

いまさら聞けない Oracle の基本

初級編

本ドキュメントは Oracle Database を初めて使うという方向けのドキュメントです。Oracle Database のインストール方法から基礎的な SQL について解説しています。また、トランザクションやロックについても解説します。これからデータベース始める方はぜひお読みください。

1 データベースとは？



1-1. データベースとは？

まずはじめに、データベースの役割についておさらいしましょう。データベースを一言で言えば、「アプリケーションが管理する情報（データ）を集中的に整理したミドルウェア」と言えるでしょう。アプリケーションは、画面とデータがセットとなっています。たとえば、Amazonや楽天市場で代表されるECアプリケーションでは商品を選ぶ画面や注文を行う画面がありますが、その裏では商品データや会員、注文などのデータが管理されており、画面とやり取りしています。このデータ管理を専門に扱うのがデータベースの役割となります。

単純にデータを管理するだけならテキストファイルでも可能ですが、データベースが使われるのは、以下の機能を備えているためです。

1. データの不整合を防止

複数種類のデータを変更する際に、片方のデータだけが変更されて、関連する他のデータがそのままだったというような、データの不整合問題が発生することがあります。データベースではこれらを防止する機能が備わっています。

2. 高パフォーマンス

最初は順調に稼働していたアプリケーションも 10 万件、100 万件などのデータ量になると、画面表示に時間がかかってしまうなどのパフォーマンス問題がでることがあります。データベースでは、メモリを有効的に活用して大量データを高速に検索する仕組みや、ユーザーが個々にチューニングする機能が備わっています。

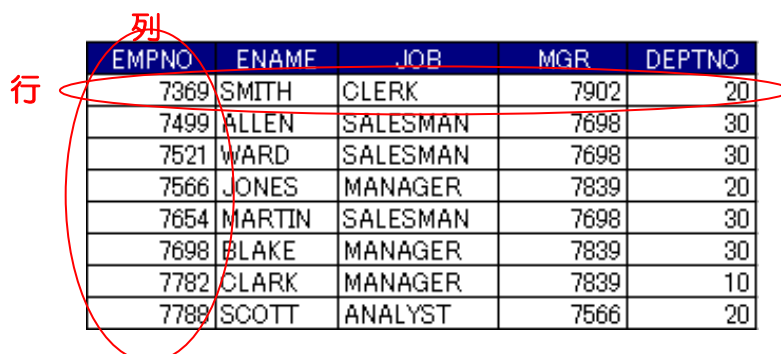
3. データの復旧

データを格納するサーバーマシンに障害が発生した場合、データが消失することがあります。データベースはバックアップファイルを定期的に自動で作成する機能があります。あらかじめバックアップを作成しておくことで、データに障害が発生した際に復旧することができます。

これらの機能は昨今のアプリケーションにおいては必要な機能ですので、データベースは必須ともいえるでしょう。具体的には 1 はトランザクション管理やロック制御機能、2 はインデックスやメモリなどのパフォーマンスチューニング、3 は自動バックアップやリカバリ機能になります。トランザクションやロックについては、6 章でご紹介します。

1-2. データベースの格納形式

昨今は列志向データベースや NoSQL など様々な形式のデータベースが登場しましたが、業務システムのバックグラウンドとして使用するデータベースは現在でも「リレーショナルデータベース形式」が主流です。（本ドキュメントでもこれ以降、リレーショナル形式データベースのことを「データベース」と呼びます。）データベースはデータを Excel のような表（テーブル）形式で管理します。表は縦軸と横軸の2次元形式で、縦軸を列（カラム）、横軸を行（ロー）といいます。縦軸はデータの属性を表します。図1の例では「社員番号」や「社員名」です。横軸はデータを表します。データの挿入や削除にあたっては、行単位で行なうことができます。



EMPNO	ENAME	JOB	MGR	DEPTNO
7369	SMITH	CLERK	7902	20
7499	ALLEN	SALESMAN	7698	30
7521	WARD	SALESMAN	7698	30
7566	JONES	MANAGER	7839	20
7654	MARTIN	SALESMAN	7698	30
7698	BLAKE	MANAGER	7839	30
7782	CLARK	MANAGER	7839	10
7789	SCOTT	ANALYST	7566	20

図1-1 リレーショナルデータベースのデータ格納形式

リレーショナルデータベースの特徴として、「社員番号」や「社員名」などの行を特定するキーをもとに、高速にデータの抽出や、複数テーブルが結合できる点があります。図1-2の例では、3つのテーブルである「社員マスタ」と「顧客マスタ」「担当顧客」の関係を表しています。社員マスタ、顧客マスタはそれぞれ社員コード、顧客コードという行ごとに一意なキーを持っており、担当顧客テーブルで、社員と、社員が担当する顧客を定義づけています。ユーザーは、この情報を元に、社員マスタと、顧客マスタを結合した状態でデータを表示することができます。

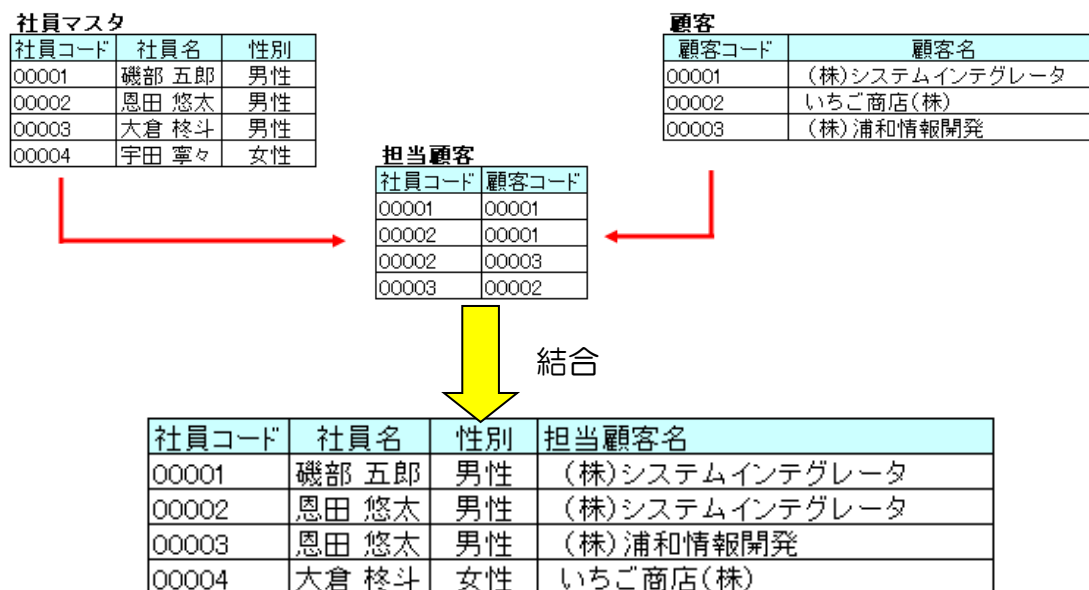


図1-2 データ結合の例

データの閲覧や編集、結合などの操作は「SQL」という専用の言語を利用します。SQL を使用することで、データベースユーザーの作成や、表などの作成/変更/削除、表にデータを追加/変更/削除などの操作が可能です。（SQL については3章以降で解説します。）

そしてリレーショナルデータベースの代表格となっているのが「Oracle Database（以下、Oracle）」です。次章からは Oracle をインストールし、サンプルの SQL を使いながらデータベース操作の基礎を学んでいきたいと思います。

2 Oracleのインストール

2-1. Oracleのインストール手順

では、Oracleをインストールしましょう。「Oracle Database 12c Enterprise Edition」を使用します。Oracle 12cからはライセンス規約が変わり、個人の開発や検証目的であれば無償で利用できるようになりました。それ以外の目的や複数人で利用する場合は有償となりますので、ご注意ください。また、インストールには6GBの空き容量が必要です。

まず、以下の手順にてOracle XEのインストーラーをダウンロードしましょう。

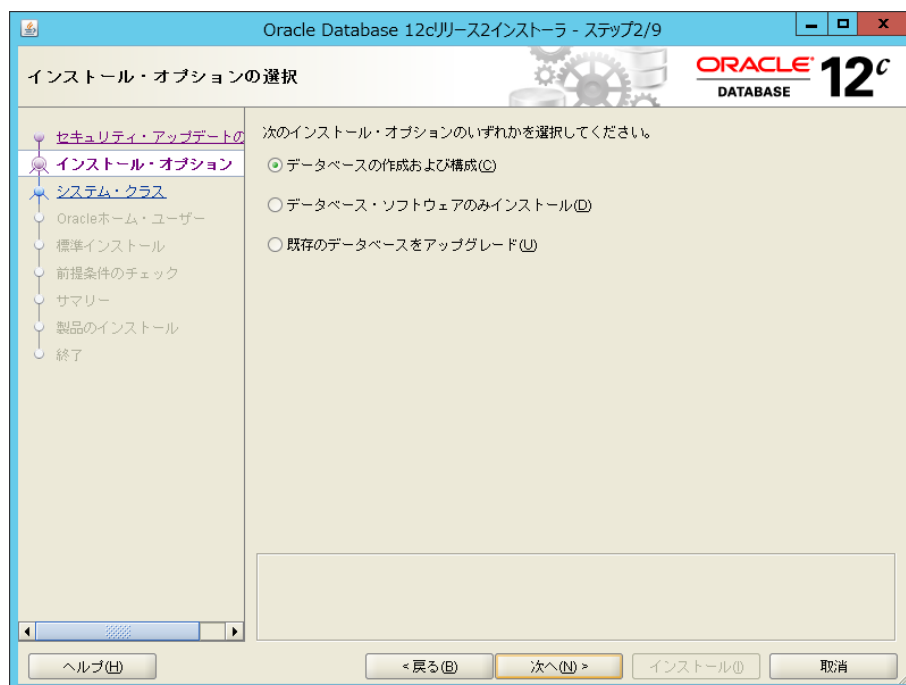
- ① Internet Explorerなどのブラウザより、Oracle Databaseのダウンロードサイト (<http://www.oracle.com/technetwork/jp/database/enterprise-edition/downloads/index.html>) にアクセスします。OTNのログイン画面が表示されます。ダウンロードにはOTNアカウントが必要となりますので、お持ちでない場合は作成してください。
- ② 本ドキュメントではWindows版を前提に説明しますので、「Standard Edition 2 and Enterprise Edition」の下にある「Microsoft Windows x64 (64-bit)」の横のリンクをクリックします。ダウンロードサイズは約2.8GBとなります。

ダウンロード後は、さっそくインストールを行います。ダウンロードしたzipファイルを解凍後、setup.exeを起動してインストーラー起動してください。なお、実行はAdministrator権限を持つユーザーで実行してください。



画面 2-1 Oracle のインストール・ウィザード画面

インストールの準備にしばらく時間がかかった後、画面2-1のようなインストール・ウィザード画面が起動します。「次へ」ボタンをクリックします。電子メールが未入力の場合警告が表示されますが、そのまま続行して問題ありません。



