ORACLE MASTER Oracle Database 12c

ノートブック: 242_IT書籍
作成: 2016/03/05 22:44 **更新:** 2016/03/19 11:38

ORACLE MASTER Oracle Database 12c

<序章> ORACLE MASTER Bronze Oracle Database 12c 資格試験の概要と学習方法

～略～

<第一章> リレーショナルデータベースとSQL

1.1 リレーショナルデータベースの基礎知識

■リレーショナルデータベースとは

Oracleサーバーはリレーショナルデータベース型及びオブジェクトリレーショナル型のデータベースを管理する、リレーショナルデータベース管理システム(RDBMS)となる。

【重要】
▶リレーショナルデータベースは、データを「表」で管理する
▶複数の表に分割されているデータの関連付けには、データの値を使用する

■リレーショナルデータベースの主な用語

【重要】
▶表は行(ROW)と列(COLUMN)で構成される
▶行と列が交差する部分を「フィールド」という
▶フィールドに値が格納されていない状態を「NULL値」(NULL値が含まれている)という
▶NULL値は空白(スペース)や数値の0とは区別される

➤主キーと外部キー

【重要】
▶行を一意に識別するための列(または、列の組み合わせ)を「主キー」という
▶主キーには、重複した値は格納出来ない
▶主キーにはNULL値を含めることは出来ない
▶同じ表または他の表の主キー(または一意キー)を参照する列(または列の組み合わせ)を「外部キー」という
▶外部キーには、NULL値を含めることができる

1.2 SQLの概要

■SQLとは

【重要】
SQLは、ANSIやISO、JISなどで標準規格化された、リレーショナルデータベースを操作するための言語

■SQL文の分類

【重要】
　SQLは下記の4種類に分類される
・DML(データ操作言語)
・DDL(データ定義言語)
・DCL(データ制御言語)
・トランザクション制御

～本章のまとめ～
■リレーショナルデータベースは、データを「表」で管理する
■複数の表に分割されているデータの関連付けには、データの値を使用する
■表は行(ROW)と列(COLUMN)で構成される
■行と列が交差する部分を「フィールド」という
■フィールドに値が格納されていない状態を「NULL値」(NULL値が含まれている)という
■NULL値は空白(スペース)や数値の0とは区別される
■行を一意に識別するための列(または、列の組み合わせ)を「主キー」という
■主キーには重複した値を格納出来ない
■主キーには、NULL値を含めることが出来ない
■同じ表または他の表の主キー(または一意キー)を参照する列(または列の組み合わせ)を「外部キー」という
■外部キーには、NULL値を含めることができる
■SQLは、ANSIやISO、JISなどで標準規格化された、リレーショナルデータベースを操作するための言語である
■SQLは以下のDML(データ操作言語)、DDL(データ定義言語)、DCL(データ制御言語)、トランザクション制御の4つのに分類される

- コマンド
  - DML
    - SELECT
    - INSERT
    - UPDATE
    - DELETE
    - MERGE

- DDL
  - CREATE
  - ALTER
  - DROP
  - RENAME
  - TRUNCATE
  - COMMENT

- DCL
  - GRANT
  - REVOKE

- トランザクション制御
  - COMMIT
  - ROLLBACK
  - SAVEPOINT

<第2章>　SELECT文を使用したデータの取得

2.1 SELECT文の基本

■SELECT文によるデータの取り出し

　下記3つの機能を利用してデータベースに格納されている大量のデータの中から、目的のデータを取り出します

- 射影
- 選択
- 結合

➤射影
　特定の列のみを取り出す機能

➤選択
　検索対象の表から特定の行のみを取り出す機能

➤結合
　複数の表を関連付けてデータを取り出す機能

【重要】
// 上記と同様

■SELECT文の基本構文

【重要】
▶SELECT句に複数の列を指定する場合は、「,」(カンマ)で区切る
▶データはSELECT句に指定した列の順番で表示される
▶SELECT句に「*(アスタリスク)」を指定すると、すべての列のデータが表示される

■表構造の表示

➤表の構造の表示
DESCRIBE 表名
※DESCRIBEは「DESC」と省略することも可能

【重要】
▶DESCRIBEコマンドを使用すると、表の構造を確認できる

♪Tips
表構造は「データ・ディクショナリ」からSELECT文で確認することも出来ます。
データ・ディクショナリとはオブジェクト(表など)の定義情報などが格納されている特別な表

2.2 SELECT句のいろいろな指定方法

■算術式の使用

SQL> SELECT EMPNO, ENAME, SAL, SAL * 12
2 FROM EMPLOYEES;

EMPNO	ENAME	SAL	SAL*12
1001	佐藤	500000	6000000
1002	鈴木	200000	2400000
1003	高橋	300000	3600000
1004	田中	355000	4260000
1005	渡辺	280000	3360000
1006	伊藤	300000	3600000
1007	山本	285000	3420000
1008	中村	245000	2940000
1009	小林	300000	3600000
1010	斉藤	150000	1800000
1011	加藤	110000	1320000
1012	吉田	295000	3540000
1013	山田	280000	3360000
1014	佐々木	230000	2760000

14行が選択されました。

➤算術演算子の優先順位

➤算術子とNULL値
NULL値の演算結果はNULLになること

【重要】
▶算術演算子の優先順位は*、/が先で、+、-が後
▶算術演算子の優先順位は()(丸括弧)を使用することで明示的に指定できる
▶NULL値を含む算術式は、計算結果もNULL値になる

■列別名の使用

- 方法1：SELECT句の列名または算術式の後ろに、1つ以上のスペースを入れて列別名を指定する
- 方法2：ASキーワードを指定して列別名を指定する

SQL> SELECT EMPNO 社員番号, ENAME 社員名, SAL AS 給与, SAL * 12 AS 年収
2 FROM EMPLOYEES;

社員番号 社員名 給与 年収

```
1001 佐藤          500000    6000000
1002 鈴木          200000    2400000
1003 高橋          300000    3600000
1004 田中          355000    4260000
1005 渡辺          280000    3360000
1006 伊藤          300000    3600000
1007 山本          285000    3420000
1008 中村          245000    2940000
1009 小林          300000    3600000
1010 斉藤          150000    1800000
1011 加藤          110000    1320000
1012 吉田          295000    3540000
1013 山田          280000    3360000
1014 佐々木        230000    2760000
```

