Oracle9*i* Database

グローバリゼーション・サポート・ガイド

リリース 2（9.2）

2002 年 7 月

部品番号：J06278-01

ORACLE®

Oracle9*i* Database グローバリゼーション・サポート・ガイド, リリース 2（9.2）

部品番号：J06278-01

原本名：Oracle9*i* Database Globalization Support Guide, Release 2 (9.2)

原本部品番号：A96529-01

原本著者：Cathy Baird

原本協力者：Dan Chiba, Winson Chu, Jessica Fan, Claire Ho, Simon Law, Geoff Lee, Peter Linsley, Keni Matsuda, Tamzin Oscroft, Shige Takeda, Linus Tanaka, Makoto Tozawa, Barry Trute, Mayumi Tsujimoto, Ying Wu, Michael Yau, Tim Yu, Chao Wang, Simon Wong, Weiran Zhang, Lei Zheng, Yan Zhu, Valarie Moore

目次

2 キャラクタ・セットの選択

3 グローバリゼーション・サポート環境の設定

4 言語ソート

5 Unicode を使用した多言語データベースのサポート

6 Unicode を使用したプログラミング

7 グローバル環境での SQL と PL/SQL のプログラミング

8 グローバル環境での OCI プログラミング

9 グローバル環境での Java プログラミング

10 キャラクタ・セットの移行

11 Character Set Scanner

12 ロケール・データのカスタマイズ

A ロケール・データ

B Unicode 文字コードの割当て

用語集

索引

はじめに

このマニュアルでは、Oracle のグローバリゼーション・サポートとその機能の使用方法について説明します。

ここでは、次の項目について説明します。

対象読者

このマニュアルは、次のタスクを実行するデータベース管理者、システム管理者およびデータベース・アプリケーション開発者を対象にしています。

- グローバリゼーション・サポート環境の設定
- キャラクタ・セットの選択、分析または移行
- 言語に応じたデータのソート
- ロケール・データのカスタマイズ
- グローバル環境でのプログラムの作成
- Unicode の使用

このマニュアルを活用するには、リレーショナル・データベースの概念、Oracle サーバーの基本概念および Oracle を稼働しているオペレーティング・システム環境についての理解が必要です。

このマニュアルの構成

このマニュアルの構成は、次のとおりです。

第 1 章「グローバリゼーション・サポートの概要」
この章では、グローバリゼーションの概要と Oracle のグローバリゼーションに対するアプローチについて説明します。

第 2 章「キャラクタ・セットの選択」
この章では、キャラクタ・セットの選択方法について説明します。

第 3 章「グローバリゼーション・サポート環境の設定」
この章では、グローバリゼーション機能を有効に使用する例を記述します。

第 4 章「言語ソート」
この章では、言語ソートについて説明します。

第 5 章「Unicode を使用した多言語データベースのサポート」
この章では、データベースに関する Unicode の考慮事項について説明します。

第 6 章「Unicode を使用したプログラミング」
この章では、Unicode 環境でのプログラミング方法について説明します。

第 7 章「グローバル環境での SQL と PL/SQL のプログラミング」
この章では、SQL プログラミングに関するグローバリゼーションの考慮事項について説明します。

第 8 章「グローバル環境での OCI プログラミング」
この章では、OCI プログラミングに関するグローバリゼーションの考慮事項について説明します。

第 9 章「グローバル環境での Java プログラミング」
この章では、Java に関するグローバリゼーションの考慮事項について説明します。

第 10 章「キャラクタ・セットの移行」
この章では、キャラクタ・セット変換の問題とキャラクタ・セットの移行について説明します。

第 11 章「Character Set Scanner」
この章では、文字データの分析に必要な Character Set Scanner ユーティリティの使用方法について説明します。

第 12 章「ロケール・データのカスタマイズ」
この章では、ロケールのカスタマイズに必要な Oracle Locale Builder ユーティリティの使用方法について説明します。また、タイム・ゾーン・ファイルとカレンダ・データのカスタマイズについても説明します。

付録 A「ロケール・データ」
この付録では、Oracle サーバーでサポートされている言語、地域、キャラクタ・セットおよびその他のロケール・データについて説明します。

付録 B「Unicode 文字コードの割当て」
この付録では、Unicode のコード割当てを記述します。

用語集
用語集では、グローバリゼーション・サポートに関する用語の定義を記述します。

関連文書

詳細は、次の Oracle のマニュアルを参照してください。

- 『Oracle9*i* SQL リファレンス』
- 『Oracle9*i* アプリケーション開発者ガイド - 基礎編』

このマニュアルに記載されている例の多くは、Oracle のインストール時にデフォルトでインストールされるシード・データベースのサンプル・スキーマを使用しています。これらのスキーマがどのように作成されているか、およびその使用方法については、『Oracle9*i* サンプル・スキーマ』を参照してください。

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

```
http://otn.oracle.co.jp/membership/
```

OTN-J のユーザー名とパスワードを取得済みであれば、次の OTN-J Web サイトの文書セクションに直接接続できます。

```
http://otn.oracle.co.jp/document/
```

表記規則

このマニュアル・セットの本文とコード例に使用されている表記規則について説明します。

- 本文の表記規則
- コード例の表記規則
- Windows オペレーティング・システムの表記規則

本文の表記規則

本文中には、特別な用語が一目でわかるように様々な表記規則が使用されています。次の表は、本文の表記規則と使用例を示しています。

表記規則	意味	例
太字	太字は、本文中に定義されている用語または用語集に含まれている用語、あるいはその両方を示します。	この句を指定する場合は、**索引構成表**を作成します。
固定幅フォントの大文字	固定幅フォントの大文字は、システムにより指定される要素を示します。この要素には、パラメータ、権限、データ型、Recovery Manager キーワード、SQL キーワード、SQL*Plus またはユーティリティ・コマンド、パッケージとメソッドの他、システム指定の列名、データベース・オブジェクトと構造体、ユーザー名、およびロールがあります。	この句は `NUMBER` 列に対してのみ指定できます。 `BACKUP` コマンドを使用すると、データベースのバックアップを作成できます。 `USER_TABLES` データ・ディクショナリ・ビューの `TABLE_NAME` 列を問い合せます。 `DBMS_STATS.GENERATE_STATS` プロシージャを使用します。
固定幅フォントの小文字	固定幅フォントの小文字は、実行可能ファイル、ファイル名、ディレクトリ名およびとサンプルのユーザー指定要素を示します。この要素には、コンピュータ名とデータベース名、ネット・サービス名、接続識別子の他、ユーザー指定のデータベース・オブジェクトと構造体、列名、パッケージとクラス、ユーザー名とロール、プログラム・ユニット、およびパラメータ値があります。 **注意：**一部のプログラム要素には、大文字と小文字の両方が使用されます。この場合は記載されているとおりに入力してください。	`sqlplus` と入力して SQL*Plus をオープンします。 パスワードは `orapwd` ファイルに指定されています。 `/disk1/oracle/dbs` ディレクトリ内で、データ・ファイルと制御ファイルのバックアップを作成します。 `department_id`、`department_name` および `location_id` の各列は、`hr.departments` 表にあります。 初期化パラメータ `QUERY_REWRITE_ENABLED` を `true` に設定します。 `oe` ユーザーで接続します。 これらのメソッドは、`JRepUtil` クラスで実装されます。

表記規則	意味	例
固定幅フォントの小文字のイタリック	固定幅フォントの小文字のイタリックは、プレースホルダまたは変数を示します。	*parallel_clause* を指定できます。 U*old_release*.SQL を実行します。*old_release* はアップグレード前にインストールしていたリリースです。

コード例の表記規則

コード例は、SQL、PL/SQL、SQL*Plus またはその他のコマンドラインを示します。次のように、固定幅フォントで、通常の本文とは区別して記載されています。

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

次の表は、コード例の記載上の表記規則と使用例を示しています。

表記規則	意味	例
[ ]	大カッコで囲まれている項目は、1 つ以上のオプション項目を示します。大カッコ自体は入力しないでください。	DECIMAL (*digits* [, *precision*])
{ }	中カッコで囲まれている項目は、そのうちの 1 つのみが必要であることを示します。中カッコ自体は入力しないでください。	{ENABLE | DISABLE}
|	縦線は、大カッコまたは中カッコ内の複数の選択肢を区切るために使用します。オプションのうち 1 つを入力します。縦線自体は入力しないでください。	{ENABLE | DISABLE} [COMPRESS | NOCOMPRESS]
...	水平の省略記号は、次のいずれかを示します。	
	■ 例に直接関係のないコード部分が省略されていること。	CREATE TABLE ... AS *subquery*;
	■ コードの一部が繰返し可能なこと。	SELECT *col1*, *col2*, ... , *coln* FROM employees;
. . .	垂直の省略記号は、例に直接関係のない数行のコードが省略されていることを示します。	SQL> SELECT NAME FROM V$DATAFILE; NAME ---------- /fs1/dbs/tbs_01.dbf /fs1/dbs/tbs_02.dbf . . . /fs1/dbs/tbs_09.dbf 9 rows selected.

表記規則	意味	例
その他の表記	大カッコ、中カッコ、縦線および省略記号以外の記号は、表示されているとおりに入力してください。	`acctbal NUMBER(11,2);` `acct    CONSTANT NUMBER(4) := 3;`
イタリック	イタリックの文字は、特定の値を指定する必要のあるプレースホルダまたは変数を示します。	`CONNECT SYSTEM/`*`system_password`* `DB_NAME = `*`database_name`*

大文字	大文字は、システムにより指定される要素を示します。これらの用語は、ユーザー定義の用語と区別するために大文字で記載されています。大カッコで囲まれている場合を除き、記載されているとおりの順序とスペルで入力してください。ただし、この種の用語は大 / 小文字区別がないため、小文字でも入力できます。	`SELECT last_name, employee_id FROM employees;`
		`SELECT * FROM USER_TABLES;`
		`DROP TABLE hr.employees;`

小文字	小文字は、ユーザー指定のプログラム要素を示します。たとえば、表名、列名またはファイル名を示します。
	
	注意 : 一部のプログラム要素には、大文字と小文字の両方が使用されます。この場合は記載されているとおりに入力してください。
		`SELECT last_name, employee_id FROM employees;`
		`sqlplus hr/hr`
		`CREATE USER mjones IDENTIFIED BY ty3MU9;`

Windows オペレーティング・システムの表記規則

次の表は、Windows オペレーティング・システムの表記規則と使用例を示しています。

表記規則	意味	例
「スタート」→を選択	プログラムの起動方法。	Database Configuration Assistant を起動するには、「スタート」→「プログラム」→「Oracle - *HOME_NAME*」→「Configuration and Migration Tools」→「Database Configuration Assistant」を選択します。
ファイル名とディレクトリ名	ファイル名とディレクトリ名には、大 / 小文字区別はありません。特殊文字のうち、左山カッコ（<）、右山カッコ（>）、コロン（:）、二重引用符（"）、スラッシュ（/）、パイプ（|）およびハイフン（-）は使用できません。特殊文字の円記号（¥）は、引用符で囲まれていても要素のセパレータとして扱われます。¥¥ で始まるファイル名は、Windows では汎用命名規則を使用しているものとみなされます。	`c:¥winnt"¥"system32` は `C:¥WINNT¥SYSTEM32` と同じです。
`C:¥>`	現行のハード・ディスク・ドライブの Windows コマンド・プロンプトを表します。コマンド・プロンプトでのエスケープ文字はカレット（^）です。プロンプトには、作業中のサブディレクトリが反映されます。このマニュアルでは、コマンド・プロンプトと呼んでいます。	`C:¥oracle¥oradata>`
特殊文字	Windows コマンド・プロンプトでは、特殊文字の二重引用符（"）のエスケープ文字として、特殊文字の円記号（¥）が必要な場合があります。カッコと一重引用符（'）には、エスケープ文字は不要です。エスケープ文字と特殊文字の詳細は、Windows オペレーティング・システムのマニュアルを参照してください。	`C:¥>exp scott/tiger TABLES=emp QUERY=¥"WHERE job='SALESMAN' and sal<1600¥"` `C:¥>imp SYSTEM/`*`password`* `FROMUSER=scott TABLES=(emp, dept)`
HOME_NAME	Oracle ホーム名を表します。ホーム名は英数字で 16 文字以内です。ホーム名に使用できる特殊文字は、アンダースコアのみです。	`C:¥> net start Oracle`*`HOME_NAME`*`TNSListener`

表記規則	意味	例
ORACLE_HOME と *ORACLE_BASE*	Oracle8 リリース 8.0 以前では、Oracle コンポーネントをインストールすると、すべてのサブディレクトリはデフォルトで次のいずれかの名前を使用して、トップレベルの *ORACLE_HOME* ディレクトリの下に置かれていました。 ■ Windows NT の場合は C:¥orant ■ Windows 98 の場合は C:¥orawin98 このリリースは、Optimal Flexible Architecture（OFA）のガイドラインに従っています。すべてのサブディレクトリがトップレベルの *ORACLE_HOME* ディレクトリの下にあるとはかぎりません。*ORACLE_BASE* というトップレベルのディレクトリがあり、これはデフォルトでは C:¥oracle です。他の Oracle ソフトウェアがインストールされていないコンピュータに最新の Oracle リリースをインストールすると、最初の Oracle ホーム・ディレクトリのデフォルト設定は C:¥oracle¥ora*nn* となります。この場合、*nn* は最新リリース番号です。Oracle ホーム・ディレクトリは、*ORACLE_BASE* の直下にあります。 このマニュアルに記載されているディレクトリ・パスの例は、すべて OFA の表記規則に準拠しています。	%*ORACLE_HOME*%¥rdbms¥admin ディレクトリにアクセスします。

グローバリゼーション・サポートの新機能

この項では、Oracle9*i* リリース 2（9.2）でのグローバリゼーション・サポートの新機能と、追加情報の参照先について説明します。

この項の構成は、次のとおりです。

Oracle9*i* リリース 2（9.2）でのグローバリゼーション・サポートの新機能

- **Unicode 3.1 のサポート**

  Oracle9*i* リリース 2（9.2）は、Unicode 3.1 をサポートしています。

  関連項目： 第 5 章「Unicode を使用した多言語データベースのサポート」

- **ALTER TABLE MODIFY 文**

  ALTER TABLE MODIFY 文を使用すると、列定義を CHAR データ型から NCHAR データ型に変更できます。また、列のデータも CHAR データ型から NCHAR データ型に変換されます。

  関連項目： 10-13 ページ「ALTER TABLE MODIFY 文を使用した CHAR 列から NCHAR 列への変更」

- **Oracle Locale Builder の拡張**

  Oracle Locale Builder でコード・チャートを表示および印刷できます。

  発音区別記号付きの文字を照合ツリーに表示できるようになりました。

  関連項目：

  - 12-19 ページ「Oracle Locale Builder を使用したコード・チャートの表示」
  - 12-35 ページ「Oracle Locale Builder を使用した新規言語ソートの作成」

- **Character Set Scanner の拡張**

  Character Set Scanner に、2 つのパラメータ EXCLUDE および PRESERVE が追加されました。

  TABLE パラメータが、複数の表をサポートするように拡張されました。

  変換可能で例外的なデータ・ディクショナリ・データは、個別例外レポートのうち、新しい「データ・ディクショナリ個別例外」に示されます。

  関連項目：

  - 11-9 ページ「Character Set Scanner パラメータ」
  - 11-27 ページ「個別例外レポート」

- **ユーロ・サポートの変更**

  欧州通貨連合（European Monetary Union: EMU）の加盟国は、2002 年 1 月 1 日から自国通貨としてユーロを使用しています。NLS_TERRITORY を対応する EMU 加盟国（オーストリア、ベルギー、フィンランド、フランス、ドイツ、ギリシャ、アイルランド、イタリア、ルクセンブルグ、オランダ、ポルトガルおよびスペイン）に設定すると、NLS_CURRENCY および NLS_DUAL_CURRENCY のデフォルト値が EURO に設定されます。Oracle9*i* リリース 2（9.2）からは、NLS_ISO_CURRENCY の値に従って EMU 加盟国用の ISO 通貨記号が EUR に設定されます。

  関連項目： 3-36 ページ「Oracle のユーロ・サポート」

- **OCIEnvNlsCreate() 関数**

  OCI 環境の作成時に OCIEnvNlsCreate 関数を使用して、クライアント側データベースと各国語キャラクタ・セットを指定します。この関数を使用すると、NLS_LANG および NLS_CHAR 初期化パラメータの設定に関係なく、アプリケーションでキャラクタ・セット情報を動的に設定できます。また、1 つのアプリケーションで、同じサーバー環境内で異なるクライアント環境に使用する複数の環境ハンドラを初期化できます。

  関連項目： 8-3 ページ「OCIEnvNlsCreate()」

- **OCINlsCharSetConvert() 関数**

  この関数は、文字列をあるキャラクタ・セットから別のキャラクタ・セットに変換します。

  関連項目： 8-52 ページ「OCINlsCharSetConvert()」

- **OCINlsCharSetNameToId() 関数**

  この関数は、指定された Oracle キャラクタ・セット名の Oracle キャラクタ・セット ID を戻します。

  関連項目： 8-10 ページ「OCINlsCharSetNameToId()」

- **OCINlsCharSetIdToName() 関数**

  この関数は、指定されたキャラクタ・セット ID の Oracle キャラクタ・セット名を戻します。

  関連項目： 8-11 ページ「OCINlsCharSetIdToName()」

- **OCINlsNumericInfoGet() 関数**

  この関数は、item で指定された数値言語情報を、出力の数値変数への OCI 環境ハンドルから生成します。

  関連項目： 8-12 ページ「OCINlsNumericInfoGet()」

- **OCINlsNameMap() 関数**

  この関数は、Oracle キャラクタ・セット名、言語名および地域名と、IANA 名および ISO 名の間のマッピングを行います。

  関連項目： 8-14 ページ「OCINlsNameMap()」

- **DBMS_LOB.LOADBLOBFROM FILE および DBMS_LOB.LOADCLOBFROM FILE**

  この 2 つの API を使用すると、新規のパラメータを使用して BFILE データのキャラクタ・セット ID を指定できます。各 API によって、指定した BFILE キャラクタ・セットから、CLOB の場合はデータベース・キャラクタ・セット、NCLOB の場合は各国語キャラクタ・セットへとデータが変換されます。

  関連項目： 7-14 ページ「LOB への外部 BFILE データのロード」

- **汎用ベース文字検索**

  ケースと発音区別記号を無視する検索を実行できます。

  関連項目： 4-16 ページ「汎用ベース文字検索の実行」

- **NCHAR データ型とキャラクタ・セマンティクスに対するオブジェクト型サポートの変更**

  オブジェクト型では、NCHAR データ型とキャラクタ・セマンティクスがサポートされるようになりました。

  関連項目：

  - 2-2 ページ「長さセマンティクス」
  - 2-18 ページ「サポート対象のデータ型の概要」
  - 『Oracle9*i* アプリケーション開発者ガイド - オブジェクト・リレーショナル機能 』

1

グローバリゼーション・サポートの概要

この章では、Oracle グローバリゼーション・サポートの概要を説明します。この章の内容は、次のとおりです。

- グローバリゼーション・サポートのアーキテクチャ
- グローバリゼーション・サポートの機能

グローバリゼーション・サポートのアーキテクチャ

Oracle のグローバリゼーション・サポートによって、データの格納、処理および取出しをネイティブ言語で実行できます。このアーキテクチャによって、データベース・ユーティリティ、エラー・メッセージ、ソート順序、日付、時刻、通貨単位、数値およびカレンダに関する規則が、各国の言語やロケールに自動的に適応されます。

Oracle のグローバリゼーション・サポート機能は、以前は National Language Support（NLS）機能と呼ばれていました。National Language Support はグローバリゼーション・サポートのサブセットであり、各国語を選択してデータを特定のキャラクタ・セットで格納する機能です。グローバリゼーション・サポートを使用すると、世界中から同時にアクセスして実行できる多言語アプリケーションやソフトウェア製品の開発が可能になります。アプリケーションでは、ユーザー・インタフェースの内容表示とデータ処理に、ユーザーの母国語と選択したロケールを使用できます。

オンデマンドのロケール・データ

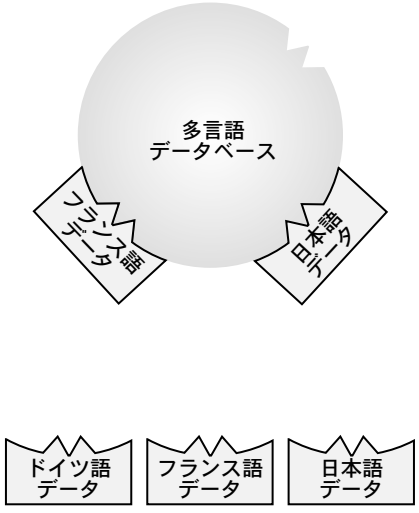
Oracle のグローバリゼーション・サポートは、Oracle NLS Runtime Library（NLSRTL）とともに実装されます。NLS ランタイム・ライブラリは、広範囲にわたる言語非依存の関数のパッケージを提供します。これによって、適切なテキストと文字の処理および言語規則に従った操作を行うことができます。特定の言語や地域に対応した関数の動作は、実行時に指定してロードされたロケール固有データ・セットによって制御されます。

ロケール固有データは、Oracle でサポートされる各ロケールごとに独立したデータ・セットとしての構造を持ちます。特定のロケールのデータを、他のロケール・データに依存せずにロードできます。この設計の利点は、次のとおりです。

- 必要なロケール・セットを選択して、Oracle9*i* のメモリー消費を管理できます。
- 他のロケールに影響を与えずに、特定のロケールのロケール・データを追加してカスタマイズできます。

図 1-1 に、実行時のロケール固有データのロードを示します。この例では、フランス語データと日本語データが多言語データベースにロードされますが、ドイツ語データはロードされません。

図 1-1 データベースへのロケール固有データのロード



ロケール固有データは、環境変数 ORA_NLS* で指定されたディレクトリに格納されています。Oracle データベース・サーバーのリリースごとに、異なる ORA_NLS データ・ディレクトリがあります。Oracle9*i* の場合は、ORA_NLS33 ディレクトリが使用されます。表 1-1 に、Oracle データベース・サーバーの様々なリリースについて、ロケール固有データの位置を指定する環境変数を示します。

表 1-1 ロケール固有データの位置を指定するリリース別の環境変数

リリース	環境変数
7.2	ORA_NLS
7.3	ORA_NLS32
8.0、8.1、9.0.1、9.2	ORA_NLS33

環境変数 ORA_NLS* が定義されていない場合、ロケール固有データには Oracle ホーム・ディレクトリに対して相対のデフォルト値が使用されます。ロケール・データのデフォルト位置は、どのリリースでも $ORACLE_HOME/ocommon/nls/admin/data です。ほとんどの場合は、このデフォルト値で十分です。ORA_NLS* 変数を定義する必要があるのは、システムの複数の Oracle ホームで NLS データ・ファイルの単一コピーが共有される場合のみです。

ロードする有効な NLS オブジェクトは、ブート・ファイルを使用して指定します。Oracle では、システムとユーザーの両方のブート・ファイルをサポートしています。ユーザーのブート・ファイルを使用すると、そのデータベースで使用可能な NLS ロケール・オブジェクトを柔軟に調整できます。また、新しいロケール・データを追加したり、一部のロケール・データ・コンポーネントをカスタマイズすることができます。

> **関連項目：** ロケール・データのカスタマイズの詳細は、第 12 章「ロケール・データのカスタマイズ」を参照してください。

多言語アプリケーションをサポートするアーキテクチャ

Oracle9*i* データベースは、データベースの構成に使用されている言語を、複数層アプリケーションとクライアント / サーバー・アプリケーションでサポートできるように実装されます。

ロケール依存操作は、クライアントとデータベース・サーバーの両方にある複数のパラメータと環境変数によって制御されます。データベース・サーバーでは、クライアント用に起動された各セッションが他のセッションと同じまたは異なるロケールで実行され、同じまたは異なる言語要件が指定されることがあります。

データベースには、セッションに依存しない NLS パラメータのセットが作成時に指定されます。その 2 つのパラメータによって、データベース・キャラクタ・セットと各国語キャラクタ・セット、つまり NCHAR、NVARCHAR2 および NCLOB データに指定できる代替 Unicode キャラクタ・セットを指定します。このパラメータは、テキスト・データをデータベースに格納するために使用するキャラクタ・セットを指定します。他のパラメータ（言語や地域など）は、CHECK 制約の評価で使用されます。

クライアント・セッションとデータベース・サーバーが異なるキャラクタ・セットを指定している場合、Oracle9*i* データベースでは、キャラクタ・セットの文字列を自動的に変換します。

グローバリゼーション・サポートの観点では、すべてのアプリケーションは、Oracle インスタンスと物理的に同じマシン上で実行している場合でも、クライアントとみなされます。たとえば、Oracle ソフトウェア所有者である UNIX ユーザーが、SQL*Plus を RDBMS ソフトウェアがインストールされている Oracle ホームから起動し、ORACLE_SID パラメータを指定してアダプタ経由でデータベースに接続している場合、この SQL*Plus はクライアントとみなされ、その動作はクライアント側の NLS パラメータによって規定されます。

別の例をあげると、中間層がアプリケーション・サーバーの場合に、アプリケーションがクライアントとみなされます。そのアプリケーション・サーバーによって起動される様々なセッションは、個別のクライアント・セッションとみなされます。

クライアント・アプリケーションが起動されると、環境設定に従ってクライアントの NLS 環境が初期化されます。ローカルで実行される NLS 操作は、すべてこの設定を使用して実行されます。ローカル NLS 操作の例を次に示します。

- Oracle Developer アプリケーションの表示書式設定
- OCI 環境ハンドルを使用して NLS OCI 関数を実行するユーザー OCI コード

アプリケーションをデータベースに接続すると、サーバー上にセッションが作成されます。新しいセッションは、初期化パラメータ・ファイルに指定された NLS インスタンス・パラメータに従って、NLS 環境を初期化します。この設定は、ALTER SESSION 文により、後で変更できます。この文によって変更されるのは、そのセッションの NLS 環境のみです。ローカル・クライアントの NLS 環境は変更されません。セッションの NLS 設定は、サーバー上で実行される SQL 文と PL/SQL 文の処理に使用されます。たとえば、ALTER SESSION 文を使用して NLS_LANGUAGE 初期化パラメータを Italian に設定します。

```
ALTER SESSION SET NLS_LANGUAGE=Italian;
```

SELECT 文を入力します。

```
SQL> SELECT last_name, hire_date, ROUND(salary/8,2) salary FROM employees;
```

結果は次のようになります。

```
LAST_NAME                 HIRE_DATE     SALARY
------------------------- --------- ----------
Sciarra                   30-SET-97      962.5
Urman                     07-MAR-98        975
Popp                      07-DIC-99      862.5
```

月名の略称にイタリア語が使用されていることに注意してください。

NLS_LANG 環境設定がクライアント側で定義されている場合は、接続直後に ALTER SESSION 文が暗黙的にクライアントとセッションの NLS 環境を同期させます。

関連項目：

- 第 8 章「グローバル環境での OCI プログラミング」
- 第 3 章「グローバリゼーション・サポート環境の設定」

多言語データベースでの Unicode の使用

Unicode とは、エンコードされたユニバーサル・キャラクタ・セットのことです。このセットを使用すると、1 つのキャラクタ・セットを使用して任意の言語の情報を格納できます。Unicode には、プラットフォーム、プログラムまたは言語に関係なく、すべての文字に対する一意のコード値が用意されています。

Oracle9*i* データベースで Unicode を使用する利点は、次のとおりです。

- キャラクタ・セット変換機能と言語ソート機能が簡素化されます。
- ネイティブのマルチバイト・キャラクタ・セットに比べてパフォーマンスが改善されます。
- Unicode 規格に基づく Unicode データ型がサポートされます。

関連項目：

- 第 5 章「Unicode を使用した多言語データベースのサポート」
- 第 6 章「Unicode を使用したプログラミング」
- 5-9 ページ「Unicode データ型を使用した多言語サポートの有効化」

グローバリゼーション・サポートの機能

Oracle の標準機能には、次の内容が含まれています。

- 言語サポート
- 地域サポート
- 日付と時刻の書式
- 通貨と数値の書式
- カレンダ機能
- 言語ソート
- キャラクタ・セット・サポート
- キャラクタ・セマンティクス
- ロケール・データとカレンダ・データのカスタマイズ
- Unicode のサポート

言語サポート

Oracle9*i* データベースでは、データの格納、処理および取出しをネイティブ言語で行うことができます。Oracle9*i* データベースに格納できる言語は、Oracle がサポートしているキャラクタ・セットでエンコードされているスクリプトで記述されたすべての言語です。Unicode データベースとデータ型を使用することで、Oracle9*i* は、ほとんどの現代言語をサポートしています。

各国語のサブセットに対しては、追加サポートが用意されています。たとえば、Oracle9*i* データベースでは、翻訳された月の名前を使用して日付を表示したり、文化的な慣習に従ってテキスト・データをソートすることができます。

このマニュアルで使用している**言語サポート**という用語は、言語依存の追加機能（日付の表示やテキストのソートなど）を指しており、その言語のテキストを格納できることを示すわけではありません。

一部のサポート対象言語については、翻訳されたエラー・メッセージやデータベース・ユーティリティに関する翻訳済みユーザー・インタフェースが用意されています。

関連項目：

- 第 3 章「グローバリゼーション・サポート環境の設定」
- Oracle がサポートするすべての言語名とその略称リストは、A-2 ページの「言語」を参照してください。
- Oracle メッセージが翻訳されている言語のリストは、A-4 ページの「翻訳済みメッセージ」を参照してください。

地域サポート

Oracle9*i* データベースでは、地域に固有な文化的慣習をサポートしています。デフォルトのローカル時刻書式、日付書式、数値および通貨単位に関する規則は、ローカル地域設定によって異なります。多様な NLS パラメータの設定によって、データベース・セッションでは様々な文化的設定を使用できます。たとえば、地域が `AMERICA` と定義されている場合でも、指定したデータベース・セッションに、主要通貨として英ポンド（`GBP`）を、第 2 通貨として日本円（`JPY`）を設定できます。

関連項目：

- 第 3 章「グローバリゼーション・サポート環境の設定」
- Oracle サーバーでサポートされている地域のリストは、A-5 ページの「地域」を参照してください。

日付と時刻の書式

時間、日、月および年に関する多様な表記規則は、ローカル書式で処理されます。たとえば、日付の表示の場合、英国では DD-MON-YYYY 書式が使用され、日本では一般的に YYYY-MM-DD 書式が使用されます。

タイム・ゾーンや夏時間もサポートされます。

関連項目：

- 第 3 章「グローバリゼーション・サポート環境の設定」
- 『Oracle9*i* SQL リファレンス』

通貨と数値の書式

通貨、貸方および借方の記号は、ローカル書式で表すことができます。基数記号と 3 桁区切りは、ロケールで定義できます。たとえば、小数点は、米国ではドット（.）ですが、フランスではカンマ（,）です。したがって、金額 $1,234 の持つ意味は、国によって異なります。

関連項目： 第 3 章「グローバリゼーション・サポート環境の設定」

カレンダ機能

世界中で様々な暦法が使用されています。Oracle では、グレゴリオ暦、日本の元号暦、台湾暦、タイ仏教暦、ペルシャ暦、英国版イスラム暦およびイスラム暦の 7 つの暦法をサポートしています。

関連項目：

- 第 3 章「グローバリゼーション・サポート環境の設定」
- サポートされているカレンダのリストは、A-27 ページの「暦法」を参照してください。

言語ソート

Oracle9*i* には、文化に応じた正確なソートおよび大 / 小文字の変換を行うために、言語の定義が用意されています。基本定義では、文字列を独立した文字の連続として扱います。拡張定義では、特殊な事例として扱う必要のある文字のペアを認識します。

基本定義を使用して大文字または小文字に変換された文字列は、常に同じ長さです。拡張定義を使用して変換された文字列は、長くなったり短くなったりすることがあります。

関連項目： 第 4 章「言語ソート」

キャラクタ・セット・サポート

Oracle では、多数のシングルバイト、マルチバイトおよび固定幅のコード体系をサポートしています。これらのコード体系は、各国の規格、国際規格およびベンダー固有の規格に基づいています。

関連項目：

- 第 2 章「キャラクタ・セットの選択」
- サポートされるキャラクタ・セットのリストは、A-7 ページの「キャラクタ・セット」を参照してください。

キャラクタ・セマンティクス

Oracle9*i* では、キャラクタ・セマンティクスが導入されています。キャラクタ・セマンティクスは、可変幅のマルチバイト文字列の記憶要件をバイト数ではなく文字数で定義する場合に役立ちます。

関連項目： 2-11 ページ「長さセマンティクス」

ロケール・データとカレンダ・データのカスタマイズ

Oracle Locale Builder を使用して、言語、キャラクタ・セット、地域または言語ソートなどのロケール・データをカスタマイズできます。

NLS カレンダ・ユーティリティを使用すると、カレンダをカスタマイズできます。

関連項目：

- 第 12 章「ロケール・データのカスタマイズ」
- 12-17 ページ「NLS カレンダ・ユーティリティを使用したカレンダのカスタマイズ」

Unicode のサポート

次の 2 つの方法で Unicode 文字を Oracle9*i* データベースに格納できます。

- UTF-8 エンコードされた文字を SQL CHAR データ型として格納できるように、Unicode データベースを作成できます。
- Unicode データ型を使用して、特定の列の多言語データをサポートできます。データベース・キャラクタ・セットの定義方法に関係なく、SQL NCHAR データ型の列に Unicode 文字を格納できます。Oracle9*i* では、NCHAR データ型は、Unicode データ型専用として再定義されています。

関連項目： 第 5 章「Unicode を使用した多言語データベースのサポート」

2

キャラクタ・セットの選択

この章では、キャラクタ・セットの選択方法について説明します。この章の内容は、次のとおりです。

キャラクタ・セット・エンコーディング

文字を処理する場合、コンピュータ・システムは文字をグラフィカルな表現としてではなく数値コードとして処理します。たとえば、データベースに文字 A を格納すると、実際はソフトウェアによって解析される数値コードが格納されます。特に、異なるキャラクタ・セット間のデータ変換を必要とする可能性があるグローバル環境では、この数値コードが重要になります。

この項の内容は、次のとおりです。

エンコードされたキャラクタ・セットについて

データベースの作成時に、エンコードされたキャラクタ・セットを指定します。キャラクタ・セットの選択によって、データベース内で表現できる言語が決定します。また、キャラクタ・セットの選択は次の方法にも影響します。

- データベース・スキーマの作成方法
- 文字データを処理するアプリケーションの開発方法
- オペレーティング・システムでのデータベースの動作
- パフォーマンス

文字の集まり（アルファベット文字、表意文字、記号、句読点および制御文字など）は、キャラクタ・セットとしてエンコードできます。**エンコードされたキャラクタ・セット**では、文字レパートリ内のそれぞれの文字に一意の数値コードが割り当てられています。数値コードは**コード・ポイント**または**エンコーディング値**と呼ばれます。表 2-1 に、ASCII キャラクタ・セットの数値コード値が割り当てられている文字の例を示します。

表 2-1 ASCII キャラクタ・セットでエンコードされた文字

文字	説明	コード値
!	感嘆符	21
#	数値記号	23
$	ドル記号	24
1	数字の 1	31

表 2-1 ASCII キャラクタ・セットでエンコードされた文字（続き）

文字	説明	コード値
2	数字の 2	32
3	数字の 3	33
A	大文字の A	41
B	大文字の B	42
C	大文字の C	43
a	小文字の a	61
b	小文字の b	62
c	小文字の c	63

コンピュータ業界では、様々にエンコードされたキャラクタ・セットが使用されています。各キャラクタ・セットは、次の点で異なります。

- 使用可能な文字数
- 使用可能な文字（**文字レパートリ**）
- 書込み用スクリプトとそれによって表される言語
- それぞれの文字に割り当てられたコード値
- 文字を表現するために使用するコード体系

Oracle では、各国の規格、国際規格およびベンダー固有のエンコードされたキャラクタ・セット規格のほとんどすべてがサポートされています。

> **関連項目：** Oracle でサポートされているすべてのキャラクタ・セットのリストは、付録 A「ロケール・データ」を参照してください。

エンコードされる文字

キャラクタ・セットでエンコードされる文字は、表現する記述法によって異なります。記述法は、1 つの言語または 1 つの言語グループを表現するために使用され、次の 2 つに分類できます。

- 表音的記述法
- 表意的記述法

この項の内容は、次のとおりです。

- 句読点、制御文字、数字および記号
- 記述方向

表音的記述法

表音的記述法は、1 つの言語に対応付けられた様々な音声を表現する、複数の記号で構成されています。たとえば、ギリシア語、ラテン語、キリル文字およびデーバナーガリ語はすべて、アルファベットに基づいた表音的記述法です。アルファベットは、複数の言語を表現できます。たとえば、ラテン・アルファベットでは、多くの西ヨーロッパ諸国の言語（フランス語、ドイツ語、英語など）を表現できます。

表音的記述法に対応付けられた文字は、文字レパートリが通常 256 文字より少ないため、一般的には 1 バイトでエンコードできます。

表意的記述法

表意的記述法は、言語の音声ではなく、単語の意味を表す表意文字または象形文字で構成されています。中国語と日本語は、何万という表意文字に基づいた表意的記述法の例です。表意的記述法を使用する言語では、**表音文字セット**を使用する場合もあります。表音文字セットでは、追加の表音的情報を伝達する方法が提供されます。たとえば、日本語には 2 つの表音文字セットがあり、通常、文法的な要素にはひらがなが、外来語や擬音語にはカタカナが使用されます。

表意的記述法に対応付けられた文字は、文字のレパートリが何万とあるため、一般的には複数バイトでエンコードする必要があります。

句読点、制御文字、数字および記号

言語スクリプトのエンコーディングに加えて、次のように他の特殊文字もエンコードする必要があります。

- カンマ、ピリオドおよびアポストロフィなどの句読点
- 数字
- 通貨記号や数学演算子などの特殊な記号
- キャリッジ・リターンやタブなどの制御文字

記述方向

西洋言語のほとんどは、ページの上から下へ向かって左から右へ記述されます。東アジア地域言語は、通常、ページの右から左に向かって上から下へ記述されます。ただし、西洋言語を翻訳した専門書には例外が多く見られます。アラビア語とヘブライ語は、上から下へ向かって右から左へ記述されます。

アラビア語とヘブライ語では、数字の記述方向が逆になります。テキストが右から左に書かれている場合でも、文中の数字は左から右に向かって書かれます。たとえば、「I wrote 32 books」は「skoob 32 etorw I」と書かれます。書込み方向に関係なく、データは論理順序で格納されます。論理順序は、入力している言語の画面上での表示方法ではなく、その言語の入力で使用されている順序を意味します。

記述方向は、文字のエンコーディングには影響しません。

キャラクタ・セットでサポートされている文字

様々なキャラクタ・セットによって、様々な文字レパートリがサポートされています。一般的に、キャラクタ・セットは特定の書込みスクリプトに基づいているため、複数の言語をサポートできます。キャラクタ・セットが最初に米国で開発されたとき、文字レパートリには制限がありました。今でも、特定の文字を複数のプラットフォームで使用すると、問題が発生する場合があります。次の CHAR 文字および VARCHAR 文字は、すべての Oracle データベース・キャラクタ・セットで表示可能で、どのプラットフォームにもトランスポートできます。

- 大文字と小文字の英文字（A ～ Z および a ～ z）
- アラビア数字（0 ～ 9）
- 句読点マーク（% ' ' ( ) * + - , . / \ : ; < > = ! _ & ~ { } | ^ ? $ # @ " [ ]）
- 制御文字（スペース、水平タブ、垂直タブ、改ページ）

これ以外の文字を使用する場合は、選択したデータベース・キャラクタ・セットでデータがサポートされているかどうか注意する必要があります。

適切なデータ変換を行うには、NLS_LANG 初期化パラメータを適切に設定する必要があります。NLS_LANG 初期化パラメータでは、クライアント・オペレーティング・システムの設定を反映するキャラクタ・セットを指定してください。NLS_LANG を適切に設定すると、クライアント・オペレーティング・システムのコード・ページからデータベース・キャラクタ・セットへと適切に変換できます。これらの設定が同じときは、送受信されるデータはデータベース・キャラクタ・セットと同一のキャラクタ・セットでエンコードされているとみなされ、妥当性チェックや変換は実行されません。このため、変換が必要な場合は、データが破損する可能性があります。

キャラクタ・セット間での変換中に、Oracle はデータが NLS_LANG 初期化パラメータで指定されたキャラクタ・セットでエンコーディングされるものと想定します。文字列に（たとえば、CHR または CONVERT SQL 関数を使用して）他の値を入力すると、その値は、データベースに送信されたときに、適切に変換されないため損なわれる可能性があります。環境を

適切に構成しており、データベース・キャラクタ・セットでデータベースに入力される可能性のある文字データのレパートリ全体がサポートされていれば、現行のデータベース・キャラクタ・セットを変更する必要はありません。ただし、企業がよりグローバルになり、追加の文字や新規の言語のサポートが必要になった場合は、より大きな文字レパートリを持つキャラクタ・セットの選択が必要になる可能性があります。このような場合は、Unicode データベースとデータ型の使用をお薦めします。

関連項目：

- 第 5 章「Unicode を使用した多言語データベースのサポート」
- CHR および CONVERT SQL 関数の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。
- 12-19 ページ「Oracle Locale Builder を使用したコード・チャートの表示」

ASCII エンコーディング

ASCII と EBCDIC のキャラクタ・セットでは、同じ文字レパートリがサポートされていますが、文字によっては異なるコード値が割り当てられている場合があります。表 2-2 に、ASCII がどのようにエンコードされているかを示します。行および列のヘッダーは、16 進数字を示しています。文字のコード値を調べるには、列の数字の次に行の数字を読みます。たとえば、文字 A の値は 0x41 です。

表 2-2 7 ビット ASCII キャラクタ・セット

-	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	'	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	TAB	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|

表 2-2 7 ビット ASCII キャラクタ・セット（続き）

-	0	1	2	3	4	5	6	7
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

世界中のユーザーの要求を満たすためにキャラクタ・セットは進化を続け、英語以外の言語をサポートするために新規キャラクタ・セットが作成されました。一般的に、この新規キャラクタ・セットでは、同じスクリプトに基づいた関連のある言語グループがサポートされています。 たとえば、ISO 8859 キャラクタ・セット・シリーズは、多様なヨーロッパ諸国の言語をサポートするために作成されました。表 2-3 に、ISO 8859 キャラクタ・セットでサポートされている言語を示します。

表 2-3 ISO 8859 キャラクタ・セット

規格	サポートしている言語
ISO 8859-1	西ヨーロッパ諸国の言語（アルバニア語、バスク語、ブルトン語、カタロニア語、デンマーク語、オランダ語、英語、フェロー語、フィンランド語、フランス語、ドイツ語、グリーンランド語、アイスランド語、アイルランドのゲール語、イタリア語、ラテン語、ルクセンブルク語、ノルウェー語、ポルトガル語、レートロマンス語、スコットランドのゲール語、スペイン語、スウェーデン語）
ISO 8859-2	東ヨーロッパ諸国の言語（アルバニア語、クロアチア語、チェコ語、英語、ドイツ語、ハンガリー語、ラテン語、ポーランド語、ルーマニア語、スロバキア語、スロベニア語、セルビア語）
ISO 8859-3	南東ヨーロッパ諸国の言語（アフリカーンス語、カタロニア語、オランダ語、英語、エスペラント語、ドイツ語、イタリア語、マルタ語、スペイン語、トルコ語）
ISO 8859-4	北ヨーロッパ諸国の言語（デンマーク語、英語、エストニア語、フィンランド語、ドイツ語、グリーンランド語、ラテン語、ラトビア語、リトアニア語、ノルウェー語、サーミ語、スロベニア語、スウェーデン語）
ISO 8859-5	東ヨーロッパ諸国の言語（キリル文字ベース : ブルガリア語、ベラルーシ語、マケドニア語、ロシア語、セルビア語、ウクライナ語）
ISO 8859-6	アラビア語
ISO 8859-7	ギリシア語
ISO 8859-8	ヘブライ語
ISO 8859-9	西ヨーロッパ諸国の言語（アルバニア語、バスク語、ブルトン語、カタロニア語、コーンウォール語、デンマーク語、オランダ語、英語、フィンランド語、フランス語、フリースランド語、ガリシア語、ドイツ語、グリーンランド語、アイルランドのゲール語、イタリア語、ラテン語、ルクセンブルク語、ノルウェー語、ポルトガル語、レートロマンス語、スコットランドのゲール語、スペイン語、スウェーデン語、トルコ語）

表 2-3 ISO 8859 キャラクタ・セット（続き）

規格	サポートしている言語
ISO 8859-10	北ヨーロッパ諸国の言語（デンマーク語、英語、エストニア語、フェロー語、フィンランド語、ドイツ語、グリーンランド語、アイスランド語、アイルランドのゲール語、ラテン語、リトアニア語、ノルウェー語、サーミ語、スロベニア語、スウェーデン語）
ISO 8859-13	バルト海沿岸諸国の言語（英語、エストニア語、フィンランド語、ラテン語、ラトビア語、ノルウェー語）
ISO 8859-14	ケルト系言語（アルバニア語、バスク語、ブルトン語、カタロニア語、コーンウォール語、デンマーク語、英語、ガリシア語、ドイツ語、グリーンランド語、アイルランドのゲール語、イタリア語、ラテン語、ルクセンブルク語、マン島のゲール語、ノルウェー語、ポルトガル語、レートロマンス語、スコットランドのゲール語、スペイン語、スウェーデン語、ウェールズ語）
ISO 8859-15	西ヨーロッパ諸国の言語（アルバニア語、バスク語、ブルトン語、カタロニア語、デンマーク語、オランダ語、英語、エストニア語、フェロー語、フィンランド語、フランス語、フリースランド語、ガルシア語、ドイツ語、グリーンランド語、アイスランド語、アイルランドのゲール語、イタリア語、ラテン語、ルクセンブルク語、ノルウェー語、ポルトガル語、レートロマンス語、スコットランドのゲール語、スペイン語、スウェーデン語）

キャラクタ・セットは、多言語を制限付きでサポートしてきました。この制限とは、類似するスクリプトに基づく言語グループにかぎりサポートしてきたという意味です。最近では、ユニバーサル・キャラクタ・セットが、多言語サポートのための有効なソリューションとみなされるようになってきました。Unicode は、このようなユニバーサル・キャラクタ・セットの 1 つで、現代世界の主要なスクリプトのほとんどを包含しています。Unicode キャラクタ・セットは、94,000 種類以上の文字をサポートしています。

関連項目： 第 5 章「Unicode を使用した多言語データベースのサポート」

文字のエンコード方法

コンピュータ業界では、様々なタイプのコード体系が作成されてきました。選択するキャラクタ・セットによって、使用するコード体系の種類が異なります。コード体系には様々なパフォーマンス特性があり、データベース・スキーマやアプリケーションの開発に影響を及ぼす場合があります。選択したキャラクタ・セットは、次のコード体系のいずれかを使用します。

- シングルバイト・コード体系
- マルチバイト・コード体系

シングルバイト・コード体系

シングルバイト・コード体系は、最も効率的なコード体系です。シングルバイトの 1 文字は 1 バイトで表現されるため、文字を表現するのに必要な領域量が最も小さく、処理およびプログラミングでの使用が容易です。シングルバイト・コード体系は、次のいずれかとして分類されます。

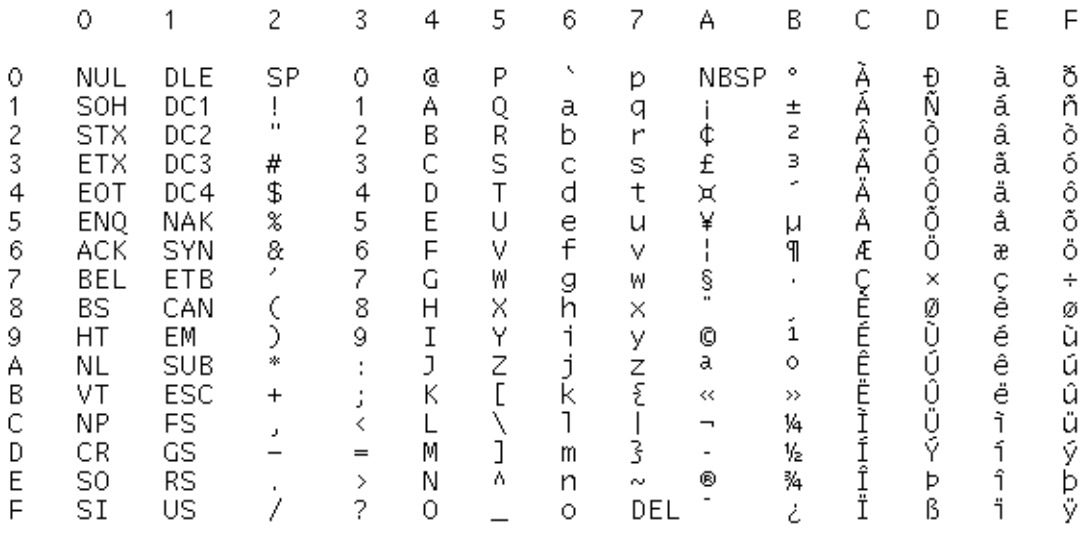
- 7 ビット・コード体系

  シングルバイト 7 ビット・コード体系は、128 文字までの文字を定義でき、通常、1 つの言語のみをサポートします。最も一般的なシングルバイト・キャラクタ・セットは、ASCII（American Standard Code for Information Interchange）で、コンピュータ処理の初期から使用されています。

- 8 ビット・コード体系

  シングルバイト 8 ビット・コード体系は、256 文字までの文字を定義でき、通常、1 つの関連言語グループをサポートします。たとえば、ISO 8859-1 は、西ヨーロッパ諸国の多数の言語をサポートしています。図 2-1 に典型的な 8 ビット・コード体系を示します。

図 2-1 8 ビット・コード体系



	0	1	2	3	4	5	6	7	A	B	C	D	E	F
0	NUL	DLE	SP	0	@	P	`	p	NBSP	°	À	Ð	à	ð
1	SOH	DC1	!	1	A	Q	a	q	¡	±	Á	Ñ	á	ñ
2	STX	DC2	"	2	B	R	b	r	¢	²	Â	Ò	â	ò
3	ETX	DC3	#	3	C	S	c	s	£	³	Ã	Ó	ã	ó
4	EOT	DC4	$	4	D	T	d	t	¤	´	Ä	Ô	ä	ô
5	ENQ	NAK	%	5	E	U	e	u	¥	µ	Å	Õ	å	õ
6	ACK	SYN	&	6	F	V	f	v	¦	¶	Æ	Ö	æ	ö
7	BEL	ETB	'	7	G	W	g	w	§	·	Ç	×	ç	÷
8	BS	CAN	(	8	H	X	h	x	¨	¸	È	Ø	è	ø
9	HT	EM	)	9	I	Y	i	y	©	¹	É	Ù	é	ù
A	NL	SUB	*	:	J	Z	j	z	ª	º	Ê	Ú	ê	ú
B	VT	ESC	+	;	K	[	k	{	«	»	Ë	Û	ë	û
C	NP	FS	,	<	L	\	l	|	¬	¼	Ì	Ü	ì	ü
D	CR	GS	-	=	M	]	m	}		½	Í	Ý	í	ý
E	SO	RS	.	>	N	^	n	~	®	¾	Î	Þ	î	þ
F	SI	US	/	?	O	_	o	DEL	¯	¿	Ï	ß	ï	ÿ

マルチバイト・コード体系

マルチバイト・コード体系は、中国語や日本語のようなアジア言語の表意文字をサポートするために必要です。これは、中国語や日本語では何千という文字が使用されるためです。このようなコード体系では、1 文字を表現するために固定または可変のバイト数を使用します。

- 固定幅マルチバイト・コード体系

  固定幅マルチバイト・コード体系では、各文字が固定のバイト数で表現されます。マルチバイト・コード体系では、バイト数は 2 以上です。

- 可変幅マルチバイト・コード体系

  可変幅コード体系は、1 バイト以上を使用して 1 つの文字を表現します。一部のマルチバイト・コード体系は、特定のビットを使用して、1 文字を表現するためのバイト数を示します。たとえば、1 文字の表現に使用する最大のバイト数が 2 バイトの場合は、最上位ビットを使用して、そのバイトがシングルバイト文字であるか、ダブルバイト文字の 1 番目のバイトであるかを示します。

- シフト・センシティブ可変幅マルチバイト・コード体系

  一部の可変幅コード体系では、制御コードを使用して、同じコード値を持つシングルバイト文字とマルチバイト文字が区別されます。シフトアウト・コードは後続の文字がマルチバイトであることを示し、シフトイン・コードは後続の文字がシングルバイトであることを示します。シフト・センシティブ・コード体系は、主に IBM プラットフォームで使用されます。ISO-2022 キャラクタ・セットは、データベース・キャラクタ・セットとしては使用できませんが、メール・サーバーなどのアプリケーションには使用できることに注意してください。

Oracle のキャラクタ・セットのネーミング規則

キャラクタ・セット名には次のネーミング規則が使用されます。

```
<language or region><number of bits representing a character><standard character set
name>[S | C]
```

注意： UTF8 および UTFE は、このネーミング規則の例外です。

オプションの S または C を使用して、サーバー（S）またはクライアント（C）でのみ使用できるキャラクタ・セットを区別します。

注意： Macintosh プラットフォームではサーバー・キャラクタ・セット（S）を使用します。Macintosh のクライアント・キャラクタ・セットは、現在使用されていません。EBCDIC プラットフォームでは、サーバー上ではサーバー・キャラクタ・セット（S）、クライアント上ではクライアント・キャラクタ・セット（C）を使用します。

次の表に、Oracle キャラクタ・セット名の例を示します。

Oracle キャラクタ・セット名	説明	地域	1 文字の表現に使用されるビット数	標準キャラクタ・セット名
US7ASCII	U.S. 7 ビット ASCII	US	7	ASCII
WE8ISO8859P1	西ヨーロッパ 8 ビット ISO 8859 Part1	WE（西ヨーロッパ）	8	ISO8859 Part 1
JA16SJIS	日本語 16 ビットシフト JIS	JA	16	SJIS

長さセマンティクス

シングルバイト・キャラクタ・セットの場合、文字列のバイト数と文字数は同じです。マルチバイト・キャラクタ・セットの場合は、1 文字または 1 つのコード単位が 1 つ以上のバイトで構成されています。可変幅キャラクタ・セットの場合は、バイト長に基づく文字数の計算が困難な場合があります。列の長さをバイト数単位で計算することを**バイト・セマンティクス**、文字数単位で計算することを**キャラクタ・セマンティクス**と呼びます。

Oracle9*i* では、キャラクタ・セマンティクスが導入されています。キャラクタ・セマンティクスは、可変幅のマルチバイト文字列の記憶要件を定義する場合に役立ちます。たとえば、Unicode データベース（AL32UTF8）で、VARCHAR2 列を英語の 5 文字とともに最大 5 文字の中国語文字を格納できるように定義する必要があるとします。バイト・セマンティクスを使用すると、この列には、長さ 3 バイトである中国語文字用に 15 バイトと、長さ 1 バイトである英語文字用に 5 バイト、合計 20 バイトが必要です。キャラクタ・セマンティクスを使用すると、この列に必要な文字数は 10 となります。

次の式では、バイト・セマンティクスが使用されています。

- `VARCHAR2(20 BYTE)`
- `SUBSTRB(`*`string`*`, 1, 20)`

VARCHAR2 式の BYTE 修飾子と SQL 関数名の B 接尾辞に注意してください。

次の式では、キャラクタ・セマンティクスが使用されています。

- `VARCHAR2(10 CHAR)`
- `SUBSTR(`*`string`*`, 1, 10)`

VARCHAR2 式の CHAR 修飾子に注意してください。

文字データ型の新しい列にバイト・セマンティクスとキャラクタ・セマンティクスのどちらが使用されるかは、NLS_LENGTH_SEMANTICS 初期化パラメータによって決定されます。このパラメータのデフォルト値は BYTE です。前述の VARCHAR2 の定義に示されている BYTE および CHAR 修飾子は、データベースにセマンティクスを混在させる結果となる可能性があるため、できるだけ使用しないでください。かわりに、初期化パラメータ・ファイル内で

NLS_LENGTH_SEMANTICS を設定し、NLS_LENGTH_SEMANTICS の値に基づいてデフォルトのセマンティクスを使用するように列のデータ型を定義します。

バイト・セマンティクスは、データベース・キャラクタ・セットのデフォルトです。文字長セマンティクスは NCHAR データ型のデフォルトであり、このデータ型に使用できる唯一の長さセマンティクスです。NCHAR の定義には、CHAR または BYTE 修飾子を指定できません。

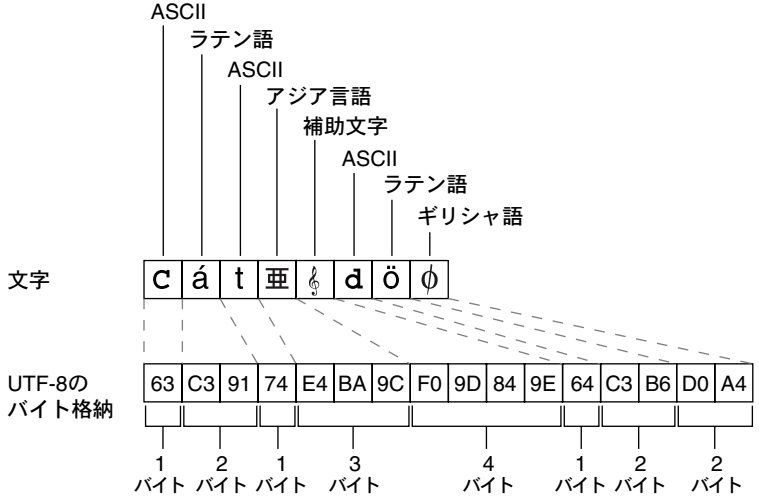
次の例を考えてみます。

```
CREATE TABLE emp
    ( empno NUMBER(4)
    , ename NVARCHAR2(10)
    , job NVARCHAR2(9)
    , mgr NUMBER(4)
    , hiredate DATE
    , sal NUMBER(7,2)
    , deptno NUMBER(2)
    ) ;
```

NCHAR のキャラクタ・セットが AL16UTF16 の場合、ename に保持できるのは、最大 10 個の Unicode コードまたは最大 20 バイトです。

図 2-2 に、UTF-8 キャラクタ・セットで各種文字を格納するために必要なバイト数を示します。ASCII 文字には 1 バイト、ラテン語文字とギリシャ語文字には 2 バイト、アジア言語の文字には 3 バイト、補助文字には 4 バイトの記憶域が必要です。

図 2-2 各種文字の格納バイト数



関連項目：

- SUBSTR および SUBSTRB 関数の詳細は、7-6 ページの「様々な長さセマンティクスに使用する SQL 関数」を参照してください。
- NLS_LENGTH_SEMANTICS 初期化パラメータの詳細は、3-41 ページの「長さセマンティクス」を参照してください。
- Unicode と NCHAR データ型の詳細は、第 5 章「Unicode を使用した多言語データベースのサポート」を参照してください。
- SUBSTRB および SUBSTR 関数と文字データ型の BYTE および CHAR 修飾子の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。

Oracle データベース・キャラクタ・セットの選択

Oracle では、次の項目でデータベース・キャラクタ・セットを使用します。

- SQL CHAR データ型（CHAR、VARCHAR2、CLOB および LONG）で格納されるデータ
- 表名、列名および PL/SQL 変数などの識別子
- SQL と PL/SQL のソース・コードの入力と格納

データベースで使用される文字コード体系は、CREATE DATABASE 文の一部として定義されます。データ・ディクショナリ内の列を含めて、すべての SQL CHAR データ型列（CHAR、CLOB、VARCHAR2 および LONG）のデータは、データベース・キャラクタ・セットに格納されます。さらに、選択するデータベース・キャラクタ・セットによって、データベース内のオブジェクトに名前を指定できる文字が決定されます。SQL NCHAR データ型列（NCHAR、NCLOB および NVARCHAR2）では、各国語キャラクタ・セットが使用されます。

注意： データベース・キャラクタ・セットがマルチバイトの場合、CLOB データは UCS-2 としてエンコードされます。データベース・キャラクタ・セットがシングルバイトの場合、CLOB データはデータベース・キャラクタ・セットで格納されます。

データベースの作成後は、一部の例外を除いて、データベースを再作成せずにキャラクタ・セットを変更することはできません。

データベース用の Oracle キャラクタ・セットを選択する場合は、次の項目について考慮する必要があります。

- データベースでサポートする必要がある言語
- データベースで将来サポートが必要になる言語
- キャラクタ・セットがオペレーティング・システムで使用可能かどうか

- クライアントで使用するキャラクタ・セット
- そのキャラクタ・セットがアプリケーションで適切に処理されるかどうか
- キャラクタ・セットがパフォーマンスに与える影響
- キャラクタ・セットに関連付けられている制限

Oracle キャラクタ・セットのリストは、付録 A「ロケール・データ」を参照してください。Oracle キャラクタ・セット名は、言語とそれが使用される地域に基づいています。地域名が付いている一部のキャラクタ・セットは、リストに言語でも明示的に示されています。

あるキャラクタ・セットに含まれている文字を確認するには、次の方法があります。

- 各国製品、国際的製品またはベンダー製品のドキュメントあるいは規格ドキュメントをチェックします。
- Oracle Locale Builder を使用します。

この項の内容は、次のとおりです。

- 現在および将来の言語要件
- クライアントのオペレーティング・システムとアプリケーションの互換性
- クライアントとサーバーの間のキャラクタ・セット変換
- データベース・キャラクタ・セットの選択がパフォーマンスに与える影響
- データベース・キャラクタ・セットに関する制限事項
- 各国語キャラクタ・セットの選択
- サポート対象のデータ型の概要

関連項目：

- 5-4 ページ「UCS-2 エンコーディング」
- 2-17 ページ「各国語キャラクタ・セットの選択」
- 2-19 ページ「データベース作成後のキャラクタ・セットの変更」
- 付録 A「ロケール・データ」
- 第 12 章「ロケール・データのカスタマイズ」

現在および将来の言語要件

複数のキャラクタ・セットを使用して現在の言語要件を満たすことができる可能性があります。データベース・キャラクタ・セットの選択時には、将来の言語要件を考慮してください。将来、追加の言語サポートが必要になると思われる場合は、後で異なるキャラクタ・セットへの移行が必要にならないように、その言語をサポートしているキャラクタ・セットを選択します。

クライアントのオペレーティング・システムとアプリケーションの互換性

Oracle には独自のグローバリゼーション・アーキテクチャがあるため、データベース・キャラクタ・セットはオペレーティング・システムに依存しません。たとえば、英語版 Windows オペレーティング・システムで、日本語キャラクタ・セットを使用してデータベースを作成し、稼働させることができます。ただし、クライアントのオペレーティング・システムがそのデータベースにアクセスするには、適切なフォントと入力メソッドでデータベース・キャラクタ・セットをサポートできる必要があります。たとえば、英語版 Windows オペレーティング・システムでは、先に日本語フォントと入力メソッドをインストールしなければ、日本語データの挿入や取出しができません。または、日本語データの挿入や取出しを行うために、日本語版オペレーティング・システムを使用してデータベース・サーバーにリモートでアクセスする方法があります。

クライアントとサーバーの間のキャラクタ・セット変換

クライアントのオペレーティング・システム上のキャラクタ・セットと異なるデータベース・キャラクタ・セットを選択した場合、Oracle データベースでは、オペレーティング・システムのキャラクタ・セットをデータベース・キャラクタ・セットに変換できます。キャラクタ・セット変換には、次のデメリットがあります。

- オーバーヘッドの増加
- データ消失の可能性

キャラクタ・セット変換がデータ消失の原因になる場合があります。たとえば、キャラクタ・セット A をキャラクタ・セット B に変換する場合、変換先のキャラクタ・セット B には、A と同じキャラクタ・セット・レパートリが含まれている必要があります。キャラクタ・セット B で使用できない文字は、置換文字に変換されます。この置換文字は、多くの場合、疑問符や言語的に関連する文字で指定されます。たとえば、ä（ウムラウト付きの a）は a に変換される場合があります。分散環境の場合には、類似した文字レパートリを持つキャラクタ・セットを使用してデータ消失を回避してください。

キャラクタ・セット変換では、データがクライアントに到達するまでに文字列が何度もバッファ間でコピーされる場合があります。データベース・キャラクタ・セットは、常に、クライアントのオペレーティング・システム固有のキャラクタ・セットと同等か、そのスーパーセットである必要があります。通常は、データベースにアクセスするクライアント・アプリケーションのキャラクタ・セットによって、最適なスーパーセットが決定します。

すべてのクライアント・アプリケーションが同じキャラクタ・セットを使用している場合は、そのキャラクタ・セットが、通常、データベース・キャラクタ・セットとして最善の選択肢です。複数のクライアント・アプリケーションが異なるキャラクタ・セットを使用している場合、データベース・キャラクタ・セットは、クライアントのすべてのキャラクタ・セットのスーパーセットである必要があります。これによって、クライアント・キャラクタ・セットからデータベース・キャラクタ・セットへの変換時に、すべての文字が確実に受け渡されます。

関連項目： 第 10 章「キャラクタ・セットの移行」

データベース・キャラクタ・セットの選択がパフォーマンスに与える影響

最適なパフォーマンスのためには、変換が不要なキャラクタ・セットを選択し、使用言語にとって最も効率的なエンコーディングを使用してください。シングルバイト・キャラクタ・セットは、パフォーマンスの点で、マルチバイト・キャラクタ・セットよりも優れており、領域要件に関しても、シングルバイト・キャラクタ・セットが最も効率的です。ただし、シングルバイト・キャラクタ・セットには、サポートできる言語の種類に制限があります。

データベース・キャラクタ・セットに関する制限事項

ASCII ベースのキャラクタ・セットがサポートされているのは、ASCII ベースのプラットフォーム上のみです。同様に、EBCDIC ベースのキャラクタ・セットが使用できるのは、EBCDIC ベースのプラットフォーム上のみです。

データベース・キャラクタ・セットは、SQL と PL/SQL ソース・コードの識別に使用します。このためには、データベース・キャラクタ・セットに、プラットフォーム固有の EBCDIC または 7 ビット ASCII のいずれかがサブセットとして含まれている必要があります。したがって、固定幅のマルチバイト・キャラクタ・セットをデータベース・キャラクタ・セットとして使用することはできません。現在、データベース・キャラクタ・セットとして使用できないのは AL16UTF16 キャラクタ・セットのみです。

名前を表すために使用するキャラクタ・セットに関する制限事項

表 2-4 に、名前を表すために使用できるキャラクタ・セットに関する制限事項を示します。

表 2-4 名前を表すために使用するキャラクタ・セットに関する制限事項

名前	シングルバイト	可変幅	コメント
列名	可能	可能	-
スキーマ・オブジェクト	可能	可能	-
コメント	可能	可能	-
データベース・リンク名	可能	不可	-

表 2-4 名前を表すために使用するキャラクタ・セットに関する制限事項（続き）

名前	シングルバイト	可変幅	コメント
データベース名	可能	不可	-
ファイル名（データ・ファイル、ログ・ファイル、制御ファイル、初期化パラメータ・ファイル）	可能	不可	-
インスタンス名	可能	不可	-
ディレクトリ名	可能	不可	-
キーワード	可能	不可	英語 ASCII または EBCDIC 文字でのみ表現可能です。
Recovery Manager ファイル名	可能	不可	-
ロールバック・セグメント名	可能	不可	ROLLBACK_SEGMENTS パラメータは、NLS をサポートしていません。
ストアド・スクリプト名	可能	可能	-
表領域名	可能	不可	-

LOB データ（LOB、BLOB、CLOB および NCLOB）などのサポート対象の文字列形式とキャラクタ・セットについては、2-18 ページの表 2-6 を参照してください。

各国語キャラクタ・セットの選択

各国語キャラクタ・セットとは、Unicode データベース・キャラクタ・セットがないデータベースに Unicode の文字データを格納するための代替キャラクタ・セットです。その他、各国語キャラクタ・セットを選択するには次の理由があります。

- 大量の文字処理操作には、別の文字コード体系のプロパティのほうが適している場合
- 各国語キャラクタ・セットでのプログラミングのほうが容易な場合

SQL NCHAR、NVARCHAR2 および NCLOB データ型は、Unicode データのみをサポートするように再定義されています。このデータは、UTF-8 エンコーディングまたは UTF-16 エンコーディングのいずれかで格納できます。

関連項目： 第 5 章「Unicode を使用した多言語データベースのサポート」

サポート対象のデータ型の概要

表 2-5 に、様々なコード体系についてサポート対象のデータ型を示します。

表 2-5 コード体系のサポート対象の SQL データ型

データ型	シングルバイト	マルチバイトの Unicode 以外	マルチバイトの Unicode
CHAR	可能	可能	可能
VARCHAR2	可能	可能	可能
NCHAR	不可	不可	可能
NVARCHAR2	不可	不可	可能
BLOB	可能	可能	可能
CLOB	可能	可能	可能
LONG	可能	可能	可能
NCLOB	不可	不可	可能

注意： BLOB では、文字は一連のバイト順序として処理されます。データは、NLS に関連した操作の対象にはなりません。

表 2-6 に、抽象データ型についてサポート対象の SQL データ型を示します。

表 2-6 SQL データ型のサポート対象の抽象データ型

抽象データ型	CHAR	NCHAR	BLOB	CLOB	NCLOB
オブジェクト	可能	可能	可能	可能	可能
コレクション	可能	可能	可能	可能	可能

次のように、NCHAR 属性を使用して抽象データ型を作成できます。

```
SQL> CREATE TYPE tp1 AS OBJECT (a NCHAR(10));
Type created.
SQL> CREATE TABLE t1 (a tp1);
Table created.
```

関連項目： オブジェクトとコレクションの詳細は、『Oracle9*i* アプリケーション開発者ガイド - オブジェクト・リレーショナル機能』を参照してください。

データベース作成後のキャラクタ・セットの変更

データベースの作成後に、データベース・キャラクタ・セットの変更が必要になる場合があります。たとえば、使用しているデータベースでサポートする必要がある言語数が増加する場合があります。ほとんどの場合は、全体エクスポートまたは全体インポートを行い、すべてのデータを新しいキャラクタ・セットに正しく変換する必要があります。ただし、新しいキャラクタ・セットが現行のキャラクタ・セットの完全なスーパーセットである場合にかぎり、ALTER DATABASE CHARACTER SET 文を使用して、データベース・キャラクタ・セット内の変更を効率よく処理できます。

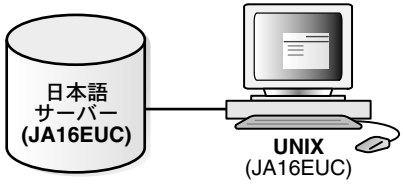
関連項目：

- 第 10 章「キャラクタ・セットの移行」
- データのエクスポートとインポートの詳細は、『Oracle9*i* データベース移行ガイド』を参照してください。
- ALTER DATABASE CHARACTER SET 文の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。

単一言語データベースの使用例

最も単純なデータベース構成の例は、クライアントとサーバーが同じ言語環境で稼働し、同じキャラクタ・セットを使用している場合です。この単一言語の使用例には、高速応答という利点があります。これは、キャラクタ・セット変換に関連したオーバーヘッドが回避されるためです。図 2-3 に、同じキャラクタ・セットを使用しているデータベース・サーバーとクライアントを示します。

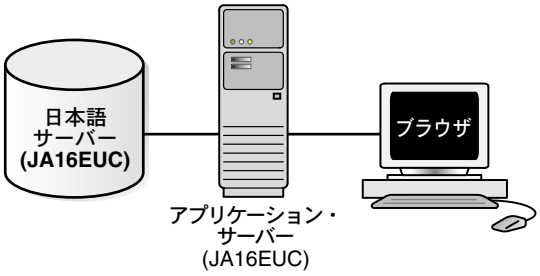
図 2-3 単一言語データベースの使用例



日本語クライアントとサーバーの両方が、JA16EUC キャラクタ・セットを使用しています。

また、複数層アーキテクチャも使用できます。図 2-4 に、データベース・サーバーとクライアントの間にあるアプリケーション・サーバーを示します。アプリケーション・サーバーとデータベース・サーバーは、単一言語の使用例と同じキャラクタ・セットを使用しています。

図 2-4 複数層の単一言語データベースの使用例



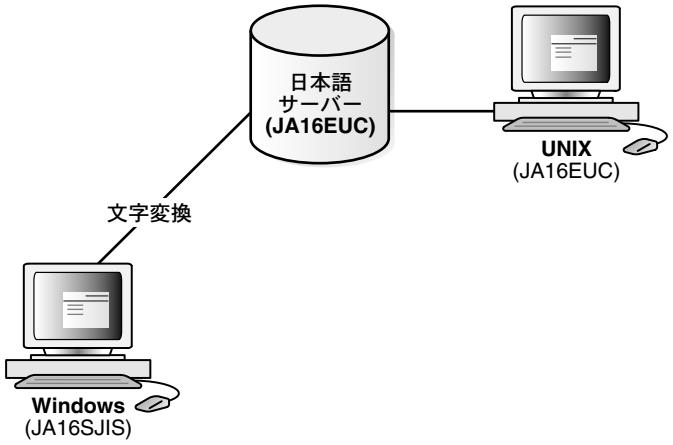
サーバー、アプリケーション・サーバーおよびクライアントは、JA16EUC キャラクタ・セットを使用しています。

単一言語の使用例でのキャラクタ・セット変換

クライアント / サーバー環境では、クライアント・アプリケーションがサーバーと異なるプラットフォームにある場合や、プラットフォームで同じ文字コード体系が使用されていない場合に、キャラクタ・セット変換が必要になります。クライアントとサーバーの間で受け渡される文字データは、2 つのコード体系間で変換される必要があります。文字の変換は、Oracle Net を介して、ユーザーが意識することなく自動的に実行されます。

任意の 2 つのキャラクタ・セット間で変換を行うことができます。図 2-5 に、サーバーと日本語 JA16EUC キャラクタ・セットを使用している 1 つのクライアントを示します。他方のクライアントは、日本語 JA16SJIS キャラクタ・セットを使用しています。

図 2-5 キャラクタ・セットの変換

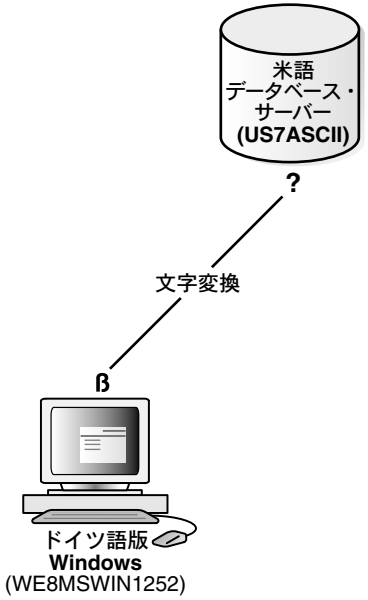


ターゲットのキャラクタ・セットにソース・データにあるすべての文字がない場合は、置換文字が使用されます。たとえば、サーバーでは US7ASCII を、ドイツ語クライアントでは WE8ISO8859P1 を使用している場合、ドイツ語の文字 ß は ? に置換され、ä は a に置換されます。

置換文字は、キャラクタ・セットの定義の一部として、特定の文字に対して定義できます。特定の置換文字を定義しないと、デフォルトの置換文字が使用されます。クライアント・キャラクタ・セットからデータベース・キャラクタ・セットへの変換時に置換文字を使用しないようにするには、サーバーのキャラクタ・セットが、クライアントのすべてのキャラクタ・セットのスーパーセットである必要があります。

図 2-6 に、データベース・キャラクタ・セットにクライアント・キャラクタ・セットのすべての文字が含まれていない場合に発生するデータ消失を示します。

図 2-6 文字変換中のデータ消失



データベース・キャラクタ・セットは US7ASCII です。クライアントのキャラクタ・セットは WE8MSWIN1252 で、クライアントの使用言語はドイツ語です。クライアントが ß を含む文字列を挿入すると、データベースでは ß が ? に置換され、データが消失します。

ドイツ語データがサーバーに格納されると想定される場合は、データ消失と変換オーバーヘッドを回避するために、サーバーとクライアントの両方で、ドイツ語文字をサポートするデータベース・キャラクタ・セットを使用する必要があります。

キャラクタ・セットの一方が可変幅マルチバイト・キャラクタ・セットの場合は、変換によって著しいオーバーヘッドが生じる可能性があります。状況を慎重に評価した上でキャラクタ・セットを選択し、変換をできるかぎり回避する必要があります。

多言語データベースの使用例

多言語サポートには、制限付きと無制限があります。この項の内容は、次のとおりです。

制限付き多言語サポート

一部のキャラクタ・セットでは、関連する記述法やスクリプトがあるため、多言語がサポートされます。たとえば、Oracle キャラクタ・セット WE8ISO8859P1 は、次の西ヨーロッパ諸国の言語をサポートしています。

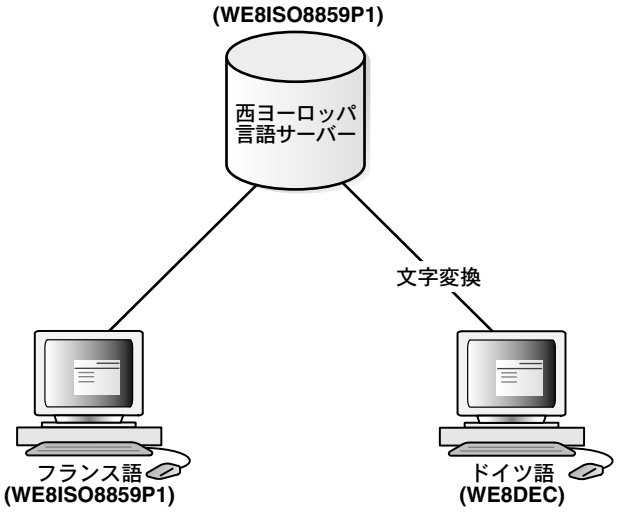
カタロニア語
デンマーク語
オランダ語
英語
フィンランド語
フランス語
ドイツ語
アイスランド語
イタリア語
ノルウェー語
ポルトガル語
スペイン語
スウェーデン語

これらの言語は、いずれもラテン語ベースの記述文字を使用します。

言語グループをサポートするキャラクタ・セットを使用する場合、データベースは**制限付き多言語サポート**機能を持つことになります。

図 2-7 に、Oracle キャラクタ・セット WE8ISO8850P1 を使用する西ヨーロッパ言語のサーバー、サーバーと同じキャラクタ・セットを使用するフランス語クライアントおよび WE8DEC キャラクタ・セットを使用するドイツ語クライアントを示します。ドイツ語クライアントは、サーバーと異なるキャラクタ・セットを使用しているため、文字変換が必要です。

図 2-7 制限付き多言語サポート



無制限多言語サポート

無制限多言語サポートが必要な場合は、サーバーのデータベース・キャラクタ・セットに Unicode などのユニバーサル・キャラクタ・セットを使用します。Unicode には、主に次の 2 つのコード体系があります。

- UTF-16: 各文字は長さ 2 バイトまたは 4 バイトです。
- UTF-8: 各文字の格納には 1 ～ 4 バイト必要です。

Oracle9*i* データベースでは、データベース・キャラクタ・セットとして UTF-8 が、各国語キャラクタ・セットとして UTF-8 と UTF-16 の両方がサポートされています。

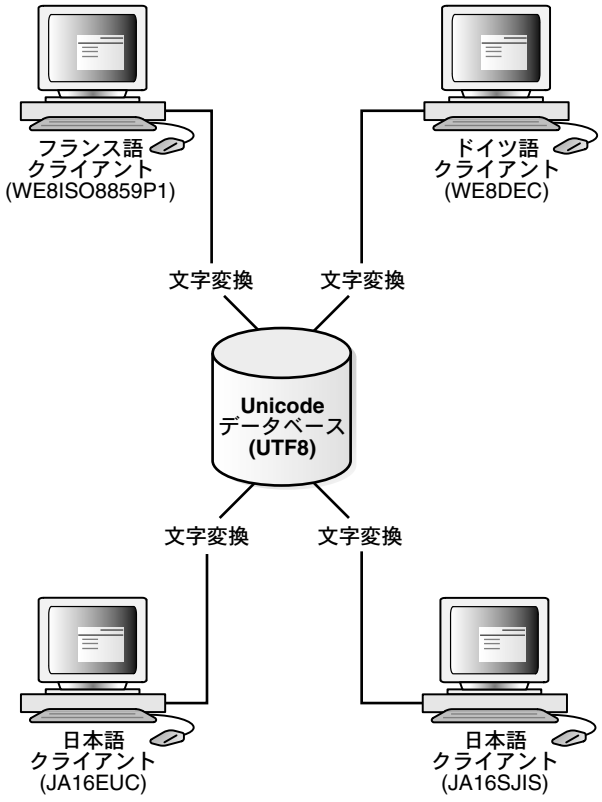
UTF-8 のデータベースとシングルバイト・キャラクタ・セットとの間でのキャラクタ・セット変換では、ごくわずかなオーバーヘッドが生じます。

UTF-8 とマルチバイト・キャラクタ・セットとの間での変換では、多少のオーバーヘッドが生じます。変換によるデータ消失はありませんが、これには次の例外があります。

- 一部のマルチバイト・キャラクタ・セットは、UTF-8 との間のキャラクタ・セット変換で、ユーザー定義文字をサポートしていません。
- 一部の Unicode 文字は、他のキャラクタ・セットで複数文字にマップされます。たとえば、1 つの Unicode 文字は JA16SJIS キャラクタ・セットの 3 文字にマップされます。これは、ラウンドトリップ変換でオリジナルの JA16SJIS 文字に変換されない場合があることを意味します。

図 2-8 に、Unicode UTF-8 キャラクタ・セットに基づく Oracle キャラクタ・セット AL32UTF8 を使用するサーバーを示します。

図 2-8 クライアント / サーバー構成での無制限多言語サポートの使用例



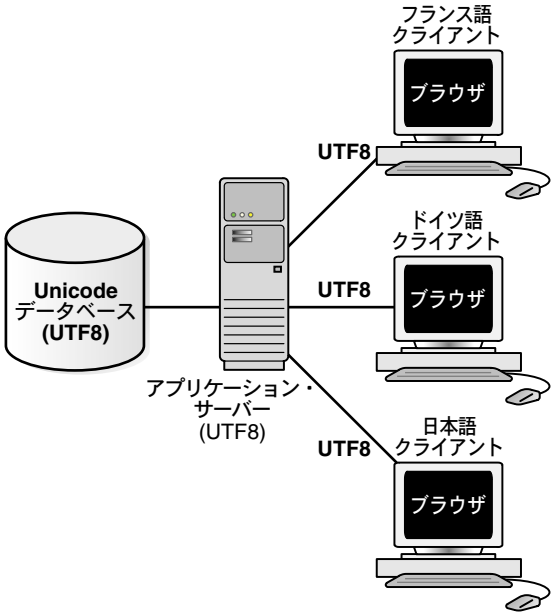
次の 4 つのクライアントがあるとします。

- Oracle キャラクタ・セット WE8ISO8859P1 を使用するフランス語クライアント
- WE8DEC キャラクタ・セットを使用するドイツ語クライアント
- JA16EUC キャラクタ・セットを使用する日本語クライアント
- JA16SJIS キャラクタ・セットを使用する日本語クライアント

各クライアントとサーバーの間で文字変換が発生しますが、AL32UTF8 がユニバーサル・キャラクタ・セットであるためデータ消失は生じません。ドイツ語クライアントが日本語クライアントの 1 つからデータを取り出そうとすると、キャラクタ・セット変換中にデータに含まれる日本語文字がすべて消失します。

図 2-9 に、複数層構成での Unicode ソリューションを示します。

図 2-9 複数層構成での複数層の無制限多言語サポートの使用例



データベース、アプリケーション・サーバーおよび各クライアントは、AL32UTF8 キャラクタ・セットを使用しています。このため、クライアントがフランス語、ドイツ語および日本語であっても、文字変換を行う必要はありません。

関連項目： 第 5 章「Unicode を使用した多言語データベースのサポート」

3

グローバリゼーション・サポート環境の設定

この章では、グローバリゼーション・サポート環境の設定方法について説明します。この章の内容は、次のとおりです。

NLS パラメータの設定

NLS パラメータによって、クライアントとサーバーの両方でのロケール固有の動作が決定します。NLS パラメータは、次の 4 通りの方法で指定できます。

- サーバー上の初期化パラメータとして指定します。

  パラメータを初期化パラメータ・ファイルに組み込むと、デフォルトのセッション NLS 環境を指定できます。この設定は、クライアント側には影響を与えません。サーバーの動作のみを制御します。次に例を示します。

  ```
  NLS_TERRITORY = "CZECH REPUBLIC"
  ```

- クライアントの環境変数として指定します。

  NLS パラメータを使用すると、クライアントに対してロケール依存の動作を指定できます。また、初期化パラメータ・ファイルのセッションに設定されているデフォルト値をオーバーライドできます。次に、UNIX システム上での例を示します。

  ```
  % setenv NLS_SORT FRENCH
  ```

- ALTER SESSION 文を使用して指定します。

  ALTER SESSION 文に設定されている NLS パラメータを使用すると、初期化パラメータ・ファイルのセッションに設定されたデフォルト値や環境変数を使用してクライアントで設定したデフォルト値をオーバーライドできます。

  ```
  ALTER SESSION SET NLS_SORT = FRENCH;
  ```

  関連項目： ALTER SESSION 文の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。

- SQL 関数内で指定します。

  NLS パラメータを明示的に使用すると、SQL 関数内での NLS 動作をハードコード化できます。この指定によって、初期化パラメータ・ファイルのセッションに設定されたデフォルト値、環境変数を使用してクライアントで設定されたデフォルト値、または ALTER SESSION によってセッションに設定されたデフォルト値がオーバーライドされます。次に例を示します。

  ```
  TO_CHAR(hiredate, 'DD/MON/YYYY', 'nls_date_language = FRENCH')
  ```

  関連項目： TO_CHAR 関数など、SQL 関数の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。

表 3-1 に、NLS パラメータの各種設定方法の優先順位を示します。優先順位の低い設定は、優先順位の高い設定によってオーバーライドされます。たとえば、デフォルト値の優先順位が最も低く、他のすべての方法でオーバーライドされます。また、SQL 関数内で NLS パラメータを設定すると、他のすべての設定方法がオーバーライドされます。

表 3-1 NLS パラメータの設定方法とその優先順位

優先順位	方法
1（最も高い優先順位）	SQL 関数での明示的な設定
2	ALTER SESSION 文による設定
3	環境変数としての設定
4	初期化パラメータ・ファイル内での指定
5	デフォルト

表 3-2 に、Oracle サーバーで使用可能な NLS パラメータを示します。

表 3-2 NLS パラメータ

パラメータ	説明	デフォルト	有効範囲： I = 初期化パラメータ・ファイル E= 環境変数 A = ALTER SESSION
NLS_CALENDAR	暦法	Gregorian（グレゴリオ暦）	I、E、A
NLS_COMP	SQL、PL/SQL 演算子の比較	BINARY	I、E、A
NLS_CREDIT	貸方の会計記号	NLS_TERRITORY から導出	E
NLS_CURRENCY	各国通貨記号	NLS_TERRITORY から導出	I、E、A
NLS_DATE_FORMAT	日付書式	NLS_TERRITORY から導出	I、E、A
NLS_DATE_LANGUAGE	曜日名と月名に使用する言語	NLS_LANGUAGE から導出	I、E、A
NLS_DEBIT	借方の会計記号	NLS_TERRITORY から導出	E
NLS_ISO_CURRENCY	ISO 国際通貨記号	NLS_TERRITORY から導出	I、E、A

表 3-2 NLS パラメータ（続き）

パラメータ	説明	デフォルト	有効範囲： I = 初期化パラメータ・ファイル E= 環境変数 A = ALTER SESSION
NLS_LANG **関連項目：**3-5 ページ「環境変数 NLS_LANG を使用したロケールの選択」	言語、地域、キャラクタ・セット	AMERICAN_AMERICA.US7ASCII	E
NLS_LANGUAGE	言語	NLS_LANG から導出	I、A
NLS_LENGTH_SEMANTICS	文字列の処理方法	BYTE	I、A
NLS_LIST_SEPARATOR	リスト内項目の区切り文字	NLS_TERRITORY から導出	E
NLS_MONETARY_CHARACTERS	ドルとセント（または相当する通貨）に使用する通貨記号	NLS_TERRITORY から導出	E
NLS_NCHAR_CONV_EXCP	キャラクタ・タイプ変換時のデータ消失のレポート	FALSE	I、A
NLS_NUMERIC_CHARACTERS	小数点文字とグループ・セパレータ	NLS_TERRITORY から導出	I、E、A
NLS_SORT	文字のソート基準	NLS_LANGUAGE から導出	I、E、A
NLS_TERRITORY	地域	NLS_LANG から導出	I、A
NLS_TIMESTAMP_FORMAT	タイムスタンプ	NLS_TERRITORY から導出	I、E、A
NLS_TIMESTAMP_TZ_FORMAT	タイム・ゾーン付きのタイムスタンプ	NLS_TERRITORY から導出	I、E、A
NLS_DUAL_CURRENCY	第 2 通貨記号	NLS_TERRITORY から導出	I、E、A

環境変数 NLS_LANG を使用したロケールの選択

ロケールとは、システムやプログラムを実行する言語的および文化的な環境のことです。ロケール動作を指定する最も簡単な方法は、NLS_LANG 環境パラメータを設定することです。このパラメータによって、クライアント・アプリケーションで使用される言語と地域が設定されます。また、クライアントのキャラクタ・セット（クライアント・プログラムによって入力または表示されるデータのキャラクタ・セット）も設定されます。

NLS_LANG パラメータには、3 つのコンポーネント、language、territory および charset があります。このパラメータは、句読点を含めて次の書式で指定します。

```
NLS_LANG = language_territory.charset
```

たとえば、Oracle のインストーラで NLS_LANG が移入されない場合、その値は AMERICAN_AMERICA.US7ASCII となります。この場合の language は AMERICAN、territory は AMERICA、charset は US7ASCII です。

NLS_LANG パラメータの各コンポーネントによって、グローバリゼーション・サポート機能のサブセットの操作が制御されます。

- *language*

  Oracle のメッセージ、ソート、曜日名および月名に使用する言語などの規則を指定します。各サポート対象言語には、AMERICAN、FRENCH または GERMAN などの一意の名前が付いています。language の引数によって、territory および charset の引数にデフォルト値が指定されます。language を指定しない場合、デフォルト値は AMERICAN に設定されます。

- *territory*

  デフォルトの日付、通貨単位および数値書式などの規則を指定します。各サポート対象地域には、AMERICA、FRANCE または CANADA などの一意の名前が付いています。territory を指定しない場合、値は language の値から導出されます。

- *charset*

  クライアント・アプリケーションが使用するキャラクタ・セット（通常はユーザーの端末で使用するキャラクタ・セット）を指定します。サポート対象の各キャラクタ・セットには、US7ASCII、WE8ISO8859P1、WE8DEC、WE8MSWIN1252 または JA16EUC などの一意の頭字語が付いています。各言語には、その言語に対応したデフォルトのキャラクタ・セットがあります。

注意： NLS_LANG 定義のコンポーネントはすべてオプションです。特に選択しない項目はデフォルト値に設定されます。territory または charset を指定する場合は、先行デリミタを付ける必要があります。先行デリミタは、territory の場合はアンダースコア（_）で、charset の場合はピリオド（.）です。先行デリミタを付けないと、値は言語名として解析されます。

NLS_LANG の 3 つの引数は、次の例のように、様々な組合せで指定できます。

```
NLS_LANG = AMERICAN_AMERICA.WE8MSWIN1252

NLS_LANG = FRENCH_CANADA.WE8DEC

NLS_LANG = JAPANESE_JAPAN.JA16EUC
```

非論理的な組合せも設定できますが、正しく動作しません。たとえば、次の指定では、西ヨーロッパ諸国のキャラクタ・セットを使用して日本語をサポートしようとしています。

```
NLS_LANG = JAPANESE_JAPAN.WE8DEC
```

WE8DEC キャラクタ・セットでは日本語文字がサポートされないため、この定義を NLS_LANG に使用すると日本語データを格納できません。

これ以降の内容は、次のとおりです。

- NLS_LANG の値の指定
- 言語指定と地域指定のオーバーライド

> **関連項目：** サポート対象のすべての言語、地域およびキャラクタ・セットのリストは、付録 A「ロケール・データ」を参照してください。

NLS_LANG の値の指定

NLS_LANG を環境変数として、コマンドラインで設定します。たとえば、UNIX オペレーティング・システム上では、次のような文を入力して、NLS_LANG の値を指定します。

```
% setenv NLS_LANG FRENCH_FRANCE.WE8DEC
```

NLS_LANG は環境変数であるため、クライアント・アプリケーションによって起動時に読み込まれます。クライアントは、NLS_LANG によって定義された情報を、データベース・サーバーへの接続時にサーバーに送信します。

次の例では、日付および数値の書式が NLS_LANG パラメータによってどのように影響を受けるかを示します。

例 3-1 NLS_LANG を American_America.WE8ISO8859P1 に設定する場合

言語が AMERICAN、地域が AMERICA、Oracle キャラクタ・セットが WE8ISO8859P1 となるように、NLS_LANG を設定します。

```
% setenv NLS_LANG American_America.WE8ISO8859P1
```

SELECT 文を入力します。

```
SQL> SELECT last_name, hire_date, ROUND(salary/8,2) salary FROM employees;
```

結果は次のようになります。

```
LAST_NAME                 HIRE_DATE     SALARY
------------------------- --------- ----------
Sciarra                   30-SEP-97      962.5
Urman                     07-MAR-98        975
Popp                      07-DEC-99      862.5
```

例 3-2 NLS_LANG を French_France.WE8ISO8859P1 に設定する場合

言語が FRENCH、地域が FRANCE、Oracle キャラクタ・セットが WE8ISO8859P1 となるように、NLS_LANG を設定します。

```
% setenv NLS_LANG French_France.WE8ISO8859P1
```

例 3-1 に示した問合せでは、次の出力が戻されます。

```
LAST_NAME                 HIRE_DAT     SALARY
------------------------- -------- ----------
Sciarra                   30/09/97      962,5
Urman                     07/03/98        975
Popp                      07/12/99      862,5
```