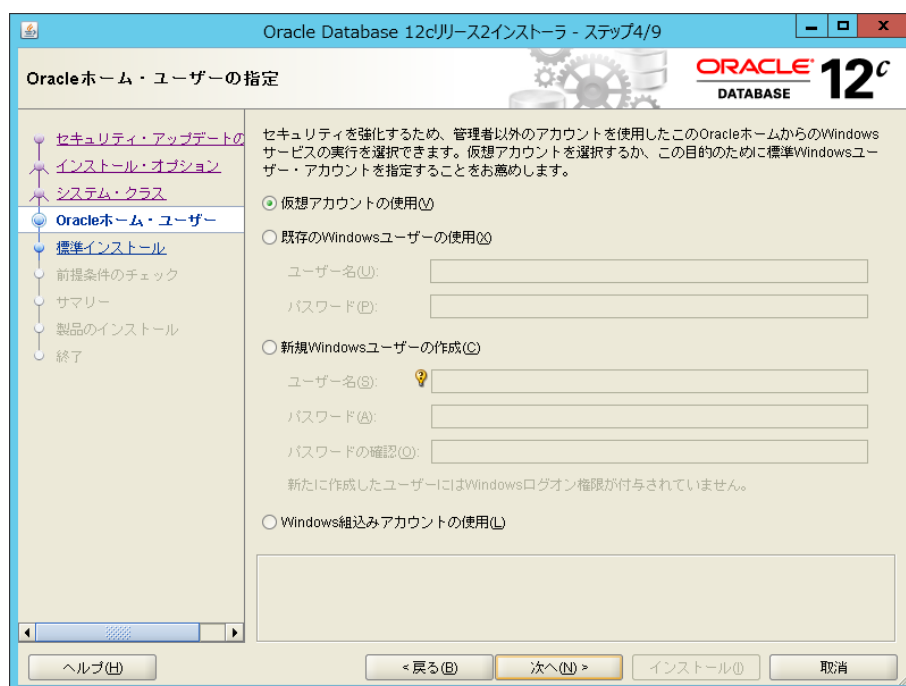
画面2-2 インストール・オプション

「データベースの作成および構成」を選択したまま「次へ」をクリックします。



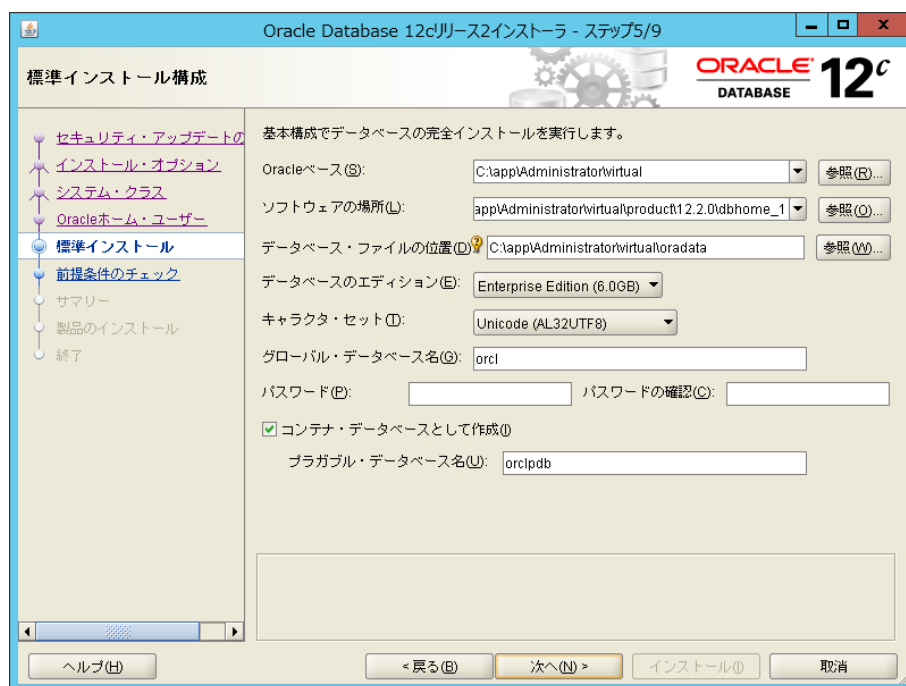
画面2-3 システム・クラスの選択

クラスを選択する画面が表示されます。本ドキュメントでは「デスクトップ・クラス」で問題ありませんので、そのまま「次へ」をクリックします。



画面 2-4 Oracle ホーム・ユーザーの設定

Oracle をバックグラウンドで起動する Windows ユーザーを指定します。「仮想アカウントの使用」のままで構いませんので「次へ」をクリックします。



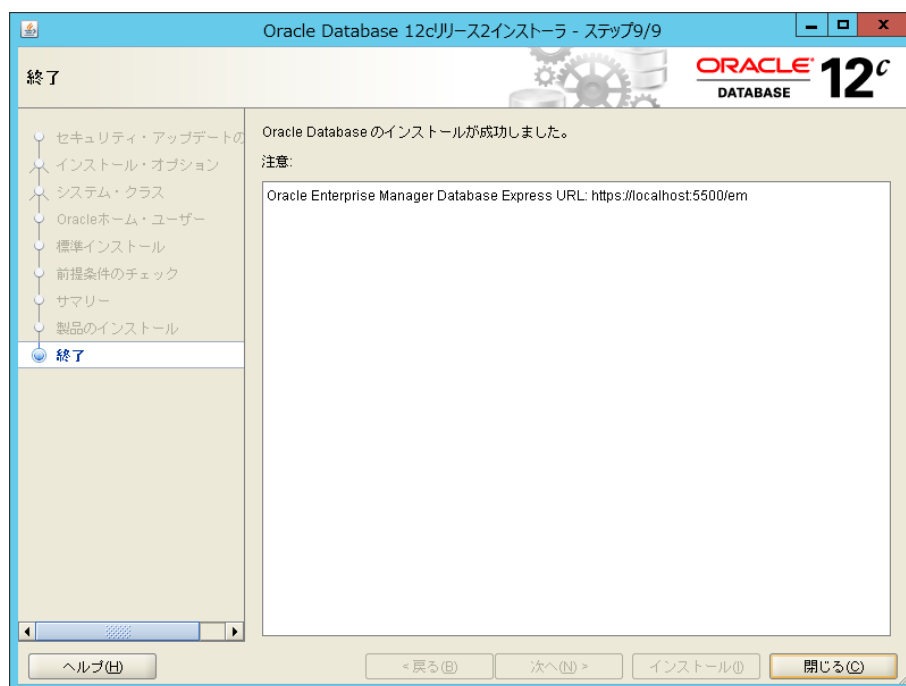
画面 2-5 インストールの設定

インストールディレクトリや管理者パスワードなどを設定します。パスワードを入力して「次へ」をクリックします。ここで入力したパスワードは忘れないようにしてください。



画面2-6 インストール確認

ディスク容量などのインストールチェックが行われた後、サマリーが表示されます。「インストール」をクリックします。



画面2-7 インストール完了

インストールが実施されます。完了まで大分時間がかかります。最後に「Oracle のインストールが成功しました」と表示されればインストール成功です。



2-2.SQLの実行

インストール完了後は、データベースの操作言語である SQL が実行できるかテストしましょう。本ドキュメントでは Oracle で標準で搭載されているツールの「SQL*Plus」を使用して SQL を実行します。SQL*Plus は Windows のコマンドプロンプト上で動作するコンソール型のツールですが、入門には十分です。Windows のコマンドプロンプトを開き「sqlplus」と入力してください。ユーザー名の入力を求められますので、「SYSTEM」でエンターキーを押します。次にパスワードを求められますので、画面 2-5 で設定したパスワードを入力します。ログインに成功すると「Oracle Database 12c Enterprise Edition Release 12.2.0.1.0 - 64bit Production に接続されました。」と表示されます。続けて以下の SQL 文を入力して、エンターキーを押してください。

SELECT SYSDATE FROM DUAL;

この SQL は現在日時を表示する SQL です。画面 2-8 のように現在日時が表示されれば成功です。

画面 2-8 SQL*Plus の画面

これで、データベースの環境構築は整いました。次章より、SQL を使用して、データベースユーザーや、テーブルを作成していきます。



SQLは最後に「; (セミコロン)」,または単独行で「/ (スラッシュ)」で実行することができます。本ドキュメントではSQLが1行の場合は「;」、複数行の場合は「/」を使う方法で統一します。

3 ユーザーとテーブルの作成

3-1.ユーザーとテーブルの概要

Microsoft Windows や Linux OS などのオペレーティングシステムと同様、データベースにもユーザー管理の仕組みがあります。データベースにおけるユーザーとはデータベースに作成するテーブルの「所有者」です。ユーザーとテーブルの関係は1対多（ユーザーは複数のテーブルを所有し、テーブルはいずれかのユーザーに所有される関係）となります。

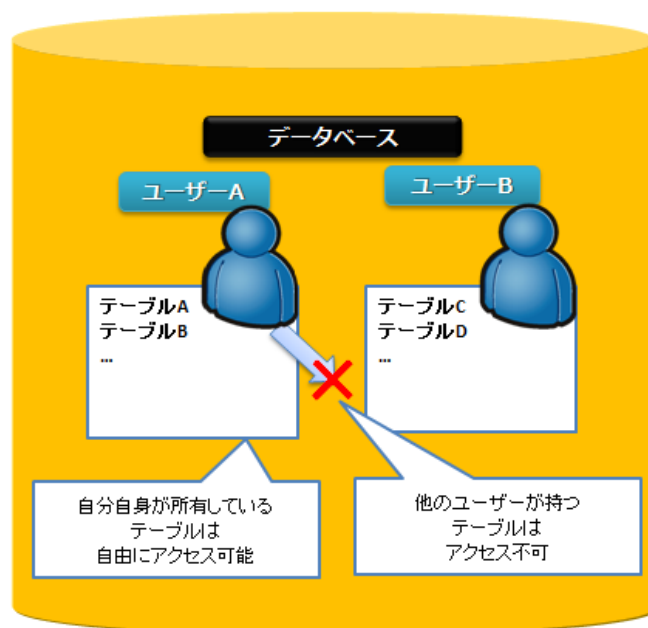


図3-1.ユーザーとテーブルの関係

ユーザーは自分が所有するテーブルを編集したり、削除したりすることはできますが、他のユーザーがもつテーブルにアクセスすることは、通常できません。



Oracleデータベースにおいては、ユーザーはテーブルを「所有する」という意味で「スキーマ」と呼ばれますが、本連載では、一般的に「データベースユーザー」または「ユーザー」という名称で統一します。

Oracle をインストール後は、標準で以下のようなユーザーが作成されています。

ユーザー名	説明
SYS	データベースの管理者ユーザー、データベースに管理されているオブジェクトなどの情報「データディクショナリ」の所有者となります。
SYSTEM	データベースの管理者ユーザーです。SYS ユーザーとの違いは、データディクショナリはもっていないこと、データベースの起動や停止が行うための「SYSDBA 権限」を所有していない点異なります。
HR	サンプル用のユーザー。サンプルテーブルとして、人事管理関係のテーブルが格納され

	ています。
PUBLIC	すべてのユーザーからアクセスを実現するための特殊なユーザーです。「パブリックシノニム」と呼ばれる機能を使用することが、すべてのユーザーがアクセスできるオブジェクトを作成することができます。 ※パブリックシノニム詳しくは次回以降で説明します。

表3-1. データベースインストール後に作成される代表的なユーザー

その他にも多くのユーザーが作成されていますが、データベースを使用したアプリケーション開発を行う際は、アプリケーション専用のユーザーを作成するのが一般的です。「SYS」や「SYSTEM」などの標準ユーザーでも同様にテーブルを作成することは可能ですが、これらのユーザーは OS でいうところの「Administrator」ユーザーとなり、他ユーザーのテーブルが編集できるなど、強力な権限をもっているため、セキュリティ上、これらのユーザーを使うことは望ましくありません。そこで、専用ユーザーを作成し、そのユーザーの下にテーブルを作成するのが一般的です。