14行が選択されました。

列別名にスペースや特殊文字(＃や$など)を使用する場合、また大文字、小文字を区別する必要がある場合は列別名を「”」(二重引用符)で囲む必要があります。

SQL> SELECT EMPNO AS Empo,
2 ENAME AS “Ename”,
3 SAL AS “Sal”,
4 SAL * 12 AS “Annual Salary”
5 FROM EMPLOYEES;

```
EMPO Ename           Sal Annual Salary
```

```
1001 佐藤          500000      6000000
1002 鈴木          200000      2400000
1003 高橋          300000      3600000
1004 田中          355000      4260000
1005 渡辺          280000      3360000
1006 伊藤          300000      3600000
1007 山本          285000      3420000
1008 中村          245000      2940000
1009 小林          300000      3600000
1010 斉藤          150000      1800000
1011 加藤          110000      1320000
1012 吉田          295000      3540000
1013 山田          280000      3360000
1014 佐々木        230000      2760000
```

14行が選択されました。

♪Tips「”」(二重引用符)で囲まなかった場合、そのSQL構文はエラーになるので注意
列別名は算術式に組み込んだり、SELECT句に指定する列名の代わりに使用したりすることは出来ません。

【重要】
列別名にスペースや特殊文字を含む場合や、大文字小文字を区別する場合は、列別名を「”」(二重引用符)で囲む必要がある

連結演算子の使用

SELECT句に「||」(連結演算子)を指定すると、複数の列値やリテラル(定数値)、算術演算式の結果を連結する文字式を記述することが出来ます。
➤連結演算子
列名 || 列名

SQL> SELECT ENAME || ‘さんの入社日は’ || HIREDATE || ‘です。’ AS 入社日確認
2 FROM EMPLOYEES;

入社日確認
――――――――――――――
佐藤さんの入社日は01-02-25です。
鈴木さんの入社日は00-03-26です。
高橋さんの入社日は00-05-30です。
田中さんの入社日は02-06-02です。
渡辺さんの入社日は02-07-11です。
伊藤さんの入社日は08-01-06です。
山本さんの入社日は00-08-09です。
中村さんの入社日は00-09-17です。
小林さんの入社日は06-10-21です。
斉藤さんの入社日は01-12-17です。
加藤さんの入社日は06-10-21です。
吉田さんの入社日は09-03-13です。
山田さんの入社日は01-03-13です。
佐々木さんの入社日は04-05-02です。

14行が選択されました。

【重要】
文字リテラルや日付リテラルを文字式で指定する場合は「’」(一重引用符)で囲む必要がある

➤代替引用符
文字列リテラルの一部に「’」を使用したい場合、「代替引用符(q)演算子」を使用する。
この演算子を使用すると、SQL文内の以下の文字を独自の引用符デリミタ(区切り文字)として指定できます。

- 任意のシングルバイト文字やダブルバイト文字
- ¥[ ], { }, ( ), < >の各組み合わせ

q’引用符デリミタ …　引用符デリミタ’

【代替引用符演算子を使用した引用符デリミタの変更】
SQL> SELECT YOMI || q’?’s Salary : ?’ || SAL “Monthly Salary”
2 FROM EMPLOYEES;

Monthly Salary

sato’s Salary : 500000
suzuki’s Salary : 200000
takahashi’s Salary : 300000
tanaka’s Salary : 355000
watanabe’s Salary : 280000
ito’s Salary : 300000
yamamoto’s Salary : 285000
nakamura’s Salary : 245000
kobayashi’s Salary : 300000
saito’s Salary : 150000
kato’s Salary : 110000
yoshida’s Salary : 295000
yamada’s Salary : 280000
sasaki’s Salary : 230000

14行が選択されました。

【カッコを使用した引用符デリミタの指定】
SQL> SELECT YOMI || q’[’s Salary :]’ || SAL “Manthly Salary”
2 FROM EMPLOYEES;

Manthly Salary

sato’s Salary : 500000
suzuki’s Salary : 200000
takahashi’s Salary : 300000
tanaka’s Salary : 355000
watanabe’s Salary : 280000
ito’s Salary : 300000
yamamoto’s Salary : 285000
nakamura’s Salary : 245000
kobayashi’s Salary : 300000
saito’s Salary : 150000
kato’s Salary : 110000
yoshida’s Salary : 295000
yamada’s Salary : 280000
sasaki’s Salary : 230000

14行が選択されました。

【重要】
▶代替引用符(q)演算子を使用すると、引用符デリミタを変更できる
▶この場合の引用符デリミタには、任意のシングルバイト文字やダブルバイト文字、[ ]、{ }、( )、< >の各組み合せを指定できる

重複行の排除

「DISTINCT」キーワードを指定

```
SQL> SELECT DISTINCT JOB
  2  FROM   EMPLOYEES;

JOB
---------
主任
社長
部長
営業
事務
```

【重要】
▶「DISTINCT列名」を指定すると、その列の重複業が排除される
▶DISTINCTキーワードにつづいて複数の列を指定すると、指定した列の値の組み合わせが一意になる行のみ表示される
▶DISTINCTキーワードはSELECTキーワードの直後に1度だけ記述する

～本章のまとめ～
■検索対象の表から特定の列のみを取り出す機能を「射影」という
■検索対象の表から特定の行のみを取り出す機能を「選択」という
■複数の表を関連付けてデータを取り出す機能を「結合」という
■SELECT句に複数の列を指定する場合は、「,(カンマ)」で区切る
■データはSELECT句に指定した列の順番で表示される
■SELECT句に「*」(アスタリスク)を指定すると、すべての列のデータが表示される
■DESCRIBEコマンドを使用すると、表の構造を確認できる
■算術演算子の優先順位は*,/が先で、+,-が後
■算術演算子の優先順位は()(丸括弧)を使用することで明示的に指定できる
■NULL値を含む算術式は、計算結果もNULL値になる
■列別名にスペースや特殊文字を含む場合や、大文字小文字を区別する場合は、列別名を「"」(二重引用符)で囲む必要がある
■文字リテラルや日付リテラルを文字式で指定する場合は「'」(一重引用符)で囲む必要がある
■代替引用符(q)演算子を使用して引用符デリミタを変更できる
■引用符デリミタには、任意のシングルバイト文字やダブルバイト文字、[]、{}、()、<>の各組み合わせを指定できる
■「DISTINCT 列名」を指定すると、その列の重複行が排除される
■DISTINCTキーワードに続いて複数の列を指定すると、指定した列の値の組み合わせが一意になる
■DISTINCTキーワードはSELECTキーワードの直後に1度だけ記述する

<第3章>　データの制限およびソート

3.1 問い合わせで取得する行の制限

WHERE句を使用した行の制限(選択)

➤WHERE句の条件の基本構文
WHERE 列名 比較演算子 {定数 | 値のリスト | 式 | 列名}

比較演算子
= 等しい

> 大きい
>= 以上
< 小さい
<= 以下

<>, !=, ^= 等しくない
BETWEEN a AND b a以上b以下
IN(値1,[値2…])
LIKE 文字のパターンが一致
IS NULL 値がNULL値の場合にTRUE