日付書式と数値書式が変更されていることに注意してください。基礎となるデータは同じであるため、数値に変更はありません。

言語指定と地域指定のオーバーライド

NLS_LANG パラメータは、サーバー・セッション（SQL コマンドの実行など）とクライアント・アプリケーション（Oracle のツール製品での書式設定表示など）の両方で使用する言語と地域の環境を設定します。このパラメータを使用することで、データベースとクライアント・アプリケーションの言語環境が自動的に同じになります。

NLS_LANG パラメータの language コンポーネントと territory コンポーネントによって、他の詳細な NLS パラメータ（日付書式、数字および言語ソートなど）のデフォルト値が決定されます。NLS_LANG パラメータがすでに設定されている場合は、これらの詳細パラメータをそれぞれクライアント環境で設定してデフォルト値をオーバーライドできます。

NLS_LANG パラメータを設定しない場合、サーバー・セッション環境は、初期化パラメータ・ファイルの NLS_LANGUAGE、NLS_TERRRITORY および他の NLS インスタンスのパラメータの値によって初期化された状態のままです。これらのパラメータを変更して、インスタンスを再起動すると、デフォルト値を変更できます。

セッション中に NLS 環境の動的な変更が必要な場合があります。その場合は、ALTER SESSION 文を使用して NLS_LANGUAGE、NLS_TERRITORY および他の NLS パラメータを変更できます。

注意： クライアント・キャラクタ・セットの設定は、ALTER SESSION 文では変更できません。

ALTER SESSION 文が変更するのは、セッション環境のみです。ローカル・クライアントの NLS 環境は、クライアントが新しい設定を明示的に取得してローカル環境を変更しないかぎり、変更されません。

関連項目：

- 3-16 ページ「セッション中の NLS_LANGUAGE および NLS_TERRITORY のデフォルト値のオーバーライド」
- 『Oracle9*i* SQL リファレンス』

NLS_LANG 設定とデータベース・キャラクタ・セットを一致させる必要があるかどうか

NLS_LANG のキャラクタ・セットには、オペレーティング・システムのクライアントの設定を反映させる必要があります。たとえば、データベース・キャラクタ・セットが UTF8 で、クライアントのオペレーティング・システムが Windows の場合、UTF8 WIN32 クライアントはないため、クライアント・キャラクタ・セットとして UTF8 を設定しないでください。かわりに、NLS_LANG の設定にクライアントのコード・ページを反映させる必要があります。

UNIX プラットフォーム上では、NLS_LANG はローカルの環境変数として設定されます。

Windows プラットフォームでは、NLS_LANG はレジストリ内で設定されます。たとえば、英語版 Windows クライアントでは、コード・ページは WE8MSWIN1252 です。NLS_LANG の適切な設定は、AMERICAN_AMERICA.WE8MSWIN1252 です。

NLS_LANG を適切に設定すると、クライアント・オペレーティング・システムのコード・ページからデータベース・キャラクタ・セットへと適切に変換できます。これらの設定が同じときは、送受信されるデータはデータベース・キャラクタ・セットと同一のキャラクタ・セットでエンコードされているとみなされ、妥当性チェックや変換は実行されません。このため、クライアントのコード・ページとデータベース・キャラクタ・セットが異なっていて、変換が必要な場合は、データが破損する可能性があります。

関連項目： Windows で NLS_LANG パラメータに一般に使用できる値の詳細は、『Oracle9*i* Database for Windows インストレーション・ガイド』を参照してください。

NLS データベース・パラメータ

CREATE DATABASE 文の実行中に新しいデータベースが作成されると、NLS データベース環境が設定されます。現行の NLS インスタンス・パラメータは、データベース・キャラクタ・セットおよび各国語キャラクタ・セットとともにデータ・ディクショナリに格納されます。NLS インスタンス・パラメータは、インスタンスの起動時に初期化パラメータ・ファイルから読み込まれます。

NLS パラメータの値は、次のいずれかを使用して検索できます。

- NLS データ・ディクショナリ・ビュー
- NLS 動的パフォーマンス・ビュー
- OCINlsGetInfo() 関数

NLS データ・ディクショナリ・ビュー

アプリケーションでは、次のデータ・ディクショナリ・ビューに問い合せて、セッション、インスタンスおよびデータベースの NLS パラメータをチェックできます。

- NLS_SESSION_PARAMETERS には、このビューに問合せ中のセッションの NLS パラメータとその値が表示されます。キャラクタ・セットに関する情報は表示されません。
- NLS_INSTANCE_PARAMETERS には、明示的に設定されている現行の NLS インスタンス・パラメータとその値が表示されます。
- NLS_DATABASE_PARAMETERS には、データベースの作成時に使用された NLS パラメータの値が表示されます。

NLS 動的パフォーマンス・ビュー

アプリケーションでは、次の NLS 動的パフォーマンス・ビューをチェックできます。

- V$NLS_VALID_VALUES には、NLS パラメータ NLS_LANGUAGE、NLS_SORT、NLS_TERRITORY および NLS_CHARACTERSET の値が表示されます。
- V$NLS_PARAMETERS には、NLS パラメータ NLS_CALENDAR、NLS_CHARACTERSET、NLS_CURRENCY、NLS_DATE_FORMAT、NLS_DATE_LANGUAGE、NLS_ISO_CURRENCY、NLS_LANGUAGE、NLS_NUMERIC_CHARACTERS、NLS_SORT、NLS_TERRITORY、NLS_NCHAR_CHARACTERSET、NLS_COMP、NLS_LENGTH_SEMANTICS、NLS_NCHAR_CONV_EXP、NLS_TIMESTAMP_FORMAT、NLS_TIMESTAMP_TZ_FORMAT、NLS_TIME_FORMAT および NLS_TIME_TZ_FORMAT の現行の値が表示されます。

関連項目：『Oracle9*i* データベース・リファレンス』

OCINlsGetInfo() 関数

ユーザー・アプリケーションでは、OCINlsGetInfo() 関数を使用して、クライアントの NLS 設定を問い合せることができます。

> **関連項目：** OCINlsGetInfo() 関数の詳細は、第 8 章「グローバル環境での OCI プログラミング」を参照してください。

言語および地域のパラメータ

この項では、次のパラメータについて説明します。

- NLS_LANGUAGE
- NLS_TERRITORY

NLS_LANGUAGE

パラメータ・タイプ： 文字列

パラメータの有効範囲： 初期化パラメータおよび ALTER SESSION

デフォルト値： NLS_LANG から導出

値の範囲： 有効な任意の言語名

NLS_LANGUAGE は、次のセッション特性に対してデフォルトの規則を指定します。

- サーバー・メッセージの言語
- 曜日名と月名の言語およびその略称（SQL 関数 TO_CHAR と TO_DATE で指定されます）
- AM、PM、AD および BC に相当する記号（A.M.、P.M.、A.D. および B.C. は、NLS_LANGUAGE が AMERICAN に設定されている場合のみ有効です）
- ORDER BY の指定時に使用する文字データに対するデフォルトのソート基準（ORDER BY を指定しない場合、GROUP BY はバイナリ・ソートを使用します）
- 書込み方向
- 肯定的および否定的な応答文字列（YES および NO など）

初期化パラメータ・ファイルの NLS_LANGUAGE に指定されている値は、そのインスタンス内のすべてのセッションに対するデフォルト値になります。たとえば、デフォルトのセッション言語をフランス語に指定するには、パラメータを次のように設定する必要があります。

```
NLS_LANGUAGE = FRENCH
```

サーバー・メッセージ

```
ORA-00942:  table or view does not exist
```

言語がフランス語の場合は、次のように表示されます。

```
ORA-00942:  table ou vue inexistante
```

サーバーが使用するメッセージは、$ORACLE_HOME/*product_name*/mesg ディレクトリにあるバイナリ形式ファイル、あるいはそれに相当するオペレーティング・システム用ファイルに格納されます。このファイルには、次のファイル名規則に従って、サポート対象言語ごとに 1 バージョンずつ、複数のバージョンが存在します。

```
<product_id><language_abbrev>.MSB
```

たとえば、F はフランス語の略称であるため、フランス語のサーバー・メッセージが含まれたファイルは ORAF.MSB と呼ばれます。

メッセージは、言語とオペレーティング・システムに応じて、特定キャラクタ・セットごとにこれらのファイルに格納されます。このキャラクタ・セットがデータベース・キャラクタ・セットと異なる場合、メッセージ・テキストはデータベース・キャラクタ・セットに自動的に変換されます。クライアント・キャラクタ・セットがデータベース・キャラクタ・セットと異なる場合、メッセージ・テキストは、必要に応じてクライアント・キャラクタ・セットに変換されます。このようにキャラクタ・セット変換の制限に従うことによって、メッセージはユーザーの端末に正しく表示されます。

NLS_LANGUAGE のデフォルト値が、オペレーティング・システム固有の値である場合があります。初期化パラメータ・ファイルでその値を変更してインスタンスを再起動すると、NLS_LANGUAGE パラメータを変更できます。

> **関連項目：** NLS_LANGUAGE のデフォルト値の詳細は、オペレーティング・システム固有の Oracle マニュアルを参照してください。

メッセージとテキストはすべて同じ言語にする必要があります。たとえば、Oracle Developer アプリケーションの実行時にユーザーが参照するメッセージとボイラープレート・テキストは、次の 3 つのソースから導出されます。

- サーバーからのメッセージ
- Oracle Forms によって生成されたメッセージとボイラープレート・テキスト
- アプリケーションによって生成されたメッセージとボイラープレート・テキスト

最初の 2 種類のテキストに使用される言語は、NLS によって決定されます。アプリケーションは、そのメッセージとボイラープレート・テキストに使用する言語を受け持ちます。

次の例に、NLS_LANGUAGE を異なる値に設定した場合の動作を示します。

例 3-3 NLS_LANGUAGE=ITALIAN

ALTER SESSION 文を使用して、NLS_LANGUAGE をイタリア語に設定します。

```
ALTER SESSION SET NLS_LANGUAGE=Italian;
```

SELECT 文を入力します。

```
SQL> SELECT last_name, hire_date, ROUND(salary/8,2) salary FROM employees;
```

結果は次のようになります。

```
LAST_NAME                 HIRE_DATE     SALARY
------------------------- --------- ----------
Sciarra                   30-SET-97      962.5
Urman                     07-MAR-98        975
Popp                      07-DIC-99      862.5
```

月名の略称にイタリア語が使用されていることに注意してください。

> **関連項目：** ALTER SESSION 文の使用方法の詳細は、3-16 ページの「セッション中の NLS_LANGUAGE および NLS_TERRITORY のデフォルト値のオーバーライド」を参照してください。

例 3-4 NLS_LANGUAGE=GERMAN

ALTER SESSION 文を使用して、言語をドイツ語に変更します。

```
SQL> ALTER SESSION SET NLS_LANGUAGE=German;
```

前述と同じ SELECT 文を入力します。

```
SQL> SELECT last_name, hire_date, ROUND(salary/8,2) salary FROM employees;
```

結果は次のようになります。

```
LAST_NAME                 HIRE_DATE     SALARY
------------------------- --------- ----------
Sciarra                   30-SEP-97      962.5
Urman                     07-MÄR-98        975
Popp                      07-DEZ-99      862.5
```

月名の略称の言語が変更されていることに注意してください。

NLS_TERRITORY

パラメータ・タイプ: 文字列

パラメータの有効範囲: 初期化パラメータおよびALTER SESSION

デフォルト値: NLS_LANGから導出

値の範囲: 有効な任意の地域名

NLS_TERRITORYは、次のデフォルトの日付および数値の書式特性に関する規則を指定します。

- 日付書式
- 小数点文字とグループ・セパレータ
- 各国通貨記号
- ISO通貨記号
- 第2通貨記号
- 週の最初の曜日
- 貸方および借方記号
- ISO週フラグ
- リスト・セパレータ

初期化パラメータ・ファイルのNLS_TERRITORYに指定されている値は、そのインスタンスに対するデフォルト値になります。たとえば、デフォルトをフランスに指定するには、パラメータを次のように設定する必要があります。

```
NLS_TERRITORY = FRANCE
```

地域がFRANCEの場合、数値は小数点文字にカンマを使用して書式設定されます。

初期化パラメータ・ファイルでその値を変更してインスタンスを再起動すると、NLS_TERRITORYパラメータを変更できます。NLS_TERRITORYのデフォルト値が、オペレーティング・システム固有の値である場合があります。

NLS_LANGがクライアント環境に指定されている場合、初期化パラメータ・ファイル内のNLS_TERRITORYの値は接続時にオーバーライドされます。

地域は、ALTER SESSION文にNLS_TERRITORYの新しい値を指定することで、セッション中に動的に変更できます。NLS_TERRITORYを変更すると、すべての導出NLSセッション・パラメータは、新しい地域のデフォルト値に再設定されます。

セッション中に地域をフランスに変更するには、次の ALTER SESSION 文を発行します。

```
ALTER SESSION SET NLS_TERRITORY=France;
```

次の例に、NLS_TERRITORY と NLS_LANGUAGE を異なる値に設定した場合の動作を示します。

例 3-5 NLS_LANGUAGE=AMERICAN、NLS_TERRITORY=AMERICA

次の SELECT 文を入力します。

```
SQL> SELECT TO_CHAR(salary,'L99G999D99') salary FROM employees;
```

NLS_TERRITORY を AMERICA に設定し、NLS_LANGUAGE を AMERICAN に設定すると、次のような結果が出力されます。

```
SALARY
--------------------
$24,000.00
$17,000.00
$17,000.00
```

例 3-6 NLS_LANGUAGE=AMERICAN、NLS_TERRITORY=GERMANY

次の ALTER SESSION 文を使用して地域をドイツに変更します。

```
ALTER SESSION SET NLS_TERRITORY = Germany;
Session altered.
```

前述と同じ SELECT 文を入力します。

```
SQL> SELECT TO_CHAR(salary,'L99G999D99') salary FROM employees;
```

結果は次のようになります。

```
SALARY
--------------------
€24.000,00
€17.000,00
€17.000,00
```

通貨記号が $ から € に変更されていることに注意してください。基礎となるデータは同じであるため、数値に変更はありません。

> **関連項目：** ALTER SESSION 文の使用方法の詳細は、3-16 ページの「セッション中の NLS_LANGUAGE および NLS_TERRITORY のデフォルト値のオーバーライド」を参照してください。

例 3-7 NLS_LANGUAGE=GERMAN、NLS_TERRITORY=GERMANY

ALTER SESSION 文を使用して言語をドイツ語に変更します。

```
ALTER SESSION SET NLS_LANGUAGE = German;
Sitzung wurde geändert.
```

サーバー・メッセージがドイツ語で表示されることに注意してください。

前述と同じ SELECT 文を入力します。

```
SQL> SELECT TO_CHAR(salary,'L99G999D99') salary FROM employees;
```

例 3-6 と同じ結果が表示されます。

```
SALARY
--------------------
€24.000,00
€17.000,00
€17.000,00
```

例 3-8 NLS_LANGUAGE=GERMAN、NLS_TERRITORY=AMERICA

次の ALTER SESSION 文を使用して地域をアメリカに変更します。

```
ALTER SESSION SET NLS_TERRITORY = America;
Sitzung wurde geändert.
```

他の例と同じ SELECT 文を入力します。

```
SQL> SELECT TO_CHAR(salary,'L99G999D99') salary FROM employees;
```

出力は次のようになります。

```
SALARY
--------------------
$24.000,00
$17.000,00
$17.000,00
```

地域がドイツからアメリカに変更されたため、通貨記号が € から $ に変更されていることに注意してください。

セッション中の NLS_LANGUAGE および NLS_TERRITORY のデフォルト値のオーバーライド

NLS_LANGUAGE と NLS_TERRITORY のデフォルト値は、ALTER SESSION 文を使用してセッション中にオーバーライドできます。

例 3-9 NLS_LANG=ITALIAN_ITALY.WE8DEC

言語がイタリア語、地域がイタリア、キャラクタ・セットが WE8DEC になるように、環境変数 NLS_LANG を設定します。

```
% setenv NLS_LANG Italian_Italy.WE8DEC
```

SELECT 文を入力します。

```
SQL> SELECT last_name, hire_date, ROUND(salary/8,2) salary FROM employees;
```

出力は次のようになります。

```
LAST_NAME                 HIRE_DATE     SALARY
------------------------- --------- ----------
Sciarra                   30-SET-97      962,5
Urman                     07-MAR-98        975
Popp                      07-DIC-99      862,5
```

月の略称の言語と小数点文字に注意してください。

例 3-10 言語、日付書式および小数点文字の変更

ALTER SESSION 文を使用して、言語、日付書式および小数点文字を変更します。

```
SQL> ALTER SESSION SET NLS_LANGUAGE=german;

Session wurde geändert.

SQL> ALTER SESSION SET NLS_DATE_FORMAT='DD.MON.YY';

Session wurde geändert.

SQL> ALTER SESSION SET NLS_NUMERIC_CHARACTERS='.,';

Session wurde geändert.
```

例 3-9 に示した SELECT 文を入力します。

```
SQL> SELECT last_name, hire_date, ROUND(salary/8,2) salary FROM employees;
```

出力は次のようになります。

```
LAST_NAME                 HIRE_DATE     SALARY
------------------------- --------- ----------
Sciarra                   30.SEP.97      962.5
Urman                     07.MÄR.98        975
Popp                      07.DEZ.99      862.5
```

月の略称の言語、日付書式および小数点文字に注意してください。

環境変数 NLS_LANG の動作によって、各セッションのデータベースの言語環境が暗黙的に決定されます。セッションでのデータベース接続時に、ALTER SESSION 文が自動的に実行され、データベース・パラメータ NLS_LANGUAGE および NLS_TERRITORY の値が、NLS_LANG の language および territory 引数で指定された値に設定されます。NLS_LANG が定義されていない場合、ALTER SESSION 文は暗黙的に実行されません。

NLS_LANG が定義されている場合は、セッションの接続先（直接接続と間接接続の両方）インスタンスすべてに対して、ALTER SESSION 文が暗黙的に実行されます。NLS パラメータの値が、ALTER SESSION によってセッション中に明示的に変更されると、その変更内容は、ユーザー・セッションの接続先のインスタンスすべてに反映されます。

日付および時間を指定するパラメータ

Oracle では、日付と時刻の表示を制御できます。この項の内容は、次のとおりです。

- 日付書式
- 時刻書式

日付書式

表 3-3 に、様々な日付書式を示します。

表 3-3 日付書式

国名	説明	例
エストニア	dd.mm.yyyy	28.02.1998
ドイツ	dd-mm-rr	28-02-98
日本	rr-mm-dd	98-02-28
英国	dd-mon-rr	28-Feb-98
米国	dd-mon-rr	28-Feb-98

この項では、次のパラメータについて説明します。

- NLS_DATE_FORMAT
- NLS_DATE_LANGUAGE

NLS_DATE_FORMAT

パラメータ・タイプ: 文字列

パラメータの有効範囲: 初期化パラメータ、環境変数および ALTER SESSION

デフォルト値: 特定地域のデフォルト書式

値の範囲: 有効な日付書式マスク

NLS_DATE_FORMAT パラメータは、TO_CHAR 関数および TO_DATE 関数で使用するデフォルトの日付書式を定義します。NLS_TERRITORY パラメータは、NLS_DATE_FORMAT のデフォルト値を決定します。NLS_DATE_FORMAT の値には、有効な任意の日付書式マスクを指定できます。この値は、次のように引用符で囲む必要があります。

```
NLS_DATE_FORMAT = "MM/DD/YYYY"
```

日付書式に文字列リテラルを追加する場合は、その文字列リテラルを二重引用符で囲みます。二重引用符などの特殊な文字を使用する場合は、必ずエスケープ文字を前に付けてください。また、式全体は一重引用符で囲んでください。次に例を示します。

```
NLS_DATE_FORMAT = '\"Today\'s date\" MM/DD/YYYY'
```

例 3-11 ローマ数字を表示する日付書式の設定

デフォルトの日付書式を設定して、月をローマ数字で表示するには、初期化パラメータ・ファイルに次の行を加えます。

```
NLS_DATE_FORMAT = "DD RM YYYY"
```

次の SELECT 文を入力します。

```
SELECT TO_CHAR(SYSDATE) currdate FROM dual;
```

今日の日付が 1997 年 2 月 12 日の場合は、次の出力が表示されます。

```
CURRDATE
---------
12 II 1997
```

NLS_DATE_FORMAT の値は、内部日付書式で格納されます。各書式要素は 2 バイトを占め、各文字列は、文字列のバイト数に終了記号の 1 バイト数を加えたバイト数を占めます。また、書式マスク全体には 2 バイトの終了記号があります。たとえば、MM/DD/YY は、内部で 12 バイトを占めます。これは、書式マスクが 3 つの書式要素（月、日および年）、2 つの

1 バイト文字列（2 個のスラッシュ）および 2 バイトの終了記号で構成されるためです。NLS_DATE_FORMAT の値の書式は、24 バイト以内で設定する必要があります。

注意： アプリケーションを設計する場合、可変長のデフォルトの日付書式を扱える様にする必要があります。また、パラメータ値は二重引用符で囲む必要があります。一重引用符は、書式マスクの一部として解釈されます。

NLS_DATE_FORMAT のデフォルト値は、次の方法で変更できます。

- 初期化パラメータ・ファイル内で値を変更してから、インスタンスを再起動します。
- ALTER SESSION SET NLS_DATE_FORMAT 文を使用します。

関連項目： 日付書式要素と ALTER SESSION 文の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。

表または索引が日付列でパーティション化されている場合や、NLS_DATE_FORMAT で指定した日付書式では年の最初の 2 桁が指定されない場合は、年に 4 文字の書式マスクを指定して TO_DATE 関数を使用する必要があります。

次に例を示します。

```
TO_DATE('11-jan-1997', 'dd-mon-yyyy')
```

関連項目： 表および索引をパーティション化する方法と TO_DATE の使用方法の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。

NLS_DATE_LANGUAGE

パラメータ・タイプ： 文字列

パラメータの有効範囲： 初期化パラメータ、環境変数および ALTER SESSION

デフォルト値： NLS_LANGUAGE から導出

値の範囲： 有効な任意の言語名

NLS_DATE_LANGUAGE パラメータは、TO_CHAR および TO_DATE 関数で生成される曜日名と月名の言語を指定します。NLS_DATE_LANGUAGE によって、NLS_LANGUAGE で暗黙的に指定された言語がオーバーライドされます。NLS_DATE_LANGUAGE の構文は、NLS_LANGUAGE パラメータの構文と同じで、サポート対象言語すべてが有効値です。

また、NLS_DATE_LANGUAGE によって、次の表記に使用する言語が決定されます。

- TO_CHAR および TO_DATE 関数で戻される月名と曜日名の略称
- デフォルトの日付書式（NLS_DATE_FORMAT）で使用される月名と曜日名の略称
- AM、PM、AD および BC の略称

例 3-12 NLS_DATE_LANGUAGE=FRENCH、月名および曜日名

日付の言語をフランス語に設定します。

```
ALTER SESSIONS SET NLS_DATE_LANGUAGE = FRENCH
```

SELECT 文を入力します。

```
SELECT TO_CHAR(SYSDATE, 'Day:Dd Month yyyy') FROM dual;
```

出力は次のようになります。

```
TO_CHAR(SYSDATE,'DAY:DDMONTHYYYY')
-----------------------------------------------------------
Vendredi:07 Décembre  2001
```

TO_CHAR 関数を使用して表記される数字には、常に英語表記が使用されます。たとえば、次の SELECT 文を入力します。

```
SQL> SELECT TO_CHAR(TO_DATE('12-Oct-2001'),'Day: ddspth Month') FROM dual;
```

出力は次のようになります。

```
TO_CHAR(TO_DATE('12-OCT-2001'),'DAY:DDSPTHMONTH')
-------------------------------------------------------------------
Vendredi: twelfth Octobre
```

例 3-13 NLS_DATE_LANGUAGE=FRENCH、月名および曜日名の略称

月名と曜日名の略称は、NLS_DATE_LANGUAGE によって決定されます。次の SELECT 文を入力します。

```
SELECT TO_CHAR(SYSDATE, 'Dy:dd Mon yyyy') FROM dual;
```

出力は次のようになります。

```
TO_CHAR(SYSDATE,'DY:DDMO
------------------------
Ve:07 Dec 2001
```

例 3-14 NLS_DATE_LANGUAGE=FRENCH、デフォルトの日付書式

デフォルトの日付書式では、NLS_DATE_LANGUAGE によって決定された月名の略称が使用されます。たとえば、デフォルトの日付書式が DD-MON-YYYY の場合、日付は次のように挿入されます。

```
INSERT INTO tablename VALUES ('12-Fév-1997');
```

関連項目：『Oracle9*i* SQL リファレンス』

時刻書式

表 3-4 に、様々な時刻書式を示します。

表 3-4 時刻書式

国名	説明	例
エストニア	hh24:mi:ss	13:50:23
ドイツ	hh24:mi:ss	13:50:23
日本	hh24:mi:ss	13:50:23
英国	hh24:mi:ss	13:50:23
米国	hh:mi:ssxff am	1:50:23.555 PM

この項では、次のパラメータについて説明します。

- NLS_TIMESTAMP_FORMAT
- NLS_TIMESTAMP_TZ_FORMAT

NLS_TIMESTAMP_FORMAT

パラメータ・タイプ： 文字列

パラメータの有効範囲： 動的、初期化パラメータ、環境変数および ALTER SESSION

デフォルト値： NLS_TERRITORY から導出

値の範囲： 有効な日時書式マスク

NLS_TIMESTAMP_FORMAT は、TO_CHAR 関数および TO_TIMESTAMP 関数で使用するデフォルトのタイムスタンプ書式を定義します。この値は、次のように引用符で囲む必要があります。

```
NLS_TIMESTAMP_FORMAT  = 'YYYY-MM-DD HH:MI:SS.FF'
```

例 3-15 タイムスタンプ書式

```
SQL> SELECT TO_TIMESTAMP('11-nov-2000 01:00:00.336', 'dd-mon-yyyy hh:mi:ss.ff')
    FROM dual;
```

出力は次のようになります。

```
TO_TIMESTAMP('11-NOV-200001:00:00.336','DD-MON-YYYYHH:MI:SS.FF')
---------------------------------------------------------------------------
11-NOV-00 01:00:00.336000000
```

NLS_TIMESTAMP_FORMAT の値を指定するには、その値を初期化パラメータ・ファイルに設定します。設定した値は、クライアントのクライアント環境変数として指定できます。

また、次の方法で NLS_TIMESTAMP_FORMAT の値を変更することもできます。

- 初期化パラメータ・ファイル内で値を変更してから、インスタンスを再起動します。
- ALTER SESSION SET NLS_TIMESTAMP_FORMAT 文を使用します。

> **関連項目：** TO_TIMESTAMP 関数と ALTER SESSION 文の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。

NLS_TIMESTAMP_TZ_FORMAT

パラメータ・タイプ： 文字列

パラメータの有効範囲： 動的、初期化パラメータ、環境変数および ALTER SESSION

デフォルト値： NLS_TERRITORY から導出

値の範囲： 有効な日時書式マスク

NLS_TIMESTAMP_TZ_FORMAT は、タイム・ゾーン付きタイムスタンプのデフォルトの書式を定義します。このパラメータは、TO_CHAR および TO_TIMESTAMP_TZ 関数で使用します。

NLS_TIMESTAMP_TZ_FORMAT の値を指定するには、その値を初期化パラメータ・ファイルに設定します。設定した値は、クライアントのクライアント環境変数として指定できます。

例 3-16 NLS_TIMESTAMP_TZ_FORMAT の設定

書式の値は、次のように引用符で囲む必要があります。

```
NLS_TIMESTAMP_TZ_FORMAT  = 'YYYY-MM-DD HH:MI:SS.FF TZH:TZM'
```

次の TO_TIMESTAMP_TZ 関数の例では、NLS_TIMESTAMP_TZ_FORMAT に指定された書式の値を使用しています。

```
SQL> SELECT TO_TIMESTAMP_TZ('2000-08-20, 05:00:00.55 America/Los_Angeles',
'yyyy-mm-dd hh:mi:ss.ff TZR') FROM dual;
```

出力は次のようになります。

```
TO_TIMESTAMP_TZ('2000-08-20,05:00:00.44AMERICA/LOS_ANGELES','YYYY-MM-DDHH:M
------------------------------------------------------------------------
20-AOU-00 05:00:00.440000000 AMERICA/LOS_ANGELES
```

NLS_TIMESTAMP_TZ_FORMAT の値は次の方法で変更できます。

- 初期化パラメータ・ファイル内で値を変更してから、インスタンスを再起動します。
- ALTER SESSION 文を使用します。

> **関連項目：** TO_TIMESTAMP_TZ 関数と ALTER SESSION 文の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。

データベースのタイム・ゾーン・パラメータ 次の項目を指定すると、特定のタイム・ゾーンを持つデータベースを作成できます。

- UTC（協定世界時、以前のグリニッジ標準時）との時差。次の例では、データベースのタイム・ゾーンを太平洋標準時（UTC より 8 時間遅れ）に設定します。

  ```
  CREATE DATABASE ... SET TIME_ZONE = '-08:00 ';
  ```

- タイム・ゾーン・リージョン。次の例も、データベースのタイム・ゾーンを米国での太平洋標準時に設定します。

  ```
  CREATE DATABASE ... SET TIME_ZONE = 'PST ';
  ```

  有効なリージョン名のリストを参照するには、V$TIMEZONE_NAMES ビューを問い合せます。

データベースのタイム・ゾーンが関連するのは、TIMESTAMP WITH LOCAL TIME ZONE 列のみです。Oracle では、TIMESTAMP WITH LOCAL TIME ZONE のデータは、ディスクへの格納時に、すべて正規化されます。SET TIME_ZONE 句を指定しない場合は、サーバーのオペレーティング・システムのタイム・ゾーンが使用されます。オペレーティング・システムのタイム・ゾーンが有効な Oracle タイム・ゾーンでない場合、データベースのタイム・ゾーンは UTC にデフォルト設定されます。Oracle のタイム・ゾーン情報は、ftp://elsie.nci.nih.gov/pub/ で使用可能な公開タイム・ゾーンから導出されます。Oracle のタイム・ゾーン情報には、このサイトから入手可能な最新のタイム・ゾーン・データが反映されていない場合があります。

データベース作成後は、ALTER DATABASE SET TIME_ZONE 文を発行してから、データベースを停止し、再起動することで、タイム・ゾーンを変更できます。次の例では、データベースのタイム・ゾーンをロンドン時間に設定します。

```
ALTER DATABASE SET TIME_ZONE = 'Europe/London ';
```

データベースのタイム・ゾーンを検索するには、次の例に示すように、DBTIMEZONE 関数を使用します。

```
SELECT dbtimezone FROM dual;

DBTIME
-------
-08:00
```

セッションのタイム・ゾーン・パラメータ ユーザー・セッションのタイム・ゾーン・パラメータを変更するには、ALTER SESSION 文を発行します。

- オペレーティング・システムのローカルのタイム・ゾーン

  ```
  ALTER SESSION SET TIME_ZONE = local;
  ```

- データベースのタイム・ゾーン

  ```
  ALTER SESSION SET TIME_ZONE = DBTIMEZONE;
  ```

- UTC からの絶対時差

  ```
  ALTER SESSION SET TIME_ZONE = '-05:00';
  ```

- 指定したリージョンのタイム・ゾーン

  ```
  ALTER SESSION SET TIME_ZONE = 'America/New_York';
  ```

環境変数 ORA_SDTZ を使用すると、クライアント・セッションのデフォルトのタイム・ゾーンを設定できます。この変数は、DB_TZ、OS_TZ、タイム・ゾーン・リージョンまたは数値タイム・ゾーン・オフセットなどの入力を取得します。ORA_SDTZ が DB_TZ に設定されている場合、セッションのタイム・ゾーンは、データベースのタイム・ゾーンと同じになります。ORA_SDTZ が OS_TZ に設定されている場合、セッションのタイム・ゾーンは、オペレーティング・システムのタイム・ゾーンと同じになります。ORA_SDTZ が無効な Oracle タイム・ゾーンに設定されている場合は、オペレーティング・システムのタイム・ゾーンが、セッションのデフォルト・タイム・ゾーンとして使用されます。オペレーティング・システムのタイム・ゾーンが有効な Oracle タイム・ゾーンでない場合、セッションのタイム・ゾーンは、UTC にデフォルト設定されます。ユーザー・セッションのタイム・ゾーンを検索するには、次の例に示すように、SESSIONTIMEZONE 関数を使用します。

```
SELECT sessiontimezone FROM dual;

SESSIONTIMEZONE
---------------
         -08:00
```

関連項目： 12-17 ページ「タイム・ゾーン・データのカスタマイズ」

カレンダ定義

この項の内容は、次のとおりです。

カレンダ書式

次のカレンダ情報が地域別に格納されます。

週の最初の曜日

一部の文化では、日曜日を最初の曜日とみなしています。また、月曜日を最初の曜日とみなす文化もあります。ドイツのカレンダは、表 3-5 に示すように月曜日から始まります。

表 3-5 ドイツのカレンダの例 : 1998 年 3 月

Mo	Di	Mi	Do	Fr	Sa	So
-	-	-	-	-	-	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31	-	-	-	-	-

週の最初の曜日は、NLS_TERRITORY パラメータによって決定されます。

関連項目： 3-13 ページ「NLS_TERRITORY」

年の最初の暦週

一部の国では、週番号を使用してスケジューリング、計画および会計処理を行います。Oracle では、この規則をサポートしています。ISO 規格では、暦年の週番号とは異なる週番号を使用できます。たとえば、1988 年 1 月 1 日は、1987 年の ISO 週番号 53 になります。ISO の週は、常に月曜日に始まって日曜日に終わります。

- 1 月 1 日が金曜日、土曜日または日曜日の場合、1 月 1 日を含む ISO の週は、その週の大半の日が前年に属するため、前年の最後の週になります。
- 1 月 1 日が月曜日、火曜日、水曜日または木曜日の場合は、1 月 1 日を含む ISO の週は、その週の大半の日が新しい年に属するため、新しい年の最初の週になります。

ISO 規格をサポートするために、Oracle には IW 日付書式要素が用意されています。この書式要素は ISO 週番号を戻します。

表 3-6 に、暦年の第 1 週に 1 月 1 日を含めて 4 日以上含まれている場合の例を示します。1 月 1 日を含む週は、1998 年の ISO の第 1 週となります。

表 3-6 年の ISO の第 1 週 : 例 1、1998 年 1 月

月	火	水	木	金	土	日	ISO 週
-	-	-	1	2	3	4	1998 年の ISO の第 1 週
5	6	7	8	9	10	11	1998 年の ISO の第 2 週
12	13	14	15	16	17	18	1998 年の ISO の第 3 週
19	20	21	22	23	24	25	1998 年の ISO の第 4 週
26	27	28	29	30	31	-	1998 年の ISO の第 5 週

表 3-7 に、暦年の第 1 週に含まれる日付が 1 月 1 日を含めて 3 日以内の場合の例を示します。1 月 1 日を含む週は 1998 年の ISO の第 53 週となり、次の週は 1999 年の ISO の第 1 週となります。

表 3-7 年の ISO の第 1 週 : 例 2、1999 年 1 月

月	火	水	木	金	土	日	ISO 週
-	-	-	-	1	2	3	1998 年の ISO の第 53 週
4	5	6	7	8	9	10	1999 年の ISO の第 1 週
11	12	13	14	15	16	17	1999 年の ISO の第 2 週
18	19	20	21	22	23	24	1999 年の ISO の第 3 週
25	26	27	28	29	30	31	1999 年の ISO の第 4 週

年の最初の暦週は、NLS_TERRITORY パラメータによって決定されます。

関連項目： 3-13 ページ「NLS_TERRITORY」

1 年の日数と月数

Oracle では、デフォルトのグレゴリオ暦の他に、次の 6 つの暦法をサポートしています。

- Japanese Imperial（日本の元号暦）－月数と日数はグレゴリオ暦と同じですが、年は各元号ごとに始まります。
- ROC Official（台湾暦）－月数と日数はグレゴリオ暦と同じですが、年は台湾の建国年から始まります。
- Persian（ペルシャ暦）－最初の 6 か月の日数はそれぞれ 31 日、次の 5 か月はそれぞれ 30 日、最後の月は 29 日または 30 日（うるう年）です。
- Thai Buddha（タイ仏教暦）－ 仏教のカレンダを使用します。
- Arabic Hijrah（イスラム歴）－ 月数は 12 で、日数は 354 または 355 です。
- English Hijrah（英語版イスラム歴）－ 月数は 12 で、日数は 354 または 355 です。

暦法は、NLS_CALENDAR パラメータで指定します。

関連項目： 3-27 ページ「NLS_CALENDAR」

紀元の年

イスラム暦は、ヒジュラ紀元の年から始まります。

日本の元号暦は、天皇が即位した最初の年から始まります。たとえば、1998 年は平成 10 年になります。ただし、日本では、グレゴリオ暦も広く理解されているので、1998 年を表現するために 98 年と平成 10 年の両方が使用されます。

NLS_CALENDAR

パラメータ・タイプ： 文字列

パラメータの有効範囲： 初期化パラメータ、環境変数および ALTER SESSION

デフォルト値： Gregorian（グレゴリオ暦）

値の範囲： 有効なカレンダ書式名

世界中で様々な暦法が使用されています。NLS_CALENDAR によって、Oracle で使用する暦法が指定されます。

NLS_CALENDAR パラメータには、次の値のいずれかを指定できます。

- Arabic Hijrah（イスラム暦）
- English Hijrah（英語版イスラム暦）
- Gregorian（グレゴリオ暦）
- Japanese Imperial（日本の元号暦）
- Persian（ペルシャ暦）
- ROC Official（台湾暦）
- Thai Buddha（タイ仏教暦）

> **関連項目：** 暦法、そのデフォルトの日付書式および日付表示に使用されるキャラクタ・セットのリストは、付録 A「ロケール・データ」を参照してください。

例 3-17 NLS_CALENDAR='Japanese Imperial'

NLS_CALENDAR を Japanese Imperial に設定します。

```
SQL> ALTER SESSION SET NLS_CALENDAR='JAPANESE IMPERIAL';
```

SELECT 文を入力して SYSDATE を表示します。

```
SELECT SYSDATE FROM dual;
```

出力は次のようになります。

```
SYSDATE
--------------------
平成 14 年 07 月 15 日
```

数値パラメータ

この項の内容は、次のとおりです。

数値書式

データベースでは、数字列を正確に解釈するために、各セッションで使用される数値書式設定の規則を認識している必要があります。たとえば、数値の入力時に小数点文字としてピリオドまたはカンマのどちらを使用するか（234.00 または 234,00）などを認識している必要があります。同じように、アプリケーションでは、数値情報をクライアント側の書式で表示できる必要があります。

表 3-8 に、数値書式の例を示します。

表 3-8 数値書式の例

国名	数値書式
エストニア	1 234 567,89
ドイツ	1.234.567,89
日本	1,234,567.89
英国	1,234,567.89
米国	1,234,567.89

数値書式は NLS_TERRITORY パラメータの設定から導出されますが、NLS_NUMERIC_CHARACTERS パラメータでオーバーライドできます。

関連項目： 3-13 ページ「NLS_TERRITORY」

NLS_NUMERIC_CHARACTERS

パラメータ・タイプ : 文字列

パラメータの有効範囲 : 初期化パラメータ、環境変数および ALTER SESSION

デフォルト値 : 特定地域に対するデフォルトの小数点文字とグループ・セパレータ

値の範囲 : 2 つの有効な数字

このパラメータでは、小数点文字とグループ・セパレータを指定します。グループ・セパレータとは、千や 100 万などを示すために整数グループを区切る文字で、G 数値書式マスクで戻されます。小数点文字は、数値の整数部と小数部を区切ります。NLS_NUMERIC_CHARACTERS を設定すると、NLS_TERRITORY の設定から導出される値がオーバーライドされます。

任意の文字を、小数点またはグループ・セパレータに指定できます。これらの 2 つの文字はシングルバイトで、互いに異なる文字である必要があります。これらの文字に、数字、プラス（+）、ハイフン（ - ）または不等号（<、>）を使用することはできません。どちらかの文字を空白にすることができます。

この文字は次の書式で指定します。

```
NLS_NUMERIC_CHARACTERS = "decimal_character group_separator"
```

例 3-18 NLS_NUMERIC_CHARACTERS の設定

小数点文字をカンマに、グループ・セパレータをピリオドに設定するには、NLS_NUMERIC_CHARACTERS を次のように定義します。

```
ALTER SESSION SET NLS_NUMERIC_CHARACTERS = ",.";
```

どちらの文字もシングルバイトで、相互に異なっています。

SQL 文には、数値リテラルまたはテキスト・リテラルを表す数値を組み込めます。数値リテラルは引用符で囲みません。数値リテラルは SQL 言語構文の一部で、小数点セパレータとして常にドットを使用し、グループ・セパレータは含まれません。テキスト・リテラルは引用符で囲みます。テキスト・リテラルは、必要に応じて、現行の NLS 設定に従って、暗黙的または明示的に数値に変換されます。

SELECT 文を入力します。

```
SELECT TO_CHAR(4000, '9G999D99') FROM dual;
```

出力は次のようになります。

```
TO_CHAR(4
---------
 4.000,00
```

NLS_NUMERIC_CHARACTERS のデフォルト値は次の方法で変更できます。

- 初期化パラメータ・ファイルで NLS_NUMERIC_CHARACTERS の値を変更してから、インスタンスを再起動します。
- このパラメータの値をセッション中に変更するには、ALTER SESSION 文を使用します。

> **関連項目：** ALTER SESSION 文の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。

通貨パラメータ

この項の内容は、次のとおりです。

通貨書式

世界中で様々な通貨書式が使用されています。表 3-9 に、標準的な通貨書式の一部を示します。

表 3-9 通貨書式の例

国名	例
エストニア	1 234,56 kr
ドイツ	1.234,56€
日本	¥1,234.56
英国	£1,234.56
米国	$1,234.56

NLS_CURRENCY

パラメータ・タイプ : 文字列

パラメータの有効範囲 : 初期化パラメータ、環境変数および ALTER SESSION

デフォルト値 : 特定地域に対するデフォルトの各国通貨記号

値の範囲 : 有効な通貨記号文字列

NLS_CURRENCY では、L 数値書式マスクで戻される文字列の各国通貨記号を指定します。NLS_CURRENCY を設定すると、NLS_TERRITORY で暗黙的に定義された設定がオーバーライドされます。

例 3-19 各国通貨記号の表示

サンプル・スキーマである oe スキーマに接続します。

```
SQL> connect oe/oe
Connected.
```

次のような SELECT 文を入力します。

```
SQL> SELECT TO_CHAR(order_total, 'L099G999D99') "total" FROM orders
    WHERE order_id > 2450;
```

出力は次のようになります。

```
total
---------------------
          $078,279.60
          $006,653.40
          $014,087.50
          $010,474.60
          $012,589.00
          $000,129.00
          $003,878.40
          $021,586.20
```

NLS_CURRENCY のデフォルト値は次の方法で変更できます。

- 初期化パラメータ・ファイル内で値を変更してから、インスタンスを再起動します。
- ALTER SESSION 文を使用します。

> **関連項目 :** ALTER SESSION 文の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。

NLS_ISO_CURRENCY

パラメータ・タイプ: 文字列

パラメータの有効範囲: 初期化パラメータ、環境変数および ALTER SESSION

デフォルト値: NLS_TERRITORY から導出

値の範囲: 有効な任意の地域名

NLS_ISO_CURRENCY では、C 数値書式マスクで戻される文字列の ISO 通貨記号を指定します。NLS_ISO_CURRENCY を設定すると、NLS_TERRITORY で暗黙的に定義された値がオーバーライドされます。

各国通貨記号は不明確な場合があります。たとえば、ドル記号（$）は米国ドルを指すこともオーストラリア・ドルを指すこともあります。ISO 仕様には、特定の地域または国に対して固有の通貨記号が定義されています。たとえば、米国ドルに対する ISO 通貨記号は USD です。オーストラリア・ドルの場合は、AUD です。

表 3-10 に、その他の ISO 通貨記号を示します。

表 3-10 ISO 通貨の例

国名	例
エストニア	1 234 567,89 EEK
ドイツ	1.234.567,89 EUR
日本	1,234,567.89 JPY
英国	1,234,567.89 GBP
米国	1,234,567.89 USD

NLS_ISO_CURRENCY の構文は NLS_TERRITORY パラメータの構文と同じで、サポート対象地域すべてが有効値です。

例 3-20 NLS_ISO_CURRENCY の設定

この例では、サンプル・スキーマ内で oe/oe として接続しているとします。

フランス用の ISO 通貨記号を指定するには、NLS_ISO_CURRENCY を次のように設定します。

```
ALTER SESSION SET NLS_ISO_CURRENCY = FRANCE;
```

SELECT 文を入力します。

```
SQL> SELECT TO_CHAR(order_total, 'C099G999D99') "TOTAL" FROM orders
    WHERE customer_id = 146;
```

出力は次のようになります。

```
TOTAL
------------------
EUR017,848.20
EUR027,455.30
EUR029,249.10
EUR013,824.00
EUR000,086.00
```

NLS_ISO_CURRENCY のデフォルト値は次の方法で変更できます。

- 初期化パラメータ・ファイル内で値を変更してから、インスタンスを再起動します。
- ALTER SESSION 文を使用します。

> **関連項目：** ALTER SESSION 文の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。

NLS_DUAL_CURRENCY

パラメータ・タイプ： 文字列

パラメータの有効範囲： 初期化パラメータ、環境変数および ALTER SESSION

デフォルト値： 特定の地域に対するデフォルトの第 2 通貨記号

値の範囲： 有効な任意の名前

NLS_DUAL_CURRENCY を使用すると、NLS_TERRITORY で暗黙的に定義されたデフォルトの第 2 通貨記号がオーバーライドされます。

NLS_DUAL_CURRENCY は、ユーロ移行期間中にユーロ通貨記号をサポートするために導入されました。表 3-11 に、ユーロ記号をサポートしているキャラクタ・セットを示します。

表 3-11 ユーロ記号をサポートしているキャラクタ・セット

キャラクタ・セット名	説明	ユーロ記号のコード値
D8EBCDIC1141	EBCDIC コード・ページ 1141 8 ビット・オーストリア・ドイツ語	0x9F
DK8EBCDIC1142	EBCDIC コード・ページ 1142 8 ビット・デンマーク語	0x5A
S8EBCDIC1143	EBCDIC コード・ページ 1143 8 ビット・スウェーデン語	0x5A
I8EBCDIC1144	EBCDIC コード・ページ 1144 8 ビット・イタリア語	0x9F
F8EBCDIC1147	EBCDIC コード・ページ 1147 8 ビット・フランス語	0x9F
WE8PC858	IBM-PC コード・ページ 858 8 ビット西ヨーロッパ語	0xDF
WE8ISO8859P15	ISO 8859-15 西ヨーロッパ語	0xA4
EE8MSWIN1250	MS Windows コード・ページ 1250 8 ビット東ヨーロッパ語	0x80
CL8MSWIN1251	MS Windows コード・ページ 1251 8 ビット・ラテン / キリル文字	0x88
WE8MSWIN1252	MS Windows コード・ページ 1252 8 ビット西ヨーロッパ語	0x80
EL8MSWIN1253	MS Windows コード・ページ 1253 8 ビット・ラテン / ギリシャ語	0x80
WE8EBCDIC1047E	Latin 1/ オープン・システム 1047	0x9F
WE8EBCDIC1140	EBCDIC コード・ページ 1140 8 ビット西ヨーロッパ語	0x9F
WE8EBCDIC1140C	EBCDIC コード・ページ 1140 クライアント 8 ビット西ヨーロッパ語	0x9F
WE8EBCDIC1145	EBCDIC コード・ページ 1145 8 ビット西ヨーロッパ語	0x9F
WE8EBCDIC1146	EBCDIC コード・ページ 1146 8 ビット西ヨーロッパ語	0x9F
WE8EBCDIC1148	EBCDIC コード・ページ 1148 8 ビット西ヨーロッパ語	0x9F
WE8EBCDIC1148C	EBCDIC コード・ページ 1148 クライアント 8 ビット西ヨーロッパ語	0x9F
EL8ISO8859P7	ISO 8859-7 ラテン / ギリシャ語	0xA4
IW8MSWIN1255	MS Windows コード・ページ 1255 8 ビット・ラテン / ヘブライ語	0x80
AR8MSWIN1256	MS Windows コード・ページ 1256 8 ビット・ラテン / アラビア語	0x80
TR8MSWIN1254	MS Windows コード・ページ 1254 8 ビット・トルコ語	0x80

表 3-11 ユーロ記号をサポートしているキャラクタ・セット（続き）

キャラクタ・セット名	説明	ユーロ記号のコード値
BLT8MSWIN1257	MS Windows コード・ページ 1257 バルト語	0x80
VN8MSWIN1258	MS Windows コード・ページ 1258 8 ビット・ベトナム語	0x80
TH8TISASCII	タイ工業規格 620-2533 - ASCII 8 ビット	0x80
AL32UTF8	Unicode 3.1 UTF-8 ユニバーサル・キャラクタ・セット	E282AC
UTF8	Unicode 3.0 UTF-8 ユニバーサル・キャラクタ・セット	E282AC
AL16UTF16	Unicode 3.1 UTF-16 ユニバーサル・キャラクタ・セット	20AC
UTFE	Unicode 3.0 の UTF-EBCDIC エンコーディング	CA4653
ZHT16HKSCS	MS Windows コード・ページ 950 香港補足キャラクタ・セット付き	0xA3E1
ZHS32GB18030	GB18030-2000	0xA2E3
WE8BS2000E	Siemens EBCDIC.DF.04 8 ビット西ヨーロッパ語	0x9F

Oracle のユーロ・サポート

欧州通貨連合（European Monetary Union: EMU）の加盟国は、2002 年 1 月 1 日から自国通貨としてユーロを使用しています。NLS_TERRITORY を対応する EMU 加盟国（オーストリア、ベルギー、フィンランド、フランス、ドイツ、ギリシャ、アイルランド、イタリア、ルクセンブルグ、オランダ、ポルトガルおよびスペイン）に設定すると、NLS_CURRENCY および NLS_DUAL_CURRENCY のデフォルト値が EUR に設定されます。

移行期間（1999 ～ 2001 年）中は、Oracle8*i* 以上の Oracle に次のユーロ・サポート機能が提供されていました。

- NLS_CURRENCY が各国の 1 次通貨として定義されていました。
- NLS_ISO_CURRENCY が、特定の地域の ISO 通貨コードとして定義されていました。
- NLS_DUAL_CURRENCY が、特定の地域の 2 次通貨記号（通常はユーロ）として定義されていました。

Oracle9*i* リリース 2（9.2）からは、NLS_ISO_CURRENCY の値に従って EMU 加盟国用の ISO 通貨記号が EUR に設定されます。たとえば、NLS_ISO_CURRENCY が FRANCE に設定されているとします。次の SELECT 文を入力します。

```
SELECT TO_CHAR(TOTAL, 'C099G999D99') "TOTAL" FROM orders WHERE customer_id=585;
```

出力は次のようになります。

```
TOTAL
-------
EUR12.673,49
```

廃止になった各国通貨記号を引き続き使用する必要がある場合は、それをサーバー上では初期化ファイルのパラメータとして、クライアント上では環境変数として定義することで、NLS_DUAL_CURRENCY または NLS_CURRENCY のデフォルトをオーバーライドできます。

> **注意：** NLS_CURRENCY または NLS_DUAL_CURRENCY を有効にするには、クライアント上で NLS_LANG も設定する必要があります。

NLS_ISO_CURRENCY の値から導出される ISO 通貨記号はオーバーライドできません。

NLS_MONETARY_CHARACTERS

パラメータ・タイプ： 文字列

パラメータの有効範囲： 環境変数

デフォルト値： NLS_TERRITORY から導出

値の範囲： 有効な任意の名前

NLS_MONETARY_CHARACTERS では、通貨式で数値グループを区切る文字を指定します。たとえば、地域が米国の場合、3 桁のセパレータはカンマで、小数点セパレータはピリオドです。

NLS_CREDIT

パラメータ・タイプ： 文字列

パラメータの有効範囲： 環境変数

デフォルト値： NLS_TERRITORY から導出

値の範囲： 最大 9 バイト（NULL を含まない）の任意の文字列

NLS_CREDIT は、財務レポートで貸方を表示する記号を設定します。このパラメータのデフォルト値は、NLS_TERRITORY によって決定されます。たとえば、空白は NLS_CREDIT の有効な値です。

このパラメータが指定できるのは、クライアント環境のみです。

このパラメータは、OCIGetNlsInfo() 関数を使用して取得できます。

NLS_DEBIT

パラメータ・タイプ: 文字列

パラメータの有効範囲: 環境変数

デフォルト値: NLS_TERRITORY から導出

値の範囲: 最大 9 バイト（NULL を含まない）の任意の文字列

NLS_DEBIT は、財務レポートで借方を表示する記号を設定します。このパラメータのデフォルト値は、NLS_TERRITORY によって決定されます。たとえば、マイナス符号（-）は、NLS_DEBIT の有効な値です。

このパラメータが指定できるのは、クライアント環境のみです。

このパラメータは、OCIGetNlsInfo() 関数を使用して取得できます。

言語ソート・パラメータ

言語ソート・パラメータを使用して、データのソート方法を選択できます。

この項の内容は、次のとおりです。

- NLS_SORT
- NLS_COMP
- NLS_LIST_SEPARATOR

関連項目: 第 4 章「言語ソート」

NLS_SORT

パラメータ・タイプ: 文字列

パラメータの有効範囲: 初期化パラメータ、環境変数および ALTER SESSION

デフォルト値: 特定の言語に対するデフォルトの文字のソート基準

値の範囲: BINARY または有効な言語定義名

NLS_SORT は、文字データのソート・タイプを指定し、NLS_LANGUAGE で暗黙的に定義された値をオーバーライドします。

NLS_SORT の構文は、次のとおりです。

```
NLS_SORT = BINARY | sort_name
```

BINARY はバイナリ・ソートを指定し、*sort_name* は言語ソート基準を指定します。

例 3-21 NLS_SORT の設定

German という言語ソート基準を指定するには、NLS_SORT を次のように設定します。

```
NLS_SORT = German
```

言語ソート基準に付ける名前は、言語名と直接の関係はありません。ただし、通常、各サポート対象言語には、同じ名前を使用した適切な言語ソート基準があります。Oracle では、単一言語ソートと多言語ソートの 2 種類の言語ソートを提供しています。また、単一言語ソートを拡張することにより特殊な事例に対応できます。拡張単一言語ソートでは、通常、文字はその ASCII 値と異なる方法でソートされます。たとえば、拡張スペイン語ソートである XSPANISH では、ch と ll が 1 つの文字として取り扱われます。つまり、XSPANISH が従来のスペイン語のソート・ルールを使用する一方で、SPANISH ソートは現代的なスペイン語の照合規則を使用します。

注意： NLS_SORT パラメータを BINARY に設定すると、オプティマイザは、索引スキャンを選択することで、ソートを行わずに ORDER BY 句を実行する場合があります。

NLS_SORT を言語ソートに設定すると、NLS_SORT で指定された言語ソート用の言語索引が存在しない場合は、ORDER BY 句を実行するためにソートが必要になります。

NLS_SORT で指定された言語ソートに対応する言語索引が存在する場合、オプティマイザは、索引スキャンを選択することで、ソートを行わずに ORDER BY 句を実行する場合があります。

NLS_SORT のデフォルト値は、次の方法で変更できます。

- 初期化パラメータ・ファイル内で値を変更してから、インスタンスを再起動します。
- ALTER SESSION 文を使用します。

関連項目：

- 4-4 ページ「多言語ソート」
- ALTER SESSION 文の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。
- 言語ソート定義の詳細リストは、付録 A「ロケール・データ」を参照してください。

NLS_COMP

パラメータ・タイプ: 文字列

パラメータの有効範囲: 初期化パラメータ、環境変数および ALTER SESSION

デフォルト値: バイナリ

値の範囲: BINARY または ANSI

NLS_COMP を使用すると、SQL 文に NLS_SORT を使用する複雑な処理を回避できます。通常、WHERE 句および PL/SQL ブロックでの比較は、バイナリで行われます。言語上の比較を使用するには、NLSSORT SQL 関数を使用する必要があります。この関数の使用は、言語ソートがすでに NLS_SORT セッション・パラメータに指定されている場合など、特に面倒な場合があります。NLS_COMP を使用すると、比較を NLS_SORT セッション・パラメータに従って言語的に行うように指定できます。このためには、セッションを次のように変更します。

```
ALTER SESSION SET NLS_COMP = ANSI;
```

WHERE 句での比較が必ずバイナリで行われるように指定するには、次の文を発行します。

```
ALTER SESSION SET NLS_COMP = BINARY;
```

NLS_COMP を ANSI に設定すると、言語索引によって、言語上の比較のパフォーマンスが向上します。

言語索引を使用可能にするには、次の構文を使用します。

```
CREATE INDEX i ON t(NLSSORT(col, 'NLS_SORT=FRENCH'));
```

関連項目: 4-12 ページ「言語索引の使用」

NLS_LIST_SEPARATOR

パラメータ・タイプ: 文字列

パラメータの有効範囲: 環境変数

デフォルト値: NLS_TERRITORY から導出

値の範囲: 有効な任意の文字

NLS_LIST_SEPARATOR は、値リストの値を区切るために使用する文字を指定します。このパラメータのデフォルト値は、NLS_TERRITORY の値から導出されます。

指定する文字はシングルバイトにする必要があります。数値の小数点文字、通貨の小数点文字、数字、プラス（+）、ハイフン（ - ）、不等号（<、>）またはピリオド（.）は、いずれも使用できません。

キャラクタ・セット変換パラメータ

この項の内容は、次のとおりです。

- NLS_NCHAR_CONV_EXCP

NLS_NCHAR_CONV_EXCP

パラメータ・タイプ: 文字列

パラメータの有効範囲: 初期化パラメータ、ALTER SESSION、ALTER SYSTEM

デフォルト値: FALSE

値の範囲: TRUE、FALSE

NLS_NCHAR_CONV_EXCP は、暗黙的または明示的なキャラクタ・タイプの変換時にデータが消失した場合に、エラーをレポートするかどうかを決定します。デフォルト値の場合、エラーはレポートされません。

関連項目: キャラクタ・セット変換中のデータ消失の詳細は、第 10 章「キャラクタ・セットの移行」を参照してください。

長さセマンティクス

この項の内容は、次のとおりです。

- NLS_LENGTH_SEMANTICS

NLS_LENGTH_SEMANTICS

パラメータ・タイプ: 文字列

パラメータの有効範囲: 動的、初期化パラメータ、ALTER SESSION および ALTER SYSTEM

デフォルト値: BYTE

値の範囲: BYTE | CHAR

デフォルトでは、文字データ型 CHAR および VARCHAR2 は、文字単位ではなくバイト単位で指定します。したがって、表定義で CHAR(20) と指定すると、文字データを格納するために 20 バイトが割り当てられます。

データベース・キャラクタ・セットでシングルバイト文字コード体系が使用されている場合は、文字数がバイト数と同じであるため、このような指定は適切に処理されます。データベース・キャラクタ・セットでマルチバイト文字コード体系が使用されている場合は、1 文

字が 1 バイト以上で構成される場合があるため、バイト数は文字数とは異なります。したがって、特定の文字数に対して予測される最大バイト数を見込んで、列幅を慎重に選択する必要があります。この問題は、列サイズの定義時にキャラクタ・セマンティクスに切り替えることで回避できます。

NLS_LENGTH_SEMANTICS によって、CHAR、VARCHAR2 および LONG の各列をバイト・セマンティクスまたはキャラクタ・セマンティクスのいずれかを使用して作成できます。NCHAR、NVARCHAR2、CLOB および NCLOB の各列は、常に文字ベースです。既存の列は、影響を受けません。

既存のアプリケーションとの互換性を維持するためには、バイト・セマンティクスを使用する必要があります。

NLS_LENGTH_SEMANTICS は、SYS および SYSTEM の表には適用されません。データ・ディクショナリでは、常にバイト・セマンティクスが使用されます。

関連項目：

- 2-11 ページ「長さセマンティクス」
- 長さセマンティクスの詳細は、『Oracle9*i* データベース概要』を参照してください。

4

言語ソート

この章では、Oracle 環境での文字のソート方法について説明します。この章の内容は、次のとおりです。

- Oracle のソート機能の概要
- バイナリ・ソートの使用
- 言語ソートの使用
- 言語ソート機能
- 言語索引の使用
- 関数索引を使用した大 / 小文字区別なしの検索パフォーマンスの改善
- 汎用ベース文字検索の実行

Oracle のソート機能の概要

ソート順序は言語によって異なります。さらに、同じアルファベットを使用している文化または国の間でも、ワードのソート方法が異なる場合があります。たとえば、デンマーク語の文字 Æ は、Z の後にきます。また、Y と Ü は同じ文字の変形とみなされます。

ソート順序には、大 / 小文字区別の有無を指定できます。**ケース**は、大文字であるか小文字であるかの条件を指します。たとえば、ラテン・アルファベットの A は、小文字の絵文字 a の大文字です。

ソート順序では、発音区別記号を無視するか考慮するかを指定できます。**発音区別記号**は、文字または文字列の上または下にある記号で、それが付いていない場合の文字とは発音が異なることを示します。たとえば、façade の場合、セディラ（,）は発音区別記号です。セディラが付いている場合は、c の発音が変化します。

表音的なソート順序や、文字の外観に基づいたソート順序も指定できます。たとえば、東南アジアの表意文字の場合は、画数に基づいたソート順序を指定できます。ソート上の一般的な問題となるものに、結合文字があります。たとえば、伝統的なスペイン語の ch は 1 つの独立した文字であり、ソート順序では c の後にきます。つまり、正しいソート順序は、cerveza、colorado、cheremoya のようになります。したがって、文字 c は、次にくる文字が h であるかどうかが確認されるまで、ソートされません。

Oracle には、次のタイプのソートが用意されています。

- バイナリ・ソート
- 単一言語ソート
- 多言語ソート

単一言語に応じた正確なソートに加えて、多言語 ISO 規格（ISO-14651）に準拠したソートが実現できます。この規格は、複数言語を同時に処理するように設計されています。

バイナリ・ソートの使用

文字データをソートする方法の 1 つは、文字コード体系によって定義された文字の数値に基づいています。このようなソートを**バイナリ・ソート**と呼びます。バイナリ・ソートは最も高速なソート・タイプであり、ASCII や EBCDIC の規格では、文字 A ～ Z を昇順の数値で定義しているため、英語のアルファベットについては正しいソート結果が得られます。

注意： ASCII 規格では、大文字はすべて小文字の前にきます。逆に、EBCDIC 規格では、小文字はすべて大文字の前にきます。

他の言語で使用されている文字が存在すると、通常、バイナリ・ソートでは正しい結果が得られません。たとえば、文字コード体系で Ä の数値が B の数値より高い場合、昇順の ORDER BY 問合せでは、ABC、ABZ、BCD、ÄBC の順で文字列が戻ります。表意文字を使用するアジア言語の場合、通常、バイナリ・ソートに言語的な意味はありません。

言語ソートの使用

文字のアルファベット順に一致するソート基準を得るには、文字コード体系内の数値に依存せずに文字をソートする別のソート方法を使用する必要があります。この方法を**言語ソート**と呼びます。言語ソートでは、各文字を、言語的に適切な文字順序を反映した数値に置換することによって、ソート操作を行います。

Oracle では、単一言語ソートと多言語ソートの 2 種類の言語ソートを提供しています。

この項の内容は、次のとおりです。

単一言語ソート

単一言語ソートの場合、文字列は 2 つの手順で比較されます。最初の手順では、メジャー値テーブルにある文字列全体のメジャー値が比較されます。通常、同じ外観の文字には、同じメジャー値があります。第 2 の手順では、マイナー値テーブルにあるマイナー値が比較されます。メジャー値とマイナー値は Oracle によって定義されます。Oracle では、マイナー値が異なる同じメジャー値を使用して、発音区別記号を持つ文字と大 / 小文字区別を定義します。

各メジャー・テーブル・エントリには、1 文字の **Unicode コード・ポイント**とメジャー値が含まれています。Unicode コード・ポイントは、1 文字を表す 16 ビットのバイナリ値です。

表 4-1 に、a、A、ä、Ä および b のソートに使用するサンプルの値を示します。

表 4-1 文字のサンプルとそのソートのメジャー値とマイナー値

絵文字	メジャー値	マイナー値
a	15	5
A	15	10
ä	15	15
Ä	15	20
b	20	5

関連項目： 5-2 ページ「Unicode の概要」

多言語ソート

Oracle9*i* には多言語ソートが用意されているため、複数言語のデータを 1 つのソートとしてソートできます。この機能は、複雑なソート・ルールや多言語データベースを持つ地域または言語に有効です。さらに、Oracle9*i* では、従来のリリースで定義されたすべてのソート順序をサポートしています。

アジア諸国の言語データや多言語データに対しては、ISO 14651 規格と Unicode 3.1 規格に基づいたソート・メカニズムが用意されています。漢字は、画数、ピンイン（中国語の発音記号）または部首で順序付けされます。

また、多言語ソートでは、標準的な同値化や補助文字も処理できます。**標準的な同値化**は、文字間または文字列間での基本的な同値化です。たとえば、ç は c と , の組合せと同じです。**補助文字**は Unicode 3.1 でのユーザー定義文字または事前定義済みの文字であり、特定のコード範囲内の 2 つのコード・ポイントを必要とします。1 つの多言語ソートに最大 110 万のコード・ポイントを定義できます。

たとえば、Oracle9*i* ではフランス語の単一言語ソート（FRENCH）がサポートされていますが、フランス語の多言語ソート（FRENCH_M）を指定できます。_M は、多言語ソートに対する ISO 14651 規格を表します。このソート順では、GENERIC_M ソート順に基づき、発音区別記号を右から左へとソートできます。したがって、表に多言語データが格納されている場合は、多言語ソートの使用をお薦めします。表にフランス語のみが含まれている場合は、フランス語の単一言語ソートを使用する方が、メモリー使用量が少ないため、パフォーマンスは向上します。メモリー使用量が少ないのは、フランス語の単一言語ソートの方が、フランス語の多言語ソートよりも定義されている文字が少ないためです。ソートの有効範囲とパフォーマンスの間には、トレードオフがあります。

関連項目：

- 4-10 ページ「標準的な同値化」
- 5-3 ページ「補助文字」

多言語ソート・レベル

Oracle では、多言語ソートを次の 3 つの精度レベルで評価します。

- 1 次レベル・ソート
- 2 次レベル・ソート
- 3 次レベル・ソート

1 次レベル・ソート

1 次レベル・ソートでは、文字 a と文字 b の相違など、**ベース文字**間の相違を識別します。a が b の前にくるか、b が a の前にくるか、あるいは同値かの定義は、個々のロケールに準じます。文字をバイナリで表現することは、意味のないことです。無視可能文字には、0（ゼロ）の 1 次レベルの**順序**（重み）を割り当てます。その結果、その文字は 1 次レベルで無視されます。他のレベルでの無視可能文字には、そのレベルで順序 0（ゼロ）が割り当てられます。

たとえば、1 次レベルでは、bat のすべてのバリエーションが、bet のすべてのバリエーションより前にきます。どちらの場合も、それぞれのバリエーションは次のように様々な順序で表示されます。

```
Bat
bat
BAT
BET
Bet
bet
```

関連項目： 4-9 ページ「無視可能文字」

2 次レベル・ソート

2 次レベル・ソートでは、ベース文字（1 次レベル・ソート）を識別してから、特定のベース文字に付いている様々な発音区別記号を識別します。たとえば、文字 Ä と文字 A の相違は、発音区別記号の有無のみです。したがって、Ä と A は 2 次レベルでは異なる文字ですが、1 次レベルでは同じ文字です。これは、両方とも同じベース文字（A）から導出されているためです。

次のリストは、1 次レベル（resume が resumes の前）と 2 次レベル（発音区別記号なしの文字列が発音区別記号付きの文字列の前）でソートされています。

```
resume
résumé
Résumé
Resumes
resumes
résumés
```

3 次レベル・ソート

3 次レベル・ソートでは、ベース文字（1 次レベル・ソート）、発音区別記号（2 次レベル・ソート）およびケース（大文字と小文字）が識別されます。さらに、+、- および * などの特殊文字も識別できます。

次に 3 次レベル・ソートの例を示します。

- 文字 a と A は、1 次レベルと 2 次レベルでは同じ文字ですが、3 次レベルでは異なる文字です。これは、ケースが異なるためです。
- 文字 ä と A は、1 次レベルでは同じ文字ですが、2 次レベルと 3 次レベルでは異なる文字です。
- ダッシュ文字 - の 1 次レベルと 2 次レベルでの順序は、0（ゼロ）です。つまり、この文字は、1 次と 2 次のレベルでは無視されます。ダッシュを 0（ゼロ）以外の 1 次レベル順序を持つ別の文字、たとえば、u と比較しても、1 次レベルでの結果は取得できません。これは、u と比較する対象の文字がないためです。この場合は、3 次レベルでのみ - と u の相違が検索されます。

次のリストは、1 次レベル（resume が resumes の前）、2 次レベル（発音区別記号なしの文字列が発音区別記号付きの文字列の前）および 3 次レベル（小文字が大文字の前）でソートされています。

```
resume
Resume
résumé
Résumé
resumes
résumés
Resumes
Résumés
```

言語ソートの例

この項の例は、バイナリ・ソート、単一言語ソートおよび多言語ソートを示しています。例を使用する準備として、表 test を作成して移入します。次の文を入力します。

```
SQL> CREATE TABLE test (name VARCHAR2(20));
SQL> INSERT INTO test VALUES('Diet');
SQL> INSERT INTO test VALUES('À voir');
SQL> INSERT INTO test VALUES('Freizeit');
```

例 4-1 バイナリ・ソート

ORDER BY 句でバイナリ・ソートを使用します。

```
SQL> SELECT * FROM test ORDER BY name;
```

次の出力が表示されます。

```
Diet
Freizeit
À voir
```

バイナリ・ソートの結果、À voir がリストの最後に表示されることに注意してください。

例 4-2 ドイツ語の単一言語ソート

NLS_SORT パラメータを german に設定して NLSSORT 関数を使用し、ドイツ語ソートを取得します。

```
SQL> SELECT * FROM test ORDER BY NLSSORT(name, 'NLS_SORT=german');
```

次の出力が表示されます。

```
À voir
Diet
Freizeit
```

ドイツ語ソートでは、À voir がリストの先頭になることに注意してください。

例 4-3 ドイツ語の単一言語ソートと多言語ソートの比較

図 4-1 に示した文字列を test に挿入します。この場合、横棒付きの D の後に ñ がきます。

図 4-1 文字列

Ðñ

NLS_SORT パラメータを german に設定して NLSSORT 関数を使用し、ドイツ語の単一言語ソートを実行します。

```
SQL> SELECT * FROM test ORDER BY NLSSORT(name, 'NLS_SORT=german');
```

ドイツ語ソートからの出力では、新しい文字列がエントリ・リストの最後に表示されます。これは、その文字がドイツ語ソートでは認識されないためです。

次の文を入力して多言語ソートを実行します。

```
SQL> SELECT * FROM test ORDER BY NLSSORT(name, 'NLS_SORT=generic_m');
```

出力では、ISO ソート・ルールに従って新しい文字列が Diet の後に表示されます。

関連項目：

- 7-10 ページ「NLSSORT 関数」
- NLS_SORT パラメータの設定と変更の詳細は、3-38 ページの「NLS_SORT」を参照してください。

言語ソート機能

この項では、次の言語ソート機能の違いについて説明します。

- ベース文字
- 無視可能文字
- 短縮文字
- 拡張文字
- 状況依存文字
- 標準的な同値化
- 逆 2 次ソート
- タイ語 / ラオ語文字に対する文字の再配列
- 特殊文字
- 特殊組合せ文字
- 特殊な大文字
- 特殊な小文字

言語ソートは、必要な特性を含むようにカスタマイズできます。

関連項目： 第 12 章「ロケール・データのカスタマイズ」

ベース文字

ベース文字はベース文字表に定義されています。この表によって、各文字がベース文字にマップされます。たとえば、a、A、ä および Ä はすべて、**ベース文字**である a にマップされます。この概念は、Oracle Text で作業する場合に特に重要です。

関連項目：『Oracle Text リファレンス』

無視可能文字

言語ソートでは、一部の文字を無視できます。このような文字を**無視可能文字**と呼びます。無視可能文字には、発音区別記号と句読点の 2 種類があります。

無視可能な発音区別記号の例は、次のとおりです。

- ^。したがって、rôle は role と同じ文字として処理されます。
- ウムラウト。したがって、naïve は naive と同じ文字として処理されます。

無視可能な句読点の例として、ダッシュ文字 - があります。この文字が無視される場合は、multi-lingual を multilingual と同じ文字、e-mail を email と同じ文字として処理できます。

短縮文字

ソート要素は、通常、単一文字で構成されていますが、一部のロケールでは、1 つの文字列に 2 つ以上の文字がある場合があります。その場合、その文字列はソート時にも単一のソート要素とみなす必要があります。たとえば、伝統的なスペイン語の文字列 ch は、2 つの文字で構成されています。これらの文字は、多言語ソートでは**短縮文字**と呼ばれ、単一言語ソートでは、**特殊組合せ文字**と呼ばれます。

合成文字を短縮文字と混同しないでください。á などの合成文字は、それぞれ独自のエンコーディングを持つ a と ' に分解できます。合成文字と短縮文字の違いは、合成文字が端末に 1 文字として表示できるのに対して、短縮文字はソートにのみ使用され、それを構成する文字はそれぞれ個別に表示される必要があることです。

拡張文字

一部のロケールでは、特定の文字を文字列であるかのようにソートする必要があります。ドイツ語の文字 ß（強調の s）がその例です。この文字は、文字列 ss と同じようにソートされます。別の例では、ö は、od の後、of の前に oe のようにソートされます。これらの文字は、多言語ソートでは、**拡張文字**と呼ばれ、単一言語ソートでは、**特殊文字**と呼ばれます。短縮文字の場合と同様、拡張文字に対する置換文字列は、ソート目的の場合のみ意味があります。

状況依存文字

日本語では、全角のダッシュ ー に似た長母音記号は、先行する文字の母音を長く伸ばすことを示す長音記号を表しています。ソート順序は、長音記号の前にある母音に応じて異なります。この方法は、状況依存ソートと呼ばれます。たとえば、文字 ka の後にくる長音記号ー は、a を長く伸ばすことを示し、a と同じように処理されます。一方、文字 ki の後にくる長音記号ー は、i を長く伸ばすことを示し、i と同じように処理されます。これをラテン文字に変換すると、ソートは次のようになります。

```
kaa
ka—   -- kaa and ka— are the same
kai   -- kai follows ka- because i is after a
kia   -- kia follows kai because i is after a
kii   -- kii follows kia because i is after a
ki—   -- kii and ki— are the same
```

標準的な同値化

1 つの Unicode のコード・ポイントが、ロケールに関係なく、一連のベース文字のコード・ポイントに発音区別記号のコード・ポイントを加えたものと同じ値である場合があります。これは、Unicode の標準的な同値化と呼ばれます。たとえば、ä は、ウムラウト付きのベース文字 a と同値です。言語フラグ CANONICAL_EQUIVALENCE=TRUE は、Unicode 3.1 に定義されている標準的な同値化規則をすべて適用する必要があることを示します。すべてのデータが合成された形式の場合、このフラグを FALSE に変更すると、比較と順序付け機能が高速になります。

> **関連項目：** 標準的な同値化フラグの設定の詳細は、12-35 ページの「Oracle Locale Builder を使用した新規言語ソートの作成」を参照してください。

逆 2 次ソート

フランス語では、発音区別記号付き文字を含む文字列をソートする場合、最初にベース文字が左から右の順に比較されますが、発音区別記号付き文字自体は右から左の順に比較されます。たとえば、デフォルトでは、発音区別記号付き文字は、発音区別記号が付かない文字の後に置かれます。そのため、フランス語ソートでは、Èdit は Edít の前にきます。この 2 つの文字列は 1 次レベルでは同値です。2 次レベルの順序は、発音区別記号付き文字を右から左へ調べてから決定されます。個別のロケールでは、発音区別記号付き文字を右から左のルールでソートするように要求できます。逆 2 次ソートを使用可能にするには、REVERSE_SECONDARY 言語フラグを TRUE に設定します。

> **関連項目：** 逆 2 次フラグの設定の詳細は、12-35 ページの「Oracle Locale Builder を使用した新規言語ソートの作成」を参照してください。

タイ語 / ラオ語文字に対する文字の再配列

タイ語とラオ語の場合、ソート前に一部の文字を後続の文字と最初に入れ替える必要があります。通常、この種の文字は、母音を表す記号であるため、次にくる文字は子音になります。子音と母音は、ソート前に入れ替える必要があります。ソート前に入れ替える必要のあるすべての文字について、SWAP_WITH_NEXT 言語フラグを設定してください。

関連項目： SWAP_WITH_NEXT フラグの設定の詳細は、12-35 ページの「Oracle Locale Builder を使用した新規言語ソートの作成」を参照してください。

特殊文字

特殊文字とは、単一言語ソートで使用する用語です。この文字は、多言語ソートでは、**拡張文字**と呼ばれます。

関連項目： 4-9 ページ「拡張文字」

特殊組合せ文字

特殊組合せ文字とは、単一言語ソートで使用する用語です。この文字は、多言語ソートでは、**短縮文字**と呼ばれます。

関連項目： 4-9 ページ「短縮文字」

特殊な大文字

1 つの小文字が複数の大文字にマップされる場合があります。たとえば、伝統的なドイツ語では、ß の大文字は、SS です。

このような大 / 小文字の変換は、NLS_UPPER、NLS_LOWER および NLS_INITCAP の各 SQL 関数によって、言語ソート基準で設定された規則に従って処理されます。SQL 関数 UPPER、LOWER および INITCAP では、これらの特殊文字を処理できません。

NLS_UPPER SQL 関数は、小文字の文字列と同じキャラクタ・セットからの大文字をすべて戻します。次の例に、NLS_SORT が XGERMAN に設定されている場合の NLS_UPPER 関数の結果を示します。

```
SELECT NLS_UPPER ('große') "Uppercase" FROM DUAL;

Upper
-----
GROSSE
```

関連項目： 『Oracle9*i* SQL リファレンス』

特殊な小文字

Oracle は、特殊な小文字をサポートしています。1 つの大文字が複数の小文字にマップされる場合があります。たとえば、トルコ語の大文字 I は、ı(小文字のドットなしの i) になります。

言語索引の使用

言語ソートは言語固有であるため、バイナリ・ソートよりもデータ処理が増えます。ASCII 文字のバイナリ・コードには文字の順序が反映されているため、言語 ASCII のバイナリ・ソートを使用すると正確で高速です。複数言語のデータがデータベースに格納されている場合、アプリケーションでは、言語に応じて異なるソート基準に従い、SELECT...ORDER BY 文から戻されたデータをソートできます。このようなソートは、言語索引を使用すると、パフォーマンスを低下させずに実現できます。列の言語索引によって、挿入と更新の操作速度が低下しますが、ORDER BY 句を使用した言語ソートのパフォーマンスは大幅に向上します。

英語以外の言語を使用する関数索引を作成できます。この索引は、NLS_SORT で決められている言語ソート順序を変更しません。単にパフォーマンスを向上させるためのものです。次の文では、ドイツ語のソートに基づく索引が作成されます。

```
CREATE TABLE my_table(name VARCHAR(20) NOT NULL)
/*NOT NULL ensures that the index will be used */
CREATE INDEX nls_index ON my_table (NLSSORT(name, 'NLS_SORT = German'));
```

索引の作成後に、次のような SELECT 文を入力します。

```
SELECT * FROM my_table ORDER BY name;
```

この文は、索引を使用しない場合の同じ SELECT 文より速く結果が戻されます。

これ以降の内容は、次のとおりです。

関連項目：

- 『Oracle9*i* データベース概要』
- 関数索引の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。

複数言語の言語索引

複数言語のデータに言語索引を作成する方法には、次の 3 通りがあります。

- アプリケーションでサポートされている各言語の言語索引を作成します。このアプローチは単純ですが、大量のディスク領域が必要になります。索引ごとに、その索引が作成されている言語以外の言語の複数行が、順番の最後にまとめて照合されます。次の例では、フランス語とドイツ語の言語索引を作成しています。

  ```
  CREATE INDEX french_index ON employees (NLSSORT(employee_id, 'NLS_
  SORT=FRENCH'));
  CREATE INDEX german_index ON employees (NLSSORT(employee_id, 'NLS_
  SORT=GERMAN'));
  ```

  使用する索引は、ORDER BY 句で指定した NLS_SORT セッション・パラメータまたは NLSSORT 関数の引数によって決定されます。たとえば、NLS_SORT セッション・パラメータが FRENCH に設定されている場合、Oracle では french_index が使用されます。GERMAN に設定されている場合は、german_index が使用されます。

- すべての言語に対して単一言語索引を作成します。そのためには、NLSSORT 関数のパラメータとして言語列（4-15 ページの「例 : フランス語の言語索引の設定」で LANG_COL の）を使用する必要があります。言語列には、索引の作成対象となる列のデータの NLS_LANGUAGE 値が含まれます。次の例では、複数言語に対する単一言語索引を作成しています。この索引では、NLS_LANGUAGE に対して同じ値を持つ行がまとめてソートされます。

  ```
  CREATE INDEX i ON t (NLSSORT(col, 'NLS_SORT=' || LANG_COL));
  ```

  問合せでは、ORDER BY 句で指定した NLSSORT 関数の引数に基づいて索引が選択されます。

- GENERIC_M や FRENCH_M などの多言語ソートのいずれかを使用して、すべての言語に単一言語索引を作成します。この索引は、ISO 14651 に定義されている規則に従って文字をソートします。次に例を示します。

  ```
  CREATE INDEX i on t (NLSSORT(col, 'NLS_SORT=GENERIC_M');
  ```

  関連項目： Unicode ソートの詳細は、4-4 ページの「多言語ソート」を参照してください。

言語索引の使用要件

言語索引を使用するための要件は、次のとおりです。

- QUERY_REWRITE_ENABLED を TRUE に設定
- NLS_COMP を ANSI に設定
- NLS_SORT を適切に設定
- オプティマイザ・モードが FIRST_ROWS に設定されているコストベース・オプティマイザの使用

この項では、次の例についても説明します。

- 例 : フランス語の言語索引の設定

QUERY_REWRITE_ENABLED を TRUE に設定

QUERY_REWRITE_ENABLED 初期化パラメータを TRUE に設定する必要があります。これは、すべての関数索引に必須です。ALTER SESSION 文を使用すると、QUERY_REWRITE_ENABLED を TRUE に設定できます。次に例を示します。

```
ALTER SESSION SET QUERY_REWRITE_ENABLED=TRUE;
```

関連項目： QUERY_REWRITE_ENABLED 初期化パラメータの詳細は、『Oracle9*i* データベース・リファレンス』を参照してください。

NLS_COMP を ANSI に設定

NLS_COMP パラメータを ANSI に設定する必要があります。NLS_COMP を設定するには、複数の方法があります。次に例を示します。

```
ALTER SESSION SET NLS_COMP = ANSI;
```

関連項目： 3-40 ページ「NLS_COMP」

NLS_SORT を適切に設定

NLS_SORT パラメータで、言語ソートに使用する言語定義を指定する必要があります。たとえば、フランス語の言語ソート順序を使用する場合は、NLS_SORT を FRENCH に設定する必要があります。また、ドイツ語の言語ソート順序を使用する場合は、NLS_SORT を GERMAN に設定する必要があります。

NLS_SORT を設定するには、複数の方法があります。すべての言語に同じ SQL 文を使用できるように、NLS_SORT をクライアントの環境変数として設定する必要があります。クライアント環境で NLS_SORT を設定すると、様々な言語索引を使用できます。

関連項目： 3-38 ページ「NLS_SORT」

オプティマイザ・モードが FIRST_ROWS に設定されているコストベース・オプティマイザの使用

オプティマイザ・モードが FIRST_ROWS に設定されているコストベース・オプティマイザを使用します。これは、言語索引がルールベースのオプティマイザでは認識されないためです。次に、オプティマイザ・モードの設定例を示します。

```
ALTER SESSION SET OPTIMIZER_MODE = FIRST_ROWS;
```

> **関連項目：** コストベース・オプティマイザの詳細は、『Oracle9*i* パフォーマンス・ガイドおよびリファレンス』を参照してください。

例 : フランス語の言語索引の設定

次の例では、フランス語の言語索引の設定方法を示します。ALTER SESSION 文を使用するかわりに、NLS_SORT をクライアントの環境変数として設定することもできます。

```
ALTER SESSION SET QUERY_REWRITE_ENABLED=TRUE;
ALTER SESSION SET NLS_COMP = ANSI;
ALTER SESSION SET NLS_SORT='FRENCH';
ALTER SESSION SET OPTIMIZER_MODE = FIRST_ROWS;
CREATE INDEX test_idx ON test(NLSSORT(col, 'NLS_SORT=FRENCH'));
SELECT * FROM test ORDER BY col;
SELECT * FROM test WHERE col > 'JJJ';
```

関数索引を使用した大 / 小文字区別なしの検索パフォーマンスの改善

関数索引を作成すると、大 / 小文字を区別しない検索のパフォーマンスが向上します。次に例を示します。

```
CREATE INDEX case_insensitive_ind ON employees(NLS_UPPER(first_name));
SELECT * FROM employees WHERE NLS_UPPER(first_name) = 'KARL';
```

汎用ベース文字検索の実行

ケースと発音区別記号を無視する検索を実行できます。次の文を入力します。

```
ALTER SESSION SET NLS_COMP=ANSI;
ALTER SESSION SET NLS_SORT=GENERIC_BASELETTER;
```

次のような文を入力します。

```
SELECT * FROM emp WHERE ename='miller';
```

この文では、次のような名前を戻すことができます。

```
Miller
MILLER
Millér
```

これは言語検索ではなく、特定の言語に基づいていないことに注意してください。この検索ではベース文字のみが使用されます。

5

Unicode を使用した多言語データベースのサポート

この章では、Oracle データベース環境での Unicode の使用方法について説明します。この章の内容は、次のとおりです。

- Unicode の概要
- Unicode の概要
- Unicode ソリューションのデータベースへの実装
- Unicode の事例
- 複数言語サポートのためのデータベース・スキーマ設計

Unicode の概要

同じアプリケーションやデータベース内で多数の異なる言語を処理することは、長い間複雑で困難な処理でした。既存の文字エンコーディングの制約を克服するために、1980 年代の後半、複数の組織がグローバル・キャラクタ・セットの作成に着手しました。グローバル・キャラクタ・セットの必要性は、1990 年代中頃に入り、World Wide Web の発展とともにますます大きくなりました。インターネットの普及によってビジネスの形態が変化し、グローバル・マーケットに重点が置かれるようになったため、ユニバーサル・キャラクタ・セットが重要な必要条件になってきました。このグローバル・キャラクタ・セットは、次の条件を満たす必要があります。

- 現在使用されているすべての主要文字を網羅していること。
- レガシー・データや実装をサポートしていること。
- 1 つのアプリケーションの 1 つの実装で、世界中で使用できるような単純なキャラクタ・セットであること。

また、グローバル・キャラクタ・セットには次の機能も必要です。

- 多言語ユーザーや組織のサポート
- 国際規格への準拠
- 世界規模でのデータ交換

このグローバル・キャラクタ・セットが、現在世界中で使用されている Unicode と呼ばれるキャラクタ・セットです。

Unicode の概要

Unicode とは、エンコードされたユニバーサル・キャラクタ・セットのことです。このセットを使用すると、1 つのキャラクタ・セットを使用して任意の言語の情報を格納できます。Unicode には、プラットフォーム、プログラムまたは言語に関係なく、すべての文字に対する一意のコード値が用意されています。

多くのソフトウェアとハードウェアのベンダーが Unicode 規格を採用しています。また、多くのオペレーティング・システムやブラウザがサポートしています。Unicode は、XML、Java、JavaScript、LDAP および WML などの規格には必須です。また、ISO/IEC 10646 規格とも同期化されています。

オラクル社が Unicode をサポートしたのは、Oracle7 のデータベース・キャラクタ・セットが最初です。Oracle9*i* では、Unicode のサポートが拡張されており、Unicode 3.1 がサポートされています。

> **関連項目：** Unicode 規格の詳細は、`http://www.unicode.org` を参照してください。

この項の内容は、次のとおりです。

補助文字

Unicode の最初のバージョンは 16 ビットの固定幅エンコーディングで、各文字のエンコーディングに 2 バイトを使用し、65,536 文字を表現できました。しかし、サポートを必要とする文字数が増え、特に中国語、日本語および韓国語市場では、追加の CJK 表意文字のサポートが必要となりました。

Unicode 3.1 では、このニーズを満たすために補助文字が定義されており、2 つの 16 ビット・コード・ポイント（補助文字とも呼ばれます）を使用して 1 文字が表現されます。このため、さらに 1,048,576 文字の定義が可能です。Unicode 3.1 規格には、最初のグループとして 44,944 文字の補助文字が追加されました。

補助文字の追加によって Unicode は複雑度を増しましたが、同じ構成で複数の異なるエンコーディングを管理するよりは単純です。

Unicode エンコーディング

Unicode 3.1 には、文字は UTF-8、UCS-2 および UTF-16 という方法でエンコーディングされます。Unicode エンコーディング間の変換は、Unicode 規格に定義されている単純なビット単位操作です。

この項の内容は、次のとおりです。

UTF-8 エンコーディング

UTF-8 とは、Unicode の 8 ビット・エンコーディングのことです。これは可変幅エンコーディングであり、ASCII の**完全なスーパーセット**です。これは、ASCII キャラクタ・セットの各文字を、UTF-8 で同じコード・ポイント値とともに使用できることを意味します。UTF-8 エンコーディングでは、1 つの Unicode 文字を、1 バイト、2 バイト、3 バイトまたは 4 バイトで表すことができます。ヨーロッパ言語の文字は、1 バイトまたは 2 バイトで表します。ほとんどのアジア言語の文字は、3 バイトで表します。補助文字は、4 バイトで表します。

UTF-8 は UNIX プラットフォーム上でサポートされる Unicode エンコーディングであり、HTML とほとんどのインターネット・ブラウザで使用されてます。Windows や Java など、他の環境では、UCS-2 エンコーディングが使用されています。

UTF-8 の利点は、次のとおりです。

- ASCII の完全なスーパーセットであるため、ヨーロッパ言語の記憶要件が小規模です。
- ASCII ベースのキャラクタ・セットと UTF-8 の間の移行が容易です。

関連項目：

- 5-3 ページ「補助文字」
- B-2 ページの表 B-2「UTF-8 文字コード用の Unicode 文字コード範囲」

UCS-2 エンコーディング

UCS-2 は固定幅の 16 ビット・エンコーディングで、各文字は 2 バイトです。この Unicode エンコーディングは、Java と Microsoft Windows NT 4.0 で使用されています。UCS-2 では Unicode 3.0 に定義されている文字がサポートされるため、補助文字のサポート機能はありません。

UTF-8 と比較した UCS-2 の利点は、次のとおりです。

- すべての文字が 2 バイトであるため、アジア言語の場合は記憶要件がさらに小規模です。
- 文字が固定幅であるため、文字列の処理が高速です。
- Java および Microsoft クライアントとの優れた互換性があります。

関連項目： 5-3 ページ「補助文字」

UTF-16 エンコーディング

UTF-16 エンコーディングとは、Unicode の 16 ビット・エンコーディングのことです。UTF-16 は、UCS-2 の拡張です。これは、各補助文字に 2 つの UCS-2 コード・ポイントを使用することで Unicode 3.1 に定義されている補助文字をサポートするためです。UTF-16 は、UCS-2 の完全なスーパーセットです。

UTF-16 では、1 文字を 2 バイトか 4 バイトで表すことができます。ヨーロッパ言語とほとんどのアジア言語の文字は、ともに 2 バイトで表します。補助文字は、4 バイトで表します。UTF-16 は、Microsoft Windows 2000 で使用されている主要 Unicode エンコーディングです。

UTF-8 と比較した UTF-16 の利点は、次のとおりです。

- 一般に使用されるアジア言語の文字のほとんどは 2 バイトで表されるため、アジア言語の場合は記憶要件がさらに小規模です。
- Java および Microsoft クライアントとの優れた互換性があります。

関連項目：

- 5-3 ページ「補助文字」
- B-2 ページの表 B-1「UTF-16 文字コード用の Unicode 文字コード範囲」

例 : UTF-16、UTF-8 および UCS-2 エンコーディング

図 5-1 に、UTF-16、UTF-8 および UCS-2 の各エンコーディングの文字とその文字コードを示します。最後の文字はト音記号（音楽記号）で、補助文字として Unicode 3.1 規格に追加されました。

図 5-1 UTF-16、UTF-8 および UCS-2 エンコーディングの例

文字	UTF-16	UTF-8	UCS-2
A	0041	41	0041
c	0063	63	0063
Ö	00F6	C3 B6	00F6
亜	4E9C	E4 BA 9C	4E9C
𝄞	D834 DD1E	F0 9D 84 9E	N/A

Oracle による Unicode のサポート

オラクル社が Unicode をサポートしたのは、Oracle7 のデータベース・キャラクタ・セットが最初です。表 5-1 に、Oracle データベース・サーバーでサポートされている Unicode キャラクタ・セットを示します。

表 5-1 Oracle データベース・サーバーでサポートされている Unicode キャラクタ・セット

キャラクタ・セット	サポートしている RDBMS のリリース	Unicode エンコーディング	Unicode のバージョン	データベース・キャラクタ・セット	各国語キャラクタ・セット
AL24UTFFSS	7.2 - 8*i*	UTF-8	1.1	可能	不可
UTF8	8.0 - 9*i*	UTF-8	Oracle8 リリース 8.0 ～ Oracle8*i* リリース 8.1.6 の場合 : 2.1 Oracle8*i* リリース 8.1.7 以上の場合 : 3.0	可能	可能（Oracle9*i* のみ）
UTFE	8.0 - 9*i*	UTF-8	Oracle8*i* リリース 8.0 ～ 8.1.6 の場合 : 2.1 Oracle8*i* リリース 8.1.7 以上の場合 : 3.0	可能	不可
AL32UTF8	9*i*	UTF-8	Oracle9*i* リリース 1（9.0.1）の場合 : 3.0 Oracle9*i* リリース 2（9.2）の場合 : 3.1	可能	不可
AL16UTF16	9*i*	UTF-16	Oracle9*i* リリース 1（9.0.1）の場合 : 3.0 Oracle9*i* リリース 2（9.2）の場合 : 3.1	不可	可能