3-2. ユーザーの作成

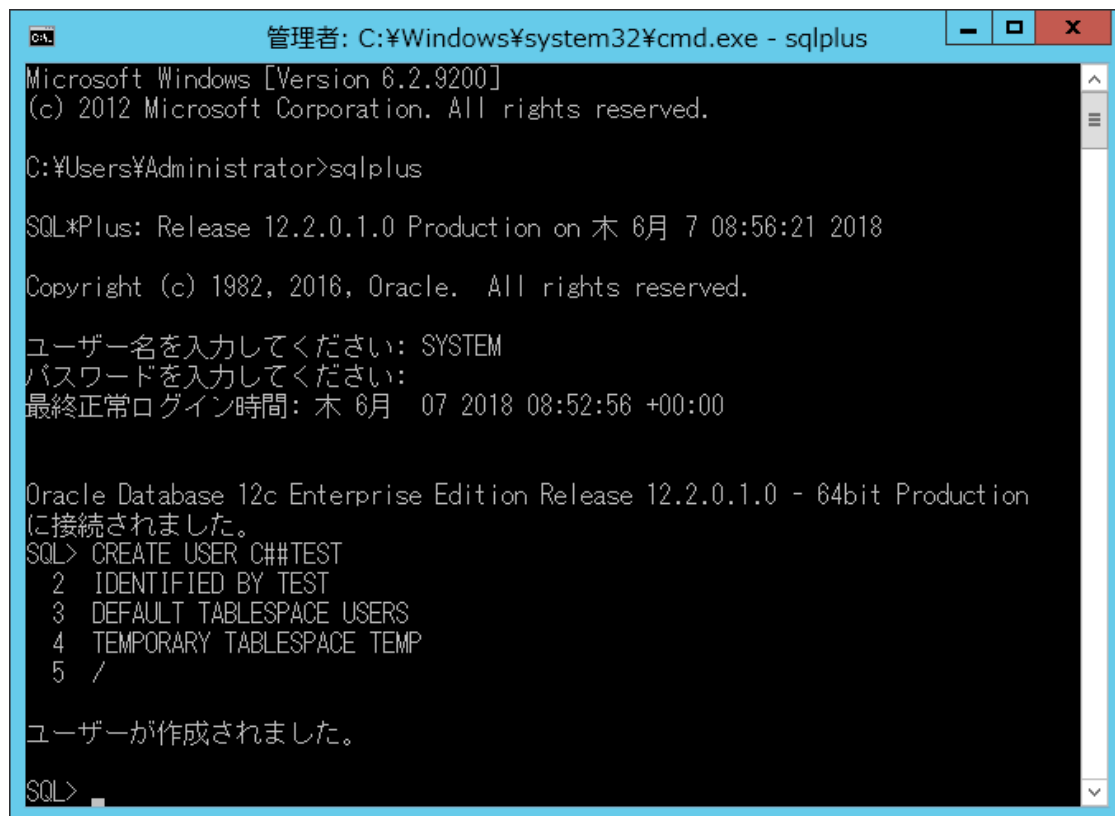
それでは、専用ユーザーを作成しましょう。SQL*Plus にて「SYSTEM」ユーザーでログインしてください。ユーザーを作成する SQL には「CREATE USER 文」と呼ばれる SQL を使用します。CREATE USER 文の書式は以下の通りです。

```
CREATE USER C##TEST
IDENTIFIED BY TEST
DEFAULT TABLESPACE USERS
TEMPORARY TABLESPACE TEMP
/
```

「CREATE USER」に続きユーザー名を入力します。基本的には任意となりますが、Oracle12c を既定の設定でインストールした場合は、コンテナデータベースという種類のデータベースとなる関係で、ユーザー名は「C##」で始まるユーザ名である必要がありますのでご注意ください。上記 SQL でも「C##TEST」としています。「IDENTIFIED BY」に続きパスワードを指定します。今回の例では「TEST」としました。

「DEFAULT TABLESPACE」、「TEMPORARY TABLESPACE」でそれぞれデフォルト表領域、一時表領域を指定しています。

こちらもユーザー画面と同様にデフォルト表領域に「USERS」、一時表領域に「TEMP」を指定しています。SQL を実行し、「ユーザーが作成されました。」と表示されれば成功です。エラーになる場合は、スペルミスなど、記述に誤りがないか確認・修正の上、再度実行してください。



```
管理者: C:\Windows\system32\cmd.exe - sqlplus
Microsoft Windows [Version 6.2.9200]
(c) 2012 Microsoft Corporation. All rights reserved.

C:\Users\Administrator>sqlplus

SQL*Plus: Release 12.2.0.1.0 Production on 木 6月 7 08:56:21 2018

Copyright (c) 1982, 2016, Oracle. All rights reserved.

ユーザー名を入力してください: SYSTEM
パスワードを入力してください:
最終正常ログイン時間: 木 6月 07 2018 08:52:56 +00:00

Oracle Database 12c Enterprise Edition Release 12.2.0.1.0 - 64bit Production
に接続されました。
SQL> CREATE USER C##TEST
 2  IDENTIFIED BY TEST
 3  DEFAULT TABLESPACE USERS
 4  TEMPORARY TABLESPACE TEMP
 5  /

ユーザーが作成されました。

SQL>
```

画面3-5 SQL コマンド画面

これで、ユーザーは作成されましたが、まだ権限の設定を行っていないため、このユーザーでログインすることはできません。権限を付与するには、「GRANT 文」と呼ばれる SQL を実行します。

GRANT CONNECT, RESOURCE, UNLIMITED TABLESPACE TO C##TEST;

「GRANDT」に続き与える権限の名前を指定します。複数の権限を指定する場合は、「,」でつなぎます。上記 SQL では「CONNECT」と「RESOURCE」「UNLIMITED TABLESPACE」のロール権限を指定しています。これにより、ログインやテーブル作成が可能となります。権限名の後は「TO」をはさみ、権限を与える対象のユーザーを指定します。上記 SQL では C##TEST ユーザーに指定しています。「権限付与が成功しました」と表示されれば成功です。

ユーザーの作成完了後は、このユーザーでログインができるか確認しましょう。現在は「SYSTEM」ユーザーでログインされている状態ですので、「exit」と入力してログアウトします。再度「sqlplus」と入力し、作成したユーザー名とパスワードでログインできればユーザーの作成は完了です。



ユーザー名、パスワードは大文字小文字を区別しますので注意が必要です。CREATE USER文でユーザー名やパスワード大文字で作成した場合は、ログイン時も大文字を入力してください。

3-3.テーブルの作成

ユーザー作成後は、引き続きテーブルを作成しましょう。今回はサンプル用に社員情報を管理するテーブルを作成します。テーブルを作成するには、大きくテーブル定義を作る、データを作成するという2つのステップに分かれます。テーブルの定義とは、「どのような種類のデータ、いくつ管理するのか」という枠組みに関する情報です。これには、「CREATE TABLE」文を使用します。テーブルを作成するには、ユーザーの作成でも使用した SQL コマンド画面(「SQL」アイコンの右側にある下向き三角形を押し、「SQL コマンド」→「コマンド入力」)より、以下の SQL を入力してください。

CREATE TABLE 社員マスタ

```
(
    社員番号          NUMBER(10,0),
    社員名            VARCHAR2(50),
    性別              VARCHAR2(6),
    入社日            DATE
)
```

「CREATE TABLE」に続けてテーブルの名前を指定します。同一ユーザーの範囲で他のテーブルと重複しない名前を指定する必要があります。その後、括弧で括り、その中に列の情報を記述します。「社員番号」「社員名」「性別」「入社日」4 列構成としています。このように複数の列を使う場合はそれぞれをカンマで区切ります。また、「列名」の後には「データ型」を指定する必要があります。「データ型」は、格納されるデータの種類を表す情報です。大きくは「文字型」、「数値型」、「日付型」にわかれます。代表的なデータ型は以下の通りです。

種類	データ型	説明
文字型	CHAR VARCHAR2	文字列を格納する場合に使用します。文字型の場合は、括弧でデータの最大の長さ指定します。単位はバイトです。 CHAR の場合、長さに満たない文字は空白で置き換えられます。 VARCHAR2 は可変型です。指定した長さよりも短い文字列を格納しても空白で埋められませんので、効率よくデータを格納できます。
数値型	NUMBER	数値データを格納する場合に使用します。長さは「NUMBER (p,s)」の形式で指定します。全体の桁数を p 桁、小数点以下の s 桁となります。精度 p には 1～38 の値を指定でき、小数点の右側にある桁数には、-84～127 の値を指定できます。
日付型	DATE TIMESTAMP	日付データを「年/月/日 時刻:分:秒」の形式で管理することができます。格納できるデータは紀元前 4712 年 1 月 1 日から、9999 年 12 月 31 日までです。TIMESTAMP を使用すると、さらに 1 秒未満の値も管理できます。日付型の場合は長さの指定は不要です。

表 3-2. データの種類と代表的なデータ型

上記以外にも、Unicode に対応した「NCHAR」「NVARCHAR2」などのデータ型があります。今回のテーブルでは「社員名」と「性別」に文字型の「VARCHAR2」、「社員番号」に数値型である「NUMBER」、「入社日」を日付型である「DATE」を使用しています。列情報が記載できれば SQL は完成ですので、SQL

を実行してみましょう。「表が作成されました」と表示されれば成功です。

テーブル作成後は、実際にデータを確認してみましょう。「SELECT 文」でデータの確認を行うことができます。以下の SQL を実行してみましょう。

```
SELECT * FROM 社員マスタ;
```

「SELECT」の後に、表示する列名を指定します。列が複数ある場合は列を「,(カンマ)」で区切ります。

「*(アスタリスク)」を指定すると、テーブルが持つすべての列を表示することができます。「FROM」に続き対象のテーブルを指定します。ただし、この時点ではテーブルの定義（枠組）しか作成していないため、データは作成されていません。そのため、「レコードが選択されませんでした。」と表示されます。



3-4.データの作成

データを作成するには「INSERT 文」を使用します。データを 1 件登録するための SQL は以下のようになります。

```
INSERT INTO 社員マスタ VALUES( 1,'川村 健二','男性','2018/01/01');
```

「INSERT INTO」に続き、登録したい対象のテーブル名を指定します。さらに VALUES 句が続きます。VALUES 句には INTO 句で指定した列の順番に、登録する値をカンマ区切りで並べます。文字列値の場合はシングルクォーテーションで囲む必要があります。上記 SQL を実行すると「文が処理されました」と表示されれば成功です。また、複数のデータをまとめて登録したい場合は、以下の SQL で可能です。

```
INSERT ALL
```

```
INTO 社員マスタ VALUES( 2,'勝田 芽衣','女性','2002/06/21')
```

```
INTO 社員マスタ VALUES( 3,'塚本 和希','女性','2005/05/21')
```

```
INTO 社員マスタ VALUES( 4,'前田 秀喜','男性','2004/02/12')
```

```
INTO 社員マスタ VALUES( 5,'永井 和真','男性','2002/01/12')
```

```
INTO 社員マスタ VALUES( 6,'川上 秀喜','男性','2006/12/01')
```

```
INTO 社員マスタ VALUES( 7,'横山 彩夏','女性','2004/07/24')
```

```
INTO 社員マスタ VALUES( 8,'半田 幸雄','男性','2009/03/26')
```

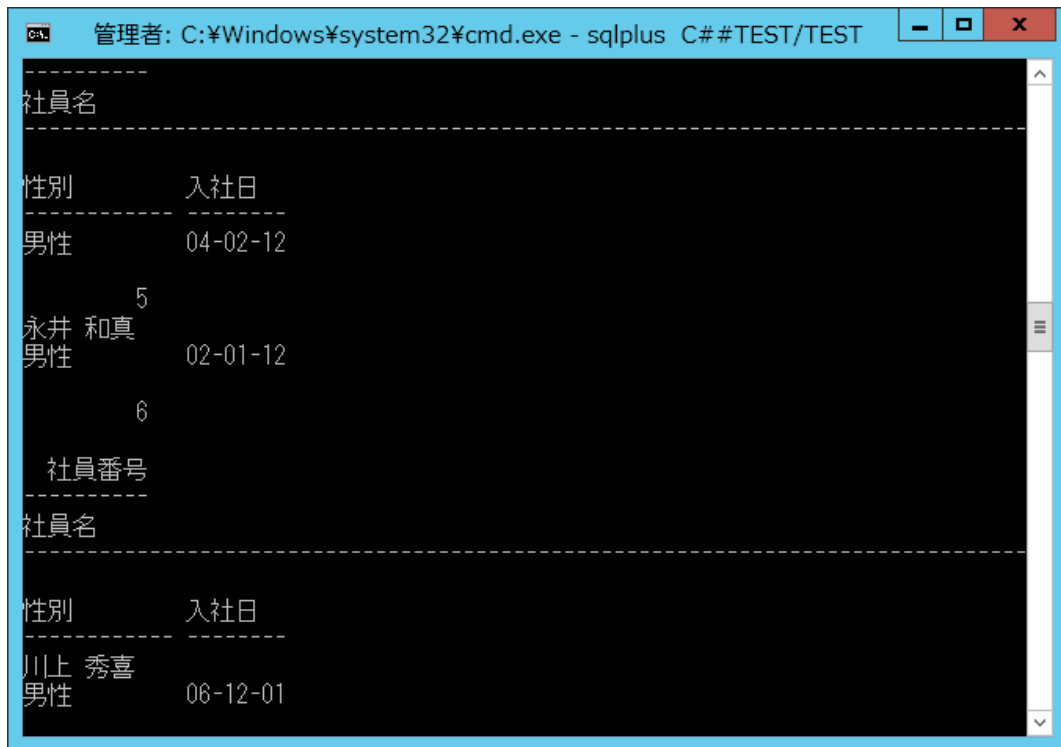
```
INTO 社員マスタ VALUES(9,'三井 雅也','女性','2006/01/27')
```

```
INTO 社員マスタ VALUES(10,'奈良 美穂','女性','2004/07/08')
```

```
SELECT * FROM DUAL
```