【重要】
▶WHERE句を使用すると、行を取り出すための条件を指定できる
▶WHERE句はFROM句の後に記述する必要がある

■等号(＝)を使用した条件の指定

　等号(＝)は最も基本的な比較演算子です。左辺の値と右辺の値が等しい行のみを取り出します
➤WHERE句に文字または日付リテラルを指定する場合の注意点
文字列を「'(一重引用符)」で囲む必要がある。

●Where句に文字リテラルを指定したSQL文の実行(1)

```
SQL> select empno, ename, deptno
  2  from employees
  3  where ename = '佐藤';

     EMPNO ENAME          DEPTNO
---------- ---------- ----------
      1001 佐藤               10
```

●Where句に日付リテラルを指定したSQL文の実行

```
SQL> select empno, ename, hiredate
  2  from employees
  3  where hiredate = '06-10-21';

     EMPNO ENAME      HIREDATE
---------- ---------- --------
      1009 小林       06-10-21
      1011 加藤       06-10-21
```

➤WHERE句への列別名の指定(エラー)
WHERE句に列の別名を指定することは出来ないことの説明

【重要】
▶条件に指定する文字リテラルは、大文字・小文字が区別される
▶条件に指定する日付リテラルは、日付書式が区別される
▶WHERE句に別名は指定できない

■不等号を使用した条件の指定

【重要】
▶不等号は、等号または不等号と組み合わせて指定できる(>=,<=,<>)

■BETWEEN演算子を使用した条件の指定

➤BETWEEN演算子を使用した条件の指定方法
WHERE 列名 BETWEEN 下限値 AND 上限値
WHERE 列名 NOT BETWEEN 下限値 AND 上限値

【重要】
▶BETWEEN演算子を使用すると、範囲を指定した条件を指定できる
▶「BETWEEN」を指定した場合、境界値も取り出す範囲に含まれる
▶「NOT BETWEEN」を指定した場合、境界値は取り出す範囲に含まれない

■IN演算子を使用した条件の指定

IN演算子を使用すると、列値と複数の値を比較する条件を指定できます。
➤IN演算子を使用した条件の指定方法
WHERE 列名 IN(値1[,値2…])
WHERE 列名 NOT IN(値1[,値2…])

【重要】
▶IN演算子を使用すると、列値を複数の値と比較する条件を指定できる

■LIKE演算子を使用した条件の指定

指定した文字パターンに一致する行を取り出すことが出来る
➤LIKE演算子を使用した条件の指定方法
WHERE 列名 LIKE '文字パターン'
WHERE 列名 NOT LIKE '文字パターン'

LIKE演算子で指定できるワイルド
- ％　0文字以上の任意の文字列と一致する
- _　任意の1文字と一致する

➤ESCAPEオプション
検索文字列の中に「％」や「_」を含む文字列があった場合は、「ESCAPEオプション」を設定して、
「％」や「_」を文字リテラルとして扱うように指定する必要があります。
●ESCAPEオプションの指定方法
WHERE 列名 LIKE '文字パターン' ESCAPE 'エスケープ文字'

「エスケープ文字」には任意の1バイト文字(￥、$、#、aなど)を指定できます。
エスケープ文字を指定すると、文字パターンに含まれるエスケープ文字の直後のワイルドカードは、
文字リテラルとして扱われる

```
SQL> select prodno, pname, price
  2  from   products
  3  where  pname like '100¥%%' escape '¥';

PRO PNAME                          PRICE
--- ---------------------------- ----------
D04 100%マッシロ修正液                  350
```

【重要】
▶LIKE演算子を使用すると、指定した文字パターンに一致する行を取り出すことができる
▶LIKE演算子では「%」と「_」の2種類のワイルドカードを使用できる
▶ワイルドカードは、全角・半角を区別しない
▶複数のワイルドカードを組み合わせることができる
▶ESCAPEオプションを使用すると、「%」や「_」を文字リテラルとして扱うことができる
▶エスケープ文字には任意の1バイト文字を指定できる

■IS NULL演算子を使用した条件の指定

➤NULL値の比較はIS NULL演算子でしか行えない
NULL値が含まれるかどうかは、IS NULL演算子、またはIS NOT NULL演算子でしか評価できません
エラーにはならない！ということも覚えておく
【重要】
▶IS NULL演算子を使用すると、列にNULL値が含まれるかどうかを確認できる
▶NULL値との比較は等号(=)や不等号(<)では評価出来ない
▶等号、不等号を使用して、NULL値を評価しても、SQL文はエラーにならないが、結果は1件も取り出されない

3.2 論理演算子による条件の指定

- AND
- OR
- NOT

■IN演算子を使用したOR演算子の書き換え

- WHERE 列1 = 値1 OR 列1 = 値2 OR 列1 = 値3
- WHERE 列1 IN(値1, 値2, 値3)

IN演算子を使用したほうが完結でわかりやすいが、
IN演算子を使用して書き換えても実行時のパフォーマンスはかわりません。。。

【重要】
▶OR演算子を使用した複数の条件は、IN演算子で書き換えられる場合がある
▶OR演算子を使用した複数の条件を、IN演算子で書き換えても実行時のパフォーマンスは同じである

3.3 SELECT文で取り出す行のソート

■ORDER BY句を使用した行のソート

NULL値はデフォルトでは「**最も大きい値**」として扱われます

➤NULL値を取り出すソート順の指定
「NULLS FIRST」
「NULLS LAST」 を指定します
●NULL値のソート順の制御
ORDER BY 列名 [NULLS FIRST | NULLS LAST]

キーワード	説明
NULLS FIRST	NULL値を含むデータを最初に取り出す
NULLS LAST	NULL値を含むデータを最後に取り出す

```
SQL> select empno, ename, comm
  2  from employees
  3  order by comm nulls first;

     EMPNO ENAME            COMM
---------- ---------- ----------
      1001 佐藤
      1002 鈴木
      1007 山本
      1013 山田
      1012 吉田
      1011 加藤
      1005 渡辺
      1014 佐々木
      1008 中村
      1009 小林
      1010 斉藤                0
      1003 高橋            30000
      1004 田中            50000
      1006 伊藤           140000

14行が選択されました。
```

➤ORDER BYとWHERE句の同時指定

【重要】
▶ORDER BY句を使用すると、ソートした(並べ替えた)データを取り出すことができる
▶ORDER BY句は**SELECT文の最後**に記述する
▶降順でソートする場合は「DESC」キーワードを使用する
▶ソート順序のデフォルトはASC(昇順)
▶NULL値は最も大きい値として扱われる
▶NULL値を含むデータを取り出す順序は「NULLS FIRST」または「NULLS LAST」キーワードで制御できる
▶ORDER BY句とWHERE句は同時に指定できる