Unicode ソリューションのデータベースへの実装

次の 2 つの方法で Unicode 文字を Oracle9*i* データベースに格納できます。

Unicode データベースを作成すると、UTF-8 エンコードされた文字を SQL CHAR データ型（CHAR、VARCHAR2、CLOB および LONG）として格納できます。

Unicode サポートを段階的に実装する場合、または特定の列の多言語データのみをサポートする必要がある場合、Unicode データは SQL NCHAR データ型（NCHAR、NVARCHAR2 および NCLOB）の UTF-16 または UTF-8 エンコーディング方式のいずれかで格納できます。SQL NCHAR データ型は、Unicode データの格納専用に使用されるため、Unicode データ型と呼ばれます。

注意： Unicode データベース・ソリューションは、Unicode データ型ソリューションと組み合せることができます。

次の項で、2 つの Unicode ソリューションの使用方法とその選択方法を説明します。

- Unicode データベースを使用した多言語サポートの有効化
- Unicode データ型を使用した多言語サポートの有効化
- Unicode データベースと Unicode データ型のソリューションの選択方法
- データベースおよびデータ型ソリューションに関する Unicode キャラクタ・セットの比較

Unicode データベースを使用した多言語サポートの有効化

データベース・キャラクタ・セットでは、SQL CHAR データ型の他に、表名、列名および SQL 文などのメタデータに使用するエンコーディングが指定されます。**Unicode データベース**とは、UTF-8 をデータベース・キャラクタ・セットとして持つデータベースのことです。UTF-8 エンコーディングを実装している Oracle キャラクタ・セットは 3 つあります。最初の 2 つは、ASCII ベースのプラットフォーム用に設計されており、3 つ目は、EBCDIC プラットフォーム上で使用する必要があります。

- AL32UTF8

  AL32UTF8 キャラクタ・セットでは、最新バージョンの Unicode 規格をサポートしています。このキャラクタ・セットでは、各文字は 1 バイト、2 バイトまたは 3 バイトでエンコードされ、補助文字には 4 バイトが必要です。ASCII ベースのプラットフォーム用です。

- UTF8

  UTF8 キャラクタ・セットは、文字を 1 バイト、2 バイトまたは 3 バイトでエンコードします。ASCII ベースのプラットフォーム用です。

Oracle8*i* リリース 8.1.7 以上は UTF8 キャラクタ・セットで Unicode 3.0 がサポートされており、Oracle データベース・サーバーの将来のリリースでも引き続きサポートされます。Unicode 3.1 までは特定の補助文字にコード・ポイントが割り当てられていませんでしたが、Unicode 3.0 では補助文字用のコード・ポイント範囲が割り当てられていました。UTF8 データベースに補助文字が挿入されても、データベース内のデータは破損しません。補助文字は、6 バイトを占める 2 つの別個のユーザー定義文字として処理されます。データベース・キャラクタ・セット内の補助文字を全面的にサポートするために、AL32UTF8 に切り替えることをお薦めします。

- UTFE

  UTFE キャラクタ・セットは EBCDIC プラットフォーム用です。このキャラクタ・セットのプロパティは、ASCII プラットフォーム上での UTF8 と同じです。

例 5-1 Unicode キャラクタ・セットを使用したデータベースの作成

AL32UTF8 キャラクタ・セットを使用してデータベースを作成するには、CREATE DATABASE 文に CHARACTER SET AL32UTF8 句を使用します。次に例を示します。

```
CREATE DATABASE sample
    CONTROLFILE REUSE
    LOGFILE
        GROUP 1 ('diskx:log1.log', 'disky:log1.log') SIZE 50K,
        GROUP 2 ('diskx:log2.log', 'disky:log2.log') SIZE 50K
    MAXLOGFILES 5
    MAXLOGHISTORY 100
    MAXDATAFILES 10
    MAXINSTANCES 2
    ARCHIVELOG
    CHARACTER SET AL32UTF8
    NATIONAL CHARACTER SET AL16UTF16
    DATAFILE
        'disk1:df1.dbf' AUTOEXTEND ON,
        'disk2:df2.dbf' AUTOEXTEND ON NEXT 10M MAXSIZE UNLIMITED
    DEFAULT TEMPORARY TABLESPACE temp_ts
    UNDO TABLESPACE undo_ts
    SET TIME_ZONE = '+02:00';
```

注意： データベース・キャラクタ・セットは、データベースの作成時に指定します。

Unicode データ型を使用した多言語サポートの有効化

データベースに Unicode データを格納する代替方法は、SQL NCHAR データ型（NCHAR、NVARCHAR、NCLOB）を使用することです。データベース・キャラクタ・セットの定義方法に関係なく、これらのデータ型の列に Unicode 文字を格納できます。Oracle9*i* では、NCHAR データ型は、Unicode データ型専用として再定義されています。つまり、このデータ型は、Unicode としてエンコードされたデータを格納します。

Oracle9*i* より前のリリースでは、NCHAR データ型は、パフォーマンスを向上させるように設計された固定幅のアジア言語キャラクタ・セットをサポートしていました。固定幅キャラクタ・セットの例に、JA16SJISFIXED および ZHT32EUCFIXED があります。Oracle9*i* までは、Unicode キャラクタ・セットは各国語キャラクタ・セットとしてサポートされていませんでした。

NVARCHAR2 と NCHAR のデータ型を使用すると、表を作成できます。NCHAR 列と NVARCHAR2 列に指定する列の長さは、次のように常にバイト数ではなく文字数です。

```
CREATE TABLE product_information
    ( product_id          NUMBER(6)
    , product_name        NVARCHAR2(100)
    , product_description VARCHAR2(1000));
```

SQL NCHAR データ型で使用するエンコーディングは、データベース用に指定される各国語キャラクタ・セットです。次の Oracle キャラクタ・セットのいずれかを、各国語キャラクタ・セットとして指定できます。

- AL16UTF16

  これは、SQL NCHAR データ型のデフォルトのキャラクタ・セットです。このキャラクタ・セットは、Unicode データを UTF-16 エンコーディングでエンコードします。また、4 バイトとして格納される補助文字をサポートします。

- UTF8

  SQL NCHAR データ型に UTF8 を指定すると、SQL データ型で格納されているデータは、UTF-8 エンコーディングになります。

NATIONAL CHARACTER SET 句を指定した CREATE DATABASE 文を使用してデータベースを作成する場合は、SQL NCHAR データ型に各国語キャラクタ・セットを指定できます。次の文では、データベース・キャラクタ・セットとして WE8ISO8859P1 を持ち、各国語キャラクタ・セットとして AL16UTF16 を持つデータベースを作成します。

例 5-2 各国語キャラクタ・セットを使用したデータベースの作成

```
CREATE DATABASE sample
    CONTROLFILE REUSE
    LOGFILE
        GROUP 1 ('diskx:log1.log', 'disky:log1.log') SIZE 50K,
        GROUP 2 ('diskx:log2.log', 'disky:log2.log') SIZE 50K
    MAXLOGFILES 5
    MAXLOGHISTORY 100
    MAXDATAFILES 10
    MAXINSTANCES 2
    ARCHIVELOG
    CHARACTER SET WE8ISO8859P1
    NATIONAL CHARACTER SET AL16UTF16
    DATAFILE
        'disk1:df1.dbf' AUTOEXTEND ON,
        'disk2:df2.dbf' AUTOEXTEND ON NEXT 10M MAXSIZE UNLIMITED
    DEFAULT TEMPORARY TABLESPACE temp_ts
    UNDO TABLESPACE undo_ts
    SET TIME_ZONE = '+02:00';
```

Unicode データベースと Unicode データ型のソリューションの選択方法

データベースに対する適切な Unicode ソリューションを選択するには、次の問題を考慮してください。

- プログラミング環境 : アプリケーションで使用されている主要なプログラミング言語は何か。これらのプログラミング言語は、どのように Unicode をサポートしているか。
- 移行の容易さ : Unicode ソリューションを利用するために、データとアプリケーションを移行することに問題はないか。
- パフォーマンス : データベースで Unicode を使用するために許容できるパフォーマンス上のオーバーヘッドはどの程度か。
- データの種類 : データの大部分は、アジア言語、あるいはヨーロッパ言語か。多言語ドキュメントを LOB 列に格納する必要があるか。
- アプリケーションの種類 : 実装しているアプリケーションの種類は何か。パッケージ・アプリケーションか、あるいはカスタマイズしたエンド・ユーザー・アプリケーションか。

この項では、Unicode データベース・ソリューションまたは Unicode データ型ソリューションを選択する場合の一般的なガイドラインを説明します。最終的な決定は、主として各自の正確な環境と要件に基づいて行います。この項の内容は、次のとおりです。

- Unicode データベースの使用が必要な状況
- Unicode データ型の使用が必要な状況

Unicode データベースの使用が必要な状況

表 5-2 に、Unicode データベースを使用する状況を示します。

表 5-2 Unicode データベースの使用

状況	説明
Java または PL/SQL に対する簡単なコード移行が必要な場合	既存のアプリケーションが主として Java と PL/SQL で作成されており、複数言語のサポートに必要なコード変更を最小限にすることが最優先の課題である場合は、Unicode データベース・ソリューションを使用できます。データの格納に使用するデータ型が SQL CHAR データ型のままである場合は、これらの列にアクセスする Java と PL/SQL のコードを変更する必要はありません。
多言語データを均等に分散した場合	多言語データが既存のスキーマ表に均等に分散されていて、多言語データが格納されている表が確認できない場合には、Unicode データベースを使用する必要があります。これは、この Unicode データベースでは、各列に格納されているデータの種類を識別する必要がないためです。
SQL 文と PL/SQL コードに Unicode データが含まれている場合	Unicode データベースを使用する必要があります。SQL 文と PL/SQL コードは、処理前にデータベース・キャラクタ・セットに変換されます。SQL 文と PL/SQL コードにデータベース・キャラクタ・セットに変換できない文字が含まれている場合、その文字は失われます。SQL 文で Unicode データを使用する一般的な場所は、文字列リテラルにあります。
多言語ドキュメントを BLOB として格納し、Oracle Text を内容検索に使用する場合	Unicode データベースを使用する必要があります。BLOB データは、Oracle Text による索引付けが行われる前に、データベース・キャラクタ・セットに変換されます。データベース・キャラクタ・セットが UTF8 でない場合は、ドキュメントにデータベース・キャラクタ・セットに変換できない文字が含まれていると、データが失われます。

Unicode データ型の使用が必要な状況

表 5-3 に、Unicode データ型を使用する状況を示します。

表 5-3 Unicode データ型の使用

状況	説明
多言語サポートを段階的に追加する場合	キャラクタ・セットを移行せずに、Unicode サポートを既存のデータベースに追加する場合は、Unicode データの格納に Unicode データ型を使用することを考慮してください。SQL NCHAR データ型の列を既存の表または新しい表に追加すると、複数言語を段階的にサポートできます。
パッケージ・アプリケーションを作成する場合	顧客に販売するパッケージ・アプリケーションを作成している場合は、SQL NCHAR データ型を使用してアプリケーションを作成できます。SQL NCHAR データ型は信頼性のある Unicode データ型であり、データは常に Unicode で格納され、データ長は常に UTF-16 コードの単位で指定されています。その結果、アプリケーションのテストは 1 回のみですみます。アプリケーションは、任意のデータベース・キャラクタ・セットを持つ顧客のデータベースで稼働します。
シングルバイトのデータベース・キャラクタ・セットを使用してパフォーマンスを改善する場合	パフォーマンスの向上が主な優先事項である場合は、シングルバイトのデータベース・キャラクタ・セットを使用して、Unicode データを SQL NCHAR データ型で格納することを考慮してください。UTF8 などのマルチバイトのデータベース・キャラクタ・セットを使用するデータベースでは、パフォーマンス上のオーバーヘッドが発生します。
Windows クライアントで UTF-16 のサポートが必要な場合	アプリケーションが Windows 上で稼働する Visual C/C++ や Visual Basic で作成されている場合は、SQL NCHAR データ型を使用できます。SQL NCHAR データ型の UTF-16 データを、Visual C/C++ の `wchar_t` バッファおよび Visual Basic の `string` バッファに格納するのと同じ方法で格納できます。クライアント・アプリケーションでのバッファのオーバーフローは回避できます。これは、`wchar_t` データ型と `string` データ型の長さは、データベースの SQL NCHAR データ型の長さと一致しているためです。

注意： Unicode データベースと Unicode データ型は併用できます。

データベースおよびデータ型ソリューションに関する Unicode キャラクタ・セットの比較

Oracle9*i* には、データベースに Unicode 文字を格納するためのソリューションが 2 つ用意されています。Unicode データベース・ソリューションと Unicode データ型ソリューションです。Unicode データベース・ソリューション、Unicode データ型ソリューションまたはその組合せを選択した後は、Unicode データベースまたは Unicode データ型で使用するキャラクタ・セットを決定します。

表 5-4 に、Unicode データベース・ソリューションの様々なキャラクタ・セットのメリットとデメリットを示します。Unicode データベース・キャラクタ・セットを使用できる Oracle キャラクタ・セットは、AL32UTF8、UTF8 および UTFE です。

表 5-4 Unicode データベース・ソリューション用キャラクタ・セットのメリットとデメリット

データベース・キャラクタ・セット	メリット	デメリット
AL32UTF8	■ 補助文字は 4 バイトで格納されるため、補助文字の取得時と挿入時にデータ変換は行われません。 ■ AL32UTF8 では、補助文字の格納に必要なディスク領域が UTF8 の場合に比較して少なくなります。	■ SQL CHAR 型の長さは、補助文字の文字数（Unicode コード・ポイント）で指定できません。補助文字は、この規格の 2 コード・ポイントではなく、1 コード・ポイントとして処理されます。 ■ データが補助文字で構成されている場合、SQL CHAR 列のバイナリ順序は、SQL NCHAR 列のバイナリ順序と同じではありません。その結果、CHAR 列と NCHAR 列では、同じ文字列に対して常に同じようにソートされるとはかぎりません。
UTF8	■ SQL CHAR 型の長さを文字数として指定できます。 ■ データが同じ補助文字で構成されている場合、SQL CHAR 列のバイナリ順序は、SQL NCHAR 列のバイナリ順序と常に同じです。その結果、CHAR 列と NCHAR 列には、同じ文字列に対して同じソートがあります。	■ 補助文字は、Unicode 3.1 で定義された 4 バイトではなく、6 バイトとして格納されます。その結果、補助文字にはデータ変換が必要になります。

表 5-4 Unicode データベース・ソリューション用キャラクタ・セットのメリットとデメリット（続き）

データベース・キャラクタ・セット	メリット	デメリット
UTFE	■ これは、EBCDIC プラットフォーム用の唯一の Unicode キャラクタ・セットです。 ■ SQL CHAR 型の長さを文字数として指定できます。 ■ データが同じ補助文字で構成されている場合、SQL CHAR 列のバイナリ順序は、SQL NCHAR 列のバイナリ順序と常に同じです。その結果、CHAR 列と NCHAR 列には、同じ文字列に対して同じソートがあります。	■ 補助文字は、Unicode 規格で定義された 4 バイトではなく、6 バイトとして格納されます。その結果、この補助文字にはデータ変換が必要になります。 ■ UTFE は、Unicode 規格の標準エンコーディングではありません。その結果、標準 UTF-8 エンコーディングが必要なクライアントは、データの取得時と挿入時に UTFE から標準エンコーディングへのデータ変換を実行する必要があります。

表 5-5 に、Unicode データ型ソリューションの様々なキャラクタ・セットのメリットとデメリットを示します。各国語キャラクタ・セットに使用できる Oracle キャラクタ・セットは、AL16UTF16 および UTF8 です。

表 5-5 Unicode データ型ソリューション用キャラクタ・セットのメリットとデメリット

各国語キャラクタ・セット	メリット	デメリット
AL16UTF16	■ AL16UTF16 でのアジア言語データは通常、UTF8 のデータに比較して小型です。その結果、データベースに格納されている多言語データの大部分がアジア言語データである場合は、ディスク領域が節減され、ディスクの I/O が削減されます。 ■ 一般的に、AL16UTF16 キャラクタ・セットでエンコードされた文字列の処理は、UTF8 でエンコードされた文字列より高速です。これは、Oracle9*i* では、AL16UTF16 でエンコードされた文字列のほとんどの文字が固定幅文字として処理されるためです。 ■ NCHAR 列と NVARCHAR2 列の最大長は、それぞれ 1000 文字と 2000 文字に制限されています。データは固定幅であるため、長さが保証されています。	■ ヨーロッパ言語の ASCII データでは、UTF8 に比較して、AL16UTF16 での格納に大きなディスク領域が必要です。データのほとんどがヨーロッパ言語の場合は、UTF8 データの場合に比較してディスク領域の使用量が多くなります。 ■ NCHAR 列と NVARCHAR2 列の最大長は、それぞれ 1000 文字と 2000 文字です。これは、UTF8 の場合の NCHAR（2000 文字）と NVARCHAR2（4000 文字）に比較して少なくなります。

表 5-5 Unicode データ型ソリューション用キャラクタ・セットのメリットとデメリット（続き）

各国語キャラクタ・セット	メリット	デメリット
UTF8	■ UTF8 でのヨーロッパ言語データは通常、AL16UTF16 でのデータに比較して小型です。その結果、データベースに格納されている多言語データの大部分がヨーロッパ言語データである場合、ディスク領域が節減され、応答時間が短縮されます。 ■ NCHAR 列と NVARCHAR2 列の最大長は、それぞれ 2000 文字と 4000 文字です。これは、AL16UTF16 の場合の NCHAR（1000 文字）と NVARCHAR2（2000 文字）に比較して多くなります。UTF8 では、NCHAR 列と NVARCHAR2 列の最大長は大きくなりますが、UTF8 での実際の格納サイズは、それぞれ 2000 バイトと 4000 バイトというバイト制限で拘束されています。たとえば、すべての文字がシングルバイトの場合は、NVARCHAR2 列には 4000 UTF8 文字を格納できますが、すべての文字が 3 バイトの場合は、格納できる文字は 4000/3 文字のみになります。	■ アジア言語データでは、AL16UTF16 に比較して、UTF8 での格納に大きなディスク領域が必要になります。データの大部分がアジア言語データの場合、ディスク領域の使用では、キャラクタ・セットが AL16UTF16 の場合ほど効率的ではありません。 ■ NCHAR と NVARCHAR に対して長さ制限の拡大を指定できますが、その制限の拡大によって指定した文字数を挿入できる保証はありません。これは、UTF8 では可変幅の文字を使用できるためです。 ■ 通常、UTF8 でエンコードされた文字列の処理は、AL16UTF16 でエンコードされた文字列より低速です。これは、UTF8 でエンコードされた文字列が可変幅の文字で構成されているためです。

Unicode の事例

この項では、Unicode 文字を Oracle9*i* データベースに格納する場合の代表的な使用例を説明します。

例 5-3 Unicode データベースを使用した Unicode ソリューション

Java アプリケーションを実行しているある米国の企業では、アプリケーションの次のリリースでドイツ語とフランス語のサポートを追加し、その後日本語のサポートを追加する予定です。この会社には、現在次のようなシステム構成があります。

- 既存のデータベースは、US7ASCII のデータベース・キャラクタ・セットです。
- 既存のデータベースの全文字データは、ASCII 文字で構成されています。
- データベースでは、PL/SQL ストアド・プロシージャが使用されています。
- データベースは、およそ 300 GB です。
- 夜間の停止時間は 4 時間です。

この場合、代表的なソリューションは、データベース・キャラクタ・セットに UTF8 を選択します。その理由は、次のとおりです。

- データベースが非常に大規模で、予定されている停止時間が短いこと。Unicode へのデータベースの移行の高速化が不可欠です。データベースが US7ASCII であるため、データベースの Unicode サポートを有効化する最も簡単で迅速な方法は、`ALTER DATABASE` 文を発行して、データベース・キャラクタ・セットを UTF8 に切り替えることです。US7ASCII は、UTF-8 のサブセットであるため、データ変換は不要です。
- コードの大部分が Java と PL/SQL で記述されているため、データベース・キャラクタ・セットの UTF8 への変更によって、既存のコードが無駄になることはないこと。Unicode サポートはアプリケーションで自動的に有効化されます。
- アプリケーションがフランス語、ドイツ語および日本語をサポートするため、補助文字はほとんどないこと。AL32UTF8 と UTF8 の両方が適しています。

例 5-4 Unicode データ型を使用した Unicode ソリューション

主として Windows プラットフォームでアプリケーションを実行しているあるヨーロッパの企業では、Visual C/C++ で作成された新しい Windows アプリケーションを追加する必要があります。このアプリケーションでは、日本語と中国語の顧客名をサポートするために、既存のデータベースが使用されます。この会社には、現在次のようなシステム構成があります。

- 既存のデータベースには、WE8ISO8859P1 のデータベース・キャラクタ・セットがあります。
- 既存のデータベースの全文字データは、西ヨーロッパの文字で構成されています。
- データベースは、およそ 50 GB です。

代表的なソリューションでは、次のアクションが実行されます。

- NCHAR および NVARCHAR2 データ型を使用して Unicode 文字が格納されます。
- WE8ISO8859P1 がデータベース・キャラクタ・セットとして維持されます。
- AL16UTF16 が各国語キャラクタ・セットとして使用されます。

このソリューションを選択する理由は、次のとおりです。

- データベース・キャラクタ・セットの WE8ISO8859P1（Latin-1 キャラクタ・セット）は UTF8 のサブセットでないため、既存のデータベースを Unicode データベースに移行するには、データ変換が必要になること。その結果、データを UTF8 に変換するときにオーバーヘッドが発生します。
- 追加した言語は、新しいアプリケーションでのみサポートされること。既存のアプリケーションまたはスキーマに対する依存性はありません。Unicode データ型は新しいスキーマで使用し、既存のスキーマは変更しないままのほうが簡単です。
- Unicode のサポートが必要なのは、顧客名の列のみであること。 1 つの NCHAR 列のみを使用することで、データベース全体を移行せずに顧客の要件を満たすことができます。
- サポート対象言語がほとんどアジア言語であるため、AL16UTF16 を各国語キャラクタ・セットとして使用する必要があり、結果として、ディスク領域をより効率的に使用できること。
- SQL NCHAR データ型の長さは文字数として定義されること。この処理方法は、Windows C/C++ プログラムで wchar_t 文字列を使用する場合と同じです。この方法は、プログラミングの複雑さを軽減します。
- 既存のスキーマを使用している既存のアプリケーションに対する影響はないこと。

例 5-5 Unicode データベースと Unicode データ型を使用した Unicode ソリューション

ある日本の企業では、Oracle9*i* 上で新しい Java アプリケーションを開発する必要があります。会社では、このアプリケーションによって、できるだけ多数の言語を長期にわたってサポートする予定です。

- そこで、ドキュメントを現状のまま格納するために、BLOB データ型を使用して複数言語のドキュメントを格納することに決定しました。
- また、B2B でのデータ交換のためにリレーショナル・データから UTF-8 XML ドキュメントを生成する必要もあります。
- バックエンドには、C/C++ で作成した Windows アプリケーションがあり、Oracle データベースにアクセスするために ODBC を使用しています。

この場合、代表的なソリューションは、データベース・キャラクタ・セットに AL32UTF8 を使用して Unicode データベースを作成し、SQL NCHAR データ型を使用して多言語データを格納します。各国語キャラクタ・セットは、AL16UTF16 に設定する必要があります。このソリューションを選択する理由は、次のとおりです。

- 異なる言語のドキュメントが BLOB として格納されている場合、Oracle Text では、データベース・キャラクタ・セットが UTF-8 キャラクタ・セットの 1 つである必要があること。アプリケーションでは、リレーショナル・データを UTF-8 XML フォーマットとして取得する可能性があるため（補助文字が 4 バイトで格納されている場合）、AL32UTF8 をデータベース・キャラクタ・セットとして使用し、UTF-8 データの取得時または挿入時のデータ変換を回避する必要があります。
- アプリケーションが新規で、Java と Windows C/C++ の両方で作成されているため、この会社ではリレーショナル・データに対して SQL NCHAR データ型を使用する必要があること。Java と Windows の両方が UTF-16 文字のデータ型をサポートしていて、文字列の長さは常に文字数で測定されます。
- データの大部分がアジア言語の場合は、AL16UTF16 を SQL NCHAR データ型と併用する必要があること。これは、AL16UTF16 によって、パフォーマンスと格納の効率が向上するためです。

複数言語サポートのためのデータベース・スキーマ設計

複数言語をサポートするようにデータベース・スキーマを設計する場合は、Unicode ソリューションの選択に加えて、次の問題も考慮する必要があります。

- 多言語データに使用する列の長さの指定
- 複数言語のデータの格納
- LOB への複数言語によるドキュメントの格納
- 多言語ドキュメントの内容検索に使用する索引の作成

多言語データに使用する列の長さの指定

多言語データの格納に NCHAR および NVARCHAR2 データ型を使用する場合、列に指定するサイズは文字数で定義します（文字数は、Unicode のコード数を意味します）。表 5-6 に、AL16UTF16 および UTF8 の各国語キャラクタ・セットに対する NCHAR および NVARCHAR2 データ型の最大サイズを示します。

表 5-6 データ型の最大サイズ

各国語キャラクタ・セット	NCHAR データ型の最大列サイズ	NVARCHAR2 データ型の最大列サイズ
AL16UTF16	1000 文字	2000 文字
UTF8	2000 バイト	4000 バイト

多言語データの格納に CHAR および VARCHAR2 のデータ型を使用する場合、各列に指定する最大長は、デフォルトではバイト数で指定します。データベースで、タイ語、アラビア語、あるいは中国語や日本語などのマルチバイトの言語をサポートする必要がある場合は、CHAR、VARCHAR および VARCHAR2 の各列の最大サイズを拡張する必要があります。これは、これらの言語を UTF8 または AL32UTF8 でエンコードするためのバイト数が、英語や西ヨーロッパの言語に比較してかなり多いためです。たとえば、タイ語キャラクタ・セットの 1 つのタイ文字は、UTF8 または AL32UTF8 で 3 バイト必要です。さらに、CHAR、VARCHAR および VARCHAR2 の各データ型の最大列長は、それぞれ 2000 バイト、4000 バイトおよび 4000 バイトとなります。アプリケーションで 4000 バイトを超えるバイト数を格納する必要がある場合は、そのデータに対して CLOB データ型を使用する必要があります。

複数言語のデータの格納

Unicode キャラクタ・セットには、世界中で使用されている記述言語の文字がほとんど含まれています。ただし、このキャラクタ・セットには、特定の文字が所属している言語に関する情報は含まれていません。たとえば、ä という文字には、それがフランス語の文字なのかドイツ語の文字なのかに関する情報は含まれていません。ユーザーが望む言語で情報を表示するには、Unicode データベースに格納されているデータに、そのデータが所属している言語の情報が含まれている必要があります。

データベース・スキーマがデータを言語に関連付ける方法は多数あります。次の各項では、様々なアプローチについて説明します。

- 言語情報をデータとともに格納
- ファイングレイン・アクセス・コントロールを使用した変換済みデータの選択

言語情報をデータとともに格納

製品説明や製品名などのデータの場合は、CHAR または VARCHAR2 のデータ型の言語列（language_id）を製品表に追加して、対応する製品情報の言語を識別できます。これによって、アプリケーションは、必要な言語による情報を取り出すことができます。この言語列に可能な値は、データベースの有効な NLS_LANGUAGE 値の 3 文字の略称です。

> **関連項目：** NLS_LANGUAGE 値とその略称のリストは、付録 A「ロケール・データ」を参照してください。

ビューを作成して、現在の言語のデータを選択することもできます。次に例を示します。

```
ALTER TABLE scott.product_information add (language_id VARCHAR2(50)):

CREATE OR REPLACE VIEW product AS
   SELECT product_id, product_name
   FROM   product_information
   WHERE  language_id = sys_context('USERENV','LANG');
```

ファイングレイン・アクセス・コントロールを使用した変換済みデータの選択

ファイングレイン・アクセス・コントロールを使用すると、表またはビューでユーザーが参照できる情報の程度に制限を与えることができます。通常、そのためには WHERE 句を追加します。WHERE 句をファイングレイン・アクセス・ポリシーとして表またはビューに追加すると、Oracle9*i* では、表にあるすべての SQL 文にその WHERE 句が実行時に自動的に追加されます。その結果、WHERE 句を満たす行のみがアクセス可能になります。

この機能を使用すると、アプリケーションの SELECT 文ごとに、ユーザーの必要な言語を WHERE 句で指定しなくてすみます。次の WHERE 句は、表のビューをユーザーが必要な言語に対応している行に制限しています。

```
WHERE language_id = sys_context('userenv', 'LANG')
```

この WHERE 句をファイングレイン・アクセス・ポリシーとして product_information に対して次のように指定します。

```
create function func1 ( sch varchar2 , obj varchar2 )
return varchar2(100);
begin
return 'language_id = sys_context(''userenv'', ''LANG'')';
end
/

DBMS_RLS.ADD_POLICY ('scott', 'product_information', 'lang_policy', 'scott',
'func1', 'select');
```

その結果、product_information 表にあるすべての SELECT 文は自動的に WHERE 句を追加します。

> **関連項目：** ファイングレイン・アクセス・コントロールの詳細は、『Oracle9*i* アプリケーション開発者ガイド - 基礎編』を参照してください。

LOB への複数言語によるドキュメントの格納

複数言語によるドキュメントを CLOB、NCLOB または BLOB データ型に格納して Oracle Text を設定すると、そのドキュメントの内容検索を使用できます。

データベース・キャラクタ・セットが UTF8 や AL32UTF8 などのマルチバイトである場合、CLOB 列のデータは UCS-2 として内部的に格納されます。ドキュメントの内容は、CLOB 列への挿入時に、UTF-16 に変換されます。つまり、データの変換時には、英語のドキュメントに必要な記憶領域は 2 倍になります。アジア言語のドキュメントを CLOB 列に格納する場合は、同じドキュメントを UTF8 を使用して LONG 列に格納する場合に比較して、必要な記憶領域が少なくなります。ドキュメントの内容にもよりますが、通常およそ 30% 少なくなります。

NCLOB 内のドキュメントは、データベース・キャラクタ・セットや各国語キャラクタ・セットに関係なく、UTF-16 として格納されます。記憶領域の必要量は、CLOB の場合と同じです。ドキュメントの内容は、NCLOB 列への挿入時に、UTF-16 に変換されます。非 Unicode データベースに多言語ドキュメントを格納する場合は、NCLOB を選択してください。ただし、NCLOB 上での内容検索は、現段階ではサポートされていません。

BLOB 書式のドキュメントは、そのまま格納されます。挿入時および取得時にデータ変換は発生しません。ただし、SQL 文字列操作関数（LENGTH や SUBSTR など）と照合関数（NLS_SORT や ORDER BY など）は、BLOB データ型に適用できません。

表 5-7 に、ドキュメント格納時の CLOB、NCLOB および BLOB データ型のメリットとデメリットを示します。

表 5-7 ドキュメント格納に関する LOB データ型の比較

データ型	メリット	デメリット
CLOB	■ 内容検索がサポートされています。 ■ 文字列操作がサポートされています。	■ データベース・キャラクタ・セットに依存します。 ■ 挿入時にデータ変換が必要です。 ■ バイナリ・ドキュメントは格納できません。
NCLOB	■ データベース・キャラクタ・セットに依存していません。 ■ 文字列操作がサポートされています。	■ 内容検索がサポートされていません。 ■ 挿入時にデータ変換が必要です。 ■ バイナリ・ドキュメントは格納できません。
BLOB	■ データベース・キャラクタ・セットに依存していません。 ■ 内容検索がサポートされています。 ■ データ変換がなく、データをそのまま格納できます。 ■ Microsoft Word や Microsoft Excel などのバイナリ・ドキュメントを格納できます。	■ 文字列操作がサポートされていません。

多言語ドキュメントの内容検索に使用する索引の作成

Oracle Text では、索引を作成して、CLOB および BLOB として格納されている多言語ドキュメントの内容を検索できます。言語固有のレクサーを使用して、CLOB や BLOB のデータを解析し、検索可能なキーワードのリストを生成します。

多言語ドキュメントを検索するには、マルチレクサーを作成します。マルチレクサーは、言語列に基づいて、行ごとに言語固有のレクサーを選択します。この項では、複数言語のドキュメントに対して索引を作成するための高水準な手順を説明します。この項の内容は、次のとおりです。

- マルチレクサーの作成
- CLOB として格納されたドキュメントに対する索引の作成
- BLOB として格納されたドキュメントに対する索引の作成

関連項目：『Oracle Text リファレンス』

マルチレクサーの作成

マルチレクサーを作成する最初の手順は、サポート対象言語ごとに言語固有のレクサー・プリファレンスを作成することです。次の例では、英語、ドイツ語および日本語のレクサーをPL/SQLプロシージャで作成しています。

```
ctx_ddl.create_preference('english_lexer', 'basic_lexer');
ctx_ddl.set_attribute('english_lexer','index_themes','yes');
ctx_ddl.create_preference('german_lexer', 'basic_lexer');
ctx_ddl.set_attribute('german_lexer','composite','german');
ctx_ddl.set_attribute('german_lexer','alternate_spelling','german');
ctx_ddl.set_attribute('german_lexer','mixed_case','yes');
ctx_ddl.create_preference('japanese_lexer', 'JAPANESE_VGRAM_LEXER');
```

作成された言語固有のレクサー・プリファレンスは、単一のマルチレクサー・プリファレンスの下に集める必要があります。最初に、MULTI_LEXERオブジェクトを使用して、次のようにマルチレクサー・プリファレンスを作成します。

```
ctx_ddl.create_preference('global_lexer','multi_lexer');
```

ここで、add_sub_lexerコールを使用して、言語固有のレクサーをマルチレクサー・プリファレンスに追加します。

```
ctx_ddl.add_sub_lexer('global_lexer', 'german', 'german_lexer');
ctx_ddl.add_sub_lexer('global_lexer', 'japanese', 'japanese_lexer');
ctx_ddl.add_sub_lexer('global_lexer', 'default','english_lexer');
```

これによって、german_lexerプリファレンスはドイツ語ドキュメントを処理し、japanese_lexerプリファレンスは日本語ドキュメントを処理し、english_lexerプリファレンスは、言語にDEFAULTを使用して、それ以外のすべてのドキュメントを処理するように指名されます。

CLOBとして格納されたドキュメントに対する索引の作成

マルチレクサーによって、表の言語列に基づいて、行ごとに使用するレクサーが決定されます。これは、テキスト列内のドキュメントの言語が格納されているキャラクタ列です。Oracleの言語名を使用して、この列にあるドキュメントの言語を識別します。たとえば、CLOBを使用してドキュメントを格納する場合は、言語列をそのドキュメントが格納されている表に追加します。

```
CREATE TABLE globaldoc
  (doc_id    NUMBER       PRIMARY KEY,
  language  VARCHAR2(30),
  text      CLOB);
```

この表に索引を作成するには、次のようにマルチレクサー・プリファレンスを使用して言語列名を指定します。

```
CREATE INDEX globalx ON globaldoc(text)
  indextype IS ctxsys.context
  parameters ('lexer
               global_lexer
               language
               column
               language');
```

BLOB として格納されたドキュメントに対する索引の作成

言語列の他に、キャラクタ・セットと書式列を、ドキュメントが格納されている表に追加する必要があります。キャラクタ・セット列には、Oracle キャラクタ・セット名を使用してドキュメントのキャラクタ・セットを格納します。書式列には、ドキュメントがテキスト・ドキュメントであるかバイナリ・ドキュメントであるかを指定します。たとえば、CREATE TABLE 文では、次のように列 characterset および format を指定できます。

```
CREATE TABLE globaldoc (
   doc_id       NUMBER       PRIMARY KEY,
   language     VARCHAR2(30),
   characterset VARCHAR2(30),
   format       VARCHAR2(10),
   text         BLOB
  );
```

ワードプロセッシングまたはスプレッドシートのドキュメントを表に入力し、format 列に binary を指定できます。HTML、XML およびテキスト・フォーマットのドキュメントの場合は、表にドキュメントを入力して、format 列に text を指定します。

キャラクタ・セットを指定する列があるため、様々なキャラクタ・セットによるテキスト・ドキュメントを格納できます。

索引の作成時に、書式列とキャラクタ・セット列の名前を指定します。

```
CREATE INDEX globalx ON globaldoc(text)
  indextype is ctxsys.context
  parameters ('filter inso_filter
               lexer global_lexer
               language column language
               format  column format
               charset column characterset');
```

全ドキュメントがテキスト・フォーマットの場合は、charset_filter を使用できます。charset_filter によって、データは charset 列で指定されたキャラクタ・セットからデータベース・キャラクタ・セットに変換されます。

6

Unicode を使用したプログラミング

この章では、Oracle のデータベース・アクセス製品と Unicode を併用する方法について説明します。この章の内容は、次のとおりです。

Unicode を使用したプログラミングの概要

Oracle9*i* には、Unicode データの挿入と取出しを行うためのデータベース・アクセス製品がいくつか用意されています。Oracle では、Java や C/C++ などの一般的に使用されているプログラミング環境に対応したデータベース・アクセス製品を提供しています。データはデータベースとクライアント・プログラム間で透過的に変換されるため、クライアント・プログラムがデータベース・キャラクタ・セットと各国語キャラクタ・セットの影響を受けないことが保証されます。さらに、クライアント・プログラムは、データベースで使用される NCHAR や CHAR などの文字データ型の影響も受けない場合があります。

データ変換操作によってデータベース・サーバーに負荷をかけないように、Oracle9*i* は常に、これらの操作をクライアント側のデータベース・アクセス製品に移動しようとします。データベースでデータ変換が必要な場合があり、この操作はパフォーマンスに影響します。この章では、データ変換の流れの詳細を説明します。

データベース・アクセス製品のスタックおよび Unicode

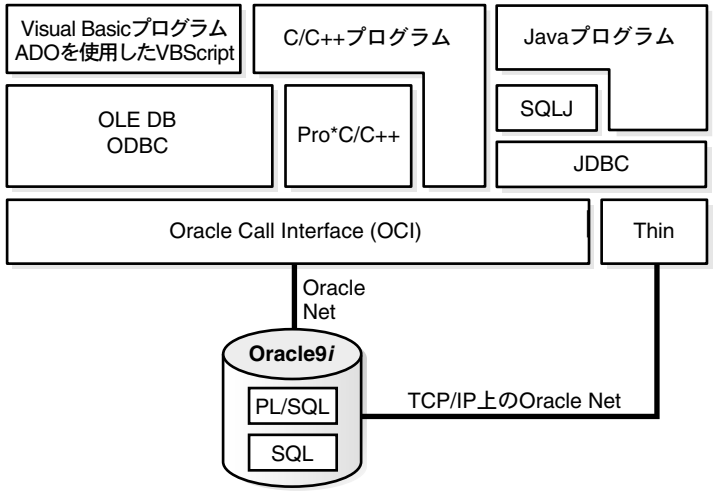
オラクル社では、データベース・アクセス製品の包括的なセットを提供しています。これらの製品によって、異なる開発環境のプログラムは、データベースに格納された Unicode データにアクセスできます。表 6-1 にこれらの製品を示します。

表 6-1 Oracle のデータベース・アクセス製品

プログラミング環境	Oracle のデータベース・アクセス製品
C/C++	Oracle Call Interface（OCI） Oracle Pro*C/C++ Oracle ODBC ドライバ Oracle OLE DB ドライバ
Visual Basic	Oracle ODBC ドライバ Oracle OLE DB ドライバ
Java	Oracle JDBC OCI または Thin ドライバ Oracle SQLJ
PL/SQL	Oracle PL/SQL および SQL

図 6-1 は、データベース・アクセス製品によるデータベースへのアクセス方法を示しています。

図 6-1 Oracle のデータベース・アクセス製品



OCI は、残りのクライアント側データベース・アクセス製品が使用する最下位レベルの API です。SQL CHAR データ型および NCHAR データ型で格納されている Unicode に、C/C++ プログラムでアクセスするための柔軟な方法を提供します。OCI を使用すると、データの挿入または取出しに使用するキャラクタ・セット（UTF-8 や UTF-16 など）をプログラムで指定できます。データベースには、Oracle Net を介してアクセスします。

Oracle Pro*C/C++ を使用すると、プログラム内に SQL および PL/SQL を埋め込むことができます。OCI の Unicode 機能を使用して、SQL CHAR データ型と NCHAR データ型に対する UTF-16 と UTF-8 のデータ・アクセスを提供します。

Oracle ODBC ドライバを使用すると、Windows プラットフォームで実行されている C/C++、Visual Basic および VBScript のプログラムで、データベースに SQL CHAR データ型および NCHAR データ型で格納されている Unicode データにアクセスできます。ODBC 標準仕様に指定されている SQLWCHAR インタフェースを実装することによって、UTF-16 データ・アクセスを提供します。

Oracle OLE DB ドライバを使用すると、Windows プラットフォームで実行されている C/C++、Visual Basic および VBScript のプログラムで、SQL CHAR データ型および NCHAR データ型で格納されている Unicode データにアクセスできます。ワイド文字列の OLE DB データ型を通して、UTF-16 データ・アクセスを提供します。

Oracle JDBC ドライバは、Oracle9*i* データベースにアクセスするための主要な Java プログラム・インタフェースです。クライアント側の JDBC ドライバは 2 つ用意されています。

- Java アプリケーションで使用され、OCI ライブラリを必要とする JDBC OCI ドライバ
- 純粋な Java ドライバで、Java アプレットで主に使用され、TCP/IP を介した Oracle Net プロトコルをサポートする JDBC Thin ドライバ

この 2 つのドライバは、データベースに格納されている SQL CHAR データ型および NCHAR データ型への Unicode データ・アクセスをサポートします。

Oracle SQLJ には、Java プログラム内の埋込み SQL を JDBC コールを使用した Java ソース・ファイルに変換するという、プリプロセッサのような役割があります。データベースにアクセスするための高度なプログラム・インタフェースを提供します。JDBC と同様に、SQLJ は、データベースに格納されている SQL CHAR データ型および NCHAR データ型への Unicode データ・アクセスを提供します。

PL/SQL と SQL のエンジンは、OCI などのクライアント側プログラムやサーバー側の PL/SQL ストアド・プロシージャのかわりに、PL/SQL プログラムと SQL 文を処理します。これらのエンジンを使用すると、PL/SQL プログラムで NCHAR 変数および NVARCHAR2 変数を宣言し、データベースの SQL NCHAR データ型にアクセスできます。

次の項では、各データベース・アクセス製品が、Oracle9*i* データベースへの Unicode データ・アクセスをどのようにサポートしているかを説明し、これらの製品の使用例を示します。

- Unicode を使用した SQL と PL/SQL のプログラミング
- Unicode を使用した OCI プログラミング
- Unicode を使用した Pro*C/C++ プログラミング
- Unicode を使用した JDBC と SQLJ のプログラミング
- Unicode を使用した ODBC と OLE DB のプログラミング

Unicode を使用した SQL と PL/SQL のプログラミング

SQL は、すべてのプログラムとユーザーが、Oracle データベースのデータに直接的または間接的にアクセスするときに使用する基本言語です。PL/SQL は、SQL のデータ操作能力と手続き型言語のデータ処理能力を結合した手続き型言語です。SQL と PL/SQL は両方とも、他のプログラム言語に埋め込むことができます。この項では、多言語アプリケーションに配布可能な SQL と PL/SQL の Unicode 関連機能について説明します。

この項の内容は、次のとおりです。

関連項目：

- 『Oracle9*i* SQL リファレンス』
- 『PL/SQL ユーザーズ・ガイドおよびリファレンス』

SQL NCHAR データ型

SQL NCHAR データ型は次の 3 つです。

NCHAR データ型

表の列または PL/SQL 変数を NCHAR データ型として定義する場合、長さは常に文字数として指定します。たとえば、次の文を考えます。

```
CREATE TABLE table1 (column1 NCHAR(30));
```

この文では、最大文字長が 30 の列が作成されます。この列の最大バイト数は、次のように決定されます。

```
maximum number of bytes = (maximum number of characters) x (maximum number of bytes
per character)
```

たとえば、各国語キャラクタ・セットが UTF8 の場合、最大バイト長は 30 文字に 1 文字当たり 3 バイトを乗算した値、つまり 90 バイトとなります。

すべての NCHAR データ型に使用する各国語キャラクタ・セットは、データベースの作成時に定義します。Oracle9*i* では、各国語キャラクタ・セットは UTF8 または AL16UTF16 のいずれかです。デフォルトは AL16UTF16 です。

使用可能な最大列サイズは、各国語キャラクタ・セットが UTF8 の場合は 2000 文字、AL16UTF16 の場合は 1000 文字です。実際のデータが最大バイト制限の 2000 の対象となります。2 つのサイズ制限は同時に満たす必要があります。PL/SQL では、NCHAR データの最大長は 32767 バイトです。NCHAR 変数は 32767 文字まで定義できますが、実際のデータは 32767 バイトを超えることはできません。列の長さより短い値を挿入すると、最大文字長と最大バイト長のいずれか小さいほうの値まで空白で埋められます。

注意： UTF8 は可変幅キャラクタ・セットであるため、パフォーマンスに影響する場合があります。NCHAR フィールドの空白埋めが過剰に行われると、パフォーマンスが低下します。NVARCHAR データ型を使用するか、NCHAR データ型のキャラクタ・セットを AL16UTF16 に変更することを考慮してください。

NVARCHAR2 データ型

NVARCHAR2 データ型は、各国語キャラクタ・セットを使用する可変長文字列を指定します。NVARCHAR2 列を使用して表を作成するときは、列の最大文字数を指定します。NVARCHAR2 の長さは、NCHAR の場合と同様に常に文字単位です。続いて、値が列の最大長を超えないかぎり、ユーザーが指定したとおりに各値が列内に格納されます。文字列の値が最大長まで埋め込まれることはありません。

使用可能な最大列サイズは、各国語キャラクタ・セットが UTF8 の場合は 4000 文字、AL16UTF16 の場合は 2000 文字です。NVARCHAR2 列の最大バイト長は 4000 です。バイト数の上限と文字数の上限の両方を満たす必要があるため、実際に NVARCHAR2 列に使用できる最大文字数は、4000 バイトに書込み可能な文字数となります。

PL/SQL では、NVARCHAR2 変数の最大長は 32767 バイトです。NVARCHAR2 変数は 32767 文字まで定義できますが、実際のデータは 32767 バイトを超えることはできません。

次の CREATE TABLE 文では、最大文字長が 2000 の NVARCHAR2 列を 1 つ含む表が作成されます。各国語キャラクタ・セットが UTF8 の場合、次の文では、最大文字長が 2000 で、最大バイト長が 4000 の列が作成されます。

```
CREATE TABLE table2 (column2 NVARCHAR2(2000));
```

NCLOB データ型

NCLOB は、最大 4GB のマルチバイト・キャラクタを格納する、キャラクタ・ラージ・オブジェクトです。BLOB とは異なり、NCLOB では完全なトランザクション・サポートがあるため、SQL、DBMS_LOB パッケージまたは OCI を使用して行われた変更は、完全にトランザクションに組み込まれます。NCLOB 値の操作は、コミットおよびロールバックできます。ただし、あるトランザクションで PL/SQL または OCI 変数に保存した NCLOB ロケータを別のトランザクションまたはセッションで使用することはできません。

NCLOB 値は、各国語キャラクタ・セットに関係なく、固定幅の AL16UTF16 キャラクタ・セットを使用してデータベースに格納されます。格納された Unicode 値は、クライアントまたはサーバーで要求された固定幅または可変幅のキャラクタ・セットに変換されます。可変幅のキャラクタ・セットを使用して NCLOB 列に挿入したデータは、データベースに格納される前に AL16UTF16 に変換されます。

> **関連項目：** NCLOB の詳細は、『Oracle9*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。

NCHAR データ型と他のデータ型の間の暗黙的な変換

Oracle では、SQL NCHAR データ型と、他の Oracle データ型（CHAR、VARCHAR2、NUMBER、DATE、ROWID および CLOB など）との間の暗黙的な変換がサポートされています。SQL NCHAR データ型については、CHAR および VARCHAR2 データ型に関する暗黙的な変換もサポートされています。SQL NCHAR データ型は、SQL CHAR データ型と同様に使用できます。

暗黙的な変換に関しては、次の事項に注意してください。

- SQL CHAR データ型と SQL NCHAR データ型の間の型変換では、データベース・キャラクタ・セットと各国語キャラクタ・セットが異なるときに、キャラクタ・セットの変換が行われる場合があります。ターゲット・データが CHAR または NCHAR の場合は、空白埋めが発生することがあります。
- CLOB データ型と NCLOB データ型の間の暗黙的な変換を行うことはできません。ただし、Oracle の明示的な変換関数 TO_CLOB および TO_NCLOB を使用できます。

> **関連項目：** 『Oracle9*i* SQL リファレンス』

データ型変換中のデータ消失に対する例外処理

キャラクタ・セット変換が必要なときは、データ型変換時にデータを消失する可能性があります。最初のキャラクタ・セット内の文字がターゲットのキャラクタ・セット内で定義されていない場合、その場所には置換文字が使用されます。たとえば、NCHAR データを通常の CHAR 列に挿入しようとしたときに、NCHAR（Unicode）フォーム内の文字データがデータベース・キャラクタ・セットに変換できない場合、その文字はデータベース・キャラクタ・セットで定義されている置換文字に置き換えられます。キャラクタ・タイプ変換時のデータ消失に関する対処方法は、NLS_NCHAR_CONV_EXCP 初期化パラメータによって制御されます。このパラメータが TRUE に設定されているときは、データ消失が発生する SQL 文は

ORA-12713 エラーを戻し、対応する操作は異常終了します。このパラメータが FALSE に設定されているときは、データ消失はレポートされず、変換不可能な文字は置換文字で置き換えられます。デフォルト値は TRUE です。このパラメータは、暗黙的な変換と明示的な変換の両方に対して機能します。

PL/SQL の場合は、SQL CHAR データ型と NCHAR データ型の変換時にデータ消失が発生すると、暗黙的な変換の場合も明示的な変換の場合も LOSSY_CHARSET_CONVERSION 例外が発生します。

暗黙的なデータ型変換の規則

状況によって、一方向のデータ型変換のみが可能な場合と、両方向のデータ型変換が可能な場合があります。Oracle では、データ型間の変換に関して一連の規則が定義されています。表 6-2 に、データ型間の変換に関する規則を示します。

表 6-2 データ型間の変換に関する規則

文	規則
INSERT/UPDATE 文	値は、ターゲット・データベース列のデータ型に変換されます。
SELECT INTO 文	データベースのデータは、ターゲット変数のデータ型に変換されます。
変数の代入	等号の右辺の値は、等号の左辺のターゲット変数のデータ型に変換されます。
SQL 関数と PL/SQL 関数のパラメータ	CHAR、VARCHAR2、NCHAR および NVARCHAR2 は同じ方法でロードされます。CHAR、VARCHAR2、NCHAR または NVARCHAR2 データ型の引数は、CHAR、VARCHAR2、NCHAR または NVARCHAR2 データ型のいずれかの仮パラメータと比較されます。引数と仮パラメータのデータ型が正確に一致していない場合は、データが関数の入口でパラメータにコピーされるとき、および関数の出口で引数にコピーアウトされるときに、暗黙的な変換が行われます。
連結（||）操作または CONCAT 関数	あるオペランドが SQL CHAR データ型または NCHAR データ型で、他のオペランドが NUMBER またはその他の非文字データ型の場合、他のデータ型は VARCHAR2 または NVARCHAR2 に変換されます。文字データ型間の連結については、6-9 ページの「SQL NCHAR データ型と SQL CHAR データ型」を参照してください。
SQL CHAR または NCHAR データ型と NUMBER データ型	文字の値は NUMBER データ型に変換されます。
SQL CHAR または NCHAR データ型と DATE データ型	文字の値は DATE データ型に変換されます。
SQL CHAR または NCHAR データ型と ROWID データ型	文字データ型は ROWID データ型に変換されます。
SQL NCHAR データ型と SQL CHAR データ型	文字の値は NUMBER データ型に変換されます。

表 6-2 データ型間の変換に関する規則（続き）

文	規則
SQL CHAR または NCHAR データ型と NUMBER データ型	文字の値は NUMBER データ型に変換されます。
SQL CHAR または NCHAR データ型と DATE データ型	文字の値は DATE データ型に変換されます。
SQL CHAR または NCHAR データ型と ROWID データ型	文字の値は ROWID データ型に変換されます。
SQL NCHAR データ型と SQL CHAR データ型	SQL NCHAR データ型と SQL CHAR データ型との比較はさらに複雑です。これは、これらのデータ型が異なるキャラクタ・セットでエンコードされている場合があるためです。 CHAR 値と VARCHAR2 値が比較される場合は、CHAR 値が VARCHAR2 値に変換されます。 NCHAR 値と NVARCHAR2 値が比較される場合は、NCHAR 値が NVARCHAR2 値に変換されます。 SQL NCHAR データ型と SQL CHAR データ型との間で比較が行われるとき、これらのデータ型が異なるキャラクタ・セットでエンコードされている場合は、キャラクタ・セット変換が行われます。SQL NCHAR データ型のキャラクタ・セットは常に Unicode で、UTF8 エンコーディングまたは AL16UTF16 エンコーディングのいずれかです。これらの文字レパートリは同じですが、Unicode 規格では異なるエンコーディングです。SQL CHAR データ型は、データベース・キャラクタ・セットを使用します。このデータベース・キャラクタ・セットは、Oracle がサポートするいずれかのキャラクタ・セットです。Unicode は Oracle でサポートされているキャラクタ・セットのスーパーセットであるため、SQL CHAR データ型から SQL NCHAR データ型への変換は、いつでもデータ消失なしに行うことができます。

Unicode データ型の SQL 関数

SQL NCHAR データ型は、明示的な変換関数を使用して、SQL CHAR データ型や他のデータ型と相互に変換できます。この項の例では、次の文で作成された表を使用します。

```
CREATE TABLE customers
  (id NUMBER, name NVARCHAR2(50), address NVARCHAR2(200), birthdate DATE);
```

例 6-1 TO_NCHAR 関数を使用した customers 表の移入

TO_NCHAR 関数は実行時にデータを変換しますが、N 関数はコンパイル時にデータを変換します。

```
INSERT INTO customers VALUES (1000,
  TO_NCHAR('John Smith'),N'500 Oracle Parkway',sysdate);
```

例 6-2 TO_CHAR 関数を使用した customers 表からの選択

次の文では、name の値が各国語キャラクタ・セットの文字からデータベース・キャラクタ・セットの文字に変換され、その後で LIKE 句に従って選択されます。

```
SELECT name FROM customers WHERE TO_CHAR(name) LIKE '%Sm%';
```

次の出力が表示されます。

```
NAME
-------------------------------------
John Smith
```

例 6-3 TO_DATE 関数を使用した customers 表からの選択

N 関数を使用すると、NCHAR または CHAR データ型のデータを TO_DATE 関数のパラメータとして渡すことができます。データ型変換は実行時に行われるため、データ型を混在させることができます。

```
DECLARE
ndatestring NVARCHAR2(20) := N'12-SEP-1975';
BEGIN
SELECT name into ndstr FROM customers
WHERE (birthdate)> TO_DATE(ndatestring, 'DD-MON-YYYY', N'NLS_DATE_LANGUAGE =
AMERICAN');
END;
```

例 6-3 に示したように、SQL NCHAR データ型のデータを明示的な変換関数に渡すことができます。複数の文字列パラメータを使用すると、SQL CHAR および NCHAR データ型のデータを混在させることができます。

> **関連項目：** SQL NCHAR データ型の明示的な変換関数の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。

その他の SQL 関数

大部分の SQL 関数は、SQL NCHAR データ型および混合した文字データ型の引数を取ることができます。戻されるデータ型は、最初の引数の型に基づきます。これらの関数に渡された NUMBER や DATE などの非文字列データ型は、VARCHAR2 に変換されます。次の例では、6-10 ページの「Unicode データ型の SQL 関数」で作成した customers 表を使用します。

例 6-4 INSTR 関数

```
SELECT INSTR(name, N'Sm', 1, 1) FROM customers;
```

例 6-5 CONCAT 関数

```
SELECT CONCAT(name,id) FROM customers;
```

id は NVARCHAR2 に変換された後、name と連結されます。

例 6-6 RPAD 関数

```
SELECT RPAD(name,100,' ') FROM customers;
```

出力は次のようになります。

```
RPAD(NAME,100,'')
----------------------------------------
John Smith
```

スペース文字 ' ' が NCHAR キャラクタ・セットの対応する文字に変換された後、表示の全長が 100 になるように name の右側に埋め込まれます。

関連項目：『Oracle9*i* SQL リファレンス』

Unicode 文字列リテラル

Unicode 文字列リテラルは、次の方法で SQL および PL/SQL に入力できます。

- 引用符で囲まれた文字列リテラルの前に接頭辞 N を付加します。この接頭辞によって、その後に続く文字列リテラルが NCHAR 文字列リテラルであることが明示的に示されます。たとえば、N'12-SEP-1975' は NCHAR 文字列リテラルです。
- 文字列リテラルを引用符で囲みます。Oracle では SQL NCHAR データ型への暗黙的な変換がサポートされているため、文字列リテラルは必要に応じて SQL NCHAR データ型に変換されます。

> **注意：** 文字列リテラルが問合せに含まれていて、その問合せが SQL*Plus などのクライアント側のツールを通して発行されると、すべての問合せは、クライアントのキャラクタ・セットでエンコードされた後、処理前にサーバーのデータベース・キャラクタ・セットに変換されます。そのため、文字列リテラルがサーバーのデータベース・キャラクタ・セットに変換できない場合は、データが消失する可能性があります。

- NCHR(*n*) SQL 関数を使用します。この関数は、各国語キャラクタ・セット（AL32UTF8 または AL16UTF16）内の *n* と等価のバイナリを持つ文字を戻します。複数の NCHR(*n*) 関数を連結した結果は、NVARCHAR2 データになります。このように、クライアントとサーバーのキャラクタ・セット変換を行わずに、NVARCHAR2 文字列を直接作成できます。たとえば、NCHR(32) は空白文字を表します。

  NCHR(*n*) は各国語キャラクタ・セットに関連付けられているため、結果値の移植性はその各国語キャラクタ・セットで実行されるアプリケーションに限定されます。この移植性が重要な場合は、UNISTR 関数を使用して移植性の制限を解除します。

- UNISTR(*string*) SQL 関数を使用します。UNISTR(*string*) は、文字列を取得して、それを Unicode に変換します。変換結果は、データベースの各国語キャラクタ・セットとなります。文字列内に escape *bbbb* を埋め込むことができます。このエスケープ記号は、その後に 16 進数 0x*bbbb* を付けて UTF-16 コード・ポイントの値を表します。たとえば、UNISTR('G\0061ry') は 'Gary' を表します。

最後の 2 つの方法を使用すると、すべての Unicode 文字列リテラルをエンコードできます。

NCHAR データを使用した UTL_FILE パッケージの使用

UTL_FILE パッケージは、Unicode 各国語キャラクタ・セットのデータを処理するために、Oracle9*i* で拡張されています。次の関数とプロシージャが追加されています。

- FOPEN_NCHAR

  この関数は、最大行サイズを指定して、入力用または出力用のファイルを Unicode でオープンします。この関数を使用すると、データベース・キャラクタ・セットではなく Unicode でテキスト・ファイルの読取り / 書込みを行うことができます。

- GET_LINE_NCHAR

  このプロシージャは、ファイル・ハンドルで識別されたオープン・ファイルからテキストを読み込み、そのテキストを出力バッファ・パラメータに入れます。このプロシージャを使用すると、データベース・キャラクタ・セットではなく Unicode でテキスト・ファイルを読み込むことができます。

- PUT_NCHAR

  このプロシージャは、バッファ・パラメータに格納されているテキスト文字列を、ファイル・ハンドルで識別されたオープン・ファイルに書き込みます。このプロシージャを使用すると、データベース・キャラクタ・セットではなく Unicode でテキスト・ファイルを書き込むことができます。

- PUT_LINE_NCHAR

  このプロシージャは、バッファ・パラメータに格納されているテキスト文字列を、ファイル・ハンドルで識別されたオープン・ファイルに書き込みます。このプロシージャを使用すると、データベース・キャラクタ・セットではなく Unicode でテキスト・ファイルを書き込むことができます。

- PUTF_NCHAR

  このプロシージャは、PUT_NCHAR プロシージャに書式指定が追加されたものです。このプロシージャを使用すると、データベース・キャラクタ・セットではなく Unicode でテキスト・ファイルを書き込むことができます。

  関連項目： UTL_FILE パッケージの詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

Unicode を使用した OCI プログラミング

OCI はデータベースにアクセスするための最下位レベルの API であるため、可能な最大のパフォーマンスを提供します。OCI で Unicode を使用するときは、次の内容を考慮してください。

- Unicode プログラミング用の OCIEnvNlsCreate() 関数
- OCI Unicode のコード変換
- OCI での NLS_LANG キャラクタ・セットが UTF8 または AL32UTF8 の場合
- OCI での SQL CHAR データ型のバインドと定義
- OCI での SQL NCHAR データ型のバインドと定義
- OCI での CLOB Unicode データおよび NCLOB Unicode データのバインドと定義

  関連項目： 第 8 章「グローバル環境での OCI プログラミング」

Unicode プログラミング用の OCIEnvNlsCreate() 関数

OCIEnvNlsCreate() 関数を使用して、OCI 環境の作成時に SQL CHAR のキャラクタ・セットと SQL NCHAR のキャラクタ・セットを指定します。これは OCIEnvCreate() 関数の拡張バージョンであり、2 つのキャラクタ・セット ID を指定できるように拡張された引数を取ります。OCI_UTF16ID UTF-16 キャラクタ・セット ID によって、Oracle9*i* リリース 1（9.0.1）で導入された Unicode モードが置き換えられています。次に例を示します。

```
OCIEnv *envhp;
status = OCIEnvNlsCreate((OCIEnv **)&envhp,
(ub4)0,
(void *)0,
(void *(*) ()) 0,
(void *(*) ()) 0,
(void(*) ()) 0,
(size_t) 0,
(void **)0,
(ub2)OCI_UTF16ID, /* Metadata and SQL CHAR character set */
(ub2)OCI_UTF16ID /* SQL NCHAR character set */);
```

Unicode モードでは、OCIEnvCreate() 関数とともに OCI_UTF16 フラグが使用されますが、このモードは使用されなくなりました。

SQL CHAR のキャラクタ・セットと SQL NCHAR のキャラクタ・セットの両方に OCI_UTF16ID を指定すると、すべてのメタデータのバインドされた定義済みデータは、UTF-16 でエンコードされます。メタデータには、SQL 文、ユーザー名、エラー・メッセージおよび列名が含まれます。そのため、継承される操作はいずれも NLS_LANG の設定に依存せず、すべてのメタテキスト・データ・パラメータ（text*）は、UTF-16 エンコーディングでは Unicode テキスト・データ型（utext*）とみなされます。

OCI_UTF16ID キャラクタ・セット ID を使用して OCIEnv() 関数を初期化するときに SQL 文を準備するには、（utext*）文字列を指定して OCIStmtPrepare() 関数をコールします。次の例は、Windows プラットフォームでのみ動作します。他のプラットフォームの場合は、wchar_t データ型を変更する必要があります。

```
const wchar_t sqlstr[] = L"SELECT * FROM ENAME=:ename";
...
OCIStmt* stmthp;
sts = OCIHandleAlloc(envh, (void **)&stmthp, OCI_HTYPE_STMT, 0,
NULL);
status = OCIStmtPrepare(stmthp, errhp,(const text*)sqlstr,
wcslen(sqlstr),
                        OCI_NTV_SYNTAX, OCI_DEFAULT);
```

OCIEnv() 関数はすでに UTF-16 のキャラクタ・セット ID を使用して初期化されているため、データをバインドして定義するために OCI_ATTR_CHARSET_ID 属性を設定する必要はありません。バインド変数名も UTF-16 文字列であることが必要です。

```
/* Inserting Unicode data */
OCIBindByName(stmthp1, &bnd1p, errhp, (const text*)L":ename",
(sb4)wcslen(L":ename"),
              (void *) ename, sizeof(ename), SQLT_STR, (void
*)&insname_ind,
              (ub2 *) 0, (ub2 *) 0, (ub4) 0, (ub4 *)0,
OCI_DEFAULT);
OCIAttrSet((void *) bnd1p, (ub4) OCI_HTYPE_BIND, (void *)
&ename_col_len,
           (ub4) 0, (ub4)OCI_ATTR_MAXDATA_SIZE, errhp);
...
/* Retrieving Unicode data */
OCIDefineByPos (stmthp2, &dfn1p, errhp, (ub4)1, (void *)ename,
              (sb4)sizeof(ename), SQLT_STR, (void *)0, (ub2 *)0,
(ub2*)0,
            (ub4)OCI_DEFAULT);
```

この操作は OCIExecute() 関数で実行されます。

関連項目： 8-3 ページ「OCIEnvNlsCreate()」

OCI Unicode のコード変換

クライアントとサーバーのキャラクタ・セットが異なる場合は、OCI クライアントとデータベース・サーバーの間で Unicode キャラクタ・セット変換が行われます。この変換は、環境によってクライアントまたはサーバーのいずれの側でも行われますが、通常はクライアント側で行われます。

データ整合性

OCI API を適切にコールしないと、変換時にデータが消失する可能性があります。サーバーとクライアントのキャラクタ・セットが異なる場合は、変換先キャラクタ・セットが変換元キャラクタ・セットより小さいセットであると、データが消失する可能性があります。両方のキャラクタ・セットが Unicode キャラクタ・セット（UTF8 や AL16UTF16 など）の場合、この潜在的な問題は回避できます。

SQL NCHAR データ型をバインドまたは定義するときは、OCI_ATTR_CHARSET_FORM 属性を SQLCS_NCHAR に設定する必要があります。この設定を行わないと、データはデータベース・キャラクタ・セットに変換されてから、各国語キャラクタ・セットとの間で変換されるため、データが消失する可能性があります。データが消失するのは、データベース・キャラクタ・セットが Unicode でない場合のみです。

Unicode 使用時の OCI パフォーマンスに与える影響

冗長なデータ変換は、OCI アプリケーションのパフォーマンス低下の原因となる場合があります。このような冗長な変換は、次の 2 つの場合に発生します。

- SQL CHAR データ型をバインドまたは定義して、OCI_ATTR_CHARSET_FORM 属性を SQLCS_NCHAR に設定すると、クライアントのキャラクタ・セットから各国語キャラクタ・セットへのデータ変換と、各国語キャラクタ・セットからデータベース・キャラクタ・セットへのデータ変換が行われます。データ消失はないと思われますが、必要な変換が 1 回の場合でも変換が 2 回行われます。
- SQL NCHAR データ型をバインドまたは定義して、OCI_ATTR_CHARSET_FORM を設定しないと、クライアントのキャラクタ・セットからデータベース・キャラクタ・セットへの変換と、データベース・キャラクタ・セットから各国語データベース・キャラクタ・セットへのデータ変換が行われます。最悪の場合、データベース・キャラクタ・セットがクライアントのキャラクタ・セットより小さいと、データが消失する可能性があります。

パフォーマンス上の問題を回避するために、ターゲット列のデータ型に基づいて OCI_ATTR_CHARSET_FORM を常に適切に設定する必要があります。ターゲットのデータ型が不明の場合に、バインドおよび定義を行うときは OCI_ATTR_CHARSET_FORM 属性を SQLCS_NCHAR に設定してください。

表 6-3 に、OCI のキャラクタ・セット変換を示します。

表 6-3 OCI のキャラクタ・セットの変換

OCI クライアント・バッファのデータ型	OCI_ATTR_CHARSET_FORM	データベース内のターゲット列のデータ型	変換元と変換先	コメント
utext	SQLCS_IMPLICIT	CHAR、VARCHAR2、CLOB	OCI での、UTF-16 とデータベース・キャラクタ・セット	予期しないデータ消失はありません。
utext	SQLCS_NCHAR	NCHAR、NVARCHAR2、NCLOB	OCI での、UTF-16 と各国語キャラクタ・セット	予期しないデータ消失はありません。
utext	SQLCS_NCHAR	CHAR、VARCHAR2、CLOB	OCI での、UTF-16 と各国語キャラクタ・セット データベース・サーバーでの、各国語キャラクタ・セットとデータベース・キャラクタ・セット	予期しないデータ消失はありませんが、各国語キャラクタ・セットを経由して変換が行われるため、パフォーマンスが低下する可能性があります。

表 6-3 OCI のキャラクタ・セットの変換（続き）

OCI クライアント・バッファのデータ型	OCI_ATTR_CHARSET_FORM	データベース内のターゲット列のデータ型	変換元と変換先	コメント
utext	SQLCS_IMPLICIT	NCHAR、NVARCHAR2、NCLOB	OCI での、UTF-16 とデータベース・キャラクタ・セット データベース・サーバーでの、データベース・キャラクタ・セットと各国語キャラクタ・セット	データベース・キャラクタ・セットが Unicode でない場合は、データが消失する可能性があります。
text	SQLCS_IMPLICIT	CHAR、VARCHAR2、CLOB	OCI での、NLS_LANG キャラクタ・セットとデータベース・キャラクタ・セット	予期しないデータ消失はありません。
text	SQLCS_NCHAR	NCHAR、NVARCHAR2、NCLOB	OCI での、NLS_LANG キャラクタ・セットと各国語キャラクタ・セット	予期しないデータ消失はありません。
text	SQLCS_NCHAR	CHAR、VARCHAR2、CLOB	OCI での、NLS_LANG キャラクタ・セットと各国語キャラクタ・セット データベース・サーバーでの、各国語キャラクタ・セットとデータベース・キャラクタ・セット	予期しないデータ消失はありませんが、各国語キャラクタ・セットを経由して変換が行われるため、パフォーマンスが低下する可能性があります。
text	SQLCS_IMPLICIT	NCHAR、NVARCHAR2、NCLOB	OCI での、NLS_LANG キャラクタ・セットとデータベース・キャラクタ・セット データベース・サーバーでの、データベース・キャラクタ・セットと各国語キャラクタ・セット	データベース・キャラクタ・セットを経由して変換が行われるため、データが消失する可能性があります。

OCI Unicode のデータ拡張

データ変換によってデータが拡張し、バッファがオーバーフローする原因となる場合があります。バインド操作では、OCI_ATTR_MAXDATA_SIZE 属性をサーバー上に拡張後のデータを格納するのに十分なサイズに設定する必要があります。この設定が困難な場合は、表スキーマの変更を考慮する必要があります。定義操作では、拡張後のデータに対して十分なバッファ領域をクライアント・アプリケーションで割り当てる必要があります。バッファのサイズは、拡張後のデータの最大長に設定してください。次の計算で最大バッファ長を見積もることができます。

1. 列データのバイト・サイズを取得します。
2. 取得したサイズに、クライアント・キャラクタ・セットの 1 文字当たりの最大バイト数を乗算します。

この方法は最も簡単で速い方法ですが、正確ではないためメモリーを浪費する可能性があります。この方法は、キャラクタ・セットのすべての組合せに適用できます。たとえば、UTF-16 データのバインドと定義の場合は、次の例でクライアント・バッファを計算します。

```
ub2 csid = OCI_UTF16ID;
oratext *selstmt = "SELECT ename FROM emp";
counter = 1;
...
OCIStmtPrepare(stmthp, errhp, selstmt, (ub4)strlen((char*)selstmt),
               OCI_NTV_SYNTAX, OCI_DEFAULT);
OCIStmtExecute ( svchp, stmthp, errhp, (ub4)0, (ub4)0,
                 (CONST OCISnapshot*)0, (OCISnapshot*)0,
                 OCI_DESCRIBE_ONLY);
OCIParamGet(stmthp, OCI_HTYPE_STMT, errhp, &myparam, (ub4)counter);
OCIAttrGet((void*)myparam, (ub4)OCI_DTYPE_PARAM, (void*)&col_width,
           (ub4*)0, (ub4)OCI_ATTR_DATA_SIZE, errhp);
...
maxenamelen = (col_width + 1) * sizeof(utext);
cbuf = (utext*)malloc(maxenamelen);
...
OCIDefineByPos(stmthp, &dfnp, errhp, (ub4)1, (void *)cbuf,
               (sb4)maxenamelen, SQLT_STR, (void *)0, (ub2 *)0,
               (ub2*)0, (ub4)OCI_DEFAULT);
OCIAttrSet((void *) dfnp, (ub4) OCI_HTYPE_DEFINE, (void *) &csid,
           (ub4) 0, (ub4)OCI_ATTR_CHARSET_ID, errhp);
OCIStmtFetch(stmthp, errhp, 1, OCI_FETCH_NEXT, OCI_DEFAULT);
...
```

OCI での NLS_LANG キャラクタ・セットが UTF8 または AL32UTF8 の場合

OCI のクライアント・アプリケーションに対して NLS_LANG を設定することで、UTF8 と AL32UTF8 を使用できます。補助文字が不要な場合は、UTF8 または AL32UTF8 のいずれを選択するかは問題ではありません。ただし、OCI アプリケーションで補助文字を処理する可能性がある場合は、決定する必要があります。UTF8 では各文字に 3 バイト必要な場合があるため、1 つの補助文字は 2 コード・ポイント、つまり合計 6 バイトで表されます。AL32UTF8 の場合、1 つの補助文字は 1 コード・ポイント、つまり合計 4 バイトで表されます。

AL16UTF16 はサーバー用の各国語キャラクタ・セットであるため、NLS_LANG を AL16UTF16 に設定しないでください。UTF-16 を使用する必要がある場合は、データをバインドまたは定義するときに、OCIAttrSet() 関数を使用してクライアント・キャラクタ・セットを OCI_UTF16ID に指定してください。

OCI での SQL CHAR データ型のバインドと定義

SQL CHAR データ型を使用してデータをバインドおよび定義するための Unicode キャラクタ・セットを指定するには、OCIBind() または OCIDefine() API 後に、OCIAttrSet() 関数をコールして、適切なキャラクタ・セット ID を設定する必要があります。次に一般的な 2 つの例を示します。

- OCIBind() または OCIDefine() の後に OCIAttrSet() をコールして、UTF-16 Unicode キャラクタ・セット・エンコーディングを指定します。次に例を示します。

```
...
ub2 csid = OCI_UTF16ID;
utext ename[100]; /* enough buffer for ENAME */
...
/* Inserting Unicode data */
OCIBindByName(stmthp1, &bnd1p, errhp, (oratext*)":ENAME",
              (sb4)strlen((char *)":ENAME"), (void *) ename, sizeof(ename),
              SQLT_STR, (void *)&insname_ind, (ub2 *) 0, (ub2 *) 0, (ub4) 0,
              (ub4 *)0, OCI_DEFAULT);
OCIAttrSet((void *) bnd1p, (ub4) OCI_HTYPE_BIND, (void *) &csid,
           (ub4) 0, (ub4)OCI_ATTR_CHARSET_ID, errhp);
OCIAttrSet((void *) bnd1p, (ub4) OCI_HTYPE_BIND, (void *) &ename_col_len,
           (ub4) 0, (ub4)OCI_ATTR_MAXDATA_SIZE, errhp);
...
/* Retrieving Unicode data */
OCIDefineByPos (stmthp2, &dfn1p, errhp, (ub4)1, (void *)ename,
                (sb4)sizeof(ename), SQLT_STR, (void *)0, (ub2 *)0,
                (ub2*)0, (ub4)OCI_DEFAULT);
OCIAttrSet((void *) dfn1p, (ub4) OCI_HTYPE_DEFINE, (void *) &csid,
           (ub4) 0, (ub4)OCI_ATTR_CHARSET_ID, errhp);
...
```

バインドされたバッファが utext データ型の場合は、OCIBind() または OCIDefine() のコール時にキャスト（text*）を追加する必要があります。OCI_ATTR_MAXDATA_SIZE 属性の値は通常、サーバーのキャラクタ・セットの列サイズによって決まります。これは、このサイズが、バインド操作の際に変換用の一時的なバッファ領域をサーバーに割り当てるためにのみ使用されているためです。

- NLS_LANG キャラクタ・セットを UTF8 または AL32UTF8 として指定して、OCIBind() または OCIDefine() をコールします。

  環境変数 NLS_LANG に UTF8 または AL32UTF8 を設定できます。Unicode を使用していない場合と同様に、OCIBind() および OCIDefine() をコールします。環境変数 NLS_LANG を UTF8 または AL32UTF8 に設定し、次の OCI プログラムを実行します。

```
...
oratext ename[100]; /* enough buffer size for ENAME */
...
/* Inserting Unicode data */
OCIBindByName(stmthp1, &bnd1p, errhp, (oratext*)":ENAME",
              (sb4)strlen((char *)":ENAME"), (void *) ename, sizeof(ename),
              SQLT_STR, (void *)&insname_ind, (ub2 *) 0, (ub2 *) 0,
              (ub4) 0, (ub4 *)0, OCI_DEFAULT);
OCIAttrSet((void *) bnd1p, (ub4) OCI_HTYPE_BIND, (void *) &ename_col_len,
           (ub4) 0, (ub4)OCI_ATTR_MAXDATA_SIZE, errhp);
...
/* Retrieving Unicode data */
OCIDefineByPos (stmthp2, &dfn1p, errhp, (ub4)1, (void *)ename,
                (sb4)sizeof(ename), SQLT_STR, (void *)0, (ub2 *)0, (ub2*)0,
                (ub4)OCI_DEFAULT);
...
```

OCI での SQL NCHAR データ型のバインドと定義

OCI 使用時は、UTF-16 のバインドまたは定義を使用して SQL NCHAR データ型にアクセスすることをお薦めします。Oracle9*i* から、SQL NCHAR データ型は、UTF8 または AL16UTF16 のいずれかのエンコーディングを使用した Unicode データ型になっています。SQL NCHAR データ型のデータにアクセスするには、データ消失が発生しない適切なデータ変換が実行されるように、バインドまたは定義してから実行するまでに OCI_ATTR_CHARSET_FORM 属性を SQLCS_NCHAR に設定します。SQL NCHAR データ型のデータ長は常に、Unicode コード・ポイントの数で表されます。

次のプログラムは、NCHAR データ列に対するデータの挿入とフェッチの一般的な例です。

```
...
ub2 csid = OCI_UTF16ID;
ub1 cform = SQLCS_NCHAR;
utext ename[100]; /* enough buffer for ENAME */
...
/* Inserting Unicode data */
OCIBindByName(stmthp1, &bnd1p, errhp, (oratext*)":ENAME",
              (sb4)strlen((char *)":ENAME"), (void *) ename,
              sizeof(ename), SQLT_STR, (void *)&insname_ind, (ub2 *) 0,
              (ub2 *) 0, (ub4) 0, (ub4 *)0, OCI_DEFAULT);
OCIAttrSet((void *) bnd1p, (ub4) OCI_HTYPE_BIND, (void *) &cform, (ub4) 0,
           (ub4)OCI_ATTR_CHARSET_FORM, errhp);
OCIAttrSet((void *) bnd1p, (ub4) OCI_HTYPE_BIND, (void *) &csid, (ub4) 0,
           (ub4)OCI_ATTR_CHARSET_ID, errhp);
OCIAttrSet((void *) bnd1p, (ub4) OCI_HTYPE_BIND, (void *) &ename_col_len,
           (ub4) 0, (ub4)OCI_ATTR_MAXDATA_SIZE, errhp);
...
/* Retrieving Unicode data */
OCIDefineByPos (stmthp2, &dfn1p, errhp, (ub4)1, (void *)ename,
                (sb4)sizeof(ename), SQLT_STR, (void *)0, (ub2 *)0, (ub2*)0,
                (ub4)OCI_DEFAULT);
OCIAttrSet((void *) dfn1p, (ub4) OCI_HTYPE_DEFINE, (void *) &csid, (ub4) 0,
           (ub4)OCI_ATTR_CHARSET_ID, errhp);
OCIAttrSet((void *) dfn1p, (ub4) OCI_HTYPE_DEFINE, (void *) &cform, (ub4) 0,
           (ub4)OCI_ATTR_CHARSET_FORM, errhp);
...
```

OCI での CLOB Unicode データおよび NCLOB Unicode データのバインドと定義

CLOB または NCLOB の列に対して UTF-16 データの書込み（バインド）および読込み（定義）を行うには、UTF-16 キャラクタ・セット ID を OCILobWrite() および OCILobRead() として指定する必要があります。UTF-16 データを CLOB 列に書き込むときは、OCILobWrite() を次のようにコールしてください。

```
...
ub2 csid = OCI_UTF16ID;
err = OCILobWrite (ctx->svchp, ctx->errhp, lobp, &amtp, offset, (void *) buf,
                   (ub4) BUFSIZE, OCI_ONE_PIECE, (void *)0,
                   (sb4 (*)()) 0, (ub2) csid, (ub1) SQLCS_IMPLICIT);
```

amtp パラメータは、Unicode コード・ポイント数でのデータ長です。offset パラメータは、データ列の開始からのデータのオフセットを示しています。csid パラメータは、UTF-16 データ用に設定する必要があります。

CLOB 列から UTF-16 データを読み込むには、OCILobRead() を次のようにコールします。

```
...
ub2 csid = OCI_UTF16ID;
err = OCILobRead(ctx->svchp, ctx->errhp, lobp, &amtp, offset, (void *) buf,
                 (ub4)BUFSIZE , (void *) 0, (sb4 (*)()) 0, (ub2)csid,
                 (ub1) SQLCS_IMPLICIT);
```

データ長は常に、Unicode コード・ポイントの数で表されます。エンコーディングが UTF-16 であるため、1 つの Unicode 補助文字は 2 コード・ポイントとしてカウントされます。LOB 列のバインドまたは定義後、OCILobGetLength() を使用して LOB 列に格納されているデータ長を測定できます。UTF-16 としてデータをバインドまたは定義している場合、戻り値はコード・ポイント数によるデータ長です。

```
err = OCILobGetLength(ctx->svchp, ctx->errhp, lobp, &lenp);
```

NCLOB を使用している場合は、OCI_ATTR_CHARSET_FORM を SQLCS_NCHAR に設定する必要があります。

Unicode を使用した Pro*C/C++ プログラミング

Pro*C/C++ には、データベースとの間で Unicode データの挿入や取出しを行う方法が 3 通り用意されています。

- Pro*C/C++ の VARCHAR データ型またはネイティブな C/C++ の text データ型を使用すると、UTF8 または AL32UTF8 データベースの SQL CHAR データ型で格納されている Unicode データにプログラムでアクセスできます。C/C++ のネイティブな text 型をプログラムで使用することもできます。
- Pro*C/C++ の UVARCHAR データ型またはネイティブな C/C++ の utext データ型を使用すると、データベースの NCHAR データ型で格納されている Unicode データにプログラムでアクセスできます。
- Pro*C/C++ の NVARCHAR データ型を使用すると、NCHAR データ型で格納されている Unicode データにプログラムでアクセスできます。Pro*C/C++ プログラムでの UVARCHAR と NVARCHAR の相違は、UVARCHAR データ型のデータは utext バッファに格納されているのに対して、NVARCHAR データ型のデータは text データ型で格納されている点です。

Pro*C/C++ では、SQL テキストに対して Unicode OCI API を使用しません。そのため、埋込み SQL テキストは、環境変数 NLS_LANG に指定されているキャラクタ・セットでエンコードする必要があります。

この項の内容は、次のとおりです。

- Pro*C/C++ での NVARCHAR データ型の使用
- Pro*C/C++ での UVARCHAR データ型の使用

Unicode での Pro*C/C++ データ変換

データ変換は OCI レイヤーで行われますが、Pro*C/C++ プログラムで使用されているデータ型に基づいて、どの変換手順を使用するかを OCI に対して指示するのは、Pro*C/C++ プリプロセッサです。表 6-4 に変換手順を示します。

表 6-4 Pro*C/C++ のバインドと定義でのデータ変換

Pro*C/C++ データ型	SQL データ型	変換手順
VARCHAR または text	CHAR	OCI で発生する、NLS_LANG キャラクタ・セットとデータベース・キャラクタ・セットの間の変換
VARCHAR または text	NCHAR	OCI で発生する、NLS_LANG キャラクタ・セットとデータベース・キャラクタ・セットの間の変換 データベース・サーバーで発生する、データベース・キャラクタ・セットと各国語キャラクタ・セットの間の変換
NVARCHAR	NCHAR	OCI で発生する、NLS_LANG キャラクタ・セットと各国語キャラクタ・セットの間の変換
NVARCHAR	CHAR	OCI で発生する、NLS_LANG キャラクタ・セットと各国語キャラクタ・セットの間の変換 データベース・サーバーでの、各国語キャラクタ・セットとデータベース・キャラクタ・セットの間の変換
UVARCHAR または utext	NCHAR	OCI で発生する、UTF-16 と各国語キャラクタ・セットの間の変換
UVARCHAR または utext	CHAR	OCI で発生する、UTF-16 と各国語キャラクタ・セットの間の変換 データベース・サーバーで発生する、各国語キャラクタ・セットからデータベース・キャラクタ・セットへの変換

Pro*C/C++ での VARCHAR データ型の使用

Pro*C/C++ の VARCHAR データ型は、length フィールドと text バッファ・フィールド付きの構造体に事前処理されます。次の例では、C/C++ のシステム固有のデータ型 text と Pro*C/C++ の VARCHAR データ型を使用して、表の列をバインドおよび定義します。

```
#include <sqlca.h>
main()
{
   ...
   /* Change to STRING datatype:    */
   EXEC ORACLE OPTION (CHAR_MAP=STRING) ;
   text ename[20] ;                  /* unsigned short type */
   varchar address[50] ;             /* Pro*C/C++ uvarchar type */

   EXEC SQL SELECT ename, address INTO :ename, :address FROM emp;
   /* ename is NULL-terminated */
   printf(L"ENAME = %s, ADDRESS = %.*s\n", ename, address.len, address.arr);
   ...
}
```

Pro*C/C++ プログラムで VARCHAR データ型またはシステム固有の text データ型を使用すると、プリプロセッサでは、そのプログラムがデータベースの SQL NCHAR データ型ではなく、SQL CHAR データ型の列にアクセスしているとみなします。次に、プリプロセッサは、SQLCS_IMPLICIT 値を使用してバインドまたは定義することで、この動きを反映するための C/C++ コードを OCI_ATTR_CHARSET_FORM 属性に対して生成します。その結果、バインド変数または定義変数がデータベースの SQL NCHAR データ型の列にバインドされると、データベース・サーバーでの暗黙的な変換によって、データベース・キャラクタ・セットと各国語データベース・キャラクタ・セットの間でデータの変換が行われます。データベース・キャラクタ・セットが各国語キャラクタ・セットより小さい場合は、変換時にデータが消失します。

Pro*C/C++ での NVARCHAR データ型の使用

Pro*C/C++ の NVARCHAR データ型は、Pro*C/C++ の VARCHAR データ型と類似しています。データベースの SQL NCHAR データ型にアクセスするために使用する必要があります。このデータ型は、SQL NCHAR データ型の列にテキスト・バッファをバインドまたは定義するように Pro*C/C++ プリプロセッサに通知します。プリプロセッサは、バインド変数または定義変数の OCI_ATTR_CHARSET_FORM 属性に SQLCS_NCHAR 値を指定します。この結果、データベースで暗黙的な変換は行われません。

SQL CHAR データ型の列に対して NVARCHAR バッファがバインドされると、（NLS_LANG キャラクタ・セットでエンコードされている）NVARCHAR バッファのデータは、OCI で各国語キャラクタ・セットとの変換が行われ、次に、データベース・サーバーでデータベース・キャラクタ・セットに変換されます。NLS_LANG キャラクタ・セットがデータベース・キャラクタ・セットより大きい場合は、データが消失する可能性があります。

Pro*C/C++ での UVARCHAR データ型の使用

UVARCHAR データ型は、length フィールドと utext バッファ・フィールド付きの構造体に事前処理されます。次のコード例には、2 つのホスト変数 ename と address が含まれています。ename ホスト変数は、Unicode 文字を 20 文字格納する utext バッファとして宣言されています。address ホスト変数は、Unicode 文字を 50 文字格納する uvarchar バッファとして宣言されており、len フィールドと arr フィールドは構造体のフィールドとしてアクセス可能です。

```
#include <sqlca.h>
#include <sqlucs2.h>

main()
{
   ...
   /* Change to STRING datatype:    */
   EXEC ORACLE OPTION (CHAR_MAP=STRING) ;
   utext ename[20] ;                  /* unsigned short type */
 uvarchar address[50] ;               /* Pro*C/C++ uvarchar type */

   EXEC SQL SELECT ename, address INTO :ename, :address FROM emp;
   /* ename is NULL-terminated */
wprintf(L"ENAME = %s, ADDRESS = %.*s\n", ename, address.len,
address.arr);
...
}
```

Pro*C/C++ プログラムで UVARCHAR データ型またはシステム固有の utext データ型を使用すると、プリプロセッサでは、プログラムが SQL NCHAR データ型にアクセスしているとみなします。次に、プリプロセッサは、SQLCS_NCHAR 値を使用してバインドまたは定義することで、OCI_ATTR_CHARSET_FORM 属性に対する C/C++ コードを生成します。この結果、バインド変数または定義変数が SQL NCHAR データ型の列にバインドされると、データベース・サーバーでは各国語キャラクタ・セットからのデータの暗黙的な変換が行われます。ただし、各国語キャラクタ・セットは常にデータベース・キャラクタ・セットより大きいため、この状況でデータが消失することはありません。

Unicode を使用した JDBC と SQLJ のプログラミング

Oracle には、データベースの Unicode データに Java プログラムでアクセスするための JDBC ドライバが 3 つ用意されています。

- JDBC OCI ドライバ
- JDBC Thin ドライバ
- JDBC KPRB ドライバ

Java プログラムでは、SQL CHAR データ型および NCHAR データ型の列との間で Unicode データの挿入や取出しを行うことができます。特に、JDBC を使用すると、Java プログラムで Java 文字列を SQL CHAR データ型および NCHAR データ型にバインドしたり定義できます。Java の `string` データ型は UTF-16 でエンコードされているため、データベースとの間で取り出したり挿入するデータは、UTF-16 とデータベース・キャラクタ・セットまたは各国語キャラクタ・セット間で変換する必要があります。SQLJ プリプロセッサを使用すると、Java プログラムに SQL 文を埋め込み、データベース・アクセス・コードを簡素化できます。このプリプロセッサは、Java プログラムの埋込み SQL 文を、対応する JDBC コールに変換します。JDBC と同様に、SQLJ を使用すると、プログラムで Java 文字列を SQL CHAR 列または NCHAR 列にバインドしたり定義できます。JDBC と SQLJ を使用すると、PL/SQL 文と SQL 文を Java 文字列に指定できるため、非 ASCII スキーマ・オブジェクト名を Java プログラムで参照できます。

この項の内容は、次のとおりです。

関連項目： 第 9 章「グローバル環境での Java プログラミング」

Unicode での Java 文字列のバインドと定義

Oracle JDBC ドライバを使用すると、Java 文字列のバインド変数または定義変数を使用して、データベースの SQL CHAR データ型にアクセスできます。次のコードは、Java 文字列を CHAR 列にバインドまたは定義する方法を示しています。

```
int empno = 12345;
String ename = "Joe"
PreparedStatement pstmt = conn.prepareStatement("INSERT INTO" +
    "emp (ename, empno) VALUES (?, ?)");
pstmt.setString(1, ename);
pstmt.setInt(2, empno);
pstmt.execute();                /* execute to insert into first row */
empno += 1;                     /* next employee number */
ename = "\uFF2A\uFF4F\uFF45";    /* Unicode characters in name */
pstmt.setString(1, ename);
pstmt.setInt(2, empno);
pstmt.execute();                /* execute to insert into second row */
```

Java 文字列の変数を SQL NCHAR データ型にバインドまたは定義するために、Oracle は JDBC 仕様を拡張して PreparedStatement.setFormOfUse() メソッドを追加しています。このメソッドを使用して、バインド変数のターゲット列が SQL NCHAR データ型であることを明示的に指定できます。次のコードは、Java 文字列を NCHAR 列にバインドする方法を示しています。

```
int empno = 12345;
String ename = "Joe"
oracle.jdbc.OraclePreparedStatement pstmt =
    (oracle.jdbc.OraclePreparedStatement)
    conn.prepareStatement("INSERT INTO emp (ename, empno) VALUES (?, ?)");
pstmt.setFormOfUse(1, oracle.jdbc.OraclePreparedStatement.FORM_NCHAR);
pstmt.setString(1, ename);
pstmt.setInt(2, empno);
pstmt.execute();                /* execute to insert into first row */
empno += 1;                     /* next employee number */
ename = "\uFF2A\uFF4F\uFF45";    /* Unicode characters in name */
pstmt.setString(1, ename);
pstmt.setInt(2, empno);
pstmt.execute();                /* execute to insert into second row */
```

setFormOfUse() で引数を明示的に指定しなくても、NCHAR 列に対して Java 文字列をバインドまたは定義できます。その場合は、次のことに注意してください。

- setString() メソッドに引数を指定しないと、JDBC はバインド変数または定義変数が SQL CHAR 列用であるとみなします。その結果、JDBC はこれらの変数をデータベース・キャラクタ・セットに変換しようとします。データがデータベースに到着すると、そのデータベースはデータベース・キャラクタ・セットのデータを各国語キャラクタ・セットに暗黙的に変換します。データベース・キャラクタ・セットが各国語キャラクタ・セットのサブセットの場合は、この変換時にデータが消失する可能性があります。各国語キャラクタ・セットは UTF8 または AL16UTF16 であるため、データベース・キャラクタ・セットが UTF8 または AL32UTF8 以外の場合、データは消失します。
- SQL CHAR データ型から SQL NCHAR データ型への暗黙的な変換はデータベースで行われるため、データベースのパフォーマンスが低下します。

また、SQL CHAR データ型の列に対して Java 文字列をバインドまたは定義し、setFormOfUse() で引数を指定した場合でも、データベースのパフォーマンスが低下します。ただし、各国語キャラクタ・セットは常にデータベース・キャラクタ・セットより大きいため、データが消失することはありません。

Unicode での Java データ変換

Java 文字列は常に UTF-16 でエンコードされているため、JDBC ドライバは、データベース・キャラクタ・セットから UTF-16 または各国語キャラクタ・セットにデータを透過的に変換します。変換手順は、3 つの JDBC ドライバでそれぞれ異なります。