/

登録後は、再度 SELECT 文でデータを確認してください。先ほど登録した全てのデータが表示されます。



画面3-6 SELECT 文の実行結果（整形前）

ただ、既定の設定のままですと SELECT 結果が見づらくなっています。以下のコマンドを入力してください。

```
SET LINESIZE 10000
```

このコマンドは SQL 文ではなく、SQL*Plus のオプション文です。一行の最大表示バイト数を 10000 バイトにしています。上記コマンドを実行後、再度 SELECT 結果を実行すると非常に見やすくなっているのがわかります。

```
管理者: C:\Windows\system32\cmd.exe - sqlplus C##TEST/TEST
SQL> SET LINESIZE 10000
SQL> SELECT * FROM 社員マスタ
2 /

社員番号 社員名
-----
1 川村 健二
2 勝田 芽衣
3 塚本 和希
4 前田 秀喜
5 永井 和真
6 川上 秀喜
7 横山 彩夏
8 半田 幸雄
9 三井 雅也
10 奈良 美穂

性別
-----
男性
女性
女性
男性
男性
男性
女性
男性
女性
女性

入社日
-----
18-01-01
02-06-21
05-05-21
04-02-12
02-01-12
06-12-01
04-07-24
09-03-26
06-01-27
04-07-08

10行が選択されました。
SQL> _
```

画面3-7 SELECT 文の実行結果（整形後）



3-5.データの更新・削除

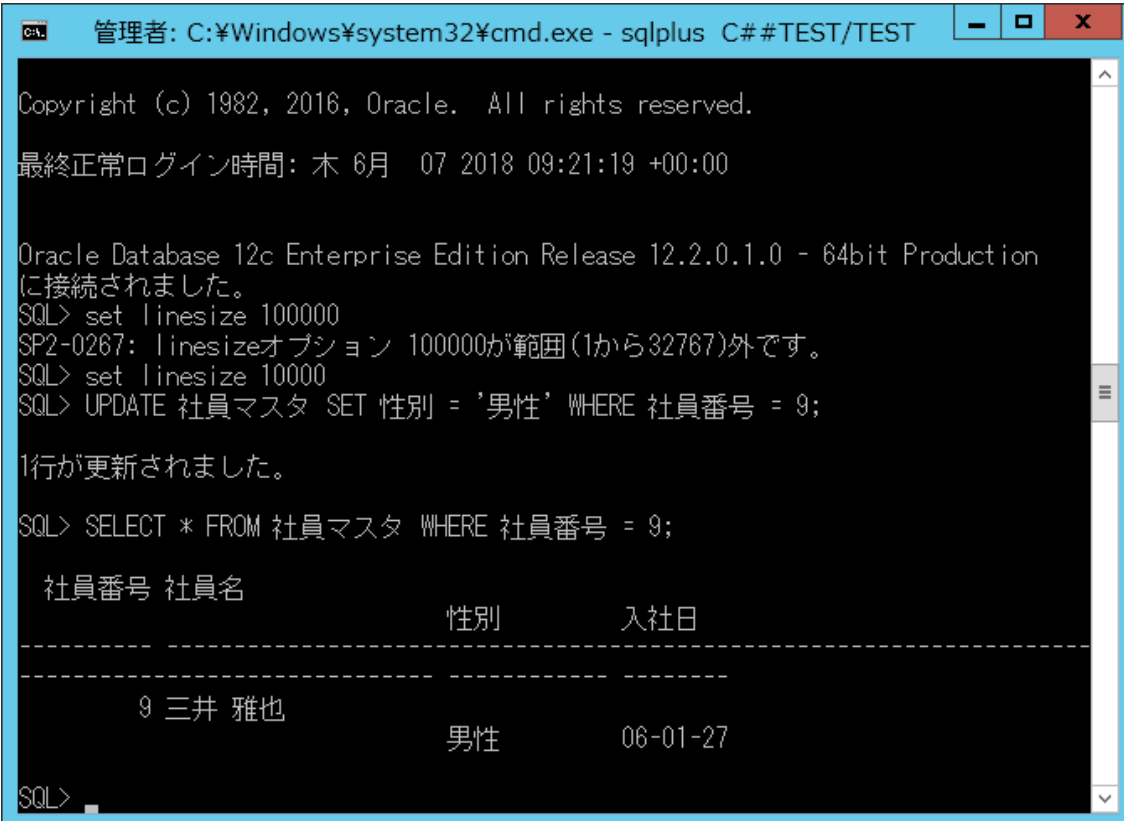
では、続けて既に登録したデータを変更する SQL を実行してみましょう。データを更新するための SQL は以下ようになります。

```
UPDATE 社員マスタ SET 性別 = '男性' WHERE 社員番号 = 9;
```

「UPDATE」の後にテーブル名を指定し、さらに、「SET」の後に更新する対象の列名を指定します。「=」の後に更新後の値です。また、「WHERE」に引き続きでは更新対象となるレコードの条件を指定します。この検索条件に一致する行に対して、指定された列に指定された値で更新が行なわれることになります。「WHERE」以降は省略することも可能です。WHERE 移行がない場合は、指定されたテーブルの全ての

レコードが更新されることになります。この例では、社員番号が「9」の人に対して、性別を男性に更新しています。SQL を実行すると「1 行が更新されました。」と表示されれば成功です。実際に更新されたかを SELECT 文で確認してみましょう。今度は、以下の SELECT 文を実行してください。

```
SELECT * FROM 社員マスタ WHERE 社員番号 = 9;
```



```
Copyright (c) 1982, 2016, Oracle. All rights reserved.
最終正常ログイン時間: 木 6月 07 2018 09:21:19 +00:00

Oracle Database 12c Enterprise Edition Release 12.2.0.1.0 - 64bit Production
に接続されました。
SQL> set linesize 100000
SP2-0267: linesizeオプション 100000が範囲(1から32767)外です。
SQL> set linesize 10000
SQL> UPDATE 社員マスタ SET 性別 = '男性' WHERE 社員番号 = 9;

1行が更新されました。

SQL> SELECT * FROM 社員マスタ WHERE 社員番号 = 9;

社員番号 社員名
-----
9 三井 雅也

性別      入社日
-----
男性      06-01-27

SQL>
```

画面 3-8 SELECT 文の実行結果

上記 SQL は SELECT 文に「WHERE」を使用して更新した対象のレコードのみを表示しています。このように、SELECT 文でも WHERE を指定して対象レコードを絞り込んで表示することが可能です。最後に、データを削除するための SQL は以下になります。

```
DELETE FROM 社員マスタ WHERE 入社日付 < '2004/12/31';
```

「DELETE FROM」の後に対象となるレコードを指定します。「WHERE」の後には、UPDATE 文と同様に、削除対象となるレコードの条件を指定します。この例では入社日付が 2004 年 12 月 31 日以前の人を削除対象としています。WHERE 句では「=」以外にも、不等号や、「<>」で不一致の指定をすることが可能です。なお、WHERE 移行を省略した場合は全レコードが削除されますので、誤って記述漏れしないよう、気を付けてください。上記 SQL を実行すると、「5 行が削除されました」と表示されます。DELETE 文を実行した場合は、削除した行数がメッセージで表示されます。

以上で、基本的な SQL をご紹介しました。最初 SQL の書き方は覚えるのは手間ですが、いずれも頻繁に使用する SQL ですので、ぜひ覚えてください。これらの SQL を使用すれば、自由にテーブルの作成や、データの管理が可能になります。ただし、実際のシステム開発においては、複数のテーブルデータを関連付けて SELECT や、データを誤って更新してしまった場合に元に戻すといったような、さらに応用の技術が必要となります。この方法は第6章で詳しくご紹介します。

4 データの結合と集計

4-1. データの結合と集計方法

実際のシステム開発においては、テーブルのデータをそのまま取得するだけでなく、データをさらに組み合わせることで情報を求めたい場合があります。

Amazon のような EC サイトの管理者を例に考えてみましょう。通常、商品や顧客といった情報がテーブル管理されていますので、これらのテーブルデータを参照して、すべての商品の価格や、注文データを取得することは可能です。ただ、それらの個々のテーブルを見ただけでは、以下のような情報を得ることはできません。

- 一番顧客ごとの購入金額は？
- 一番売れている商品は？

顧客マスタ

顧客コード	顧客名	性別
1	甲斐 敏和	男性
2	市川 琢郎	男性
3	松木 春奈	女性
4	田川 太陽	男性
5	岩瀬 圭吾	男性

商品マスタ

商品コード	商品名	単価	在庫数
1	Tシャツ	2000	20
2	ジーンズ	12000	15
4	ジャケット	15000	10
5	バッグ	7500	5
3	ニット帽	2000	1

注文

注文番号	顧客コード	商品コード	数量
1	4	4	50
2	2	1	80
3	2	2	20
4	3	1	50
5	4	1	30
6	4	3	50
7	2	2	50
8	1	1	70
9	2	3	10
10	2	4	40



図4-1. 様々な要件

これらの情報を求めるには、それぞれのテーブルデータを見ながら、それらを突き合わせる必要があります。ただ、この作業をユーザーが行うのは一苦労です。Oracle などのデータベースでは、複数のテーブルを結合する機能が提供されており、複数のテーブルを組み合わせることが可能です。たとえば、各注文ごとに顧客名と購入金額を求めるには、「商品マスタ」や「顧客マスタ」「注文テーブル」のデータを結合することで可能です。

顧客マスタ

顧客コード	顧客名	性別
1	甲斐 敏和	男性
2	市川 琢郎	男性
3	松本 春奈	女性
4	田川 太陽	男性
5	岩瀬 圭吾	男性

商品マスタ

商品コード	商品名	単価	在庫数
1	Tシャツ	2000	20
2	ジーンズ	12000	15
4	ジャケット	15000	10
5	バッグ	7500	5
3	ニット帽	2000	1

注文

注文番号	顧客コード	商品コード	数量
1	4	4	50
2	2	1	80
3	2	2	20
4	3	1	50
5	4	1	30
6	4	3	50
7	2	2	50
8	1	1	70
9	2	3	10
10	2	4	40



注文番号	顧客名	単価	数量
1	田川 太陽	15000	50
2	市川 琢郎	2000	80
3	市川 琢郎	12000	20
4	松本 春奈	2000	50
5	田川 太陽	2000	30
6	田川 太陽	2000	50
7	市川 琢郎	12000	50
8	甲斐 敏和	2000	70
9	市川 琢郎	2000	10
10	市川 琢郎	15000	40

複数のテーブルから
データを組み合わせて取得

図4-2. 結合のイメージ

実際に SQL を実行して動作を確認してみましょう。まず、サンプル用にテーブルを作成します。SQL*Plus にて以下の SQL をそれぞれ記述し、実行してください。

(1)注文テーブルを作成

```
CREATE TABLE 注文
(
    注文番号          VARCHAR2(5) NOT NULL,
    顧客コード        NUMBER(5,0) NOT NULL,
    商品コード        NUMBER(5,0) NOT NULL,
    数量              NUMBER(5,0) NOT NULL
)
```

(2)顧客マスタを作成

```
CREATE TABLE 顧客マスタ
(
    顧客コード        NUMBER(5,0) NOT NULL,
    顧客名            VARCHAR2(20) NOT NULL,
    性別              VARCHAR2(8) NOT NULL
)
```

(3)商品マスタを作成

```
CREATE TABLE 商品マスタ
```

```
(
  商品コード          NUMBER(5,0) NOT NULL,
  商品名              VARCHAR2(20) NOT NULL,
  単価                NUMBER(8,0) NOT NULL,
  在庫数              NUMBER(5,0) NOT NULL
)
/
```

(4) 注文マスタのデータを作成

```
INSERT ALL
  INTO 注文 VALUES('00001',4,4,50)
  INTO 注文 VALUES('00002',2,1,80)
  INTO 注文 VALUES('00003',2,2,20)
  INTO 注文 VALUES('00004',3,1,50)
  INTO 注文 VALUES('00005',4,1,30)
  INTO 注文 VALUES('00006',4,3,50)
  INTO 注文 VALUES('00007',2,2,50)
  INTO 注文 VALUES('00008',1,1,70)
  INTO 注文 VALUES('00009',2,3,10)
  INTO 注文 VALUES('00010',2,4,40)
  SELECT * FROM DUAL
/
```