■いろいろなソート

➤列別名でのソート
WHERE句には列別名を指定できませんが、ORDER BY句には列別名を指定できる

```
SQL> select empno, ename, sal * 12 annsal
  2  from employees
  3  order by annsal;

     EMPNO ENAME          ANNSAL
---------- ---------- ----------
      1011 加藤           1320000
      1010 斉藤           1800000
```

```
      1002 鈴木          2400000
      1014 佐々木        2760000
      1008 中村          2940000
      1013 山田          3360000
      1005 渡辺          3360000
      1007 山本          3420000
      1012 吉田          3540000
      1009 小林          3600000
      1003 高橋          3600000
      1006 伊藤          3600000
      1004 田中          4260000
      1001 佐藤          6000000

14行が選択されました。
```

♪Tips
列別名の大文字小文字を区別したり、スペースを含めたりする為に、"Annsal"のように文字列を「"」(二重引用符)で囲んだ場合は
ORDER BY句でも"Annsal"と指定することが必要。覚えておく！！

```
SQL> select empno, ename, sal * 12 "Annsal"
  2  from employees
  3  order by "Annsal";

     EMPNO ENAME          Annsal
---------- ---------- ----------
      1011 加藤          1320000
      1010 斉藤          1800000
      1002 鈴木          2400000
      1014 佐々木        2760000
      1008 中村          2940000
      1013 山田          3360000
      1005 渡辺          3360000
      1007 山本          3420000
      1012 吉田          3540000
      1009 小林          3600000
      1003 高橋          3600000
      1006 伊藤          3600000
      1004 田中          4260000
      1001 佐藤          6000000

14行が選択されました。
```

➤列の位置を指定したソート
SELECT句に指定した「列の位置」を指定することもできる。

➤複数の列を指定したソート
～他のDBと同様なので略～

【重要】
▶ORDER BY句には、**列別名**を指定できる
▶ORDER BY句には、SELECT句に指定した「**列の位置**」を指定できる
▶ORDER BY句には、**複数の列**を指定できる

▶複数の列を指定した場合、左側にしてした列から順番にソートされる
▶複数の列を指定した場合、列ごとにASCまたはDESCを指定できる

■SQL行制限

Oracle12cで新しく、SELECT文の結果として戻される行の数を制限する機能が追加された。
➤row_limiting_clauseによる行数の制限
行数を制限するには、以下のようにrow_limiting_clause(行制限の条件)を記述します
ORDER BY句を指定する場合はORDER BY句の後ろに記述します

●row_limiting_clauseを使用したSQL文

```
SELECT      列名 [,列名…]
FROM        表名
[WHERE      条件]
[ORDER BY   列名, [列名…]]
[OFFSET offset { ROW | ROWS }]
[FETCH { FIRST     | NEXT }
       { row_count | percent PERCENT }
       { ROW       | ROWS }
       { ONLY      | WITH TIES }]
```

●OFFSET句とFETCH句

種類	説明
OFFSET句	スキップする行数を指定する。OFFSET句を省略するとスキップする行数が「0」になる為、行制限の開始が最初の行になる
FETCH句	返される行数または行の割合を指定する。

●OFFSET句

キーワード	説明
OFFSET	OFFSET句を開始するキーワード
offset	スキップする行数。数値で指定する。負の値は「0」とみなされる。また問い合わせから返される行数以上の数値を指定した場合や、NULLを指定した場合は、一行も返されない(返される行数は0行)
ROWまたはROWS	意味を明確にする為に用意されているキーワード。ROWとROWSに違いはなく、同じ意味となる(区別はない)。OFFSET句が指定されている場合は省略不可

●FETCH句の指定

キーワード	説明
FETCH	FETCH句を開始するキーワード

キーワード	説明
FETCHまたはNEXT	意味を明確にするために用意されているキーワード。FIRSTとNEXTに違いはなく、同じ意味となる(区別はない)。FETCHが指定されている場合は省略不可能
row_countまたはpercent PERCENT	返される行数(row_count)または返される行の割合(percent PERCENT)を指定する。row_count及びpercentには数値を指定する。percent PERCENTには「選択された行の合計行数のうち、返される行数の割合」を指定する
ROWまたはROWS	意味を明確にする為に用意されているキーワード。ROWとROWSに違いはなく、同じ意味となる(区別はない)。OFFSET句が指定されている場合は省略不可
ONLYまたはWITH TIES	ONLYを指定すると、row_countまたはpercent PERCENTに指定された行数が**正確に**返される。 WITH TIESを指定すると、row_countまたはpercent PERCENTに指定された行数の最後の行と同じソートキーまでが返される。 WITH TIESを指定する場合は、ORDER BY句の指定が必須。ORDER BY句が指定されていない場合、追加の行は表示されない

【重要】
▶結果セットで戻される行の数を制限するには、row_limiting_clauseを使用する
▶row_limiting_clauseにはOFFSET句とFETCH句が含まれる
▶OFFSET句には、スキップする行数(制限を開始する行の1つ前の行数)を指定する
▶FETCH句には、返される行数または行の割合を指定する
▶ROWとROWSに区別はないが、OFFSET句、FETCH句ともに省略不可
▶FIRSTとNEXTに区別はないが、FETCH句を指定する場合は省略不可

❒column
SQL*Plusでは「置換変数」を使用することで、WHERE句の条件に指定する値をSQL文の実行時に指定して置き換えることが出来ます。
置換関数の種類

置換変数	説明
&置換変数	SQL文の実行時に置き換える値の入力を要求。**保存されない**
&&置換変数	SQL文の実行時に置き換える値の入力を要求。**保存される**

実行時に新旧の値が表示されない時は
SET VERIFY {ON | OFF}
上記コマンドにて「VERIFY」変数にONを設定する

第4章 単一行関数

4.1 SQL関数の概要と単一行関数の基本

■SQL関数の概要

SQL関数(SQLファンクション)とはOracleデータベースに予め組み込まれている関数
大きく下記2つに分かれている

- 単一行関数　　：入力データごとに結果を一つ戻す関数
- グループ関数　：集計処理を行って得たデータを一つ返す
  【重要】
  ▶単一行間数は、1件の入力データごとに処理を行い、入力データごとに1行の結果を戻す
  ▶グループ関数は、複数件の入力データをグループ化して集計処理を行った結果を1つ返す

■単一行関数の基本構文とタイプ

SELECT句の他、WHERE句やORDER BY句などでも使用出来ます。
➤単一行関数の基本構文
関数名([引数1[,引数2…]])
SYSDATE関数のように引数を取らない関数も存在する