- OCI ドライバ用のデータ変換
- Thin ドライバ用のデータ変換
- JDBC ドライバ用のデータ変換

OCI ドライバ用のデータ変換

OCI ドライバの場合、SQL 文は常に、処理するためにデータベースに送信される前に、ドライバによってデータベース・キャラクタ・セットに変換されます。Java string のバインド変数または定義変数については、表 6-5 に様々な状況に対応する変換手順が要約されています。

表 6-5 OCI ドライバの変換手順

使用フォーム	SQL データ型	変換手順
`Const.CHAR`（デフォルト）	`CHAR`	JDBC ドライバで発生する、Java 文字列とデータベース・キャラクタ・セットの間の変換
`Const.CHAR`（デフォルト）	`NCHAR`	JDBC ドライバで発生する、Java 文字列とデータベース・キャラクタ・セットの間の変換 データベース・サーバーで発生する、データベース・キャラクタ・セットと各国語キャラクタ・セットの間のデータ変換
`Const.NCHAR`	`NCHAR`	JDBC ドライバで発生する、Java 文字列と各国語キャラクタ・セットの間の変換
`Const.NCHAR`	`CHAR`	JDBC ドライバで発生する、Java 文字列と各国語キャラクタ・セットの間の変換 データベース・サーバーで発生する、各国語キャラクタ・セットとデータベース・キャラクタ・セットの間のデータ変換

Thin ドライバ用のデータ変換

Thin ドライバの場合、SQL 文は常に、処理するためにデータベースに送信される前に、ドライバによってデータベース・キャラクタ・セットまたは UTF-8 のいずれかに変換されます。また、Thin ドライバは、処理前に SQL 文をさらに変換する必要があることをデータベースに通知します。その場合、データベースは SQL 文をデータベース・キャラクタ・セットに変換します。Java `string` のバインド変数または定義変数の場合、Thin ドライバに対しては、表 6-6 に示されている変換手順が使用されます。

表 6-6 Thin ドライバの変換手順

使用フォーム	SQL データ型	データベース・キャラクタ・セット	変換手順
Const.CHAR（デフォルト）	CHAR	US7ASCII または WE8ISO8859P1	Thin ドライバで発生する、Java 文字列とデータベース・キャラクタ・セットの間の変換
Const.CHAR（デフォルト）	NCHAR	US7ASCII または WE8ISO8859P1	Thin ドライバで発生する、Java 文字列とデータベース・キャラクタ・セットの間の変換
			データベース・サーバーで発生する、データベース・キャラクタ・セットと各国語キャラクタ・セットの間のデータ変換
Const.CHAR（デフォルト）	CHAR	非 ASCII および非 WE8ISO8859P1	Thin ドライバで発生する、Java 文字列と UTF-8 の間の変換
			データベース・サーバーで発生する、UTF-8 とデータベース・キャラクタ・セットの間のデータ変換
Const.CHAR（デフォルト）	CHAR	非 ASCII および非 WE8ISO8859P1	Thin ドライバで発生する、Java 文字列と UTF-8 の間の変換
			データベース・サーバーで発生する、UTF-8 と各国語キャラクタ・セットの間のデータ変換
Const.NCHAR	CHAR		Thin ドライバで発生する、Java 文字列と各国語キャラクタ・セットの間の変換
			データベース・サーバーで発生する、各国語キャラクタ・セットとデータベース・キャラクタ・セットの間のデータ変換
Const.NCHAR	NCHAR		Thin ドライバで発生する、Java 文字列と各国語キャラクタ・セットの間の変換

JDBC ドライバ用のデータ変換

サーバー側 JDBC 内部ドライバはサーバーで動作し、変換はすべてデータベース・サーバーで行われます。Java 文字列として指定された SQL 文は、データベース・キャラクタ・セットに変換されます。setFormOfUse() で引数が指定されていない場合、Java `string` のバインド変数または定義変数は、データベース・キャラクタ・セットに変換されます。引数が指定されている場合は、各国語キャラクタ・セットに変換されます。

Unicode を使用した ODBC と OLE DB のプログラミング

Windows プラットフォームの使用時に Oracle9*i* にアクセスするには、Oracle の ODBC および OLE DB ドライバを使用する必要があります。この項では、これらのドライバによる Unicode のサポート方法を説明します。この項の内容は、次のとおりです。

ODBC と OLE DB の Unicode 対応ドライバ

Oracle の ODBC と OLE DB ドライバは、データを消失することなく、Unicode データを正しく処理できます。たとえば、日本語フォントと日本語文字を入力するための IME をインストールしている場合は日本語データが含まれた Unicode ODBC アプリケーションを英語版 Windows で実行できます。

Oracle9*i* では、Windows プラットフォーム固有の ODBC ドライバと OLE DB ドライバのみが提供されています。UNIX プラットフォームについては、ベンダーにお問い合せください。

Unicode での OCI 依存性

Unicode データを処理するために、ODBC ドライバと OLE DB ドライバは、OCI Unicode のバインド機能と定義機能を使用します。OCI Unicode データのバインド機能と定義機能は NLS_LANG の影響を受けません。つまり、プラットフォームの NLS_LANG 設定に関係なく、Unicode データは正しく処理されます。

関連項目： 6-13 ページ「Unicode を使用した OCI プログラミング」

Unicode での ODBC と OLE DB のコード変換

通常は、サーバーと異なるデータ型をクライアントのデータ型に指定しないかぎり、冗長なデータ変換は発生しません。たとえば、Unicode バッファ SQL_C_WCHAR を NCHAR などの Unicode データ列にバインドすると、ODBC ドライバと OLE DB ドライバは、アプリケーションと OCI レイヤー間の変換を行いません。

フェッチする前にデータ型を指定せずに、クライアントのデータ型で SQLGetData をコールすると、表 6-7 の変換が行われます。

表 6-7 ODBC の暗黙的なバインド・コード変換

ODBC クライアント・バッファのデータ型	データベース内のターゲット列のデータ型	フェッチ時の変換	コメント
SQL_C_WCHAR	CHAR、VARCHAR2、CLOB	データベース・キャラクタ・セットが NLS_LANG キャラクタ・セットのサブセットである場合、変換は次の順序で行われます。 ■ データベース・キャラクタ・セット ■ NLS_LANG ■ OCI での UTF-16 ■ ODBC での UTF-16	予期しないデータ消失はありません。 データベース・キャラクタ・セットが NLS_LANG キャラクタ・セットのサブセットである場合は、パフォーマンスが低下する可能性があります。
SQL_C_CHAR	CHAR、VARCHAR2、CLOB	データベース・キャラクタ・セットが NLS_LANG キャラクタ・セットのサブセットの場合 : OCI での、データベース・キャラクタ・セットから NLS_LANG への変換 データベース・キャラクタ・セットが NLS_LANG キャラクタ・セットのサブセットでない場合 : OCI と ODBC での、データベース・キャラクタ・セット、UTF-16 から NLS_LANG キャラクタ・セットへの変換	予期しないデータ消失はありません。 データベース・キャラクタ・セットが NLS_LANG キャラクタ・セットのサブセットでない場合は、パフォーマンスが低下する可能性があります。

挿入操作と更新操作のためには、データ型を指定する必要があります。

ODBC クライアント・バッファのデータ型は、SQLGetData のコール時に提供されますが、即時ではありません。このため、SQLFetch には情報はありません。

ODBC ドライバではデータ整合性が保証されるため、暗黙的なバインドを実行すると、冗長な変換が行われ、パフォーマンスが低下する場合があります。明示的なバインドでパフォーマンスを優先するか、暗黙的なバインドで利便性を優先するかのいずれかを選択します。

OLE DB のコード変換

ODBC とは異なり、OLE DB では、データの挿入、更新およびフェッチに対する暗黙的なバインドのみ実行できます。中間のキャラクタ・セットを決定するための変換アルゴリズムは、ODBC の暗黙的なバインドの場合と同じです。

表 6-8 OLE DB の暗黙的なバインド

OLE_DB クライアント・バッファのデータ型	データベース内のターゲット列のデータ型	インバインドとアウトバインドの変換	コメント
DBTYPE_WCHAR	CHAR、VARCHAR2、CLOB	データベース・キャラクタ・セットが NLS_LANG キャラクタ・セットのサブセットの場合: OCI での、データベース・キャラクタ・セットと NLS_LANG キャラクタ・セットの間の変換 OLE DB での、NLS_LANG キャラクタ・セットから UTF-16 への変換	予期しないデータ消失はありません。 データベース・キャラクタ・セットが NLS_LANG キャラクタ・セットのサブセットである場合は、パフォーマンスが低下する可能性があります。
		データベース・キャラクタ・セットが NLS_LANG キャラクタ・セットのサブセットでない場合: OCI での、データベース・キャラクタ・セットと UTF-16 の間の変換	
DBTYPE_CHAR	CHAR、VARCHAR2、CLOB	データベース・キャラクタ・セットが NLS_LANG キャラクタ・セットのサブセットの場合: OCI での、データベース・キャラクタ・セットと NLS_LANG の間の変換	予期しないデータ消失はありません。 データベース・キャラクタ・セットが NLS_LANG キャラクタ・セットのサブセットでない場合は、パフォーマンスが低下する可能性があります。
		データベース・キャラクタ・セットが NLS_LANG キャラクタ・セットのサブセットでない場合: OCI での、データベース・キャラクタ・セットと UTF-16 の間の変換 OLE DB での、UTF-16 から NLS_LANG キャラクタ・セットへの変換	

ODBC の Unicode データ型

ODBC の Unicode アプリケーションでは、SQLWCHAR を使用して Unicode データを格納します。SQLWCHAR のデータ操作には、すべての標準的な Windows Unicode 関数を使用できます。たとえば、wcslen は SQLWCHAR データの文字数をカウントします。

```
SQLWCHAR sqlStmt[] = L"select ename from emp";
len = wcslen(sqlStmt);
```

Microsoft の ODBC 3.5 仕様には、クライアントの SQL_C_WCHAR、SQL_C_WVARCHAR および SQL_WLONGVARCHAR に対して 3 つの Unicode データ型識別子が定義され、サーバーの SQL_WCHAR、SQL_WVARCHAR および SQL_WLONGVARCHAR に対して 3 つの Unicode データ型識別子が定義されています。

バインド操作の場合は、SQLBindParameter を使用して、クライアントとサーバーの両方にデータ型を指定します。次は Unicode バインドの例です。この例のクライアント・バッファ Name は、Unicode データ（SQL_C_WCHAR）が Unicode 列（SQL_WCHAR）に関連付けられている最初のバインド変数にバインドされていることを示しています。

```
SQLBindParameter(StatementHandle, 1, SQL_PARAM_INPUT, SQL_C_WCHAR,
SQL_WCHAR, NameLen, 0, (SQLPOINTER)Name, 0, &Name);
```

表 6-9 に、SQL NCHAR データ型に対するサーバーの ODBC Unicode データ型のデータ型マッピングを示します。

表 6-9 サーバーの ODBC Unicode データ型マッピング

ODBC データ型	Oracle データ型
SQL_WCHAR	NCHAR
SQL_WVARCHAR	NVARCHAR2
SQL_WLONGVARCHAR	NCLOB

SQL_WCHAR、SQL_WVARCHAR および SQL_WLONGVARCHAR は、ODBC 仕様に従って Unicode データとして処理されます。したがって、これらのデータ型はバイト数ではなく文字数で測定されます。

OLE DB の Unicode データ型

OLE DB では、Unicode のクライアント C データ型に対して wchar_t *、BSTR および OLESTR データ型が用意されています。実際には、wchar_t が最も一般的なデータ型で、他のデータ型は特定の目的に使用されます。次の例では、静的な SQL 文が割り当てられます。

```
wchar_t *sqlStmt = OLESTR("SELECT ename FROM emp");
```

OLESTR マクロは、"L" 修飾子と同様に機能して Unicode 文字列を示します。OLESTR を使用して Unicode データ・バッファを動的に割り当てる必要がある場合は、IMalloc アロケート関数（例 : CoTaskMemAlloc）を使用します。ただし、OLESTR の使用は、可変長データに対する通常の方法ではありません。一般的な文字列型には、かわりに wchar_t* を使用してください。BSTR も機能は同様です。BSTR は、文字列の前のメモリー位置に長さの接頭辞が付いている文字列です。一部の関数とメソッドでは、BSTR Unicode データ型のみが受け入れられます。したがって、割当てを行う SysAllocString やメモリーを解放する SysFreeString などの特別な関数では、BSTR Unicode 文字列を操作する必要があります。

ODBC とは異なり、OLE DB ではサーバーのデータ型を明示的に指定することはできません。クライアントのデータ型を設定すると、OLE DB ドライバは、必要に応じて自動的にデータ変換を実行します。

表 6-10 に、OLE DB のデータ型マッピングを示します。

表 6-10 OLE DB のデータ型マッピング

OLE DB データ型	Oracle データ型
DBTYPE_WCHAR	NCHAR または NVARCHAR2

DBTYPE_BSTR が指定されていると、DBTYPE_WCHAR であるとみなされます。これは、両方とも Unicode 文字列であるためです。

ADO アクセス

ADO は、OLE DB ドライバと ODBC ドライバを使用してデータベースにアクセスするための高水準 API です。ほとんどのデータベース・アプリケーション開発者が、Windows 上で ADO インタフェースを使用しています。これは、このインタフェースが、Internet Information Server（IIS）用の Active Server Pages（ASP）に対する主要なスクリプト言語である Visual Basic から簡単にアクセスできるためです。OLE DB および ODBC ドライバに対しては、ADO は単なる OLE DB コンシューマまたは ODBC アプリケーションです。ADO は、OLE DB および ODBC ドライバが Unicode 認識コンポーネントであることを前提としているため、常に Unicode データを操作しようとします。

7

グローバル環境での SQL と PL/SQL のプログラミング

この章では、グローバリゼーション・サポート環境での SQL プログラミングに役立つ情報について説明します。この章の内容は、次のとおりです。

- オプションの NLS パラメータを伴うロケール依存の SQL 関数
- ロケール依存のその他の SQL 関数
- グローバル環境での SQL と PL/SQL のプログラミングに関するその他のトピック

オプションの NLS パラメータを伴うロケール依存の SQL 関数

動作をグローバリゼーション・サポートの規則に依存しているすべての SQL 関数で、NLS パラメータを指定できます。これらの関数は、次のとおりです。

- TO_CHAR
- TO_DATE
- TO_NUMBER
- NLS_UPPER
- NLS_LOWER
- NLS_INITCAP
- NLSSORT

これらの関数の NLS パラメータを明示的に指定すると、セッションの NLS パラメータに依存せずに関数を評価できます。この機能は、数字や日付が文字列リテラルとして含まれている SQL 文で重要となる場合があります。

たとえば、次の問合せは、日付に対する言語が AMERICAN に指定されている場合、正しく評価されます。

```
SELECT last_name FROM employees WHERE hire_date > '01-JAN-1999';
```

このような問合せを現行の日付言語に依存しないようにするには、次のような文を使用します。

```
SELECT last_name FROM employees WHERE hire_date >
TO_DATE('01-JAN-1999','DD-MON-YYYY', 'NLS_DATE_LANGUAGE = AMERICAN');
```

このようにして、セッションの言語に依存しない SQL 文を、必要に応じて定義できます。文字列リテラルが、ビュー、CHECK 制約またはトリガー内の SQL 文にある場合には、このような文の指定が必要となる場合があります。

すべての文字関数は、シングルバイトとマルチバイトの両方の文字をサポートします。単位を明示的に示した場合を除いて、文字関数は、バイト単位ではなく文字単位で動作します。

これ以降の内容は、次のとおりです。

SQL 関数の NLS パラメータのデフォルト値

SQL 関数でビューやトリガーが評価される場合、NLS 関数パラメータには現行のセッションからのデフォルト値が使用されます。SQL 関数で CHECK 制約が評価される場合は、データベースの作成時に NLS パラメータに指定されたデフォルト値が使用されます。

SQL 関数の NLS パラメータの指定

NLS パラメータは、SQL 関数で次のように指定できます。

```
'parameter = value'
```

次に例を示します。

```
'NLS_DATE_LANGUAGE = AMERICAN'
```

SQL 関数では、次の NLS パラメータを指定できます。

- NLS_DATE_LANGUAGE
- NLS_NUMERIC_CHARACTERS
- NLS_CURRENCY
- NLS_ISO_CURRENCY
- NLS_SORT

表 7-1 に、特定の SQL 関数に有効な NLS パラメータを示します。

表 7-1 SQL 関数と有効な NLS パラメータ

SQL 関数	有効な NLS パラメータ
TO_DATE	NLS_DATE_LANGUAGE NLS_CALENDAR
TO_NUMBER	NLS_NUMERIC_CHARACTERS NLS_CURRENCY NLS_DUAL_CURRENCY NLS_ISO_CURRENCY
TO_CHAR	NLS_DATE_LANGUAGE NLS_NUMERIC_CHARACTERS NLS_CURRENCY NLS_ISO_CURRENCY NLS_DUAL_CURRENCY NLS_CALENDAR

表 7-1 SQL 関数と有効な NLS パラメータ（続き）

SQL 関数	有効な NLS パラメータ
TO_NCHAR	NLS_DATE_LANGUAGE NLS_NUMERIC_CHARACTERS NLS_CURRENCY NLS_ISO_CURRENCY NLS_DUAL_CURRENCY NLS_CALENDAR
NLS_UPPER	NLS_SORT
NLS_LOWER	NLS_SORT
NLS_INITCAP	NLS_SORT
NLSSORT	NLS_SORT

次の例に、SQL 関数で NLS パラメータを使用する方法を示します。

```
TO_DATE ('1-JAN-99', 'DD-MON-YY',
   'nls_date_language = American')

TO_CHAR (hire_date, 'DD/MON/YYYY',
   'nls_date_language = French')

TO_NUMBER ('13.000,00', '99G999D99',
   'nls_numeric_characters = '',.''')

TO_CHAR (salary, '9G999D99L', 'nls_numeric_characters = '',.''
   nls_currency = '' Dfl''')

TO_CHAR (salary, '9G999D99C', 'nls_numeric_characters = ''.,''
   nls_iso_currency = Japan')

NLS_UPPER (last_name, 'nls_sort = Swiss')

NLSSORT (last_name, 'nls_sort = German')
```

注意： 一部の言語では、様々な小文字が複数の大文字に、またはその逆に対応する場合もあります。その結果、NLS_UPPER、NLS_LOWER および NLS_INITCAP 関数の出力の長さが入力の長さと異なる場合があります。

関連項目： 4-11 ページ「特殊な大文字」および 4-12 ページ「特殊な小文字」

SQL 関数で受け入れられない NLS パラメータ

次の NLS パラメータは、NLSSORT を除く SQL 関数では受け入れられません。

- NLS_LANGUAGE
- NLS_TERRITORY
- NLS_DATE_FORMAT

NLS_DATE_FORMAT は必要な書式マスクを妨げる可能性があるため、パラメータとしては受け入れられません。TO_CHAR 関数または TO_DATE 関数に NLS パラメータがある場合は、常に、日付書式を指定する必要があります。このため、NLS_DATE_FORMAT は、TO_CHAR または TO_DATE 関数に対する有効な NLS パラメータではありません。

TO_CHAR、TO_NUMBER または TO_DATE 関数に NLS_LANGUAGE または NLS_TERRITORY を指定する場合は、関数の第 2 パラメータとして書式マスクも指定する必要があります。たとえば、次の指定は有効です。

```
TO_CHAR (hire_date, 'DD/MON/YYYY', 'nls_date_language = French')
```

次の指定は、書式マスクがないため無効です。

```
TO_CHAR (hire_date, 'nls_date_language = French')
```

次の指定は、書式マスクが関数の第 2 パラメータとして指定されていないため無効です。

```
TO_CHAR (hire_date, 'nls_date_language = French', 'DD/MON/YY')
```

ロケール依存のその他の SQL 関数

この項の内容は、次のとおりです。

CONVERT 関数

CONVERT 関数は、キャラクタ・セット間で文字データを変換します。

CONVERT 関数は、あるキャラクタ・セットの文字列のバイナリ表現を別のキャラクタ・セットのバイナリ表現に変換します。この関数は、データベースとクライアントの間のキャラクタ・セット変換と同じ方法を使用します。したがって、この関数では置換文字を使用するため、同じ制限があります。

> **関連項目：** 2-15 ページ「クライアントとサーバーの間のキャラクタ・セット変換」

CONVERT の構文は、次のとおりです。

```
CONVERT(char, dest_char_set[, source_char_set])
```

source_char_set は変換元キャラクタ・セットで、dest_char_set は変換先キャラクタ・セットです。source_char_set パラメータが指定されていない場合は、デフォルト値としてデータベース・キャラクタ・セットが使用されます。

異なるキャラクタ・セットを使用するクライアント / サーバー環境で変換を実行するには、CONVERT のかわりに TRANSLATE ...USING 関数を使用します。クライアントまたはサーバーに NCHAR または NVARCHAR2 データがある場合は、TRANSLATE...USING 関数を使用する必要があります。

> **関連項目：**
>
> - CONVERT 関数と TRANSLATE...USING 関数の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。
> - CONVERT 関数にのみ使用されるキャラクタ・セット・エンコーディングについては、A-20 ページの「キャラクタ・セット変換のサポート」を参照してください。

様々な長さセマンティクスに使用する SQL 関数

Oracle9*i* には、様々な長さセマンティクスに従って動作する SQL 関数が用意されています。この種の SQL 関数は、SUBSTR、LENGTH および INSTR という 3 つのグループに分かれています。グループ内の各関数は異なる種類の長さセマンティクスに基づいており、関数名に追加される文字または数字で区別されます。たとえば、SUBSTRB はバイト・セマンティクスに基づいています。

SUBSTR 関数は、サブストリングの要求された部分を戻します。LENGTH 関数は、文字列の長さを戻します。INSTR 関数は、文字列内のサブストリングを検索します。

SUBSTR 関数は、文字列の長さを様々な方法で計算します。表 7-1 に、計算方法を示します。

表 7-2 SUBSTR 関数による文字列の長さの計算方法

関数	計算方法
SUBSTR	文字列の長さを、データ型のキャラクタ・セットに関連付けられた長さセマンティクスに基づいて、文字数単位で計算します。たとえば、AL32UTF8 の文字は、UCS-4 コードの数で計算されます。UTF8 および AL16UTF16 の文字は、UCS-2 コードの数で計算されます。1 つの補助文字は、AL32UTF8 では 1 文字、UTF8 および AL16UTF16 では 2 文字としてカウントされます。VARCHAR および NVARCHAR では異なるキャラクタ・セットが使用されている場合があるため、2 つの文字列が同一の場合にも、データ型が異なると SUBSTR での結果が異なる可能性があります。アプリケーションで一貫性が必要な場合は、SUBSTR2 または SUBSTR4 を使用して、すべてのセマンティクスの計算を強制的にそれぞれ UCS-2 または UCS-4 にしてください。
SUBSTRB	文字列の長さをバイト数単位で計算します。
SUBSTR2	文字列の長さを、Java 文字列と Windows クライアント環境に準拠する UCS-2 コードの数で計算します。各文字は、UCS-2 または 16 ビット Unicode 値で表されます。補助文字は 2 コード単位としてカウントされます。
SUBSTR4	文字列の長さが UCS-4 コードの数で計算されます。各文字は、UCS-4 または 32 ビット Unicode 値で表されます。補助文字は 1 コード単位としてカウントされます。
SUBSTRC	文字列の長さを完全な Unicode 文字数で計算します。補助文字と複合文字は、1 文字としてカウントされます。

LENGTH および INSTR 関数は、関数名に追加された文字または数字に従って、文字列の長さを同様の方法で計算します。

次に、データベース・キャラクタ・セットが AL32UTF8 のデータベースで、SUBSTR と SUBSTRB を実行した場合の相違がわかる例を示します。

Fußball 文字列の場合、次の文では 2 文字目からの長さ 4 文字のサブストリングが戻されます。

```
SELECT SUBSTR ('Fußball', 2 , 4) SUBSTR FROM dual;

SUBS
----
ußba
```

Fußball 文字列の場合、次の文では 2 バイト目からの長さ 4 バイトのサブストリングが戻されます。

```
SELECT SUBSTRB ('Fußball', 2 , 4) SUBSTRB FROM dual;

SUB
---
ußb
```

関連項目： SUBSTR、LENGTH および INSTR 関数の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。

様々な長さセマンティクスに使用する LIKE 条件

LIKE 条件によって、パターン一致を使用するテストを指定します。等価演算子（=）は、ある文字の値と別の文字の値を完全に照合しますが、LIKE 条件は、1 番目の値の中に 2 番目で指定されたパターンがあるかどうかを検索し、ある文字の値と別の文字の値の一部を照合します。

LIKE では、入力のキャラクタ・セットに関連付けられた長さセマンティクスを使用して、文字列の長さが文字数単位で計算されます。表 7-3 に、LIKE2、LIKE4 および LIKEC の各条件を示します。

表 7-3 LIKE 条件

関数	説明
LIKE2	文字が UCS-2 のセマンティクスで表される場合に使用します。補助文字は 2 コード単位とみなされます。
LIKE4	文字が UCS-4 のセマンティクスで表される場合に使用します。補助文字は 1 コード単位とみなされます。
LIKEC	文字が完全な Unicode キャラクタ・セマンティクスで表される場合に使用します。合成文字は 1 コード単位として処理されます。

LIKEB 条件はありません。

キャラクタ・セットの SQL 関数

2 つの SQL 関数 NLS_CHARSET_NAME および NLS_CHARSET_ID で、キャラクタ・セット ID 番号とキャラクタ・セット名の間の変換ができます。これらの関数は、OCI を介して変数をバインドするために、キャラクタ・セット ID 番号の判断が必要なプログラムで使用されます。

もう 1 つの SQL 関数 NLS_CHARSET_DECL_LEN は、NCHAR 列の長さを戻します。

この項の内容は、次のとおりです。

関連項目：『Oracle9*i* SQL リファレンス』

キャラクタ・セット番号からキャラクタ・セット名への変換

NLS_CHARSET_NAME(*n*) 関数は、ID 番号 *n* に対応するキャラクタ・セットの名前を戻します。*n* がキャラクタ・セット ID として認識されない値である場合、関数は NULL を戻します。

キャラクタ・セット名からキャラクタ・セット番号への変換

NLS_CHARSET_ID(*text*) は、*text* によって指定された名前に対応するキャラクタ・セット ID を戻します。*text* には、ランタイム VARCHAR2 の値としてキャラクタ・セット名が定義されます。text の値は、データベース・キャラクタ・セットや各国語キャラクタ・セット以外のキャラクタ・セットに変換される NLSRTL 名とすることができます。

値 CHAR_CS を *text* に入力すると、この関数はサーバーのデータベース・キャラクタ・セット ID を戻します。値 NCHAR_CS を *text* に入力すると、この関数はサーバーの各国語キャラクタ・セット ID を戻します。*text* が名前として認識されない場合、この関数は NULL を戻します。

注意： *text* の値は、大文字で入力してください。

NCHAR 列の長さを戻す

NLS_CHARSET_DECL_LEN(*BYTECNT*、*CSID*) は、NCHAR 列の宣言の長さを文字数で戻します。*BYTECNT* は列のバイト単位の長さです。*CSID* は列のキャラクタ・セット ID です。

NLSSORT 関数

NLSSORT 関数では、ORDER BY 句に言語ソートを使用できます。文字列は言語ソート・メカニズムで使用される等価のソート文字列で置換されるため、置換文字列をソートして必要なソート基準を生成できます。バイナリ・ソートの場合、ソート文字列は入力文字列と同じです。

ORDER BY 句で使用される言語ソートの種類は NLS_SORT セッション・パラメータによって決定されますが、NLSSORT 関数を明示的に使用することでオーバーライドできます。

例 7-1 では、NLS_SORT セッション・パラメータを使用してドイツ語ソートを指定しています。

例 7-1 NLS_SORT セッション・パラメータを使用したドイツ語ソートの指定

```
ALTER SESSION SET NLS_SORT = GERMAN;
SELECT * FROM table1
ORDER BY column1;
```

例 7-2 NLSSORT 関数を使用したフランス語ソートの指定

この例では、最初に NLS_SORT セッション・パラメータを GERMAN に設定しますが、NLSSORT 関数でフランス語ソートを指定することでオーバーライドします。

```
ALTER SESSION SET NLS_SORT = GERMAN;
SELECT * FROM table1
ORDER BY NLSSORT(column1, 'NLS_SORT=FRENCH');
```

WHERE 句では、デフォルトで言語上の比較ではなくバイナリ比較が使用されますが、これは WHERE 句に NLSSORT 関数を使用してオーバーライドできます。

例 7-3 WHERE 句を使用した言語上の比較

```
ALTER SESSION SET NLS_COMP = ANSI;
SELECT * FROM table1
WHERE NLSSORT(column1, 'NLS_SORT=FRENCH')>
    NLSSORT(column2, 'NLS_SORT=FRENCH');
```

NLS_COMP セッション・パラメータを ANSI に設定すると、WHERE 句には NLS_SORT 値が使用されます。

これ以降の内容は、次のとおりです。

- NLSSORT の構文
- WHERE 句の文字列の比較

NLSSORT の構文

NLSSORT は、次の 4 通りの方法で使用できます。

- NLSSORT() (NLS_SORT パラメータに従います。)
- NLSSORT(column1, 'NLS_SORT=*xxxx*')
- NLSSORT(column1, 'NLS_LANG=*xxxx*')
- NLSSORT(column1, 'NLS_LANGUAGE=*xxxx*')

NLSSORT 関数の NLS_LANG パラメータは、クライアント環境設定の NLS_LANG とは異なります。NLSSORT 関数では、NLS_LANG によって言語の略称（米語は US、ポーランド語は PL など）が指定されます。次に例を示します。

```
SELECT * FROM table1
ORDER BY NLSSORT(column1, 'NLS_LANG=PL');
```

WHERE 句の文字列の比較

NLSSORT によって、アプリケーションは、アルファベットの規則に従った文字列の一致を実行できます。通常、WHERE 句の文字列は、文字のバイナリ値を使用して比較されます。データベース・キャラクタ・セットでのバイナリ値が大きい場合、その文字は他の文字よりも大きいとみなされます。文字のバイナリ値に基づいた文字順が言語のアルファベット順と一致しない場合があるため、このような比較では、アルファベットの規則に従わないことがあります。たとえば、列 (column1) に ISO 8859-1 の 8 ビット・キャラクタ・セットの値 ABC、ABZ、BCD および ÄBC が含まれている場合、Ä の数値は B より大きいため、次の問合せでは BCD と ÄBC の両方が戻されます。

```
SELECT column1 FROM table1 WHERE column1 > 'B';
```

Ä は、ドイツ語ではアルファベット順で B の前にソートされますが、スウェーデン語では Z の後にソートされます。ドイツ語では WHERE 句で NLSSORT を使用すると、次のように言語上の比較を行うことができます。

```
WHERE NLSSORT(col) comparison_operator NLSSORT(comparison_string)
```

NLSSORT は、比較演算子の両側に指定する必要があります。次に例を示します。

```
SELECT column1 FROM table1 WHERE NLSSORT(column1) > NLSSORT('B');
```

ドイツ語の言語ソートを設定していると、この文では Ä で始まる文字列は戻されません。これは、ドイツ語では、アルファベット順で Ä は B より前になるためです。スウェーデン語の言語ソートを設定していると、Ä で始まる文字列が戻されます。これは、スウェーデン語では、アルファベット順で Ä は Z より後になるためです。

NLS_COMP パラメータを使用した WHERE 句での比較の簡素化

デフォルトでは、WHERE 句または PL/SQL ブロックでの比較はバイナリで行われます。言語上の比較に NLSSORT 関数を使用するのは、特に NLS_SORT セッション・パラメータにすでに言語ソートが指定されている場合は、面倒な場合があります。NLS_COMP パラメータを使用すると、WHERE 句または PL/SQL ブロックでの比較に、NLS_SORT セッション・パラメータに従った言語を使用するように指定できます。

注意： NLS_COMP パラメータは、パーティション表に対する比較動作には影響しません。VALUES LESS THAN パーティションに基づく文字列の比較は、常にバイナリで行われます。

関連項目： 3-40 ページ「NLS_COMP」

ORDER BY 句の制御

言語ソートを使用中の場合、ORDER BY 句では文字データに対して暗黙的な NLSSORT が使用されます。ORDER BY 句のソート方法（言語またはバイナリ）は、アプリケーション側で意識することはありません。ただし、NLSSORT 関数が ORDER BY 句に明示的に指定されている場合、暗黙的な NLSSORT は実行されません。

NLS_SORT セッション・パラメータで言語ソートが定義されている場合、アプリケーションでは ORDER BY 句に暗黙的な NLSSORT 関数が使用されます。明示的な NLSSORT 関数を指定すると、暗黙的な NLSSORT 関数がオーバーライドされます。

ソート方法が言語として定義されている場合、通常、ORDER BY 句に NLSSORT 関数は不要です。

ソート方法がデフォルトであるか、バイナリとして定義されている場合、次のような問合せではバイナリ・ソートが使用されます。

```
SELECT last_name FROM employees
ORDER BY last_name;
```

次の問合せによって、ドイツ語の言語ソートを取得できます。

```
SELECT last_name FROM employees
ORDER BY NLSSORT(last_name, 'NLS_SORT = GERMAN');
```

関連項目： 4-15 ページ「関数索引を使用した大 / 小文字区別なしの検索パフォーマンスの改善」

グローバル環境での SQL と PL/SQL のプログラミングに関するその他のトピック

この項の内容は、次のとおりです。

> **関連項目：** 書式マスクの詳細は、『Oracle9*i* SQL リファレンス』を参照してください。

SQL の日付書式マスク

TO_CHAR、TO_DATE および TO_NUMBER 関数とともに複数の書式マスクが用意されています。

RM（Roman Month）書式要素は、ローマ数字で月を戻します。RM または rm を使用して、大文字または小文字を指定できます。たとえば、1998 年 9 月 7 日の場合、DD-rm-YYYY では 07-ix-1998 が戻り、DD-RM-YYYY では 07-IX-1998 が戻ります。

書式マスク MON および DY は、長さ 3 文字以内の月および曜日の略称を明示的にサポートします。たとえば、フランス語の Lundi と Mardi に対して、それぞれの略称 Lu と Ma を指定できます。

週番号の計算

WW 書式マスクで戻される週番号は、アルゴリズム int(dayOfYear+6)/7 に従って算出されます。この週番号のアルゴリズムは、ISO 規格（2015、1992-06-15）に従っていません。

ISO 規格をサポートするために、IW 書式要素が用意されています。この書式要素は ISO 週番号を戻します。また、Y、YY、YYY および YYYY の各書式要素の動作と同等の書式要素 I、IY、IYY および IYYY は、ISO 週番号に関連する年を戻します。

ISO 規格では、ISO 週番号に関連する年が、暦年と異なることがあります。たとえば、1988 年 1 月 1 日は、1987 年の ISO 週番号 53 になります。週は、常に月曜日に始まって日曜日に終わります。週番号は、次の規則に従って決定されます。

- 1 月 1 日が金曜日、土曜日または日曜日の場合、1 月 1 日を含む週は、その週の大半の日が前年に属するため、前年の最後の週になります。
- 1 月 1 日が月曜日、火曜日、水曜日または木曜日の場合は、1 月 1 日を含む週は、その週の大半の日が新しい年に属するため、新しい年の最初の週になります。

たとえば、1991 年 1 月 1 日は火曜日であるため、1990 年 12 月 31 日の月曜日から 1991 年 1 月 6 日の日曜日までが第 1 週になります。したがって、1990 年 12 月 31 日の ISO 週番号および年は、1 および 1991 となります。ISO 週番号を取得するには、週番号に IW 書式マスクを使用し、年に IY 書式のいずれかを使用します。

SQL の数値書式マスク

数値を書式設定するための書式要素がいくつか用意されています。

- D（Decimal）は、小数点文字を戻します。
- G（Group）は、グループ・セパレータを戻します。
- L（Local Currency）は、各国通貨記号を戻します。
- C（International Currency）は、ISO 通貨記号を戻します。
- RN（Roman Numeral）は、数値を同等のローマ数字として戻します。

ローマ数字の場合は、RN または rn を使用して、大文字または小文字のいずれかを指定できます。変換される数値は、1 ～ 3999 の範囲の整数であることが必要です。

連結演算子

データベース・キャラクタ・セットが垂直バー | を各国語キャラクタに置換すると、連結演算子（ASCII 124 としてエンコード）を使用する SQL 文はすべて失敗します。たとえば、プロシージャの作成は、連結を使用する再帰的 SQL 文を生成するため失敗します。データベース・キャラクタ・セットに D7DEC、F7DEC、SF7ASCII などの 7 ビットの置換キャラクタ・セットを使用すると、垂直バーは連結演算子として解釈されるため、垂直バーを置換する各国語キャラクタはオブジェクト名に使用できません。

ユーザーは、データベース・キャラクタ・セットが同じか、互換性がある場合（つまり、両方のキャラクタ・セットが垂直バーを同じ各国語キャラクタに置換する場合）は、7 ビットの置換キャラクタ・セットを使用できます。

LOB への外部 BFILE データのロード

DBMS_LOB PL/SQL パッケージを使用すると、外部 BFILE データを LOB にロードできます。以前のリリースの Oracle では、バイナリ・データを CLOB または NCLOB にロードする前のキャラクタ・セット変換は実行されませんでした。そのため、正常に動作させるには、BFILE データをデータベース・キャラクタ・セットまたは各国語キャラクタ・セットと同じキャラクタ・セットにする必要がありました。Oracle9*i* リリース 2（9.2）で導入された API では、新規パラメータを使用して BFILE データのキャラクタ・セット ID を指定できます。API では、指定した BFILE のキャラクタ・セットから、CLOB の場合はデータベース・キャラクタ・セット、NCLOB の場合は各国語キャラクタ・セットにデータが変換されます。クライアント上では BFILE データはサポートされないため、ロードはサーバー上で発生します。

- BLOB にロードするには、DBMS_LOB.LOADBLOBFROMFILE を使用します。
- CLOB および NCLOB にロードするには、DBMS_LOB.LOADCLOBFROMFILE を使用します。

関連項目：

- 『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』
- 『Oracle9*i* アプリケーション開発者ガイド - ラージ・オブジェクト』

8

グローバル環境での OCI プログラミング

この章では、OCI プログラミングに役立つ情報について説明します。この章の内容は、次のとおりです。

- OCI NLS 関数の使用
- OCI でのキャラクタ・セットの指定
- OCI でのロケール情報の取得
- Oracle と他の規格とのロケール情報のマッピング
- OCI での文字列操作
- OCI での文字の分類
- OCI でのキャラクタ・セットの変換
- OCI メッセージ関数

OCI NLS 関数の使用

多くの OCI NLS 関数では、環境ハンドルまたはユーザー・セッション・ハンドルのいずれかを受け入れます。OCI 環境ハンドルは、クライアント NLS 環境に関連付けられており、クライアント NLS 環境変数によって初期化されます。サーバーに対して ALTER SESSION 文が使用されても、この環境は変わりません。環境ハンドルに関連付けられているキャラクタ・セットは、クライアント・キャラクタ・セットです。OCISessionBegin によって戻される OCI セッション・ハンドルは、サーバー・セッション環境に関連付けられています。ALTER SESSION 文よってセッション環境が変更されると、この NLS 設定も変更されます。セッション・ハンドルに関連付けられているキャラクタ・セットは、データベース・キャラクタ・セットです。

OCI セッション・ハンドルは、そのセッションで最初のトランザクションが始まるまで、どの NLS 設定とも関連付けられていません。SELECT 文では、トランザクションは始まりません。

OCI でのキャラクタ・セットの指定

OCI 環境の作成時に OCIEnvNlsCreate 関数を使用して、クライアント側データベースと各国語キャラクタ・セットを指定します。この関数を使用すると、NLS_LANG および NLS_CHAR 初期化パラメータの設定に関係なく、アプリケーションでキャラクタ・セット情報を動的に設定できます。また、1 つのアプリケーションで、同じサーバー環境内で異なるクライアント環境に使用する複数の環境ハンドルを初期化できます。

AL16UTF16 を除く Oracle キャラクタ・セット ID を指定するには、OCIEnvNlsCreate 関数を通じてメタデータ、SQL CHAR データおよび SQL NCHAR データのエンコーディングを指定します。UTF-16 データを指定するには、Oracle 9*i* リリース 2（9.2）で導入された OCIEnvNlsCreate 関数で OCI_UTF16ID を使用します。Oracle9*i* リリース 1（9.0.1）で導入された OCIEnvCreate 関数の OCI_UTF16 パラメータは、Unicode モードと呼ばれていましたが、使用されなくなったことに注意してください。

> **関連項目：** OCIEnvNlsCreate 関数および OCIEnvCreate 関数の詳細は、『Oracle Call Interface プログラマーズ・ガイド』を参照してください。

OCIEnvNlsCreate()

構文

```
sword OCIEnvNlsCreate   ( OCIEnv        **envhpp,
                          ub4           mode,
                          dvoid         *ctxp,
                          dvoid         *(*malocfp)
                                        (dvoid *ctxp,
                                         size_t size),
                          dvoid         *(*ralocfp)
                                        (dvoid *ctxp,
                                         dvoid *memptr,
                                         size_t newsize),
                          void          (*mfreefp)
                                        (dvoid *ctxp,
                                         dvoid *memptr))
                          size_t        xtramemsz,
                          dvoid         **usrmempp
                          ub2           charset,
                          ub2           ncharset );
```

用途

OCI 関数の動作に使用される環境ハンドルを作成し、初期化します。この関数は、OCIEnvCreate() 関数の拡張バージョンです。

パラメータ

envhpp（OUT）

エンコーディング設定が *mode* で指定されている環境ハンドルへのポインタ。この設定は、*envhpp* から導出された文ハンドルによって継承されます。

mode（IN）

モードの初期化を指定します。有効なモードは、次のとおりです。

- OCI_DEFAULT: デフォルト値である非 UTF-16 エンコーディングです。
- OCI_THREADED: スレッド化環境を使用します。ユーザーに公開されない内部データ構造は、複数のスレッドによって同時アクセスから保護されます。
- OCI_OBJECT: オブジェクト機能を使用します。
- OCI_UTF16: 環境ハンドルとそこから継承されるハンドルでは、UTF-16 エンコーディングが想定されます。この設定は使用されなくなりました。かわりに、*charset* と *ncharset* の両方に OCI_UTF16ID を指定してください。
- OCI_SHARED: 共有データ構造を使用します。

- OCI_EVENTS: パブリッシュ・サブスクライブ通知を使用します。
- OCI_NO_UCB: *OCIEnvCallback* 動的コールバック・ルーチンのコールを抑制します。デフォルト動作では、環境の作成時に *OCIEnvCallback* をコールできます。
- OCI_ENV_NO_MUTEX: このモードでは mutex 化は行われません。環境ハンドルまたは環境ハンドルから導出されるハンドルでの OCI コールは、すべてシリアル化する必要があります。

ctxp（IN）
メモリー・コールバック・ルーチンのユーザー定義コンテキストを指定します。

malocfp（IN）
ユーザー定義のメモリー割当て関数を指定します。OCI_THREADED モードの場合、このメモリー割当てルーチンはスレッド・セーフである必要があります。

ctxp（IN）
ユーザー定義のメモリー割当て関数のコンテキスト・ポインタを指定します。

size（IN）
ユーザー定義のメモリー割当て関数で割り当てるメモリーのサイズを指定します。

ralocfp（IN）
ユーザー定義のメモリー再割当て関数を指定します。OCI_THREADED モードの場合、このメモリー割当てルーチンはスレッド・セーフである必要があります。

ctxp（IN）
ユーザー定義のメモリー再割当て関数のコンテキスト・ポインタを指定します。

memp（IN）
メモリー・ブロックへのポインタです。

newsize（IN）
割り当てるメモリーの新規サイズを指定します。

mfreefp（IN）
ユーザー定義のメモリー解放関数を指定します。OCI_THREADED モードの場合、このメモリー解放ルーチンはスレッド・セーフである必要があります。

ctxp（IN）
ユーザー定義のメモリー解放関数のコンテキスト・ポインタを指定します。

memptr（IN）
解放するメモリーへのポインタです。

xtramemsz（IN）
環境の継続時間に対して割り当てるユーザー・メモリーの量を指定します。

usrmempp（OUT）
このコールでユーザーに割り当てるサイズ *xtramemsz* のユーザー・メモリーへのポインタを戻します。

charset（IN）
現行の環境ハンドル用のクライアント側キャラクタ・セットです。このパラメータが 0 の場合は、NLS_LANG 設定が使用されます。OCI_UTF16ID は有効な設定です。この設定は、メタデータと CHAR データに影響します。

ncharset（IN）
現行の環境ハンドル用のクライアント側各国語キャラクタ・セットです。このパラメータが 0 の場合は、NLS_NCHAR 設定が使用されます。OCI_UTF16ID は有効な設定です。この設定は NCHAR データに影響します。

戻り値

OCI_SUCCESS: 環境ハンドルは正常に作成されています。

OCI_ERROR: エラーが発生しました。

コメント

> **注意：** このコールは、他の OCI コールの前に起動する必要があります。また、このコールを OCIInitialize() および OCIEnvInit() コールのかわりに使用してください。OCIInitialize() および OCIEnvInit() コールは、下位互換性を保つためにサポートされています。

この関数は、NLS_LANG および NLS_NCHAR で指定されたキャラクタ・セットを置き換えて、0（ゼロ）以外の *charset* および *ncharset* をクライアント側のデータベース・キャラクタ・セットおよび各国語キャラクタ・セットとして設定します。*charset* および *ncharset* が 0 の場合の動作は、OCIEnvCreate() と同じです。特に、*charset* ではメタデータと暗黙的なフォーム属性を持つデータのエンコーディングを制御し、*ncharset* では SQLCS_NCHAR フォーム属性を持つデータのエンコーディングを制御します。

OCI_UTF16ID は OCIEnvNlsCreate() で設定できますが、NLS_LANG と NLS_NCHAR の場合は UTF-16 を設定できません。

NLS_LANG および NLS_NCHAR 内のキャラクタ・セット ID は、OCINlsEnvironmentVariableGet() を使用して取り出すことができます。

このコールで戻される環境ハンドルが、他の OCI 関数で使用されます。OCI に複数の環境が存在し、それぞれが独自の環境モードを使用する場合があります。また、この関数は、いずれかのモードで要求される場合は、プロセス・レベルの初期化を実行します。たとえば、環境を OCI_THREADED として初期化する必要がある場合は、OCI で使用されるすべてのライブラリもスレッド化モードで初期化されます。

OCI ライブラリを使用して DLL または共有ライブラリを作成する場合は、`OCIInitialize()` および `OCIEnvInit()` コールのかわりにこのコールを使用する必要があります。

関連項目： 8-13 ページ「OCINlsEnvironmentVariableGet()」

OCI でのロケール情報の取得

Oracle のロケールは、言語、地域およびキャラクタ・セットの定義で構成されています。ロケールによって、日付、時刻、数字および通貨の書式と、曜日名、月名などの規則が定義されます。国際化されたアプリケーションは、ユーザーのロケール設定と文化的な慣習に従って動作します。たとえば、ロケールがドイツ語に設定されている場合、曜日名と月名はドイツ語で表示されます。

`OCINlsGetInfo()` 関数を使用すると、次の情報を取得できます。

曜日（変換後）
曜日の略称（変換後）
月名（変換後）
月の略称（変換後）
Yes/No（変換後）
AM/PM（変換後）
AD/BC（変換後）
数値書式
借方 / 貸方
日付書式
通貨書式
デフォルトの言語
デフォルトの地域
デフォルトのキャラクタ・セット
デフォルトの言語ソート
デフォルトのカレンダ

この項の内容は、次のとおりです。

OCINlsGetInfo()

構文

```
sword OCINlsGetInfo(dvoid *hndl, OCIError *errhp, OraText *buf, size_t buflen, ub2
item)
```

用途

この関数は、item で指定されたロケール情報を OCI 環境ハンドルまたはユーザー・セッション・ハンドル（hndl）から取得し、buflen で指定されたサイズ制限の範囲内で buf で示された配列に格納します。

戻り値

OCI_SUCCESS、OCI_INVALID_HANDLE または OCI_ERROR

パラメータ

hndl（IN/OUT）
オブジェクト・モードで初期化された OCI 環境ハンドルまたはユーザー・セッション・ハンドル。

errhp（IN/OUT）
OCI エラー・ハンドル。エラーがある場合は、errhp に記録され、NULL ポインタが戻されます。診断情報は OCIErrorGet() をコールすると取得できます。

buf（OUT）
宛先バッファへのポインタ。戻される文字列は NULL 文字で終了します。

buflen（IN）
宛先バッファのサイズ。それぞれの情報の最大長は、OCI_NLS_MAXBUFSZ バイトです。

item（IN）
OCI 環境ハンドルが戻す項目を指定します。次のいずれかの値を指定できます。

OCI_NLS_DAYNAME1: 月曜日のネイティブの名前
OCI_NLS_DAYNAME2: 火曜日のネイティブの名前
OCI_NLS_DAYNAME3: 水曜日のネイティブの名前
OCI_NLS_DAYNAME4: 木曜日のネイティブの名前
OCI_NLS_DAYNAME5: 金曜日のネイティブの名前
OCI_NLS_DAYNAME6: 土曜日のネイティブの名前
OCI_NLS_DAYNAME7: 日曜日のネイティブの名前
OCI_NLS_ABDAYNAME1: 月曜日のネイティブの略称
OCI_NLS_ABDAYNAME2: 火曜日のネイティブの略称
OCI_NLS_ABDAYNAME3: 水曜日のネイティブの略称
OCI_NLS_ABDAYNAME4: 木曜日のネイティブの略称
OCI_NLS_ABDAYNAME5: 金曜日のネイティブの略称
OCI_NLS_ABDAYNAME6: 土曜日のネイティブの略称
OCI_NLS_ABDAYNAME7: 日曜日のネイティブの略称
OCI_NLS_MONTHNAME1: 1 月のネイティブの名前
OCI_NLS_MONTHNAME2: 2 月のネイティブの名前
OCI_NLS_MONTHNAME3: 3 月のネイティブの名前
OCI_NLS_MONTHNAME4: 4 月のネイティブの名前
OCI_NLS_MONTHNAME5: 5 月のネイティブの名前
OCI_NLS_MONTHNAME6: 6 月のネイティブの名前
OCI_NLS_MONTHNAME7: 7 月のネイティブの名前
OCI_NLS_MONTHNAME8: 8 月のネイティブの名前
OCI_NLS_MONTHNAME9: 9 月のネイティブの名前
OCI_NLS_MONTHNAME10: 10 月のネイティブの名前
OCI_NLS_MONTHNAME11: 11 月のネイティブの名前
OCI_NLS_MONTHNAME12: 12 月のネイティブの名前
OCI_NLS_ABMONTHNAME1: 1 月のネイティブの略称
OCI_NLS_ABMONTHNAME2: 2 月のネイティブの略称
OCI_NLS_ABMONTHNAME3: 3 月のネイティブの略称
OCI_NLS_ABMONTHNAME4: 4 月のネイティブの略称
OCI_NLS_ABMONTHNAME5: 5 月のネイティブの略称
OCI_NLS_ABMONTHNAME6: 6 月のネイティブの略称
OCI_NLS_ABMONTHNAME7: 7 月のネイティブの略称
OCI_NLS_ABMONTHNAME8: 8 月のネイティブの略称
OCI_NLS_ABMONTHNAME9: 9 月のネイティブの略称
OCI_NLS_ABMONTHNAME10: 10 月のネイティブの略称
OCI_NLS_ABMONTHNAME11: 11 月のネイティブの略称
OCI_NLS_ABMONTHNAME12: 12 月のネイティブの略称
OCI_NLS_YES: ネイティブの肯定的な応答の文字列
OCI_NLS_NO: ネイティブの否定的な応答
OCI_NLS_AM: AM に相当するネイティブの文字列
OCI_NLS_PM: PM に相当するネイティブの文字列

OCI_NLS_AD: AD に相当するネイティブの文字列
OCI_NLS_BC: BC に相当するネイティブの文字列
OCI_NLS_DECIMAL: 小数点文字
OCI_NLS_GROUP: グループ・セパレータ
OCI_NLS_DEBIT: ネイティブの借方記号
OCI_NLS_CREDIT: ネイティブの貸方記号
OCI_NLS_DATEFORMAT: Oracle の日付書式
OCI_NLS_INT_CURRENCY: 国際通貨記号
OCI_NLS_DUAL_CURRENCY: 第 2 通貨記号
OCI_NLS_LOC_CURRENCY: 各国通貨記号
OCI_NLS_LANGUAGE: 言語名
OCI_NLS_ABLANGUAGE: 言語名の略称
OCI_NLS_TERRITORY: 地域名
OCI_NLS_CHARACTER_SET: キャラクタ・セット名
OCI_NLS_LINGUISTIC_NAME: 言語ソート名
OCI_NLS_CALENDAR: カレンダ名
OCI_NLS_WRITING_DIR: 言語の書込み方向
OCI_NLS_ABTERRITORY: 地域の略称
OCI_NLS_DDATEFORMAT: Oracle のデフォルト日付書式
OCI_NLS_DTIMEFORMAT: Oracle のデフォルト時刻書式
OCI_NLS_SFDATEFORMAT: ローカルの日付書式
OCI_NLS_SFTIMEFORMAT: ローカルの時刻書式
OCI_NLS_NUMGROUPING: 数値のグループ・フィールド
OCI_NLS_LISTSEP: リスト・セパレータ
OCI_NLS_MONDECIMAL: 通貨の小数点文字
OCI_NLS_MONGROUP: 通貨のグループ・セパレータ
OCI_NLS_MONGROUPING: 通貨のグループ・フィールド
OCI_NLS_INT_CURRENCYSEP: 国際通貨セパレータ

OCI_NLS_MAXBUFSZ

OCINlsGetInfo() をコールする場合は、戻された情報を格納するバッファを割り当てる必要があります。バッファ・サイズは、どの項目に対して問合せを行うか、どのエンコーディングを使用して情報を格納するかによって異なります。JA16SJIS エンコーディングを使用して January を日本語で格納するために必要なバイト数を、開発者が意識せずに済むことが必要です。OCI_NLS_MAXBUFSZ 属性によって、バッファは OCINlsGetInfo() が戻す最大の項目を保持するために十分な大きさであることが保証されます。

例 : OCI でのロケール情報の取得

次のサンプル・コードは、情報を取得してエラーの有無をチェックします。

```
sword MyPrintLinguisticName(envhp, errhp)
OCIEnv   *envhp;
OCIError *errhp;
{
  OraText  infoBuf[OCI_NLS_MAXBUFSZ];
  sword ret;

  ret = OCINlsGetInfo(envhp,                          /* environment handle */
                      errhp,                                /* error handle */
                      infoBuf,                        /* destination buffer */
                      (size_t) OCI_NLS_MAXBUFSZ,             /* buffer size */
                      (ub2) OCI_NLS_LINGUISTIC_NAME);               /* item */

  if (ret != OCI_SUCCESS)
  {
    checkerr(errhp, ret, OCI_HTYPE_ERROR);
    ret = OCI_ERROR;
  }
  else
  {
    printf("NLS linguistic: %s\n", infoBuf);
   }
  return(ret);
}
```

OCINlsCharSetNameToId()

構文

```
ub2 OCINlsCharSetNameToId(dvoid *hndl, const oratext *name)
```

用途

この関数は、指定された Oracle キャラクタ・セット名の Oracle キャラクタ・セット ID を戻します。

戻り値

指定されたキャラクタ・セット名と OCI ハンドルが有効な場合は、キャラクタ・セット ID。それ以外の場合は 0 を戻します。

パラメータ

hndl（IN/OUT）
OCI 環境ハンドルまたはセッション・ハンドル。ハンドルが無効な場合は、0（ゼロ）を戻します。

name（IN）
ヌル文字で終了する Oracle キャラクタ・セット名へのポインタ。キャラクタ・セット名が無効な場合は、0（ゼロ）を戻します。

OCINlsCharSetIdToName()

構文

```
sword OCINlsCharSetIdToName( dvoid *hndl, oratext *buf, size_t buflen, ub2 id)
```

用途

この関数は、指定されたキャラクタ・セット ID の Oracle キャラクタ・セット名を戻します。

戻り値

OCI_SUCCESS、OCI_INVALID_HANDLE または OCI_ERROR

パラメータ

hndl（IN/OUT）
OCI 環境ハンドルまたはセッション・ハンドル。ハンドルが無効な場合は、OCI_INVALID_HANDLE を戻します。

buf（OUT）
宛先バッファへのポインタ。この関数が OCI_SUCCESS を戻す場合、このパラメータにはキャラクタ・セット名を表すヌル文字で終了する文字列が含まれます。

buflen（IN）
宛先バッファのサイズ。Oracle キャラクタ・セット名の格納を保証するための推奨サイズは、OCI_NLS_MAXBUFSZ です。宛先バッファのサイズがキャラクタ・セット名の長さより小さいと、この関数は OCI_ERROR を戻します。

id（IN）
Oracle キャラクタ・セット ID。

OCINlsNumericInfoGet()

構文

```
sword OCINlsNumericInfoGet( dvoid *hndl, OCIError *errhp, sb4 *val, ub2 item)
```

用途

この関数は、item で指定された数値言語情報を、出力の数値変数への OCI 環境ハンドルから取得します。

戻り値

OCI_SUCCESS、OCI_INVALID_HANDLE または OCI_ERROR

パラメータ

hndl（IN/OUT）
OCI 環境ハンドルまたはセッション・ハンドル。ハンドルが無効な場合は、OCI_INVALID_HANDLE を戻します。

errhp（IN/OUT）
OCI エラー・ハンドル。エラーがある場合は、errhp に記録され、NULL ポインタが戻されます。診断情報は OCIErrorGet() をコールすると取得できます。

val（OUT）
出力の数値変数へのポインタ。この関数が OCI_SUCCESS を戻す場合、このパラメータには要求された NLS 数値情報が含まれます。

item（IN）
OCI 環境ハンドルから取得する項目を指定します。次のいずれかの値を指定できます。

- OCI_NLS_CHARSET_MAXBYTESZ: OCI 環境ハンドルまたはセッション・ハンドルのキャラクタ・セットの最大文字バイト・サイズ。
- OCI_NLS_CHARSET_FIXEDWIDTH: 固定幅キャラクタ・セットの場合は文字バイト・サイズ、可変幅キャラクタ・セットの場合は 0。

OCINlsEnvironmentVariableGet()

用途

NLS_LANG からのキャラクタ・セット ID、または NLS_NCHAR からの各国語キャラクタ・セット ID を戻します。

構文

```
sword OCINlsEnvironmentVariableGet ( dvoid     *val,
                                     size_t    size,
                                     ub2       item,
                                     ub2 charset,
                                     size_t    *rsize );
```

パラメータ

val（IN/OUT）
NLS_LANG のキャラクタ・セット ID または NLS_NCHAR のキャラクタ・セット ID など、NLS 環境変数の値を戻します。

size（IN）
特定の出力値のサイズを指定します。これは、文字列データにのみ適用されます。それぞれの情報の最大長は、OCI_NLS_MAXBUFSZ バイトです。数値データの場合、この引数は無視されます。

item（IN）
NLS 環境変数から取得する値として次のいずれかを指定します。

- OCI_NLS_CHARSET_ID: ub2 データ型の NLS_LANG のキャラクタ・セット ID。
- OCI_NLS_NCHARSET_ID: ub2 データ型の NLS_NCHAR のキャラクタ・セット ID。

charset（IN）
取得される文字列データのキャラクタ・セット ID を指定します。このパラメータが 0 の場合は、NLS_LANG の値が使用されます。OCI_UTF16ID は、この引数の有効値です。数値データの場合、この引数は無視されます。

rsize（OUT）
戻り値のバイト長。

戻り値

OCI_SUCCESS: この関数は正常終了しました。

OCI_ERROR: エラーが発生しました。

コメント

NLS の規則では、NLS_NCHAR が設定されていない場合、各国語キャラクタ・セット ID はキャラクタ・セット ID と同じです。NLS_LANG が設定されていない場合は、デフォルトのキャラクタ・セット ID が戻されます。

この関数の将来の拡張（環境変数からの他の値の取得）に対処できるように、出力 val のデータ型は dvoid へのポインタです。文字列データは、ヌル文字で終了しません。

この関数は環境ハンドルを取らないため、戻されるキャラクタ・セット ID と各国語キャラクタ・セット ID は、OCI 環境ハンドルに保存されている値ではなく、NLS_LANG および NLS_NCHAR で指定された値であることに注意してください。OCI 環境ハンドルで使用されるキャラクタ・セット ID を取得するには、OCI_ATTR_ENV_CHARSET および OCI_ATTR_ENV_NCHARSET について OCIAttrGet() をコールします。

Oracle と他の規格とのロケール情報のマッピング

OCINlsNameMap 関数は、Oracle キャラクタ・セット名、言語名および地域名と、Internet Assigned Numbers Authority （IANA）名および国際標準化機構（ISO）名の間のマッピングを行います。

OCINlsNameMap()

構文

```
sword OCINlsNameMap( dvoid *hndl, oratext *buf, size_t buflen, const oratext
*srcbuf, uword flag)
```

用途

この関数は、Oracle キャラクタ・セット名、言語名および地域名と、IANA 名および ISO 名の間のマッピングを行います。

戻り値

OCI_SUCCESS、OCI_INVALID_HANDLE または OCI_ERROR

パラメータ

hndl（IN/OUT）
OCI 環境ハンドルまたはセッション・ハンドル。ハンドルが無効な場合は、OCI_INVALID_HANDLE を戻します。

buf（OUT）
宛先バッファへのポインタ。この関数が OCI_SUCCESS を戻す場合、このパラメータには要求された名前を表すヌル文字で終了する文字列が含まれます。

buflen（IN）
宛先バッファのサイズ。NLS 名の格納を保証するための推奨サイズは、OCI_NLS_MAXBUFSZ です。宛先バッファのサイズが名前の長さより小さいと、この関数は OCI_ERROR を戻します。

srcbuf（IN）
ヌル文字で終了する NLS 名へのポインタ。有効な名前でない場合は、OCI_ERROR を戻します。

flag（IN）
名前マッピングの方向を指定します。次のいずれかの値を使用できます。

OCI_NLS_CS_IANA_TO_ORA: キャラクタ・セット名を IANA から Oracle にマップします。
OCI_NLS_CS_ORA_TO_IANA: キャラクタ・セット名を Oracle から IANA にマップします。
OCI_NLS_LANG_ISO_TO_ORA: 言語名を ISO から Oracle にマップします。
OCI_NLS_LANG_ORA_TO_ISO: 言語名を Oracle から ISO にマップします。
OCI_NLS_TERR_ISO_TO_ORA: 地域名を ISO から Oracle にマップします。
OCI_NLS_TERR_ORA_TO_ISO: 地域名を Oracle から ISO にマップします。
OCI_NLS_TERR_ISO3_TO_ORA: 地域名を 3 文字の ISO 略称から Oracle にマップします。
OCI_NLS_TERR_ORA_TO_ISO3: 地域名を Oracle から 3 文字の ISO 略称にマップします。

OCI での文字列操作

文字列操作では、次の 2 種類のデータ構造がサポートされています。

- マルチバイト文字列
- ワイド・キャラクタ文字列

マルチバイト文字列は、Oracle のネイティブ・キャラクタ・セットでエンコードされます。マルチバイト文字列を処理する関数は、その文字列の長さをバイト単位で計算して 1 単位として扱います。ワイド・キャラクタ（wchar）文字列関数は、文字列操作の柔軟性を高めます。この種の関数は、文字列の長さを文字数で計算し、文字ベースと文字列ベースの操作をサポートします。

ワイド・キャラクタのデータ型は Oracle 固有のデータ型です。ANSI/ISO の C 規格で定義されている wchar_t データ型と混同しないでください。Oracle のワイド・キャラクタのデータ型は、すべてのプラットフォームで常に 4 バイトですが、wchar_t のサイズは、実装とプラットフォームに依存します。Oracle のワイド・キャラクタのデータ型によって、マルチバイト・キャラクタは簡単に処理できる固定幅になるように正規化されます。これにより、Oracle のワイド・キャラクタ・セットとネイティブ・キャラクタ・セットの間でラウンドトリップ変換時にデータ消失が発生しないことが保証されます。

文字列操作は、次のように分類できます。

- マルチバイト・キャラクタとワイド・キャラクタ間の文字列の変換
- 文字の分類
- 大 / 小文字の変換
- 表示長の計算
- 比較、連結、検索などの一般的な文字列操作

表 8-1 に、OCI 文字列操作関数を示します。各関数の詳細は後述します。

表 8-1 OCI 文字列操作関数

関数	説明
OCIMultiByteToWideChar()	ヌル文字で終了する文字列全体を wchar 書式に変換します。
OCIMultiByteInSizeToWideChar()	文字列の一部を wchar 書式に変換します。
OCIWideCharToMultiByte()	ヌル文字で終了するワイド・キャラクタ文字列全体をマルチバイト文字列に変換します。
OCIWideCharInSizeToMultiByte()	ワイド・キャラクタ文字列の一部をマルチバイト書式に変換します。
OCIWideCharToLower()	wc で指定された wchar 文字を、指定したロケールに存在する場合は対応する小文字に変換します。対応する小文字が存在しない場合は、wc 自体を戻します。
OCIWideCharToUpper()	wc で指定された wchar 文字を、指定したロケールに存在する場合は対応する大文字に変換します。対応する大文字が存在しない場合は、wc 自体を戻します。
OCIWideCharStrcmp()	2 つのワイド・キャラクタ文字列を、バイナリ、言語または大 / 小文字を区別しない比較方法で比較します。
OCIWideCharStrncmp()	この関数は OCIWideCharStrcmp() に似ており、2 つのワイド・キャラクタ文字列を、バイナリ、言語または大 / 小文字を区別しない比較方法で比較します。str1 から最大 len1 バイト、str2 から最大 len2 バイトが比較されます。
OCIWideCharStrcat()	wsrcstr で示された文字列のコピーを追加します。その後、結果文字列の文字数を戻します。
OCIWideCharStrncat()	wsrcstr で示された文字列のコピーを追加します。その後、結果文字列の文字数を戻します。最大 *n* 文字が追加されます。
OCIWideCharStrchr()	wstr で示された文字列の中で、最初に出現する wc を検索します。検索が正常終了した場合は、wchar へのポインタを戻します。
OCIWideCharStrrchr()	wstr で示された文字列の中で、最後に出現する wc を検索します。

表 8-1 OCI 文字列操作関数（続き）

関数	説明
OCIWideCharStrcpy()	wsrcstr で示された wchar 文字列を wdststr で示された配列にコピーします。その後、コピーされた文字数を戻します。
OCIWideCharStrncpy()	wsrcstr で示された wchar 文字列を wdststr で示された配列にコピーします。その後、コピーされた文字数を戻します。最大 *n* 文字が配列からコピーされます。
OCIWideCharStrlen()	wstr で示された wchar 文字列にある文字数を計算し、その数を戻します。
OCIWideCharStrCaseConversion()	wsrcstr で示されたワイド・キャラクタ文字列をフラグの指定により大文字または小文字に変換し、その結果を wdststr で示された配列にコピーします。
OCIWideCharDisplayLength()	wc を表示するために必要な列位置の数を決定します。
OCIWideCharMultibyteLength()	wc をマルチバイト・エンコーディングで表す場合に必要なバイト数を決定します。
OCIMultiByteStrcmp()	2 つのマルチバイト・キャラクタ文字列を、バイナリ、言語または大 / 小文字を区別しない比較方法で比較します。
OCIMultiByteStrncmp()	2 つのマルチバイト・キャラクタ文字列を、バイナリ、言語または大 / 小文字を区別しない比較方法で比較します。str1 から最大 len1 バイト、str2 から最大 len2 バイトが比較されます。
OCIMultiByteStrcat()	srcstr で示されたマルチバイト文字列のコピーを追加します。
OCIMultiByteStrncat()	srcstr で示されたマルチバイト文字列のコピーを追加します。srcstr から最大 *n* バイトが dststr に追加されます。
OCIMultiByteStrcpy()	srcstr で示されたマルチバイト文字列を dststr で示された配列にコピーします。その後、コピーされたバイト数を戻します。
OCIMultiByteStrncpy()	srcstr で示されたマルチバイト文字列を dststr で示された配列にコピーします。その後、コピーされたバイト数を戻します。srcstr で示された配列から最大 *n* バイトが、dststr で示された配列にコピーされます。
OCIMultiByteStrlen()	str で示されたマルチバイト文字列のバイト数を戻します。
OCIMultiByteStrnDisplayLength()	*n* バイトの範囲内のすべての文字が占有する表示位置の大きさを戻します。
OCIMultiByteStrCaseConversion()	文字列の一部をあるキャラクタ・セットから別のキャラクタ・セットに変換します。

OCIMultiByteToWideChar()

構文

```
sword OCIMultiByteToWideChar(dvoid *hndl, OCIWchar *dst, CONST OraText *src, size_t
*rsize);
```

用途

このルーチンは、ヌル文字で終了する文字列全体を wchar 書式に変換します。wchar アウトプット・バッファは、ヌル文字で終了します。OCIEnvNlsCreate 関数で SQL CHAR データに OCI_UTF16ID を指定すると、この関数でエラーが生成されます。

戻り値

OCI_SUCCESS、OCI_INVALID_HANDLE または OCI_ERROR

パラメータ

hndl（IN/OUT）
文字列のキャラクタ・セットを決定するための OCI 環境ハンドルまたはユーザー・セッション・ハンドル。

dst（OUT）
wchar の宛先バッファ。

src（IN）
変換するソース文字列。

rsize（OUT）
変換された文字数（ヌル終端文字を含む）。NULL ポインタの場合は、戻り値がないことを意味します。

OCIMultiByteInSizeToWideChar()

構文

```
sword OCIMultiByteInSizeToWideChar(dvoid *hndl, OCIWchar *dst, size_t dstsz, CONST
OraText *src, size_t srcsz, size_t *rsize)
```

用途

このルーチンは、文字列の一部を wchar 書式に変換します。アウトプット・バッファ・サイズの上限または入力バッファ・サイズの上限に到達するまで、あるいはソース文字列がヌル終端文字に到達するまで、すべての文字を変換します。出力バッファは、空き領域がある場合はヌル文字で終了します。dstsz が 0（ゼロ）の場合、この関数は変換された文字列に必要な文字数（ヌル終端文字を含まない）のみを戻します。OCIEnvNlsCreate 関数で SQL CHAR データに OCI_UTF16ID を指定すると、この関数でエラーが生成されます。

戻り値

OCI_SUCCESS、OCI_INVALID_HANDLE または OCI_ERROR

パラメータ

hndl（IN/OUT）
文字列のキャラクタ・セットを決定するための OCI 環境ハンドルまたはユーザー・セッション・ハンドル。

dst（OUT）
wchar の宛先バッファへのポインタ。dstsz が 0（ゼロ）の場合は、NULL ポインタになります。

dstsz（IN）
宛先バッファ・サイズ（文字単位）。0（ゼロ）の場合、この関数は変換に必要な文字数のみを戻します。

src（IN）
変換するソース文字列。

srcsz（IN）
ソース文字列の長さ（バイト単位）。

rsize（OUT）
宛先バッファに書き込まれた文字数、または dstsz が 0（ゼロ）の場合は、変換された文字列に必要な文字数。NULL ポインタの場合は、戻り値がないことを意味します。

OCIWideCharToMultiByte()

構文

```
sword OCIWideCharToMultiByte(dvoid *hndl, OraText *dst, CONST OCIWchar *src, size_t
*rsize)
```

用途

このルーチンは、NULL で終了するワイド・キャラクタ文字列全体をマルチバイト文字列に変換します。出力バッファは、ヌル文字で終了します。OCIEnvNlsCreate 関数で SQL CHAR データに OCI_UTF16ID を指定すると、この関数でエラーが生成されます。

戻り値

OCI_SUCCESS、OCI_INVALID_HANDLE または OCI_ERROR

パラメータ

hndl（IN/OUT）
文字列のキャラクタ・セットを決定するための OCI 環境ハンドルまたはユーザー・セッション・ハンドル。

dst（OUT）
マルチバイト文字列の宛先バッファ。

src（IN）
変換するソース wchar 文字列。

srcsz（IN）
ソース文字列の長さ（文字単位）。

rsize（OUT）
宛先バッファに書き込まれたバイト数。NULL ポインタの場合は、戻り値がないことを意味します。

OCIWideCharInSizeToMultiByte()

構文

```
sword OCIWideCharInSizeToMultiByte(dvoid *hndl, OraText *dst, size_t dstsz, CONST
OCIWchar *src, size_t srcsz, size_t *rsize)
```

用途

このルーチンは、wchar 文字列の一部をマルチバイト書式に変換します。出力バッファ・サイズまたは入力バッファ・サイズに到達するまで、あるいはソース文字列がヌル終端文字に到達するまで、すべての文字を変換します。出力バッファは、空き領域がある場合はヌル文字で終了します。dstsz が 0（ゼロ）の場合、この関数は変換された文字列を格納するために必要なサイズ（ヌル終端文字を含まない）のみをバイト単位で戻します。
OCIEnvNlsCreate 関数で SQL CHAR データに OCI_UTF16ID を指定すると、この関数でエラーが生成されます。

戻り値

OCI_SUCCESS、OCI_INVALID_HANDLE または OCI_ERROR

パラメータ

hndl（IN/OUT）
文字列のキャラクタ・セットを決定するための OCI 環境ハンドルまたはユーザー・セッション・ハンドル。

dst（OUT）
マルチバイトの宛先バッファ。dstsz が 0（ゼロ）の場合は、NULL ポインタになります。

dstsz（IN）
宛先バッファ・サイズ（バイト単位）。0（ゼロ）の場合は、変換された文字列に必要なサイズ（バイト単位）を戻します。

src（IN）
変換するソース wchar 文字列。

srcsz（IN）
ソース文字列の長さ（文字単位）。

rsize（OUT）
宛先バッファに書き込まれたバイト数、または dstsz が 0（ゼロ）の場合は、変換された文字列を格納するために必要なバイト数。NULL ポインタの場合は、戻り値がないことを意味します。

OCIWideCharToLower()

構文

```
OCIWchar OCIWideCharToLower(dvoid *hndl, OCIWchar wc)
```

用途

この関数は、wc で指定された wchar 文字を、指定されたロケールに存在する場合は対応する小文字に変換します。対応する小文字が存在しない場合は、wc 自体を戻します。OCIEnvNlsCreate() 関数で SQL CHAR データに OCI_UTF16ID を指定すると、この関数でエラーが生成されます。

戻り値

wchar

パラメータ

hndl（IN/OUT）
キャラクタ・セットを決定するための OCI 環境ハンドルまたはユーザー・セッション・ハンドル。

wc（IN）
小文字に変換する wchar。

OCIWideCharToUpper()

構文

```
OCIWchar OCIWideCharToUpper(dvoid *hndl, OCIWchar wc)
```

用途

この関数は、wc で指定された wchar 文字を、指定されたロケールに存在する場合は対応する大文字に変換します。対応する大文字が存在しない場合は、wc 自体を戻します。OCIEnvNlsCreate() 関数で SQL CHAR データに OCI_UTF16ID を指定すると、この関数でエラーが生成されます。

戻り値

wchar

パラメータ

hndl（IN/OUT）
キャラクタ・セットを決定するための OCI 環境ハンドルまたはユーザー・セッション・ハンドル。

wc（IN）
大文字に変換する wchar。

OCIWideCharStrcmp()

構文

```
int OCIWideCharStrcmp(dvoid *hndl, CONST OCIWchar *wstr1, CONST OCIWchar *wstr2, int
flag)
```

用途

2 つの wchar 文字列を、（wchar のエンコーディング値に基づいた）バイナリ、言語または大 / 小文字を区別しない比較方法で比較します。OCIEnvNlsCreate 関数で SQL CHAR データに OCI_UTF16ID を指定すると、この関数でエラーが生成されます。

戻り値

- wstr1 = wstr2 の場合は 0（ゼロ）
- wstr1 > wstr2 の場合は正の値
- wstr1 < wstr2 の場合は負の値

パラメータ

hndl（IN/OUT）
キャラクタ・セットを決定するための OCI 環境ハンドルまたはユーザー・セッション・ハンドル。

wstr1（IN）
ヌル文字で終了する wchar 文字列へのポインタ。

wstr2（IN）
ヌル文字で終了する wchar 文字列へのポインタ。

flag（IN）
比較方法を決定するために使用されるフラグ。次のいずれかの値を指定できます。

- OCI_NLS_BINARY: バイナリ比較。これがデフォルト値です。
- OCI_NLS_LINGUISTIC: ロケールの定義に指定されている言語比較。

大 / 小文字を区別せずに比較するには、このフラグを OCI_NLS_CASE_INSENSITIVE とともに使用します。たとえば、OCI_NLS_LINGUISTIC|OCI_NLS_CASE_INSENSITIVE を使用すると、大 / 小文字を区別せずに文字列の言語上の比較を行うことができます。

OCIWideCharStrncmp()

構文

```
int OCIWideCharStrncmp(dvoid *hndl, CONST OCIWchar *wstr1, size_t len1, CONST
OCIWchar *wstr2, size_t len2, int flag)
```

用途

この関数は OCIWideCharStrcmp() に類似しており、2 つのワイド・キャラクタ文字列を、バイナリ、言語または大 / 小文字を区別しない比較方法で比較します。wstr1 から最大 len1 バイト、wstr2 から最大 len2 バイトが比較されます。比較には、ヌル終端文字が使用されます。OCIEnvNlsCreate 関数で SQL CHAR データに OCI_UTF16ID を指定すると、この関数でエラーが生成されます。

戻り値

- wstr1 = wstr2 の場合は 0（ゼロ）
- wstr1 > wstr2 の場合は正の値
- wstr1 < wstr2 の場合は負の値

パラメータ

hndl（IN/OUT）
キャラクタ・セットを決定するための OCI 環境ハンドルまたはユーザー・セッション・ハンドル。

wstr1（IN）
最初の wchar 文字列へのポインタ。

len1（IN）
比較する最初の文字列の長さ。

wstr2（IN）
2 番目の wchar 文字列へのポインタ。

len2（IN）
比較する 2 番目の文字列の長さ。

flag（IN）
比較方法を決定するために使用されるフラグ。次のいずれかの値を指定できます。

- OCI_NLS_BINARY: バイナリでの比較。これは、デフォルト値です。
- OCI_NLS_LINGUISTIC: ロケールで指定された言語上の比較。

大 / 小文字を区別せずに比較するには、このフラグを OCI_NLS_CASE_INSENSITIVE とともに使用します。たとえば、OCI_NLS_LINGUISTIC|OCI_NLS_CASE_INSENSITIVE を使用すると、大 / 小文字を区別せずに文字列の言語上の比較を行うことができます。

OCIWideCharStrcat()

構文

```
size_t OCIWideCharStrcat(dvoid *hndl, OCIWchar *wdststr, CONST OCIWchar *wsrcstr)
```

用途

この関数は、wsrcstr で示された wchar 文字列のコピーを、wdststr で示された wchar 文字列にヌル終端文字も含めて追加します。OCIEnvNlsCreate 関数で SQL CHAR データに OCI_UTF16ID を指定すると、この関数でエラーが生成されます。

戻り値

結果文字列の文字数（ヌル終端文字を含まない）

パラメータ

hndl（IN/OUT）
キャラクタ・セットを決定するための OCI 環境ハンドルまたはユーザー・セッション・ハンドル。

wdststr（IN/OUT）
追加する宛先の wchar 文字列へのポインタ。

wsrcstr（IN）
追加するソース wchar 文字列へのポインタ。

OCIWideCharStrncat()

構文

```
size_t OCIWideCharStrncat(dvoid *hndl, OCIWchar *wdststr, CONST OCIWchar *wsrcstr,
size_t n)
```

用途

この関数は、OCIWideCharStrcat() に類似しています。wsrcstr から wdststr に追加されるのは最大 *n* 文字までです。wsrcstr にヌル終端文字があると、そこで追加処理は停止します。wdststr はヌル文字で終了します。OCIEnvNlsCreate 関数で SQL CHAR データに OCI_UTF16ID を指定すると、この関数でエラーが生成されます。

戻り値

結果文字列の文字数（ヌル終端文字を含まない）

パラメータ

hndl（IN/OUT）
キャラクタ・セットを決定するための OCI 環境ハンドルまたはユーザー・セッション・ハンドル。

wdststr（IN/OUT）
追加する宛先 wchar 文字列へのポインタ。

wsrcstr（IN）
追加するソース wchar 文字列へのポインタ。

n（IN）
wsrcstr から追加する文字数。

OCIWideCharStrchr()

構文

```
OCIWchar *OCIWideCharStrchr(dvoid *hndl, CONST OCIWchar *wstr, OCIWchar wc)
```

用途

この関数は、wstr で示された wchar 文字列の中で、最初に出現する wc を検索します。OCIEnvNlsCreate 関数で SQL CHAR データに OCI_UTF16ID を指定すると、この関数でエラーが生成されます。

戻り値

正常終了した場合は wchar へのポインタ、それ以外の場合は NULL ポインタ

パラメータ

hndl（IN/OUT）
キャラクタ・セットを決定するための OCI 環境ハンドルまたはユーザー・セッション・ハンドル。

wstr（IN）
検索するソース wchar 文字列へのポインタ。

wc（IN）
検索する wchar。

OCIWideCharStrrchr()

構文

```
OCIWchar *OCIWideCharStrrchr(dvoid *hndl, CONST OCIWchar *wstr, OCIWchar wc)
```

用途

この関数は、wstr で示された wchar 文字列の中で、最後に出現する wc を検索します。OCIEnvNlsCreate 関数で SQL CHAR データに OCI_UTF16ID を指定すると、この関数でエラーが生成されます。

戻り値

正常終了した場合は wchar へのポインタ、それ以外の場合は NULL ポインタ

パラメータ

hndl（IN/OUT）
キャラクタ・セットを決定するための OCI 環境ハンドルまたはユーザー・セッション・ハンドル。

wstr（IN）
検索するソース wchar 文字列へのポインタ。

wc（IN）
検索する wchar。

OCIWideCharStrcpy()

構文

```
size_t OCIWideCharStrcpy(dvoid *hndl, OCIWchar *wdststr, CONST OCIWchar *wsrcstr)
```

用途

この関数は、wsrcstr で示された wchar 文字列を、wdststr で示された配列にヌル終端文字も含めてコピーします。OCIEnvNlsCreate 関数で SQL CHAR データに OCI_UTF16ID を指定すると、この関数でエラーが生成されます。

戻り値

コピーされた文字数（ヌル終端文字を含まない）

パラメータ

hndl（IN/OUT）
キャラクタ・セットを決定するための OCI 環境ハンドルまたはユーザー・セッション・ハンドル。

wdststr（OUT）
宛先 wchar バッファへのポインタ。

wsrcstr（IN）
ソース wchar 文字列へのポインタ。

OCIWideCharStrncpy()

構文

```
size_t OCIWideCharStrncpy(dvoid *hndl, OCIWchar *wdststr, CONST OCIWchar *wsrcstr,
size_t n)
```

用途

この関数は、OCIWideCharStrcpy() に類似していますが、wsrcstr で示された配列から wdststr で示された配列にコピーされるのは最大 *n* 文字までです。wdststr にヌル終端文字があると、そこでコピー処理は停止します。結果文字列はヌル文字で終了します。OCIEnvNlsCreate 関数で SQL CHAR データに OCI_UTF16ID を指定すると、この関数でエラーが生成されます。

戻り値

コピーされた文字数（ヌル終端文字を含まない）

パラメータ

hndl（IN/OUT）
キャラクタ・セットを決定するための OCI 環境ハンドルまたはユーザー・セッション・ハンドル。

wdststr（OUT）
宛先 wchar バッファへのポインタ。

wsrcstr（IN）
ソース wchar 文字列へのポインタ。

n（IN）
wsrcstr からコピーする文字数。

OCIWideCharStrlen()

構文

```
size_t OCIWideCharStrlen(dvoid *hndl, CONST OCIWchar *wstr)
```

用途

この関数は、wstr で示された wchar 文字列の文字数（ヌル終端文字を含まない）を計算し、その数を戻します。OCIEnvNlsCreate 関数で SQL CHAR データに OCI_UTF16ID を指定すると、この関数でエラーが生成されます。

戻り値

文字数（ヌル終端文字を含まない）

パラメータ

hndl（IN/OUT）
キャラクタ・セットを決定するための OCI 環境ハンドルまたはユーザー・セッション・ハンドル。

wstr（IN）
ソース wchar 文字列へのポインタ。

OCIWideCharStrCaseConversion()

構文

```
size_t OCIWideCharStrCaseConversion(dvoid *hndl, OCIWchar *wdststr, CONST
OCIWchar*wsrcstr, ub4 flag)
```

用途

この関数は、wsrcstr で示されたワイド char 文字列をフラグで指定された大文字または小文字に変換し、その結果を wdststr で示された配列にコピーします。結果文字列は、ヌル文字で終了します。OCIEnvNlsCreate 関数で SQL CHAR データに OCI_UTF16ID を指定すると、この関数でエラーが生成されます。

戻り値

結果文字列の文字数（ヌル終端文字を含まない）

パラメータ

hndl（IN/OUT）
OCI 環境ハンドルまたはユーザー・セッション・ハンドル。

wdststr（OUT）
宛先の配列へのポインタ。

wsrcstr（IN）
ソース文字列へのポインタ。

flag（IN）
大文字または小文字のどちらに変換するかを指定するフラグ。

- OCI_NLS_UPPERCASE: 大文字への変換。
- OCI_NLS_LOWERCASE: 小文字への変換。

ロケールでの言語設定を大 / 小文字の変換で使用するには、このフラグとともに OCI_NLS_LINGUISTIC を指定します。

OCIWideCharDisplayLength()

構文

```
size_t OCIWideCharDisplayLength(dvoid *hndl, OCIWchar wc)
```

用途

この関数は、wc を表示するために必要な列位置の数を決定します。列位置の数、または wc がヌル終端文字の場合は 0（ゼロ）を戻します。OCIEnvNlsCreate 関数で SQL CHAR データに OCI_UTF16ID を指定すると、この関数でエラーが生成されます。

戻り値

表示位置の数

パラメータ

hndl（IN/OUT）
キャラクタ・セットを決定するための OCI 環境ハンドルまたはユーザー・セッション・ハンドル。

wc（IN）
wchar 文字。

OCIWideCharMultiByteLength()

構文

```
size_t OCIWideCharMultiByteLen(dvoid *hndl, OCIWchar wc)
```

用途

この関数は、wc をマルチバイト・エンコーディングで表す場合に必要なバイト数を決定します。OCIEnvNlsCreate 関数で SQL CHAR データに OCI_UTF16ID を指定すると、この関数でエラーが生成されます。

戻り値

wc に必要なバイト数

パラメータ

hndl（IN/OUT）
キャラクタ・セットを決定するための OCI 環境ハンドルまたはユーザー・セッション・ハンドル。

wc（IN）
wchar 文字。

OCIMultiByteStrcmp()

構文

```
int OCIMultiByteStrcmp(dvoid *hndl, CONST OraText *str1, CONST OraText *str2, int
flag)
```

用途

2 つのマルチバイト・キャラクタ文字列を、バイナリ、言語または大 / 小文字を区別しない比較方法で比較します。OCIEnvNlsCreate 関数で SQL CHAR データに OCI_UTF16ID を指定すると、この関数でエラーが生成されます。

戻り値

- str1 = str2 の場合は 0（ゼロ）
- str1 > str2 の場合は正の値
- str1 < str2 の場合は負の値

パラメータ

hndl（IN/OUT）
OCI 環境ハンドルまたはユーザー・セッション・ハンドル。

str1（IN）
ヌル文字で終了する文字列へのポインタ。

str2（IN）
ヌル文字で終了する文字列へのポインタ。

flag（IN）
比較方法を決定するために使用されるフラグ。次のいずれかの値を指定できます。

- OCI_NLS_BINARY: バイナリ比較。これがデフォルト値です。
- OCI_NLS_LINGUISTIC: ロケールで指定されている言語比較。

大 / 小文字を区別せずに比較するには、このフラグを OCI_NLS_CASE_INSENSITIVE とともに使用します。たとえば、OCI_NLS_LINGUISTIC|OCI_NLS_CASE_INSENSITIVE を使用すると、大 / 小文字を区別せずに文字列の言語上の比較を行うことができます。

OCIMultiByteStrncmp()

構文

```
int OCIMultiByteStrncmp(dvoid *hndl, CONST OraText *str1, size_t len1, OraText
*str2, size_t len2, int flag)
```

用途

この関数は、OCIMultiByteStrcmp() に類似していますが、str1 の先頭 len1 バイトと str2 の先頭 len2 バイトのみが比較されます。比較には、ヌル終端文字が使用されます。OCIEnvNlsCreate 関数で SQL CHAR データに OCI_UTF16ID を指定すると、この関数でエラーが生成されます。

戻り値

- str1 = str2 の場合は 0（ゼロ）
- str1 > str2 の場合は正の値
- str1 < str2 の場合は負の値

パラメータ

hndl（IN/OUT）
OCI 環境ハンドルまたはユーザー・セッション・ハンドル。

str1（IN）
最初の文字列へのポインタ。

len1（IN）
比較する最初の文字列の長さ。

str2（IN）
2 番目の文字列へのポインタ。

len2（IN）
比較する 2 番目の文字列の長さ。

flag（IN）
比較方法を決定するために使用されるフラグ。次のいずれかの値を指定できます。

- OCI_NLS_BINARY: バイナリ比較。これがデフォルト値です。
- OCI_NLS_LINGUISTIC: ロケールで指定されている言語比較。

大 / 小文字を区別せずに比較するには、このフラグを OCI_NLS_CASE_INSENSITIVE とともに使用します。たとえば、OCI_NLS_LINGUISTIC|OCI_NLS_CASE_INSENSITIVE を使用すると、大 / 小文字を区別せずに文字列の言語上の比較を行うことができます。

OCIMultiByteStrcat()

構文

```
size_t OCIMultiByteStrcat(dvoid *hndl, OraText *dststr, CONST OraText *srcstr)
```

用途

この関数は、srcstr で示されたマルチバイト文字列のコピーを、dststr で示された文字列の最後にヌル終端文字も含めて追加します。OCIEnvNlsCreate 関数で SQL CHAR データに OCI_UTF16ID を指定すると、この関数でエラーが生成されます。

戻り値

結果文字列のバイト数（ヌル終端文字を含まない）

パラメータ

hndl（IN/OUT）
キャラクタ・セットを決定するための OCI 環境ハンドルまたはユーザー・セッション・ハンドル。

dststr（IN/OUT）
追加する宛先のマルチバイト文字列へのポインタ。

srcstr（IN）
追加するソース文字列へのポインタ。

OCIMultiByteStrncat()

構文

```
size_t OCIMultiByteStrncat(dvoid *hndl, OraText *dststr, CONST OraText *srcstr,
size_t n)
```

用途

この関数は OCIMultiByteStrcat() に類似しています。srcstr から最大 *n* バイトが dststr に追加されます。srcstr にヌル終端文字があると、そこで追加処理は停止し、*n* バイト内でできるだけ多くの文字が追加されます。dststr は、ヌル文字で終了します。OCIEnvNlsCreate 関数で SQL CHAR データに OCI_UTF16ID を指定すると、この関数でエラーが生成されます。

戻り値

結果文字列のバイト数（ヌル終端文字を含まない）

パラメータ

hndl（IN/OUT）
OCI 環境ハンドルまたはユーザー・セッション・ハンドルへのポインタ。

dststr（IN/OUT）
追加する宛先のマルチバイト文字列へのポインタ。

srcstr（IN）
追加するソース・マルチバイト文字列へのポインタ。

n（IN）
srcstr から追加するバイト数。

OCIMultiByteStrcpy()

構文

```
size_t OCIMultiByteStrcpy(dvoid *hndl, OraText *dststr, CONST OraText *srcstr)
```

用途

この関数は、srcstr で示されたマルチバイト文字列を、dststr で示された配列にヌル終端文字も含めてコピーします。OCIEnvNlsCreate 関数で SQL CHAR データに OCI_UTF16ID を指定すると、この関数でエラーが生成されます。

戻り値

コピーされたバイト数（ヌル終端文字を含まない）

パラメータ

hndl（IN/OUT）
OCI 環境ハンドルまたはユーザー・セッション・ハンドルへのポインタ。

dststr（OUT）
宛先バッファへのポインタ。

srcstr（IN）
ソース・マルチバイト文字列へのポインタ。

OCIMultiByteStrncpy()

構文

```
size_t OCIMultiByteStrncpy(dvoid *hndl, OraText *dststr, CONST OraText *srcstr,
size_t n)
```

用途

この関数は OCIMultiByteStrcpy() に類似しています。srcstr で示された配列から最大 *n* バイトが、dststr で示された配列にコピーされます。srcstr にヌル終端文字があると、そこでコピー処理は停止し、*n* バイト内でできるだけ多くの文字がコピーされます。結果文字列は、ヌル文字で終了します。OCIEnvNlsCreate 関数で SQL CHAR データに OCI_UTF16ID を指定すると、この関数でエラーが生成されます。

戻り値
コピーされたバイト数（ヌル終端文字を含まない）

パラメータ

hndl（IN/OUT）
OCI 環境ハンドルまたはユーザー・セッション・ハンドルへのポインタ。

srcstr（OUT）
宛先バッファへのポインタ。

dststr（IN）
ソース・マルチバイト文字列へのポインタ。

n（IN）
srcstr からコピーするバイト数。

OCIMultiByteStrlen()

構文
```
size_t OCIMultiByteStrlen(dvoid *hndl, CONST OraText *str)
```

用途
この関数は、str で示されたマルチバイト文字列のバイト数（ヌル終端文字を含まない）を戻します。OCIEnvNlsCreate 関数で SQL CHAR データに OCI_UTF16ID を指定すると、この関数でエラーが生成されます。

戻り値
バイト数（ヌル終端文字を含まない）

パラメータ

hndl（IN/OUT）
OCI 環境ハンドルまたはユーザー・セッション・ハンドルへのポインタ。

str（IN）
ソース・マルチバイト文字列へのポインタ。

OCIMultiByteStrnDisplayLength()

構文

```
size_t OCIMultiByteStrnDisplayLength(dvoid *hndl, CONST OraText *str1, size_t n)
```

用途

この関数は、*n* バイトの範囲内のすべての文字が占有する表示位置の数を戻します。
OCIEnvNlsCreate 関数で SQL CHAR データに OCI_UTF16ID を指定すると、この関数でエラーが生成されます。