(5)顧客マスタのデータを作成

```
INSERT ALL
  INTO 顧客マスタ VALUES(2,'市川 琢郎','男性')
  INTO 顧客マスタ VALUES(3,'松木 春奈','女性')
  INTO 顧客マスタ VALUES(1,'甲斐 敏和','男性')
  INTO 顧客マスタ VALUES(4,'田川 太陽','男性')
  INTO 顧客マスタ VALUES(5,'岩瀬 圭吾','男性')
  SELECT * FROM DUAL
/
```

(6)商品マスタのデータを作成

```
INSERT ALL
  INTO 商品マスタ VALUES(1,'Tシャツ',2000,20)
  INTO 商品マスタ VALUES(2,'ジーンズ',12000,15)
  INTO 商品マスタ VALUES(4,'ジャケット',15000,10)
  INTO 商品マスタ VALUES(3,'ニット帽',2000,1)
  INTO 商品マスタ VALUES(5,'バッグ',7500,5)
```

```
SELECT * FROM DUAL
```

顧客マスタと商品マスタができた後は、実際に結合操作を行ってみましょう。
以下の SQL を実行して下さい。

```
SELECT 注文.注文番号, 顧客マスタ.顧客名, 商品マスタ.単価 * 注文.数量
FROM 注文, 顧客マスタ, 商品マスタ
WHERE 注文.顧客コード = 顧客マスタ.顧客コード
      AND 注文.商品コード = 商品マスタ.商品コード
ORDER BY 注文.注文番号
```

結合を行うには、SELECT 文の FROM 句で対象のテーブルをカンマ区切りで指定し、さらに WHERE 句にて「結合条件」を指定します。WHERE 句は通常、「商品マスタ.金額 > 1000」といったようにレコードを絞り込むための条件を記述する個所ですが、複数テーブルの結合を行う場合は、それぞれのキーとなる列を結びつけます。今回の例では、「注文」テーブルにある顧客、商品に関連付けるための「顧客コード」「商品コード」がありますので、これらの列を顧客マスタ、商品マスタのそれぞれの同列にイコールで結んでいます。



WHERE 句の結合条件を指定せず、実行してもエラーにはなりませんが、結合条件を指定しない場合、各テーブルのレコード数をかけあわせた数だけ、結果レコードが表示されてしまいます。FROM 句に複数テーブルを指定した場合は、テーブル同士を WHERE 句で結び付けるのを忘れないようにしましょう。

複数のテーブルを FROM で指定している場合、SELECT 句や WHERE 句では「テーブル名.列名」と記述する必要があります。これは、もし、異なるテーブルに同じ列名が使われている場合、どのテーブルの列を表示するのかをきちんと指定しなければいけないためです。また、SELECT 句では、商品マスタの「単価」と、注文マスタで管理されている「数量」を掛け算によって求めています。このように SELECT 句に計算式を記述することで、その計算結果を表示することもできます。

また、「ORDER BY」に続けて対象の列を指定することで、その列のデータをもとに並び替えて表示することができます。この SQL では「ORDER BY 注文番号」と記述していますので、注文番号順に並び替えて表示されます。「ORDER BY 注文番号 DESC」と指定して、注文番号の大きい順に並び替えることもできます。

なお、Oracle 以外のデータベースにて結合の SQL を記載する場合は、以下の SQL にする必要があります。上記 Oracle 用の SQL の FROM 句でカンマ区切りで複数テーブルをしているする代わりに、「INNER JOIN 結合するテーブル名」とし、「ON」に続けて結合キーをイコールで結んでいます。

```
SELECT 注文.注文番号, 顧客マスタ.顧客名, 商品マスタ.単価 * 注文.数量
FROM 注文
INNER JOIN 顧客マスタ ON 注文.顧客コード = 顧客マスタ.顧客コード
INNER JOIN 商品マスタ ON 注文.商品コード = 商品マスタ.商品コード
ORDER BY 注文.注文番号
```

SQL が記述できた後は実際に実行してください。それぞれのテーブルが結合され、画面 1 のような結果が表示され、注文ごとの顧客と、商品の購入金額が表示されます。

```
管理者: C:\Windows\system32\cmd.exe - sqlplus C##TEST/TEST
10行が選択されました。

SQL> SELECT 注文.注文番号, 顧客マスタ.顧客名, 商品マスタ.単価 * 注文.数量
2 FROM 注文
3 INNER JOIN 顧客マスタ ON 注文.顧客コード = 顧客マスタ.顧客コード
4 INNER JOIN 商品マスタ ON 注文.商品コード = 商品マスタ.商品コード
5 ORDER BY 注文.注文番号
6 /

注文番号 顧客名 商品マスタ.単価*注文.数量
-----
00001 田川 太陽 750000
00002 市川 琢郎 160000
00003 市川 琢郎 240000
00004 松木 春奈 100000
00005 田川 太陽 60000
00006 田川 太陽 100000
00007 市川 琢郎 600000
00008 甲斐 敏和 140000
00009 市川 琢郎 20000
00010 市川 琢郎 600000

10行が選択されました。

SQL>
```

画面 4-1. 結合の実行結果

4-2. 外部結合と集計関数

さきほどの SQL で各注文の顧客名・購入金額を確認しました。しかし、「顧客ごとに 1 番売れている金額を求めたい場合」はこの SQL では少し不十分です。まず、同一顧客で複数のレコードが表示されているため、顧客ごとの金額をひとめで判断することができません。また、よくみると「岩瀬 圭吾」さんのデータが 1 件も表示されていません。この原因は注文テーブルに岩瀬さんの注文が 1 件もないためですが、このように注文がない顧客も含めて結果を表示したい場合もあるでしょう。これらの問題に対しては、外部結合、集計関数を使うことで改善できます。以下のように SELECT 文を変更します。

```
SELECT C.顧客名, SUM(M.単価 * O.数量) AS 購入金額
FROM 注文 O, 顧客マスタ C, 商品マスタ M
WHERE O.顧客コード(+) = C.顧客コード
AND O.商品コード = M.商品コード(+)
GROUP BY C.顧客名
ORDER BY 購入金額 DESC
/
```

WHERE 句では、「顧客コードの(+)」がついています。このように指定すると、注文テーブルになく、顧客マスタにしかないデータも結合することができます。(+) 記号は右辺か左辺のどちらの列に記述します。今回のように、顧客マスタにはあるが、注文にない場合は、注文テーブル側の顧客コードに「(+)」を付与します。



商品コードにも(+)が付与されていますが、顧客コードの外部結合の結果、商品コードがの

結合が思い通りにデータが表示されないためです。あまり気にせず読み進めてください。

この SQL の実行結果は以下の通りです。「金額」列は空白（NULL）で、「岩瀬 圭吾」さんのレコードも表示されています。

なお、Oracle 以外のデータベースにて結合の SQL を記載する場合は、以下の SQL となります。上記 Oracle 用の SQL の WHERE 句にある (+) の記号が左側にある場合は「LEFT OUTER JOIN」、右側にある場合は、「RIGHT OUTER JOIN」に置き換えます。「ON」に続けて結合キーを指定します。

```
SELECT C.顧客名, SUM(M.単価 * O.数量) AS 購入金額
FROM 注文 O
RIGHT OUTER JOIN 顧客マスタ C ON O.顧客コード = C.顧客コード
LEFT OUTER JOIN 商品マスタ M ON O.商品コード = M.商品コード
GROUP BY C.顧客名
ORDER BY 購入金額 DESC
/
```

また、Oracle などのデータベースには合計金額や、注文件数などの集計結果を求めるために「集計関数」が用意されています。任意の列をキーをもとに集計した結果を求めることができます。集計関数を使用するにはまず、WHERE 句の後に「GROUP BY 集計列」の形式で記述します。「集計列」には、どの列をもとに集計するのかという列名を指定します。今回は顧客ごとに集計した結果を求めたいため「顧客名」を指定しています。

さらに、SELECT 句で集計関数と括弧の中に対象列を指定します。集計関数の種類は以下の通りです。

種類	説明
SUM	指令した列の合計を求めることができます。数値型のみ使用できます。
AVERAGE	指令した列の平均値を求めることができます。数値型のみ使用できます。
MAX	集計した中で一番大きな値を求めることができます。文字型や日付型でも使用可能です。文字型の列を指定して使用した場合はアルファベット順で並び替えた場合に最後にくる文字列が表示されます。
MIN	集計した中で一番小さな値を求めることができます。文字型や日付型でも使用可能です。文字型の列を指定して使用した場合はアルファベット順で並び替えた場合に最初にくる文字列が表示されます。
COUNT	件数を求めることができます。NULL のデータは除かれた件数となります。 * (アスタリスク) を指定した場合は NULL も含めたレコード件数となります。

表 4-1. 集計関数の種類

今回の SQL では SELECT 句で「SUM（購入金額）」としていますので、顧客ごとの合計購入金額を求めることができます。

また、SELECT にて「列名 AS 別名」と指定することで、テーブルの列名を別名として結果表示することができます。購入金額については、2 つの列を計算によって求めています。実行結果には計算式がそのまま表示されてしまうため、別名を使って「購入金額」に変更しています。別名を使用する場合は「AS」を省略し、「列名 別名」と記述でも構いません。



Oracle 以外のデータベースの場合は「列名 AS 別名」と書かないとエラーになることがありますのでご注意ください。

別名はテーブルに使用することも可能です。今回の SQL でもそれぞれのテーブルをそれぞれ「O」「C」「M」と別名を付けています。複数テーブルを結合する場合、SQL の各列名の個所は「テーブル名.列名」と記載する必要がありますが、テーブルの別名を使うと「C.顧客名」というようにテーブル名の部分に別

名を使うことができるようになり、SQL 文の記述が楽になります。

また、この SQL では「ORDER BY SUM(購入金額) DESC」と記述していますので、合計金額の大きい順に並び替えて表示されます。ORDER BY 句には通常、対象の列名を指定しますが、このように、別名や、集計関数を指定しても問題ありません。

SQL が記述できたら、実際に実行してください。画面 4-2 のような結果が表示され、顧客ごとに購入金額の合計が表示されました。

```
管理者: C:\Windows\system32\cmd.exe - sqlplus C##TEST/TEST
00006 田川 太陽 100000
00007 市川 琢郎 600000
00008 甲斐 敏和 140000
00009 市川 琢郎 20000
00010 市川 琢郎 600000

10行が選択されました。

SQL> SELECT C.顧客名, SUM(M.単価 * O.数量) AS 購入金額
2 FROM 注文 O, 顧客マスタ C, 商品マスタ M
3 WHERE O.顧客コード(+) = C.顧客コード
4 AND O.商品コード = M.商品コード(+)
5 GROUP BY C.顧客名
6 ORDER BY 購入金額 DESC
7 /

顧客名 購入金額
-----
岩瀬 圭吾
市川 琢郎 1620000
田川 太陽 910000
甲斐 敏和 140000
松木 春奈 100000

SQL>
```

画面 4-2.外部結合と集計の実行結果

購入金額が 1 番多いのは「市川 琢郎」さんだということがわかりました。また、「岩瀬圭吾」さんも表示されました。なお、外部結合したデータの場合、購入金額は空で表示されています。データベースの世界ではこの空データのことを「NULL」と言います。



ORDER BYにて指定した列にNULLデータがある場合、NULLはもっとも大きいデータとして並び替えられてしまいます。そのため、岩瀬さんは一番上に表示されていますが、これについてはNVL関数を使用することで一番下に表示されることも可能です。詳しくは次章でご紹介します。

今回は、結合と集計などのデータ加工方法について学びました。これらのテクニックを使うと、簡単に顧客や、1 番売れている商品を求めるなど、システムの高度な分析が可能になります。実際の現場開発では、このように複数のテーブルデータを連携することが頻繁にありますので、ぜひこれらのテクニックを身につけましょう。

5 副問い合わせと関数

5-1.副問い合わせ

前回はSELECT文で利用できる結合と集計についてご説明しました。さらにSELECT文で利用できる「副問い合わせ」をご紹介します。副問い合わせも結合と同じく、よく使われるテクニックですので、是非習得しましょう。