➤単一行関数のネスト
単一行関数はネストすることが出来ます。
➤単一行間数のタイプ

単一行関数のタイプ

タイプ	説明
文字関数	文字値を引数として受け入れて、文字値または数値を戻す関数
数値関数	数値を引数として受け入れて、数値を戻す関数
日付関数	日付値を引数として受け入れて、日付値または数値を戻す関数
変換関数	引数として受け入れた値を、別のデータ型に変換する関数
汎用関数	引数として受け入れて、NULL値に関連する処理を行う関数

4.2 文字関数

大文字小文字変換関数

➤UPPER関数

```
SQL> select yomi, upper(yomi) from employees;

YOMI                 UPPER(YOMI)
-------------------- --------------------
sato                 SATO
suzuki               SUZUKI
takahashi            TAKAHASHI
tanaka               TANAKA
watanabe             WATANABE
ito                  ITO
yamamoto             YAMAMOTO
nakamura             NAKAMURA
kobayashi            KOBAYASHI
saito                SAITO
kato                 KATO
yoshida              YOSHIDA
yamada               YAMADA
```

```
sasaki                SASAKI

14行が選択されました。
```

DBの中に大文字小文字で登録されているかわからないデータがあった場合、変換して、文字列比較すれば、容易に文字列を取り出すことができる

```
SQL> select empno, ename, yomi
  2  from employees
  3  where upper(yomi) = 'TAKAHASHI';

     EMPNO ENAME      YOMI
---------- ---------- --------------------
      1003 高橋       takahashi
```

```
SQL> select 'small characters' as 小文字, upper('large characters') as 大文字
  2  from dual;

小文字            大文字
----------------- -----------------
small characters LARGE CHARACTERS
```

❒column DUAL表
「データが1件だけ入っていることが保証されている表」です

DUAL表にはデータが1件しかないため、前述の実行例のように、関数から戻される情報が常に1件だけ表示されます。

➤LOWER関数
文字列をすべて小文字に変換する

➤INITCAP関数
「単語の先頭文字」を大文字に、2文字目以降を小文字に変換して戻す関数
●INITCAP関数
INITCAP(文字列)

```
SQL> select INITCAP('test characters')
  2  from dual;

INITCAP('TESTCH
---------------
Test Characters
```

上記の例では、単語をスペースで区切っていますが、
「-」(ハイフン)
「,」(カンマ)
などで区切ることも出来ます。

■文字操作関数

➤CONCAT関数
2つの文字列を結合して戻す関数。
連結演算子と使用して2つの文字列を結合した場合と同じ結果を返します。
3つ以上の文字列を結合することは出来ません。

```
SQL> select concat(initcap('oracle'),initcap('server')) from dual;

CONCAT(INITC
------------
OracleServer
```

➤SUBSTR関数
m番目の文字からn文字文の文字列を戻す関数(部分文字列を戻す関数)です
nが省略された場合は、m文字目から末尾の文字までを戻します。

```
SQL> select substr('Oracle Server', 2, 3), substr('Oracle Server', 2)
  2  from dual;

SUB SUBSTR('ORAC
--- ------------
rac racle Server
```

2つ目の引数mに負の値を指定すると、文字列の後ろから数えてm文字目からn文字分の文字列が戻されます

```
SQL> select substr('Oracle Server', -6, 3), substr('Oracle Server', -6)
  2  from dual;

SUB SUBSTR
--- ------
Ser Server
```

➤LENGTH関数
「文字数」を戻す関数

```
SQL> select length('Oracle Server')
  2  from dual;

LENGTH('ORACLESERVER')
----------------------
                    13
```

LENGTH関数では全角の文字や記号も1文字としてカウントされます！

```
SQL> select length('あいうえお')
  2  from dual;

LENGTH('あいうえお')
```

```
--------------------
                   5
```

➤INSTR関数
「指定した文字パターンが現れる位置」を戻す関数
引数として受け入れた文字列1のm文字目から文字列2を検索し、n回目に一致した文字列の位置を戻します。
m及びnが省略された時のデフォルト値は「1」です
文字列の最後まで文字列2がない場合は「0」(ゼロ)を戻します。
●INSTR関数
INSTR(文字列1, 文字列2 [,m] [,n])

```
SQL> select instr('Oracle Server', 'er', 1, 2),
  2         instr('Oracle Server', 'er')
  3  from dual;

INSTR('ORACLESERVER','ER',1,2) INSTR('ORACLESERVER','ER')
------------------------------ --------------------------
                            12                          9
```

➤LPAD関数、RPAD関数
引数として受け入れた文字列が、n文字になるように、「埋め込み文字」を埋め込んで戻す関数
LPADは左側に、RPADは右側に埋め込み文字が埋め込まれます。
●LPAD関数、RPAD関数
LPAD(文字列, n, '埋め込み文字')
RPAD(文字列, n, '埋め込み文字')
埋め込み文字が省略された場合は半角スペースが入る

```
SQL> select lpad(yomi, 10, '*'), rpad(yomi, 10, '$')
  2  from employees;

LPAD(YOMI,10,'*')    RPAD(YOMI,10,'$')
-------------------- --------------------
******sato           sato$$$$$$
****suzuki           suzuki$$$$
*takahashi           takahashi$
****tanaka           tanaka$$$$
**watanabe           watanabe$$
*******ito           ito$$$$$$$
**yamamoto           yamamoto$$
**nakamura           nakamura$$
*kobayashi           kobayashi$
*****saito           saito$$$$$
******kato           kato$$$$$$
***yoshida           yoshida$$$
****yamada           yamada$$$$
****sasaki           sasaki$$$$

14行が選択されました。
```

➤TRIM関数
前後にある「削除文字(任意の1文字)」を取り除いて戻す関数

●TRIM関数
TRIM([LEADING | TRAILING | BOTH] [削除文字] FROM 文字列)
TRIM(文字列)

```
SQL> select trim(leading 'O' from 'Oracle Server')
  2  from dual;

TRIM(LEADING
------------
racle Server
```

```
SQL> select '     Oracle Server     ', trim('     Oracle Server     ')
  2  from dual;

'ORACLESERVER'           TRIM('ORACLES
------------------------ ------------
     Oracle Server       Oracle Server
```

➤REPLACE関数
引数として受け入れた文字列のうち、変更前の文字列を変更後文字列に置き換えた文字列を戻す関数
変更後文字列が省略された場合は変更前文字列を削除した文字列を戻します
●REPLACE関数
REPLACE(文字列, 変更前文字列 [,変更後文字列])

【重要】
▶CONCAT関数は、引数に指定された2つの文字列を結合して戻す
▶SUBSTR関数は、引数に指定した文字列のm番目の文字列からn文字の文字列を戻す
▶LENGTH関数は、引数に指定された文字列の「文字数」を戻す
▶INSTR関数は「指定した文字パターンが現れる位置」を戻す
▶LPAD関数、RPAD関数は引数に指定された文字列の左右(前後)にある任意の文字列(削除文字)を取り除いて戻す
▶REPLACE関数は、引数に指定された文字列のうち、任意の文字列(変更前文字)を別の文字列(変更後文字)に置き換える