戻り値

表示位置の数

パラメータ

hndl（IN/OUT）
OCI 環境ハンドルまたはユーザー・セッション・ハンドル。

str（IN）
マルチバイト文字列へのポインタ。

n（IN）
検査するバイト数。

OCIMultiByteStrCaseConversion()

構文

```
size_t OCIMultiByteStrCaseConversion(dvoid *hndl, OraText *dststr, CONST OraText
*srcstr, ub4 flag)
```

用途

この関数は、srcstr で示されたマルチバイト文字列をフラグで指定された大文字または小文字に変換し、その結果を dststr で示された配列にコピーします。結果文字列は、ヌル文字で終了します。OCIEnvNlsCreate 関数で SQL CHAR データに OCI_UTF16ID を指定すると、この関数でエラーが生成されます。

戻り値

結果文字列のバイト数（ヌル終端文字を含まない）

パラメータ

hndl（IN/OUT）
OCI 環境ハンドルまたはユーザー・セッション・ハンドル。

dststr（OUT）
宛先の配列へのポインタ。

srcstr（IN）
ソース文字列へのポインタ。

flag（IN）
大文字または小文字のどちらに変換するかを指定するフラグ。

- OCI_NLS_UPPERCASE: 大文字への変換。
- OCI_NLS_LOWERCASE: 小文字への変換。

ロケールでの言語設定を大 / 小文字の変換で使用するには、このフラグとともに OCI_NLS_LINGUISTIC を指定します。

例 : OCI での文字列操作

次の例に、文字列操作の単純な例を示します。

```
size_t MyConvertMultiByteToWideChar(envhp, dstBuf, dstSize, srcStr)
OCIEnv     *envhp;
OCIWchar   *dstBuf;
size_t      dstSize;
OraText       *srcStr;                          /* null terminated source string */
{
  sword  ret;
  size_t dstLen = 0;
  size_t srcLen;

  /* get length of source string */
  srcLen = OCIMultiByteStrlen(envhp, srcStr);

  ret = OCIMultiByteInSizeToWideChar(envhp,       /* environment handle */
                dstBuf,                               /* destination buffer */
                dstSize,                          /* destination buffer size */
                srcStr,                                    /* source string */
                srcLen,                           /* length of source string */
                &dstLen);                 /* pointer to destination length */

  if (ret != OCI_SUCCESS)
  {
    checkerr(envhp, ret, OCI_HTYPE_ENV);
  }
  return(dstLen);
}
```

関連項目 : 『Oracle Call Interface プログラマーズ・ガイド』

OCI での文字の分類

表 8-2 に、OCI の文字の分類関数を示します。各関数の詳細は後述します。

表 8-2 OCI の文字の分類関数

関数	説明
OCIWideCharIsAlnum()	ワイド・キャラクタが、文字であるか 10 進数字であるかをテストします。
OCIWideCharIsAlpha()	ワイド・キャラクタが、アルファベット文字であるかどうかをテストします。
OCIWideCharIsCntrl()	ワイド・キャラクタが、制御文字であるかどうかをテストします。
OCIWideCharIsDigit()	ワイド・キャラクタが、10 進数字であるかどうかをテストします。
OCIWideCharIsGraph()	ワイド・キャラクタが、図形文字であるかどうかをテストします。
OCIWideCharIsLower()	ワイド・キャラクタが、小文字であるかどうかをテストします。
OCIWideCharIsPrint()	ワイド・キャラクタが、印字可能文字であるかどうかをテストします。
OCIWideCharIsPunct()	ワイド・キャラクタが、句読点文字であるかどうかをテストします。
OCIWideCharIsSpace()	ワイド・キャラクタが、スペース文字であるかどうかをテストします。
OCIWideCharIsUpper()	ワイド・キャラクタが、大文字であるかどうかをテストします。
OCIWideCharIsXdigit()	ワイド・キャラクタが、16 進数字であるかどうかをテストします。
OCIWideCharIsSingleByte()	wc がマルチバイトに変換されるとき、シングルバイト文字であるかどうかをテストします。

OCIWideCharIsAlnum()

構文

```
boolean OCIWideCharIsAlnum(dvoid *hndl, OCIWchar wc)
```

用途

wc が、文字であるか 10 進数字であるかをテストします。

戻り値

TRUE または FALSE

パラメータ

hndl（IN/OUT）
キャラクタ・セットを決定するための OCI 環境ハンドルまたはユーザー・セッション・ハンドル。

wc（IN）
テストする wchar。

OCIWideCharIsAlpha()

構文

```
boolean OCIWideCharIsAlpha(dvoid *hndl, OCIWchar wc)
```

用途

wc が、アルファベット文字であるかどうかをテストします。

戻り値

TRUE または FALSE

パラメータ

hndl（IN/OUT）
キャラクタ・セットを決定するための OCI 環境ハンドルまたはユーザー・セッション・ハンドル。

wc（IN）
テストする wchar。

OCIWideCharIsCntrl()

構文

```
boolean OCIWideCharIsCntrl(dvoid *hndl, OCIWchar wc)
```

用途

wc が、制御文字であるかどうかをテストします。

戻り値

TRUE または FALSE

パラメータ

hndl（IN/OUT）
キャラクタ・セットを決定するための OCI 環境ハンドルまたはユーザー・セッション・ハンドル。

wc（IN）
テストする wchar。

OCIWideCharIsDigit()

構文

```
boolean OCIWideCharIsDigit(dvoid *hndl, OCIWchar wc)
```

用途

wc が、10 進数字であるかどうかをテストします。

戻り値

TRUE または FALSE

パラメータ

hndl（IN/OUT）
キャラクタ・セットを決定するための OCI 環境ハンドルまたはユーザー・セッション・ハンドル。

wc（IN）
テストする wchar。

OCIWideCharIsGraph()

構文

```
boolean OCIWideCharIsGraph(dvoid *hndl, OCIWchar wc)
```

用途

wc が、図形文字であるかどうかをテストします。図形文字は、視覚表現を持つ文字で、通常、アルファベット文字、10 進数字および句読点がこれに含まれます。

戻り値

TRUE または FALSE

パラメータ

hndl（IN/OUT）
キャラクタ・セットを決定するための OCI 環境ハンドルまたはユーザー・セッション・ハンドル。

wc（IN）
テストする wchar。

OCIWideCharIsLower()

構文

```
boolean OCIWideCharIsLower(dvoid *hndl, OCIWchar wc)
```

用途

wc が、小文字であるかどうかをテストします。

戻り値

TRUE または FALSE

パラメータ

hndl（IN/OUT）
キャラクタ・セットを決定するための OCI 環境ハンドルまたはユーザー・セッション・ハンドル。

wc（IN）
テストする wchar。

OCIWideCharIsPrint()

構文

```
boolean OCIWideCharIsPrint(dvoid *hndl, OCIWchar wc)
```

用途

wc が、印字可能文字であるかどうかをテストします。

戻り値

TRUE または FALSE

パラメータ

hndl（IN/OUT）
キャラクタ・セットを決定するための OCI 環境ハンドルまたはユーザー・セッション・ハンドル。

wc（IN）
テストする wchar。

OCIWideCharIsPunct()

構文

```
boolean OCIWideCharIsPunct(dvoid *hndl, OCIWchar wc)
```

用途

wc が、句読点文字であるかどうかをテストします。

戻り値

TRUE または FALSE

パラメータ

hndl（IN/OUT）
キャラクタ・セットを決定するための OCI 環境ハンドルまたはユーザー・セッション・ハンドル。

wc（IN）
テストする wchar。

OCIWideCharIsSpace()

構文

```
boolean OCIWideCharIsSpace(dvoid *hndl, OCIWchar wc)
```

用途

wc が、スペース文字であるかどうかをテストします。スペース文字とは、表示テキスト上では文字として表示されないもの（空白、タブ、改行、垂直タブ、改ページなど）となるものです。

戻り値

TRUE または FALSE

パラメータ

hndl（IN/OUT）
キャラクタ・セットを決定するための OCI 環境ハンドルまたはユーザー・セッション・ハンドル。

wc（IN）
テストする wchar。

OCIWideCharIsUpper()

構文

```
boolean OCIWideCharIsUpper(dvoid *hndl, OCIWchar wc)
```

用途

wc が、大文字であるかどうかをテストします。

戻り値

TRUE または FALSE

パラメータ

hndl（IN/OUT）
キャラクタ・セットを決定するための OCI 環境ハンドルまたはユーザー・セッション・ハンドル。

wc（IN）
テストする wchar。

OCIWideCharIsXdigit()

構文

```
boolean OCIWideCharIsXdigit(dvoid *hndl, OCIWchar wc)
```

用途

wc が、16 進数字（0 ～ 9、A ～ F、a ～ f）であるかどうかをテストします。

戻り値

TRUE または FALSE

パラメータ

hndl（IN/OUT）
キャラクタ・セットを決定するための OCI 環境ハンドルまたはユーザー・セッション・ハンドル。

wc（IN）
テストする wchar。

OCIWideCharIsSingleByte()

構文

```
boolean OCIWideCharIsSingleByte(dvoid *hndl, OCIWchar wc)
```

用途

wc がマルチバイトに変換されるとき、シングルバイト文字であるかどうかをテストします。

戻り値

TRUE または FALSE

パラメータ

hndl（IN/OUT）
キャラクタ・セットを決定するための OCI 環境ハンドルまたはユーザー・セッション・ハンドル。

wc（IN）
テストする wchar。

例 : OCI での文字の分類

次の例に、OCI での文字の分類方法を示します。

```
boolean MyIsNumberWideCharString(envhp, srcStr)
OCIEnv   *envhp;
OCIWchar *srcStr;                                   /* wide char source string */
{
  OCIWchar *pstr = srcStr;                          /* define and init pointer */
  boolean status = TRUE;            /* define and initialize status variable */

  /* Check input */
  if (pstr == (OCIWchar*) NULL)
    return(FALSE);


  if (*pstr == (OCIWchar) NULL)
    return(FALSE);

                                                 /* check each character for digit */
  do
  {
    if (OCIWideCharIsDigit(envhp, *pstr) != TRUE)
    {
      status = FALSE;
      break;                                       /* non-decimal digit character */
    }
  } while ( *++pstr != (OCIWchar) NULL);

  return(status);
}
```

OCI でのキャラクタ・セットの変換

Oracle キャラクタ・セットと Unicode（16 ビットの固定幅 Unicode エンコーディング）間の変換がサポートされています。Unicode から Oracle キャラクタ・セットへのマッピングがない場合は、置換文字が使用されます。したがって、データ消失なしで元のキャラクタ・セットへ戻る変換が常に可能なわけではありません。

表 8-3 に、OCI のキャラクタ・セット変換関数を示します。各関数の詳細は後述します。

表 8-3 OCI キャラクタ・セット変換関数

関数	説明
OCICharsetToUnicode()	src で示されたマルチバイト文字列を Unicode に変換し、dst で示された配列に格納します。
OCIUnicodeToCharset()	src で示された Unicode 文字列をマルチバイトに変換し、dst で示された配列に格納します。
OCINlsCharSetConvert()	文字列をあるキャラクタ・セットから別のキャラクタ・セットに変換します。
OCICharSetConversionIsReplacementUsed()	OCINlsCharsetConvert() または OCICharSetToUnicode() の最後の起動時に変換できなかった文字に、置換文字が使用されたかどうかを示します。

OCICharsetToUnicode()

構文

```
sword OCICharSetToUnicode(dvoid *hndl, ub2 *dst, size_t dstlen, CONST OraText *src,
size_t srclen, size_t *rsize)
```

用途

この関数は、src で示されたマルチバイト文字列を Unicode に変換し、dst で示された配列に格納します。変換元文字列サイズまたは変換先文字列サイズに到達すると、変換は停止します。関数は、Unicode 文字列に変換された文字数を戻します。dstlen が 0 の場合、この関数は文字列をスキャンして文字数をカウントし、その数を rsize に戻しますが、文字列の変換は行いません。

OCIEnvNlsCreate 関数で SQL CHAR データに OCI_UTF16ID を指定すると、この関数でエラーが生成されます。

戻り値

OCI_SUCCESS、OCI_INVALID_HANDLE または OCI_ERROR

パラメータ

hndl（IN/OUT）
OCI 環境ハンドルまたはユーザー・セッション・ハンドルへのポインタ。

dst（OUT）
宛先バッファへのポインタ。

dstlen（IN）
宛先バッファのサイズ（文字単位）。

src（IN）
ソース・マルチバイト文字列へのポインタ。

srclen（IN）
ソース文字列のサイズ（バイト単位）。

rsize（OUT）
変換された文字数。NULL ポインタの場合は、戻り値がないことを意味します。

OCIUnicodeToCharset()

構文

```
sword OCIUnicodeToCharSet(dvoid *hndl, OraText *dst, size_t dstlen, CONST ub2 *src,
size_t srclen, size_t *rsize)
```

用途

この関数は、src で示された Unicode 文字列をマルチバイト文字列に変換し、dst で示された配列に格納します。変換元文字列サイズまたは変換先文字列サイズに到達すると、変換は停止します。関数は、マルチバイト文字列に変換されたバイト数を戻します。dstlen が 0（ゼロ）の場合は、変換せずに rsize にバイト数を戻します。

Unicode 文字を OCI 環境ハンドルまたはユーザー・セッション・ハンドルで指定されたキャラクタ・セットで変換できない場合は、置換文字が使用されます。この場合、OCICharsetConversionIsReplacementUsed() は TRUE を戻します。

OCIEnvNlsCreate 関数で SQL CHAR データに OCI_UTF16ID を指定すると、この関数でエラーが生成されます。

戻り値

OCI_SUCCESS、OCI_INVALID_HANDLE または OCI_ERROR

パラメータ

hndl（IN/OUT）
OCI 環境ハンドルまたはユーザー・セッション・ハンドルへのポインタ。

dst（OUT）
宛先バッファへのポインタ。

dstlen（IN）
宛先バッファのサイズ（バイト単位）。

src（IN）
Unicode 文字列へのポインタ。

srclen（IN）
ソース文字列のサイズ（文字単位）。

rsize（OUT）
変換されたバイト数。NULL ポインタの場合は、戻り値がないことを意味します。

OCINlsCharSetConvert()

構文

```
sword OCINlsCharSetConvert(dvoid *envhp, OCIError *errhp,ub2 dstid, dvoid *dstp,
size_t dstlen,ub2 srcid, CONST dvoid *srcp, size_tsrclen, size_t *rsize);
```

用途

この関数は、src で示された文字列を srcid で指定されたキャラクタ・セットから dstid で指定されたキャラクタ・セットに変換し、dst で示された配列に格納します。ソースまたは宛先のデータ・サイズ制限に達すると、変換処理が停止します。関数は、宛先バッファに変換されたバイト数を戻します。ソースまたは宛先のキャラクタ・セット ID を OCI_UTF16ID として指定できますが、オリジナル・データと変換済みデータの長さは、文字数ではなくバイト数で表されます。NULL データがあっても変換は停止しないことに注意してください。キャラクタ・セット名からキャラクタ・セット ID を取得するには、OCINlsCharSetNameToId() を使用します。宛先バッファ内の導出データに置換文字が含まれているかどうかをチェックするには、OCICharSetConversionIsReplacementUsed() を使用します。バッファは、キャラクタ・セットに適切なバイト境界に位置合せする必要があります。たとえば、UTF-16 で文字列を保持するには、ub2 データ型を使用してください。

戻り値

OCI_SUCCESS または OCI_ERROR、変換されたバイト数

パラメータ

errhp（IN/OUT）
OCI エラー・ハンドル。エラーがある場合は、errhp に記録され、NULL ポインタが戻されます。診断情報は OCIErrorGet() をコールすると取得できます。

dstid（IN）
宛先バッファのキャラクタ・セット ID。

dstp（OUT）
宛先バッファへのポインタ。

dstlen（IN）
宛先バッファの最大サイズ（バイト単位）。

srcid（IN）
ソース・バッファのキャラクタ・セット ID。

srcp（IN）
ソース・バッファへのポインタ。

srclen（IN）
ソース・バッファのバイト長。

rsize（OUT）
変換された文字数。NULL ポインタの場合は、戻り値がないことを意味します。

OCICharSetConversionIsReplacementUsed()

構文

```
boolean OCICharSetConversionIsReplacementUsed(dvoid *hndl)
```

用途

この関数は、OCICharSetToUnicode() または OCICharSetConvert() の最後の起動時に変換できなかった文字に、置換文字が使用されたかどうかを示します。

戻り値

OCICharSetConvert() または OCICharSetToUnicode() の最後の起動時に置換文字が使用された場合、この関数は TRUE を戻します。それ以外の場合は FALSE を戻します。

パラメータ

hndl（IN/OUT）
OCI 環境ハンドルまたはユーザー・セッション・ハンドルへのポインタ。

Oracle キャラクタ・セットと Unicode（16 ビットの固定幅 Unicode エンコーディング）間の変換がサポートされています。Unicode から Oracle キャラクタ・セットへの文字マッピングがない場合は、置換文字が使用されます。つまり、すべての文字に元の文字へのラウンドトリップ変換が可能なわけではありません。特定の文字にはデータ消失が発生します。

例 : OCI でのキャラクタ・セットの変換

次の例に、Unicode への変換の簡単な例を示します。

```
size_t MyConvertMultiByteToUnicode(envhp, dstBuf, dstSize, srcStr)
OCIEnv *envhp;
ub2    *dstBuf;
size_t  dstSize;
OraText   *srcStr;
{
  sword  ret;
  size_t dstLen = 0;
  size_t srcLen;

  /* get length of source string */
  srcLen = OCIMultiByteStrlen(envhp, srcStr);


  ret = OCICharSetToUnicode(envhp,                    /* environment handle */
                dstBuf,                               /* destination buffer */
                dstSize,                        /* size of destination buffer */
                srcStr,                                    /* source string */
                srcLen,                           /* length of source string */
                &dstLen);                  /* pointer to destination length */

  if (ret != OCI_SUCCESS)
  {
    checkerr(envhp, ret, OCI_HTYPE_ENV);
  }
  return(dstLen);
}
```

OCI メッセージ関数

ユーザー・メッセージ API には、カートリッジ開発者のための簡易インタフェースが用意されています。これによって、Oracle メッセージと同様にユーザー自身のメッセージを取り出すことができます。

関連項目：『Oracle9*i* Data Cartridge Developer's Guide』

表 8-4 に、OCI メッセージ関数を示します。

表 8-4 OCI メッセージ関数

関数	説明
OCIMessageOpen()	hndl で示された言語で、メッセージ・ハンドルをオープンします。
OCIMessageGet()	msgno で識別されるメッセージ番号の付いたメッセージを取り出します。バッファが 0（ゼロ）でない場合、この関数は msgbuf で示されたバッファにメッセージをコピーします。
OCIMessageClose()	msgh で示されたメッセージ・ハンドルをクローズし、そのハンドルに関連付けられているメモリーをすべて解放します。

この項の内容は、次のとおりです。

OCIMessageOpen()

構文

```
sword OCIMessageOpen(dvoid *hndl, OCIError *errhp, OCIMsg **msghp, CONST OraText
*product, CONST OraText *facility, OCIDuration dur)
```

用途

この関数は、hndl で示された言語で、メッセージ処理機能をオープンします。最初に、hndl に対応するメッセージ・ファイルをオープンします。メッセージ・ファイルのオープンに成功すると、そのファイルを使用してメッセージ・ハンドルを初期化します。言語に対応するメッセージ・ファイルが見つからない場合は、代替として主要な言語ファイルが検索

されます。たとえば、スペイン語（南米）ファイルが見つからない場合は、スペイン語ファイルのオープンが試行されます。代替に失敗すると、デフォルトのメッセージ・ファイルが使用されます。このファイルの言語は AMERICAN です。この関数は、メッセージ・ハンドルへのポインタを msghp パラメータに戻します。

戻り値

OCI_SUCCESS、OCI_INVALID_HANDLE または OCI_ERROR

パラメータ

hndl（IN/OUT）
メッセージ言語用の OCI 環境ハンドルまたはユーザー・セッション・ハンドルへのポインタ。

errhp（IN/OUT）
OCI エラー・ハンドル。エラーがある場合は、errhp に記録され、NULL ポインタが戻されます。診断情報は OCIErrorGet() をコールすると取得できます。

msghp（OUT）
戻されるメッセージ・ハンドル。

product（IN）
製品名へのポインタ。製品名を使用して、メッセージ用のディレクトリが検索されます。ディレクトリの位置は、オペレーティング・システムに依存します。たとえば、Solaris の場合、製品 rdbms のメッセージ・ファイルのディレクトリは $ORACLE_HOME/rdbms です。

facility（IN）
製品の機能名へのポインタ。これは、メッセージ・ファイル名を構成するために使用されます。メッセージ・ファイル名には変換言語が使用され、接頭辞として facility が付きます。たとえば、img 機能の米語でのメッセージ・ファイル名は、imgus.msb になります。この場合、us は米語の略称で、msb はメッセージのバイナリ・ファイルという意味の拡張子です。

dur（IN）
戻されたメッセージ・ハンドルのメモリー割当て継続期間。次のいずれかの値を使用できます。

```
OCI_DURATION_PROCESS
OCI_DURATION_SESSION
OCI_DURATION_STATEMENT
```

OCIMessageGet()

構文

```
OraText *OCIMessageGet(OCIMsg *msgh, ub4 msgno, OraText *msgbuf, size_t buflen)
```

用途

この関数は、msgno で識別されるメッセージ番号の付いたメッセージを取得します。buflen が 0（ゼロ）でない場合は、msgbuf で示されたバッファにメッセージをコピーします。buflen が 0（ゼロ）の場合、取得したメッセージは msgh で示されたメッセージ・ハンドル内のメッセージ・バッファにコピーされます。

戻り値

ヌル文字で終了するメッセージ文字列へのポインタ。翻訳済みメッセージが見つからない場合は、等価の英語メッセージが戻されます。等価の英語メッセージが見つからない場合は、NULL ポインタが戻されます。

パラメータ

msgh（IN/OUT）
OCIMessageOpen() によってあらかじめオープンされているメッセージ・ハンドルへのポインタ。

msgno（IN）
メッセージを取得するためのメッセージ番号。

msgbuf（OUT）
取り出したメッセージを格納する宛先バッファへのポインタ。buflen が 0（ゼロ）の場合は、NULL ポインタになります。

buflen（IN）
宛先バッファのサイズ。

OCIMessageClose()

構文

```
sword OCIMessageClose(dvoid *hndl, OCIError *errhp, OCIMsg *msgh)
```

用途

この関数は、msgh で示されたメッセージ・ハンドルをクローズし、そのハンドルに関連付けられているメモリーをすべて解放します。

戻り値

OCI_SUCCESS、OCI_INVALID_HANDLE または OCI_ERROR

パラメータ

hndl（IN/OUT）
メッセージ言語用の OCI 環境ハンドルまたはユーザー・セッション・ハンドルへのポインタ。

errhp（IN/OUT）
OCI エラー・ハンドル。エラーがある場合は、errhp に記録され、NULL ポインタが戻されます。診断情報は OCIErrorGet() をコールすると取得できます。

msgh（IN/OUT）
OCIMessageOpen() によってあらかじめオープンされているメッセージ・ハンドルへのポインタ。

例 : テキスト・メッセージ・ファイルからのメッセージの取出し

この例では、メッセージ・ハンドルを作成し、impus.msg からメッセージを取り出すように初期化し、メッセージ番号 128 を取り出して、メッセージ・ハンドルをクローズします。この例では、OCI 環境ハンドル、OCI セッション・ハンドル、製品、機能およびキャッシュ・サイズが適切に初期化されているものとします。

```
OCIMsg msghnd;                                                  /* message handle */
         /* initialize a message handle for retrieving messages from impus.msg*/
err = OCIMessageOpen(hndl,errhp, &msghnd, prod,fac,OCI_DURATION_SESSION);
if (err != OCI_SUCCESS)
                                                                /* error handling */
...
                              /* retrieve the message with message number = 128 */
msgptr = OCIMessageGet(msghnd, 128, msgbuf, sizeof(msgbuf));
                           /* do something with the message, such as display it */
...
      /* close the message handle when there are no more messages to retrieve */
OCIMessageClose(hndl, errhp, msghnd);
```

lmsgen ユーティリティ

用途

lmsgen ユーティリティは、テキスト・ベースのメッセージ・ファイル（.msg）をバイナリ形式（.msb）に変換します。これにより、ユーザーが提供する Oracle メッセージと OCI メッセージを、必要な言語で OCI 関数に戻すことができます。

構文

LMSGEN *text_file product facility* [*language*]

text_file は、メッセージ・テキスト・ファイルです。
product は、製品名です。
facility は、機能名です。

language は、NLS_LANG パラメータで指定した言語に対応するオプションのメッセージ言語です。language パラメータは、メッセージ・ファイルが言語に正しくタグ付けされていない場合に必要です。

テキスト・メッセージ・ファイル

テキスト・メッセージ・ファイルは、次のガイドラインに従う必要があります。

- / および // で始まる行は内部コメントとして扱われ、無視されます。
- メッセージ・ファイルを特定の言語でタグ付けするには、次のような 1 行を追加します。

```
#   CHARACTER_SET_NAME= Japanese_Japan.JA16EUC
```

- 各メッセージには、次の 3 つのフィールドがあります。

```
message_number, warning_level, message_text
```

  message_number は、メッセージ・ファイル内で一意であることが必要です。
  warning_level は、現在使用されていません。0（ゼロ）を使用してください。
  message_text は、76 バイトを超えることはできません。

次に、Oracle メッセージ・テキスト・ファイルの例を示します。

```
/ Copyright (c) 2001 by the Oracle Corporation.  All rights reserved.
/ This is a test us7ascii message file
# CHARACTER_SET_NAME= american_america.us7ascii
/
00000, 00000, "Export terminated unsuccessfully\n"
00003, 00000, "no storage definition found for segment(%lu, %lu)"
```

例 : テキスト・メッセージ・ファイルからのバイナリ・メッセージ・ファイルの作成

次の表に、lmsgen パラメータのサンプル値を示します。

パラメータ	値
product	$HOME/myApplication
facility	imp
language	AMERICAN
text_file	impus.msg

テキスト・メッセージ・ファイルは、次の場所にあります。

```
$HOME/myApp/mesg/impus.msg
```

テキスト・メッセージ・ファイル内の 1 行は、次のとおりです。

```
00128,2, "Duplicate entry %s found in %s"
```

lmsgen ユーティリティは、テキスト・メッセージ・ファイル（impus.msg）をバイナリ形式のファイル impus.msb に変換します。

```
% lmsgen impus.msg $HOME/myApplication imp AMERICAN
```

出力は次のようになります。

```
Generating message file impus.msg -->
/home/scott/myApplication/mesg/impus.msb

NLS Binary Message File Generation Utility: Version 9.2.0.1.0 -Production

Copyright (c) Oracle Corporation 1979, 2001.  All rights reserved.

CORE    9.2.0.1.0       Production
```

9

グローバル環境での Java プログラミング

この章では、各 Java コンポーネントに対するグローバリゼーション・サポートについて説明します。この章の内容は、次のとおりです。

- Oracle9i Java サポートの概要
- JDBC ドライバのグローバリゼーション・サポート
- SQLJ のグローバリゼーション・サポート
- Java Virtual Machine のグローバリゼーション・サポート
- Java ストアド・プロシージャのグローバリゼーション・サポート
- 多言語アプリケーションの構成
- SQLJ の多言語デモ・アプリケーション

Oracle9*i* Java サポートの概要

Java プログラムの開発と配布を可能にするために、Java サポートは、複数層コンピューティング環境のすべての階層に組み込まれています。Java クラスは、Oracle9*i* データベースの Java Virtual Machine（Oracle JVM）上で、Java ストアド・プロシージャとして実行できます。Java クラスを開発してデータベースにロードし、SQL からコールできるストアド・プロシージャとしてパッケージ化できます。

Java プログラムから Oracle9*i* データベースへのアクセスを可能にするプログラム・インタフェースとして、JDBC ドライバおよび SQLJ Translator も提供されています。ユーザーは、JDBC または SQLJ プログラムの埋込み SQL 文を使用して、データベースにアクセスする Java アプリケーションを記述できます。すべての Java コンポーネントには、グローバリゼーション・サポートが提供されています。これによって、キャラクタ・セットと言語環境の異なる複数のデータベースで、Java コンポーネントが正しく機能することが保証され、Oracle9*i* の多言語 Java アプリケーションの開発と配布を可能にしています。

この章では、各 Java コンポーネントに対するグローバリゼーション・サポートについて説明します。多言語アプリケーション配布のための典型的なデータベースとクライアントの構成およびその構成での Java コンポーネントの使用方法を説明します。Oracle の Java サポートによって多言語環境でアプリケーションがどのように実行されているかを示すために、サンプル・アプリケーションの設計と実装について説明します。

Java コンポーネントはグローバリゼーション・サポートを提供し、多言語キャラクタ・セットとして Unicode を使用します。表 9-1 に、Oracle9*i* の Java コンポーネントを示します。

表 9-1 Oracle9*i* の Java コンポーネント

Java コンポーネント	説明
JDBC ドライバ	Oracle は、Oracle9*i* データベースにアクセスするためのプログラム・インタフェースの中核として、JDBC を提供します。Oracle が提供する JDBC ドライバは 4 つあり、2 つはクライアント・アクセス用、2 つはサーバー・アクセス用です。 ■ JDBC OCI ドライバは、Java アプリケーションが使用します。 ■ JDBC Thin ドライバは、主に Java アプレットが使用します。 ■ サーバー側 Oracle JDBC Thin ドライバは、クライアント側 JDBC Thin ドライバと同じ機能を提供します。リモート・データベースにアクセスするために、主にデータベース・サーバーの JVM 上で動作する Java クラスで使用されます。 サーバー側 JDBC 内部ドライバはサーバー側のドライバで、データベース・サーバーの JVM 上で動作する Java クラスで使用されます。

表 9-1 Oracle9*i* の Java コンポーネント（続き）

Java コンポーネント	説明
SQLJ translator	SQLJ には、SQLJ プログラム・ファイル内の埋込み SQL を JDBC コールを使用した Java ソース・ファイルに変換するという、プリプロセッサのような役割があります。このトランスレータは、データベースにアクセスするための高度なプログラム・インタフェースをプログラマに提供します。
Java Virtual Machine（JVM）	JVM は、JDK に基づくもので、データベース・サーバーに統合されて、Java クラスを Java ストアド・プロシージャとして実行可能にします。データベースに格納された Java クラスを管理するライブラリ・マネージャなどの、サポート用サービス・セットが付属しています。

JDBC ドライバのグローバリゼーション・サポート

Oracle JDBC ドライバは、Oracle9*i* データベースの SQL CHAR データ型および SQL NCHAR データ型の列との間でデータの取出しと挿入を可能にすることによって、グローバリゼーション・サポートを提供します。Java の文字列は、JDBC プログラム用の UTF-16（16 ビット Unicode）でエンコードされているため、クライアントのターゲット・キャラクタ・セットは常に UTF-16 です。CHAR、VARCHAR2、LONG および CLOB データ型で格納されているデータは、JDBC によって、データベース・キャラクタ・セットから UTF-16 に透過的に変換されます。NCHAR、NVARCHAR2 および NCLOB データ型で格納されている Unicode データは、JDBC によって、各国語キャラクタ・セットから UTF-16 に透過的に変換されます。

次に、キャラクタ・セット変換に関係の深い JDBC 用の Java メソッドのうち、一般的なものをいくつか示します。

- `java.sql.ResultSet` クラスの `getString()` メソッドは、データベースからの値を Java 文字列として戻します。
- `java.sql.ResultSet` クラスの `getUnicodeStream()` メソッドは、値を Unicode 文字のストリームとして戻します。
- `oracle.sql.CLOB` クラスの `getSubString()` メソッドは、CLOB の内容を Unicode ストリームとして戻します。
- `oracle.sql.CHAR` クラスの `getString()`、`toString()` および `getStringWithReplacement()` メソッドは、オブジェクトからの値を Java 文字列として戻します。

データベースの接続時、JDBC クラス・ライブラリは、サーバーの NLS_LANGUAGE および NLS_TERRITORY パラメータを、JDBC ドライバを実行する JVM のロケールに対応する値に設定します。この操作は、JDBC OCI ドライバと JDBC Thin ドライバでのみ実行され、サーバーと Java クライアントが確実に同じ言語で通信できるようになります。その結果、サーバーから戻される Oracle エラー・メッセージは、クライアントのロケールと同じ言語になります。

この項の内容は、次のとおりです。

JDBC を使用した SQL CHAR データ型へのアクセス

SQL CHAR データ型のデータベース列に Java 文字列を挿入するには、`PreparedStatement.setString()` メソッドを使用してバインド変数を指定します。Oracle の JDBC ドライバは、Java 文字列をデータベース・キャラクタ・セットに透過的に変換します。次の例に、Java 文字列 `last_name` を `VARCHAR2` 列 `last_name` にバインドする方法を示します。

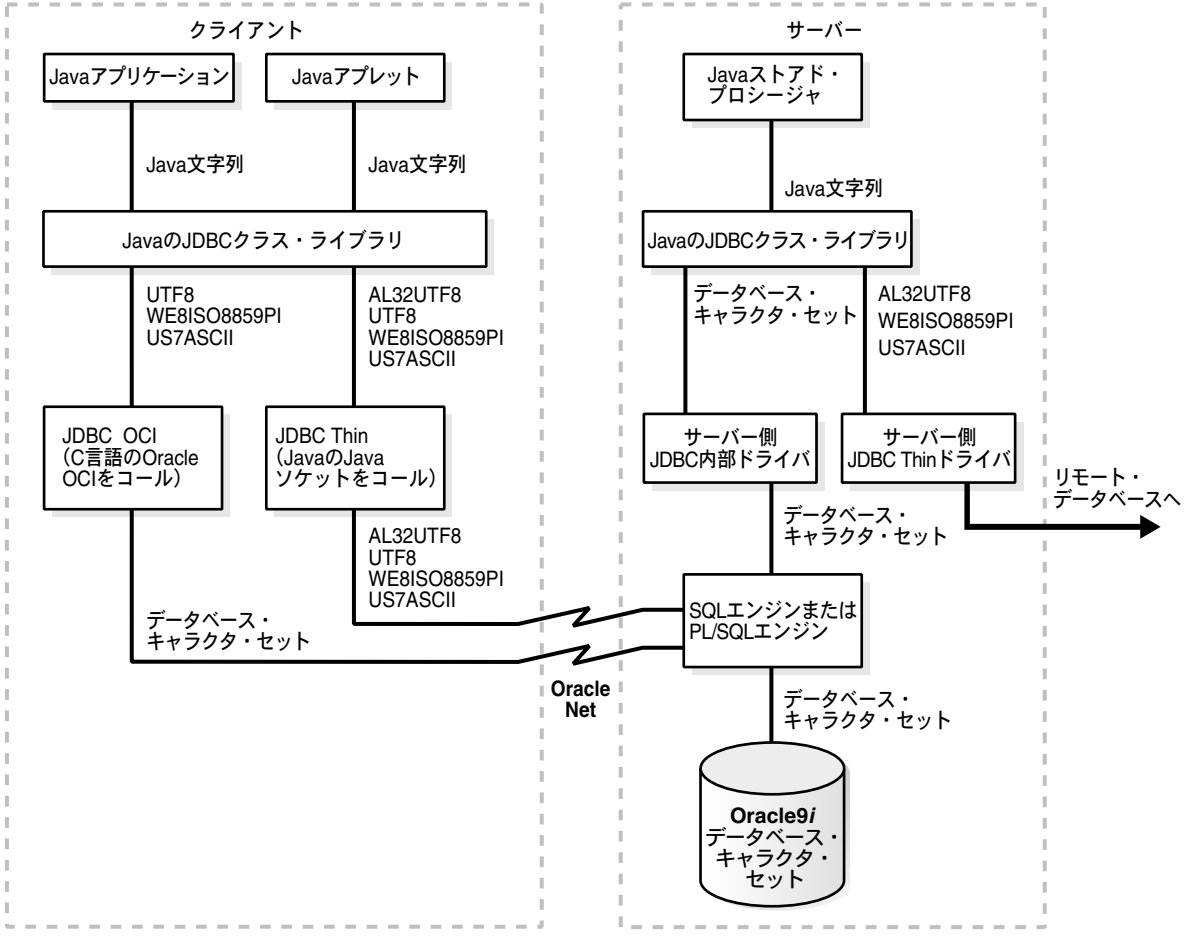
```
int employee_id= 12345;
String last_name= "\uFF2A\uFF4F\uFF45";
PreparedStatement pstmt =
    conn.prepareStatement ("INSERT INTO employees (employee_id, last_name)
    VALUES(?,?)");
pstmt.setInt(1, employee_id);
pstmt.setString(2, last_name);
pstmt.execute();
pstmt.close();
```

SQL CHAR データ型で格納されているデータの場合、Oracle のドライバが Java アプリケーション用に実行するキャラクタ・セットの変換方法は、データベースで使用されているキャラクタ・セットによって異なります。最も変換が簡単なのは、データベースが US7ASCII または WE8ISO8859P1 キャラクタ・セットを使用している場合です。この場合、ドライバは、データをデータベース・キャラクタ・セットから Java アプリケーションで使用される UTF-16 に直接変換します。

US7ASCII または WE8ISO8859P1 以外のキャラクタ・セット（JA16SJIS や KO16KSC5601 など）を使用するデータベースを取り扱う場合、ドライバは、最初にデータを UTF-8 に変換してから UTF-16 に変換します。次の項では、異なる JDBC ドライバの変換の流れについて説明します。

図 9-1 に、JDBC ドライバでのデータの変換方法を示します。

図 9-1 JDBC ドライバのデータ変換



JDBC クラス・ライブラリのキャラクタ・セット変換

JDBC クラス・ライブラリは、JDBC インタフェースを実装する Java レイヤーです。このレイヤーと対話するのは、Java アプリケーション、アプレットおよびストアド・プロシージャです。ライブラリは、常に US7ASCII、UTF8 または WE8ISO8859P1 でエンコードされた文字列データを、JDBC ドライバの入力ストリームから受け入れます。また、JDBC Thin ドライバの場合は AL32UTF8 データを、サーバー側 JDBC ドライバの場合はデータベース・キャラクタ・セットを受け入れます。JDBC クラス・ライブラリは、入力ストリームを UTF-16 に変換してから、クライアント・アプリケーションに渡します。AL32UTF8 は、UTF-8 エン

コーディングで Unicode 文字のエンコーディングに UTF8 とともに使用されるキャラクタ・セットであり、補足的な Unicode 文字をサポートしています。入力ストリームが UTF8 または AL32UTF8 の場合、JDBC クラス・ライブラリは、UTF-8 から UTF-16 への変換アルゴリズムで定義されたビット単位処理を使用して、UTF8 または AL32UTF8 でエンコードされた文字列を UTF-16 に変換します。入力ストリームが US7ASCII または WE8ISO8859P1 の場合は、バイトを Java キャラクタにキャストすることによって、入力文字列を UTF-16 に変換します。入力ストリームが US7ASCII、WE8ISO8859P1、UTF8 および AL32UTF8 のいずれでもない場合、JDBC クラス・ライブラリは Oracle キャラクタ・セット変換機能をコールすることで入力ストリームを変換します。この変換の流れは、サーバー側 JDBC ドライバにのみ使用されます。

JDBC OCI ドライバのキャラクタ・セット変換

JDBC OCI ドライバの場合は、データベース・キャラクタ・セットの他に、クライアント側のキャラクタ・セットがあります。クライアント・キャラクタ・セットは、クライアントの起動時に、クライアント上の環境変数 NLS_LANG の値によって決定されます。データベース・キャラクタ・セットは、データベースの作成時に決定されます。クライアントが使用するキャラクタ・セットは、サーバーのデータベースが使用するキャラクタ・セットと異なる場合があります。キャラクタ・セット変換の実行時、JDBC OCI ドライバは、次の 3 つを考慮する必要があります。

- データベースのキャラクタ・セットと言語
- クライアントのキャラクタ・セットと言語
- Java アプリケーションのキャラクタ・セット

JDBC OCI ドライバは、データベースのキャラクタ・セットで、データをサーバーからクライアントに転送します。環境変数 NLS_LANG の値に従って、ドライバは、次のいずれかの方法でキャラクタ・セット変換を処理します。

- NLS_LANG に値が指定されていない場合、あるいは US7ASCII または WE8ISO8859P1 キャラクタ・セットが設定されている場合、JDBC OCI ドライバは Java を使用して、JDBC クラス・ライブラリ内でキャラクタ・セットを US7ASCII または WE8ISO8859P1 から UTF-16 に直接変換します。
- NLS_LANG の値が US7ASCII または WE8ISO8859P1 以外のキャラクタ・セットに設定されている場合、ドライバはクライアント・キャラクタ・セットとして UTF8 を使用します。この変更は自動的に行われ、ユーザーの介入は不要です。その後、OCI はデータをデータベース・キャラクタ・セットから UTF8 に変換します。この UTF8 データは、JDBC OCI ドライバから JDBC クラス・ライブラリに渡され、そこで UTF-16 に変換されます。

JDBC Thin ドライバのキャラクタ・セット変換

アプリケーションまたはアプレットで JDBC Thin ドライバを使用する場合、Oracle クライアントはインストールされません。このため、C 言語の OCI クライアント変換ルーチンは使用できません。この場合、JDBC Thin ドライバのクライアント変換ルーチンは、JDBC OCI ドライバの変換ルーチンとは異なるものになります。

データベース・キャラクタ・セットが US7ASCII、WE8ISO8859P1、UTF8 または AL32UTF8 の場合、データは変換されずにクライアントに転送されます。JDBC クラス・ライブラリは、データを Java で UTF-16 に変換します。

それ以外の場合、サーバーは、クライアントに転送する前にデータを UTF8 または AL32UTF8 に変換します。クライアントでは、そのデータを JDBC クラス・ライブラリが Java で UTF-16 に変換します。

サーバー側 JDBC 内部ドライバのキャラクタ・セット変換

Java クラスは、Oracle9*i* サーバーの JVM 上で動作するため、SQL 処理で SQL エンジンまたは PL/SQL エンジンと対話するには、サーバー側 JDBC 内部ドライバを使用します。サーバー側 JDBC 内部ドライバは、Oracle サーバー・プロセスと同じアドレス空間で動作するため、SQL エンジンまたは PL/SQL エンジンに対してはローカル関数コールを行います。SQL エンジンまたは PL/SQL エンジンとの間で送受信されるデータは、データベース・キャラクタ・セットでエンコードされます。サーバー側 JDBC 内部ドライバは変換を行わず、データはそのまま JDBC クラス・ライブラリとの間でやりとりされます。必要な変換は、JDBC クラス・ライブラリに委譲されます。

JDBC を使用した SQL NCHAR データ型へのアクセス

JDBC を使用すると、Oracle9*i* データベースの SQL NCHAR データ型の列に Java プログラムでアクセスできます。SQL NCHAR データ型のデータ変換は、SQL CHAR データ型のデータ変換とは異なります。すべての Oracle JDBC ドライバは、SQL NCHAR 列のデータを、各国語キャラクタ・セット（UTF8 または AL16UTF16）から UTF-16 でエンコードされた Java 文字列に直接変換します。次の Java プログラムでは、Java 文字列 `last_name` を NVARCHAR2 列 `last_name` にバインドできます。

```
int employee_id = 12345;
String ename = "\uFF2A\uFF4F\uFF45";
oracle.jdbc.OraclePreparedStatement pstmt =
    (oracle.jdbc.OraclePreparedStatement)
    conn.prepareStatement("INSERT INTO employees (empoyee_id, last_name) VALUES (?,
?)");
pstmt.setFormOfUse(2, oracle.jdbc.OraclePreparedStatement.FORM_NCHAR);
pstmt.setInt(1, employee_id);
pstmt.setString(2, last_name);
pstmt.execute();
pstmt.close();
```

関連項目： SQL NCHAR データ型に対するプログラミングの詳細は、6-27 ページの「Unicode での Java 文字列のバインドと定義」を参照してください。

oracle.sql.CHAR クラスの使用

oracle.sql.CHAR クラスには、文字データを変換するための特別な機能があります。Oracle キャラクタ・セットは、oracle.sql.CHAR クラスの主な属性です。oracle.sql.CHAR オブジェクトの構成時には、常に Oracle キャラクタ・セットが渡されます。既知のキャラクタ・セットがない場合、oracle.sql.CHAR オブジェクトのデータのバイトには意味がありません。

oracle.sql.CHAR クラスは、文字データを文字列に変換するために、次のメソッドを提供します。

- getString()

  oracle.sql.CHAR オブジェクトが表現する一連の文字を文字列に変換し、Java String オブジェクトを戻します。キャラクタ・セットが認識されない場合、getString() は SQLException を戻します。

- toString()

  getString() に類似していますが、キャラクタ・セットが認識されない場合、toString() は oracle.sql.CHAR データを 16 進表現で戻し、SQLException は戻しません。

- getStringWithReplacement()

  getString() に類似していますが、この oracle.sql.CHAR オブジェクトのキャラクタ・セットに Unicode 表現のない文字は、デフォルトの置換文字で置き換えられます。デフォルトの置換文字は、キャラクタ・セットによって異なりますが、通常は疑問符（?）です。

oracle.sql.CHAR オブジェクトを（プリコンパイルされた SQL 文に渡すなどの目的で）手動で構成できます。oracle.sql.CHAR オブジェクトを構成する場合は、oracle.sql.CharacterSet クラスのインスタンスを使用して、oracle.sql.CHAR オブジェクトにキャラクタ・セット情報を提供する必要があります。oracle.sql.CharacterSet クラスの各インスタンスは、Oracle でサポートしているキャラクタ・セットの 1 つを表します。

次のタスクを実行して oracle.sql.CHAR オブジェクトを構成します。

1. 静的 CharacterSet.make() メソッドをコールして、CharacterSet インスタンスを作成します。これは、キャラクタ・セットのクラスを作成するメソッドで、入力として有効な Oracle キャラクタ・セット（OracleId）が必要です。次に例を示します。

   ```
   int OracleId = CharacterSet.JA16SJIS_CHARSET; // this is character set 832
   ...
   CharacterSet mycharset = CharacterSet.make(OracleId);
   ```

   Oracle でサポートされている各キャラクタ・セットには、一意の事前定義済み OracleId があります。OracleId は、*Oracle_character_set_name*_CHARSET として指定したキャラクタ・セットとしていつでも参照できます。*Oracle_character_set_name* は、Oracle キャラクタ・セットです。

2. oracle.sql.CHAR オブジェクトを構成します。コンストラクタに、文字列（または文字列を表現するバイト）と、キャラクタ・セットに基づくバイトの解析方法を示す CharacterSet オブジェクトを渡します。次に例を示します。

   ```
   String mystring = "teststring";
   ...
   oracle.sql.CHAR mychar = new oracle.sql.CHAR(teststring, mycharset);
   ```

   oracle.sql.CHAR クラスには複数のコンストラクタがあり、CharacterSet オブジェクトとともに文字列、バイト配列またはオブジェクトを入力としてとることができます。文字列の場合は、oracle.sql.CHAR オブジェクトに置かれる前に、CharacterSet オブジェクトが示すキャラクタ・セットに変換されます。

サーバー（データベース）とクライアント（またはクライアントで実行中のアプリケーション）は、異なるキャラクタ・セットを使用できます。このクラスのメソッドを使用してサーバーとクライアントの間でデータを転送する場合、JDBC ドライバはサーバー・キャラクタ・セットとクライアント・キャラクタ・セットの間でデータを変換する必要があります。

oracle.sql.CHAR クラスを使用したデータの挿入と取出し

バインド変数を oracle.sql.CHAR オブジェクトとして取得するために OracleResultSet.getCHAR() メソッドをコールすると、JDBC は、データベースから文字データが読み出された後で、oracle.sql.CHAR オブジェクトの構成および移入を行います。同様に、OraclePreparedStatement.sql.CHAR() メソッドをコールすると、oracle.sql.CHAR オブジェクトを使用してバインド変数を設定できます。次に例を示します。

```
int employee_id = 12345;
String ename = "\uFF2A\uFF4F\uFF45";
String eaddress = "Address of \uFF2A\uFF4F\uFF45";
/* CharacterSet object for VARCHAR2 column */
CharacterSet dbCharset = CharacterSet.make(CharacterSet.JA16SJIS_CHARSET);
```

```
/* CharacterSet object for NVARCHAR2 column */
CharacterSet ncCharset = CharacterSet.make(CharacterSet.AL16UTF16_CHARSET);

/* last_name is in VARCHAR2 and address is in NVARCHAR2 */
oracle.jdbc.OraclePreparedStatement pstmt =
    (oracle.jdbc.OraclePreparedStatement)
    conn.prepareStatement("INSERT INTO employees (empoyee_id, last_name,
address)
    VALUES (?, ?, ?)");
pstmt.setFormOfUse(3, oracle.jdbc.OraclePreparedStatement.FORM_NCHAR);
pstmt.setInt(1, employee_id);
pstmt.setCHAR(2, new oracle.sql.CHAR(ename, dbCharset));
pstmt.setCHAR(3, new oracle.sql.CHAR(eaddress, ncCharset));
pstmt.execute();
pstmt.close();
```

Oracle オブジェクト型の oracle.sql.CHAR

Oracle9*i* では、JDBC ドライバが Oracle オブジェクト型をサポートしています。Oracle オブジェクトは、データベース・キャラクタ・セットで表現したオブジェクトとして、常にデータベースからクライアントに送信されます。つまり、図 9-1 に示したデータ変換の流れは、Oracle オブジェクトのアクセスには適用されないことを意味します。かわりに、データベースからクライアントにオブジェクト型の SQL CHAR および SQL NCHAR データを送信するために、`oracle.sql.CHAR` クラスが使用されます。次に、SQL でオブジェクト型を作成する例を示します。

```
CREATE TYPE person_type AS OBJECT (name VARCHAR2(30), address NVARCHAR(256), age
NUMBER);
CREATE TABLE employees (id NUMBER, person PERSON_TYPE);
```

このオブジェクト型に対応する Java クラスは、次のように構成できます。

```
public class person implement SqlData
{
   oracle.sql.CHAR name;
   oracle.sql.CHAR address;
   oracle.sql.NUMBER age;
   // SqlData interfaces
   getSqlType() {...}
   writeSql(SqlOutput stream) {...}
   readSql(SqlInput stream, String sqltype) {...}
}
```

ここでは、oracle.sql.CHAR クラスは、Oracle オブジェクト型の NAME 属性（VARCHAR2 データ型）にマップするために使用されています。JDBC は、このクラスに、データベースの VARCHAR2 データのバイト表現およびデータベース・キャラクタ・セットに対応する CharacterSet オブジェクトを導入します。次のコードでは、employees 表から person オブジェクトを取り出しています。

```
TypeMap map = ((OracleConnection)conn).getTypeMap();
map.put("PERSON_TYPE", Class.forName("person"));
conn.setTypeMap(map);
    .       .       .
    .       .       .
ResultSet rs = stmt.executeQuery("SELECT PERSON FROM EMPLOYEES");
rs.next();
person p = (person) rs.getObject(1);
oracle.sql.CHAR sql_name = p.name;
oracle.sql.CHAR sql_address=p.address;
String java_name = sql_name.getString();
String java_name = sql_address.getString();
```

oracle.sql.CHAR クラスの getString() メソッドは、Oracle の Java データ変換クラスをコールして Java 文字列を戻すことで、データベース・キャラクタ・セットのバイト配列を UTF-16 に変換します。rs.getObject(1) コールが動作するように、SqlData インタフェースをクラス person に実装し、オブジェクト型 PERSON_TYPE の Java クラスへのマッピングを示すように Typemap map を設定する必要があります。

JDBC を使用した場合の SQL CHAR データへのアクセスの制限

この項の内容は、次のとおりです。

JDBC Thin ドライバを使用した場合の SQL CHAR データ・サイズの制限

データベース・キャラクタ・セットが ASCII（US7ASCII）または ISO Latin-1（WE8ISO8859P1）のいずれでもない場合、JDBC Thin ドライバでは、SQL CHAR のバインド・パラメータに対して、通常のデータベース・サイズ制限より厳しい制限を設定する必要があります。このサイズ制限は、変換時のデータ拡張を許容するために必要な制限です。

JDBC Thin ドライバは、set*XXX*() メソッド（setCharacterStream() メソッド以外）がコールされるときに、SQL CHAR のバインド・サイズをチェックします。データ・サイズがサイズ制限を超えている場合、ドライバは、set*XXX*() コールから SQL 例外（データ・サイズがこの型の最大サイズを超えています。）を戻します。この制限は、文字データ変換でデータ長が増加した場合に、データの破損を防ぐために必要です。この制限は、次の場合に規定されます。

- JDBC Thin ドライバを使用する
- （定義ではなく）バインドを使用する
- SQL CHAR データ型を使用する
- キャラクタ・セットが ASCII（US7ASCII）または ISO Latin-1（WE8ISO8859P1）のいずれでもないデータベースに接続する

データベース・キャラクタ・セットが US7ASCII または WE8ISO8859P1 のいずれでもない場合、JDBC Thin ドライバは、SQL CHAR のバインドに対して、Java UTF-16 の文字を UTF-8 エンコーディング・バイトに変換します。次に、UTF-8 エンコーディング・バイトはデータベースに転送され、データベース・キャラクタ・セット・エンコーディングに変換されます。

このキャラクタ・セット・エンコーディングへの変換によって、データの格納に必要なバイト数が増えることがあります。データベース・キャラクタ・セットの拡張因数は、UTF-8 からキャラクタ・セットへの変換での最大拡張率を示します。データベース・キャラクタ・セットが UTF8 または AL32UTF8 の場合、拡張因数（exp_factor）は 1 です。それ以外の場合、拡張因数はデータベース・キャラクタ・セットの最大文字サイズ（バイト単位）です。

表 9-2 に、SQL CHAR データのデータベース・サイズ制限、および SQL CHAR バインドの JDBC Thin ドライバ・サイズ制限の計算式を示します。データベースの制限はバイト単位です。計算式によって、UTF-8 エンコーディングの最大許容サイズ（バイト単位）を判断します。

表 9-2 SQL CHAR の最大バインド・サイズ

Oracle のバージョン	データ型	データベースの最大許容バインド・サイズ	最大バインド・サイズを判断するための計算式（UTF-8 のバイト単位）
Oracle8 以上	CHAR	2000 バイト	4000/*exp_factor*
Oracle8 以上	VARCHAR2	4000 バイト	4000/*exp_factor*
Oracle8 以上	LONG	2^{31} - 1 バイト	$(2^{31} - 1)$/*exp_factor*

この計算式は、データが UTF-8 からデータベース・キャラクタ・セットに変換された後、サイズがデータベースの最大許容サイズを超えないことを保証します。

サポート可能な UTF-16 の文字数は、データの 1 文字当たりのバイト数によって異なります。UTF-8 エンコーディングでは、ASCII 文字はすべて 1 バイトです。他のキャラクタ・タイプは、2 バイトや 3 バイトの場合があります。

表 9-3 に、一般的なサーバー・キャラクタ・セットの拡張因数を示します。この表は、各キャラクタ・セットの CHAR および VARCHAR2 データに関する JDBC Thin ドライバの最大バインド・サイズも示しています。

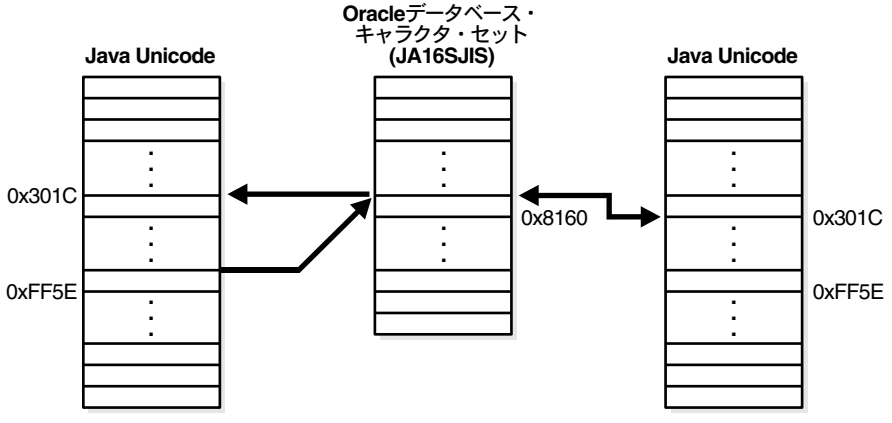
表 9-3 一般的なサーバー・キャラクタ・セットの拡張因数と最大バインド・サイズ

サーバー・キャラクタ・セット	拡張因数	SQL CHAR データに関する JDBC Thin ドライバの最大バインド・サイズ（UTF-8 のバイト単位）
WE8DEC	1	4000 バイト
JA16SJIS	2	2000 バイト
JA16EUC	3	1333 バイト
AL32UTF8	1	4000 バイト

マルチバイト・データベース環境での文字整合性の問題

Oracle JDBC ドライバは、文字データがデータベースに挿入された場合またはデータベースから取り出された場合に、適切なキャラクタ・セット変換を実行します。このドライバは、Java クライアントが使用する Unicode 文字と Oracle データベース・キャラクタ・セットの間の変換を行います。Java Unicode キャラクタ・セットとデータベース・キャラクタ・セットの間のラウンドトリップ変換で、Java に戻された文字データは、情報の一部が消失している場合があります。この一部消失は、複数の Unicode 文字が、データベース・キャラクタ・セットの 1 つの文字にマップされた場合に発生します。たとえば、Unicode の全角のチルド付き文字（0xFF5E）が、Oracle の JA16SJIS キャラクタ・セットにマップされている場合があります。Unicode 文字のラウンドトリップ変換で、この Unicode 文字は 0x301C となります。これは波形のダッシュ（日本語で、一般的に範囲を示す場合に使用されます）で、チルドではありません。

図 9-2 文字整合性



この問題は、Oracle の JDBC のバグではありません。異なるオペレーティング・システムでの文字マッピング仕様が不確定であるために発生する問題です。この問題は、一部の Oracle キャラクタ・セット（JA16SJIS、JA16EUC、ZHT16BIG5 および KO16KS5601）の、わずかな文字についてのみ発生します。問題を回避するために、これらの文字に対する完全なラウンドトリップ変換は避けてください。

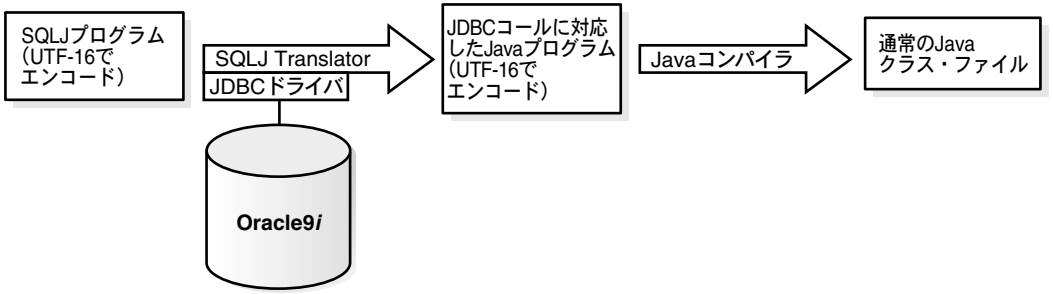
SQLJ のグローバリゼーション・サポート

SQLJ は、SQL から Java へのトランスレータです。このトランスレータは、使用する JDBC ドライバに関係なく、Java プログラムの埋込み SQL 文を対応する JDBC コールに変換します。また、Oracle9*i* データベース・サーバーがサーバー側の Java プログラムの埋込み SQL を透過的に変換するために使用する、コール可能インタフェースを提供します。SQLJ はそれ自体が Java アプリケーションで、SQLJ プログラム（埋込み SQL 文を含む Java プログラム）を読み込んで、JDBC コールに対応する Java プログラム・ファイルを生成します。変換時に埋込み SQL 文をデータベースと照合してチェックするためのチェッカを指定するオプションがあります。その後、`javac` コンパイラを使用して、生成された Java プログラム・ファイルを通常の Java クラス・ファイルにコンパイルします。

図 9-3 に、SQLJ Translator の動作を示します。この図については後述します。

- SQLJ プログラムでの Unicode 文字の使用
- oracle.sql.NString クラスの使用

図 9-3 SQLJ Translator の使用



SQLJ プログラムでの Unicode 文字の使用

SQLJ は、SQLJ ファイルを異なるコード体系（JDK でサポートされるもの）でエンコードできるようにすることで、多言語 Java アプリケーションの開発を可能にします。図 9-3 では、UTF-16 でエンコードされた SQLJ プログラムが SQLJ Translator に渡され、出力される Java プログラムも UTF-16 でエンコードされています。SQLJ は、ソースのエンコーディングをターゲットに保持します。ソースのエンコーディングを指定するには、次のように `-encoding` オプションを使用します。

```
sqlj -encoding Unicode source_file
```

Unicode 表記 `\uXXXX`（Unicode のエスケープ・シーケンス）は、埋込み SQL 文の中で、SQLJ プログラム・ファイルのエンコーディングで表現できない文字に対して使用できます。この表記によって、UTF-16 でエンコードされた SQLJ ファイルを使用せずに、SQL 文に多言語オブジェクト名を指定できます。次の SQLJ コードは、埋込み SQL および文字列リテラルでの Unicode エスケープ・シーケンスの使用方法を示しています。

```
int employee_id = 12345;
String name last_name = "\uFF2A\uFF4F\uFF45";
double raise = 0.1;

#sql { INSERT INTO E\u006D\u0070 (last_name, employee_id) VALUES (:last_name,
:employee_id)};
#sql { UPDATE employees SET salary = :(getNewSal(raise, last_name))
WHERE last_name = :last_name};
```

> **関連項目：** 多言語 Java アプリケーションでの SQLJ 使用方法の例は、9-22 ページの「SQLJ の多言語デモ・アプリケーション」を参照してください。

oracle.sql.NString クラスの使用

Oracle9*i* では、NVARCHAR2、NCHAR および NCLOB の Unicode データ型をサポートするために、SQLJ に `oracle.sql.NString` クラスが導入されています。`oracle.sql.NString` 型の Java オブジェクトを使用して、NCHAR 列のバインドを宣言し、それを SQLJ プログラムの埋込み SQL 文で使用できます。

```
int employee_id = 12345;
oracle.sql.NString last_name = new oracle.sql.NString ("\uFF2A\uFF4F\uFF45");
double raise = 0.1;
#sql { INSERT INTO E\u006D\u0070 (last_name, employee_id VALUES (:last_name,
:employee_id)};
#sql { UPDATE employees SET salary = :(getNewSal(raise, last_name)) = :last_name};
```

この例では、`oracle.sql.NString` データ型の `last_name` オブジェクトを、データベースの NVARCHAR2 列の `last_name` にバインドしています。

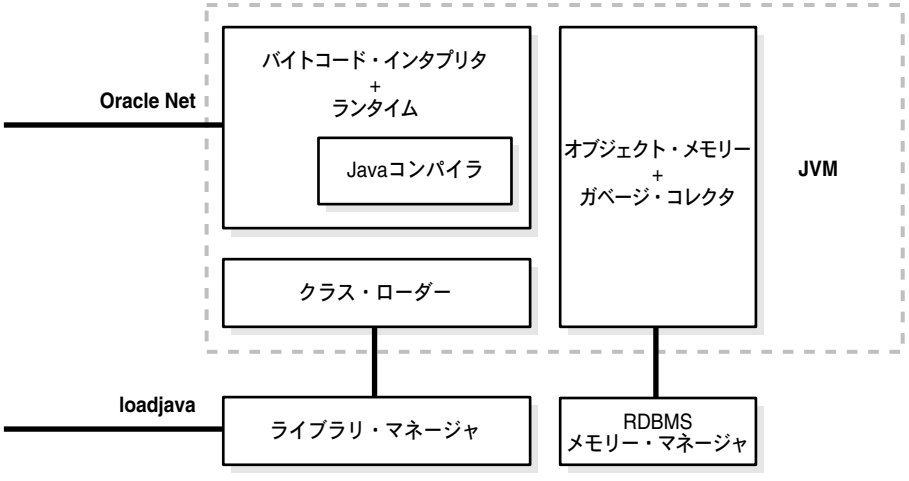
関連項目： SQLJ での SQL NCHAR データ型のサポートの詳細は、6-27 ページの「Unicode での Java 文字列のバインドと定義」を参照してください。

Java Virtual Machine のグローバリゼーション・サポート

Oracle9*i* の Java Virtual Machine（JVM）はデータベース・サーバーに統合され、データベースに格納された Java クラスの実行を可能にします。Oracle9*i* では、Java クラス・ファイル、Java または SQLJ のソース・ファイルおよび Java リソース・ファイルをデータベースに格納できます。これにより、SQL または PL/SQL から Java をコールできるように SQL への Java エントリ・ポイントを公開し、Java バイトコードを実行できます。

Java バイトコードを解析するエンジンの他に、Oracle JVM には Java Development Kit（JDK）のコア・ランタイム・クラスが含まれています。図 9-4 に、JVM のコンポーネントを示します。

図 9-4 Oracle の Java Virtual Machine のコンポーネント



JVM が提供する機能は、次のとおりです。

- データベースにローカルで格納された Java クラスの位置の特定、ロードおよび初期化を行う埋込み Java クラス・ローダー
- 標準 Java プログラムを標準 Java `.class` バイナリ表現に変換する Java コンパイラ

また、ライブラリ・マネージャも含まれており、Java プログラム、クラスおよびリソースのファイルを、**ライブラリ・ユニット**と呼ばれるスキーマ・オブジェクトとして管理します。このライブラリ・マネージャは、データベースの Java ファイルのロードおよび管理のみでなく、Java ネームスペースのライブラリ・ユニットへのマップも行います。次に例を示します。

```
public class Greeting
{
   public String Hello(String name)
   {
     return ("Hello" + name + "!");
   }
}
```

この Java コードは、コンパイル後、次のようにデータベースにロードされます。

```
loadjava Greeting.class
```

この結果、Greeting というライブラリ・ユニットが、スキーマ・オブジェクトとしてデータベースに作成されます。

データベース・キャラクタ・セットでは表現できない文字が含まれているクラス名とメソッド名は、US7ASCII のライブラリ・ユニット名を生成し、そのユニット名を RAW 列に格納されている実際のクラス名にマップすることで、処理されます。この処理によって、Java プログラムがサーバー上で実行されたときに、クラス・ローダーは実際のクラス名に対応するライブラリ・ユニットを検索できます。つまり、ライブラリ・マネージャとクラス・ローダーは、データベース・キャラクタ・セットのネームスペース外のクラス名とメソッド名をサポートします。

Java ストアド・プロシージャのグローバリゼーション・サポート

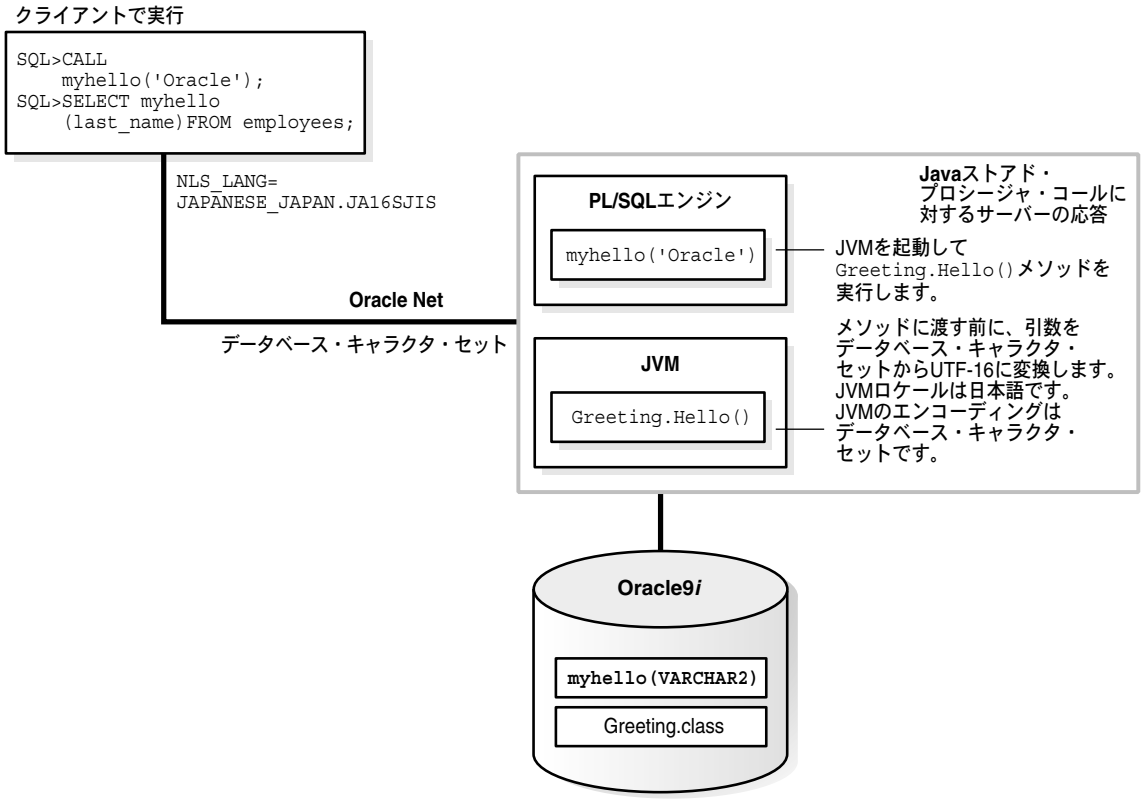
Java ストアド・プロシージャまたはストアド関数には、それを実装している Java クラスのライブラリ・ユニットが、あらかじめデータベースに存在していることが必要です。前の項の Greeting ライブラリ・ユニットの例を使用して、次のコール仕様データ定義言語（DDL）では、メソッド Greeting.Hello() を Java ストアド関数として公開しています。

```
CREATE FUNCTION myhello(name VARCHAR2) RETURN VARCHAR2
  AS LANGUAGE JAVA NAME
'Greeting.Hello(java.lang.String) return java.lang.String';
```

この DDL は、Java メソッド、パラメータ・タイプおよび戻り型を対応する SQL にマップします。ユーザーにとっては、Java ストアド関数のコール構文は、他の PL/SQL ストアド関数と同じです。ユーザーは、PL/SQL ストアド・プロシージャをコールする場合と同じ方法で、Java ストアド・プロシージャをコールできます。

図 9-5 に、ストアド関数のランタイム環境を示します。

図 9-5 Java ストアド・プロシージャの実行



JVM のロケールは日本語で、エンコーディングはデータベース・キャラクタ・セットです。クライアントの環境変数 NLS_LANG は、JAPANESE_JAPAN.JA16SJIS として定義されています。Oracle Net は、文字が異なる場合に、クライアント側の JA16SJIS 文字をデータベース・キャラクタ・セットの文字に変換します。

クライアントからプロキシ PL/SQL myhello() を起動すると、Java エントリ・ポイントの Greeting.Hello() がコールされます。クライアントの処理を行うサーバー・プロセスは、通常の PL/SQL ストアド関数として実行し、同じ構文を使用します。PL/SQL エンジンは、Java メソッドのコール仕様を認識すると、JVM をコールします。次に、実行する Java ストアド関数のメソッド名および引数を JVM に渡します。JVM は制御を受け取り、VARCHAR2 引数をデータベース・キャラクタ・セットから UTF-16 に変換するコードを使用して、SQL を Java にコールし、Greeting クラスをロードし、変換した引数で Hello() メソッドを実行します。Hello() から戻された文字列は、再度データベース・キャラクタ・セットに変換され、コール元に VARCHAR2 文字列として戻されます。

国際化された Java ストアド・プロシージャの配布および開発を可能にするグローバリゼーション・サポートには、次のものが含まれます。

- Java ストアド・プロシージャの引数の文字列は、データベース・キャラクタ・セットの SQL データ型から UTF-16 でエンコードされた Java 文字列に、自動的に変換されます。
- JVM のデフォルトの Java ロケールは、クライアントの環境変数 NLS_LANG から導出された現行のデータベース・セッションの言語設定に従います。このため、Oracle の言語名と地域名は、Java ロケール名にマッピングされています。また、JVM のデフォルトのエンコーディングは、データベース・キャラクタ・セットに従います。
- loadjava ユーティリティは、JDK がサポートするエンコーディングによってエンコードされた Java および SQLJ ソース・ファイルのロードをサポートします。Java または SQLJ プログラムの内容は、データベース・キャラクタ・セットの制約を受けません。また、プログラム・ファイルの Unicode エスケープ・シーケンスもサポートされます。

注意： Java ストアド・プロシージャのエントリ・メソッド名とクラス名は、DDL として SQL に公開する必要があるため、データベース・キャラクタ・セットで指定してください。

多言語アプリケーションの構成

Oracle9*i* の多言語 Java アプリケーションを開発および配布するために、あらかじめターゲット・システム用のデータベース構成とクライアント環境を決定しておく必要があります。

この項の内容は、次のとおりです。

多言語データベースの構成

多言語データを Oracle9*i* データベースに格納するには、データベースを適切に構成する必要があります。Unicode データをデータベースに格納するには、次の 2 つの方法があります。

- Unicode データベースに SQL CHAR データ型として格納する方法
- Unicode 以外のデータベースに SQL NCHAR データ型として格納する方法

関連項目： Unicode ソリューションの選択および Unicode に対するデータベースの構成の詳細は、第 5 章「Unicode を使用した多言語データベースのサポート」を参照してください。

Java ストアド・プロシージャのグローバリゼーション・サポート

Oracle9*i* セッションごとに、Java ストアド・プロシージャを実行するために JVM インスタンスがサーバー上に個別に作成され、Oracle9*i* Java サポートによって、JVM インスタンスのロケールがクライアント JVM のロケールと同じであることが保証されます。したがって、Java ストアド・プロシージャは常に、クライアント・ロケールと同じデータベースのロケールで実行されます。

Java 以外のクライアントでは、JVM インスタンスのデフォルトのロケールは、クライアントの環境変数 NLS_LANG を引き継ぐセッション・パラメータ NLS_LANGUAGE および NLS_TERRITORY に対応した最適な Java ロケールになります。

Java コードの国際化

Java ストアド・プロシージャは、言語環境の異なるクライアントからアクセスできるサーバー・オブジェクトです。そのため、JVM の Java ロケール（クライアントのロケールに初期化されている）を区別できるように国際化される必要があります。

JDK 国際化サポートを使用すると、Java ロケール・オブジェクトを、ロケールを区別する任意のメソッドに指定するか、これらのメソッドに対して JVM のデフォルト Java ロケールを使用できます。次に、Java ストアド・プロシージャを国際化する方法の例を示します。

- ローカライズできる文字列またはオブジェクトはすべて、Java コードからリソース・バンドルに外部化し、このリソース・バンドルをプロシージャの一部とします。リソース・バンドルから戻されたすべてのメッセージは、クライアント・ロケールまたはユーザーが指定したロケールの言語になります。
- 日付、時刻、数字および通貨を書式設定するための DateFormat や NumberFormat などの Java 書式設定クラスを、これらのクラスがコール元のクライアントのロケールを反映していると仮定して使用します。
- Character、Collator および BreakIterator などの Java のロケールを区別する文字列クラスを使用して文字の分類をチェックし、2 つの文字列を言語的に比較し、文字列を文字ごとに解析します。

多言語データの転送

すべての Java サーバー・オブジェクトは、サーバー側 JDBC 内部ドライバを使用してデータベースにアクセスし、Java 文字列または oracle.sql.CHAR を使用してデータベースとやり取りする文字列データを表現する必要があります。Java 文字列は常に UTF-16 でエンコードされ、データベース・キャラクタ・セットから UTF-16 への変換は透過的に行われます。oracle.sql.CHAR は、データベース・データをバイト配列に格納し、キャラクタ・セット ID でタグを付けます。データに文字列操作が必要でない場合に、oracle.sql.CHAR を使用する必要があります。たとえば、データベース内のある表から別の表に文字列データを転送する場合などに最も適しています。

言語が異なるクライアント

クライアント（または中間層）は、異なる言語環境、データベース・アクセス・メカニズムおよび Java Runtime Environment を伴う可能性があります。次に、一般的に使用されるクライアントの構成をいくつか示します。

- ブラウザ上で実行する Java アプレット

  ブラウザ上で実行する Java アプレットは、JDBC Thin ドライバを経由して Oracle9*i* データベースにアクセスできます。クライアント側の Oracle ライブラリは不要です。アプレットは JDBC Thin ドライバを使用して、Java ストアド・プロシージャの他に、SQL、PL/SQL も起動します。JDBC Thin ドライバは、アプレットを実行する JVM と同じロケールで Java ストアド・プロシージャが実行されることを保証します。

- ブラウザ上の動的 HTML

  HTML ページは、HTTP 上の URL を使用して Java Servlet を起動します。中間層で実行される Java Servlet は、動的 HTML ページを作成し、ブラウザに戻します。Java Servlet は、ユーザーのロケールを判断し、ユーザーの言語と文化的な慣習の環境に従ってページを作成し、JDBC を使用してデータベースに接続する必要があります。

- クライアント JVM で実行する Java アプリケーション

  クライアント・マシンの JVM で実行する Java アプリケーションは、JDBC OCI ドライバまたは JDBC Thin ドライバのいずれかを経由してデータベースにアクセスできます。Java アプリケーションは、Web サーバーで実行する中間層サーブレットにすることもできます。このアプリケーションは JDBC ドライバを使用して、Java ストアド・プロシージャの他に、SQL、PL/SQL も起動します。JDBC Thin ドライバと JDBC OCI ドライバは、クライアント JVM と同じロケールで Java ストアド・プロシージャを実行することを保証します。

- OCI、Pro*C/C++ および ODBC などの C クライアント

  Java 以外のクライアントは、PL/SQL ストアド・プロシージャをコールする場合と同じ方法で、Java ストアド・プロシージャをコールできます。JVM ロケールは、クライアントの環境変数 `NLS_LANG` を伝播した Oracle の言語設定 `NLS_LANGUAGE` と `NLS_TERRITORY` に最も適応したロケールです。その結果、クライアントは常に `NLS_LANG` に指定された言語で、サーバーからのメッセージを受け取ります。クライアントのデータは、OCI によってデータベース・キャラクタ・セットとの間で変換されます。

SQLJ の多言語デモ・アプリケーション

この項では、SQLJ で記述された簡単な書店向けアプリケーションを使用した例で、言語の異なる書籍情報を格納するデータベースと、そのデータベースの書籍情報に SQLJ および JDBC を使用してアクセスする方法を示します。また、国際化 Java ストアド・プロシージャを使用して、データベース・サーバーでトランザクション作業を実行する例を示します。サンプル・プログラムは、次のコンポーネントで構成されています。

- 店で扱う書籍のリストを表示し、ユーザーが在庫に対して書籍の追加と削除を行うための SQLJ クライアント Java アプリケーション
- 新しい書籍を在庫に追加する Java ストアド・プロシージャ
- 既存の書籍を在庫から削除する Java ストアド・プロシージャ

この項の内容は、次のとおりです。

多言語デモ・アプリケーション用のデータベース・スキーマ

AL32UTF8 は、書名や著者名などの書籍情報を世界中の言語で格納するために使用されるデータベース・キャラクタ・セットです。

表 9-4 に、book 表の定義を示します。

表 9-4 多言語デモの book 表の列

列名	データ型
ID（主キー）	NUMBER(10)
NAME	VARCHAR(300)
PUBLISH_DATE	DATE
AUTHOR	VARCHAR(120)
PRICES	NUMBER(10,2)

表 9-5 に、inventory 表の定義を示します。

表 9-5 多言語デモの invertory 表の列

列名	データ型
ID（主キー）	NUMBER(10)
LOCATION（主キー）	VARCHAR(90)
QUANTITY	NUMBER(3)

この他に、book 表の NAME および AUTHOR 列には、書籍検索中のパフォーマンスを改善するための索引が作成されています。一意の書籍 ID を生成するために、順序 BOOKSEQ が作成されます。

多言語デモ・アプリケーション用の Java ストアド・プロシージャ

在庫に対して書籍を削除または追加するタスクを実行するメソッド Book.remove() および Book.add() を実装するために、Book という Java クラスが作成されます。これらのメソッドは、後述のコードでそれぞれ定義されています。このクラスでは、remove() メソッドとコンストラクタのみが示されます。ローカライズできるメッセージを格納するために、リソース・バンドル BookRes.class が使用されます。remove() メソッドは、現行の JVM ロケールに従って、リソース・バンドルから取得したメッセージを戻します。このストアド・プロシージャは、すでにデータベース・セッションのコンテキストで実行中のため、データベースにアクセスするための JDBC 接続は不要です。

```
import java.sql.*;
import java.util.*;
import sqlj.runtime.ref.DefaultContext;
/* The book class implementation the transaction logics of the
   Java stored procedures.*/
public class Book
{
   static ResourceBundle rb;
   static int q, id;
   static DefaultContext ctx;
   public Book()
   {
     try
     {
         DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
         DefaultContext.setDefaultContext(ctx);
         rb = java.util.ResourceBundle.getBundle("BookRes");
      }
      catch (Exception e)
      {
```

```
      System.out.println("Transaction failed: " + e.getMessage());
    }
 }
public static String remove(int id, int quantity, String location) throws
  SQLException
{
    rb = ResourceBundle.getBundle("BookRes");
    try
    {
      #sql {SELECT QUANTITY INTO :q FROM INVENTORY WHERE ID = :id AND
      LOCATION = :location};
      if (id == 1) return rb.getString ("NotEnough");
     }
    catch (Exception e)
    {
        return rb.getString ("NotEnough");
    }
    if ((q - quantity) == 0)
    {
        #sql {DELETE FROM INVENTORY WHERE ID = :id AND LOCATION = :location};
        try
        {
            #sql {SELECT SUM(QUANTITY) INTO :q FROM INVENTORY WHERE ID = :id};
        }
        catch (Exception e)
        {
            #sql { DELETE FROM BOOK WHERE ID = :id };
            return rb.getString("RemoveBook");
        }
        return rb.getString("RemoveInventory");
    }
    else
    {
       if ((q-quantity) < 0) return rb.getString ("NotEnough");
       #sql { UPDATE INVENTORY SET QUANTITY = :(q-quantity) WHERE ID = :id and
       LOCATION = :location };
       return rb.getString("DecreaseInventory");
    }
}
public static String add( String bname, String author, String location,
       double price, int quantity, String publishdate )  throws SQLException
{
   rb = ResourceBundle.getBundle("BookRes");
   try
   {
       #sql { SELECT ID into :id FROM BOOK WHERE NAME = :bname AND AUTHOR =
      :author };
```

```
    }
    catch (Exception e)
    {
        #sql { SELECT BOOKSEQ.NEXTVAL INTO :id FROM DUAL };
        #sql { INSERT INTO BOOK VALUES (:id, :bname,
        TO_DATE(:publishdate,'YYYY-MM-DD'), :author, :price) };
        #sql { INSERT INTO INVENTORY VALUES (:id, :location, :quantity) };
        return rb.getString("AddBook");
    }
    try
    {
        #sql { SELECT QUANTITY INTO :q FROM INVENTORY WHERE ID = :id
        AND LOCATION = :location };
    }
    catch (Exception e)
    {
         #sql { INSERT INTO INVENTORY VALUES (:id, :location, :quantity) };
         return rb.getString("AddInventory");
    }
    #sql { UPDATE INVENTORY SET QUANTITY = :(q + quantity) WHERE ID = :id
    AND LOCATION = :location };
    return rb.getString("IncreaseInventory");
  }
}
```

定義されたメソッド Book.remove() および Book.add() は、次のように、データベース内の Java ストアド関数 REMOVEBOOK() および ADDBOOK() として公開されます。

```
CREATE FUNCTION REMOVEBOOK (ID NUMBER, QUANTITY NUMBER,
    LOCATION VARCHAR2)
    RETURN VARCHAR2
    AS LANGUAGE JAVA NAME
    'Book.remove(int, int, java.lang.String)  return java.lang.String';

CREATE FUNCTION ADDBOOK (NAME VARCHAR2, AUTHOR VARCHAR2,
   LOCATION VARCHAR2, PRICE NUMBER, QUANTITY NUMBER, PUBLISH_DATE DATE)
   RETURN VARCHAR2
   AS LANGUAGE JAVA NAME
   'Book.add(java.lang.String, java.lang.String, java.lang.String,
   double, int, java.sql.Date) return java.lang.String';
```

戻された Java 文字列は、データベース・キャラクタ・セットでエンコードされた VARCHAR2 文字列に変換されてから、クライアントに戻されます。データベース・キャラクタ・セットが AL32UTF8 または UTF8 のいずれでもない場合、データベース・キャラクタ・セットで表現できない Java 文字列の中の Unicode 文字は、置換文字で置換されます。同様に、データベース・キャラクタ・セットでエンコードされている VARCHAR2 文字列は、Java 文字列に変換されてから、Java メソッドに渡されます。

多言語デモ・アプリケーション用の SQLJ クライアント

SQLJ クライアントは、JDBC Thin ドライバまたは JDBC OCI ドライバのいずれかを使用する GUI Java アプリケーションです。クライアントからデータベースに接続して、指定された検索基準による書籍のリストの表示、選択された書籍の在庫からの削除、および新しい書籍の在庫への追加を行います。これらのタスクを実行するために、クラス BookDB が作成されます。これは、後述のコードに定義されています。

BookDB オブジェクトは、ユーザー名、パスワードおよびデータベースの位置を指定してサンプル・プログラムを起動したときに作成されます。メソッドはアプリケーションの GUI 部分からコールされます。removeBook() および addBook() メソッドは、データベース上の対応する Java ストアド関数をコールし、トランザクションのステータスを戻します。メソッド searchByName() および searchByAuthor() は、それぞれ書名および著者の指定に従って書籍を検索し、結果を BookDB オブジェクトの中の books イテレータに格納します（SQLJ によって、BookRecs クラスが生成されます）。次に、GUI コードは、getNextBook() 関数をコールして、null が返るまでイテレータ・オブジェクトから書籍のリストを取り出します。getNextBook() 関数は、単純にイテレータから次の行をフェッチします。

```
package sqlj.bookstore;

import java.sql.*;
import sqlj.bookstore.BookDescription;
import sqlj.runtime.ref.DefaultContext;
import java.util.Locale;
/*The iterator used for a book description when communicating with the server*/
#sql iterator BooksRecs( int ID, String NAME, String AUTHOR, Date PUBLISH_DATE,
                         String LOCATION, int QUANTITY, double PRICE);
/*This is the class used for connection to the server.*/
class BookDB
{
    static public final String DRIVER = "oracle.jdbc.driver.OracleDriver";
    static public final String URL_PREFIX = "jdbc:oracle:thin:@";
    private DefaultContext m_ctx = null;
    private String msg;
    private BooksRecs books;
    /*Constructor - registers the driver*/
    BookDb()
    {
        try
        {
            DriverManager.registerDriver
                ((Driver) (Class.forName(DRIVER).newInstance()));
        }
        catch (Exception e)
        {
            System.exit(1);
        }
```

```
}
/*Connect to the database.*/
DefaultContext connect(String id, String pwd, String userUrl) throws
SQLException
{
    String url = new String(URL_PREFIX);
    url = url.concat(userUrl);
    Connection conn = null;
    if (m_ctx != null) return m_ctx;
    try
    {
        conn = DriverManager.getConnection(url, id, pwd);
    }
    catch (SQLException e)
    {
      throw(e);
    }
    if (m_ctx == null)
    {
        try
        {
            m_ctx = new DefaultContext(conn);
        }
        catch (SQLException e)
        {
           throw(e);
        }
    }
    return m_ctx;
}
/*Add a new book to the database.*/
public String addBook(BookDescription book)
{
    String  name = book.getTitle();
    String  author = book.getAuthor();
    String  date = book.getPublishDateString();
    String  location = book.getLocation();
    int     quantity = book.getQuantity();
    double  price = book.getPrice();
    try
    {
      #sql [m_ctx] msg = {VALUE ( ADDBOOK ( :name, :author, :location,
      :price, :quantity, :date))};
      #sql [m_ctx] {COMMIT};
    }
    catch (SQLException e)
    {
```

```
        return (e.getMessage());
    }
    return msg;
}
/*Remove a book.*/
public String removeBook(int id, int quantity, String location)
{
    try
    {
      #sql [m_ctx] msg = {VALUE ( REMOVEBOOK ( :id, :quantity,
      :location))};
      #sql [m_ctx] {COMMIT};
    }
    catch (SQLException e)
    {
      return (e.getMessage());
    }
    return msg;
}
/*Search books by the given author.*/
public void searchByAuthor(String author)
{
    String key = "%" + author + "%";
    books = null;
    System.gc();
    try
    {
      #sql [m_ctx] books = { SELECT BOOK.ID, NAME, AUTHOR, PUBLISH_DATE,
      LOCATION, QUANTITY, PRICE
      FROM BOOK, INVENTORY WHERE BOOK.ID = INVENTORY.ID AND AUTHOR LIKE
      :key ORDER BY BOOK.ID};
    }
    catch (SQLException e) {}
}
/*Search books with the given title.*/
public void searchByTitle(String title)
{
    String key = "%" + title + "%";
    books = null;
    System.gc();
    try
    {
      #sql [m_ctx] books = { SELECT BOOK.ID, NAME, AUTHOR, PUBLISH_DATE,
      LOCATION, QUANTITY, PRICE
      FROM BOOK, INVENTORY WHERE BOOK.ID = INVENTORY.ID AND NAME LIKE
      :key ORDER BY BOOK.ID};
    }
```

```
        catch (SQLException e) {}
    }
    /*Returns the next BookDescription from the last search, null if at the
      end of the result list.*/
    public BookDescription getNextBook()
    {
       BookDescription book = null;
       try
       {
         if (books.next())
         {
            book = new BookDescription(books.ID(), books.AUTHOR(), books.NAME(),
                   books.PUBLISH_DATE(), books.PRICE(),
                          books.LOCATION(), books.QUANTITY());
         }
        }
        catch (SQLException e) {}
        return book;
    }
}
```

10

キャラクタ・セットの移行

この章では、キャラクタ・セット変換とキャラクタ・セットの移行について説明します。この章の内容は、次のとおりです。

- キャラクタ・セットの移行の概要
- 既存のデータベースのデータベース・キャラクタ・セットの変更
- Oracle9i の NCHAR データ型の使用への移行
- キャラクタ・セット移行後にデータベース・スキーマをリカバリするタスク

キャラクタ・セットの移行の概要

使用しているデータベースに適切なキャラクタ・セットを選択することは、重要な決定事項です。データベース・キャラクタ・セットの選択時には、次の要因を考慮する必要があります。

- 格納する必要があるデータの型
- データベースで現在および将来対応する必要のある言語
- 各キャラクタ・セットの異なるサイズ要件とその要件がパフォーマンスに与える影響

関連項目として、既存のデータベース用の新規キャラクタ・セットの選択があります。既存のデータベース用のデータベース・キャラクタ・セットを変更することを、**キャラクタ・セットの移行**と呼びます。あるデータベース・キャラクタ・セットから別のデータベース・キャラクタ・セットに移行するには、新規データベースのキャラクタ・セットを選択する場合よりも多くのことを考慮する必要があります。次の原因によるデータ消失を最小限に抑えるように、キャラクタ・セットの移行プランを作成してください。

- データの切捨て
- キャラクタ・セット変換の問題

関連項目： 第 2 章「キャラクタ・セットの選択」

データの切捨て

データベースがバイト・セマンティクスを使用して作成されている場合、CHAR および VARCHAR2 データ型のサイズは、文字単位ではなくバイト単位で指定されます。たとえば、表定義で CHAR(20) と指定すると、文字データを格納するために 20 バイトが割り当てられます。データベース・キャラクタ・セットでシングルバイト文字コード体系が使用されている場合は、文字数とバイト数が同じであるため、このような指定は適切に処理されます。データベース・キャラクタ・セットでマルチバイト文字コード体系が使用されている場合は、1 文字が 1 バイト以上で構成される場合があるため、バイト数は文字数とは異なります。

新規キャラクタ・セットへの移行時には、既存の CHAR および VARCHAR2 列の列幅を検証することが重要です。これは、マルチバイトを格納する必要があるエンコーディングをサポートするために、列幅の拡張が必要となる場合があるためです。変換によってデータが拡張されると、データが切り捨てられる可能性があります。

図 10-1 に、シングルバイト文字がマルチバイトになる場合のデータ拡張例を示します。たとえば、ä（ウムラウト付きの a）は、WE8MSWIN1252 ではシングルバイト文字ですが、UTF8 では 2 バイト文字になります。また、ユーロ記号は 1 バイトから 3 バイトに拡張されます。

図 10-1 シングルバイトとマルチバイトのエンコーディング

文字	WE8MSWIN1252	UTF8
ä	E4	C3 A4
ö	F6	C3 B6
©	A9	C2 A9
€	80	E2 82 AC

CHAR および VARCHAR2 データ型の最大バイト数は、それぞれ 2000 バイトおよび 4000 バイトです。新規キャラクタ・セットのデータに、CHAR および VARCHAR2 データ型について 2000 および 4000 バイトを超える列幅が必要な場合は、スキーマを変更する必要があります。

関連項目： 2-11 ページ「長さセマンティクス」

データの切捨てによって生じるその他の問題

データの切捨てによって次の問題が発生する可能性があります。

- データベース・データ・ディクショナリ内でのスキーマ・オブジェクト名は、30 バイトを超えることはできません。スキーマ・オブジェクトは、表、クラスタ、ビュー、索引、シノニム、表領域およびユーザー名です。スキーマ・オブジェクト名が新規データベース・キャラクタ・セットで 30 バイトを超える場合は、そのオブジェクト名を改名する必要があります。たとえば、タイ語の各国語キャラクタ・セットではタイ語 1 文字に 1 バイト必要ですが、UTF8 では 3 バイト必要です。タイ語 11 文字の名前を持つ表を定義している場合、データベース・キャラクタ・セットを UTF8 に変更するときは、この表名を 10 文字以下に短縮する必要があります。
- 既存の Oracle ユーザー名またはパスワードが文字ベースで作成されており、その内容が変換先のキャラクタ・セットでサイズ変更された場合、新規キャラクタ・セットへの移行後は認証がエラーとなるため、ユーザーのログインが困難になります。これは、データ・ディクショナリに格納されている暗号化されたユーザー名とパスワードが、新規キャラクタ・セットへの移行時に更新されないためです。たとえば、現行のデータベース・キャラクタ・セットが WE8MSWIN1252 で、変換先のデータベース・キャラクタ・セットが UTF8 の場合、ユーザー名 scött（ウムラウト付きの o）の長さは 5 バイトから 6 バイトに変更されます。UTF8 では、ユーザー名が異なるため、scött はログインできなくなります。ユーザー名とパスワードには ASCII 文字を使用することをお薦めします。ASCII 文字以外の場合は、新規キャラクタ・セットへの移行後に、影響のあるユーザー名とパスワードを再設定する必要があります。