まず、動作確認用にテーブルと初期データを作成しましょう。SQL*Plusで以下のSQLをそれぞれ記述し、実行してください。

(1)社員マスタテーブルを作成

```
CREATE TABLE 社員
```

```
(
```

社員コード	VARCHAR2(8) NOT NULL,
社員名	VARCHAR2(20) NOT NULL,
性別	VARCHAR2(8) NOT NULL,
役職	VARCHAR2(10),
給与	NUMBER(8,0) NOT NULL,
EMAIL	VARCHAR2(50) NOT NULL,
入社日	DATE NOT NULL

```
)
```

```
/
```

(2)社員マスタのデータを作成

```
INSERT ALL
```

```
INTO 社員 VALUES('1','堀越 秀喜','男性','取締役',800000,  
                  'HORIKOSHI@xxxxxxxx.co.jp','2003/01/01')  
INTO 社員 VALUES('2','三谷 奈央子','女性','部長',560000,  
                  'MITANI@xxxxxxxx.co.jp','2001/01/01')  
INTO 社員 VALUES('3','出口 真琴','女性','部長',520000,  
                  'DEGUCHI@xxxxxxxx.co.jp','2003/01/01')  
INTO 社員 VALUES('4','毛利 徹','男性','主任',475000,  
                  't.mouri@xxxxxxxx.co.jp','2003/01/01')  
INTO 社員 VALUES('5','栗林 和歌子','女性','主任',460000,  
                  'w.kuribayashi@xxxxxxxx.co.jp','2003/04/01')  
INTO 社員 VALUES('6','首藤 克典','男性',null,335000,  
                  'k.shutou@xxxxxxxx.co.jp','2004/05/14')  
INTO 社員 VALUES('7','神谷 歩美','女性',null,380000,  
                  'k.kamiya@xxxxxxxx.co.jp','2005/08/22')  
INTO 社員 VALUES('8','服部 静香','女性',null,400000,
```

```

                                'h.hattori@xxxxxxx.co.jp','2006/04/28')
INTO 社員 VALUES('9','大山田 寧々','女性',null,315000,
                                'n.ooyamada@xxxxxxx.co.jp','2007/08/24')
INTO 社員 VALUES('10','岡 史郎','男性',null,305000,
                                's.oka@xxxxxxx.co.jp','2009/08/01')

SELECT * FROM DUAL
/

```

準備ができれば、副問い合わせを実行してみましょう。先程作成した社員マスタでは給与列がありますが、この給与が「平均額以上」の社員を求める SELECT 文はどのようにすれば求めることができるでしょうか？このような場合は副問い合わせを使用することで取得することが可能です。副問い合わせとは、SELECT 文中に別の SELECT 文を埋め込むことで、サブクエリーとも呼ばれます。実際に副問い合わせを使用した SQL を記述してみましょう。サンプルのテーブル作成後、引き続き SQL コマンド画面で以下の SQL を記述してください。

```

SELECT 社員名,給与
FROM 社員
WHERE 給与 >= (SELECT AVG(給与) FROM 社員)
/

```

上記 SQL の弧内の中が副問い合わせとなります。副問い合わせはこのような WHERE 句の列名と比較演算子 (=や>など) に続けて、さらに SELECT 文を記述したものになります。このとき、括弧が必ず必要ですので忘れずに記述しましょう。この SQL では、副問い合わせで前回ご紹介した集計関数である「AVG」関数を使用して給与の平均を求め、その金額以上の社員に絞り込んだ SELECT 文をさらに記述しています。SQL が記述できた後は実際に実行してください。条件に合致する社員が5レコード表示されたと思います。

```
管理者: C:\Windows\system32\cmd.exe - sqlplus C##TEST/TEST
16 INTO 社員 VALUES('8','服部 静香','女性',null,400000,
17 'h.hattori@xxxxxxxx.co.jp','2006/04/28')
18 INTO 社員 VALUES('9','大山田 寧々','女性',null,315000,
19 'n.ooyamada@xxxxxxxx.co.jp','2007/08/24')
20 INTO 社員 VALUES('10','岡 史郎','男性',null,305000,
21 's.oka@xxxxxxxx.co.jp','2009/08/01')
22 SELECT * FROM DUAL
23 /

10行が作成されました。

SQL> SELECT 社員名,給与
2 FROM 社員
3 WHERE 給与 >= (SELECT AVG(給与) FROM 社員)
4 /

社員名                給与
-----
堀越 秀喜                800000
三谷 奈央子              560000
出口 真琴                520000
毛利 徹                  475000
栗林 和歌子              460000

SQL>
```

画面5-1. 副問い合わせを使用した SQL の実行結果

ただし、この結果では平均給与額がいくらまでは表示されていないため、正しい結果なのかどうかを検証することができません。大元の SELECT 文においては、副問い合わせはあくまで WHERE 条件での指定となるため、最終的な SELECT 結果としては表示されません。副問い合わせが返す平均額がいくらなのかを確認したい場合、副問い合わせの SQL だけで再度実行してください。平均額が 455,000 円ということがわかります。最初の実行結果ではこの金額以上の給与のレコードだけが表示されているので正しい結果であるということがわかります。このように副問い合わせの SELECT 結果を部分的に確認することで、SQL に記述に誤りがないかなどの検証を行うことができます。



文字選択して部分的に実行する機能はApplicationExpressの機能となります。他のSQL実行アプリケーションによっては部分実行ができない場合もあります。

```
管理者: C:\Windows\system32\cmd.exe - sqlplus C##TEST/TEST
22 SELECT * FROM DUAL
23 /

10行が作成されました。

SQL> SELECT 社員名,給与
2 FROM 社員
3 WHERE 給与 >= (SELECT AVG(給与) FROM 社員)
4 /

社員名                                給与
-----
堀越 秀喜                             800000
三谷 奈央子                           560000
出口 真琴                             520000
毛利 徹                               475000
栗林 和歌子                           460000

SQL> SELECT AVG(給与) FROM 社員;

AVG(給与)
-----
455000

SQL>
```

画面5-2.副問い合わせのみの実行結果

副問い合わせの SQL 文中にさらに、副問い合わせを使うなど、副問い合わせはネストして使うことも可能です。そのため、実際の開発現場でも副問い合わせは頻繁に使われます。ただし、あまり副問い合わせを使うと、パフォーマンスが悪化するため、テーブルの構造上、どうしても副問い合わせを使用しなければ求められない場合のみ使うようにした方がよいでしょう。



5-2.関数

「関数」とはテーブルの列やリテラルを加工したり、内部のデータ型を変換するなど、SQL を柔軟にするための多くの拡張機能です。関数を使用したい場合は SQL 文中に「関数名(引数名, 引数名…)」の書式で指定します。たとえば TO_CHAR という関数を使用する場合は以下のように使用します。

SELECT TO_CHAR(列名) FROM テーブル名

SELECT 文では SELECT 句や WHERE 句、GROUP BY 句、ORDER BY 句の他、INSERT 文の VALUE 句や、UPDATE 文の SET 句などで広く使用することができます。関数には、数値関数、文字列関数、日付関数などの各種関数が数多く用意されていますが、今回は、現場でもよく使われる代表的な関数をご紹介します。ただし、関数については SQL 規格で統一されているわけではなく、各データベースごとに独自に仕様が決められています。そのため、データベースによっては使えなかったり、関数名や引数の仕様がデータベースによって異なっています。今回ご紹介する関数も Oracle のものとなり、SQLServer など他のデータベースではエラーになってしまいますので、ご注意ください。



関数名等の違いはあるものの、今回ご紹介するような代表的な関数については、他のデータベースでも実装されています。もし他のデータベースで使いたい場合は、各データベースの SQL リファレンスを参照してください。

・文字データを置き換える REPLACE 関数

REPLACE 関数はテーブルに格納された文字データをそのまま表示するのではなく、置き換えた形で結果表示したい場合に使用します。例えば社員マスタにある性別を英語に置き換えて表示する場合、以下のように使用することで可能です。

```
SELECT 社員名, REPLACE(性別, '男性', 'MALE') FROM 社員;
```



画面5-3. REPLACE を使用して男性を「MALE」で表示した結果

REPLACE 関数の書式は第 1 引数が対象の列名、第 2 引数が置換前の文字列、第 3 引数が置換後の文字列になります。この例では、「男性」を「MALE」と英語表記にして表示しています。ただし、この結果は「女性」がそのまま日本語で表示されています。男性だけでなく、女性も英語表記に置き換えたい場合は、以下のように記述します。

```
SELECT 社員名, REPLACE(REPLACE(性別, '男性', 'MALE'), '女性', 'FEMALE') FROM 社員;
```

```
管理者: C:\Windows\system32\cmd.exe - sqlplus C##TEST/TEST
SQL> SELECT 社員名, REPLACE(REPLACE(性別,'男性','MALE'),'女性','FEMALE') FROM
社員;

社員名                                REPLACE(REPLACE(性別,'男性','MALE'),'女
性','FEMALE')
-----
堀越 秀喜                                MALE
三谷 奈央子                            FEMALE
出口 真琴                                FEMALE
毛利 徹                                    MALE
栗林 和歌子                            FEMALE
首藤 克典                                MALE
神谷 歩美                                FEMALE
服部 静香                                FEMALE
大山田 寧々                            FEMALE
岡 史郎                                    MALE

10行が選択されました。
SQL>
```

画面5-4. REPLACE を使用して性別を英語表記にした結果

この SQL では REPLACE の第 1 引数に、さらに別の REPLACE 関数をネストして使用しています。この SQL によって、まず「男性」が「MALE」に置き換えられた結果に対して、さらに「女性」が「FEMALE」に置き換えられ、最終的に両方英語に置き換えられた結果が表示されます。

・文字データを部分的に取り出す SUBSTR 関数（と検索文字列を検索する INSTR 関数）

SUBSTR 関数はデータのある一部分だけを表示したい場合に使用することができます。例えば、今回のサンプルデータでは社員名がフルネームで格納されていますが、苗字だけにして表示したい場合、以下の SQL で求めることができます。

```
管理者: C:\Windows\system32\cmd.exe - sqlplus C##TEST/TEST
大山田 寧々                女性
岡  史郎                MALE

10行が選択されました。

SQL> SELECT SUBSTR(社員名, 1, INSTR(社員名, ' ')) FROM 社員;

SUBSTR(社員名,1,INSTR(社員名,' '))
-----
堀越
三谷
出口
毛利
栗林
首藤
神谷
服部
大山田
岡

10行が選択されました。

SQL>
```

画面5-5. SUBSTR 関数と INSTR 関数を使用して苗字だけにした結果

第 1 引数が列名に続き、第 2 引数で開始位置、第 3 引数で文字数を指定します。今回の例では苗字の開始位置は必ず最初からのため第 2 引数は「1」を指定しますが、各社員によって苗字の長さが変わるため、そこから何文字までを取り出せばよいかはわかりません。そこで、第 3 引数では、INSTR という別の関数を使用しています。INSTR 関数では、検索対象の文字列から、ある文字を検索して、その文字が文字列から何番目にあるのかを返す関数です。苗字と名前の間には半角スペースがありますので、' 'の位置を検索し、その文字位置の結果を SUBSTR 関数を第 3 引数として返します。この方法により、苗字が何文字の人であっても正常に結果を求めることができます。

• NULL を別の値に変換する NVL 関数

NVL 関数は REPLACE 関数と似ていますが、NULL 専用の変換関数です。NVL 関数を使用すると、NULL データを別の結果に置き換えた上で表示することができます。役職を表示する場合に NULL データは他の値に置き換えた上で表示したい場合は以下のようにします。

```
SELECT 社員名, NVL(役職, '一般') FROM 社員;
```



```
管理者: C:\Windows\system32\cmd.exe - sqlplus C##TEST/TEST
神谷
服部
大山田
岡

10行が選択されました。

SQL> SELECT 社員名, NVL(役職, '一般') FROM 社員;

社員名                                NVL(役職, '一般')
-----
堀越 秀喜                            取締役
三谷 奈央子                          部長
出口 真琴                            部長
毛利 徹                              主任
栗林 和歌子                          主任
首藤 克典                            一般
神谷 歩美                            一般
服部 静香                            一般
大山田 寧々                          一般
岡 史郎                              一般

10行が選択されました。

SQL>
```