4.3 数値関数

■主な数値関数

➤ROUND関数
引数に指定された数値を小数点以下n桁に四捨五入して戻す関数

```
SQL> select round(12345.678, 1), round(12345.678, 0),
  2         round(12345.678)
  3  from   dual;

ROUND(12345.678,1) ROUND(12345.678,0) ROUND(12345.678)
------------------ ------------------ ----------------
            12345.7              12346            12346
```

引数nを省略すると、整数値に四捨五入される
引数にマイナスを指定することも可能。

```
SQL> select round(12345.678, -1), round(12345.678, -2)
  2  from dual;

ROUND(12345.678,-1) ROUND(12345.678,-2)
------------------- -------------------
              12350               12300
```

➤TRUNC関数
引数に指定された数値を小数点以下n桁に切り捨てて戻す関数
●TRUNK関数
TRUNK(数値 [,n])

```
SQL> select  trunc(12345.678, 2), trunc(12345.678, -1),
  2          trunc(12345.678)
  3  from dual;

TRUNC(12345.678,2) TRUNC(12345.678,-1) TRUNC(12345.678)
------------------ ------------------- ----------------
          12345.67               12340            12345
```

➤MOD関数
引数nを引数mで割ったあまりを戻す関数
●MOD関数
MOD(n, m)

```
SQL> select mod(10, 3)
  2  from dual;

 MOD(10,3)
----------
         1
```

商ではなくあまりが戻されている点に注意

【重要】
省略～

4.4 日付関数

■日付値の基礎知識

Oracleデータベースが扱う日付値には以下の特徴がある

- 日付値は「世紀、年、月、日、時、分、秒」を表す内部的な数値書式で格納される
- 日付値にはデフォルトの表示書式がある
- 英語環境のデフォルトの表示書式は「DD-MON-RR」(日-月-年)

- 日本語環境(本書の実行環境)のデフォルトの表示書式は「RR-MM-DD」(年-月-日)
- 日付値に対して加算、減算などを行うことができる

❐column
➤日付の表示書式及び表示言語の変更
ALTER SESSION SET nls_date_format='表示形式';
ALTER SESSION SET nls_date_language='言語';

```
SQL> alter session set nls_date_format='DD-MON-RR';

セッションが変更されました。

SQL> select ename, hiredate
  2  from employees
  3  where deptno = 10;

ENAME      HIREDATE
---------- --------------
佐藤       25-2月 -01
中村       17-9月 -00
佐々木     02-5月 -04
```

➤日付の表示言語の変更

```
SQL> alter session set nls_date_language = 'AMERICAN';

セッションが変更されました。

SQL> select ename, hiredate
  2  from employees
  3  where deptno = 10;

ENAME      HIREDATE
---------- -----------
佐藤       25-FEB-01
中村       17-SEP-00
佐々木     02-MAY-04
```

➤日本語環境のデフォルト設定に戻すSQL文
ALTER SESSION SET nls_date_format = 'RR-MM-DD'
ALTER SESSION SET nls_date_language = 'JAPANESE'

■日付値の計算

- 「日付値＋数値」及び「日付値－数値」
- 「日付値＋数値/24」及び「日付値-数値/24」
- 「日付値-日付値」

日付値に対して除算や乗算は実行出来ない
日付値同士を減算することはできるが、日付値同士を加算することは出来ない

➤日付値＋数値、日付値－数値

```
SQL> select ename, hiredate, hiredate + 90, hiredate - 30
  2  from employees
  3  where deptno = 10;

ENAME      HIREDATE HIREDATE HIREDATE
---------- -------- -------- --------
佐藤       01-02-25 01-05-26 01-01-26
中村       00-09-17 00-12-16 00-08-18
佐々木     04-05-02 04-07-31 04-04-02
```

➤日付値＋数値/24、日付値-数値/24
指定した数値が時間数として加算・減算されます。

```
SQL> select sysdate, sysdate + 12/24, sysdate - 12/24
  2  from dual;

SYSDATE  SYSDATE+ SYSDATE-
-------- -------- --------
16-03-11 16-03-12 16-03-11
```

➤日付値-日付値
指定した2つの日付値の間に経過した「日数」が戻されます(1日以下の値は少数として戻されます)

```
SQL> select empno, sysdate, sysdate - hiredate
  2  from employees
  3  where deptno = 10;

     EMPNO SYSDATE  SYSDATE-HIREDATE
---------- -------- ----------------
      1001 16-03-11       5493.94677
      1008 16-03-11       5654.94677
      1014 16-03-11       4331.94677
```

【重要】
▶日付値に数値を加算すると、指定した数値が日数として加算される
▶日付値に「数値/24」を加算すると、指定した数値が時間数として加算される
▶日付値に日付値を減算すると指定した2つの日付値の間に経過した「日数」が戻される
▶日付地に対して乗算や除算を実行することは出来ない
▶日付値同士を加算することは出来ない

■日付関数

➤SYSDATE関数
現在の日付を返す。引数は受け入れません。

➤MONTHS_BETWEEN関数
●MONTHS_BETWEEN(日付1,日付2)

```
SQL> select ename,
  2         months_between(sysdate, hiredate),
  3         trunc(months_between(sysdate,hiredate))
  4  from employees
```

```
  5  where deptno = 10;

ENAME      MONTHS_BETWEEN(SYSDATE,HIREDATE) TRUNC(MONTHS_BETWEEN(SYSDATE,HIREDAT
E))
---------- -------------------------------- ------------------------------------
--
佐藤                                180.579073                                   1
80
中村                                185.837137                                   1
85
佐々木                              142.321008                                   1
42
```

➤ADD_MONTHS関数
引数に指定した日付のnヶ月後の日付を戻す関数

```
SQL> select sysdate, add_months(sysdate, 3), add_months(sysdate, -1)
  2  from dual;

SYSDATE  ADD_MONT ADD_MONT
-------- -------- --------
16-03-11 16-06-11 16-02-11
```

●ADD_MONTHS関数での「月の最終日」の扱い
月の最終日は特別な値として扱われるので注意が必要です
引数の日付に月の最終日を指定すると、戻される値も2番めの引数で指定された月数後の月の最終日になります

```
SQL> select add_months('14-02-28', 1)
  2  from dual;

ADD_MONT
--------
14-03-31
```

➤NEXT_DAY関数
引数に指定された「日付」の翌日以降に、指定された曜日になる最初の日付を戻す関数
●NEXT_DAY関数
NEXT_DAY(日付,'曜日')