- CHAR データに、新規キャラクタ・セットへの移行後に拡張される文字が含まれている場合、デフォルトでは、データベースのエクスポート時に空白埋込みは削除されません。つまり、これらの行は、新規キャラクタ・セットでデータベースにインポートされるときに拒否されます。これを回避するには、CHAR データをインポートする前に BLANK_TRIMMING 初期化パラメータを TRUE に設定します。

  関連項目： BLANK_TRIMMING 初期化パラメータの詳細は、『Oracle9*i* データベース・リファレンス』を参照してください。

キャラクタ・セット変換の問題

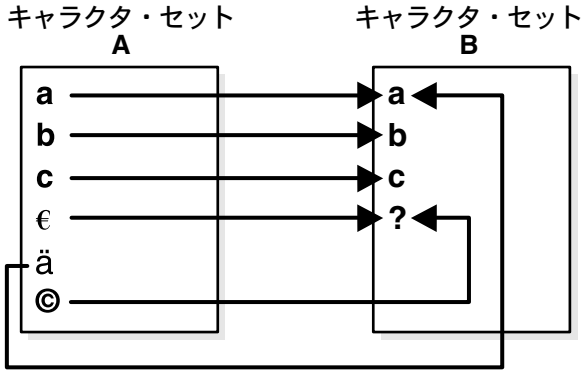
この項の内容は、次のとおりです。

エクスポート・ユーティリティとインポート・ユーティリティの使用による置換文字

エクスポート・ユーティリティとインポート・ユーティリティでは、オリジナルのデータベース・キャラクタ・セットから新規データベース・キャラクタ・セットに変換できます。ただし、キャラクタ・セット変換によって、データが消失したり、破損する可能性があります。たとえば、キャラクタ・セット A からキャラクタ・セット B に移行する場合に、変換先キャラクタ・セット B はキャラクタ・セット A のスーパーセットであることが必要です。変換先キャラクタ・セット B が**スーパーセット**であるのは、キャラクタ・セット A に定義されているすべての文字が含まれている場合です。キャラクタ・セット B で使用できない文字は、置換文字に変換されます。この置換文字は、通常、?、¿ または使用不可能な文字に関連する 1 文字で指定されます。たとえば、ä（ウムラウト付きの a）は、a で置換できます。置換文字は、変換先キャラクタ・セットで定義されます。

図 10-2 に、著作権記号とユーロ記号が ? に変換され、ä が a に変換されるキャラクタ・セット変換の例を示します。

図 10-2 キャラクタ・セット変換での置換文字



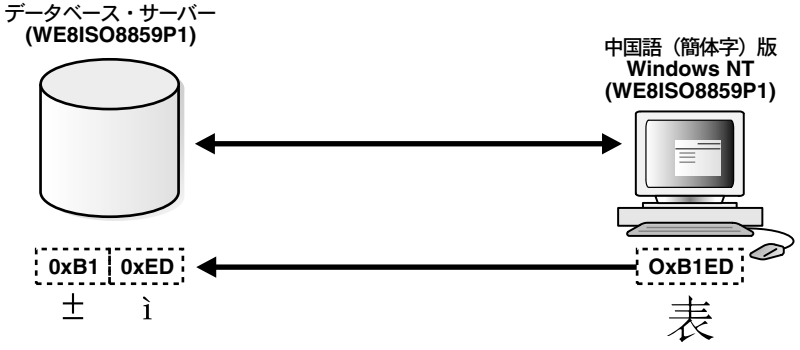
データ消失のリスクを削減するには、類似する文字レパートリを持つキャラクタ・セットを変換先に選択してください。UTF8 には既存のキャラクタ・セットのほとんどの文字が含まれているため、Unicode への移行は考慮に値する選択肢です。

クライアントの NLS_LANG パラメータ設定が不適切であるために生じる無効なデータ

データ消失の可能性があるキャラクタ・セット移行の使用例として、無効なデータを含むデータベースの移行が考えられます。通常は、NLS_LANG パラメータがクライアント上で適切に設定されていないために、データベースに無効なデータが発生します。NLS_LANG 値には、クライアント・オペレーティング・システムのコード・ページを反映させる必要があります。たとえば、英語版 Windows 環境では、コード・ページは WE8MSWIN1252 です。NLS_LANG パラメータを適切に設定すると、データベースはクライアント・オペレーティング・システムからのデータを自動的に変換できます。NLS_LANG パラメータが適切に設定されていないと、データベースに入ってくるデータは適切に変換されません。たとえば、データベース・キャラクタ・セットが UTF8 で、クライアントが英語版 Windows オペレーティング・システムで、クライアント上での NLS_LANG 設定が UTF8 であるとします。データベースに入ってくるデータは WE8MSWIN1252 でエンコードされており、クライアント上の NLS_LANG 設定がデータベース・キャラクタ・セットと一致するため、UTF8 データには変換されません。そのため、Oracle では変換は不要であるとみなされ、データベースに無効なデータが入力されます。

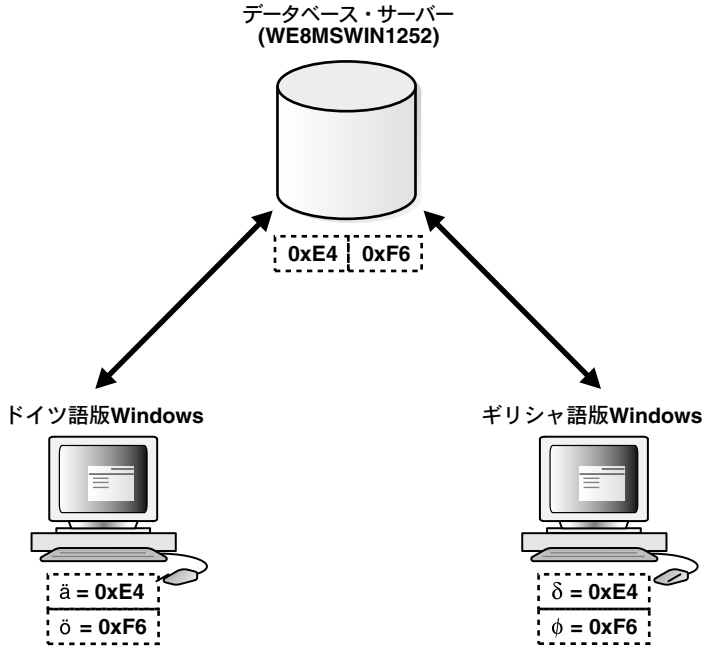
この結果、データの非一貫性の問題が 2 つ発生する可能性があります。一方の問題が発生するのは、データベースに含まれているデータのキャラクタ・セットがデータベース・キャラクタ・セットとは異なっているが、両方のキャラクタ・セットに同じコード・ポイントが存在する場合です。たとえば、データベース・キャラクタ・セットが WE8ISO8859P1 で、中国語版 Windows NT クライアントの NLS_LANG 設定が SIMPLIFIED CHINESE_CHINA.WE8ISO8859P1 の場合、（ZHS16GBK キャラクタ・セットの）すべてのマルチバイトの中国語データは、シングルバイト WE8ISO8859P1 データのペアとして格納されます。これは、Oracle ではこれらの文字がシングルバイト WE8ISO8859P1 文字として扱われることを意味します。そのため、SUBSTR や LENGTH などの SQL 文字列操作関数はすべて、文字ではなくバイトを基準にして処理されます。ZHS16GBK データを構成するすべてのバイトは、有効な WE8ISO8859P1 コードです。このようなデータベースを UTF8 などの別のキャラクタ・セットに移行する場合、文字コードは、WE8ISO8859P1 内の文字コードと同様に変換されます。このように、ZHS16GBK 文字の 2 バイトはそれぞれ個別に変換され、UTF8 では意味のない値になります。図 10-3 は、この不適切なキャラクタ・セット置換の例です。

図 10-3 不適切なキャラクタ・セットの置換



2 つ目の問題は、混在するキャラクタ・セットのデータがデータベースにある場合です。たとえば、データ・キャラクタ・セットが WE8MSWIN1252 で、ドイツ語とギリシャ語を使用している 2 つの別々の Windows クライアントがともに、WE8MSWIN1252 に設定した NLS_LANG キャラクタ・セットを使用している場合、データベースにはドイツ語とギリシャ語の文字が混在することになります。図 10-4 は、異なるクライアントが、同一データベースの異なるキャラクタ・セットを使用する方法を示しています。

図 10-4 キャラクタ・セットの混在



データベース・キャラクタ・セットの移行を適切に行うには、この 2 つの状況の場合、手作業による調整が必要です。これは、Oracle では格納されているデータのキャラクタ・セットを判断できないためです。

既存のデータベースのデータベース・キャラクタ・セットの変更

データベース・キャラクタ・セットの移行には、データ・スキャニングとデータ変換という2つの段階があります。データベース・キャラクタ・セットを変更する前に、考えられるデータベース・キャラクタ・セット変換の問題とデータの切捨てを識別する必要があります。このステップは**データ・スキャニング**と呼ばれます。

データ・スキャニングによって、データベース・キャラクタ・セットの変更前に、データを新しい文字コード体系に移行するために必要な作業量が識別されます。データ・スキャニング時に検出される例には、列幅の拡張が必要なスキーマ・オブジェクトの数や、変換先の文字レパートリに存在しないデータの分布などがあります。この情報は、データベース・キャラクタ・セットの最適な変換方法の決定に役立ちます。

データベース・キャラクタ・セットから別のデータベース・キャラクタ・セットにデータを変換するには、3通りの方法があります。ただし、そのデータベースには、10-4ページの「キャラクタ・セット変換の問題」で説明した非一貫性がないことが前提です。このような非一貫性があるデータベースを移行する方法は、ここでは説明しません。詳細は、オラクル社カスタマ・サポート・センターにお問い合せください。

この項の内容は、次のとおりです。

> **関連項目：** データ・スキャニングの詳細は、第11章「Character Set Scanner」を参照してください。

全体エクスポートおよび全体インポートを使用した文字データの移行

多くの場合、すべてのデータを新規キャラクタ・セットに正しく変換するには、全体エクスポートおよび全体インポートをお薦めします。文字データ型の列は、サイズ増加を処理するためにインポート前に拡張が必要となる場合があるため、データ切捨ての問題に注意することが重要です。既存のPL/SQLコードをレビューして、`LENGTHB`、`SUBSTRB`および`INSTRB`などのバイトベースのすべてのSQL関数、またはPL/SQLの`CHAR`および`VARCHAR2`の宣言が有効であることを確認する必要があります。

> **関連項目：** エクスポート・ユーティリティとインポート・ユーティリティの詳細は、『Oracle9*i* データベース・ユーティリティ』を参照してください。

ALTER DATABASE CHARACTER SET 文を使用した文字データの移行

ALTER DATABASE CHARACTER SET 文は、キャラクタ・セットの移行方法としては最も高速ですが、使用できるのは特定の場合のみです。ALTER DATABASE CHARACTER SET 文ではデータ変換は実行されないため、この文を使用できるのは、新規キャラクタ・セットが現行のキャラクタ・セットの完全なスーパーセットである場合のみです。

新規キャラクタ・セットが現行のキャラクタ・セットの完全なスーパーセットであるのは、次の場合です。

- 現行のキャラクタ・セット内のすべての文字が、新規キャラクタ・セットで使用可能な場合。
- 現行のキャラクタ・セット内のすべての文字が、新規キャラクタ・セット内で同じコード・ポイント値を持つ場合。たとえば、US7ASCII は、多数のキャラクタ・セットの完全なサブセットです。

また、ALTER DATABASE CHARACTER SET 文を使用できるのは、2 つのシングルバイト・キャラクタ・セット間、または 2 つのマルチバイト・キャラクタ・セット間で移行する場合のみであるという制限もあります。シングルバイトからマルチバイトへのキャラクタ・セットの移行を計画している場合は、エクスポート・ユーティリティとインポート・ユーティリティを使用してください。

この ALTER DATABASE CHARACTER SET 文の使用制限は、CLOB データによるものです。Oracle9*i* では、データ・ディクショナリの一部の内部フィールドが CLOB 列に格納されます。また、ユーザーがデータを CLOB フィールドに格納している場合もあります。データベース・キャラクタ・セットがマルチバイトの場合、Oracle9*i* の CLOB データは UCS-2 データ（2 バイトの固定幅 Unicode）として格納されます。データベース・キャラクタ・セットがシングルバイトの場合、CLOB データはデータベース・キャラクタ・セットを使用して格納されます。ALTER DATABASE CHARACTER SET 文ではデータは変換されないため、データベース・キャラクタ・セットをシングルバイトからマルチバイトに移行すると、CLOB 列は元のデータベース・キャラクタ・セットのエンコーディングのままとなります。これによって、CLOB 列でのデータの非一貫性が生じます。

ALTER DATABASE CHARACTER SET 文の構文は、次のとおりです。

```
ALTER DATABASE [db_name] CHARACTER SET new_character_set;
```

db_name はオプションです。キャラクタ・セット名は、引用符を付けずに指定してください。たとえば、次のように指定します。

```
ALTER DATABASE CHARACTER SET AL32UTF8;
```

データベース・キャラクタ・セットを変更する手順は、次のとおりです。

1. SHUTDOWN IMMEDIATE 文または SHUTDOWN NORMAL 文のいずれかを使用して、データベースをシャットダウンします。
2. ALTER DATABASE CHARACTER SET 文はロールバックできないため、データベースの全体バックアップを実行します。

3. 次の文を完了します。

```
STARTUP MOUNT;
ALTER SYSTEM ENABLE RESTRICTED SESSION;
ALTER SYSTEM SET JOB_QUEUE_PROCESSES=0;
ALTER SYSTEM SET AQ_TM_PROCESSES=0;
ALTER DATABASE OPEN;
ALTER DATABASE CHARACTER SET new_character_set;
SHUTDOWN IMMEDIATE; -- or SHUTDOWN NORMAL;
STARTUP;
```

関連項目：

- ALTER DATABASE CHARACTER SET 文の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。
- 全スーパーセット・キャラクタ・セットのリストは、付録 A「ロケール・データ」を参照してください。

Oracle9*i* Real Application Clusters 環境での ALTER DATABASE CHARACTER SET 文の使用

Oracle9*i* Real Application Clusters 環境では、ALTER DATABASE CHARACTER SET 文を発行する前に、ユーザーが接続に使用しているインスタンスに関連付けられたバックグラウンド・プロセス以外に、Oracle バックグラウンド・プロセスが実行されていないことを確認してください。次の SQL 文を使用して環境を検証してください。

```
SELECT SID, SERIAL#, PROGRAM FROM V$SESSION;
```

キャラクタ・セットの変更を完了できるように、CLUSTER_DATABASE 初期化パラメータを FALSE に設定します。Oracle9*i* Real Application Cluster 環境では、排他的な起動が不十分であるため、この設定が必要です。

ALTER DATABASE CHARACTER SET 文と選択式のインポートを使用した文字データの移行

もう 1 つの文字データ移行方法は、ALTER DATABASE CHARACTER SET 文を実行した後に選択式のインポートを指定することです。この方法は、少数の表に格納されている変換可能なデータの分散がわかっている場合に最適です。この例にある全体エクスポートや全体インポートは、非常にコストがかかります。たとえば、100GB のデータベースに 300 以上の表があり、そのうちキャラクタ・セット変換を必要とするのは 3 つの表のみで、それ以外のデータは変換先キャラクタ・セットと同じエンコーディングであるとします。ALTER DATABASE CHARACTER SET 文の発行後、この 3 つの表をエクスポートし、新規データベースにインポートして、戻すことができます。

不適切なデータ変換はデータを破損する恐れがあるため、新規キャラクタ・セットにデータを移行する前に、データベース全体のバックアップを作成します。

Oracle9*i* の NCHAR データ型の使用への移行

Oracle9*i* では、NCHAR データ型の列に格納されるデータには、データベース・キャラクタ・セットに関係なく Unicode エンコーディングのみが使用されます。このため、データベース・キャラクタ・セットとして Unicode を使用しないデータベースに、Unicode を格納できます。

この項の内容は、次のとおりです。

Oracle8 の NCHAR 列の Oracle9*i* への移行

リリース 8.0 の Oracle サーバーでは、各国語キャラクタ・データ型（NCHAR）を導入しました。このデータ型によって、データベース・キャラクタ・セットに加えて、2 つ目の代替キャラクタ・セットが使用できるようになりました。NCHAR データ型は、複数の固定幅のアジア言語のキャラクタ・セットをサポートしています。これらのキャラクタ・セットは、アジア言語の文字データの処理パフォーマンスを向上させるために導入されたものです。

Oracle9*i* では、SQL NCHAR データ型の使用は、Unicode キャラクタ・セット・エンコーディング（UTF8 と AL16UTF16）に制限されています。JA16SJISFIXED などのアジア言語のキャラクタ・セットも含めて、NCHAR データ型で使用可能であった Oracle8 Server の他のキャラクタ・セットは、現在ではサポートされていません。

既存の NCHAR、NVARCHAR2 および NCLOB の各列を Oracle9*i* の NCHAR データ型に移行する手順は、次のとおりです。

1. すべての NCHAR 列を Oracle8 または Oracle8*i* のデータベースからエクスポートします。
2. NCHAR 列を削除します。
3. データベースを Oracle9*i* にアップグレードします。
4. NCHAR 列を Oracle9*i* にインポートします。

また、Oracle9*i* 移行ユーティリティでは、Oracle8 および Oracle8*i* の NCHAR 列を 9*i* の NCHAR 列に変換できます。移行ユーティリティでは、`utlchar.sql` という SQL NCHAR のアップグレード・スクリプトが提供されています。このスクリプトをデータベース移行操作の最後に実行して、Oracle8 と Oracle8*i* の NCHAR 列を Oracle9*i* の NCHAR 列に変換します。スクリプトの実行後は、データをダウングレードできません。Oracle8 または Oracle8*i* に移動して戻る唯一の方法は、すべての NCHAR 列を削除してデータベースをダウングレードし、古い NCHAR データを前の Oracle8 または Oracle8*i* のエクスポート・ファイルからインポートすることです。データベースを将来ダウングレードする必要がある場合は、Oracle8 または Oracle8*i* の NCHAR データのバックアップ（エクスポート・ファイル）があることを確認してください。

関連項目：

- エクスポートとインポートの手順の詳細は、『Oracle9*i* データベース・ユーティリティ』を参照してください。
- NCHAR の移行の詳細は、『Oracle9*i* データベース移行ガイド』を参照してください。

各国語キャラクタ・セットの変更

各国語キャラクタ・セットを変更するには、ALTER DATABASE NATIONAL CHARACTER SET 文を使用します。この文の構文は、次のとおりです。

```
ALTER DATABASE [db_name] NATIONAL CHARACTER SET new_NCHAR_character_set;
```

db_name はオプションです。キャラクタ・セット名は、引用符を付けずに指定してください。

必要な場合は、ALTER DATABASE CHARACTER SET 文と ALTER DATABASE NATIONAL CHARACTER SET 文を一度に発行できます。

関連項目： ALTER DATABASE NATIONAL CHARACTER SET 文の構文は、『Oracle9*i* SQL リファレンス』を参照してください。

Oracle9*i* データベースでの CHAR 列から NCHAR 列への移行

次の方法で列のデータ型定義を変更できます。

- ALTER TABLE MODIFY 文
- 表のオンライン再定義

ALTER TABLE MODIFY 文には、表のオンライン再定義に比べて次の利点があります。

- 使用方法が簡単
- 制限が少ない

表のオンライン再定義には、ALTER TABLE MODIFY 文に比べて次の利点があります。

- 列に大量のデータが含まれている場合に高速である
- 一度に複数の列を移行可能
- ほとんどの移行プロセス中に表を DML に使用可能
- 表の断片化が回避されるため、領域が節約され、データへの高速アクセスが可能
- CLOB データ型から NCLOB データ型への移行に使用可能

この項の内容は、次のとおりです。

ALTER TABLE MODIFY 文を使用した CHAR 列から NCHAR 列への変更

ALTER TABLE MODIFY 文を使用すると、表の列定義を CHAR データ型から NCHAR データ型に変更できます。また、列のすべてのデータをデータベース・キャラクタ・セットから NCHAR キャラクタ・セットに変換できます。ALTER TABLE MODIFY 文の構文は、次のとおりです。

```
ALTER TABLE table_name MODIFY (column_name datatype);
```

移行する列に索引が作成されている場合、索引は各行とともに更新されるため、索引を削除すると ALTER TABLE MODIFY 文のパフォーマンスを改善できます。

NCHAR 列と NVARCHAR2 列の最大長は、それぞれ 2000 バイトと 4000 バイトです。NCHAR のキャラクタ・セットが AL16UTF16 の場合、NCHAR 列と NVARCHAR2 列の最大長は 1000 文字と 2000 文字、つまり 2000 バイトと 4000 バイトです。移行中にこのサイズ制限に対する違反となる場合は、かわりに列を NCLOB データ型に変更することを考慮してください。

注意： ALTER TABLE MODIFY 文を使用して CLOB 列を NCLOB 列に移行することはできません。列を CLOB データ型から NCLOB データ型に変更するには、表のオンライン再定義を使用します。

関連項目： 10-13 ページ「表のオンライン再定義を使用した大きい表の Unicode への移行」

表のオンライン再定義を使用した大きい表の Unicode への移行

多数の行を含む大きい表を Unicode データ型に移行するには、長時間かかります。この移行中は、列データを読込みや更新に使用できません。表のオンライン再定義を使用すると、移行時間を大幅に短縮できます。また、この場合、移行時間中の大部分は、表に DML でアクセスできます。

次のタスクを実行し、表のオンライン再定義を使用して、表を Unicode データ型に移行します。

1. DBMS_REDEFINITION.CAN_REDEF_TABLE PL/SQL プロシージャを使用して、表をオンラインで再定義できることを確認します。たとえば、scott.emp 表を移行するには、次のコマンドを入力します。

   ```
   DBMS_REDEFINITION.CAN_REDEF_TABLE('scott','emp');
   ```

2. 再定義する表と同じスキーマに空の仮表を作成します。作成時には、属性に NCHAR データ型を指定します。たとえば、次のように文を入力します。

```
CREATE TABLE int_emp(
    empno NUMBER(4),
    ename NVARCHAR2(10),
    job NVARCHAR2(9),
    mgr NUMBER(4),
    hiredate DATE,
    sal NUMBER(7,2),
    deptno NUMBER(2),
    org NVARCHAR2(10));
```

3. 表のオンライン再定義を開始します。次のようにコマンドを入力します。

```
DBMS_REDEFINITION.START_REDEF_TABLE('scott',
'emp',
'int_emp',
'empno empno,
to_nchar(ename) ename,
to_nchar(job) job,
mgr mgr,
hiredate hiredate,
sal sal,
deptno deptno,
to_nchar(org) org');
```

CLOB 列を NCLOB 列に移行する場合は、TO_NCHAR SQL 関数のかわりに TO_NCLOB SQL 変換関数を使用します。

4. 仮表のトリガー、索引、権限付与および制約を作成します。仮表に適用される参照制約は、DISABLED モードで作成する必要があります（仮表は、参照制約の親表または子表です）。仮表に定義するトリガーは、表のオンライン再定義プロセスが完了するまで実行されません。

5. 仮表を元の表と同期化できます。オンライン再定義の開始後に元の表に多数の DML 操作が適用されている場合は、DBMS_REDEFINITION.SYNC_INTERIM_TABLE プロシージャを実行します。これにより、DBMS_REDEFINITION.FINISH_REDEF_TABLE プロシージャの所要時間が短縮されます。次のようにコマンドを入力します。

```
DBMS_REDEFINITION.SYNC_INTERIM_TABLE('scott', 'emp', 'int_emp');
```

6. DBMS_REDEFINITION.FINISH_REDEF_TABLE プロシージャを実行します。次のようにコマンドを入力します。

```
DBMS_REDEFINITION.RINISH_REDEF_TABLE('scott', 'emp', 'int_emp');
```

このプロセスが完了すると、次の条件が満たされます。

- 元の表は、仮表のすべての属性、索引、制約、権限付与およびトリガーを持つように再定義されます。
- 仮表に適用される参照制約が、再定義後の元の表に適用されます。

7. 仮表を削除します。次のような文を入力します。

```
DROP TABLE int_emp;
```

表のオンライン再定義タスクの結果は、次のようになります。

- 元の表が Unicode 列に移行されます。
- START_REDEF_TABLE サブプログラムから FINISH_REDEF_TABLE サブプログラムまでに、仮表に対して定義されたトリガー、権限付与、索引および制約が、再定義後の元の表に定義されます。仮表に適用される参照制約が、再定義後の元の表に適用され、使用可能になります。
- 再定義前に元の表に定義されていたトリガー、権限付与、索引および制約が仮表に転送され、仮表を削除すると一緒に削除されます。仮表に再定義を適用する前に元の表に適用されていた参照制約が、使用禁止になります。
- 再定義前に元の表に定義されていた PL/SQL プロシージャとカーソルが無効になります。これらは、次回に使用されるときに自動的に再検証されます。表の定義が変更されているため、再検証は失敗することがあります。

関連項目： 表のオンライン再定義の詳細は、『Oracle9*i* データベース管理者ガイド』を参照してください。

キャラクタ・セット移行後にデータベース・スキーマをリカバリするタスク

移行後のデータベース・スキーマを元の状態にリカバリするために、追加のタスクが必要になる場合があります。表 10-1 に、考慮が必要な問題を示します。

表 10-1 移行したデータベース・スキーマのリカバリ中の問題

問題	説明
索引	ALTER TABLE MODIFY 文によって表の列が CHAR データ型から NCHAR データ型に変更されると、その列に作成される索引はデータベースによって自動的に変更されます。このため、ALTER TABLE MODIFY 文のパフォーマンスが低下します。ALTER TABLE MODIFY 文を発行する前に索引を削除した場合は、移行後に再作成します。
制約	移行前に制約を使用禁止にした場合は、移行後に再び使用可能にします。
トリガー	移行前にトリガーを使用禁止にした場合は、移行後に再び使用可能にします。
レプリケーション	Unicode データ型に移行する列が複数のサイト間でレプリケートされる場合は、マスター定義サイトで変更を実行する必要があります。これにより、変更が他のサイトに伝播します。
バイナリ順序	データベースと NCHAR データのキャラクタ・セットが異なる場合、CHAR データ型から NCHAR データ型への移行にはキャラクタ・セット変換が伴います。エンコーディングが異なると、同じデータのバイナリ順序が異なる可能性があります。これは、バイナリ順序に依存するアプリケーションに影響します。

11

Character Set Scanner

この章では、Character Set Scanner について説明します。このグローバリゼーション・サポート・ユーティリティは、キャラクタ・セットの移行前にデータをチェックします。この章の内容は、次のとおりです。

Character Set Scanner の概要

Character Set Scanner によって、Oracle データベースを新しいデータベース・キャラクタ・セットに移行するときの実現可能性と潜在的な問題点を評価できます。この Character Set Scanner によって、データベースのすべての文字データがチェックされ、キャラクタ・セット・エンコーディングの変更による影響と問題点がテストされます。スキャン終了時には、データベース・スキャンのサマリー・レポートが生成されます。このレポートには、データベースを新規キャラクタ・セットに変換するために必要な作業の範囲が表示されます。

サマリー・レポートの情報を参考にして、管理者は、データベースのキャラクタ・セットを移行する最も適切な方法を決定できます。次の移行方法があります。

- エクスポート・ユーティリティとインポート・ユーティリティ
- ALTER DATABASE CHARACTER SET 文
- 選択式のエクスポートとインポートを伴った ALTER DATABASE CHARACTER SET 文

注意： Character Set Scanner によって変換例外がレポートされている場合は、前述のいずれかの方法で変換を実行する前に、問題点を修正する必要があります。この修正には、問題のデータを修正して、これらの例外を取り除く作業も含まれます。極端な例では、データベースとアプリケーションの両方の修正が必要となる場合もあります。データベース・キャラクタ・セットの移行に関するサービスについては、オラクル社カスタマ・サポート・センターにお問い合せください。

関連項目： 10-8 ページ「既存のデータベースのデータベース・キャラクタ・セットの変更」

文字データの変換テスト

Character Set Scanner は、文字データを読み込み、各データ・セルについて次の条件をテストします。

- 新規キャラクタ・セットへの変換時に、データ・セルの文字コード・ポイントが変わるかどうか
- データ・セルは新規キャラクタ・セットに正常に変換できるかどうか
- 変換後のデータは現行の列サイズに適合するかどうか

Character Set Scanner は、CHAR、VARCHAR2、LONG、CLOB、NCHAR、NVARCHAR2 および NCLOB 列のデータについてのみ読み込んでテストします。LONG、CLOB および NCLOB 列については、変換後の列サイズのテストは実行しません。

アクセス権限

Character Set Scanner を使用するには、Oracle データベースの DBA 権限が必要です。

制限事項

CHAR、VARCHAR2、LONG および CLOB 列の文字ベースのデータはすべて、データベース・キャラクタ・セットに格納されます。このキャラクタ・セットは、データベースを最初に作成したときに、CREATE DATABASE 文で指定したデータベース・キャラクタ・セットです。ただし、構成によっては、意図的にまたは意図せずに、データベース・キャラクタ・セットと異なるキャラクタ・セットにデータを格納する可能性があります。これは、NLS_LANG キャラクタ・セットがデータベース・キャラクタ・セットと同一の場合に最もよく発生します。これは、Oracle がデータを変換または検証せずにそのまま送受信するためです。ただし、2 つのキャラクタ・セットの一方が他方のスーパーセットである場合にも発生します。この状況では、コード・ポイントの大部分が変換されない場合と同様に表示されます。たとえば、NLS_LANG が WE8ISO8859P1 に設定されていて、データベース・キャラクタ・セットが WE8MSWIN1252 の場合は、クライアント / サーバー変換を実行しても、128 ～ 159 の範囲以外のすべてのコード・ポイントが保持されます。

データベース・キャラクタ・セットにないデータが含まれているデータベースは、11-2 ページの「Character Set Scanner の概要」に記述されている 3 つの方法では別のキャラクタ・セットに変換できませんが、Character Set Scanner を使用すると、データがデータベース・キャラクタ・セットにあった場合に起こり得る、変換の影響をテストできます。

異なるキャラクタ・セットのエンコーディングでは、異なる文字に同じコード・ポイントが使用できます。どのような文字が意図されているのかを自動的に検出する方法はありません。ほとんどのヨーロッパ言語のキャラクタ・セットは、8 ビット範囲の自由な使用を共有してネイティブ文字をエンコードするため、不適切な理由でセルが互換性ありとレポートされる可能性が高くなります。

たとえば、FROMCHAR パラメータを WE8MSWIN1252 に設定して Character Set Scanner を使用すると、この問題が発生する可能性があります。このシングルバイト・キャラクタ・セットは、使用可能なすべてのコード・ポイントで文字をエンコードするため、スキャン対象がどのようなデータであっても、スキャナは常にデータ・セルを変換元キャラクタ・セットで使用可能と識別します。

FROMCHAR を設定する場合、すべての文字データがキャラクタ・セットにあっても、Character Set Scanner では妥当性を正確に判断できないものとみなすことになります。FROMCHAR パラメータは慎重に設定してください。

Character Set Scanner は、VARRAY コレクション型のスキャンをサポートしていません。

2 つ以上のキャラクタ・セットからのデータを含むデータベース

データベースに複数のキャラクタ・セットのデータが含まれている場合、Character Set Scanner は、データベース・キャラクタ・セットの変更による、そのデータベースへの影響を正確にテストできません。これは、Character Set Scanner ではそれらのキャラクタ・セットを簡単に区別できないためです。データを各キャラクタ・セットに対して 1 つずつ、2 つの表に分割できる場合、Character Set Scanner では 2 つの単一表のスキャンを実行してデータの妥当性を検証できます。

スキャンごとに、異なる値の FROMCHAR パラメータを使用し、指定したキャラクタ・セットにある場合と同様に、表内のターゲット列すべてを処理するように、Character Set Scanner に対して指示できます。

データベース・キャラクタ・セットからのデータを含まないデータベース

データベース・キャラクタ・セットにはなく、1 つのキャラクタ・セットのみにあるデータがデータベースに含まれている場合、Character Set Scanner では、全データベース・スキャンを実行できます。FROMCHAR パラメータを使用して、データが存在しているキャラクタ・セットを Character Set Scanner に通知します。

Character Set Scanner のスキャン・モード

Character Set Scanner には、次の 3 つのデータベース・スキャン・モードがあります。

- 全データベース・スキャン
- ユーザー・スキャン
- 表スキャン

全データベース・スキャン

Character Set Scanner は、データ・ディクショナリ（SYS ユーザー）も含めたデータベース内の全ユーザーに属しているすべての表の文字データを読み込んで検証し、新規データベース・キャラクタ・セットへの擬似的な移行による影響をレポートします。ストアド・パッケージ、プロシージャと関数およびオブジェクト名も含めて、すべてのスキーマ・オブジェクトをスキャンします。

新規データベース・キャラクタ・セットへの移行の実現可能性を把握するには、全データベース・スキャンを実行する必要があります。

ユーザー・スキャン

Character Set Scanner は、指定したユーザーに属しているすべての表の文字データを読み込んで検証し、キャラクタ・セットの変更による表への影響をレポートします。

Character Set Scanner は、表名や列名などの表定義はテストしません。スキーマ定義への影響を把握するには、全データベース・スキャンを実行する必要があります。

表スキャン

Character Set Scanner は、指定した表の文字データを読み込んで検証し、キャラクタ・セットの変更による表への影響をレポートします。

Character Set Scanner は、表名や列名などの表定義はテストしません。スキーマ定義への影響を把握するには、全データベース・スキャンを実行する必要があります。

Character Set Scanner の使用

この項では、スキャン前に実行しておく必要があるステップや Character Set Scanner の起動方法に関する手順など、Character Set Scanner の使用方法について説明します。この項の内容は、次のとおりです。

Character Set Scanner の使用前

Character Set Scanner を使用するには、スキャンするデータベースで、`csminst.sql` スクリプトを実行する必要があります。`csminst.sql` スクリプトを実行する必要があるのは 1 回のみです。このスクリプトは、次の作業を実行して、データベースをスキャンする準備を行います。

- ユーザー名 `CSMIG` の作成
- 必要な権限の `CSMIG` への割当て
- デフォルト表領域の `CSMIG` への割当て
- `CSMIG` での接続
- `CSMIG` の下に Character Set Scanner のシステム表を作成

デフォルトでは、SYSTEM 表領域が CSMIG に割り当てられるため、データベースをスキャンする前に、SYSTEM 表領域に使用可能な記憶域が十分あることを確認する必要があります。必要な領域の量は、スキャンのタイプおよびデータベース内のデータの性質によって異なります。

> **関連項目：** 11-29 ページ「Character Set Scanner の記憶域とパフォーマンスに関する考慮事項」

csminst.sql スクリプトを編集すると、CSMIG のデフォルト表領域を変更できます。csminst.sql で次の文を変更し、次のように、指定した表領域を CSMIG に割り当てます。

```
ALTER USER csmig DEFAULT TABLESPACE tablespace_name;
```

その後、次のコマンドと SQL 文を使用して csminst.sql を実行します。

```
% cd $ORACLE_HOME/rdbms/admin
% sqlplus "system/manager as sysdba"
SQL> START csminst.sql
```

Character Set Scanner の互換性

Character Set Scanner は、いずれのプラットフォームの Oracle データベースでも、同じリリースで実行されている場合は保証されます。ただし、ASCII ベースと EBCDIC ベースのプラットフォームは混在できません。たとえば、ASCII ベースのクライアント・プラットフォーム上の Oracle9*i* リリース 2（9.2）の Character Set Scanner は、ASCII ベースのいずれのプラットフォーム上の Oracle9*i* リリース 2（9.2）でも実行が保証されます。一方、EBCDIC ベースのクライアントは、EBCDIC プラットフォーム上のいずれの Oracle9*i* データベースでも実行が保証されます。

可能な場合は、データベースと同じ Oracle ホームで Character Set Scanner を実行することをお薦めします。

Character Set Scanner の起動

Character Set Scanner は、次の方法の 1 つを使用して起動できます。

- パラメータ・ファイルを使用する方法

  ```
  csscan system/manager PARFILE=filename
  ```

  PARFILE は、通常使用する Character Set Scanner パラメータが記述されているファイルです。

- コマンドラインを使用する方法

  ```
  csscan system/manager full=y tochar=utf8 array=10240 process=3
  ```

- 対話型セッションを使用する方法

  ```
  csscan system/manager
  ```

  対話型セッションでは、次のパラメータの入力が求められます。

  ```
  FULL/TABLE/USER
  TOCHAR
  ARRAY
  PROCESS
  ```

  ここにリストされていないパラメータを指定する場合は、パラメータ・ファイルまたはコマンドラインを使用して Character Set Scanner を起動する必要があります。

Character Set Scanner のオンライン・ヘルプの利用

Character Set Scanner には、オンライン・ヘルプが用意されています。コマンドラインで csscan help=y を入力して、ヘルプ画面を起動します。

CSSCAN コマンドの後にユーザー名とパスワードを入力すると、パラメータを要求するプロンプトを表示できます。たとえば、次のように指定します。

```
CSSCAN SYSTEM/MANAGER
```

また、CSSCAN コマンドの後に各種パラメータを入力すると、Character Set Scanner の実行方法を制御できます。パラメータを指定するには、キーワードを使用します。たとえば、次のように指定します。

```
CSSCAN SYSTEM/MANAGER FULL=y TOCHAR=utf8 ARRAY=102400 PROCESS=3
```

次に、Character Set Scanner のキーワードのリストを示します。

```
Keyword    Default Prompt Description
---------- ------- ------ --------------------------------------------------
USERID             yes    username/password
FULL       N       yes    scan entire database
USER               yes    user name of the table to scan
TABLE              yes    list of tables to scan
EXCLUDE                   list of tables to exclude from scan
TOCHAR             yes    new database character set name
FROMCHAR                  current database character set name
TONCHAR                   new NCHAR character set name
FROMNCHAR                 current NCHAR character set name
ARRAY      10240   yes    size of array fetch buffer
PROCESS    1       yes    number of scan process
MAXBLOCKS                 split table if larger than MAXBLOCKS
CAPTURE    N              capture convertible data
```

```
SUPPRESS                  suppress error log by N per table
FEEDBACK                  feedback progress every N rows
BOUNDARIES                list of column size boundaries for summary report
LASTRPT    N              generate report of the previous database scan
LOG        scan           base name of log files
PARFILE                   parameter file name
PRESERVE   N              preserve existing scan results
HELP       N              show help screen
```

パラメータ・ファイル

パラメータ・ファイルを使用すると、ファイルに Character Set Scanner パラメータを指定でき、パラメータの変更や再利用を簡単に実行できます。パラメータ・ファイルは、フラット・ファイル・テキスト・エディタを使用して作成します。コマンドライン・オプション PARFILE=*filename* によって、コマンドラインからではなく、指定したファイルからパラメータを読み込むことを Character Set Scanner に指示します。次に例を示します。

```
csscan parfile=filename
```

または

```
csscan username/password parfile=filename
```

パラメータ・ファイル指定の構文は、次のいずれかです。

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

パラメータ・ファイルの例を次に示します。

```
USERID=system/manager
USER=HR # scan HR's tables
TOCHAR=utf8
ARRAY=40960
PROCESS=2  # use two concurrent scan processes
FEEDBACK=1000
```

シャープ（#）記号を前に付けて文を記述すると、パラメータ・ファイルにコメントを追加できます。シャープ記号の右側の文字はすべて無視されます。

Character Set Scanner パラメータ

この項の内容は、次のとおりです。

ARRAY Character Set Scanner パラメータ

デフォルト値: 10240

最小値: 4096

最大値: 無制限

用途: データのフェッチに使用される配列バッファのサイズをバイト単位で指定します。配列バッファのサイズによって、一度にフェッチされる行数が決まります。

次の式で、一度にフェッチされる行数が見積もられます。

```
(rows in array) =
(ARRAY buffer size) / (sum of the CHAR and VARCHAR2 column sizes of a given table)
```

CHAR と VARCHAR2 の列サイズの合計が配列バッファ・サイズを超える場合は、一度に 1 行のみがフェッチされます。LONG、CLOB または NCLOB 列がある表は、一度に 1 行のみがフェッチされます。

このパラメータは、データベース・スキャンの処理時間に影響を与えます。通常、配列バッファのサイズが大きくなると、処理時間は短くなります。各スキャン・プロセスごとに、指定したサイズの配列バッファが割り当てられます。

BOUNDARIES Character Set Scanner パラメータ

デフォルト値 : なし

用途 : アプリケーション・データの変換サマリー・レポートに使用される列境界サイズのリストを指定します。このパラメータは、CHAR、VARCHAR2、NCHAR および NVARCHAR2 データ型のアプリケーション・データの分散位置を示すために使用されます。

たとえば、BOUNDARIES 値を (10, 100, 1000) と指定すると、アプリケーション・データの変換サマリー・レポートは、CHAR データをその列の長さによって、CHAR(1..10)、CHAR(11..100) および CHAR(101..1000) のグループに分割します。この動作は、VARCHAR2、NCHAR および NVARCHAR2 データ型の場合も同様です。

CAPTURE Character Set Scanner パラメータ

デフォルト値 : N

値の範囲 : Y または N

用途 : 例外行の格納のデフォルトと同様に、変換可能な個々の行に関する情報を取得するかどうかを示します。CAPTURE パラメータを Y に設定すると、変換可能行の情報が CSM$ERRORS 表に書き込まれます。この情報を使用すると、選択式のエクスポートとインポートによって変換先のキャラクタ・セットに変換する必要があるレコード数を推定できます。

EXCLUDE Character Set Scanner パラメータ

デフォルト値: なし

用途: スキャンから除外する表の名前を指定します。

このパラメータを指定すると、指定した表がスキャン対象から除外されます。表の名前は次の方法で指定できます。

- *schemaname* では、除外する表を含むユーザー・スキーマの名前を指定します。
- *tablename* では、除外する 1 つ以上の表の名前を指定します。

たとえば、次のコマンドでは、employees および departments 表を除き、hr サンプル・スキーマに属している表がすべてスキャンされます。

```
cssan system/manager USER=HR EXCLUDE=(HR.EMPLOYEES , HR.DEPARTMENTS) ...
```

FEEDBACK Character Set Scanner パラメータ

デフォルト値: なし

最小値: 100

最大値: 100000

用途: N 行がスキャンされるたび、ドットが 1 つ表示される形式で進捗メーターを表示することを指定します。

たとえば、FEEDBACK=1000 と指定すると、1000 行がスキャンされるたび、ドットが 1 つ表示されます。FEEDBACK 値は、スキャン対象の表すべてに適用されます。表ごとに個別に設定することはできません。

FROMCHAR Character Set Scanner パラメータ

デフォルト値: なし

用途: データベース内の CHAR、VARCHAR2、LONG および CLOB データ型の現行のキャラクタ・セット名を指定します。デフォルトでは、前述のデータ型のキャラクタ・セットは、データベース・キャラクタ・セットであるとみなされます。

このパラメータを使用すると、データベース内の CHAR、VARCHAR2、LONG および CLOB データのデフォルトのデータベース・キャラクタ・セット定義をオーバーライドできます。

FROMNCHAR Character Set Scanner パラメータ

デフォルト値: なし

用途: データベース内の NCHAR、NVARCHAR2 および NCLOB データ型の現行の各国語データベース・キャラクタ・セット名を指定します。デフォルトでは、前述のデータ型のキャラクタ・セットは、データベースの各国語キャラクタ・セットであるとみなされます。

このパラメータを使用すると、データベース内の NCHAR、NVARCHAR2 および NCLOB データのデフォルトのデータベース・キャラクタ・セット定義をオーバーライドできます。

FULL Character Set Scanner パラメータ

デフォルト値: N

値の範囲: Y または N

用途: 全データベース・スキャンを実行するかどうか（つまり、データ・ディクショナリも含めてデータベース全体をスキャンするかどうか）を示します。全データベース・モードでスキャンする場合は FULL=Y を指定します。

関連項目: 全データベース・スキャンの詳細は、11-4 ページの「Character Set Scanner のスキャン・モード」を参照してください。

HELP Character Set Scanner パラメータ

デフォルト値: N

値の範囲: Y または N

用途: Character Set Scanner パラメータの説明が記述されたヘルプ・メッセージを表示します。

関連項目: 11-7 ページ「Character Set Scanner のオンライン・ヘルプの利用」

LASTRPT Character Set Scanner パラメータ

デフォルト値: N

値の範囲: Y または N

用途: 直前のデータベース・スキャンで収集した統計情報に基づいて、Character Set Scanner レポートを再生成するかどうかを示します。

LASTRPT=Y と指定すると、データベースはスキャンされず、かわりに以前のデータベース・スキャン・セッションが残した情報を使用してレポート・ファイルが作成されます。

LASTRPT=Y を指定した場合は、USERID、BOUNDARIES および LOG パラメータのみが有効です。

LOG Character Set Scanner パラメータ

デフォルト値: scan

用途: 次の Character Set Scanner レポート・ファイルのベース・ファイル名を指定します。

- 拡張子が .txt のデータベース・スキャンのサマリー・レポート・ファイル
- 拡張子が .err の個別例外レポート・ファイル
- 拡張子が .out の画面ログ・ファイル

デフォルトでは、現行のディレクトリに scan.txt、scan.err および scan.out の 3 つのテキスト・ファイルが生成されます。

MAXBLOCKS Character Set Scanner パラメータ

デフォルト値: なし

最小値: 1000

最大値: 無制限

用途: 表ごとの最大ブロック・サイズを指定し、サイズの大きい表を Character Set Scanner が処理するための小さいチャンクに分割します。

たとえば、MAXBLOCKS パラメータを 1000 に設定すると、サイズが 1000 ブロックよりも大きい表は、n 個のチャンクに分割されます。この場合は、
n=CEIL(表ブロック・サイズ /1000) です。

大きい表をいくつかの小さい部分に分割する処理が有効なのは、PROCESS で設定されたプロセスの数が 1 より大きい場合のみです。MAXBLOCKS パラメータが設定されていない場合、Character Set Scanner は独自の最適化ルールに基づいて大きい表の分割を試みます。

PARFILE Character Set Scanner パラメータ

デフォルト値: なし

用途: Character Set Scanner パラメータのリストが含まれているファイルの名前を指定します。

関連項目: 11-8 ページ「パラメータ・ファイル」

PRESERVE Character Set Scanner パラメータ

デフォルト値: N

値の範囲: Y または N

用途: 前回のスキャン・セッションから収集された統計を保つかどうかを示します。

PRESERVE=Y と指定すると、前回のスキャンからの統計がすべて保たれます。現行のスキャン要求でスキャンされる表の新規統計は、追加（PRESERVE=Y の場合）または上書き（PRESERVE=N の場合）されます。

PROCESS Character Set Scanner パラメータ

デフォルト値: 1

最小値: 1

最大値: 32

用途: データベース・スキャンに利用するための同時実行スキャン・プロセスの数を指定します。

SUPPRESS Character Set Scanner パラメータ

デフォルト値: 未設定（行数は無制限）

最小値: 0

最大値: 無制限

用途: ログに記録されるデータ例外の最大数を表ごとに指定します。

データ・セルで例外が検出されると、CSM$ERRORS 表に個々の例外レコード情報が挿入されます。表は、レポートされる例外数に応じて大きくなります。

このパラメータを使用して、表ごとに指定した数の例外が挿入された後、個々の例外情報のロギングを抑制します。たとえば、SUPPRESS を 100 に設定すると、表ごとに最大 100 個の例外レコードが記録されます。

関連項目: 11-29 ページ「記憶域に関する考慮事項」

TABLE Character Set Scanner パラメータ

デフォルト値: なし

用途: スキャンする表の名前を指定します。

表の名前は次の方法で指定できます。

- *schemaname* では、スキャンする表を含むユーザー・スキーマの名前を指定します。
- *tablename* では、スキャンする 1 つ以上の表の名前を指定します。

たとえば、次のコマンドでは、hr サンプル・スキーマ内の employees および departments 表がスキャンされます。

```
csscan system/manager TABLE=(HR.EMPLOYEES , HR.DEPARTMENTS) ...
```

TOCHAR Character Set Scanner パラメータ

デフォルト値: なし

用途: CHAR、VARCHAR2、LONG および CLOB データの変換先データベース・キャラクタ・セット名を指定します。

TONCHAR Character Set Scanner パラメータ

デフォルト値: なし

用途: NCHAR、NVARCHAR2 および NCLOB データの変換先データベース・キャラクタ・セット名を指定します。

TONCHAR の値を指定しない場合、NCHAR、NVARCHAR2 および NCLOB データはスキャンされません。

USER Character Set Scanner パラメータ

デフォルト値: なし

用途: スキャンする表の所有者を指定します。

USER パラメータを指定すると、そのユーザーに属しているすべての表がスキャンされます。たとえば、次の文を使用すると、ユーザー hr に属しているすべての表がスキャンされます。

```
csscan system/manager USER=hr ...
```

USERID Character Set Scanner パラメータ

デフォルト値: なし

用途: データベースをスキャン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。パスワードを省略すると、パスワードの入力を求められます。

次の例はすべて有効です。

```
username/password
username/password@connect_string
username
username@connect_string
```

例 : Character Set Scanner セッション

次の例は、コマンドラインおよびパラメータ・ファイルを使用して、全データベース、ユーザーおよび表の各スキャン・モードを使用する方法を示しています。

例 : 全データベース・スキャン

次の例は、データベースを UTF8 に移行する場合の影響を調べるために、全データベースをスキャンする方法を示しています。この例では、現行のデータベース・キャラクタ・セットは WE8ISO8859P1（または UTF8 以外のキャラクタ・セット）であると仮定しています。

パラメータ・ファイルによる方法

```
% csscan system/manager parfile=param.txt
```

param.txt ファイルには、次の情報が記述されています。

```
full=y
tochar=utf8
array=40960
process=4
```

コマンドラインによる方法

```
% csscan system/manager full=y tochar=utf8 array=40960 process=4

Scanner Messages
Database Scanner: Release 9.2.0.1 - Production

(c) Copyright 2001 Oracle Corporation. All rights reserved.
Connected to:
Oracle9i Enterprise Edition Release 9.2.0.1 - Production
With the Objects option
PL/SQL Release 9.2.0.1 - Production

Enumerating tables to scan...

. process 1 scanning SYSTEM.REPCAT$_RESOLUTION
. process 1 scanning SYS.AQ$_MESSAGE_TYPES
. process 1 scanning SYS.ARGUMENT$
. process 2 scanning SYS.AUD$
. process 3 scanning SYS.ATTRIBUTE$
. process 4 scanning SYS.ATTRCOL$
. process 2 scanning SYS.AUDIT_ACTIONS
. process 2 scanning SYS.BOOTSTRAP$
. process 2 scanning SYS.CCOL$
. process 2 scanning SYS.CDEF$
```

```
 :
 :
. process 3 scanning SYSTEM.REPCAT$_REPOBJECT
. process 1 scanning SYSTEM.REPCAT$_REPPROP
. process 2 scanning SYSTEM.REPCAT$_REPSCHEMA
. process 3 scanning MDSYS.MD$DIM
. process 1 scanning MDSYS.MD$DICTVER
. process 2 scanning MDSYS.MD$EXC
. process 3 scanning MDSYS.MD$LER
. process 1 scanning MDSYS.MD$PTAB
. process 2 scanning MDSYS.MD$PTS
. process 3 scanning MDSYS.MD$TAB

Creating Database Scan Summary Report...

Creating Individual Exception Report...

Scanner terminated successfully.
```

例 : ユーザー・スキャン

次の例は、ユーザー表を UTF8 に移行する場合の影響を調べるために、ユーザー表をスキャンする方法を示しています。この例では、現行のデータベース・キャラクタ・セットは US7ASCII であるが、格納されている実際のデータは西ヨーロッパの WE8MSWIN1252 エンコーディングであると仮定しています。

パラメータ・ファイルによる方法

```
% csscan system/manager parfile=param.txt
```

param.txt ファイルには、次の情報が記述されています。

```
user=hr
fromchar=we8mswin1252
tochar=utf8
array=40960
process=1
```

コマンドラインによる方法

```
% csscan system/manager user=hr fromchar=we8mswin1252 tochar=utf8 array=40960
process=1
```

Character Set Scanner メッセージ

```
Database Scanner: Release 9.2.0.1 - Production

(c) Copyright 2001 Oracle Corporation. All rights reserved.

Connected to:
Oracle9i Enterprise Edition Release 9.2.0.1 - Production
With the Objects option
PL/SQL Release 9.2.0.1 - Production

Enumerating tables to scan...

. process 1 scanning HR.JOBS
. process 1 scanning HR.DEPARTMENTS
. process 1 scanning HR.JOB_HISTORY
. process 1 scanning HR.EMPLOYEES

Creating Database Scan Summary Report...

Creating Individual Exception Report...

Scanner terminated successfully.
```

例 : 単一表のスキャン

次の例は、単一表を WE8MSWIN1252 に移行する場合の影響を調べるために、単一表をスキャンする方法を示しています。この例では、現行のデータベース・キャラクタ・セットは US7ASCII であると仮定しています。

パラメータ・ファイルによる方法

```
% csscan system/manager parfile=param.txt
```

param.txt ファイルには、次の情報が記述されています。

```
table=employees
tochar=we8mswin1252
array=40960
process=1
supress=100
```

コマンドラインによる方法

```
% csscan system/manager table=employees tochar=we8mswin1252 array=40960 process=1
supress=100

Scanner Messages
Database Scanner: Release 9.2.0.1 - Production

(c) Copyright 2001 Oracle Corporation. All rights reserved.

Connected to:
Oracle9i Enterprise Edition Release 9.2.0.1 - Production
With the Objects option
PL/SQL Release 9.2.0.1 - Production

. process 1 scanning HR.EMPLOYEES

Creating Database Scan Summary Report...

Creating Individual Exception Report...

Scanner terminated successfully.
```

Character Set Scanner レポート

Character Set Scanner では、スキャンごとに次の 2 つのレポートが生成されます。

- データベース・スキャンのサマリー・レポート
- 個別例外レポート

データベース・スキャンのサマリー・レポート

データベース・スキャンのサマリー・レポートは、次の各項で構成されています。

- Character Set Scanner のデータベース・パラメータ
- データベース・サイズ
- スキャン・サマリー
- データ・ディクショナリの変換サマリー
- アプリケーション・データの変換サマリー
- 列サイズ境界ごとのアプリケーション・データの変換サマリー
- 表ごとの変換可能データの分散状態

- 列ごとの変換可能データの分散状態
- 再構築対象の索引

各項に出力される情報は、選択したスキャンのタイプおよびパラメータによって異なります。

Character Set Scanner のデータベース・パラメータ

この項には、選択したパラメータおよびスキャンのタイプが記述されます。次に出力例を示します。

```
Parameter                              Value
-------------------------------------- ------------------------------
Scan type                              Full database
Scan CHAR data?                        YES
Current database character set         WE8ISO8859P1
New database character set             UTF8
Scan NCHAR data?                       NO
Array fetch buffer size                102400
Number of processes                    4
-------------------------------------- ------------------------------
```

データベース・サイズ

この項には、現行のデータベース・サイズが記述されます。次に出力例を示します。

```
TABLESPACE                     Total(MB)        Used(MB)         Free(MB)
------------------------------ ---------------- ---------------- ----------------
APPS_DATA                       1,340.000        1,331.070            8.926
CTX_DATA                           30.000            3.145           26.852
INDEX_DATA                        140.000          132.559            7.438
RBS_DATA                          310.000          300.434            9.563
SYSTEM_DATA                       150.000          144.969            5.027
TEMP_DATA                         160.000                           159.996
TOOLS_DATA                         35.000           22.148           12.848
USERS_DATA                        220.000          142.195           77.801
------------------------------ ---------------- ---------------- ----------------
Total                           2,385.000        2,073.742          311.227
```

スキャン・サマリー

このレポートには、データベース・キャラクタ・セット移行の実現可能性が示されます。データベースのキャラクタ・セット移行の実現可能性を判断する基本的な規準は 2 つあります。1 つはデータ・ディクショナリの状態で、もう 1 つはアプリケーション・データの状態です。

スキャン・サマリー・セクションは、次の 2 つのステータス行で構成されています。スキャン・モードと結果によって、データ・ディクショナリとアプリケーション・データに関して出力されるステータスが決定されます。

表 11-1 データ・ディクショナリとアプリケーション・データのスキャン・サマリー

データ・ディクショナリに可能なステータス	アプリケーション・データに可能なステータス
データ・ディクショナリ内のキャラクタ・タイプ・データはすべて、新規キャラクタ・セットでも変わりません。	キャラクタ・タイプのアプリケーション・データはすべて、新規キャラクタ・セットでも変わりません。
データ・ディクショナリ内のキャラクタ・タイプ・データはすべて、新規キャラクタ・セットに変換できます。	キャラクタ・タイプのアプリケーション・データはすべて、新規キャラクタ・セットに変換できます。
データ・ディクショナリ内のキャラクタ・タイプ・データの一部は、新規キャラクタ・セットに変換できません。	キャラクタ・タイプのアプリケーション・データの一部は、新規キャラクタ・セットに変換できません。

次に出力例を示します。

```
All character type data in the data dictionary remains the same in the new character
set
All character type application data remains the same in the new character set
```

すべてのデータが新規キャラクタ・セットでも変わらない場合は、変換元キャラクタ・セットのデータ・エンコーディングが、変換先キャラクタ・セットと同一であることを意味します。この場合は、ALTER DATABASE CHARACTER SET 文を使用して、キャラクタ・セットを移行できます。

すべてのデータが新規キャラクタ・セットに変換できる場合、データは、エクスポート・ユーティリティとインポート・ユーティリティを使用して、安全に移行できることを意味します。ただし、移行後のデータ・エンコーディングは、変換元のキャラクタ・セットと異なることがあります。

関連項目：

- 変換不可能なデータの詳細は、11-27 ページの「個別例外レポート」を参照してください。
- 10-9 ページ「ALTER DATABASE CHARACTER SET 文を使用した文字データの移行」
- 10-8 ページ「全体エクスポートおよび全体インポートを使用した文字データの移行」

データ・ディクショナリの変換サマリー

この項には、データ・ディクショナリの変換サマリーの統計情報が記述されます。統計情報はデータ型別にレポートされます。表 11-2 に、レポートできるステータスのタイプを示します。

表 11-2 データ・ディクショナリのデータ変換サマリー

ステータス	説明
Changeless	新規キャラクタ・セットでも変わらないデータ・セルの数。
Convertible	新規キャラクタ・セットに正常に変換されるデータ・セルの数。
Exceptional	変換できないデータ・セルの数。変換する場合は、一部の文字が失われるか、またはデータが切り捨てられます。

この情報は、全データベース・スキャンが実行される場合にのみ使用可能です。次に出力例を示します。

```
Datatype        Changeless      Convertible      Exceptional            Total
--------- ---------------- ---------------- ---------------- ----------------
VARCHAR2           971,300                1                0          971,301
CHAR                     7                0                0                7
LONG                60,325                0                0           60,325
CLOB
--------- ---------------- ---------------- ---------------- ----------------
Total           1,031,632                1                0        1,031,633
```

Convertible 列と Exceptional 列の両方の数値がすべて 0（ゼロ）の場合は、データ・ディクショナリのすべてのデータが新規キャラクタ・セットでも変わらないことを意味します。

Exceptional 列の数値がすべて 0（ゼロ）で、Convertible 列に 0（ゼロ）以外の数値がある場合は、データ・ディクショナリのすべてのデータが新規キャラクタ・セットに変換できることを意味します。対象となるデータは、インポート時に変換されます。

Exceptional 列の数値が 0（ゼロ）以外の場合は、データ・ディクショナリに変換できないデータがあることを意味します。したがって、現行データベースの新規キャラクタへの移行は実現しません。これは、エクスポートとインポートのプロセスではデータを新規キャラクタ・セットに変換できないためです。たとえば、表名に無効な文字が使用されていたり、新規キャラクタ・セットにマップできないデータが PL/SQL プロシージャのコメント行に含まれている場合があります。スキーマ・オブジェクトに対するこのような変更は、新規キャラクタ・セットへの移行前に、手動で訂正する必要があります。

アプリケーション・データの変換サマリー

この項には、アプリケーション・データの変換サマリーの統計情報が記述されます。統計情報はデータ型別にレポートされます。表 11-3 に、レポートできるステータスのタイプを示します。

表 11-3 アプリケーション・データのデータ変換サマリー

ステータス	説明
Changeless	新規キャラクタ・セットでも変わらないデータ・セルの数。
Convertible	新規キャラクタ・セットに正常に変換されるデータ・セルの数。
Exceptional	変換できないデータ・セルの数。変換する場合は、一部の文字が失われるか、またはデータが切り捨てられます。

次に出力例を示します。

```
Datatype         Changeless      Convertible      Exceptional            Total
---------- ---------------- ---------------- ---------------- ----------------
VARCHAR2         23,213,745            1,324                0       23,215,069
CHAR                423,430                0                0          423,430
LONG                  8,624               33                0            8,657
CLOB                 58,839           11,114               28           69,981
---------- ---------------- ---------------- ---------------- ----------------
Total            23,704,638           12,471               28       23,717,137
```

列サイズ境界ごとのアプリケーション・データの変換サマリー

この項には、CHAR および VARCHAR2 のアプリケーション・データの変換サマリーが記述されます。統計は、BOUNDARIES パラメータで指定した列サイズ境界別にレポートされます。表 11-4 に、使用可能なステータスのタイプを示します。

表 11-4 アプリケーション・データの列のデータ変換サマリー

ステータス	説明
Changeless	新規キャラクタ・セットでも変わらないデータ・セルの数。
Convertible	新規キャラクタ・セットに正常に変換されるデータ・セルの数。
Exceptional	変換できないデータ・セルの数。変換する場合は、一部の文字が失われるか、またはデータが切り捨てられます。

この情報は、BOUNDARIES パラメータが指定されている場合にのみ使用可能です。

次に出力例を示します。

```
Datatype               Changeless   Convertible     Exceptional            Total
-------------------   ------------ ------------ --------------- ----------------
VARCHAR2(1..10)         1,474,825            0               0        1,474,825
VARCHAR2(11..100)       9,691,520           71               0        9,691,591
VARCHAR2(101..4000)    12,047,400        1,253               0       12,048,653
-------------------   ------------ ------------ --------------- ----------------
CHAR(1..10)               423,413            0               0          423,413
CHAR(11..100)                  17            0               0               17
CHAR(101..4000)
-------------------   ------------ ------------ --------------- ----------------
Total                  23,637,175        1,324               0       23,638,499
```

表ごとの変換可能データの分散状態

次のレポートは、Convertible データと Exceptional データがデータベース内にどのように分散しているかを示します。統計は、表別にレポートされます。リストに数行のみ表示されている場合は、Convertible データが局所的に集中していることを意味します。リストに多数の行が表示されている場合は、Convertible データがデータベース全体に分散していることを意味します。

次に出力例を示します。

```
USER.TABLE                                          Convertible      Exceptional
--------------------------------------------------- ---------------- ----------------
SMG.SOURCE                                                        1                0
SMG.HELP                                                         12                0
SMG.CLOSE_LIST                                                   16                0
SMG.ATTENDEES                                                     8                0
SGT.DR_010_I1T1                                                   7                0
SGT.DR_011_I1T1                                                   7                0
```

```
SGT.MRK_SRV_PROFILE                                 2               0
SGT.MRK_SRV_PROFILE_TEMP                            2               0
SGT.MRK_SRV_QUESTION                                3               0
------------------------------------------- --------------- ---------------
```

列ごとの変換可能データの分散状態

次のレポートは、Convertible データと Exceptional データがデータベース内にどのように分散しているかを示します。統計は、列別にレポートされます。次に出力例を示します。

```
USER.TABLE|COLUMN                             Convertible      Exceptional
------------------------------------------- --------------- ---------------
SMG.SOURCE|SOURCE                                     1               0
SMG.HELP|INFO                                        12               0
SMG.CLOSE_LIST|FNAME                                  1               0
SMG.CLOSE_LIST|LNAME                                  1               0
SMG.CLOSE_LIST|COMPANY                                1               0
SMG.CLOSE_LIST|STREET                                 8               0
SMG.CLOSE_LIST|CITY                                   4               0
SMG.CLOSE_LIST|STATE                                  1               0
SMG.ATTENDEES|ATTENDEE_NAME                           1               0
SMG.ATTENDEES|ADDRESS1                                3               0
SMG.ATTENDEES|ADDRESS2                                2               0
SMG.ATTENDEES|ADDRESS3                                2               0
SGT.DR_010_I1T1|WORD_TEXT                             7               0
SGT.DR_011_I1T1|WORD_TEXT                             7               0
SGT.MRK_SRV_PROFILE|FNAME                             1               0
SGT.MRK_SRV_PROFILE|LNAME                             1               0
SGT.MRK_SRV_PROFILE_TEMP|FNAME                        1               0
SGT.MRK_SRV_PROFILE_TEMP|LNAME                        1               0
SGT.MRK_SRV_QUESTION|ANSWER                           3               0
```

再構築対象の索引

データベース・キャラクタ・セットの移行によって影響を受ける全索引のリストが生成されます。これらの索引は、データのインポート後に再作成できます。次に例を示します。

```
USER.INDEX on USER.TABLE(COLUMN)
-------------------------------------------------------------------------------
CD2000.COMPANY_IX_PID_BID_NNAME on CD2000.COMPANY(CO_NLS_NAME)
CD2000.I_MASHINE_MAINT_CONT on CD2000.MACHINE(MA_MAINT_CONT#)
CD2000.PERSON_NEWS_SABUN_CONT_CONT on
CD2000.PERSON_NEWS_SABUN_CONT(CONT_BID)
CD2000.PENEWSABUN3_PEID_CONT on CD2000.PE_NEWS_SABUN_3(CONT_BID)
PMS2000.CALLS_IX_STATUS_SUPPMGR on PMS2000.CALLS(SUPPMGR)
PMS2000.MAILQUEUE_CHK_SUB_TOM on PMS2000.MAIL_QUEUE(TO_MAIL)
PMS2000.MAILQUEUE_CHK_SUB_TOM on PMS2000.MAIL_QUEUE(SUBJECT)
PMS2000.TMP_IX_COMP on PMS2000.TMP_CHK_COMP(COMP_NAME)
-------------------------------------------------------------------------------
```

個別例外レポート

個別例外レポートは、次のサマリーで構成されています。

- データベース・スキャン・パラメータ
- データ・ディクショナリの個別例外
- アプリケーション・データ個別例外

データベース・スキャン・パラメータ

この項には、選択したパラメータおよびスキャンのタイプが記述されます。次に出力例を示します。

```
Parameter                                Value
---------------------------------------- ------------------------------
Scan type                                Full database
Scan CHAR data?                          YES
Current database character set           we8mswin1252
New database character set               utf8
Scan NCHAR data?                         NO
Array fetch buffer size                  102400
Number of rows to heap up for insert     10
Number of processes                      1
---------------------------------------- ------------------------------
```

データ・ディクショナリの個別例外

この項では、変換可能なデータ・ディクショナリと例外があるデータ・ディクショナリのデータが識別されます。例外には 2 つのタイプがあります。

- 列サイズの超過 (exceed column size)
- 非可逆式変換 (lossy conversion)

次に、変換可能なデータを含むデータ・ディクショナリに関する出力例を示します。

```
User  : SYS
Table : METASTYLESHEET
Column: STYLESHEET
Type  : CLOB
Number of Exceptions         : 0
Max Post Conversion Data Size: 0

ROWID              Exception Type      Size Cell Data(first 30 bytes)
------------------ ------------------ ----- ------------------------------
AAAAHMAABAAAAs+AAA convertible
AAAAHMAABAAAAs+AAB convertible
------------------ ------------------ ----- ------------------------------
```

関連項目： 例外の詳細は、11-28 ページの「アプリケーション・データ個別例外」を参照してください。

アプリケーション・データ個別例外

このレポートには、例外があるデータが示されるため、必要に応じて該当データを修正できます。

例外には 2 つのタイプがあります。

- 列サイズの超過 (exceed column size)

  最大列幅を超過している場合は、列サイズを拡張する必要があります。列サイズを拡張しないと、データの切捨てが発生します。

- 非可逆式変換 (lossy conversion)

  新規キャラクタ・セットへの移行前にデータを訂正する必要があります。データを訂正しないと、無効な文字が置換文字に変換されます。置換文字は通常、? または ¿、あるいは言語的に同様の文字です。

次の個別例外レポートの例では、データベース・キャラクタ・セットを WE8ISO8859P1 から UTF8 に変更するときに発生する可能性があるいくつかの問題が示されています。

```
User:   HR
Table:  EMPLOYEES
Column: LAST_NAME
Type:   VARCHAR2(10)
Number of Exceptions: 2
Max Post Conversion Data Size: 11

ROWID              Exception Type      Size Cell Data(first 30 bytes)
------------------ ------------------ ----- ------------------------------
AAAA2fAAFAABJwQAAg exceed column size    11 Ährenfeldt
AAAA2fAAFAABJwQAAu lossy conversion         órâclë8™
AAAA2fAAFAABJwQAAu exceed column size    11 órâclë8™
------------------ ------------------ ----- ------------------------------
```

値 Ährenfeldt および órâclë8™ は、列サイズ（10 バイト）を超過しています。これは、Ä、ó、â および ë の各文字が、WE8ISO8859P1 では 1 バイトですが UTF8 では 2 バイトであるためです。値 órâclë8™ は、商標記号 ™（コード 153）が有効な WE8ISO8859P1 文字ではないことに起因する UTF8 への不可逆変換です。この文字は、WE8ISO8859P1 のスーパーセットである WE8MSWIN1252 の文字です。

SELECT 文を発行すると、例外があるデータを表示できます。

```
SELECT last_name FROM hr.employees
WHERE ROWID='AAAA2fAAFAABJwQAAu';
```

UPDATE 文を発行すると、例外があるデータを修正できます。

```
UPDATE hr.employees SET last_name = 'Oracle8 TM'
WHERE ROWID='AAAA2fAAFAABJwQAAu';
```

関連項目：

- 10-2 ページ「データの切捨て」
- 10-4 ページ「キャラクタ・セット変換の問題」

Character Set Scanner の記憶域とパフォーマンスに関する考慮事項

この項では、Character Set Scanner の記憶域とパフォーマンスに関する問題について説明します。この項の内容は、次のとおりです。

- 記憶域に関する考慮事項
- パフォーマンスに関する考慮事項

記憶域に関する考慮事項

この項では、Character Set Scanner のシステム表のサイズと拡張について説明し、それらをメンテナンスする方法について説明します。データベースに格納されているデータの性質に応じて、急速に増大する可能性があるシステム表が 3 つあります。

csminst.sql スクリプトを修正すると、ユーザー CSMIG に大きい表領域を割り当てることができます。デフォルトでは、SYSTEM 表領域がユーザー CSMIG に割り当てられます。

この項の内容は、次のとおりです。

- CSM$TABLES
- CSM$COLUMNS
- CSM$ERRORS

CSM$TABLES

スキャンする必要があるすべての表が、CSM$TABLES 表に列挙されます。

次の SQL 文を発行すると、データベース内の表の数を調べる（CSM$TABLES がどの程度大きくなるかの見積りを取得する）ことができます。

```
SELECT COUNT(*) FROM DBA_TABLES;
```

CSM$COLUMNS

スキャンされた各列の統計情報が、CSM$COLUMNS 表に格納されます。

次の SQL 文を発行すると、データベース内のキャラクタ・タイプ列の数を調べる（CSM$COLUMNS がどの程度大きくなるかの見積りを取得する）ことができます。

```
SELECT COUNT(*) FROM DBA_TAB_COLUMNS
WHERE DATA_TYPE IN ('CHAR', 'VARCHAR2', 'LONG', 'CLOB');
```

CSM$ERRORS

セル・データで例外が検出されると、個々の例外情報が CSM$ERRORS 表に挿入されます。この情報は個別例外レポートに表示され、必要に応じて修正するレコードの識別に役立ちます。

データベースに Exceptional または Convertible（パラメータ CAPTURE=Y が設定されている場合）の印が付いたデータが多数含まれている場合は、CSM$ERRORS 表が極端に大きくなる可能性があります。SUPPRESS パラメータを使用すると、CSM$ERRORS 表が不必要に増大することを防止できます。

SUPPRESS パラメータはすべての表に適用されます。指定した数の例外が挿入された後は、個々の Exceptional 情報の挿入が抑制されます。記録される例外数を制限しても、例外が異なる表に散在している場合は役に立たない可能性があります。

パフォーマンスに関する考慮事項

この項では、データベース・スキャン時のパフォーマンスを向上させる方法を説明します。

マルチ・スキャン・プロセスの使用

比較的大規模なデータベース（50GB 以上のデータベースなど）のスキャンを計画している場合は、マルチ・スキャン・プロセスの利用を考慮できます。マルチ・スキャン・プロセスを使用すると、マシンで使用可能な CPU やメモリーなどのハードウェア・リソースを使用することによって、データベース・スキャンの継続時間が短縮されます。使用するスキャン・プロセスの数を判断するためのガイドラインは、CPU_COUNT 初期化パラメータ値と同じ値に設定することです。

配列フェッチ・バッファ・サイズ

配列フェッチを許可すると、1 回に複数行がフェッチされます。通常は、大きなサイズの配列フェッチ・バッファを使用すると、パフォーマンスが向上します。各プロセスでは、独自のフェッチ・バッファが割り当てられます。

例外および変換可能ログの抑制

Exceptional および Convertible（CAPTURE=Y の場合）の個別情報は、CSM$ERRORS 表に挿入されます。通常、CSM$ERRORS 表への挿入は、データ・フェッチよりもリソースを

消費します。データベースに Exceptional または Convertible の印がついたデータが多数あると、多数の挿入文が発行されるため、パフォーマンスが低下します。SUPRESS パラメータを使用して、記録する例外行の数に制限を設定することをお薦めします。

Character Set Scanner のビューとメッセージ

この項には、次のリファレンス・データが含まれています。

- Character Set Scanner のビュー
- Character Set Scanner エラー・メッセージ

Character Set Scanner のビュー

Character Set Scanner では、次のビューが使用されます。

CSMV$COLUMNS

このビューには、スキャンされた列の統計情報が含まれます。

列	データ型	NULL	説明
OWNER_ID	NUMBER	NOT NULL	表所有者のユーザー ID
OWNER_NAME	VARCHAR2(30)	NOT NULL	表所有者のユーザー名
TABLE_ID	NUMBER	NOT NULL	表のオブジェクト ID
TABLE_NAME	VARCHAR2(30)	NOT NULL	表のオブジェクト名
COLUMN_ID	NUMBER	NOT NULL	列 ID
COLUMN_INTID	NUMBER	NOT NULL	内部列 ID（抽象データ型用）
COLUMN_NAME	VARCHAR2(30)	NOT NULL	列名
COLUMN_TYPE	VARCHAR2(9)	NOT NULL	列データ型
TOTAL_ROWS	NUMBER	NOT NULL	この表の行数
NULL_ROWS	NUMBER	NOT NULL	NULL データ・セルの数
CONV_ROWS	NUMBER	NOT NULL	変換する必要があるデータ・セルの数
ERROR_ROWS	NUMBER	NOT NULL	エラーがあるデータ・セルの数
EXCEED_SIZE_ROWS	NUMBER	NOT NULL	例外があるデータ・セルの数
DATA_LOSS_ROWS	NUMBER	-	不可逆変換を行ったデータ・セルの数
MAX_POST_CONVERT_SIZE	NUMBER	-	変換後の最大データ・サイズ

CSMV$CONSTRAINTS

このビューには、スキャンされた列の統計情報が含まれます。

列	データ型	NULL	説明
OWNER_ID	NUMBER	NOT NULL	制約所有者のユーザー ID
OWNER_NAME	VARCHAR2(30)	NOT NULL	制約所有者のユーザー名
CONSTRAINT_ID	NUMBER	NOT NULL	制約のオブジェクト ID
CONSTRAINT_NAME	VARCHAR2(30)	NOT NULL	制約のオブジェクト名
CONSTRAINT_TYPE#	NUMBER	NOT NULL	制約タイプ番号
CONSTRAINT_TYPE	VARCHAR2(11)	NOT NULL	制約タイプ名
TABLE_ID	NUMBER	NOT NULL	表のオブジェクト ID
TABLE_NAME	VARCHAR2(30)	NOT NULL	表のオブジェクト名
CONSTRAINT_RID	NUMBER	NOT NULL	ルート制約 ID
CONSTRAINT_LEVEL	NUMBER	NOT NULL	制約レベル

CSMV$ERRORS

このビューには、セル・データおよびオブジェクト定義の個別例外情報が含まれます。

列	データ型	NULL	説明
OWNER_ID	NUMBER	NOT NULL	表所有者のユーザー ID
OWNER_NAME	VARCHAR2(30)	NOT NULL	表所有者のユーザー名
TABLE_ID	NUMBER	NOT NULL	表のオブジェクト ID
TABLE_NAME	VARCHAR2(30)	-	表のオブジェクト名
COLUMN_ID	NUMBER	-	列 ID
COLUMN_INTID	NUMBER	-	内部列 ID（抽象データ型用）
COLUMN_NAME	VARCHAR2(30)	-	列名
DATA_ROWID	VARCHAR2(1000)	-	データの行 ID
COLUMN_TYPE	VARCHAR2(9)	-	オブジェクト型の列データ型
ERROR_TYPE	VARCHAR2(11)	-	発生したエラーのタイプ

CSMV$INDEXES

このビューには、索引に関する個々の例外情報が含まれます。

列	データ型	NULL	説明
INDEX_OWNER_ID	NUMBER	NOT NULL	索引所有者のユーザー ID
INDEX_OWNER_NAME	VARCHAR2(30)	NOT NULL	索引所有者のユーザー名
INDEX_ID	NUMBER	NOT NULL	索引のオブジェクト ID
INDEX_NAME	VARCHAR2(30)	-	索引のオブジェクト名
INDEX_STATUS#	NUMBER	-	索引のステータス番号
INDEX_STATUS	VARCHAR2(8)	-	索引のステータス
TABLE_OWNER_ID	NUMBER	-	表所有者のユーザー ID
TABLE_OWNER_NAME	VARCHAR2(30)	-	表所有者のユーザー名
TABLE_ID	NUMBER	-	表のオブジェクト ID
TABLE_NAME	VARCHAR2(30)	-	表のオブジェクト名
COLUMN_ID	NUMBER	-	列 ID
COLUMN_INTID	NUMBER	-	内部列 ID（抽象データ型用）
COLUMN_NAME	VARCHAR2(30)	-	列名

CSMV$TABLES

このビューには、スキャンするデータベース表に関する情報が含まれます。スキャンするすべての表がこのビューに列挙されます。

列	データ型	NULL	説明
OWNER_ID	NUMBER	NOT NULL	表所有者のユーザー ID
OWNER_NAME	VARCHAR2(30)	NOT NULL	表所有者のユーザー名
TABLE_ID	NUMBER	-	表のオブジェクト ID
TABLE_NAME	VARCHAR2(30)	-	表のオブジェクト名
MIN_ROWID	VARCHAR2(18)	-	表の分割範囲の最小行 ID
MAX_ROWID	VARCHAR2(18)	-	表の分割範囲の最大行 ID
BLOCKS	NUMBER	-	分割範囲内のブロック数
SCAN_COLUMNS	NUMBER	-	スキャンされる列数
SCAN_ROWS	NUMBER	-	スキャンされる行数

列	データ型	NULL	説明
SCAN_START	VARCHAR2(8)	-	表スキャンの開始時間
SCAN_END	VARCHAR2(8)	-	表スキャンの終了時間

Character Set Scanner エラー・メッセージ

Character Set Scanner には、次のエラー・メッセージがあります。

CSS-00100 failed to allocate memory size of number
原因： サイズ 0（ゼロ）または最大サイズよりも大きい値でメモリーを割り当てようとしました。
処置： 内部エラーです。オラクル社カスタマ・サポート・センターにお問い合せください。

CSS-00101 failed to release memory
原因： 無効なポインタでメモリーを解放しようとしました。
処置： 内部エラーです。オラクル社カスタマ・サポート・センターにお問い合せください。

CSS-00102 failed to release memory, null pointer given
原因： NULL ポインタでメモリーを解放しようとしました。
処置： 内部エラーです。オラクル社カスタマ・サポート・センターにお問い合せください。

CSS-00105 failed to parse BOUNDARIES parameter
原因： BOUNDARIES パラメータが無効な書式で指定されました。
処置： 正しい構文はマニュアルを参照してください。

CSS-00106 failed to parse SPLIT parameter
原因： SPLIT パラメータが無効な書式で指定されました。
処置： 正しい構文はマニュアルを参照してください。

CSS-00107 Character set migration utility schem not installed
原因： CSM$VERSION がデータベースにありません。
処置： データベースで CSMINST.SQL を実行してください。

CSS-00108 Character set migration utility schema not compatible
原因： 非互換の CSM$* 表がデータベースにあります。
処置： データベースで CSMINST.SQL を実行してください。

CSS-00110 failed to parse userid
原因： USERID パラメータが無効な書式で指定されました。
処置： 正しい構文はマニュアルを参照してください。

CSS-00111 failed to get RDBMS version
原因： データベースのバージョン値の取得に失敗しました。
処置： 内部エラーです。オラクル社カスタマ・サポート・センターにお問い合せください。

CSS-00112 database version not supported
原因： データベースのバージョンがリリース 8.0.5.0.0 より前です。
処置： データベースをリリース 8.0.5.0.0 以上にアップグレードして、再試行してください。

CSS-00113 user %s is not allowed to access data dictionary
原因： 指定したユーザーは、データ・ディクショナリにアクセスできません。
処置： O7_DICTIONARY_ACCESSIBILITY パラメータを TRUE に設定するか、SYS ユーザーを使用してください。

CSS-00114 failed to get database character set name
原因： NLS_DATABASE_PARAMETERS ビューからの NLS_CHARACTERSET または NLS_NCHAR_CHARACTERSET パラメータの値の取得に失敗しました。
処置： 内部エラーです。オラクル社カスタマ・サポート・センターにお問い合せください。

CSS-00115 invalid character set name %s
原因： 指定したキャラクタ・セットは、有効な Oracle キャラクタ・セットではありません。

> **関連項目：** 適切なキャラクタ・セット名については、付録 A「ロケール・データ」を参照してください。

CSS-00116 failed to reset NLS_LANG/NLS_NCHAR parameter
原因： NLS_LANG キャラクタ・セットをデータベース・キャラクタ・セットと一致させることができませんでした。
処置： 内部エラーです。オラクル社カスタマ・サポート・センターにお問い合せください。

CSS-00117 failed to clear previous scan log
原因： CSM$* 表からの全行削除に失敗しました。
処置： 内部エラーです。オラクル社カスタマ・サポート・センターにお問い合せください。

CSS-00118 failed to save command parameters
原因： CSM$PARAMETERS 表への複数行の挿入に失敗しました。
処置： 内部エラーです。オラクル社カスタマ・サポート・センターにお問い合せください。

CSS-00119 failed to save scan start time
原因： CSM$PARAMETERS 表への行の挿入に失敗しました。
処置： 内部エラーです。オラクル社カスタマ・サポート・センターにお問い合せください。

CSS-00120 failed to enumerate tables to scan
原因： スキャンする表の CSM$TABLES 表への列挙に失敗しました。
処置： 内部エラーです。オラクル社カスタマ・サポート・センターにお問い合せください。

CSS-00121 failed to save scan complete time
原因： CSM$PARAMETERS 表への行の挿入に失敗しました。
処置： 内部エラーです。オラクル社カスタマ・サポート・センターにお問い合せください。

CSS-00122 failed to create scan report
原因： データベース・スキャン・レポートの作成に失敗しました。
処置： 内部エラーです。オラクル社カスタマ・サポート・センターにお問い合せください。

CSS-00123 failed to check if user %s exist
原因： 指定したユーザーがデータベースに存在するかどうかをチェックする Select 文でエラーが発生しました。
処置： 内部エラーです。オラクル社カスタマ・サポート・センターにお問い合せください。

CSS-00124 user %s not found
原因： 指定したユーザーはデータベースに存在していません。
処置： ユーザー名をチェックしてください。

CSS-00125 failed to check if table %s.%s exist
原因： 指定した表がデータベースに存在するかどうかをチェックする Select 文でエラーが発生しました。
処置： 内部エラーです。オラクル社カスタマ・サポート・センターにお問い合せください。

CSS-00126 table %s.%s not found
原因： 指定した表はデータベースに存在していません。
処置： ユーザー名と表名をチェックしてください。

CSS-00127 user %s does not have DBA privilege
原因： 指定したユーザーには、データベースのスキャンに必要な DBA 権限がありません。
処置： DBA 権限が付与されているユーザーを選択してください。

CSS-00128 failed to get server version string
原因： データベースのバージョン文字列の取得に失敗しました。
処置： ありません。

CSS-00130 failed to initialize semaphore
原因： 不明です。
処置： 内部エラーです。オラクル社カスタマ・サポート・センターにお問い合せください。

CSS-00131 failed to spawn scan process %d
原因： 不明です。
処置： 内部エラーです。オラクル社カスタマ・サポート・センターにお問い合せください。

CSS-00132 failed to destroy semaphore
原因： 不明です。
処置： 内部エラーです。オラクル社カスタマ・サポート・センターにお問い合せください。

CSS-00133 failed to wait semaphore
原因： 不明です。
処置： 内部エラーです。オラクル社カスタマ・サポート・センターにお問い合せください。

CSS-00134 failed to post semaphore
原因： 不明です。
処置： 内部エラーです。オラクル社カスタマ・サポート・センターにお問い合せください。

CSS-00140 failed to scan table (tid=%d, oid=%d)
原因： この特定の表でデータ・スキャンに失敗しました。
処置： 内部エラーです。オラクル社カスタマ・サポート・センターにお問い合せください。

CSS-00141 failed to save table scan start time
原因： CSM$TABLES 表の行の更新に失敗しました。
処置： 内部エラーです。オラクル社カスタマ・サポート・センターにお問い合せください。

CSS-00142 failed to get table information
原因： 表のユーザー ID およびオブジェクト ID からの様々な情報の取得に失敗しました。
処置： 内部エラーです。オラクル社カスタマ・サポート・センターにお問い合せください。

CSS-00143 failed to get column attributes
原因： 表の列属性の取得に失敗しました。
処置： 内部エラーです。オラクル社カスタマ・サポート・センターにお問い合せください。

CSS-00144 failed to scan table %s.%s
原因： この特定の表でデータ・スキャンが正常に終了しませんでした。
処置： 内部エラーです。オラクル社カスタマ・サポート・センターにお問い合せください。

CSS-00145 failed to save scan result for columns
原因： CSM$COLUMNS 表への行の挿入に失敗しました。
処置： 内部エラーです。オラクル社カスタマ・サポート・センターにお問い合せください。

CSS-00146 failed to save scan result for table
原因： CSM$TABLES 表の行の更新に失敗しました。
処置： 内部エラーです。オラクル社カスタマ・サポート・センターにお問い合せください。

CSS-00147 unexpected data truncation
原因： フェッチ・バッファ用に列バイト・サイズと正確に同じサイズのメモリーが割り当てられました。その結果、予期しないデータ切捨てが行われました。
処置： 内部エラーです。オラクル社カスタマ・サポート・センターにお問い合せください。

CSS-00150 failed to enumerate table
原因： 指定した表情報の取得に失敗しました。
処置： 内部エラーです。オラクル社カスタマ・サポート・センターにお問い合せください。

CSS-00151 failed to enumerate user tables
原因： 指定したユーザーに属する表すべてを列挙できませんでした。
処置： 内部エラーです。オラクル社カスタマ・サポート・センターにお問い合せください。

CSS-00152 failed to enumerate all tables
原因： データベース内の表すべてを列挙できませんでした。
処置： 内部エラーです。オラクル社カスタマ・サポート・センターにお問い合せください。

CSS-00153 failed to enumerate character type columns
原因： スキャンする表の CHAR、VARCHAR2、LONG および CLOB 列すべてを列挙できませんでした。
処置： 内部エラーです。オラクル社カスタマ・サポート・センターにお問い合せください。

CSS-00154 failed to create list of tables to scan
原因： 表の CSM$TABLES 表への列挙に失敗しました。
処置： 内部エラーです。オラクル社カスタマ・サポート・センターにお問い合せください。

CSS-00155 failed to split tables for scan
原因： 指定した表の分割に失敗しました。
処置： 内部エラーです。オラクル社カスタマ・サポート・センターにお問い合せください。

CSS-00156 failed to get total number of tables to scan
原因： スキャンする表の数を取得する Select 文でエラーが発生しました。
処置： 内部エラーです。オラクル社カスタマ・サポート・センターにお問い合せください。

CSS-00157 failed to retrieve list of tables to scan
原因： 表 ID すべてをスキャナ・メモリーに読み込むことができませんでした。
処置： 内部エラーです。オラクル社カスタマ・サポート・センターにお問い合せください。

CSS-00158 failed to retrieve index defined on column
原因： 列に定義された索引を取得する Select 文でエラーが発生しました。
処置： 内部エラーです。オラクル社カスタマ・サポート・センターにお問い合せください。

CSS-00160 failed to open summary report file
原因： ファイル・オープン関数でエラーが戻されました。
処置： ディスクに対する作成 / 書込み権限があるかどうかをチェックし、LOG パラメータに指定したファイル名が有効であるかどうかをチェックしてください。

CSS-00161 failed to report scan elapsed time
原因： 不明です。
処置： 内部エラーです。オラクル社カスタマ・サポート・センターにお問い合せください。

CSS-00162 failed to report database size information
原因： 不明です。
処置： 内部エラーです。オラクル社カスタマ・サポート・センターにお問い合せください。

CSS-00163 failed to report scan parameters
原因： 不明です。
処置： 内部エラーです。オラクル社カスタマ・サポート・センターにお問い合せください。

CSS-00164 failed to report Scan summary
原因： 不明です。
処置： 内部エラーです。オラクル社カスタマ・サポート・センターにお問い合せください。

CSS-00165 failed to report conversion summary
原因： 不明です。
処置： 内部エラーです。オラクル社カスタマ・サポート・センターにお問い合せください。

CSS-00166 failed to report convertible data distribution
原因： 不明です。
処置： 内部エラーです。オラクル社カスタマ・サポート・センターにお問い合せください。

CSS-00167 failed to open exception report file
原因： ファイル・オープン関数でエラーが戻されました。
処置： ディスクに対する作成 / 書込み権限があるかどうかをチェックし、LOG パラメータに指定したファイル名が有効であるかどうかをチェックしてください。

CSS-00168 failed to report individual exceptions
原因： 不明です。
処置： 内部エラーです。オラクル社カスタマ・サポート・センターにお問い合せください。

CSS-00170 failed to retrieve size of tablespace %
原因： 不明です。
処置： 内部エラーです。オラクル社カスタマ・サポート・センターにお問い合せください。

CSS-00171 failed to retrieve free size of tablespace %s
原因： 不明です。
処置： 内部エラーです。オラクル社カスタマ・サポート・センターにお問い合せください。

CSS-00172 failed to retrieve total size of tablespace %s
原因： 不明です。
処置： 内部エラーです。オラクル社カスタマ・サポート・センターにお問い合せください。

CSS-00173 failed to retrieve used size of the database
原因： 不明です。
処置： 内部エラーです。オラクル社カスタマ・サポート・センターにお問い合せください。

CSS-00174 failed to retrieve free size of the database
原因： 不明です。
処置： 内部エラーです。オラクル社カスタマ・サポート・センターにお問い合せください。

CSS-00175 failed to retrieve total size of the database
原因： 不明です。
処置： 内部エラーです。オラクル社カスタマ・サポート・センターにお問い合せください。

CSS-00176 failed to enumerate user tables in bitmapped tablespace
原因： ローカル管理表領域での表の列挙に失敗しました。
処置： 内部エラーです。オラクル社カスタマ・サポート・センターにお問い合せください。

12

ロケール・データのカスタマイズ

この章では、ロケール・データのカスタマイズ方法について説明します。この章の内容は、次のとおりです。

Oracle Locale Builder ユーティリティの概要

Oracle Locale Builder は、ロケール・データをカスタマイズするための簡単で効率的な方法を提供します。また、ロケール固有のデータを簡単に表示、変更および定義できる GUI を提供します。Oracle Locale Builder は、テキストやバイナリの定義ファイルからデータを抽出し、読取り可能な書式で表します。したがって、これらのファイル内で使用されているフォーマットを気にせずに、情報を処理できます。

Oracle Locale Builder では、言語、地域、キャラクタ・セットおよび言語ソートの 4 種類のロケール定義を処理します。ユーザー定義文字やカスタマイズした言語規則もサポートします。既存のテキスト定義ファイルやバイナリ定義ファイルの定義を表示して、変更を加えたり、独自の定義を作成できます。

この項の内容は、次のとおりです。

Oracle Locale Builder 用の Unicode フォントの構成

Oracle Locale Builder では、多くの機能で Unicode 文字が使用されます。たとえば、Oracle Locale Builder では、Unicode コード・ポイントへのローカル文字コード・ポイントのマッピングが表示されます。したがって、Oracle Locale Builder を完全にサポートする Unicode フォントの使用をお薦めします。ローカル・フォントで文字を表現できない場合、その文字は空の四角として表示される可能性があります。

Windows でのフォント構成

Windows には、Unicode をサポートする多数の `TrueType` フォントおよび `OpenType` フォントがあります。オラクル社では、Microsoft 社の Arial Unicode MS フォントの使用をお薦めします。このフォントには、約 51,000 の絵文字が含まれ、Unicode 3.1 のほとんどの文字をサポートしています。