画面5-6. NVL 関数を使用して NULL データを「一般」に置き換えた結果

NVL 関数では第 1 引数が対象の列名、第 2 引数が NULL だった場合に置き換える文字列を指定します。この例では、役職が NULL のデータに関しては一般として表示しています。このとき、役職が NULL 以外のデータに関してはそのままの状態が表示されます。NULL データは、ORDER BY で指定した場合に一番上になるなど、特殊な動作をするデータです。このような動作をさせたくない場合は、NVL 関数を使用することで解決することができます。

・英文字の全てを大文字・小文字に変換する UPPER、LOWER 関数

UPPER/LOWER は、引数で与えられた文字列を大文字/小文字に変換して表示します。UPPER 関数で大文字、LOWER 関数で小文字に変換します。例えば、EMAIL 列で並び替えを行いたい場合、大文字、小文字が混在していると、大文字のほうが優先されてしまい、アルファベット順に並び替えることができません。

SELECT 社員名, EMAIL FROM 社員 ORDER BY EMAIL;

```
管理者: C:\Windows\system32\cmd.exe - sqlplus C##TEST/TEST
服部 静香                      一般
大山田 寧々                    一般
岡 史郎                        一般

10行が選択されました。

SQL> SELECT 社員名, EMAIL FROM 社員 ORDER BY EMAIL;

社員名                      EMAIL
-----
出口 真琴                    DEGUCHI@xxxxxxxx.co.jp
堀越 秀喜                    HORIKOSHI@xxxxxxxx.co.jp
三谷 奈央子                  MITANI@xxxxxxxx.co.jp
服部 静香                    h.hattori@xxxxxxxx.co.jp
神谷 歩美                    k.kamiya@xxxxxxxx.co.jp
首藤 克典                    k.shutou@xxxxxxxx.co.jp
大山田 寧々                  n.ooyamada@xxxxxxxx.co.jp
岡 史郎                      s.oka@xxxxxxxx.co.jp
毛利 徹                      t.mouri@xxxxxxxx.co.jp
栗林 和歌子                  w.kuribayashi@xxxxxxxx.co.jp

10行が選択されました。

SQL>
```

画面5-7.EMAIL 列で並び替えた結果

このような場合に、UPPER 関数で並び替えると、意図どおりにアルファベットに並び替えることが可能です。

SELECT 社員名, UPPER(EMAIL) FROM 社員 ORDER BY UPPER(EMAIL);

```
管理者: C:\Windows\system32\cmd.exe - sqlplus C##TEST/TEST
岡 史郎 s.oka@xxxxxxxx.co.jp
毛利 徹 t.mouri@xxxxxxxx.co.jp
栗林 和歌子 w.kuribayashi@xxxxxxxx.co.jp

10行が選択されました。

SQL> SELECT 社員名, UPPER(EMAIL) FROM 社員 ORDER BY UPPER(EMAIL);

社員名                                UPPER(EMAIL)
-----
出口 真琴 DEGUCHI@XXXXXXXX.CO.JP
服部 静香 H.HATTORI@XXXXXXXX.CO.JP
堀越 秀喜 HORIKOSHI@XXXXXXXX.CO.JP
神谷 歩美 K.KAMIYA@XXXXXXXX.CO.JP
首藤 克典 K.SHUTOU@XXXXXXXX.CO.JP
三谷 奈央子 MITANI@XXXXXXXX.CO.JP
大山田 寧々 N.OOYAMADA@XXXXXXXX.CO.JP
岡 史郎 S.OKA@XXXXXXXX.CO.JP
毛利 徹 T.MOURI@XXXXXXXX.CO.JP
栗林 和歌子 W.KURIBAYASHI@XXXXXXXX.CO.JP

10行が選択されました。

SQL>
```

画面5-8.EMAIL 列を大文字に変換した上で並び替えた結果

それ以外にも、アプリケーションからデータ検索機能を実装する際に、大文字小文字を区別せずに検索したい場合などに使用することができます。

・データ型を変換する TO_NUMBER, TO_CHAR, TO_DATE 関数

テーブルの列はそれぞれ、「データ型」を持っています。これは、テーブル作成時に指定したデータ型で、数値型、文字型、日付型のいずれかになります。データ型によって SQL の動作が異なるケースがあります。例えば、ORDER BY 句を使って社員コードで並び替えたい場合、以下の SQL を実行してもうまく並び替えがされません。

```
SELECT 社員コード, 社員名 FROM 社員 ORDER BY 社員コード;
```

```
管理者: C:\Windows\system32\cmd.exe - sqlplus C##TEST/TEST
大山田 寧々 N.OOYAMADA@XXXXXXXX.CO.JP
岡 史郎 S.OKA@XXXXXXXX.CO.JP
毛利 徹 T.MOURI@XXXXXXXX.CO.JP
栗林 和歌子 W.KURIBAYASHI@XXXXXXXX.CO.JP

10行が選択されました。

SQL> SELECT 社員コード, 社員名 FROM 社員 ORDER BY 社員コード;

社員コード      社員名
-----
1              堀越 秀喜
10             岡 史郎
2              三谷 奈央子
3              出口 真琴
4              毛利 徹
5              栗林 和歌子
6              首藤 克典
7              神谷 歩美
8              服部 静香
9              大山田 寧々

10行が選択されました。

SQL>
```

画面5-9.社員コードで並び替えた結果

社員コード「10」のデータが2番目に並び替えられています。社員コード列は文字型（VARCHAR2型）ですが、文字型データの場合、「2」よりも「10」の方が小さい値とみなされてしまうためです。このような場合、データ型を変換することでうまくいきます。TO_NUMBER は数値型、TO_CHAR で文字型、TO_DATE で日付型に変換することができます。今回のケースでは以下のSQLのように数値型に変換した上で並び替えることで、うまく並び替えることができます。

```
SELECT 社員コード, 社員名 FROM 社員 ORDER BY TO_NUMBER(社員コード);
```

ORDER BY 句に TO_NUMBER 関数を使用しています。数値型として並び返され、正常に1から10まで並び替えられます。TO_NUMBER の引数は変換したい対象の列名のみを指定します。TO_CHAR 関数、TO_DATE 関数も同様です。

```
管理者: C:\Windows\system32\cmd.exe - sqlplus C##TEST/TEST
6      首藤 克典
7      神谷 歩美
8      服部 静香
9      大山田 寧々

10行が選択されました。

SQL> SELECT 社員コード, 社員名 FROM 社員 ORDER BY TO_NUMBER(社員コード);

社員コード      社員名
-----
1      堀越 秀喜
2      三谷 奈央子
3      出口 真琴
4      毛利 徹
5      栗林 和歌子
6      首藤 克典
7      神谷 歩美
8      服部 静香
9      大山田 寧々
10     岡 史郎

10行が選択されました。

SQL>
```

画面5-10.TO_NUMBER関数を使用して、数値型に変換した上で並び替えた結果

変換関数は、上記ケース以外にも、WHERE句で2つの列を「=」演算子で結合したい場合にそれぞれのデータ型が異なってエラーになってしまう場合や、関数の引数に使用したいがデータ型が異なるために使用できないという場合などに使用されます。

その他にも数値を四捨五入する ROUND 関数や日付型を月末に変換する LAST_DAY 関数など、様々な関数が用意されています。一度、SQL のリファレンスを読んで、どのような機能があるか覚えておくことをお勧めいたします。そうしておく、もしもの時に役立てることができるでしょう。

6 トランザクションとロック

6-1. データの整合性とは

これまででテーブルデータの作成する INSERT 文、更新するための UPDATE 文、削除するための DELETE 文等についてご説明しました。これらの SQL を使ってデータの管理が自由に扱うことが可能ですが、システム開発においては、「データの整合性」を損なわないようにデータを扱わなければいけません。データの整合性とは、「ユーザーやアプリケーションの動作により、データに矛盾が生じないこと」です。例えば、EC サイトアプリケーションにて、商品を1つ購入した際、内部では以下のようなデータの変更が行われたとします。

1. 「商品マスタ」より、商品の在庫が残っているか確認する
2. 「商品マスタ」の商品の在庫を-1 する
3. 「注文テーブル」に注文データを作成する

商品マスタの「在庫」列を SELECT 文で確認し、在庫があれば UPDATE 文で「在庫」列を-1 した上で、最後に注文テーブルにデータを INSERT されます。しかし、もし、3の INSERT 処理で何らかのエラーが発生し、注文データが作れなかった場合、注文データがないにもかかわらず、商品マスタの在庫が-1 されたままとなってしまいます。

商品マスタ

商品コード	商品名	単価	在庫数
1	Tシャツ	2000	20
2	ジーンズ	12000	15
3	ジャケット	15000	10
4	バッグ	7500	5
5	ニット帽	2000	1

注文

注文番号	商品コード	数量

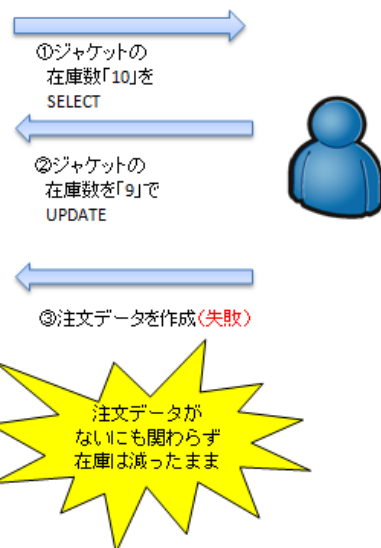


図6-1. 処理の途中でエラーが発生した場合

いずれの場合も、在庫データと、実際の在庫数が合わないという問題が生じます。このような状態を「データの不整合」と呼ばれます。データの不整合が起きないようにするためには、アプリケーションが途中でエラーにならないようにロジックミスなくすることが最善の方法です。しかし、アプリケーションの2次的な不具合や、ユーザーが想定していないデータを入力するなど、すべてのエラーを完全に防ぐことは現実的に難しいと言えます。もし、処理途中でエラーが起きてもデータに不整合が起きない方法はないでしょうか？実は、データベースには「トランザクション制御」という機能があり、これによってデータの不整合を防止することができます。

●トランザクション制御機能

「トランザクション」とはユーザーの1つの操作にたいして、内部で発生するテーブルの一連の更新処理のことです。上記の例では、商品マスタ、注文テーブルに対するデータの変更処理の1つのトランザクションとなります。実は、データベースでは、データのINSERTやUPDATEを実行しただけでは、変更操作が完全に確定されません。変更操作を確定するためには、別途「COMMIT（コミット）」と呼ばれる命令が必要となります。また、COMMITするまでに行なわれたデータの変更は、「ROLLBACK（ロールバック）」という命令でデータ変更の操作をキャンセルすることができます。トランザクションが最後まで正常終了した際にCOMMITを行い、トランザクションの途中でエラー終了してしまった場合はROLLBACKすることで、データの不整合を起こさないようにすることができます。



なお、COMMITもROLLBACKもせず、データベース接続を終了した場合は、自動でCOMMITされます。また、Oracleでは、データの編集以外のCREATE TABLE文などのSQLはROLLBACKすることはできません。

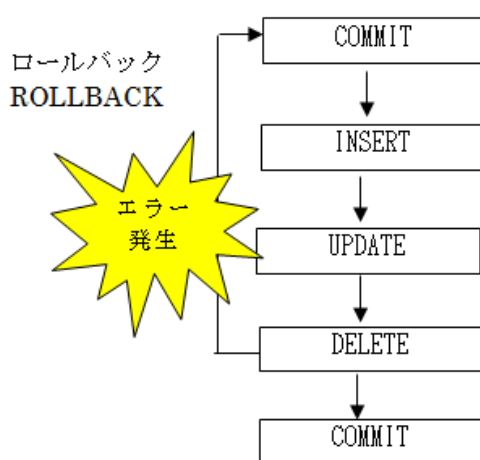


図6-2.コミットとロールバック

SQL*Plus で実際に動作を確認してみましょう。まず、サンプル用にテーブルを作成します。以下のSQLをそれぞれ記述し、実行してください。

(1)商品マスタを作成

```
CREATE TABLE 商品
```

```
(
```

```
    商品コード          NUMBER(5,0) NOT NULL,  
    商品名              VARCHAR2(20) NOT NULL,  
    単価                NUMBER(8,0) NOT NULL,  
    在庫数              NUMBER(5,0) NOT NULL
```

```
)
```

```
/
```