実行環境と曜日の指定形式

実行環境	曜日の指定形式
日本語環境	「日曜日」「月曜日」…の形式、または「日」「月」…の形式
英語環境	「SUN」、「MON」、「TUE」、「WED」、「THU」、「FRI」、「SAT」(英語スペルの先頭3文字。大文字小文字は区別されれない)

```
SQL> select next_day('14-01-24', '日曜日')
  2  from dual;

NEXT_DAY
--------
14-01-26
```

➤LAST_DAY関数
引数に指定された日付を含む月の、最終日の日付を戻す関数
これを利用すればうるう年であったかどうかが簡単にわかります

```
SQL> select last_day('12-02-01')
  2  from dual;

LAST_DAY
--------
12-02-29
```

➤ROUND関数
引数に指定した日付を四捨五入して戻す。どの単位で四捨五入するかは「書式」で指定します
●ROUND関数
ROUND(日付 [,'書式'])

指定できる主な書式

書式	説明
YEAR	指定した日付が6月30日以前の場合は当年の1月1日午前0時を、7月1日以降の場合は翌年の1月1日午前0時を戻す
MONTH	指定した日付が15日以前の場合は当月の1日午前0時を、16日以降の場合は翌月の1日午前0時を戻す
DD	指定した日付が正午より前の場合は当日の午前0時を、正午以降の場合は翌日の午前0時日付を戻す。**デフォルト値**

```
SQL> select sysdate, round(sysdate, 'YEAR'), round(sysdate, 'MONTH')
  2  from dual;

SYSDATE  ROUND(SY ROUND(SY
-------- -------- --------
16-03-11 16-01-01 16-03-01
```

➤TRUNC関数

引数に指定された日付値を切り捨てて戻す関数
書式はROUND関数と同様

```
SQL> select sysdate, trunc(sysdate, 'year'), trunc(sysdate, 'month')
  2  from dual;
```

```
SYSDATE   TRUNC(SY TRUNC(SY
--------- -------- --------
16-03-11  16-01-01 16-03-01
```

第5章 変換関数・汎用関数と条件式の指定

5.1 データ型の変換と変換関数

データ型の変換には、「暗黙的なデータ型変換」と「明示的なデータ型変換」の2種類があります。

■暗黙的なデータ型変換

【重要】
▶暗黙的なデータ型変換とは、Oracleサーバーが自動的に行うデータ型の変換である
▶暗黙的なデータ型変換は「データ型の変換が意味を持つ場合」にのみ成功する

■明示的なデータ型変換と主な変換関数

明示的な型変換のほうがわずかにSQLのパフォーマンスが向上します。
Oracleサーバーでは明示的なデータ型変換の使用が推奨されています。

➤明示的なデータ型変換の実行方法

- TO_CHAR関数　　：数値を文字値に変換
- TO_DATE関数　　：文字値を日付値に変換
- TO_NUMBER関数　：文字値を数値に変換
  ※注意：日付値を数値に変換する関数、数値を日付値に変換する関数は存在しないので注意！

■TO_CHAR関数(日付値→文字値)

引数に日付値を指定すると、TO_CHAR関数は指定された「日付書式」を使用して日付値を文字値に変換して戻します。
また、「NLSパラメータ」を指定することで、日付書式の言語環境を設定することも出来ます。
●TO_CHAR関数(日付値→文字値)
TO_CHAR(日付 [,'日付書式'] [,'NLSパラメータ'])

主な日付書式の要素
～略～

```
SQL> select to_char(sysdate, 'YYYY-MM-DD HH24:MI:SS')
  2  from dual;

TO_CHAR(SYSDATE,'YY
-------------------
2016-03-12 11:34:58
```

以下の3点に注意して実行。。。

- 「/」や「-」、「(」などの半角記号はそのまま結果に表示される

- 「年」や「月」、「日」のような漢字やひらがな、カタカナ、アルファベットなどを「"」(二重引用符)で囲むとそのまま結果に表示される
- 日付書式の要素及び半角記号以外をそのまま指定するとエラーになる

```
SQL> select to_char(sysdate, 'YYYY/MM/DD'),
  2         to_char(sysdate, 'YYYY"年"MM"月"DD"日"(DAY)')
  3  from dual;

TO_CHAR(SY TO_CHAR(SYSDATE,'YYYY"年"
---------- --------------------------
2016/03/12 2016年03月12日(土曜日)
```

➤NLSパラメータによる言語環境の指定
TO_CHAR関数の引数に「NLSパラメータ」を指定することで、関数内で使用する言語環境を任意のものに変更することができる
セッション単位で変更したい場合は、ALTER SESSION文を実行します

●言語環境の変更
TO_CHAR(日付 [, 日付書式] [, nls_date_language='言語'])

```
SQL> select sysdate,
  2         to_char(sysdate, 'MONTH:MON:DAY:DY'),
  3         to_char(sysdate, 'MONTH:MON:DAY:DY',
  4                 'nls_date_language = AMERICAN')
  5  from dual;

SYSDATE  TO_CHAR(SYSDATE,'MONTH:MON:DAY TO_CHAR(SYSDATE,'MONTH:MON:DAY:DY','NLS_D
-------- ------------------------------ ----------------------------------------
16-03-12 3月  :3月  :土曜日:土           MARCH     :MAR:SATURDAY :SAT
```

日付書式の大文字、小文字は区別されます。
日本語環境の場合は先頭文字だけを大文字にしても表示結果はかわりません。
このことも覚えておく

★全部小文字のパターン

```
SQL> select sysdate,
  2         to_char(sysdate, 'MONTH:MON:DAY:DY'),
  3         to_char(sysdate, 'month:mon:day:dy',
  4                 'nls_date_language = AMERICAN')
  5  from dual;

SYSDATE  TO_CHAR(SYSDATE,'MONTH:MON:DAY TO_CHAR(SYSDATE,'MONTH:MON:DAY:DY','NLS_D
-------- ------------------------------ ----------------------------------------
16-03-12 3月  :3月  :土曜日:土           march     :mar:saturday :sat
```

★先頭文字だけ大文字のパターン

```
SQL> select sysdate,
  2         to_char(sysdate, 'MONTH:MON:DAY:DY'),
  3         to_char(sysdate, 'Month:Mon:Day:Dy',
  4                 'nls_date_language = AMERICAN')
  5  from dual;

SYSDATE   TO_CHAR(SYSDATE,'MONTH:MON:DAY TO_CHAR(SYSDATE,'MONTH:MON:DAY:DY','NLS_D
--------- ------------------------------ ----------------------------------------
16-03-12 3月  :3月  :土曜日:土            March     :Mar:Saturday :Sat
```