インストールした Unicode フォントは、Oracle Locale Builder で使用できるように、Java Runtime の `font.properties` ファイルに追加します。この `font.properties` ファイルは、`$JAVAHOME/lib` ディレクトリにあります。たとえば、Arial Unicode MS フォントの場合は、`font.properties` ファイルに次のエントリを追加します。

```
dialog.n=Arial Unicode MS, DEFAULT_CHARSET
dialoginput.n=Arial Unicode MS, DEFAULT_CHARSET
serif.n=Arial Unicode MS, DEFAULT_CHARSET
sansserif.n=Arial Unicode MS, DEFAULT_CHARSET
```

n は、フォント・リスト内で Arial Unicode MS フォントに割り当てるための次の順序番号です。Java Runtime は、各仮想フォントについてフォント・マッピング・リストを検索し、システムで使用可能な最初のフォントを使用します。

font.properties ファイルの編集後に、Oracle Locale Builder を再起動してください。

> **関連項目：** font.properties ファイルの詳細は、Sun 社の国際化 Web サイトを参照してください。

その他のプラットフォームでのフォント構成

Windows 以外のプラットフォーム用 Unicode フォントは、Windows プラットフォームよりも選択肢が限られています。文字を十分にカバーする Unicode フォントが見つからない場合は、異なる言語に複数のフォントを使用します。前述の Windows プラットフォームの場合の手順に従って、各フォントをインストールし、そのフォントのエントリを font.properties ファイルに追加してください。

たとえば、フォント ricoh-hg mincho を使用して Sun 社の Solaris で日本語の文字を表示するには、エントリを $JAVAHOME/lib にある既存の font.properties ファイルの dialog、dialoginput、serif および sansserif セクションに追加します。次に例を示します。

```
serif.plain.3=-ricoh-hg mincho l-medium-r-normal--*-%d-*-*-m-*-jisx0201.1976-0
```

> **関連項目：** 使用可能なフォントの詳細は、プラットフォーム固有のマニュアルを参照してください。

Oracle Locale Builder のユーザー・インタフェース

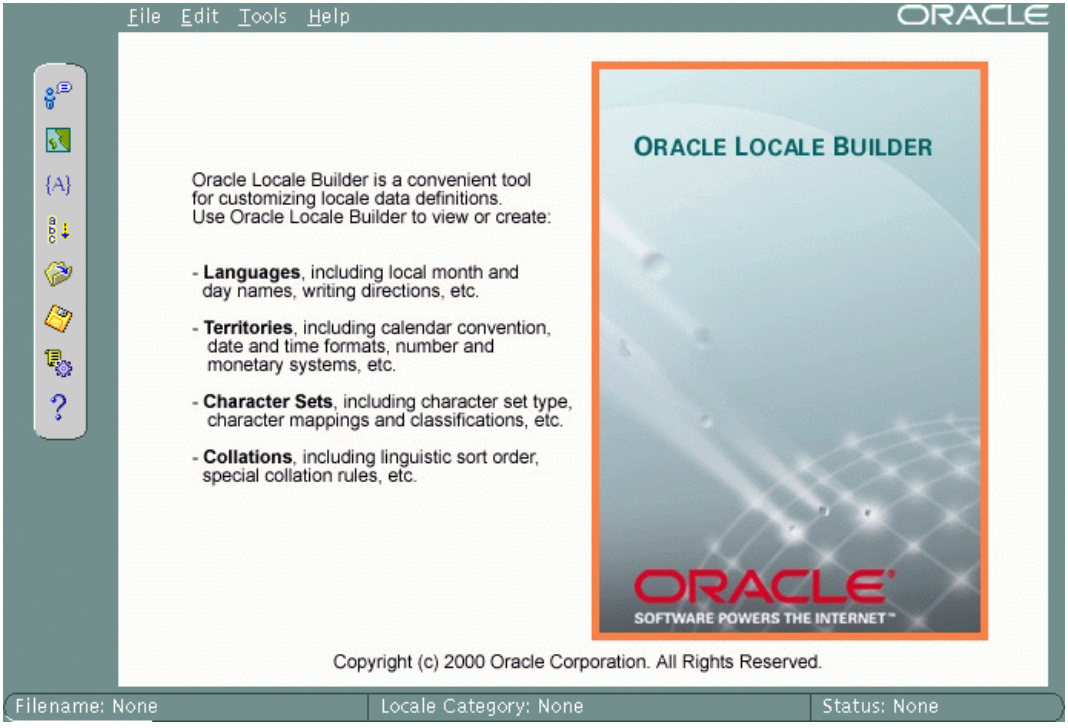
ビルダーを起動する前に、ORACLE_HOME 初期化パラメータが設定されていることを確認してください。

Oracle Locale Builder を起動するには、$ORACLE_HOME/ocommon/nls/lbuilder ディレクトリに変更して次のコマンドを発行します。

```
% lbuilder
```

Oracle Locale Builder を起動すると、図 12-1 の画面が表示されます。

図 12-1 Oracle Locale Builder ユーティリティ



Oracle Locale Builder の画面とダイアログ・ボックス

Oracle Locale Builder での特定のタスクを説明する前に、次のような画面とダイアログ・ボックスについて説明します。

- 「Existing Definitions」ダイアログ・ボックス
- 「Session Log」ダイアログ・ボックス
- 「Preview NLT」画面
- 「Open File」ダイアログ・ボックス

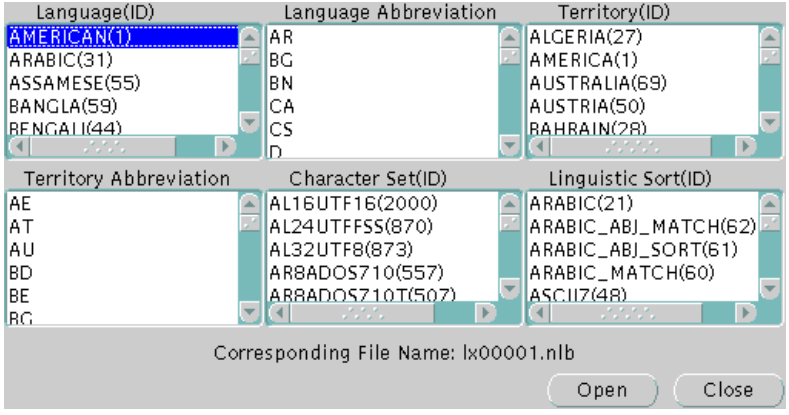
> **注意：** Oracle Locale Builder には、オンライン・ヘルプが付属しています。

「Existing Definitions」ダイアログ・ボックス

「New Language」、「New Territory」、「New Character Set」または「New Linguistic Sort」を選択すると、最初に「General」画面が表示されます。「Show Existing Definitions」をクリックすると、「Existing Definitions」ダイアログ・ボックスが表示されます。

「Existing Definitions」ダイアログ・ボックスでは、ロケール・オブジェクトを名前でオープンできます。起動する特定の言語、地域、言語ソート（照合）またはキャラクタ・セットがわかっている場合は、表示されている名前をクリックします。たとえば、図 12-2 では、AMERICAN 言語定義ファイルをオープンできます。

図 12-2「Existing Definitions」ダイアログ・ボックス



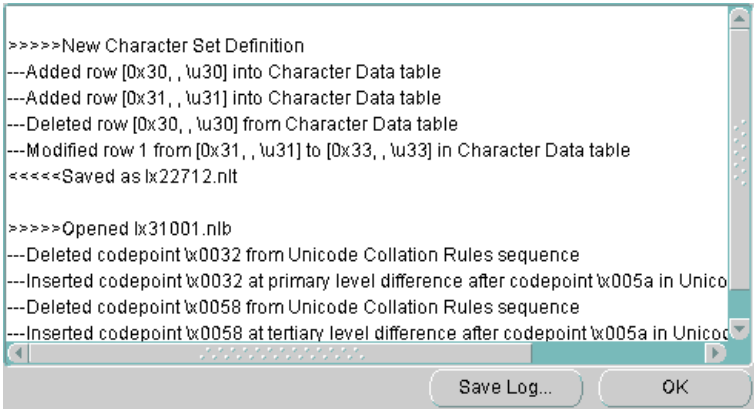
「AMERICAN」を選択すると、lx00001.nlb ファイルがオープンします。

言語と地域の略称は参照用のため、オープンできません。

「Session Log」ダイアログ・ボックス

「Tools」→「View Log」を選択すると、「Session Log」ダイアログ・ボックスが表示されます。「Session Log」ダイアログ・ボックスには、現行のセッションで行われた処理が表示されます。「Save Log」ボタンをクリックすると、すべての変更のレコードを保存できます。図 12-3 に、セッション・ログの例を示します。

図 12-3「Session Log」ダイアログ・ボックス

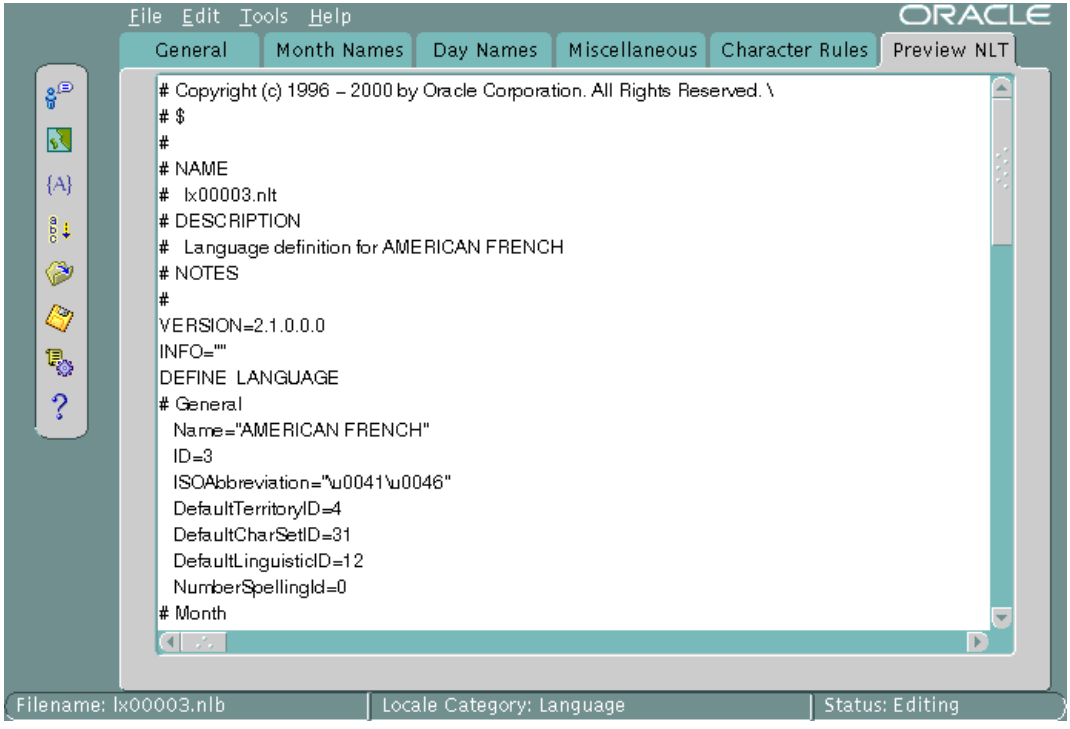


「Preview NLT」画面

NLT ファイルは、ファイル拡張子が .nlt のテキスト・ファイルです。この拡張子は、特定の言語、地域、キャラクタ・セットまたは言語ソートの設定を示します。「Preview NLT」画面には、ファイルが読取り可能な書式で表示されるため、変更内容が適切かどうかを確認できます。「Preview NLT」画面からは NLT ファイルを変更できません。NLT ファイルを変更するには、Oracle Locale Builder の特定の要素を使用する必要があります。

図 12-4 に、ユーザー定義言語 AMERICAN FRENCH の「Preview NLT」画面の例を示します。

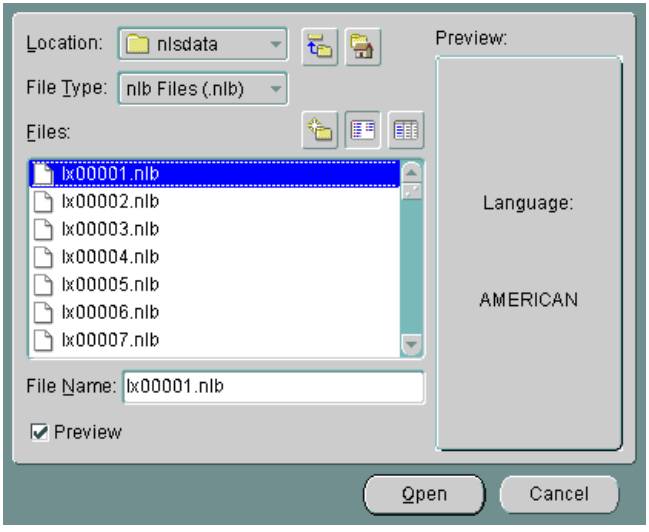
図 12-4 NLT ファイルのプレビュー



「Open File」ダイアログ・ボックス

「File」→「Open」→「By File Name」を選択すると、「Open File」ダイアログ・ボックスを表示できます。次に、変更する NLB ファイルまたはテンプレートとして使用する NLB ファイルを選択します。NLB ファイルは、ファイル拡張子が .nlb のバイナリ・ファイルで、その NLT ファイル内の情報に相当するバイナリが含まれています。図 12-5 に、lx00001.nlb ファイルが選択されている「Open File」ダイアログ・ボックスを示します。「Preview」パネルは、この NLB ファイルが AMERICAN 言語用であることを示しています。

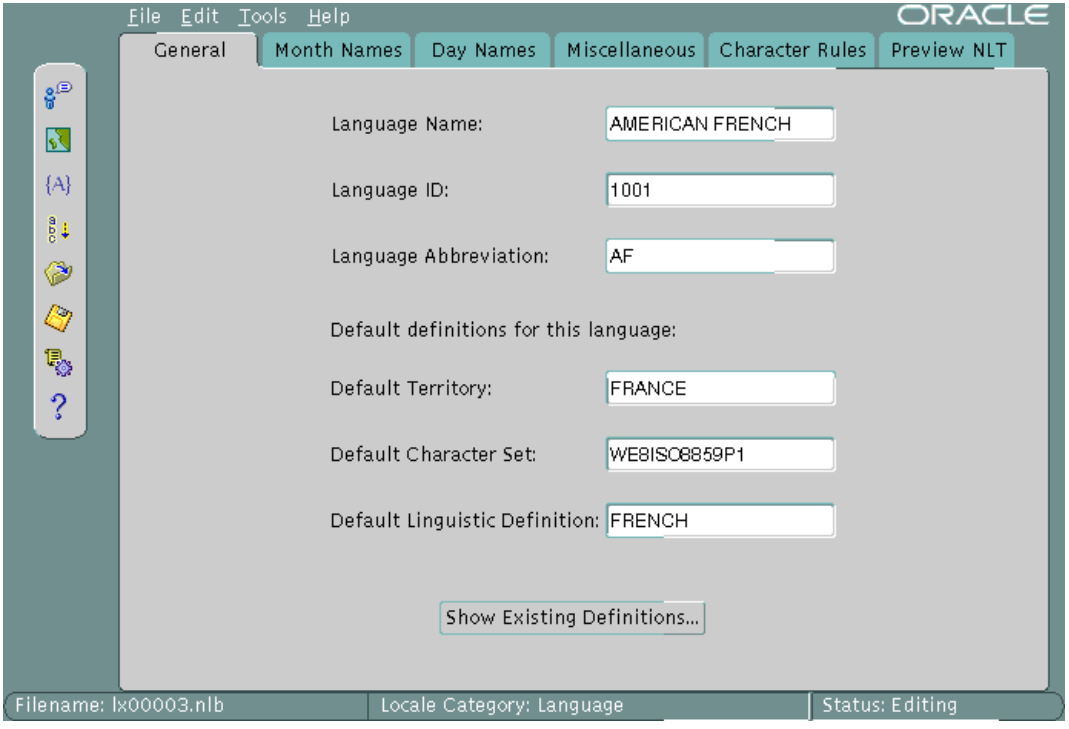
図 12-5「Open File」ダイアログ・ボックス



Oracle Locale Builder を使用した新規言語定義の作成

この項では、フランス語に基づいて新規言語を作成する方法について説明します。この新しい言語は、AMERICAN FRENCH と呼ぶことにします。最初に、「Existing Definitions」ダイアログ・ボックスから FRENCH をオープンします。次に、「General」ダイアログ・ボックスで言語名を AMERICAN FRENCH に変更し、「Language Abbreviation」を「AF」に変更します。他の設定はデフォルト値のままにします。図 12-6 に、設定後の「General」ダイアログ・ボックスを示します。

図 12-6 言語の一般情報



言語などのロケール・オブジェクトの名前を選択するときには、次の制限事項が適用されます。

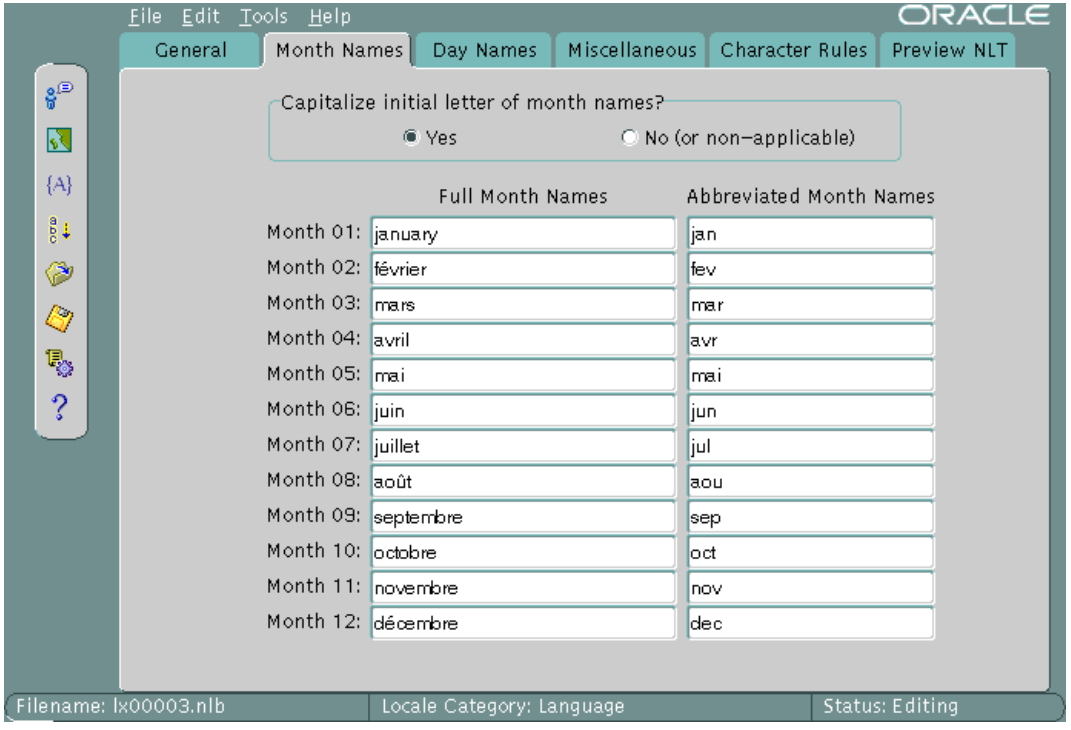
- 名前には ASCII 文字のみを使用する必要があります。
- 名前は文字で開始する必要があります。
- 言語、地域およびキャラクタ・セット名にはアンダースコアを使用できません。

ユーザー定義言語の場合、「Language ID」フィールドの有効範囲は 1,000 ～ 10,000 です。Oracle Locale Builder で提供される値を受け入れるか、この範囲内の値を指定できます。

注意： 特定の ID 範囲は、ユーザー定義の LANGUAGE、TERRITORY、CHARACTER SET、MONOLINGUAL COLLATION および MULTILINGUAL COLLATION の定義に対してのみ有効です。この章の各項では、各タイプのユーザー定義ロケール・オブジェクトに関係する範囲が指定されています。

図 12-7 に、「Month Names」タブを使用して月の名前を設定する方法を示します。

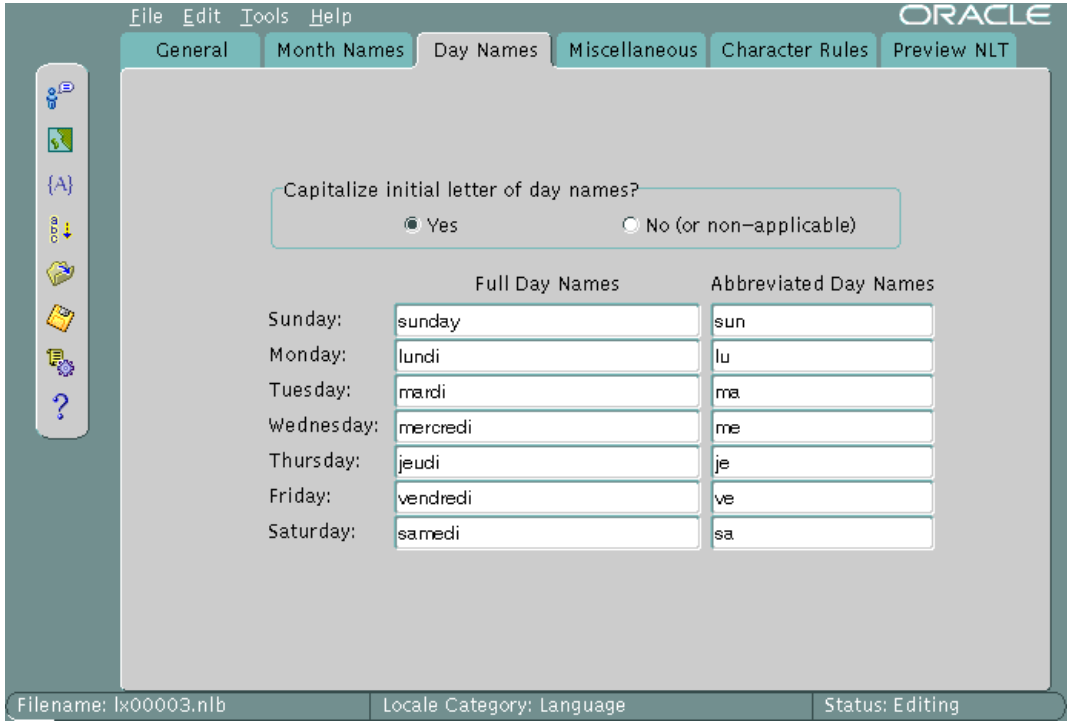
図 12-7 言語定義の月情報



すべての名前は、NLT ファイルに記述されているとおりに表示されます。「Capitalize initial letter of month names?」に「Yes」を選択すると、アプリケーションでは月の名前の 1 文字目に大文字が使用されますが、「Month Names」画面には大文字で表示されません。

図 12-8 に、「Day Names」画面を示します。

図 12-8 言語定義のタイプ情報



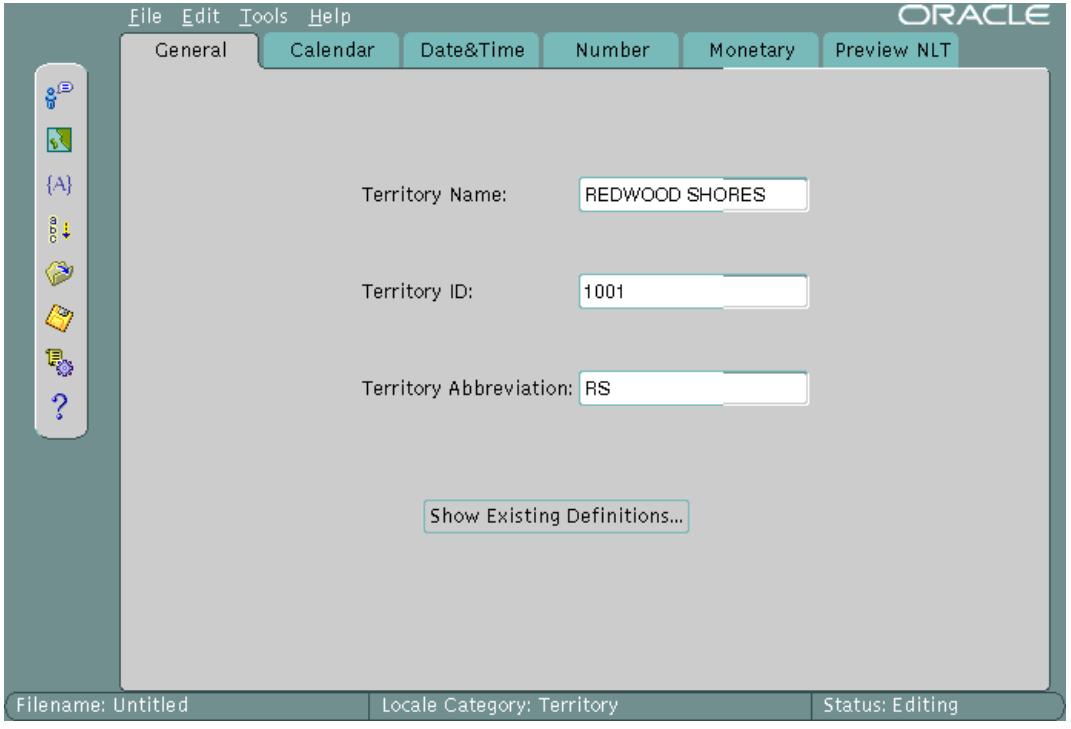
ユーザー定義言語用の曜日名を選択できます。すべての名前は、NLT ファイルに記述されているとおりに表示されます。「Capitalize initial letter of day names?」に「Yes」を選択すると、アプリケーションでは曜日名の 1 文字目に大文字が使用されますが、「Day Names」画面には大文字で表示されません。

Oracle Locale Builder を使用した新規地域定義の作成

この項では、新規の地域 REDWOOD SHORES を作成し、地域の略称として RS を使用する方法について説明します。新規の地域は、既存の地域定義に基づいていません。

基本タスクは、地域名を割り当てて、カレンダ、数値、日付、時刻および通貨の書式を選択することです。図 12-9 に、「Territory Name」として REDWOOD SHORES、「Territory ID」として 1001、「Territory Abbreviation」として RS が設定されている「General」画面を示します。

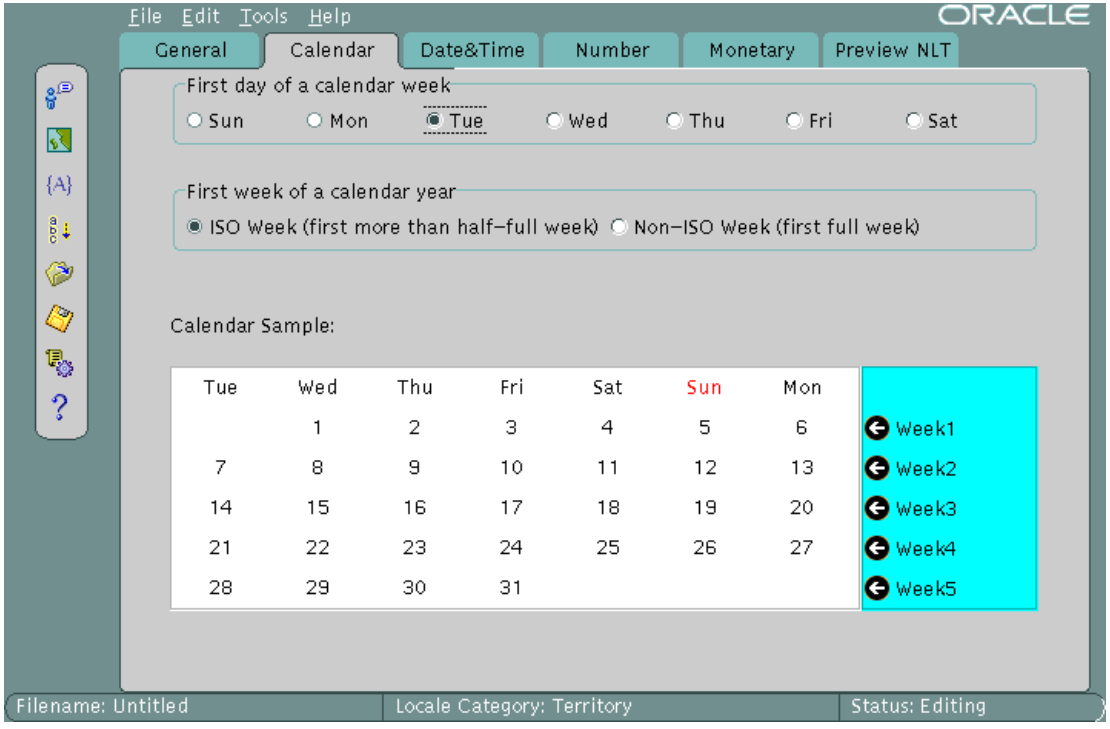
図 12-9 新規地域の定義



ユーザー定義地域に対する「Territory ID」フィールドの有効範囲は、1,000 ～ 10,000 です。

図 12-10 に、カレンダ書式の設定を示します。

図 12-10 カレンダ書式の選択



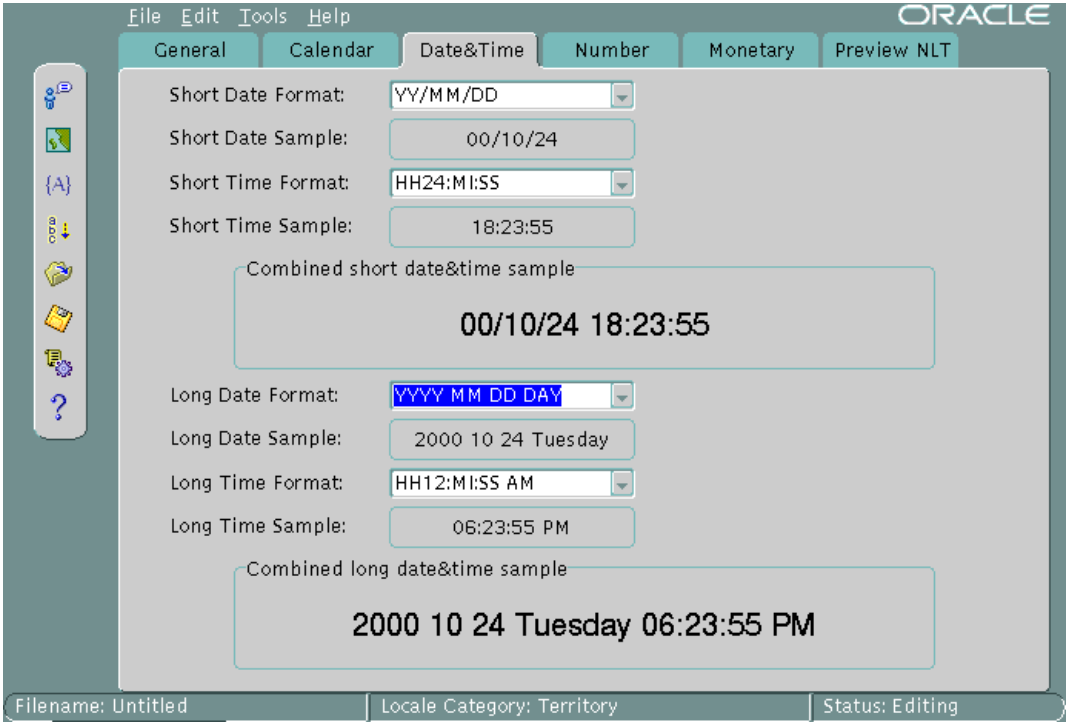
火曜日が週の開始日として設定され、暦年の第 1 週が ISO 週として設定されています。画面には、サンプル・カレンダが表示されます。

関連項目：

- 週の第 1 日と暦年の第 1 週を選択する方法の詳細は、3-25 ページの「カレンダ書式」を参照してください。
- カレンダ自体をカスタマイズする方法は、12-17 ページの「NLS カレンダ・ユーティリティを使用したカレンダのカスタマイズ」を参照してください。

図 12-11 に、日付と時刻の設定を示します。

図 12-11 日付書式と時刻書式の選択



ドロップダウン・メニューから設定を選択すると、サンプル・フォーマットが表示されます。この場合は、「Short Date Format」が `YY/MM/DD`、「Short Time Format」が `HH24:MI:SS`、「Long Date Format」が `YYYY/MM/DD DAY`、「Long Time Format」が `HH12:MI:SS AM` に設定されています。

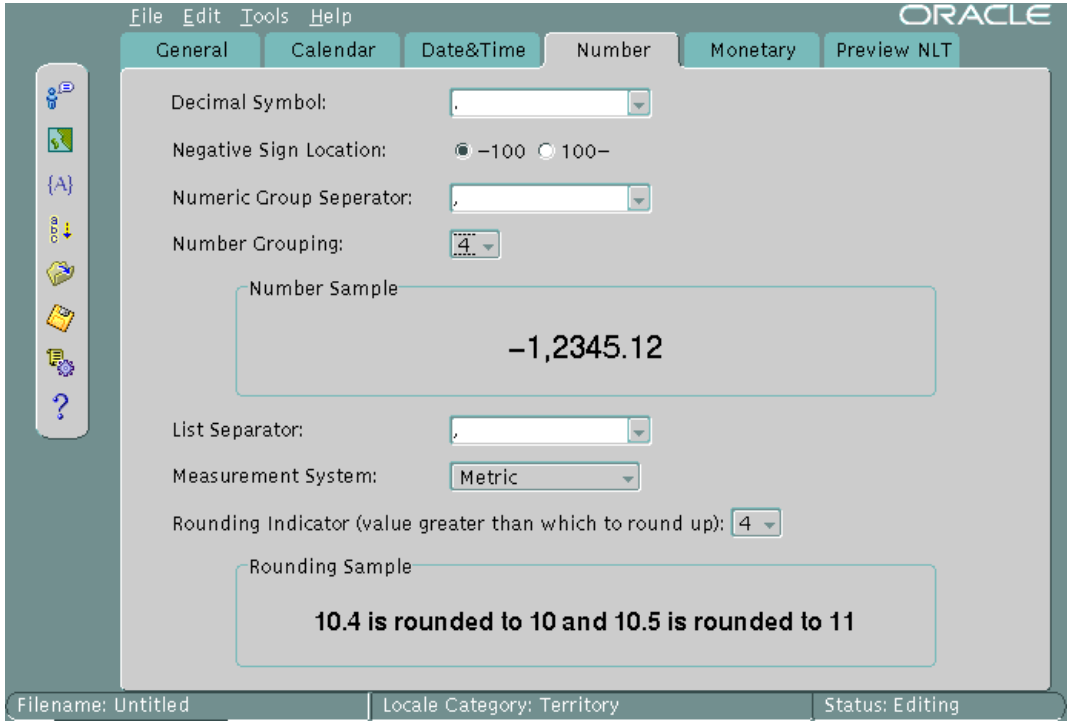
ドロップダウン・メニューから選択せずに、独自のフォーマットを入力することもできます。

関連項目：

- 3-17 ページ「日付書式」
- 3-21 ページ「時刻書式」
- 12-17 ページ「タイム・ゾーン・データのカスタマイズ」

図 12-12 に、数値書式の設定を示します。

図 12-12 数値書式の選択



「Decimal Symbol」にピリオドが選択されています。「Negative Sign Location」は、数値の左に設定されており、「Numeric Group Separator」はカンマです。「Number Grouping」は 4 桁、「List Separator」はカンマ、「Measurement System」は Metric、「Rounding Indicator」は 4 に設定されています。

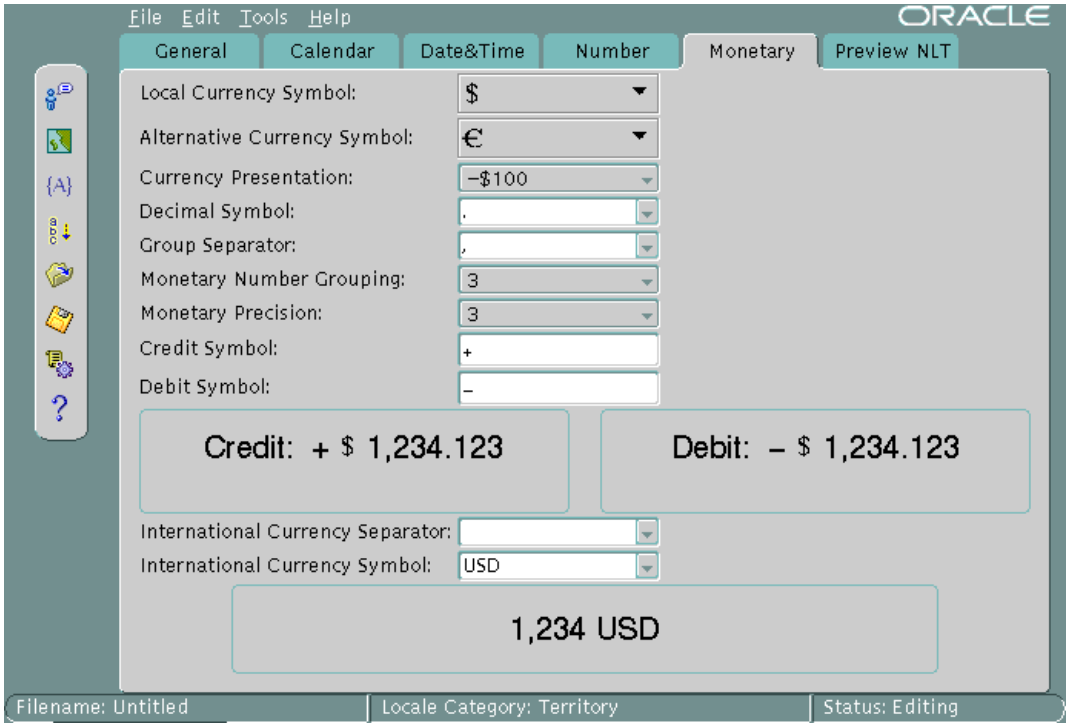
ドロップダウン・メニューを使用せずに、独自の値を入力することもできます。

ドロップダウン・メニューから設定を選択すると、サンプル・フォーマットが表示されます。

関連項目： 3-29 ページ「数値書式」

図 12-13 に、「Monetary」ダイアログ・ボックスでの通貨書式の設定を示します。

図 12-13 通貨書式の選択



「Local Currency Symbol」は $、「Alternative Currency Symbol」はユーロ記号に設定されています。「Currency Presentation」は、各国通貨記号、借方記号および番号に可能な複数の順序の 1 つを示しています。「Decimal Symbol」はピリオド、「Group Separator」はカンマ、「Monetary Number Grouping」は 3 です。「Monetary Precision」、つまり小数点以下の桁数は 3 です。「Credit Symbol」は +、「Debit Symbol」は - です。「International Currency Separator」は空白であるため、画面には表示されていません。「International Currency Symbol」（ISO 通貨記号）は USD です。選択した値に基づいて、サンプルの通貨書式が表示されます。

ドロップダウン・メニューを使用せずに、独自の値を入力することもできます。

関連項目： 3-31 ページ「通貨書式」

これ以降の内容は、次のとおりです。

- タイム・ゾーン・データのカスタマイズ
- NLS カレンダ・ユーティリティを使用したカレンダのカスタマイズ

タイム・ゾーン・データのカスタマイズ

タイム・ゾーン・ファイルには、有効なタイム・ゾーン名が含まれています。各タイム・ゾーンには、次の情報が含まれています。

- 協定世界時（UTC）からのオフセット。
- 夏時間への移行時間。
- 標準時間と夏時間の略称。略称は、タイム・ゾーン名とともに使用されます。

Oracle ホーム・ディレクトリには、2 つのタイム・ゾーン・ファイルがあります。デフォルト・ファイルは oracore/zoneinfo/timezone.dat で、最も一般的なタイム・ゾーンが含まれています。より大きいタイム・ゾーン・セットは、oracore/zoneinfo/timezlrg.dat に含まれています。大きいタイム・ゾーン・セットが必要でないかぎり、デフォルトのタイム・ゾーン・ファイルを使用してください。これにより、データベースのパフォーマンスが向上します。

大きい方のタイム・ゾーン・ファイルを使用する手順は、次のとおりです。

1. データベースをシャットダウンします。
2. 環境変数 ORA_TZFILE を timezlrg.dat ファイルのフルパス名に設定します。
3. データベースを再起動します。

timezlrg.dat ファイルを使用した後は、他のタイム・ゾーンのいずれもデータベースに格納されているデータに使用されていないことを確認しないかぎり、この大きなファイルを継続して使用する必要があります。また、情報を共有しているすべてのデータベースは、同じタイム・ゾーン・ファイルを使用する必要があります。

タイム・ゾーン名を表示するには、次の文を入力します。

```
SQL> SELECT * FROM V$TIMEZONE_NAMES;
```

NLS カレンダ・ユーティリティを使用したカレンダのカスタマイズ

Oracle は、複数のカレンダをサポートしています。これらのカレンダはすべて Oracle のグローバリゼーション・サポートから導出されるデータで定義されていますが、一部は、将来、元号やうるう年の追加が必要になる場合があります。Oracle データベース・サーバーの新リリースまで待たずにこの情報を追加するには、カレンダ機能の実行時に自動的にロードされる外部ファイルを使用できます。

カレンダのデータは、最初にテキスト・ファイルに定義します。このテキスト定義ファイルは、バイナリ形式に変換する必要があります。NLS カレンダ・ユーティリティ（lxegen）を使用すると、テキスト定義ファイルをバイナリ形式に変換できます。

テキスト定義ファイルの名前と位置は、プラットフォーム依存の値でハードコード化されています。UNIX プラットフォームの場合、ファイル名は lxecal.nlt で、$ORACLE_HOME/ocommon/nls ディレクトリにあります。テキスト定義ファイルのサンプルは、このディレクトリに含まれています。

lxegen ユーティリティでは、テキスト定義ファイルからバイナリ・ファイルが生成されます。バイナリ・ファイル名もプラットフォーム依存の値でハードコード化されています。UNIX プラットフォームの場合、バイナリ・ファイル名は lxecal.nlb です。このバイナリ・ファイルは、テキスト・ファイルと同じディレクトリに生成され、既存のバイナリ・ファイルが上書きされます。

バイナリ・ファイルが生成されると、そのファイルはシステムの初期化時に自動的にロードされます。このファイルを移動したり名前を変更しないでください。

カレンダ・ユーティリティを起動するには、コマンドラインから次のように入力します。

```
% lxegen
```

関連項目：

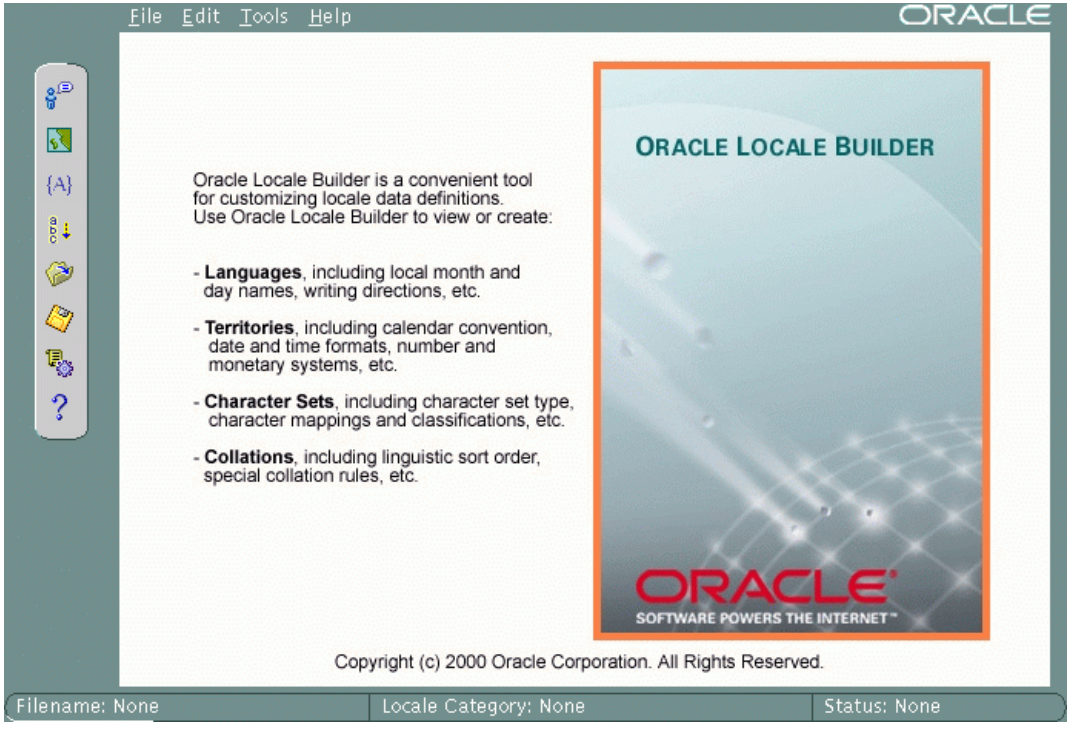
- システム上のファイルの位置については、プラットフォーム固有のマニュアルを参照してください。
- A-27 ページ「暦法」

Oracle Locale Builder を使用したコード・チャートの表示

Oracle Locale Builder では、キャラクタ・セットのコード・チャートを表示して印刷できます。

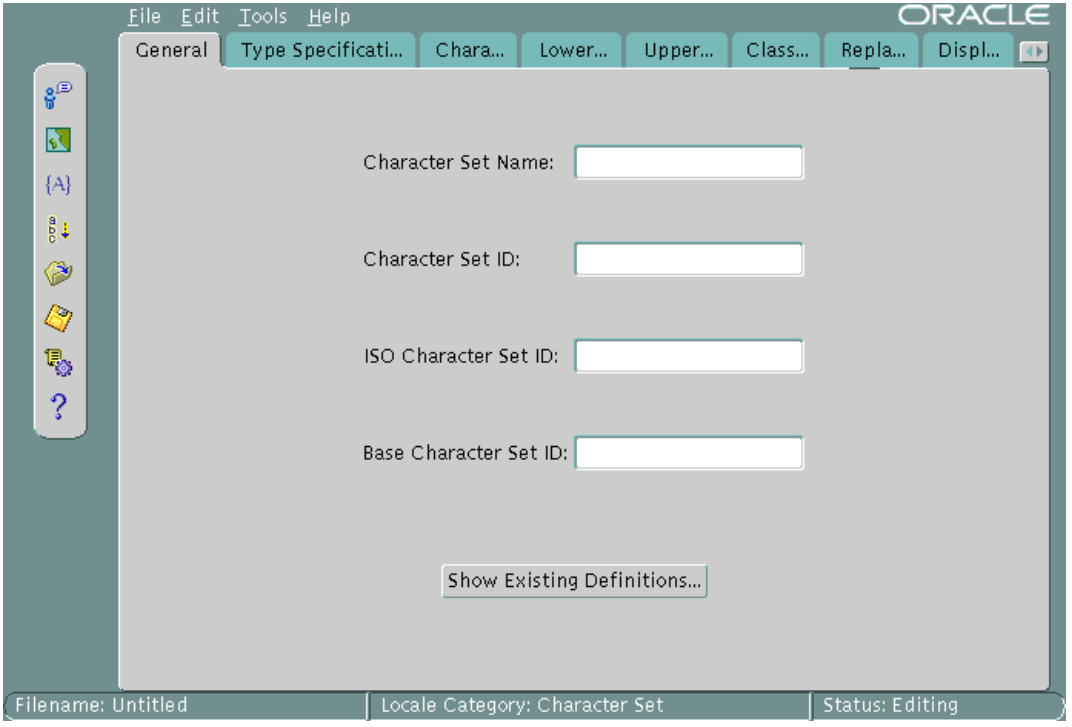
図 12-14 に、Oracle Locale Builder の初期画面を示します。

図 12-14 Oracle Locale Builder の初期画面



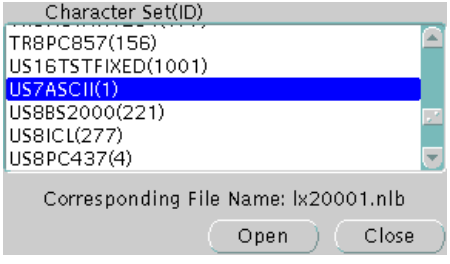
「File」→「New」を選択します。「New」→「Character Set」を選択します。図 12-15 に、表示される画面を示します。

図 12-15「General Character Set」画面



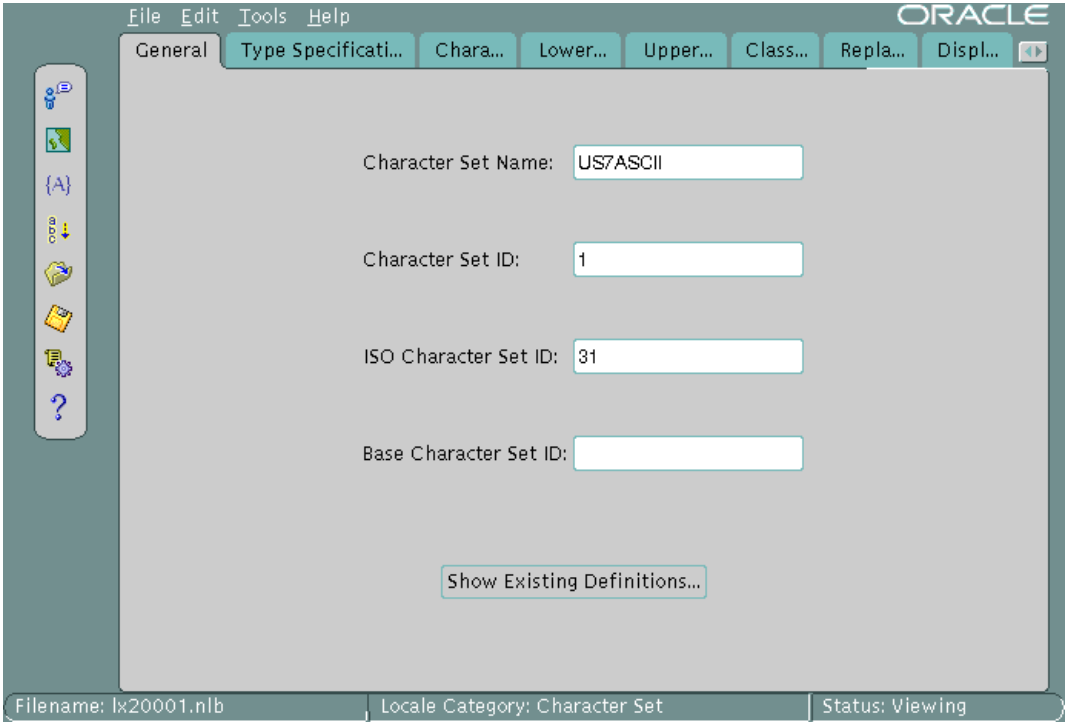
「Show Existing Definitions」をクリックします。表示するキャラクタ・セットを選択します。図 12-16 に、US7ASCII が選択されている「Existing Definitions」ダイアログ・ボックスを示します。

図 12-16「Existing Definitions」ダイアログ・ボックスでの US7ASCII の選択



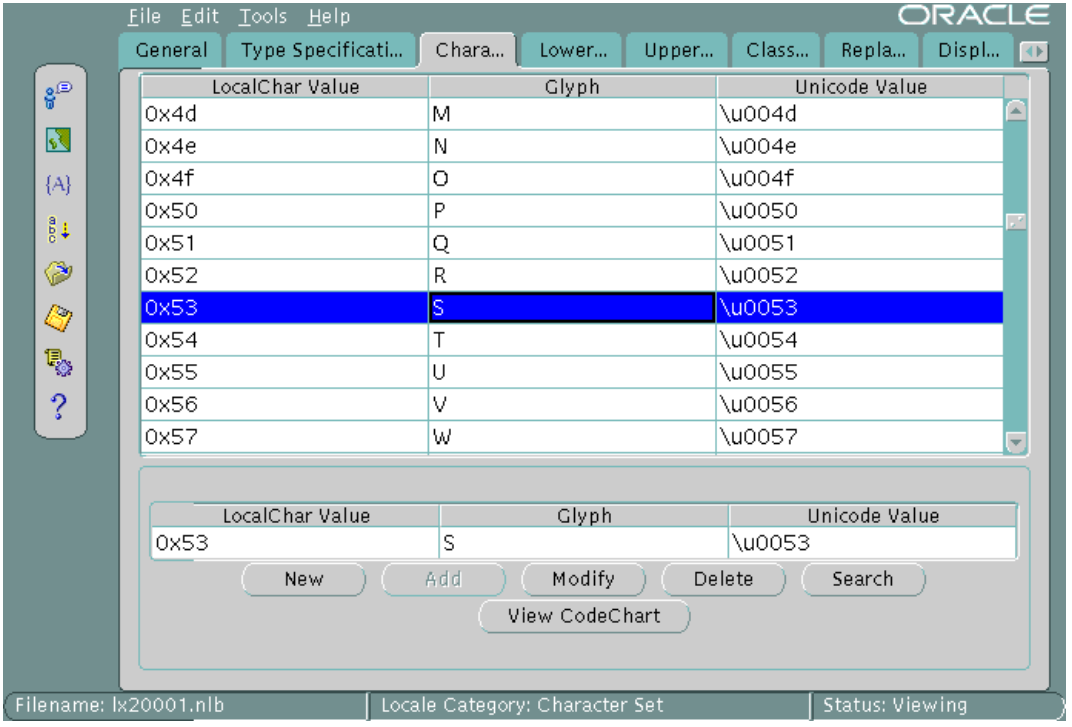
「Open」をクリックしてキャラクタ・セットを選択します。図 12-17 に、US7ASCII が選択されている「General」画面を示します。

図 12-17 US7ASCII がロードされている場合の「General」画面



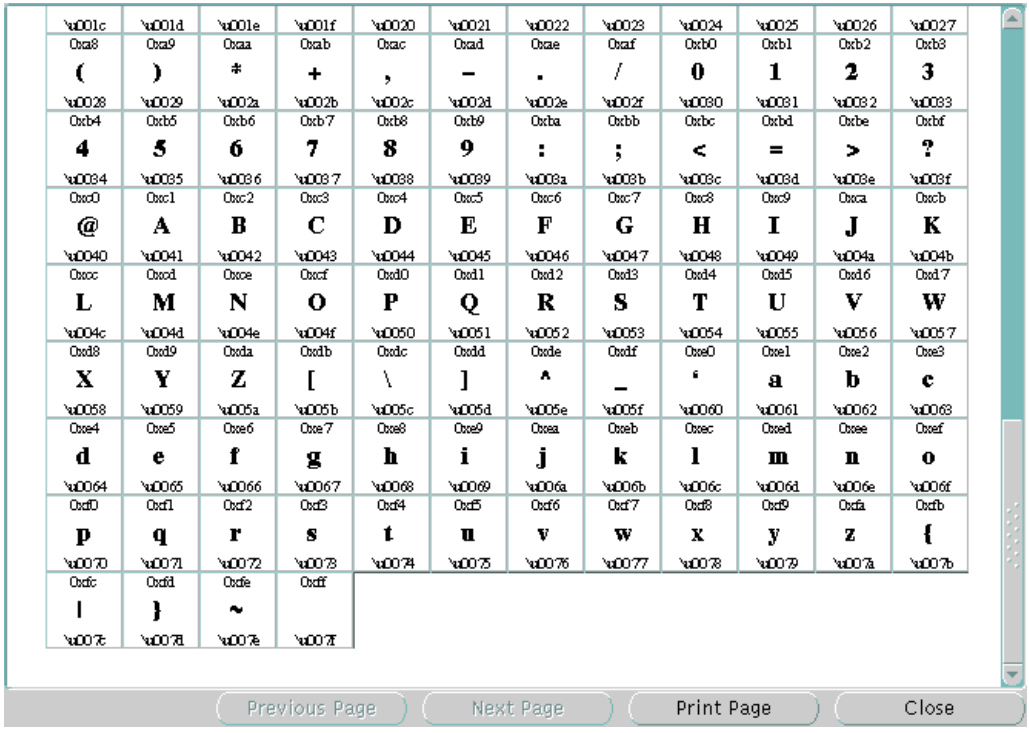
「Character Data Mapping」タブをクリックします。図 12-18 に、US7ASCII の「Character Data Mapping」画面を示します。

図 12-18 US7ASCII の「Character Data Mapping」画面



「View CodeChart」をクリックします。図 12-19 に、US7ASCII のコード・チャートを示します。

図 12-19 US7ASCII コード・チャート



このコード・チャートは、ローカル・キャラクタ・セットでの各文字のエンコーディング値、各文字に関連付けられた絵文字、および Unicode 値を示しています。

コード・チャートを印刷する場合は、「Print Page」をクリックします。

Oracle Locale Builder を使用した新規キャラクタ・セット定義の作成

特定のユーザーのニーズにあわせてキャラクタ・セットをカスタマイズできます。Oracle9*i* では、エンコードされた既存のキャラクタ・セットの定義を拡張できます。ユーザー定義文字は、多くの場合、次のような文字を表現する特殊文字をエンコードするために使用されます。

- 固有名
- 既存のキャラクタ・セット規格で定義されていない旧漢字
- ベンダー固有の文字
- ユーザーが新たに定義した記号または文字

この項では、Oracle によるユーザー定義文字のサポート方法を説明します。この項の内容は、次のとおりです。

マルチバイト・キャラクタ・セットに新規キャラクタ・セット定義を作成する場合は、既存の 8 ビットまたはマルチバイトのキャラクタ・セットをベースにしてください。7 ビットのキャラクタ・セットは、マルチバイト・キャラクタ・セットの分類検証が適用されないため、ベースにしないでください。

ユーザー定義文字（UDC）とキャラクタ・セット

一般的に、ユーザー定義文字は東アジア諸国のキャラクタ・セットでサポートされています。東アジア諸国のキャラクタ・セットには、ユーザー定義文字を使用するために、コード・ポイントの予約域が少なくとも 1 つ用意されています。たとえば、日本のシフト JIS では、表 12-1 に示す 1880 のコード・ポイントがユーザー定義文字用に確保されています。

表 12-1 シフト JIS のユーザー定義文字の範囲

日本語シフト JIS のユーザー定義文字の範囲	コード・ポイント数
F040-F07E、F080-F0FC	188
F140-F17E、F180-F1FC	188
F240-F27E、F280-F2FC	188
F340-F37E、F380-F3FC	188
F440-F47E、F480-F4FC	188
F540-F57E、F580-F5FC	188
FF640-F67E、F680-F6FC	188
F740-F77E、F780-F7FC	188
F840-F87E、F880-F8FC	188
F940-F97E、F980-F9FC	188

表 12-2 に示す Oracle キャラクタ・セットには、ユーザー定義文字をサポートする事前定義済みの範囲が含まれています。

表 12-2 ユーザー定義文字の範囲を含む Oracle キャラクタ・セット

キャラクタ・セット名	ユーザー定義文字に使用可能なコード・ポイント数
JA16DBCS	4370
JA16EBCDIC930	4370
JA16SJIS	1880
JA16SJISYEN	1880
KO16DBCS	1880
KO16MSWIN949	1880
ZHS16DBCS	1880
ZHS16GBK	2149

表 12-2 ユーザー定義文字の範囲を含む Oracle キャラクタ・セット （続き）

キャラクタ・セット名	ユーザー定義文字に使用可能なコード・ポイント数
ZHT16DBCS	6204
ZHT16MSWIN950	6217

Oracle のキャラクタ・セット変換アーキテクチャ

特定の文字を表すコード・ポイントの値は、キャラクタ・セットによって異なります。図 12-20 に、日本語の漢字を示します。

図 12-20 日本語の漢字

亜

次の表に、この文字が様々なキャラクタ・セットでどのようにエンコードされるかを示します。

Unicode エンコーディング	JA16SJIS エンコーディング	JA16EUC エンコーディング	JA16DBCS エンコーディング
4E9C	889F	B0A1	4867

Oracle では、すべてのキャラクタ・セットが Unicode 3.1 のコード・ポイントの見地から定義されています。つまり、それぞれの文字は Unicode 3.1 のコード値として定義されています。文字変換は、中間フォームとして Unicode を使用して行われ、ユーザーが意識することはありません。たとえば、JA16SJIS のクライアントが JA16EUC のデータベースに接続する場合、JA16SJIS のクライアントから入力した図 12-20 の文字のコード・ポイント値は 889F となります。この文字は内部的に Unicode（コード・ポイント値は 4E9C）に変換された後、JA16EU（コード・ポイント値は B0A1）に変換されます。

Unicode 3.1 の Private Use Area

Unicode 3.1 では、Private Use Area（PUA）用に E000 ～ F8FF の範囲が予約されています。PUA は、エンド・ユーザーまたはベンダーがプライベートで使用する文字を定義するためのものです。

ユーザー定義文字は、標準文字と同様に中間フォームとして Unicode 3.1 の PUA を使用して、2 つの Oracle キャラクタ・セット間で変換されます。

キャラクタ・セット間でのユーザー定義文字のクロス・リファレンス

日本語キャラクタ・セット、韓国語キャラクタ・セット、簡体字中国語キャラクタ・セットおよび繁体字中国語キャラクタ・セットでのユーザー定義文字クロス・リファレンスは、次の配布セットに含まれています。

```
$ORACLE_HOME/ocommon/nls/demo/udc_ja.txt
$ORACLE_HOME/ocommon/nls/demo/udc_ko.txt
$ORACLE_HOME/ocommon/nls/demo/udc_zhs.txt
$ORACLE_HOME/ocommon/nls/demo/udc_zht.txt
```

ユーザー定義文字を複数のオペレーティング・システムに登録する場合、これらのクロス・リファレンスが役に立ちます。たとえば、新規ユーザー定義文字を日本語シフト JIS のオペレーティング・システムと日本語 IBM ホスト・オペレーティング・システムの両方に登録する場合、この新規ユーザー定義文字に対して、シフト JIS のオペレーティング・システムでは F040、IBM ホスト・オペレーティング・システムでは 6941 を使用できます。これによって、JA16SJIS と JA16DBCS の間での変換が正しく行われます。シフト JIS の UDC 値 F040 と IBM ホストの UDC 値 6941 は、ユーザー定義文字クロス・リファレンスでは、同じ Unicode PUA 値 E000 にマップされています。

関連項目： 付録 B「Unicode 文字コードの割当て」

既存のキャラクタ・セットから新規キャラクタ・セットを作成する際のガイドライン

デフォルトの場合、Oracle Locale Builder は次に使用可能なキャラクタ・セット名を生成します。独自のキャラクタ・セット名を生成することもできます。キャラクタ・セット定義用 NLT ファイルのネーミングには、次の書式を使用します。

```
lx2dddd.nlt
```

dddd は、16 進数で 4 桁のキャラクタ・セット ID です。

キャラクタ・セットを変更する場合は、次のガイドラインに従ってください。

- 既存の文字を再マップしないでください。
- すべての文字のマッピングは一意であることが必要です。
- 新規文字は、Unicode のプライベート使用範囲（e000 ～ f4ff）にマップする必要があります。（実際の Unicode 3.1 のプライベート使用範囲は e000 ～ f8ff ですが、f500 ～ f8ff は Oracle 固有のプライベート使用のために予約されています。）
- キャラクタ・セット定義ファイルでは、1 行の最大文字数は 80 文字です。

既存の Oracle キャラクタ・セットから導出されたキャラクタ・セットの場合は、次のキャラクタ・セットのネーミング規則を使用することをお薦めします。

```
<Oracle_character_set_name><organization_name>EXT<version>
```

たとえば、Sun 社などの会社が JA16EUC キャラクタ・セットにユーザー定義文字を追加する場合、適切なキャラクタ・セット名は次のようになります。

```
JA16EUCSUNWEXT1
```

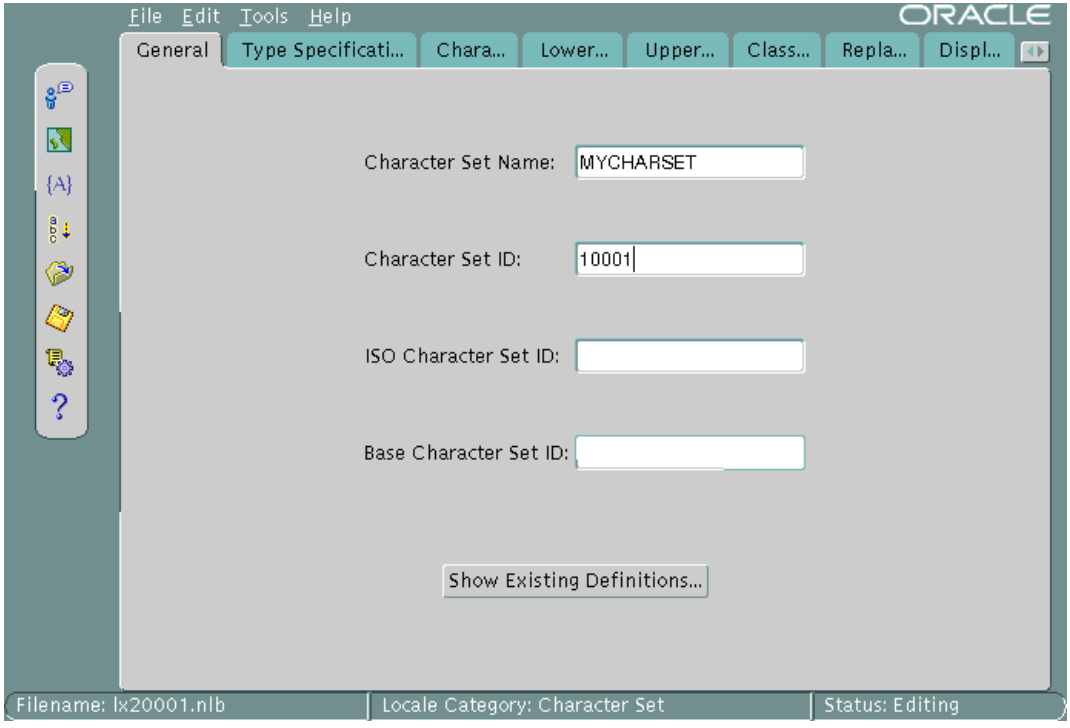
このキャラクタ・セット名は、次の各部分で構成されています。

- JA16EUC は、Oracle が定義したキャラクタ・セット名です。
- SUNW は、会社名（Sun 社の株式取引上の略称）を表します。
- EXT は、JA16EUC キャラクタ・セットに対する拡張要素であることを示します。
- 1 はバージョンを示します。

例 : Oracle Locale Builder を使用した新規キャラクタ・セット定義の作成

この項では、キャラクタ・セット ID に 10001 を指定して、新規のキャラクタ・セット MYCHARSET を作成する方法について説明します。この例では US7ASCII キャラクタ・セットから始めて中国語の 10 文字を追加します。図 12-21 に、「General」画面を示します。

図 12-21 キャラクタ・セットの一般情報

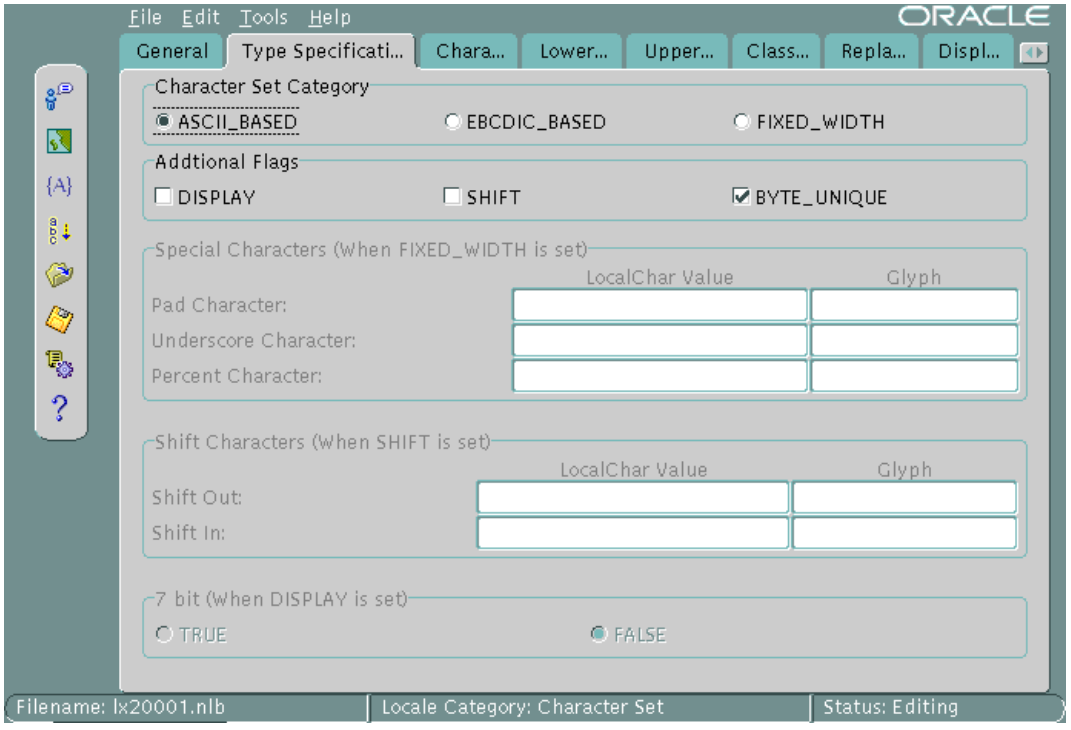


「Show Existing Definitions」をクリックし、「Existing Definitions」ダイアログ・ボックスから US7ASCII キャラクタ・セットを選択します。

「ISO Character Set ID」フィールドと「Base Character Set ID」フィールドはオプションです。「Base Character Set ID」は値の継承に使用され、基本キャラクタ・セットのプロパティがテンプレートとして使用されます。「Character Set ID」は自動的に生成されますが、オーバーライドして変更することもできます。ユーザー定義キャラクタ・セット ID の有効範囲は、10,000 ～ 20,000 です。ユーザー定義キャラクタ・セットの場合、「ISO Character Set ID」フィールドは空白のままです。

図 12-22 に、「Type Specification」画面を示します。

図 12-22 キャラクタ・セットの型指定



「Character Set Category」は ASCII_BASED で、「BYTE_UNIQUE」フラグがオンになっています。

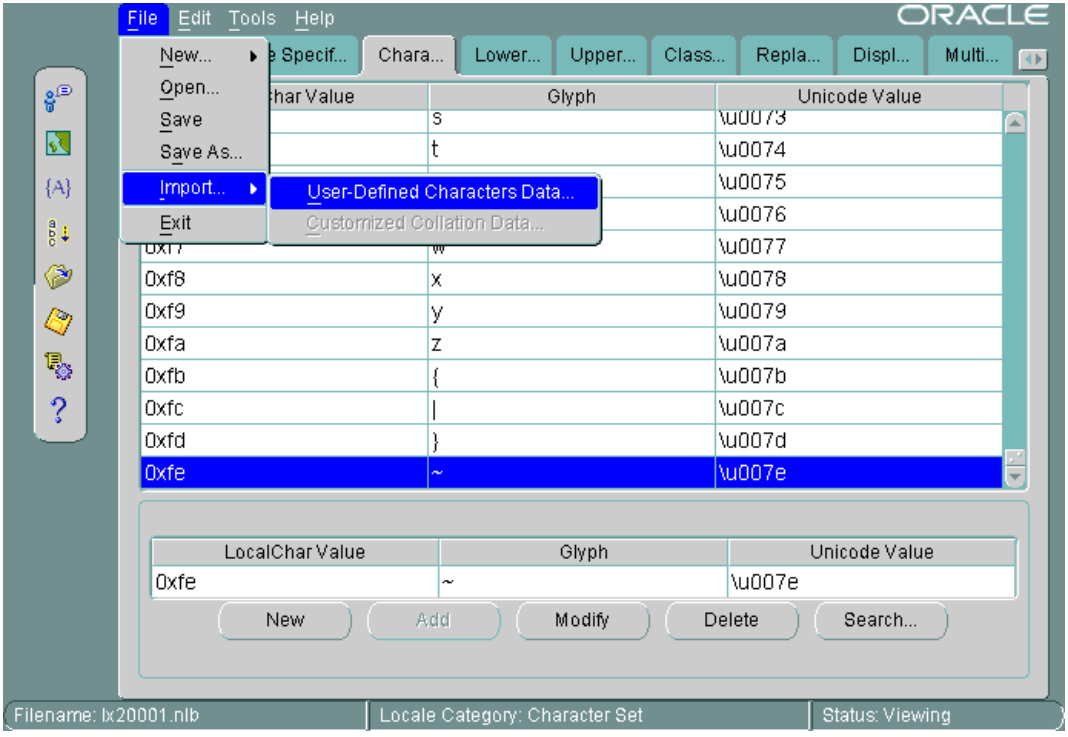
既存のキャラクタ・セットを選択した場合、「Type Specification」画面のフィールドはすでに適切な値に設定されています。変更する特別な理由がないかぎり、これらの値を保持してください。設定を変更する必要がある場合は、次のガイドラインを使用します。

- 「FIXED_WIDTH」は、同じ長さの文字のキャラクタ・セットを示します。
- 「BYTE_UNIQUE」は、コード・ポイントのシングルバイト範囲がマルチバイト範囲と区別されていることを意味します。先頭バイトのコードは、その文字がシングルバイトかマルチバイトかを示します（JA16EUC など）。
- 「DISPLAY」は、格納用ではなくクライアントでの表示用にのみ使用されるキャラクタ・セットを示します。一部のアラビア語、デーバナーガリ語およびヘブライ語のキャラクタ・セットは、表示用キャラクタ・セットです。
- 「SHIFT」は、シングルバイト文字とマルチバイト文字とを区別するために、追加のシフト文字が必要なキャラクタ・セットに対して指定します。

関連項目： シフトイン・キャラクタ・セットとシフトアウト・キャラクタ・セットの詳細は、2-10 ページの「可変幅マルチバイト・コード体系」を参照してください。

図 12-23 に、ユーザー定義文字の追加方法を示します。

図 12-23 ユーザー定義文字データのインポート



「Character Data Mapping」画面をオープンします。キャラクタ・セット内で追加する文字の前になる文字を選択します。この例では、`0xfe` ローカル文字の値が選択されています。

1 回に 1 文字を追加することも、テキスト・ファイルを使用して大量の文字をインポートすることも可能です。この例では、テキスト・ファイルをインポートしています。最初の列は、ローカル文字の値です。2 列目は Unicode 値です。このファイルには、次の文字の値が含まれています。

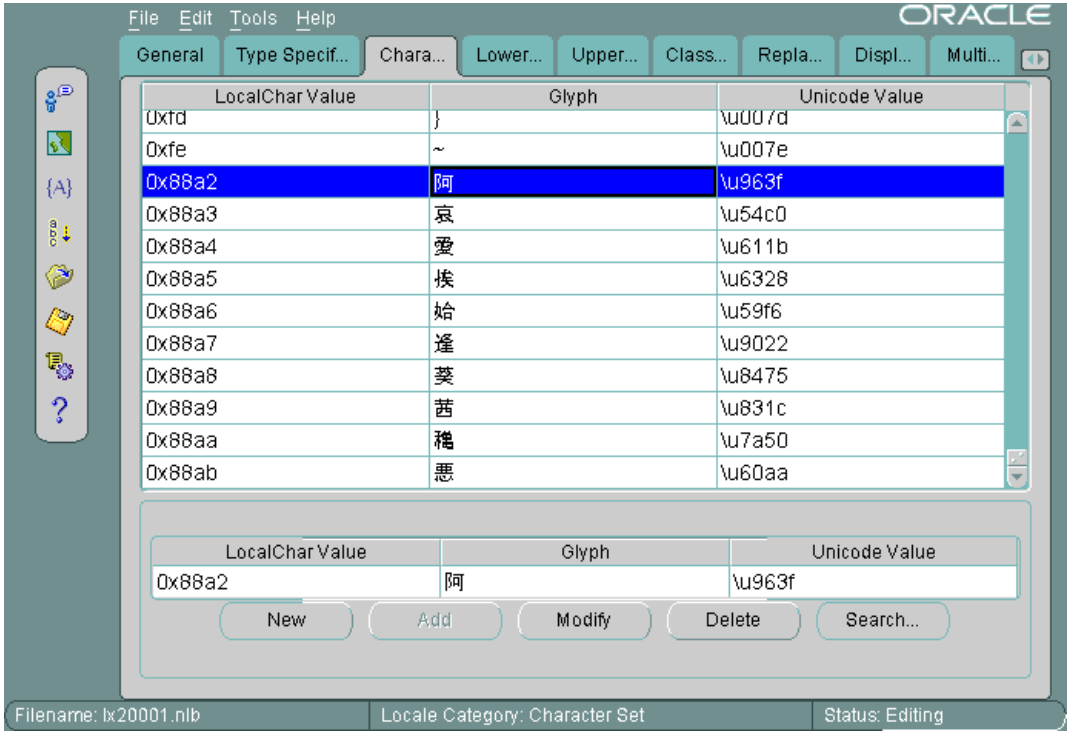
```
88a2    963f
88a3    54c0
88a4    611b
```

88a5　6328
88a6　59f6
88a7　9022
88a8　8475
88a9　831c
88aa　7a50
88ab　60aa

「File」→「Import User-Defined Customers Data」を選択します。

図 12-24 に、インポートした文字をキャラクタ・セットの `0xfe` の後に追加した状態を示します。

図 12-24 キャラクタ・セットでの新しい文字



Java でのユーザー定義文字のサポート

アプリケーションで JDBC や SQLJ などの Java 製品を使用しており、ユーザー定義文字をサポートする必要がある場合は、必要に応じてキャラクタ・セットをカスタマイズします。その後、特殊な Java zip ファイル（gss_custom.zip）を生成し、Oracle ホーム・ディレクトリにインストールします。

UNIX の場合は、次のようなコマンドを入力します。

```
$ORACLE_HOME/JRE/bin/jre -classpath $ORACLE_HOME/jlib/gss-1_1.zip:
    $ORACLE_HOME/jlib/gss_charset-1_2.zip Ginstall lx22710.nlt
```

Windows の場合は、次のようなコマンドを入力します。

```
%JREHOME%¥bin¥jre.exe -classpath %ORACLE_HOME%¥jlib¥gss-1_1.zip:
    %ORACLE_HOME%¥jlib¥gss_charset-1_2.zip Ginstall lx22710.nlt
```

%JREHOME% は、C:¥Program Files¥Oracle¥jre¥version_num ディレクトリです。

lx22710.nlt は、Oracle Locale Builder を使用してキャラクタ・セットをカスタマイズすることで作成された NLT ファイルの例です。

前述のコマンドによって、gss_custom.zip ファイルがカレント・ディレクトリに生成されます。カスタマイズした複数のキャラクタ・セットに対するサポートを追加する必要がある場合は、カスタマイズして追加するキャラクタ・セットごとに前述のコマンドを繰り返し発行し、同一の gss_custom.zip ファイルにそれぞれの定義を追加できます。たとえば、UNIX の場合は次のコマンドを入力します。

```
$ORACLE_HOME/JRE/bin/jre -classpath $ORACLE_HOME/jlib/gss-1_1.zip:
               $ORACLE_HOME/jlib/gss_charset-1_2.zip Ginstall lx22710.nlt


$ORACLE_HOME/JRE/bin/jre -classpath $ORACLE_HOME/jlib/gss-1_1.zip:
               $ORACLE_HOME/jlib/gss_charset-1_2.zip Ginstall lx22711.nlt

$ORACLE_HOME/JRE/bin/jre -classpath $ORACLE_HOME/jlib/gss-1_1.zip:
               $ORACLE_HOME/jlib/gss_charset-1_2.zip Ginstall lx22712.nlt
```

gss_custom.zip には、lx22710.nlt、lx22711.nlt および lx22712.nlt が組み込まれます。

作成した gss_custom.zip ファイルは、$ORACLE_HOME/ocommon/nls/admin/data ディレクトリに格納します。次のコマンドを入力します。

```
% cp gss_custom.zip $ORACLE_HOME/ocommon/nls/admin/data
```

Java コンポーネントへのカスタム Zip ファイルの追加

次の Java コンポーネントに gss_custom.zip ファイルを追加できます。

- Java Virtual Machine
- Oracle HTTP Server
- クライアント上の JDBC

Java Virtual Machine zip ファイルをデータベースにロードします。

UNIX の場合は、次のコマンドを入力します。

```
%loadjava -u sys/passwd -grant EXECUTE -synonym -r -r -v gss_custom.zip
```

Windows の場合は、次のコマンドを入力します。

```
loadjava -u sys/passwd -grant EXECUTE -synonym -r -r -v gss_custom.zip
```

passwd を SYS のパスワードで置き換えてください。

Oracle HTTP Server jserv.properties ファイルを編集します。

UNIX の場合は、次の行を追加します。

```
wrapper.classpath = $ORACLE_HOME/ocommon/nls/admin/data/gss_custom.zip
```

Windows の場合は、次の行を追加します。

```
wrapper.classpath = %ORACLE_HOME%¥ocommon¥nls¥admin¥data¥gss_custom.zip
```

クライアント上の JDBC CLASSPATH を変更します。

UNIX の場合は、次のコマンドを入力します。

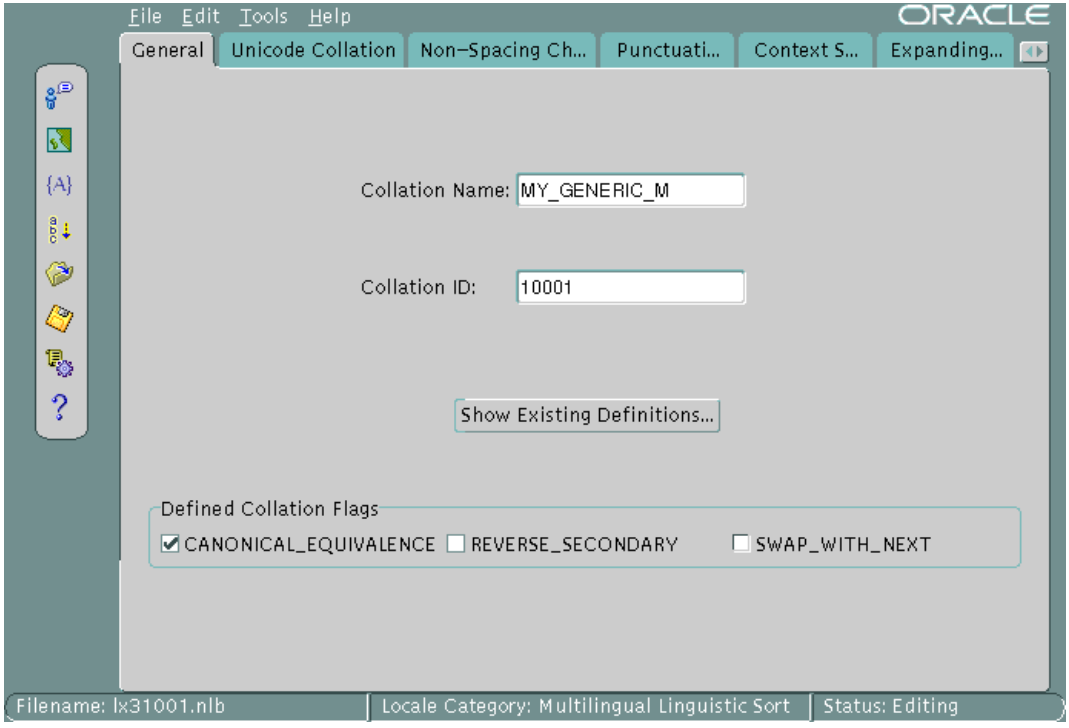
```
% setenv CLASSPATH $ORACLE_HOME/ocommon/nls/admin/data/gss_custom.zip
```

Windows の場合は、%ORACLE_HOME%¥ocommon¥nls¥admin¥data¥gss_custom.zip を既存の CLASSPATH に追加します。

Oracle Locale Builder を使用した新規言語ソートの作成

この項では、「Collation ID」を 10001 に設定して MY_GENERIC_M と呼ばれる新規の多言語ソートを作成する方法を示します。新規言語ソートの基礎として GENERIC_M 言語を使用しています。図 12-25 に、この操作の開始方法を示します。

図 12-25 照合の一般情報



フラグの設定は、自動的に導出されます。「SWAP_WITH_NEXT」は、タイ語とラオ語のソートに関係します。「REVERSE_SECONDARY」は、フランス語のソート用です。「CANONICAL_EQUIVALENCE」は、標準的な規則を使用するかどうかを決定します。この例では、「CANONICAL_EQUIVALENCE」がオンになっています。

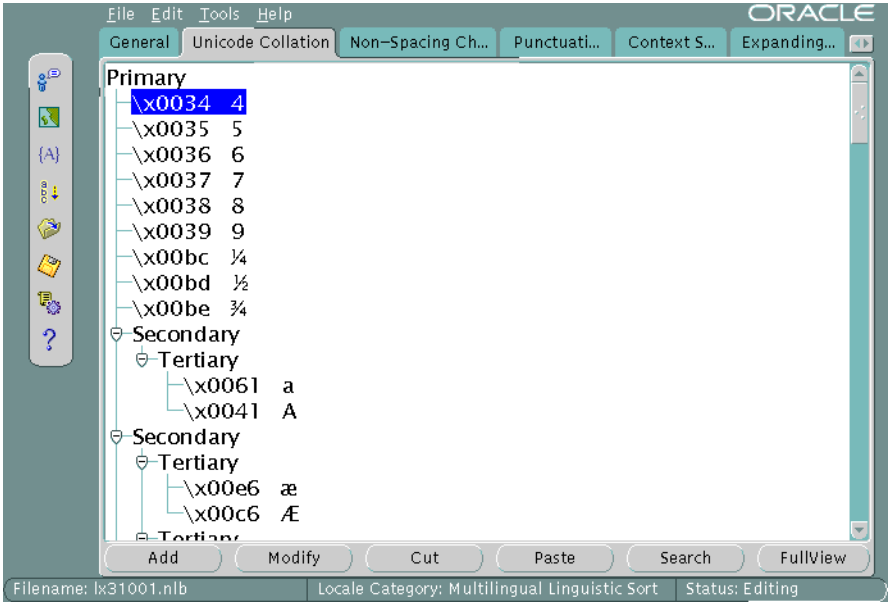
ユーザー定義ソートのための照合 ID（ソート ID）の有効範囲は、単一言語照合の場合は 1,000 ～ 2,000、多言語照合の場合は 10,000 ～ 11,000 です。

関連項目：

- 標準的な規則の詳細は、図 12-29「「Canonical Rules」ダイアログ・ボックス」を参照してください。
- 第 4 章「言語ソート」

図 12-26 に、「Unicode Collation Sequence」画面を示します。

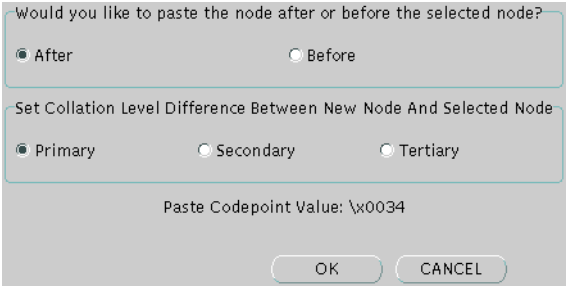
図 12-26「Unicode Collation Sequence」画面



この例では、文字の後にソートされるように数字を移動して、キャラクタ・セットをカスタマイズします。手順は次のとおりです。

1. 移動する Unicode 値を選択します。図 12-26 では、x0034 Unicode 値が選択されています。「Unicode Collation Sequence」画面での値の位置は、**ノード**と呼ばれます。
2. 「Cut」をクリックします。ノードを移動する位置を選択します。
3. 「Paste」をクリックします。「Paste」をクリックすると、図 12-27 に示す「Paste Node」ダイアログ・ボックスがオープンします。

図 12-27「Paste Node」ダイアログ・ボックス



4. 「Paste Node」ダイアログ・ボックスでは、選択した位置の後にノードを貼り付けるか前に貼り付けるかを選択できます。また、貼付け位置の隣にあるノードを基準として、貼り付けるノードのレベル（「Primary」、「Secondary」または「Tertiary」）も選択できます。

   ノードを貼り付ける位置とレベルを選択します。

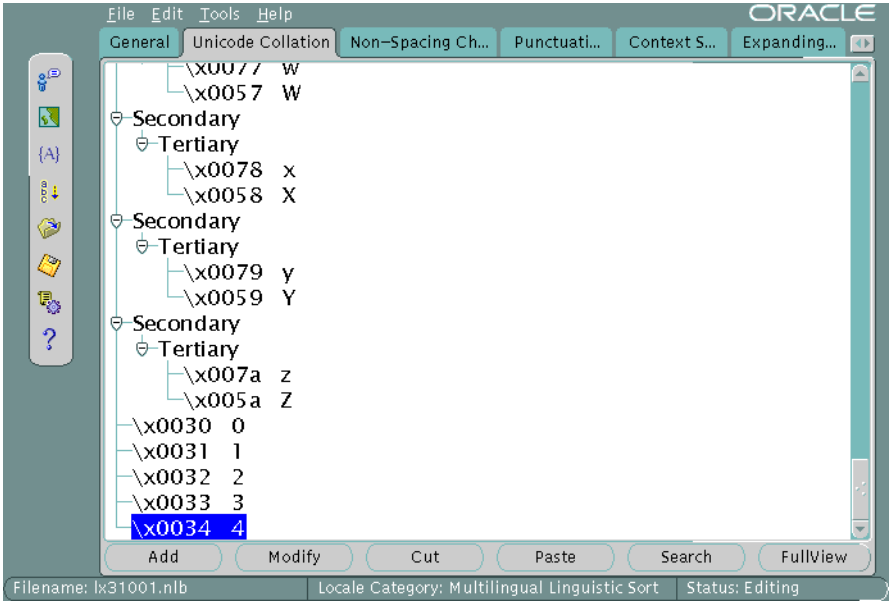
   図 12-27 では、「After」ボタンと「Primary」ボタンが選択されています。

5. 「OK」をクリックしてノードを貼り付けます。

同様の手順に従って、他の数字を文字 a ～ z の後の位置に移動します。

図 12-28 に、数字 0 ～ 4 を文字 a ～ z の後の位置に移動した後の「Unicode Collation Sequence」画面を示します。

図 12-28 変更後の「Unicode Collation Sequence」画面



これ以降の内容は、次のとおりです。

- 同じ発音区別記号を持つすべての文字のソート順序の変更
- 発音区別記号を持つ 1 文字のソート順序の変更

同じ発音区別記号を持つすべての文字のソート順序の変更

次の例に、発音区別記号付きの文字のソート順序を変更する方法を示します。そのためには、特定の発音区別記号を含むすべての文字のソート順序を変更する方法と、一度に 1 文字ずつ変更する方法があります。この例では、曲折アクセント記号付きのすべての文字（û など）のソート順序を、すべてのチルド付き文字の後になるように変更します。

「Tools」→「Canonical Rules」を選択して、現在のソート順序を確認します。図 12-29 の「Canonical Rules」ダイアログ・ボックスがオープンします。

図 12-29「Canonical Rules」ダイアログ・ボックス

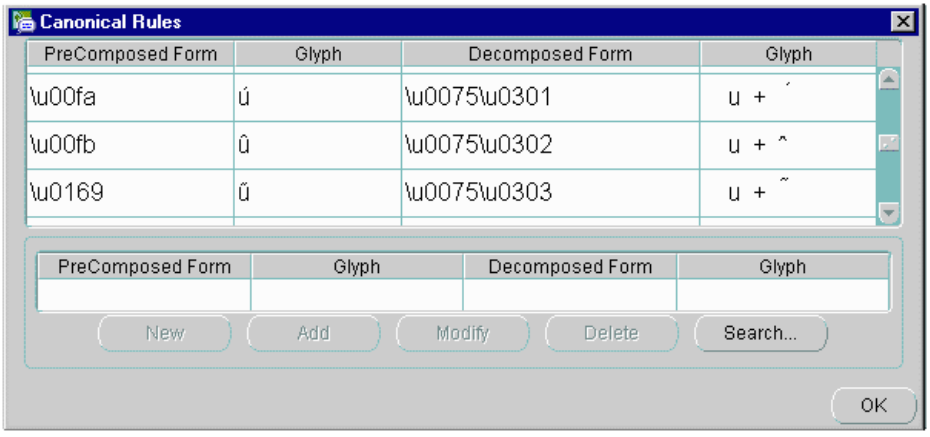
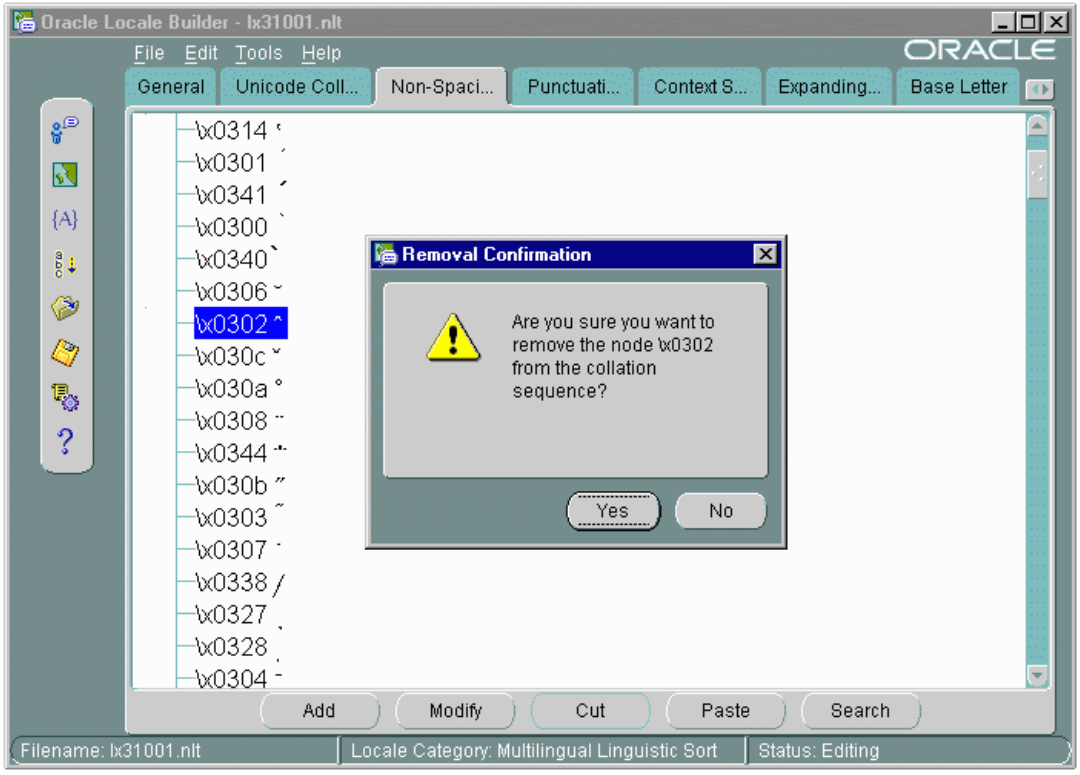


図 12-29 に、アクセント記号付き文字が、対応する標準的な文字に分解される方法と現在のソート順序を示します。たとえば、û は u と ^ の組合せとして表現されます。

