

H

I

L

N

O

P

R

S

T

U

V

W

Z

あ

い

え

お

か

き

く

け

こ

さ

し

す

せ

そ

た

ち

て

と

な

ね

は

ひ

ふ

へ

ま

み

む

め

も

ゆ

よ

ら

り

る

れ

ろ