(2)注文テーブルを作成

```
CREATE TABLE 注文テーブル
```

```
(
```

```
    注文番号            NUMBER(5,0) NOT NULL,
```

```

商品コード          VARCHAR2(20) NOT NULL,
数量                NUMBER(5,0) NOT NULL
)
/

```

(3)商品マスタのデータを作成

```

INSERT ALL
  INTO 商品 values(1,'Tシャツ',2000,20)
  INTO 商品 values(2,'ジーンズ',12000,15)
  INTO 商品 values(3,'ジャケット',15000,10)
  INTO 商品 values(4,'バッグ',7500,5)
  INTO 商品 values(3,'ニット帽',2000,1)
SELECT * FROM DUAL
/

```

(4) 操作を確定

```
COMMIT;
```

上記 SQL は、商品と注文テーブルを作成する SQL です。注文テーブルは「商品コード」をもち、商品と関連づけられています。商品にはデータもあわせて作成します。最後に COMMIT を実行し、ここまでの処理を確定しておきましょう。

テーブルを作成した後は、さきほどの例にあった商品を購入するトランザクションを実行します。今回はジャケットを 1 つ購入するとします。まず、商品マスタより在庫を確認します。以下の SQL を実行してください。

```
SELECT * FROM 商品 WHERE 商品名 = 'ジャケット';
```

在庫数が 10 と表示され、問題なく注文できることが確認できます。続けて、在庫数を 1 つ減らすために UPDATE 文を実行します。

```
UPDATE 商品 SET 在庫数 = 9 WHERE 商品名 = 'ジャケット';
```

ジャケットの在庫数が 9 になりました。最後に、注文データを作成します。以下のように記述して実行してください。

```
INSERT INTO 注文テーブル VALUES(NULL, 3, 1);
```

注文テーブル作成時、注文番号は NOT NULL（必須）として定義していましたが、今回、INSERT 文で NULL を指定していたため「注文番号に NULL は挿入できません」というエラーが表示されます。エラーになったため、注文テーブルにレコードは作成されていません。しかし、在庫数は 9 です。データの不整合が起きています。そこで、ROLLBACK を実行します。

```
ROLLBACK;
```


ROLLBACK を実行すると、これまでの一連のトランザクションが元に戻ります。商品マスタをもう一度確認してみましょう。

SELECT * FROM 商品;

ジャケットの在庫数が 10 に戻っていることが確認できます。もし、注文データを正常に作成された場合は、「ROLLBACK」のかわりに「COMMIT」を実行します。変更が確定されるため、今後、ROLLBACK しても変更は元に戻りません。Oracle ではこのようなかたちでトランザクション制御ができるようになっています。



6-2. ロックの制御機能

トランザクションの途中でエラーが発生した場合のケース以外にも、複数のユーザーが同一データの 2 重更新によって、データの不整合が起きることが考えられます。

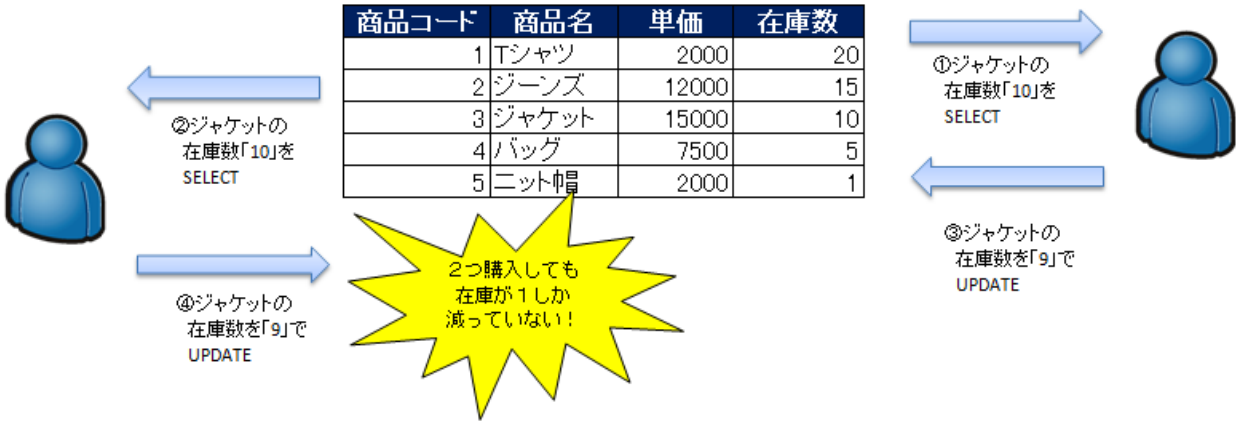


図6-4.複数のユーザーが2重更新した場合

このように2重更新はデータの不整合になるため、起きてはいけない問題です。データベースには、2重更新を防ぐ仕組みとして「ロック制御」機能が備わっています。「ロック」とは、1 人がトランザクションを COMMIT または ROLLBACK するまで、他のユーザーがデータの変更をできなくする仕組みです。これにより、データの2重更新を防止することができます。

ロックは、その強さや範囲などによっていくつかの種類があります。ロックの種類は以下の通りです。

種類	データ型	説明
ロックの強度	共有ロック	他のトランザクションは参照のみ可能となり、更新は不可
	排他ロック	他のトランザクションは参照・更新ともに付加
ロックの範囲	表レベル	表単位でかけるロック
	行レベル	表の中の特定の行（レコード）に対してかけるロック
ロックの自動/手動	暗黙的ロック	SQL を実行した際に自動でかけられるロック
	明示的ロック	ロックをかけるための SQL 文を実行してかけるロック

表6-1. ロックの種類

INSERT/UPDATE/DELETE 文を実行した場合、トランザクションが確定するまでの間、対象レコードに

対して排他ロックが自動的にかけられます。SELECT 文の場合は、読み取り可能な共有ロックとなりますが、SELECT 時に排他ロックをかけたい場合もあります。図 3 の例においては、在庫数を SELECT してから UPDATE するまでの間もロックをかけておかないと、データの不整合がおきることになります。SELECT 時に排他ロックをかけたい場合は、以下の SQL を実行します。

SELECT * FROM 商品 WHERE 商品名 = 'ジャケット' FOR UPDATE [待機時間];

SELECT 文の後に「FOR UPDATE」と指定すると、SELECT 文で取得される対象のレコードに行ロックがかかり、トランザクションが完了するまでは、他のユーザーからジャケットのレコードが更新不可となります。「FOR UPDATE」の後には、ロックしたいレコードが他のユーザーによって既にかけている場合に、待機する時間を数値か、「NOWAIT」で指定します。数値の単位は秒です。指定した時間だけ待機しても解放されない場合はエラーが返されます。「NOWAIT」を指定している場合は即座にエラーが返されます。

ロックはテーブル単位で行うことも可能です。表ロックを行うには以下の SQL を実行します。

LOCK TABLE テーブル名 IN [ロックモード];

ロックモードには以下のいずれかを指定します。

種類	説明
ROW SHARE MODE	他のトランザクションから表ロックを禁止しますが、行ロックは可能です。
ROW EXCLUSIVE MODE	他のトランザクションから表ロックを禁止しますが、行ロックは可能です。
SHARE MODE	自分自身のトランザクションも含め、表の読取りのみ許可します。 他のトランザクションも含めて表共有ロックのみを許可します。
SHARE ROW EXCLUSIVE MODE	表全体を排他ロックします。他のトランザクションからの行共有ロックのみ許可します。
EXCLUSIVE MODE	表全体を排他ロックします。他のトランザクションからは、行ロックも含めて、禁止されます。

表6-2.表ロックのロックモードの種類

行ロックの方がロックする範囲が小さいため、複数のユーザーが並行でトランザクション処理する際に効率よくなりますが、頻繁に行ロックが行われると、データベースの負荷がかえって大きくなります。トランザクションがテーブルのほとんどのレコードを更新するような場合は、表単位でロックをかける方が効率的です。

●デッドロック

ロックの仕組みにより、2重更新による不整合を防止することができますが、場合によっては「デッドロック」と呼ばれる問題が生じることがあります。デッドロックとは、複数のトランザクションがお互いの処理に必要なデータをロックし合っているために、処理が止まってしまう現象のことです。

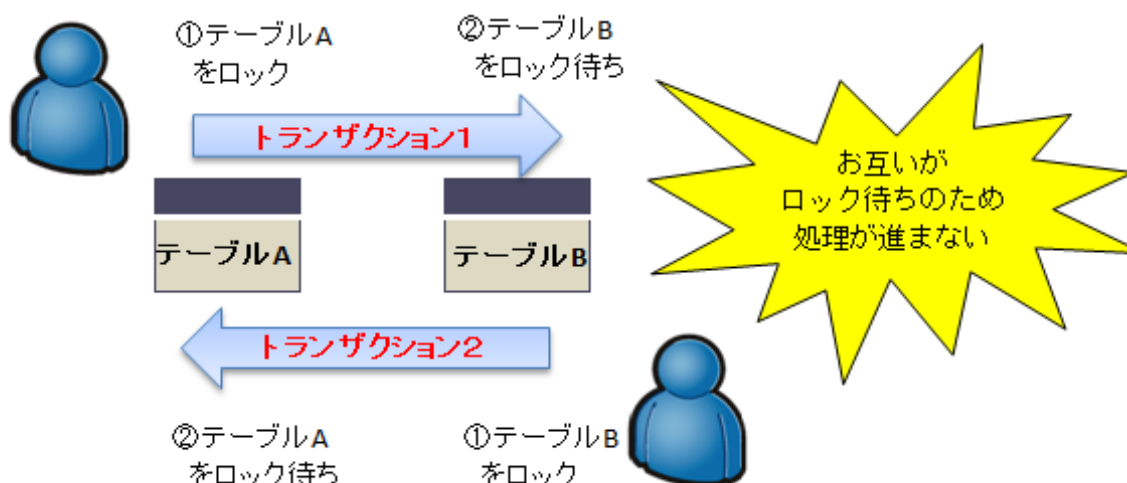


図6-5.デッドロック

Oracle でデッドロックを検知した場合は、ロックを解放するためお互いのトランザクションを強制中断してしまいます。このデッドロック問題は、複数種類のトランザクションが、同一のテーブルを別々の順序でアクセスする際に発生しますので、更新するテーブルの順序を同一にすることで、発生させないようにすることができます。また、ロック範囲を表ロックから行ロックに狭める、トランザクション自体の処理速度を向上させることで、デッドロックの発生頻度を抑えることもできます。

以上で、複数のテーブルを更新した場合におけるデータの不整合問題と、その対策について解説しました。アプリケーションの開発者にとってデータの整合性を考慮することは重要です。開発においては、「もしトランザクションの途中でエラーが起きたらどうなる?」「もし他のユーザーがデータを変更したらどうなる?」の2点を意識して、開発を心がけましょう。

最後に

本ドキュメントは以上になります。

Oracle のインストールから、テーブルの作成、結合、集計、関数などのテクニックをご紹介しました。この他にもデータベースのパフォーマンスチューニング方法やバックアップ方法など、学ぶべきことはたくさんありますが、ここまで読んでいただいた方は、きっと Oracle の基礎が身についているはずです。実際の現場で実践しながらステップアップしていただけたらと思います。

また、弊社の Oracle 開発支援ツールとして「SI Object Browser」という製品があります。GUI でもデータベース操作ができ、開発効率を高めることができるツールです。ブログでも画面入りでご紹介していますので、ご興味があればぜひご覧ください。

<製品ホームページ>

<https://products.sint.co.jp/siob>

<ブログ>

<https://products.sint.co.jp/siob/blog>



お問い合わせ先

本ドキュメントや弊社製品に関しましてご不明点がございましたらお知らせください。

株式会社システムインテグレータ Object Browser 事業部 営業担当

TEL : 03-5768-7979 東京営業所 後迫（ウシロザコ）、横島

06-4706-5471 大阪支社 永田

E-Mail : oob@sint.co.jp

URL : <https://products.sint.co.jp/siob/inquiry>

※本ドキュメントに記載されている商品名は、各社の商標または登録商標です。

※当社の許可なく、本ドキュメントの全部または一部を、広くご本人以外の方が利用可能な状態にする
(ホームページにて公開する、不特定多数の人にメールを転送するなど)ことはご遠慮ください。