> **関連項目：** 標準的な規則の詳細は、第 4 章「言語ソート」を参照してください。

Oracle Locale Builder のメイン・ウィンドウで、「Non-Spacing Characters」タブをクリックします。「Non-Spacing Characters 」画面を使用すると、発音区別記号に対する変更がすべての文字に適用されます。図 12-30 に、「Non-Spacing Characters 」画面を示します。

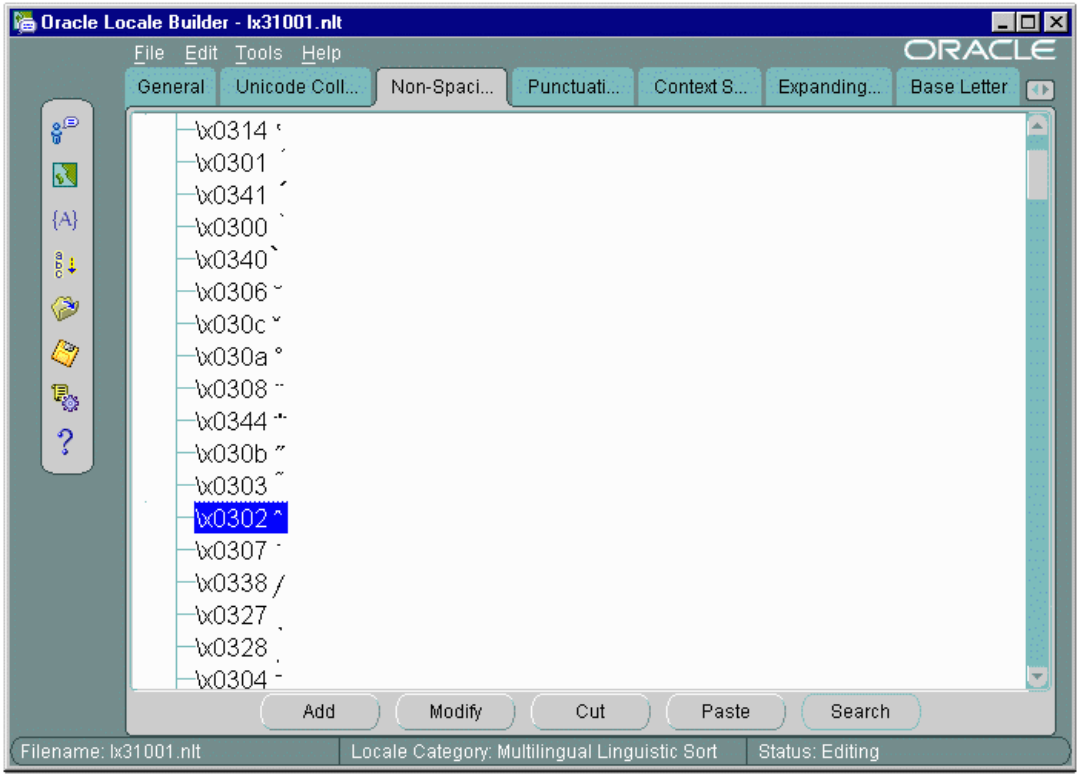
図 12-30 同じ発音区別記号を持つすべての文字のソート順序の変更



曲折アクセント記号を選択して「Cut」をクリックします。「Removal Confirmation」ダイアログ・ボックスで「Yes」をクリックします。チルドを選択して「Paste」をクリックします。「Paste Node」ダイアログ・ボックスで「After」と「Secondary」を選択して「OK」をクリックします。

図 12-31 に、新しいソート順序を示します。

図 12-31 同じ発音区別記号を持つ文字の新規のソート順序



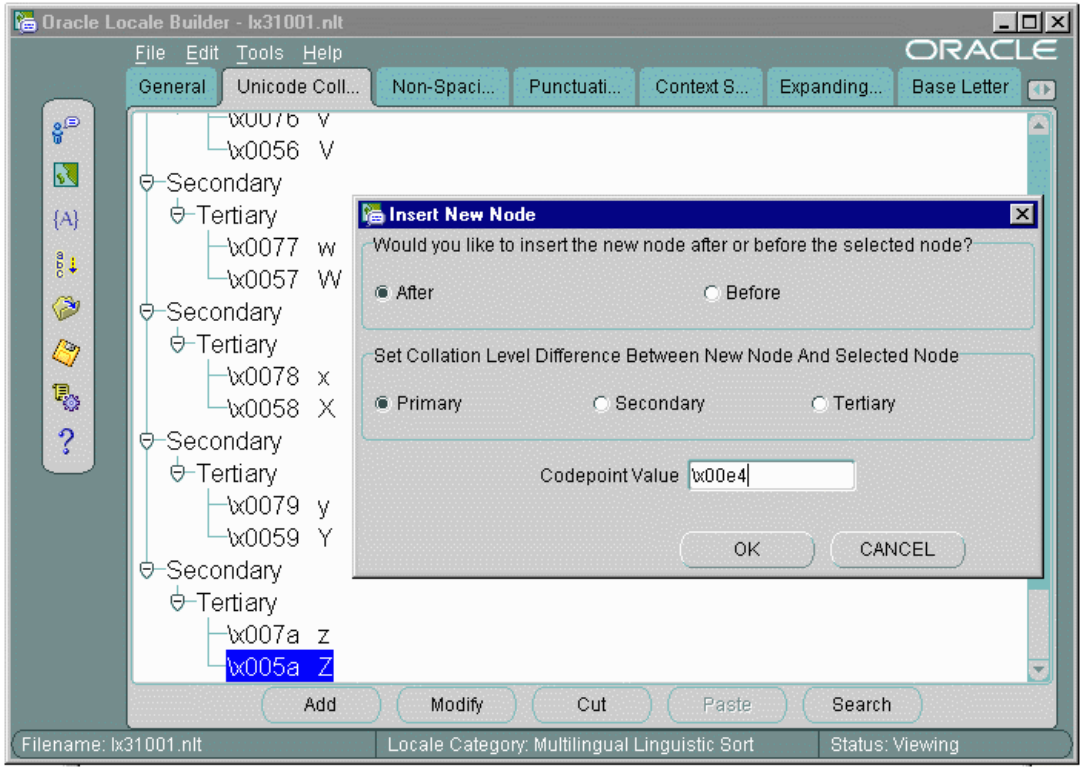
発音区別記号を持つ 1 文字のソート順序の変更

発音区別記号を持つ特定の文字の順序を変更するには、その文字を適切な位置に直接挿入します。発音区別記号付きの文字は「Unicode Collation」画面に表示されないため、新しい位置へのカット・アンド・ペーストはできません。

この例では、ä のソート順序を z の後にソートされるように変更します。

「Unicode Collation」タブをクリックします。ä の隣に配置する文字 z を選択します。「Add」をクリックします。図 12-32 の「Insert New Node」ダイアログ・ボックスが表示されます。

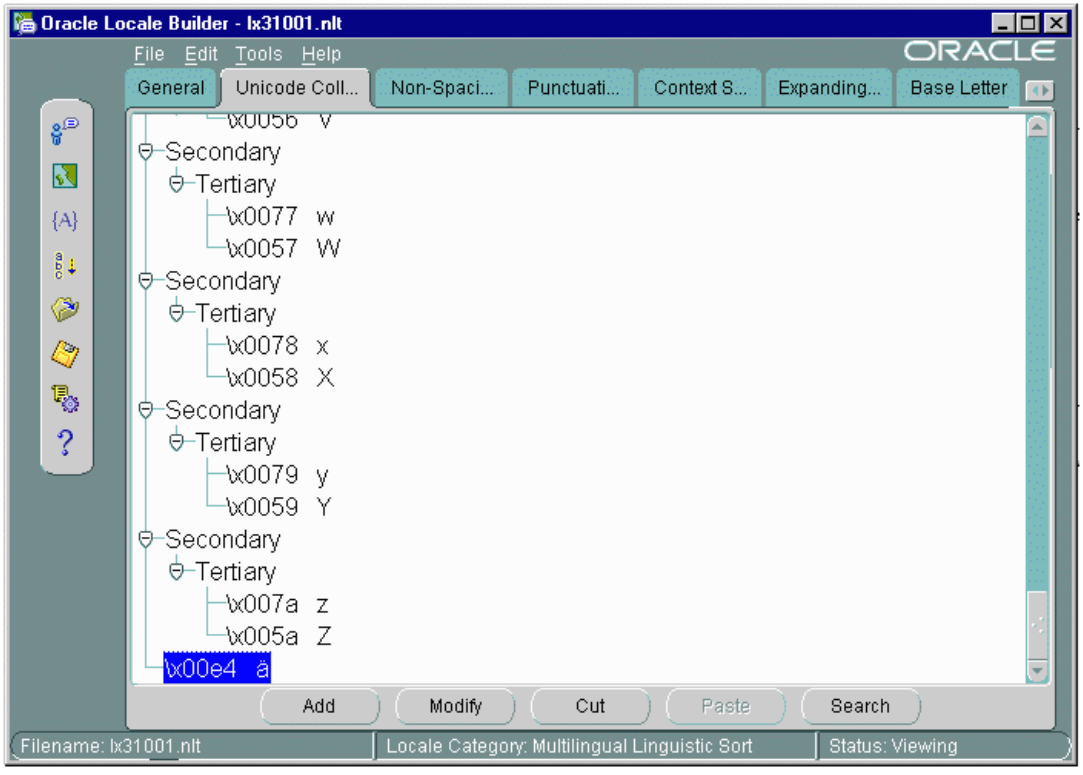
図 12-32 発音区別記号を持つ 1 文字のソート順序の変更



「Insert New Node」ダイアログ・ボックスで「After」と「Primary」を選択します。ä の Unicode コード・ポイント値を入力します。このコード・ポイント値は、`\x00e4` です。「OK」をクリックします。

図 12-33 に、変更後のソート順序を示します。

図 12-33 1 文字を変更後の新規ソート順序



NLB ファイルの生成とインストール

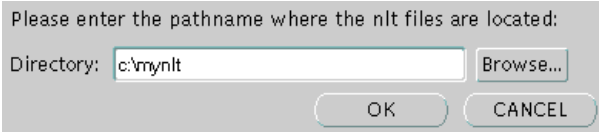
新規言語、地域、キャラクタ・セットまたは言語ソートを定義した後は、NLT ファイルから新規の NLB ファイルを生成します。

1. ORA_NLS33 ディレクトリ内で、NLS インストール・ブート・ファイル（lx0boot.nlb）と NLS システム・ブート・ファイル（lx1boot.nlb）のバックアップを作成します。UNIX プラットフォームの場合は、次のようなコマンドを入力します。

```
% cd $ORA_NLS33
% cp lx0boot.nlb lx0boot.nlb.orig
% cp lx1boot.nlb lx1boot.nlb.orig
```

2. Oracle Locale Builder で、「Tools」→「Generate NLB」を選択するか、左側のバーで「Generate NLB」アイコンをクリックします。
3. 「Browse」をクリックして、NLT ファイルが格納されているディレクトリを検索します。位置を指定するダイアログ・ボックスは、図 12-34 のとおりです。

図 12-34「Location」ダイアログ・ボックス

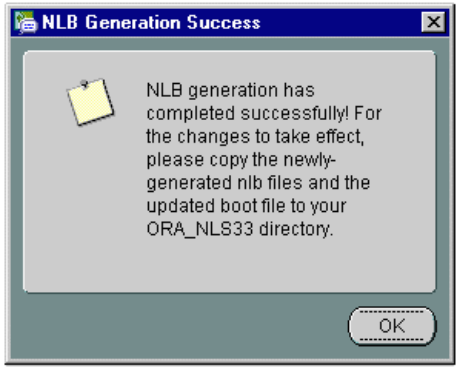


NLT ファイルは指定しないでください。Oracle Locale Builder は、各 NLT ファイルに対して NLB ファイルを生成します。

4. 「OK」をクリックして NLB ファイルを生成します。

図 12-35 は、ディレクトリ内の全 NLT ファイルに対して NLB ファイルが正常に生成されたことを示す最終的な通知を示しています。

図 12-35「NLB Generation Success」ダイアログ・ボックス



5. `lx1boot.nlb` ファイルを `ORA_NLS33` 初期化パラメータで指定されたパス（通常は `$ORACLE_HOME/OCOMMON/nls/admin/data`）にコピーします。たとえば、UNIX プラットフォームの場合は、次のようなコマンドを入力します。

```
% cp /directory_name/lx1boot.nlb $ORA_NLS33/lx1boot.nlb
```

6. 新規の NLB ファイルを ORA_NLS33 ディレクトリにコピーします。たとえば、UNIX プラットフォームの場合は、次のようなコマンドを入力します。

```
% cp /directory_name/lx22710.nlb $ORA_NLS33
% cp /directory_name/lx52710.nlb $ORA_NLA33
```

> **注意：** Oracle Locale Builder では、NLT ファイルと同じディレクトリに NLB ファイルが生成されます。

7. 各ハードウェア・プラットフォーム上で前述の手順を繰り返します。NLB ファイルは、プラットフォーム固有のバイナリ・ファイルです。新しい NLB ファイルは、サーバー・マシンとクライアント・マシンの両方でコンパイルしてインストールする必要があります。
8. 新規に作成したロケール・データを使用できるように、データベースを再起動します。
9. 新規のロケール・データをクライアント側で使用するには、NLB ファイルをインストールしてからクライアントを終了し、再起動します。

A

ロケール・データ

この付録では、Oracle サーバーでサポートされている言語、地域、キャラクタ・セットおよびその他のロケール・データについて説明します。この付録の内容は、次のとおりです。

サポートされているキャラクタ・セット、言語、地域およびソート順序に関する情報は、V$NLS_VALID_VALUES 動的パフォーマンス・ビューを問い合せても取得できます。

> **関連項目：** このビューで戻すことができるデータの詳細は、『Oracle9*i* データベース・リファレンス』を参照してください。

言語

表 A-1 に、Oracle サーバーでサポートしている言語を示します。

表 A-1 Oracle がサポートしている言語

名前	略称
AMERICAN	us
ARABIC	ar
ASSAMESE	as
BANGLA	bn
BRAZILIAN PORTUGUESE	ptb
BULGARIAN	bg
CANADIAN FRENCH	frc
CATALAN	ca
CROATIAN	hr
CZECH	cs
DANISH	dk
DUTCH	nl
EGYPTIAN	eg
ENGLISH	gb
ESTONIAN	et
FINNISH	sf
FRENCH	f
GERMAN DIN	din
GERMAN	d
GREEK	el
GUJARATI	gu
HEBREW	iw
HINDI	hi
HUNGARIAN	hu
ICELANDIC	is

表 A-1 Oracle がサポートしている言語（続き）

名前	略称
INDONESIAN	in
ITALIAN	i
JAPANESE	ja
KANNADA	kn
KOREAN	ko
LATIN AMERICAN SPANISH	esa
LATVIAN	lv
LITHUANIAN	lt
MALAY	ms
MALAYALAM	ml
MARATHI	mr
MEXICAN SPANISH	esm
NORWEGIAN	n
ORIYA	or
POLISH	pl
PORTUGUESE	pt
PUNJABI	pa
ROMANIAN	ro
RUSSIAN	ru
SIMPLIFIED CHINESE	zhs
SLOVAK	sk
SLOVENIAN	sl
SPANISH	e
SWEDISH	s
TAMIL	ta
TELUGU	te
THAI	th
TRADITIONAL CHINESE	zht

表 A-1 Oracle がサポートしている言語（続き）

名前	略称
TURKISH	tr
UKRAINIAN	uk
VIETNAMESE	vn

翻訳済みメッセージ

Oracle エラー・メッセージは、表 A-2 に示す言語に翻訳されています。

表 A-2 Oracle がサポートしているメッセージ

名前	略称
ARABIC	ar
BRAZILIAN PORTUGUESE	ptb
CANADIAN FRENCH	frc
CATALAN	ca
CZECH	cs
DANISH	dk
DUTCH	nl
FINNISH	sf
FRENCH	f
GERMAN	d
GREEK	el
HEBREW	iw
HUNGARIAN	hu
ITALIAN	i
JAPANESE	ja
KOREAN	ko
LATIN AMERICAN SPANISH	esa
NORWEGIAN	n
POLISH	pl

表 A-2 Oracle がサポートしているメッセージ（続き）

名前	略称
PORTUGUESE	pt
ROMANIAN	ro
RUSSIAN	ru
SIMPLIFIED CHINESE	zhs
SLOVAK	sk
SPANISH	e
SWEDISH	s
THAI	th
TRADITIONAL CHINESE	zht
TURKISH	tr

地域

表 A-3 に、Oracle サーバーでサポートしている地域を示します。

表 A-3 Oracle がサポートしている地域

名前	名前	名前
ALGERIA	HONG KONG	PERU
AMERICA	HUNGARY	POLAND
AUSTRALIA	ICELAND	PORTUGAL
AUSTRIA	INDIA	PUERTO RICO
BAHRAIN	INDONESIA	QATAR
BANGLADESH	IRAQ	ROMANIA
BELGIUM	IRELAND	SAUDI ARABIA
BRAZIL	ISRAEL	SINGAPORE
BULGARIA	ITALY	SLOVAKIA
CANADA	JAPAN	SLOVENIA
CATALONIA	JORDAN	SOMALIA
CHILE	KAZAKHSTAN	SOUTH AFRICA

表 A-3 Oracle がサポートしている地域（続き）

名前	名前	名前
CHINA	KOREA	SPAIN
CIS	KUWAIT	SUDAN
COLOMBIA	LATVIA	SWEDEN
COSTA RICA	LEBANON	SWITZERLAND
CROATIA	LIBYA	SYRIA
CYPRUS	LITHUANIA	TAIWAN
CZECH REPUBLIC	LUXEMBOURG	THAILAND
DENMARK	MACEDONIA	THE NETHERLANDS
DJIBOUTI	MALAYSIA	TUNISIA
EGYPT	MAURITANIA	TURKEY
EL SALVADOR	MEXICO	UKRAINE
ESTONIA	MOROCCO	UNITED ARAB EMIRATES
FINLAND	NEW ZEALAND	UNITED KINGDOM
FRANCE	NICARAGUA	UZBEKISTAN
GUATEMALA	NORWAY	VENEZUELA
GERMANY	OMAN	VIETNAM
GREECE	PANAMA	YEMEN
-	-	YUGOSLAVIA

キャラクタ・セット

ここでは、Oracle がサポートしているキャラクタ・セットを、次の 3 つの言語グループに分けて示します。

- アジア地域言語のキャラクタ・セット
- ヨーロッパ地域言語のキャラクタ・セット
- 中東地域言語のキャラクタ・セット

また、共通のサブセットとスーパーセットの組合せも示します。

一部のキャラクタ・セットは、複数の言語に対して示されています。これは、そのキャラクタ・セットが多言語をサポートしているためです。たとえば、Unicode は世界の主なスクリプトの大半をサポートしているため、アジア地域、ヨーロッパ地域および中東地域の言語グループに記述されています。

コメントには、使用されるエンコーディングのタイプを次のように示します。

SB = シングルバイト・エンコーディング

MB = マルチバイト・エンコーディング

FIXED = 固定幅マルチバイト・エンコーディング

第 3 章「グローバリゼーション・サポート環境の設定」の説明のように、エンコーディングのタイプはパフォーマンスに影響を及ぼすため、言語のニーズを満たす最も効果的なエンコーティングを使用してください。また、一部のエンコーディング・タイプは、特定のデータ型のみで使用できます。たとえば、AL16UTF16 キャラクタ・セットは NCHAR キャラクタ・セットとしてのみ使用でき、データベース・キャラクタ・セットとしては使用できません。

また、コメントには、そのキャラクタ・セットに固有の他の機能も記述されています。これらの情報は、ユーザーまたはデータベース管理者にとって重要な場合があります。たとえば、そのキャラクタ・セットが新しいユーロ通貨記号をサポートしているかどうか、ユーザー定義文字がキャラクタ・セットのカスタマイズでサポートされているかどうか、およびキャラクタ・セットが（移行の場合に ALTER DATABASE [NATIONAL] CHARACTER SET 文を使用できる）ASCII の完全なスーパーセットであるかどうかが示されています。

EURO = ユーロ記号をサポート

UDC = ユーザー定義文字をサポート

ASCII = ASCII の完全なスーパーセット

個々のコード・ページのレイアウトは記述されていません。特定のキャラクタ・セットの固有な情報、文字レパートリおよびコード・ポイントの値については、実際の各国の規格、国際規格またはベンダー固有の規格を参照してください。

アジア地域言語のキャラクタ・セット

表 A-4 に、アジア地域の言語をサポートしている Oracle キャラクタ・セットを示します。

表 A-4 アジア地域言語のキャラクタ・セット

名前	説明	コメント
BN8BSCII	バングラデシュ国内規格コード 8 ビット BSCII	SB、ASCII
ZHT16BIG5	BIG5 16 ビット繁体字中国語	MB、ASCII
ZHT16HKSCS	MS Windows コード・ページ 950 香港補足キャラクタ・セット付き	MB、ASCII、EURO
ZHS16CGB231280	CGB2312-80 16 ビット簡体字中国語	MB、ASCII
ZHS32GB18030	GB18030-2000	MB、ASCII、EURO
JA16EUC	EUC 24 ビット日本語	MB、ASCII
JA16EUCTILDE	波形のダッシュとチルドが Unicode との間でマッピングされる方法を除き、JA16EUC と同じ	MB、ASCII
JA16EUCYEN	EUC 24 ビット日本語（バックスラッシュ（\）は日本語の円記号（¥）になります。）	MB
ZHT32EUC	EUC 32 ビット繁体字中国語	MB、ASCII
ZHS16GBK	GBK 16 ビット簡体字中国語	MB、ASCII、UDC
ZHT16CCDC	HP CCDC 16 ビット繁体字中国語	MB、ASCII
JA16DBCS	IBM EBCDIC 16 ビット日本語	MB、UDC
JA16EBCDIC930	IBM DBCS コード・ページ 290 16 ビット日本語	MB、UDC
KO16DBCS	IBM EBCDIC 16 ビット韓国語	MB、UDC
ZHS16DBCS	IBM EBCDIC 16 ビット簡体字中国語	MB、UDC
ZHT16DBCS	IBM EBCDIC 16 ビット繁体字中国語	MB、UDC
KO16KSC5601	KSC5601 16 ビット韓国語	MB、ASCII
KO16KSCCS	KSCCS 16 ビット韓国語	MB、ASCII
JA16VMS	JVMS 16 ビット日本語	MB、ASCII
ZHS16MACCGB231280	Mac クライアント CGB2312-80 16 ビット簡体字中国語	MB
JA16MACSJIS	Mac クライアント・シフト JIS 16 ビット日本語	MB
TH8MACTHAI	Mac クライアント 8 ビット・ラテン語 / タイ語	SB

表 A-4 アジア地域言語のキャラクタ・セット（続き）

名前	説明	コメント
TH8MACTHAIS	Mac サーバー 8 ビット・ラテン語 / タイ語	SB、ASCII
TH8TISEBCDICS	タイ工業規格 620-2533-EBCDIC サーバー 8 ビット	SB
ZHT16MSWIN950	MS Windows コード・ページ 950 繁体字中国語	MB、ASCII、UDC
KO16MSWIN949	MS Windows コード・ページ 949 韓国語	MB、ASCII、UDC
VN8MSWIN1258	MS Windows コード・ページ 1258 8 ビット・ベトナム語	SB、ASCII、EURO
IN8ISCII	複数スクリプト・インド標準 8 ビット・ラテン / インド諸言語	SB、ASCII
JA16SJIS	シフト JIS 16 ビット日本語	MB、ASCII、UDC
JA16SJISTILDE	波形のダッシュとチルドが Unicode との間でマッピングされる方法を除き、JA16SJIS と同じ	MB、ASCII、UDC
JA16SJISYEN	シフト JIS 16 ビット日本語（バックスラッシュ（\）は日本語の円記号（¥）になります。）	MB、UDC
ZHT32SOPS	SOPS 32 ビット繁体字中国語	MB、ASCII
ZHT16DBT	台湾課税 16 ビット繁体字中国語	MB、ASCII
TH8TISASCII	タイ工業規格 620-2533 - ASCII 8 ビット	SB、ASCII、EURO
TH8TISEBCDIC	タイ工業規格 620-2533 - EBCDIC 8 ビット	SB
ZHT32TRIS	TRIS 32 ビット繁体字中国語	MB、ASCII
AL16UTF16	詳細は、A-19 ページの「ユニバーサル・キャラクタ・セット」を参照してください。	MB、EURO、FIXED
AL32UTF8	詳細は、A-19 ページの「ユニバーサル・キャラクタ・セット」を参照してください。	MB、ASCII、EURO
UTF8	詳細は、A-19 ページの「ユニバーサル・キャラクタ・セット」を参照してください。	MB、ASCII、EURO
UTFE	詳細は、A-19 ページの「ユニバーサル・キャラクタ・セット」を参照してください。	MB、EURO
VN8VN3	VN3 8 ビット・ベトナム語	SB、ASCII

ヨーロッパ地域言語のキャラクタ・セット

表 A-5 に、ヨーロッパ地域の言語をサポートしている Oracle キャラクタ・セットを示します。

表 A-5 ヨーロッパ地域言語のキャラクタ・セット

名前	説明	コメント
US7ASCII	ASCII 7 ビット米語	SB、ASCII
SF7ASCII	ASCII 7 ビット・フィンランド語	SB
YUG7ASCII	ASCII 7 ビット・ユーゴスラビア語	SB
RU8BESTA	BESTA 8 ビット・ラテン語 / キリル文字	SB、ASCII
EL8GCOS7	Bull EBCDIC GCOS7 8 ビット・ギリシア語	SB
WE8GCOS7	Bull EBCDIC GCOS7 8 ビット西ヨーロッパ語	SB
EL8DEC	DEC 8 ビット・ラテン語 / ギリシア語	SB
TR7DEC	DEC VT100 7 ビット・トルコ語	SB
TR8DEC	DEC 8 ビット・トルコ語	SB、ASCII
TR8EBCDIC1026	EBCDIC コード・ページ 1026 8 ビット・トルコ語	SB
TR8EBCDIC1026S	EBCDIC コード・ページ 1026 サーバー 8 ビット・トルコ語	SB
TR8PC857	IBM-PC コード・ページ 857 8 ビット・トルコ語	SB、ASCII
TR8MACTURKISH	MAC クライアント 8 ビット・トルコ語	SB
TR8MACTURKISHS	MAC サーバー 8 ビット・トルコ語	SB、ASCII
TR8MSWIN1254	MS Windows コード・ページ 1254 8 ビット・トルコ語	SB、ASCII、EURO
WE8BS2000L5	Siemens EBCDIC.DF.L5 8 ビット西ヨーロッパ語 / トルコ語	SB
WE8DEC	DEC 8 ビット西ヨーロッパ語	SB、ASCII
D7DEC	DEC VT100 7 ビット・ドイツ語	SB
F7DEC	DEC VT100 7 ビット・フランス語	SB
S7DEC	DEC VT100 7 ビット・スウェーデン語	SB
E7DEC	DEC VT100 7 ビット・スペイン語	SB
NDK7DEC	DEC VT100 7 ビット・ノルウェー語 / デンマーク語	SB
I7DEC	DEC VT100 7 ビット・イタリア語	SB

表 A-5 ヨーロッパ地域言語のキャラクタ・セット（続き）

名前	説明	コメント
NL7DEC	DEC VT100 7 ビット・オランダ語	SB
CH7DEC	DEC VT100 7 ビット・スイス語（ドイツ語 / フランス語）	SB
SF7DEC	DEC VT100 7 ビット・フィンランド語	SB
WE8DG	DG 8 ビット西ヨーロッパ語	SB、ASCII
WE8EBCDIC37C	EBCDIC コード・ページ 37 8 ビット Oracle/c	SB
WE8EBCDIC37	EBCDIC コード・ページ 37 8 ビット西ヨーロッパ語	SB
D8EBCDIC273	EBCDIC コード・ページ 273/1 8 ビット・ドイツ語（オーストリア）	SB
DK8EBCDIC277	EBCDIC コード・ページ 277/1 8 ビット・デンマーク語	SB
S8EBCDIC278	EBCDIC コード・ページ 278/1 8 ビット・スウェーデン語	SB
I8EBCDIC280	EBCDIC コード・ページ 280/1 8 ビット・イタリア語	SB
WE8EBCDIC284	EBCDIC コード・ページ 284 8 ビット・スペイン語（南米）	SB
WE8EBCDIC285	EBCDIC コード・ページ 285 8 ビット西ヨーロッパ語	SB
WE8EBCDIC924	Latin 9 EBCDIC 924	SB、EBCDIC
WE8EBCDIC1047	EBCDIC コード・ページ 1047 8 ビット西ヨーロッパ語	SB
WE8EBCDIC1047E	Latin 1/ オープン・システム 1047	SB、EBCDIC、EURO
WE8EBCDIC1140	EBCDIC コード・ページ 1140 8 ビット西ヨーロッパ語	SB、EURO
WE8EBCDIC1140C	EBCDIC コード・ページ 1140 クライアント 8 ビット西ヨーロッパ語	SB、EURO
WE8EBCDIC1145	EBCDIC コード・ページ 1145 8 ビット西ヨーロッパ語	SB、EURO
WE8EBCDIC1146	EBCDIC コード・ページ 1146 8 ビット西ヨーロッパ語	SB、EURO
WE8EBCDIC1148	EBCDIC コード・ページ 1148 8 ビット西ヨーロッパ語	SB、EURO
WE8EBCDIC1148C	EBCDIC コード・ページ 1148 クライアント 8 ビット西ヨーロッパ語	SB、EURO
F8EBCDIC297	EBCDIC コード・ページ 297 8 ビット・フランス語	SB
WE8EBCDIC500C	EBCDIC コード・ページ 500 8 ビット Oracle/c	SB
WE8EBCDIC500	EBCDIC コード・ページ 500 8 ビット西ヨーロッパ語	SB

表 A-5 ヨーロッパ地域言語のキャラクタ・セット（続き）

名前	説明	コメント
EE8EBCDIC870	EBCDIC コード・ページ 870 8 ビット東ヨーロッパ語	SB
EE8EBCDIC870C	EBCDIC コード・ページ 870 クライアント 8 ビット東ヨーロッパ語	SB
EE8EBCDIC870S	EBCDIC コード・ページ 870 サーバー 8 ビット東ヨーロッパ語	SB
WE8EBCDIC871	EBCDIC コード・ページ 871 8 ビット・アイスランド語	SB
EL8EBCDIC875	EBCDIC コード・ページ 875 8 ビット・ギリシア語	SB
EL8EBCDIC875R	EBCDIC コード・ページ 875 サーバー 8 ビット・ギリシア語	SB
CL8EBCDIC1025	EBCDIC コード・ページ 1025 8 ビット・キリル文字	SB
CL8EBCDIC1025C	EBCDIC コード・ページ 1025 クライアント 8 ビット・キリル文字	SB
CL8EBCDIC1025R	EBCDIC コード・ページ 1025 サーバー 8 ビット・キリル文字	SB
CL8EBCDIC1025S	EBCDIC コード・ページ 1025 サーバー 8 ビット・キリル文字	SB
CL8EBCDIC1025X	EBCDIC コード・ページ 1025（修正版）8 ビット・キリル文字	SB
BLT8EBCDIC1112	EBCDIC コード・ページ 1112 8 ビット・バルト多言語	SB
BLT8EBCDIC1112S	EBCDIC コード・ページ 1112 8 ビット・サーバー・バルト多言語	SB
D8EBCDIC1141	EBCDIC コード・ページ 1141 8 ビット・ドイツ語（オーストリア）	SB、EURO
DK8EBCDIC1142	EBCDIC コード・ページ 1142 8 ビット・デンマーク語	SB、EURO
S8EBCDIC1143	EBCDIC コード・ページ 1143 8 ビット・スウェーデン語	SB、EURO
I8EBCDIC1144	EBCDIC コード・ページ 1144 8 ビット・イタリア語	SB、EURO
F8EBCDIC1147	EBCDIC コード・ページ 1147 8 ビット・フランス語	SB、EURO
EEC8EUROASCI	EEC Targon 35 ASCI 西ヨーロッパ語 / ギリシア語	SB
EEC8EUROPA3	EEC EUROPA3 8 ビット西ヨーロッパ語 / ギリシア語	SB
LA8PASSPORT	ドイツ政府プリンタ 8 ビット全ヨーロッパ・ラテン語	SB、ASCII
WE8HP	HP LaserJet 8 ビット西ヨーロッパ語	SB
WE8ROMAN8	HP Roman8 8 ビット西ヨーロッパ語	SB、ASCII
HU8CWI2	ハンガリア語 8 ビット CWI-2	SB、ASCII

表 A-5 ヨーロッパ地域言語のキャラクタ・セット（続き）

名前	説明	コメント
HU8ABMOD	ハンガリア語 8 ビット特別 AB Mod	SB、ASCII
LV8RST104090	IBM-PC 代替コード・ページ 8 ビット・ラトビア語（ラテン語 / キリル文字）	SB、ASCII
US8PC437	IBM-PC コード・ページ 437 8 ビット米語	SB、ASCII
BG8PC437S	IBM-PC コード・ページ 437 8 ビット（ブルガリア語の修正版）	SB、ASCII
EL8PC437S	IBM-PC コード・ページ 437 8 ビット（ギリシア語の修正版）	SB、ASCII
EL8PC737	IBM-PC コード・ページ 737 8 ビット・ギリシア語 / ラテン語	SB
LT8PC772	IBM-PC コード・ページ 772 8 ビット・リトアニア語（ラテン語 / キリル文字）	SB、ASCII
LT8PC774	IBM-PC コード・ページ 774 8 ビット・リトアニア語（ラテン語）	SB、ASCII
BLT8PC775	IBM-PC コード・ページ 775 8 ビット・バルト語	SB、ASCII
WE8PC850	IBM-PC コード・ページ 850 8 ビット西ヨーロッパ語	SB、ASCII
EL8PC851	IBM-PC コード・ページ 851 8 ビット・ギリシア語 / ラテン語	SB、ASCII
EE8PC852	IBM-PC コード・ページ 852 8 ビット東ヨーロッパ語	SB、ASCII
RU8PC855	IBM-PC コード・ページ 855 8 ビット・ラテン語 / キリル文字	SB、ASCII
WE8PC858	IBM-PC コード・ページ 858 8 ビット西ヨーロッパ語	SB、ASCII、EURO
WE8PC860	IBM-PC コード・ページ 860 8 ビット西ヨーロッパ語	SB、ASCII
IS8PC861	IBM-PC コード・ページ 861 8 ビット・アイスランド語	SB、ASCII
CDN8PC863	IBM-PC コード・ページ 863 8 ビット・フランス語（カナダ）	SB、ASCII
N8PC865	IBM-PC コード・ページ 865 8 ビット・ノルウェー語	SB、ASCII
RU8PC866	IBM-PC コード・ページ 866 8 ビット・ラテン語 / キリル文字	SB、ASCII
EL8PC869	IBM-PC コード・ページ 869 8 ビット・ギリシア語 / ラテン語	SB、ASCII
LV8PC1117	IBM-PC コード・ページ 1117 8 ビット・ラトビア語	SB、ASCII
US8ICL	ICL EBCDIC 8 ビット米語	SB
WE8ICL	ICL EBCDIC 8 ビット西ヨーロッパ語	SB
WE8ISOICLUK	ICL 特別バージョン ISO8859-1	SB

表 A-5 ヨーロッパ地域言語のキャラクタ・セット（続き）

名前	説明	コメント
WE8ISO8859P1	ISO 8859-1 西ヨーロッパ語	SB、ASCII
EE8ISO8859P2	ISO 8859-2 東ヨーロッパ語	SB、ASCII
SE8ISO8859P3	ISO 8859-3 南ヨーロッパ語	SB、ASCII
NEE8ISO8859P4	ISO 8859-4 北ヨーロッパ語および北東ヨーロッパ語	SB、ASCII
CL8ISO8859P5	ISO 8859-5 ラテン語 / キリル文字	SB、ASCII
AR8ISO8859P6	ISO 8859-6 ラテン語 / アラビア語	SB、ASCII
EL8ISO8859P7	ISO 8859-7 ラテン語 / ギリシア語	SB、ASCII、EURO
IW8ISO8859P8	ISO 8859-8 ラテン語 / ヘブライ語	SB、ASCII
NE8ISO8859P10	ISO 8859-10 北ヨーロッパ語	SB、ASCII
BLT8ISO8859P13	ISO 8859-13 バルト語	SB、ASCII
CEL8ISO8859P14	ISO 8859-13 ケルト系言語	SB、ASCII
WE8ISO8859P15	ISO 8859-15 西ヨーロッパ語	SB、ASCII、EURO
LA8ISO6937	ISO 6937 8 ビット・テキスト通信用コード・キャラクタ・セット	SB、ASCII
IW7IS960	イスラエル標準 960 7 ビット・ラテン語 / ヘブライ語	SB
AR8ARABICMAC	Mac クライアント 8 ビット・ラテン語 / アラビア語	SB
EE8MACCE	Mac クライアント 8 ビット中央ヨーロッパ語	SB
EE8MACCROATIAN	Mac クライアント 8 ビット・クロアチア語	SB
WE8MACROMAN8	Mac クライアント 8 ビット拡張 Roman8 西ヨーロッパ語	SB
EL8MACGREEK	Mac クライアント 8 ビット・ギリシア語	SB
IS8MACICELANDIC	Mac クライアント 8 ビット・アイスランド語	SB
CL8MACCYRILLIC	Mac クライアント 8 ビット・ラテン語 / キリル文字	SB
AR8ARABICMACS	Mac サーバー 8 ビット・ラテン語 / アラビア語	SB、ASCII
EE8MACCES	Mac サーバー 8 ビット中央ヨーロッパ語	SB、ASCII
EE8MACCROATIANS	Mac サーバー 8 ビット・クロアチア語	SB、ASCII
WE8MACROMAN8S	Mac サーバー 8 ビット拡張 Roman8 西ヨーロッパ語	SB、ASCII
CL8MACCYRILLICS	Mac サーバー 8 ビット・ラテン語 / キリル文字	SB、ASCII

表 A-5 ヨーロッパ地域言語のキャラクタ・セット（続き）

名前	説明	コメント
EL8MACGREEKS	Mac サーバー 8 ビット・ギリシア語	SB、ASCII
IS8MACICELANDICS	Mac サーバー 8 ビット・アイスランド語	SB
BG8MSWIN	MS Windows 8 ビット・ブルガリア・キリル文字	SB、ASCII
LT8MSWIN921	MS Windows コード・ページ 921 8 ビット・リトアニア語	SB、ASCII
ET8MSWIN923	MS Windows コード・ページ 923 8 ビット・エストニア語	SB、ASCII
EE8MSWIN1250	MS Windows コード・ページ 1250 8 ビット東ヨーロッパ語	SB、ASCII、EURO
CL8MSWIN1251	MS Windows コード・ページ 1251 8 ビット・ラテン語 / キリル文字	SB、ASCII、EURO
WE8MSWIN1252	MS Windows コード・ページ 1252 8 ビット西ヨーロッパ語	SB、ASCII、EURO
EL8MSWIN1253	MS Windows コード・ページ 1253 8 ビット・ラテン語 / ギリシア語	SB、ASCII、EURO
BLT8MSWIN1257	MS Windows コード・ページ 1257 8 ビット・バルト語	SB、ASCII、EURO
BLT8CP921	ラトビア標準 LVS8-92(1) Windows/UNIX 8 ビット・バルト語	SB、ASCII
LV8PC8LR	ラトビア・バージョン IBM-PC コード・ページ 866 8 ビット・ラテン語 / キリル文字	SB、ASCII
WE8NCR4970	NCR 4970 8 ビット西ヨーロッパ語	SB、ASCII
WE8NEXTSTEP	NeXTSTEP ポストスクリプト 8 ビット西ヨーロッパ語	SB、ASCII
CL8ISOIR111	ISOIR111 キリル文字	SB
CL8KOI8R	RELCOM インターネット標準 8 ビット・ラテン語 / キリル文字	SB、ASCII
CL8KOI8U	KOI8 キリル文字（ウクライナ）	SB
US8BS2000	Siemens 9750-62 EBCDIC 8 ビット米語	SB
DK8BS2000	Siemens 9750-62 EBCDIC 8 ビット・デンマーク語	SB
F8BS2000	Siemens 9750-62 EBCDIC 8 ビット・フランス語	SB
D8BS2000	Siemens 9750-62 EBCDIC 8 ビット・ドイツ語	SB
E8BS2000	Siemens 9750-62 EBCDIC 8 ビット・スペイン語	SB
S8BS2000	Siemens 9750-62 EBCDIC 8 ビット・スウェーデン語	SB
DK7SIEMENS9780X	Siemens 97801/97808 7 ビット・デンマーク語	SB

表 A-5 ヨーロッパ地域言語のキャラクタ・セット（続き）

名前	説明	コメント
F7SIEMENS9780X	Siemens 97801/97808 7 ビット・フランス語	SB
D7SIEMENS9780X	Siemens 97801/97808 7 ビット・ドイツ語	SB
I7SIEMENS9780X	Siemens 97801/97808 7 ビット・イタリア語	SB
N7SIEMENS9780X	Siemens 97801/97808 7 ビット・ノルウェー語	SB
E7SIEMENS9780X	Siemens 97801/97808 7 ビット・スペイン語	SB
S7SIEMENS9780X	Siemens 97801/97808 7 ビット・スウェーデン語	SB
EE8BS2000	Siemens EBCDIC.DF.04 8 ビット東ヨーロッパ語	SB
WE8BS2000	Siemens EBCDIC.DF.04 8 ビット西ヨーロッパ語	SB
WE8BS2000E	Siemens EBCDIC.DF.04 8 ビット西ヨーロッパ語	SB、EURO
CL8BS2000	Siemens EBCDIC.EHC.LC 8 ビット・キリル文字	SB
AL16UTF16	詳細は、A-19 ページの「ユニバーサル・キャラクタ・セット」を参照してください。	MB、EURO、FIXED
AL32UTF8	詳細は、A-19 ページの「ユニバーサル・キャラクタ・セット」を参照してください。	MB、ASCII、EURO
UTF8	詳細は、A-19 ページの「ユニバーサル・キャラクタ・セット」を参照してください。	MB、ASCII、EURO
UTFE	詳細は、A-19 ページの「ユニバーサル・キャラクタ・セット」を参照してください。	MB、EURO

中東地域言語のキャラクタ・セット

表 A-6 に、中東地域の言語をサポートしている Oracle キャラクタ・セットを示します。

表 A-6 中東地域のキャラクタ・セット

名前	説明	コメント
AR8APTEC715	APTEC 715 サーバー 8 ビット・ラテン語 / アラビア語	SB、ASCII
AR8APTEC715T	APTEC 715 8 ビット・ラテン語 / アラビア語	SB
AR8ASMO708PLUS	ASMO 708 Plus 8 ビット・ラテン語 / アラビア語	SB、ASCII
AR8ASMO8X	ASMO 拡張 708 8 ビット・ラテン語 / アラビア語	SB、ASCII
AR8ADOS710	アラビア語 MS-DOS 710 サーバー 8 ビット・ラテン語 / アラビア語	SB、ASCII
AR8ADOS710T	アラビア語 MS-DOS 710 8 ビット・ラテン語 / アラビア語	SB
AR8ADOS720	アラビア語 MS-DOS 720 サーバー 8 ビット・ラテン語 / アラビア語	SB、ASCII
AR8ADOS720T	アラビア語 MS-DOS 720 8 ビット・ラテン語 / アラビア語	SB
TR7DEC	DEC VT100 7 ビット・トルコ語	SB
TR8DEC	DEC 8 ビット・トルコ語	SB
WE8EBCDIC37C	EBCDIC コード・ページ 37 8 ビット Oracle/c	SB
IW8EBCDIC424	EBCDIC コード・ページ 424 8 ビット・ラテン語 / ヘブライ語	SB
IW8EBCDIC424S	EBCDIC コード・ページ 424 サーバー 8 ビット・ラテン語 / ヘブライ語	SB
WE8EBCDIC500C	EBCDIC コード・ページ 500 8 ビット Oracle/c	SB
IW8EBCDIC1086	EBCDIC コード・ページ 1086 8 ビット・ヘブライ語	SB
AR8EBCDIC420S	EBCDIC コード・ページ 420 サーバー 8 ビット・ラテン語 / アラビア語	SB
AR8EBCDICX	EBCDIC XBASIC サーバー 8 ビット・ラテン語 / アラビア語	SB
TR8EBCDIC1026	EBCDIC コード・ページ 1026 8 ビット・トルコ語	SB
TR8EBCDIC1026S	EBCDIC コード・ページ 1026 サーバー 8 ビット・トルコ語	SB
AR8HPARABIC8T	HP 8 ビット・ラテン語 / アラビア語	SB
TR8PC857	IBM-PC コード・ページ 857 8 ビット・トルコ語	SB、ASCII

表 A-6 中東地域のキャラクタ・セット（続き）

名前	説明	コメント
IW8PC1507	IBM-PC コード・ページ 1507/862 8 ビット・ラテン語 / ヘブライ語	SB、ASCII
AR8ISO8859P6	ISO 8859-6 ラテン語 / アラビア語	SB、ASCII
IW8ISO8859P8	ISO 8859-8 ラテン語 / ヘブライ語	SB、ASCII
WE8ISO8859P9	ISO 8859-9 西ヨーロッパ語およびトルコ語	SB、ASCII
LA8ISO6937	ISO 6937 8 ビット・テキスト通信用コード・キャラクタ・セット	SB、ASCII
IW7IS960	イスラエル標準 960 7 ビット・ラテン語 / ヘブライ語	SB
IW8MACHEBREW	Mac クライアント 8 ビット・ヘブライ語	SB
AR8ARABICMAC	Mac クライアント 8 ビット・ラテン語 / アラビア語	SB
AR8ARABICMACT	Mac 8 ビット・ラテン語 / アラビア語	SB
TR8MACTURKISH	Mac クライアント 8 ビット・トルコ語	SB
IW8MACHEBREWS	Mac サーバー 8 ビット・ヘブライ語	SB、ASCII
AR8ARABICMACS	Mac サーバー 8 ビット・ラテン語 / アラビア語	SB、ASCII
TR8MACTURKISHS	Mac サーバー 8 ビット・トルコ語	SB、ASCII
TR8MSWIN1254	MS Windows コード・ページ 1254 8 ビット・トルコ語	SB、ASCII、EURO
IW8MSWIN1255	MS Windows コード・ページ 1255 8 ビット・ラテン語 / ヘブライ語	SB、ASCII、EURO
AR8MSWIN1256	MS Windows コード・ページ 1256 8 ビット・ラテン語 / アラビア語	SB、ASCII、EURO
IN8ISCII	複数スクリプト・インド標準 8 ビット・ラテン / インド諸言語	SB
AR8MUSSAD768	Mussa'd Alarabi/2 768 サーバー 8 ビット・ラテン語 / アラビア語	SB、ASCII
AR8MUSSAD768T	Mussa'd Alarabi/2 768 8 ビット・ラテン語 / アラビア語	SB
AR8NAFITHA711	Nafitha 拡張 711 サーバー 8 ビット・ラテン語 / アラビア語	SB、ASCII
AR8NAFITHA711T	Nafitha 拡張 711 8 ビット・ラテン語 / アラビア語	SB
AR8NAFITHA721	Nafitha International 721 サーバー 8 ビット・ラテン語 / アラビア語	SB、ASCII
AR8NAFITHA721T	Nafitha International 721 8 ビット・ラテン語 / アラビア語	SB

表 A-6 中東地域のキャラクタ・セット（続き）

名前	説明	コメント
AR8SAKHR706	SAKHR 706 サーバー 8 ビット・ラテン語 / アラビア語	SB、ASCII
AR8SAKHR707	SAKHR 707 サーバー 8 ビット・ラテン語 / アラビア語	SB、ASCII
AR8SAKHR707T	SAKHR 707 8 ビット・ラテン語 / アラビア語	SB
AR8XBASIC	XBASIC 8 ビット・ラテン語 / アラビア語	SB
WE8BS2000L5	Siemens EBCDIC.DF.04.L5 8 ビット西ヨーロッパ語 / トルコ語	SB
AL16UTF16	詳細は、A-19 ページの「ユニバーサル・キャラクタ・セット」を参照してください。	MB、EURO、FIXED
AL32UTF8	詳細は、A-19 ページの「ユニバーサル・キャラクタ・セット」を参照してください。	MB、ASCII、EURO
UTF8	詳細は、A-19 ページの「ユニバーサル・キャラクタ・セット」を参照してください。	MB、ASCII、EURO
UTFE	詳細は、A-19 ページの「ユニバーサル・キャラクタ・セット」を参照してください。	MB、EURO

ユニバーサル・キャラクタ・セット

表 A-7 に、世界言語サポートを提供している Oracle キャラクタ・セットを示します。これらのキャラクタ・セットは、アジア、ヨーロッパおよび中東地域の言語にかぎらず、それらを含めた世界中のすべての言語をサポートします。

表 A-7 ユニバーサル・キャラクタ・セット

名前	説明	コメント
AL16UTF16	Unicode 3.1 UTF-16 ユニバーサル・キャラクタ・セット	MB、EURO、FIXED
AL32UTF8	Unicode 3.1 UTF-8 ユニバーサル・キャラクタ・セット	MB、ASCII、EURO
UTF8	Unicode 3.0 UTF-8 ユニバーサル・キャラクタ・セット、CESU-8 準拠	MB、ASCII、EURO
UTFE	Unicode 3.0 UTF-8 ユニバーサル・キャラクタ・セットの EBCDIC フォーム	MB、EURO

注意： CESU-8 では、補助文字の表現を除き、UTF-8 と同じ Unicode のコード体系が定義されています。CESU-8 では、補助文字が、UTF-16 の各サロゲート・コードを UTF-8 の変換と同じ 8 ビット形式に変換した結果得られる 6 バイトの順序として表されます。ただし、最初の入力のサロゲート・ペアからスカラー値への変換では行われません。Unicode テクニカル・レポート #26 を参照してください。

関連項目： 第 5 章「Unicode を使用した多言語データベースのサポート」

キャラクタ・セット変換のサポート

次のキャラクタ・セット・エンコーディングは、変換専用としてサポートされています。データベース・キャラクタ・セットまたは各国語キャラクタ・セットとしては使用できません。

- AL16UTF16LE
- ISO2022-CN
- ISO2022-JP
- ISO2022-KR
- HZ-GB-2312

これらのキャラクタ・セットは、CONVERT 関数で source_char_set または dest_char_set に使用できます。

関連項目：

- CONVERT 関数の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。
- 7-6 ページ「CONVERT 関数」

サブセットとスーパーセット

表 A-8 に、一般的なサブセットとスーパーセットの関連を示します。

表 A-8 サブセットとスーパーセットのペア

サブセット	スーパーセット
AR8ADOS710	AR8ADOS710T
AR8ADOS720	AR8ADOS720T
AR8ADOS720T	AR8ADOS720
AR8APTEC715	AR8APTEC715T
AR8ARABICMACT	AR8ARABICMAC
AR8ISO8859P6	AR8ASMO708PLUS
AR8ISO8859P6	AR8ASMO8X
AR8MUSSAD768	AR8MUSSAD768T
AR8MUSSAD768T	AR8MUSSAD768
AR8NAFITHA711	AR8NAFITHA711T
AR8NAFITHA721	AR8NAFITHA721T
AR8SAKHR707	AR8SAKHR707T
AR8SAKHR707T	AR8SAKHR707
BLT8CP921	BLT8ISO8859P13
BLT8CP921	LT8MSWIN921
D7DEC	D7SIEMENS9780X
D7SIEMENS9780X	D7DEC
DK7SIEMENS9780X	N7SIEMENS9780X
I7DEC	I7SIEMENS9780X
I7SIEMENS9780X	IW8EBCDIC424
IW8EBCDIC424	IW8EBCDIC1086
KO16KSC5601	KO16MSWIN949
LT8MSWIN921	BLT8ISO8859P13
LT8MSWIN921	BLT8CP921
N7SIEMENS9780X	DK7SIEMENS9780X

表 A-8 サブセットとスーパーセットのペア（続き）

サブセット	スーパーセット
US7ASCII	表 A-9「US7ASCII のスーパーセット」を参照してください。
WE16DECTST	WE16DECTST2
WE16DECTST2	WE16DECTST
WE8DEC	TR8DEC
WE8DEC	WE8NCR4970
WE8ISO8859P1	WE8MSWIN1252
WE8ISO8859P9	TR8MSWIN1254
WE8NCR4970	TR8DEC
WE8NCR4970	WE8DEC
WE8PC850	WE8PC858
ZHS16GBK	ZHS32GB18030

US7ASCII は特殊で、非常に多くのキャラクタ・セットが US7ASCII のスーパーセットになっています。表 A-9 に、US7ASCII のスーパーセットをリストします。

表 A-9 US7ASCII のスーパーセット

スーパーセット	スーパーセット	スーパーセット
AL24UTFFSS	EE8MACCES	NEE8ISO8859P4
AL32UTF8	EE8MACCROATIANS	RU8BESTA
AR8ADOS710	EE8MSWIN1250	RU8PC855
AR8ADOS710T	EE8PC852	RU8PC866
AR8ADOS720	EL8DEC	SE8ISO8859P3
AR8ADOS720T	EL8ISO8859P7	TH8MACTHAIS
AR8APTEC715	EL8MACGREEKS	TH8TISASCII
AR8APTEC715T	EL8MSWIN1253	TR8DEC
AR8ARABICMACS	EL8PC437S	TR8MACTURKISHS
AR8ASMO708PLUS	EL8PC851	TR8MSWIN1254
AR8ASMO8X	EL8PC869	TR8PC857

表 A-9 US7ASCII のスーパーセット（続き）

スーパーセット	スーパーセット	スーパーセット
AR8HPARABIC8T	ET8MSWIN923	US8PC437
AR8ISO8859P6	HU8ABMOD	UTF8
AR8MSAWIN	HU8CWI2	VN8MSWIN1258
AR8MUSSAD768	IN8ISCII	VN8VN3
AR8MUSSAD768T	IS8PC861	WE8DEC
AR8NAFITHA711	IW8ISO8859P8	WE8DG
AR8NAFITHA711T	IW8MACHEBREWS	WE8ISO8859P1
AR8NAFITHA721	IW8MSWIN1255	WE8ISO8859P15
AR8NAFITHA721T	IW8PC1507	WE8ISO8859P9
AR8SAKHR706	JA16EUC	WE8MACROMAN8S
AR8SAKHR707	JA16SJIS	WE8MSWIN1252
AR8SAKHR707T	JA16TSTSET	WE8NCR4970
BG8MSWIN	JA16TSTSET2	WE8NEXTSTEP
BG8PC437S	JA16VMS	WE8PC850
BLT8CP921	KO16KSC5601	WE8PC858
BLT8ISO8859P13	KO16KSCCS	WE8PC860
BLT8MSWIN1257	KO16MSWIN949	WE8ROMAN8
BLT8PC775	KO16TSTSET	ZHS16CGB231280
BN8BSCII	LA8ISO6937	ZHS16GBK
CDN8PC863	LA8PASSPORT	ZHT16BIG5
CEL8ISO8859P14	LT8MSWIN921	ZHT16CCDC
CL8ISO8859P5	LT8PC772	ZHT16DBT
CL8KOI8R	LT8PC774	ZHT16HKSCS
CL8KOI8U	LV8PC1117	ZHT16MSWIN950
CL8ISOIR111	LV8PC8LR	ZHT32EUC
CL8MACCYRILLICS	LV8RST104090	ZHT32SOPS
CL8MSWIN1251	N8PC865	ZHT32TRIS

表 A-9 US7ASCII のスーパーセット（続き）

スーパーセット	スーパーセット	スーパーセット
EE8ISO8859P2	NE8ISO8859P10	ZHS32GB18030
ZHT32EUCTST	-	-

言語ソート

Oracle では、単一言語ソートと多言語ソートの 2 種類の言語ソートを提供しています。また、単一言語ソートを拡張することにより特殊な事例に対応できます。一般的に、（接頭辞 X で表す）この特殊な事例とは、文字がその ASCII 値とは異なる順にソートされる場合を意味します。たとえば、XSPANISH では、*ch* と *ll* が 1 つの文字として取り扱われます。

表 A-10 に、Oracle サーバーがサポートしている単一言語ソートを示します。

表 A-10 単一言語ソート

基本名	拡張名	特殊な事例
ARABIC	-	-
ARABIC_MATCH	-	-
ARABIC_ABJ_SORT	-	-
ARABIC_ABJ_MATCH	-	-
ASCII7	-	-
BENGALI	-	-
BIG5	-	-
BINARY	-	-
BULGARIAN	-	-
CANADIAN FRENCH	-	-
CATALAN	XCATALAN	æ、AE、ß
CROATIAN	XCROATIAN	D、L、N、d、l、n、ß
CZECH	XCZECH	ch、CH、Ch、ß
CZECH_PUNCTUTION	XCZECH_ PUNCTUATION	ch、CH、Ch、ß
DANISH	XDANISH	A、ß、Å、å
DUTCH	XDUTCH	ij、IJ
EBCDIC	-	-

表 A-10 単一言語ソート（続き）

基本名	拡張名	特殊な事例
EEC_EURO	-	-
EEC_EUROPA3	-	-
ESTONIAN	-	-
FINNISH	-	-
FRENCH	XFRENCH	-
GERMAN	XGERMAN	ß
GERMAN_DIN	XGERMAN_DIN	ß、ä、ö、ü、Ä、Ö、Ü
GBK	-	-
GREEK	-	-
HEBREW	-	-
HKSCS	-	-
HUNGARIAN	XHUNGARIAN	cs、gy、ny、sz、ty、zs、ß、CS、Cs、GY、Gy、NY、Ny、SZ、Sz、TY、Ty、ZS、Zs
ICELANDIC	-	-
INDONESIAN	-	-
ITALIAN	-	-
JAPANESE	-	-
LATIN	-	-
LATVIAN	-	-
LITHUANIAN	-	-
MALAY	-	-
NORWEGIAN	-	-
POLISH	-	-
PUNCTUATION	XPUNCTUATION	-
ROMANIAN	-	-
RUSSIAN	-	-
SLOVAK	XSLOVAK	dz、DZ、Dz、ß（*caron*）

表 A-10 単一言語ソート（続き）

基本名	拡張名	特殊な事例
SLOVENIAN	XSLOVENIAN	ß
SPANISH	XSPANISH	ch、ll、CH、Ch、LL、Ll
SWEDISH	-	-
SWISS	XSWISS	ß
THAI_DICTIONARY	-	-
THAI_TELEPHONE	-	-
TURKISH	XTURKISH	æ、AE、ß
UKRAINIAN	-	-
UNICODE_BINARY	-	-
VIETNAMESE	-	-
WEST_EUROPEAN	XWEST_EUROPEAN	ß

表 A-11 に、Oracle で使用可能な多言語ソートを示します。これらすべては、GENERIC_M（ラテン語に基づく文字ソートの ISO 規格）を基礎として含んでいます。多言語ソートは、ラテン語に基づく文字とともに、特定の主要な言語に使用されます。たとえば、KOREAN_M は韓国語とラテン語に基づく文字はソートしますが、中国語、タイ語または日本語の文字は照合しません。

表 A-11 多言語ソート

基本名	説明
CANADIAN_M	フランス語（カナダ）のソートは、逆 2 次ソートで特殊な拡張文字をサポートしています。
DANISH_M	デンマーク語のソートは、大文字の前に小文字をソートします。
FRENCH_M	フランス語のソートは、逆 2 次ソートをサポートしています。
GENERIC_M	ISO14651 および Unicode の標準的な同値化規則（互換性のある同値化規則を除く）に基づいた一般的なソート順序です。
JAPANESE_M	日本語のソートは、SJIS キャラクタ・セットのソート順序および SJIS には含まれない EUC 文字をサポートしています。
KOREAN_M	韓国語のソート : ハングル文字は Unicode のバイナリ順序に基づいてソートします。ハンジャ文字は発音順に基づいてソートします。すべてのハングル文字がハンジャ文字の前にソートされます。

表 A-11 多言語ソート（続き）

基本名	説明
SPANISH_M	従来のスペイン語のソートは、特殊な短縮文字をサポートしています。
THAI_M	タイ語のソートは、一部の母音と子音について、文字の交換をサポートしています。
SCHINESE_RADICAL_M	この簡体字中国語のソートは、最初に部首、次に画数に基づいて行います。
SCHINESE_STROKE_M	この簡体字中国語のソートは、最初に画数、次に部首に基づいて行います。
SCHINESE_PINYIN_M	この簡体字中国語のソートは、ピンイン（中国語の発音記号）順に行います。
TCHINESE_RADICAL_M	この繁体字中国語のソートは、最初に部首、次に画数に基づいて行います。
TCHINESE_STROKE_M	この繁体字中国語のソートは、最初に画数、次に部首に基づいて行います。補助文字がサポートされます。

暦法

デフォルトでは、ほとんどの地域定義でグレゴリオ暦が使用されています。表 A-12 に、Oracle サーバーがサポートしているその他の暦法を示します。

表 A-12 サポートしている暦法

名前	デフォルトの日付書式	デフォルトの日付書式に使用するキャラクタ・セット
Japanese Imperial（日本の元号暦）	EEYYMMDD	JA16EUC
ROC Official（台湾暦）	EEyymmdd	ZHT32EUC
Thai Buddha（タイ仏教暦）	dd month EE yyyy	TH8TISASCII
Persian（ペルシャ暦）	DD Month YYYY	AR8ASMO8X
Arabic Hijrah（イスラム歴）	DD Month YYYY	AR8ISO8859P6
English Hijrah（英語版イスラム歴）	DD Month YYYY	AR8ISO8859P6

図 A-1 に、ROC Official（台湾暦）で 1998 年 3 月 20 日がどのように表示されるかを示します。

図 A-1 ROC Official（台湾の公式の暦）の例

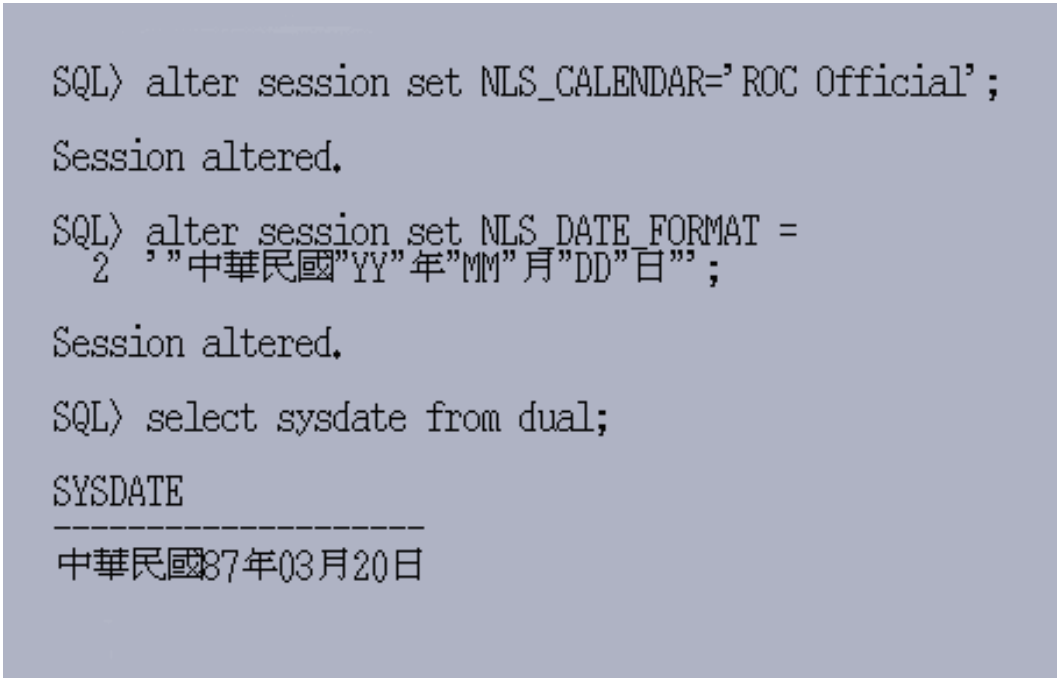


図 A-2 に、Japanese Imperial（日本の元号暦）で 1998 年 3 月 27 日がどのように表示されるかを示します。

図 A-2 Japanese Imperial（日本の元号暦）の例

```
SQL> alter session set NLS_CALENDAR =
  2  'Japanese Imperial';

Session altered.

SQL> alter session set NLS_DATE_FORMAT=
  2  '"平成"YY"年"MM"月"DD"日"'

Session altered.

SQL> select sysdate from dual;

SYSDATE
---------------
平成10年03月27日
```

廃止されたロケール・データ

Oracle サーバーのリリース 7.2 より前には、キャラクタ・セットを改名すると、その後のいくつかのリリースで、旧名が新規名とともにサポートされていました。リリース 7.2 以上では、旧名はサポートされません。

表 A-13 に、影響を受けるキャラクタ・セットを示します。次のキャラクタ・セットをコード内で参照する場合は、新規名に置き換えてください。

表 A-13 廃止になったキャラクタ・セットの新規名

旧名	新規名
AL24UTFSS	UTF8、AL32UTF8
AR8MSAWIN	AR8MSWIN1256
CL8EBCDIC875S	CL8EBCDIC875R
EL8EBCDIC875S	EL8EBCDIC875R
JVMS	JA16VMS
JEUC	JA16EUC
SJIS	JA16SJIS
JDBCS	JA16DBCS
KSC5601	KO16KSC5601
KDBCS	KO16DBCS
CGB2312-80	ZHS16CGB231280
CNS 11643-86	ZHT32EUC
JA16EUCFIXED	なし。新しい各国語キャラクタ・セット UTF8 および AL16UTF16 に置換されました。
ZHS32EUCFIXED	なし。新しい各国語キャラクタ・セット UTF8 および AL16UTF16 に置換されました。
ZHS16GBKFIXED	なし。新しい各国語キャラクタ・セット UTF8 および AL16UTF16 に置換されました。
JA16DBCSFIXED	なし。新しい各国語キャラクタ・セット UTF8 および AL16UTF16 に置換されました。
KO16DBCSFIXED	なし。新しい各国語キャラクタ・セット UTF8 および AL16UTF16 に置換されました。
ZHS16DBCSFIXED	なし。新しい各国語キャラクタ・セット UTF8 および AL16UTF16 に置換されました。

表 A-13 廃止になったキャラクタ・セットの新規名（続き）

旧名	新規名
ZHS16CGB231280 FIXED	なし。新しい各国語キャラクタ・セット UTF8 および AL16UTF16 に置換されました。
ZHT16DBCSFIXED	なし。新しい各国語キャラクタ・セット UTF8 および AL16UTF16 に置換されました。
KO16KSC5601FIXED	なし。新しい各国語キャラクタ・セット UTF8 および AL16UTF16 に置換されました。
JA16SJISFIXED	なし。新しい各国語キャラクタ・セット UTF8 および AL16UTF16 に置換されました。
ZHT16BIG5FIXED	なし。新しい各国語キャラクタ・セット UTF8 および AL16UTF16 に置換されました。
ZHT32TRISFIXED	なし。新しい各国語キャラクタ・セット UTF8 および AL16UTF16 に置換されました。

キャラクタ・セット CL8MSWINDOW31 は、サポートされなくなりました。新規キャラクタ・セット CL8MSWIN1251 は、実際には CL8MSWINDOW31 の複製で、以前のバージョンで省略されていた文字が一部含まれています。CL8MSWINDOW31 のかわりに CL8MSWIN1251 を使用してください。

AL24UTFFSS キャラクタ・セットのサポートの廃止

Unicode キャラクタ・セット AL24UTFFSS は、Oracle9*i* ではサポートされなくなりました。AL24UTFFSS は、Unicode 規格 1.1 に基づく UTF-8 コード体系をサポートする Unicode キャラクタ・セットとして Oracle7 から導入されましたが、現在では不要と考えられます。Oracle9*i* では、Unicode 規格 3.1 に基づき、Unicode 拡張が含まれている Unicode データベース・キャラクタ・セット AL32UTF8 および UTF8 を提供しています。

既存の AL24UTFFSS データベースを移行する場合は、Oracle9*i* にアップグレードする前に UTF8 にアップグレードします。新規データベース・キャラクタ・セットへの移行については、既存のデータベース・キャラクタ・セットを UTF8 に移行する前に、Character Set Scanner を使用してデータの分析を行うことをお薦めします。

関連項目： 第 11 章「Character Set Scanner」

ベンガル語定義の廃止

ベンガル語（BENGALI）定義には、Unicode 規格との互換性がありません。かわりにバングラ語（BANGLA）定義を使用することをお薦めします。バングラ語は、Oracle9*i* データベース リリース 1（9.0.1）で導入されました。

ベンガル語定義は、Oracle9*i* データベース リリース 2（9.2）でサポートされていますが、将来のリリースでサポートが廃止になる可能性があります。

チェコスロバキア地域定義の廃止

Oracle9*i* データベース リリース 2（9.2）では、チェコ共和国（CZECH REPUBLIC）またはスロバキア（SLOVAKIA）地域定義を使用することをお薦めします。チェコスロバキア（CZECHOSLOVAKIA）地域定義は、Oracle9*i* データベース リリース 2（9.2）でサポートされていますが、将来のリリースでサポートが廃止になる可能性があります。

B

Unicode 文字コードの割当て

この付録では、Unicode による文字の割当て方法を紹介します。この付録の内容は、次のとおりです。

- Unicode のコード範囲
- UTF-16 エンコーディング
- UTF-8 エンコーディング

Unicode のコード範囲

表 B-1 に、Unicode で UTF-16 文字コード用に割り当てられているコード範囲を示します。

表 B-1 UTF-16 文字コード用の Unicode 文字コード範囲

文字のタイプ	最初の 16 ビット	次の 16 ビット
ASCII	0000-007F	-
ヨーロッパ語（ASCII を除く）、アラビア語、ヘブライ語	0080-07FF	-
インド語、タイ語、記号（ユーロ記号など）、中国語、日本語、韓国語	0800-0FFF 1000 - CFFF D000 - D7FF F900-FFFF	-
Private Use Area #1	E000 - EFFF F000 - F8FF	-
補助文字：その他の中国語、日本語および韓国語の文字、旧漢字、音楽記号および数学的記号	D800 - D8BF D8C0 - DABF DAC0 - DB7F	DC00 - DFFF DC00 - DFFF DC00 - DFFF
Private Use Area #2	DB80 - DBBF DBC0 - DBFF	DC00 - DFFF DC00 - DFFF

表 B-2 に、Unicode で UTF-8 文字コード用に割り当てられているコード範囲を示します。

表 B-2 UTF-8 文字コード用の Unicode 文字コード範囲

文字のタイプ	第 1 バイト	第 2 バイト	第 3 バイト	第 4 バイト
ASCII	00-7F	-	-	-
ヨーロッパ語（ASCII を除く）、アラビア語、ヘブライ語	C2-DF	80-BF	-	-
インド語、タイ語、記号（ユーロ記号など）、中国語、日本語、韓国語	E0 E1-EC ED EF	A0-BF 80-BF 80-9F A4-BF	80-BF 80-BF 80-BF 80-BF	-

表 B-2 UTF-8 文字コード用の Unicode 文字コード範囲（続き）

文字のタイプ	第 1 バイト	第 2 バイト	第 3 バイト	第 4 バイト
Private Use Area #1	EE	80-BF	80-BF	-
	EF	80-A3	80-BF	
補助文字：その他の中国語、日本語および韓国語の文字、旧漢字、音楽記号および数学的記号	F0	90-BF	80-BF	80-BF
	F1-F2	80-BF	80-BF	80-BF
	F3	80-AF	80-BF	80-BF
Private Use Area #2	F3	B0-BF	80-BF	80-BF
	F4	80-8F	80-BF	80-BF

注意： 空白は、適用外のコード割当てを示します。文字コードは 16 進表現で示されています。

UTF-16 エンコーディング

表 B-1 のように、一部の文字（その他の中国語 / 日本語 / 韓国語の文字および Private Use Area #2）の UTF-16 文字コードは、2 単位の 16 ビットで表現されます。これらは補助文字です。1 つの補助文字は、2 つの 16 ビット値で構成されています。最初の 16 ビット値は 0xD800 ～ 0xDBFF の範囲内でエンコードされ、次の 16 ビット値は 0xDC00 ～ 0xDFFF の範囲内でエンコードされます。補助文字を使用すると、UTF-16 文字コードは 100 万種類以上の文字を表現できます。補助文字を使用しなければ、65,536 文字しか表現できません。Oracle の AL16UTF16 キャラクタ・セットは、補助文字をサポートしています。

関連項目： 5-3 ページ「補助文字」

UTF-8 エンコーディング

表 B-2 の UTF-8 文字コードは、次の条件が適用されることを示しています。

- ASCII 文字には 1 バイトを使用します。
- ヨーロッパ語（ASCII を除く）、アラビア語およびヘブライ語の文字の場合は、2 バイト必要です。
- インド語、タイ語、中国語、日本語および韓国語の文字、およびユーロなどの記号の場合は、3 バイト必要です。
- Private Use Area #1 の文字の場合は、3 バイト必要です。
- 補助文字の場合は、4 バイト必要です。
- Private Use Area #2 の文字の場合は、4 バイト必要です。

Oracle の AL32UTF8 キャラクタ・セットは、1 バイト、2 バイト、3 バイトおよび 4 バイト値をサポートしています。Oracle の UTF8 キャラクタ・セットは、1 バイト、2 バイトおよび 3 バイト値をサポートしていますが、4 バイト値はサポートしていません。

用語集

AL16UTF16

SQL NCHAR データ型に使用されるデフォルトの Oracle キャラクタ・セットで、各国語キャラクタ・セットに使用される。このキャラクタ・セットは、Unicode データを UTF-16 エンコーディングでエンコードする。

関連項目：「各国語キャラクタ・セット（national character set）」

AL32UTF8

SQL CHAR データ型に使用される Oracle キャラクタ・セットで、データベース・キャラクタ・セットに使用される。このキャラクタ・セットは、Unicode データを UTF-8 エンコーディングでエンコードする。

関連項目：「データベース・キャラクタ・セット（database character set）」

ASCII

米国の情報交換標準コード。英語用のエンコードされた共通 7 ビット・キャラクタ・セット。ASCII には、文字 A ～ Z と a ～ z、数字、句読点記号および制御文字が含まれる。Oracle キャラクタ・セット名は US7ASCII である。

EBCDIC

拡張 2 進化 10 進コード。IBM システムで最も使用されるエンコードされたキャラクタ・セット・ファミリ。

ISO

国際標準化機構（ISO）。130 か国からなる標準機関の世界的な連合。ISO では、世界規模の規格を開発および促進し、商品やサービスの国際的な交流を容易にすることを目的としている。

ISO 14651

ほとんどの言語向けに設計された国際的な多言語ソート規格。

関連項目：「多言語ソート（multilingual linguistic sort）」

ISO 8859

8 ビットのエンコードされたキャラクタ・セット・ファミリ。最も一般的なものは、ISO 8859-1（ISO Latin-1 として知られている）で、西ヨーロッパ諸国で使用されている。

ISO Latin-1

ISO 8859-1 キャラクタ・セット規格。ASCII に対する 8 ビット拡張機能で、西ヨーロッパで最も頻繁に使用される共通のラテン文字を含む 128 文字が追加されている。Oracle キャラクタ・セット名は、WE8ISO8859P1 である。

関連項目：「ISO 8859」

ISO/IEC 10646

現在、世界で使用されているほとんどの主要文字を定義しているユニバーサル・キャラクタ・セットの規格。1993 年には、ISO によって Unicode バージョン 1.1 が ISO/IEC 10646-1:1993 として承認されている。ISO/IEC 10646 には、2 バイト固定幅形式の UCS-2 と 4 バイト固定幅形式の UCS-4 がある。実装には 3 つのレベルがあり、すべてのレベルは複合文字のサポートに関係する。

- レベル 1 では、複合文字をサポートする必要はない。
- レベル 2 では、特定の文字（アラビア文字、タイ文字などのほとんどの Unicode 文字を含む）をサポートする必要がある。
- レベル 3 では、あらゆる言語の複合文字を無制限にサポートする必要がある。

ISO 通貨（ISO currency）

各国通貨を示すために使用される 3 文字の略称で、ISO 4217 規格に基づいている。たとえば、`USD` は米国のドルを表す。

NLB ファイル（NLB files）

ロケール固有のデータを定義するために Locale Builder が使用するバイナリ・ファイル。このファイルでは、特定リリースの Oracle データベース・サーバーに付属するロケール定義がすべて定義される。Oracle Locale Builder を使用すると、ユーザー定義の NLB ファイルを作成できる。

関連項目：「Oracle Locale Builder」および「NLT ファイル（NLT files）」

NLS

National Language Support。NLS によって、ユーザーは母国語でデータベースと対話できる。さらに、アプリケーションを様々な言語および文化の環境で実行できる。Oracle はかつてグローバル・ユーザーをサポートしていたため、この用語は幾分古い。

NLSRTL

National Language Support ランタイム・ライブラリ。このライブラリは、ロケールに依存しない国際化に関するアルゴリズムを提供する。ロケール固有の情報（つまり、NLSDATA）は、実行中に NLSRTL ライブラリによって読み込まれる。

NLT ファイル（NLT files）

ロケール固有のデータを定義するために Locale Builder が使用するテキスト・ファイル。このファイルはテキスト形式なので、内容を表示できる。

Oracle Locale Builder

ロケール固有のデータを表示、変更または定義する方法を提供する GUI ユーティリティ。言語、地域、キャラクタ・セットおよび言語ソートについて、独自の形式を作成することもできる。

SQL CHAR データ型（SQL CHAR datatypes）

CHAR、VARCHAR、VARCHAR2、CLOB および LONG データ型が含まれる。

SQL NCHAR データ型（SQL NCHAR datatypes）

NCHAR、NVARCHAR、NVARCHAR2 および NCLOB データ型が含まれる。

UCS-2

1993 ISO/IEC 規格のキャラクタ・セット。固定幅の 16 ビット Unicode キャラクタ・セットである。各文字は 16 ビットの領域を持つ。ISO Latin-1 文字は最初の 256 コード・ポイントであり、ISO Latin-1 の 16 ビット拡張と見なすことができる。

UCS-4

固定幅の 32 ビット Unicode キャラクタ・セット。各文字は 32 ビットの領域を持つ。UCS-2 の文字はこの規格の最初の 65,536 コード・ポイントであるため、UCS-2 の 32 ビット拡張とみなすことができる。ISO-10646 と呼ばれる場合もある。

Unicode

エンコードされたユニバーサル・キャラクタ・セット。Unicode を使用すると、1 つのキャラクタ・セットを使用して任意の言語の情報を格納できる。Unicode には、プラットフォーム、プログラムまたは言語に関係なく、すべての文字に対する一意のコード値が用意されている。

Unicode コード・ポイント（Unicode code point）

エンコードされたテキストを処理および交換する単位を表す 16 ビットのバイナリ値。U+0000 と U+FFFF 間のすべてのポイントが、コード・ポイントである。

Unicode データ型（Unicode datatype）

SQL NCHAR データ型（NCHAR、NVARCHAR2 および NCLOB）。データベース・キャラクタ・セットが Unicode でない場合も、これらのデータ型の列に Unicode 文字を格納できる。

Unicode データベース（Unicode database）

データベース・キャラクタ・セットが UTF-8 のデータベース。

UTF-16

Unicode の 16 ビット・エンコーディング。UCS-2 の拡張であり、UCS-2 コード・ポイントのペアを使用して、Unicode 3.1 に定義されている補助文字をサポートする。UTF-16 エンコーディングでは、1 つの Unicode 文字を、2 バイトか 4 バイトで表すことができる。ヨーロッパ言語とほとんどのアジア言語の文字（ASCII 文字を含む）は、ともに 2 バイトで表す。補助文字は、4 バイトで表す。

UTF-8

Unicode の 8 ビット・エンコーディング。可変幅エンコーディングである。UTF-8 エンコーディングでは、1 つの Unicode 文字を、1 バイト、2 バイト、3 バイトまたは 4 バイトで表すことができる。ヨーロッパ言語の文字は、1 バイトまたは 2 バイトで表す。ほとんどのアジア言語の文字は、3 バイトで表す。補助文字は、4 バイトで表す。

UTF8

UTF8 Oracle キャラクタ・セットは、文字を 1 バイト、2 バイトまたは 3 バイトでエンコードする。ASCII ベースのプラットフォーム用である。UTF8 キャラクタ・セットでは Unicode 3.0 がサポートされている。Unicode 3.1 までは特定の補助文字にコード・ポイントが割り当てられておらず、Unicode 3.0 では補助文字用のコード・ポイント範囲が割り当てられていた。補助文字は、6 バイトを占める 2 つの別個のユーザー定義文字として処理される。

UTFE

6 バイト補助文字のサポート付き Unicode 3.0 UTF-8 Oracle データベース・キャラクタ・セット。EBCDIC プラットフォームでのみ使用される。

絵文字（glyph）

文字の固有の表現。1 つの文字は、多数の異なる絵文字を持つことができる。たとえば、英大文字の最初の文字は、A、A、A のように印字または表示される。

これらの形式は、同じ文字を表現する異なる絵文字である。

関連項目：「文字（character）」

エンコーディング値（encoded value）

キャラクタ・セットの 1 文字の数値表現。たとえば、ASCII キャラクタ・セットでは、A のコード・ポイントは 0x41 である。文字のエンコーディング値は、その文字の**コード・ポイント**とも呼ばれる。

エンコードされたキャラクタ・セット（encoded character set）

文字コード体系が関連付けられているキャラクタ・セット。エンコードされたキャラクタ・セットは、各文字に割り当てる番号（文字コード）を指定する。

> **関連項目：**「文字コード体系（character encoding scheme）」

大 / 小文字の変換（case conversion）

ある文字を大文字から小文字に、または小文字から大文字に変換すること。

各国語キャラクタ・セット（national character set）

NCHAR、NVARCHAR2 および NCLOB 列に指定できるデータベース・キャラクタ・セットの代替キャラクタ・セット。各国語キャラクタ・セットは Unicode 内のみである。

キャラクタ・セット（character set）

特定の言語または言語グループのテキスト情報を表す要素の集まり。1 つの言語を複数のキャラクタ・セットで表現できる。

キャラクタ・セットは、必ずしも特定の文字コード体系を示すわけではない。文字コード体系は、キャラクタ・セットの各文字に対する文字コードの割当てである。

このマニュアルでは、通常、キャラクタ・セットは特定の文字コード体系を示している。したがって、このマニュアルでは、キャラクタ・セットとエンコードされたキャラクタ・セットは同じである。

キャラクタ・セットの移行

既存のデータベースのキャラクタ・セットを変更すること。

キャラクタ・セマンティクス（character semantics）

文字列を一連の文字として取り扱うこと。

> **関連項目：**「バイト・セマンティクス（byte semantics）」および「長さセマンティクス（length semantics）」

クライアント・キャラクタ・セット（client character set）

クライアントで使用される、エンコードされたキャラクタ・セット。クライアント・キャラクタ・セットは、サーバーのキャラクタ・セットとは異なる場合がある。サーバーのキャラクタ・セットは、**データベース・キャラクタ・セット**と呼ばれる。クライアント・キャラクタ・セットがデータベース・キャラクタ・セットと異なる場合は、キャラクタ・セット変換が必要である。

関連項目：「データベース・キャラクタ・セット（database character set）」

グローバリゼーション（globalization）

ソフトウェアを多様な言語および文化の環境に適したものにするプロセス。グローバリゼーションとローカライゼーションは異なるものである。ローカライゼーションとは、ソフトウェアをある固有のロケールで使用できるように準備するプロセスである。

ケース（case）

大文字であるか小文字であるかの条件を指す。たとえば、ラテン・アルファベットの A は、小文字の絵文字 a の大文字である。

言語索引（linguistic index）

言語上のソート順序に基づいた索引。

言語ソート（linguistic sort）

文字列のバイナリ表現ではなく、ロケールに関する要件に基づいた文字列の順序付け。

関連項目：「多言語ソート（multilingual linguistic sort）」および「単一言語ソート（monolingual linguistic sort）」

コード・ポイント（code point）

キャラクタ・セットの 1 文字の数値表現。たとえば、ASCII キャラクタ・セットでは、A のコード・ポイントは 0x41 である。文字のコード・ポイントは、その文字の**エンコーディング値**とも呼ばれる。

関連項目：「Unicode コード・ポイント（Unicode code point）」

サロゲート・ペア（surrogate pairs）

関連項目：「補助文字（supplementary characters）」

照合（collation）

特定のロケールの言語に関連付けられた文字のソートに関する規則に従って文字列を順序付けすること。**言語ソート**とも呼ばれる。

関連項目：

- 「言語ソート（linguistic sort）」
- 「単一言語ソート（monolingual linguistic sort）」
- 「多言語ソート（multilingual linguistic sort）」

シングルバイト（single-byte）

1 バイト。1 バイトは、通常 8 ビットで構成される。特定の言語のすべての文字に文字コードを割り当てる場合、1 バイト（8 ビット）では 256 の異なる文字を表現できる。

関連項目：「マルチバイト（multibyte）」

シングルバイト文字（single-byte character）

ある文字コード体系で、1 バイトの文字コードで構成される文字。コード体系が異なると、同じ文字に異なる文字コードが対応する場合がある。使用しているコード体系が不明な場合、Oracle ではどの文字がシングルバイト文字であるかを判断できない。たとえば、ユーロ通貨記号は、WE8MSWIN1252 のエンコードされたキャラクタ・セットでは 1 バイト、AL16UTF16 では 2 バイト、UTF8 では 3 バイトである。

関連項目：「マルチバイト・キャラクタ（multibyte character）」

シングルバイト文字列（single-byte character string）

次のいずれかで構成される文字列。

- 文字なし（**ヌル文字列**と呼ばれる）
- 1 つ以上のシングルバイト文字

スクリプト（script）

記述法で使用される関連する図形記号の集まり。一部のスクリプトは複数の言語を表現できる。また、言語によっては複数のスクリプトを使用するものもある。スクリプトの例として、ラテン文字、アラビア文字および漢字がある。

制限付き多言語サポート（restricted multilingual support）

関連する言語のグループに制限された多言語サポート。西ヨーロッパ諸国の言語は、ISO 8859-1 などで表現される。多言語サポートが制限付きの場合、タイ語は言語グループに追加できない。

多言語ソート（multilingual linguistic sort）

3 つのレベルで文字列を評価する Oracle ソート。アジア言語の場合は、1 言語によるデータしか存在しない場合にも、多言語ソートが必要である。多言語ソートは、複数言語によるデータが存在する場合にも使用される。

単一言語サポート（monolingual support）

1 つの言語のみのサポート。

単一言語ソート（monolingual linguistic sort）

文字列を 2 つのレベルで比較する Oracle ソート。単一言語ソートを使用すると、ヨーロッパ地域の大半の言語はソートできるが、アジア地域の言語には不向きである。

> **関連項目：**「多言語ソート（multilingual linguistic sort）」

置換文字（replacement character）

ソース文字がターゲット・キャラクタ・セットにない場合、その文字の変換時に使用される文字。たとえば、Oracle では ? がデフォルトの置換文字として使用される場合が多い。

データ・スキャニング（data scanning）

データベース・キャラクタ・セットの移行前に、キャラクタ・セット変換とデータの切捨てに伴う問題の可能性を識別するプロセス。

データベース・キャラクタ・セット（database character set）

テキストをデータベースに格納するために使用される、エンコードされたキャラクタ・セット。CHAR、VARCHAR2、LONG および固定幅の CLOB 列の値と、すべての SQL および PL/SQL テキストが含まれる。

長さセマンティクス（length semantics）

文字列の長さの取扱い方法を決定する。文字列の長さは、一連の文字またはバイトとして取り扱うことができる。

> **関連項目：**「キャラクタ・セマンティクス（character semantics）」および「バイト・セマンティクス（byte semantics）」

ヌル文字列（null string）

文字が含まれていない文字列。

バイト・セマンティクス（byte semantics）

文字列を一連のバイトとして取り扱うこと。

> **関連項目：**「キャラクタ・セマンティクス（character semantics）」および「長さセマンティクス（length semantics）」

バイナリ・ソート（binary sorting）

バイナリ・コード値に基づいた文字列の順序付け。

発音区別記号（diacritic）

文字または文字列の上または下にある記号で、それが付いていない場合の文字とは発音が異なることを示す。たとえば、`façade` の場合、セディラは発音区別記号である。セディラが付いている場合は、`c` の発音が変化する。

表意文字（ideograph）

概念を表現する記号。表意的記述法の例に中国語がある。

表音文字セット（syllabary）

日本語などの言語で使用する表意文字とともに表音的情報を伝達する方法を提供する。

標準的な同値化（canonical equivalence）

文字間または文字列間の基本的な同値化。たとえば、`ç` は `c` と `,` の組合せと同じである。正常にレンダリングされている場合は、両者を区別できない。

フォント（font）

キャラクタ・セット内の文字をグラフィカルに表現する順序付けられた絵文字の集まり。

補助文字（supplementary characters）

Unicode の最初のバージョンは 16 ビットの固定幅エンコーディングで、各文字のエンコーディングに 2 バイトを使用していた。このため、65,536 文字を表現できた。しかし、アジア言語の多数の表意文字の関係で、より多くの文字のサポートが必要となっている。

Unicode 3.1 では、このニーズを満たすために補助文字が定義されている。Unicode 3.1 では、2 つの 16 ビット・コード・ポイント（**サロゲート・ペア**とも呼ばれる）を使用して、1 つの文字が表現される。このため、さらに 1,048,576 文字の定義が可能である。Unicode 3.1 規格には、最初のグループとして 44,944 文字の補助文字が追加された。

マルチバイト（multibyte）

2 バイト以上であること。

特定の言語（または言語グループ）のすべての文字に対して文字コードが割り当てられた場合は、1 バイト（8 ビット）では 256 の異なる文字を表現できる。2 バイト（16 ビット）では、65,536 の異なる文字を表現できる。すべての文字を表現するために、2 バイトでは不十分な言語も多い。一部の文字には 3 または 4 バイト必要である。

たとえば、Unicode の UTF8 エンコーディングがある。UTF8 には、多数の 2 バイトおよび 3 バイトの文字がある。

また、台湾で使用する繁体字中国語もこの例の 1 つ。この言語では、80,000 以上の文字が使用される。台湾で使用されている一部の文字コード体系は、4 バイトを使用して文字をエンコードする。

関連項目：「シングルバイト（single-byte）」

マルチバイト・キャラクタ（multibyte character）

ある文字コード体系で、2 バイト以上の文字コードから構成される文字。

コード体系が異なると、同じ文字に異なる文字コードが対応する場合がある。使用している文字コード体系が不明な場合、Oracle では文字がマルチバイト・キャラクタであるかどうかを判断できない。たとえば、日本語の半角カタカナ文字は、JA16SJIS のエンコードされたキャラクタ・セットでは 1 バイト、JA16EUC では 2 バイト、UTF8 では 3 バイトである。

関連項目：「シングルバイト文字（single-byte character）」

マルチバイト・キャラクタ文字列（multibyte character string）

次のいずれかで構成される文字列。

- 文字なし（**ヌル文字列**と呼ばれる）
- 1 つ以上のシングルバイト文字
- 1 つ以上のシングルバイト文字と 1 つ以上のマルチバイト文字の混合
- 1 つ以上のマルチバイト文字

無制限多言語サポート（unrestricted multilingual support）

要求に応じて多数の言語を使用するための機能。Unicode などのユニバーサル・キャラクタ・セットを使用すると、無制限に多言語をサポートできるようになる。ユニバーサル・キャラクタ・セットでは大規模な文字レパートリをサポートしている。この文字レパートリには世界のほとんどの現代語が含まれている。

文字（character）

テキストの抽象要素。文字は、その文字の特定の表現である絵文字とは異なる。たとえば、英大文字の最初の文字は、A、A、A のように表示される。これらの形式は、同じ文字を表現する異なる絵文字である。文字、文字コードおよび絵文字には、次のような関連がある。

文字 --（エンコーディング）--> 文字コード --（フォント）--> 絵文字

たとえば、英大文字の最初の文字は、コンピュータのメモリーでは数値として表される。この数値は、**エンコーディング**または**文字コード**と呼ばれる。英大文字の最初の文字の文字コードは、ASCII コード体系では 0x41、EBCDIC コード体系では 0xc1 である。

この文字を表示または印字するには、フォントを選択する必要がある。使用可能なフォントは、使用するコード体系によって異なる。たとえば、文字を A、A または A として印字または表示できる。これらの形式は、同じ文字を表現する異なる**絵文字**である。

関連項目：「文字コード（character code）」および「絵文字（glyph）」

文字コード（character code）

特定の文字を表現する番号。この番号はコード体系によって異なる。たとえば、英大文字の最初の文字の文字コードは、ASCII コード体系では 0x41 であるが、EBCDIC コード体系では 0xc1 である。

関連項目：「文字（character）」

文字コード体系（character encoding scheme）

キャラクタ・セットのすべての文字に対して番号（文字コード）を割り当てる規則。**コード体系**、**エンコーディング・メソッド**および**エンコーディング**も、**文字コード体系**を意味する。

文字の分類（character classification）

文字の分類情報によって、各文字コードに関連付けられた文字のタイプに関する詳細が提供される。たとえば、文字には大文字、小文字、句読点、制御文字がある。

文字列（character string）

順序付きの文字グループ。

文字列に文字が含まれていない場合もある。この場合、その文字列は**ヌル文字列**と呼ばれる。ヌル文字列の文字列長は 0（ゼロ）である。

ローカライゼーション（localization）

言語固有または文化固有の情報をソフトウェア・システムに提供するプロセス。アプリケーションのユーザー・インタフェースの翻訳は、ローカライゼーションの 1 つの例である。ローカライゼーションとグローバリゼーションは異なるものである。グローバリゼーションとは、ソフトウェアを多様な言語および文化的な環境に適したものにすることである。

ロケール（locale）

特定の地域で参照される言語的および文化的な情報の集まり。一般的に、ロケールは NLS データ・ファイルに定義されている言語、地域、キャラクタ・セット、言語処理およびカレンダ情報で構成される。

ワイド・キャラクタ（wide character）

固定幅の文字形式。固定幅の領域でデータが処理されるので、大量のテキスト処理に適している。ワイド・キャラクタは、内部の文字処理をサポートすることを目的としている。

索引

数字

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

あ

い

え

お

か

き

く

け

こ

さ

し

す

せ

そ

た

ち

つ

て

と

な

は

ひ

ふ

へ

ほ

ま

む

も

ゆ