➤数値で表示される日付書式の表示形式の変更
数値で表示される日付書式の後ろに「TH」や「SP」などの接尾辞を指定すると、
これらの表示形式を順序表記やスペル表記に変更することができる

数値で表示される日付書式に使用できる接尾辞

接尾辞	説明
TH	順序表記(例:DDTH→4TH)
SP	スペル表記(例:DDSP→FOUR)
SPTHまたはTHSP	順序表記+スペル表記(例:DDSPTH→FOURTH)

```
SQL> select ename,
  2         to_char(hiredate, 'Ddthsp "of" Month, YYYY',
  3                 'nls_date_language = AMERICAN')
  4  from employees
  5  where deptno = 10;

ENAME      TO_CHAR(HIREDATE,'DDTHSP"OF"MONTH,YYY
---------- ------------------------------------
佐藤       Twenty-Fifth of February , 2001
中村       Seventeenth of September, 2000
佐々木     Second of May       , 2004
```

➤FM要素の指定
　日付書式にFM要素を指定して「埋め込みモード」(デフォルトで有効)を無効にすると、
「数値の先行0」や「文字値の前後に含まれるスペース」が取り除かれて表示されます
FM要素を指定すると、FM要素以降のすべての日付書式に対する埋め込みモードが無効になるので注意

```
SQL> select ename, to_char(hiredate, 'ddth "of" Month, YYYY', 'nls_date_language
 = AMERICAN')
  2  from employees
  3  where deptno = 10;

ENAME      TO_CHAR(HIREDATE,'DDTH"OF"M
---------- --------------------------
佐藤       25th of February , 2001
中村       17th of September, 2000
佐々木     02nd of May       , 2004
```

```
SQL> select ename, to_char(hiredate, 'fmddth "of" Month, YYYY', 'nls_date_languag
e = AMERICAN')
  2  from employees
  3  where deptno = 10;

ENAME      TO_CHAR(HIREDATE,'FMDDTH"OF
---------- --------------------------
佐藤       25th of February, 2001
中村       17th of September, 2000
佐々木     2nd of May, 2004
```

■TO_CHAR関数(数値→文字値)

主な数値書式の要素
～略～

▼表示桁数が足りないと、＃＃＃が表示される

```
SQL> select ename, to_char(sal, 'L999,990'), to_char(sal, 'L99,990')
  2  from employees
  3  where deptno = 10;

ENAME      TO_CHAR(SAL,'L999, TO_CHAR(SAL,'L99,
---------- ------------------ ----------------
佐藤                 ¥500,000 ################
中村                 ¥245,000 ################
佐々木               ¥230,000 ################
```

▼一の位に「0」を指定すると、先行0も表示されますが、「9」を指定すると先行0は表示されないので注意

```
SQL> select to_char('0.12345', '0.99999'), to_char('0.12345', '9.99999')
  2  from dual;

TO_CHAR( TO_CHAR(
-------- --------
 0.12345   .12345
```

■TO_DATE関数

文字値を日付値に変換するための関数
この関数は主に、デフォルトに日付書式と異なる形式の文字列を日付書式に変換する場合に使用される

●TO_DATE関数
TO_DATE(文字列 [,'日付書式'] [,NLSパラメータ])

```
SQL> select to_date('2014年01月01日', 'YYYY"年"MM"月"DD"日"')
  2  from dual;

TO_DATE(
```

```
--------
14-01-01
```

▼暗黙的なデータ型の変換の失敗

```
SQL> select sysdate - '00-01-01'
  2  from dual;
select sysdate - '00-01-01'
                 *
行1でエラーが発生しました。:
ORA-01722: 数値が無効です。
```

▼TO_DATE関数を使用した明示的なデータ型変換

```
SQL> select sysdate - TO_DATE('00-01-01')
  2  from dual;

SYSDATE-TO_DATE('00-01-01')
---------------------------
                 5915.55226
```

♪Tips　暗黙的なデータ型変換は日付値が求められている箇所に文字値を指定した場合にも実行されますが、上記の実行例の場合は以下の点が原因で、暗黙的なデータ型変換が失敗します。

- 日付値は「日付値 - 日付値」だけでなく「日付値 - 数値」でも操作できる
- Oracleサーバーは算術式の中に文字列を見つけると、数値に変換しようとする
- その結果、Oracleサーバーが暗黙的なデータ型変換によって「'00-01-01'」を数値に変換しようとしてエラーになる

このように、変換後のデータ型として想定されるデータが複数あるときは、変換関数を使用して明示的なデータ型変換を行うことが必要になります。

➤YY要素とRR要素の違い
　日付書式のYY要素とRR要素はいずれも「年の下2桁」を表す要素ですが、受け入れた値の「世紀」の扱い方が異なります。

YY要素とRR要素

要素	説明
YY	受け入れた値(年の下2桁)を、「常に現在の世紀」として扱う
RR	受け入れた値(年の下2桁)を、「現在の年に近い世紀」として扱う

▼YY要素とRR要素の違い

```
SQL> select TO_CHAR(TO_DATE('20-10-10', 'YY-MM-DD'), 'YYYY') YY20,
  2         TO_CHAR(TO_DATE('20-10-10', 'RR-MM-DD'), 'YYYY') YY20,
  3         TO_CHAR(TO_DATE('95-10-10', 'YY-MM-DD'), 'YYYY') YY95,
  4         TO_CHAR(TO_DATE('95-10-10', 'RR-MM-DD'), 'YYYY') YY95
  5  from dual;

YY20 YY20 YY95 YY95
```

```
---- ---- ---- ----
2020 2020 2095 1995
```

【重要】
▶TO_DATE関数は、引数に指定された文字値を日付値に変換して戻す
▶時刻が指定されなかった場合は「午前0時0分0秒」として処理する
▶日にちが指定されなかった場合は「1日」として処理する
▶月が指定されない場合は「現在の月」として処理する
▶年が指定されない場合は「現在の年」として処理する
▶YY要素とRR要素は「世紀」の扱いが異なる

❐Column　文字列から日付への変換時の動作
～略～

■TO_NUMBER関数

文字値を数値に変換して戻す関数

```
SQL> select to_number('¥5,000,000', 'L9,999,999') * 2
  2  from dual;

TO_NUMBER('¥5,000,000','L9,999,999')*2
---------------------------------------
                              10000000
```

5.2 汎用関数と条件式

「汎用関数」と、SQL文の中にIF-THEN-ELSEロジックを記述する方法を解説

■主な汎用関数とその使い方

➤NVL関数
引数の「式1」に指定された値がNULL値以外の場合は「式1」を、NULL値の場合は「式2」を戻す関数
戻り値のデータ型は式1のデータ型と同じになります。
●NVL関数
NVL(式1,式2)

```
SQL> select ename, sal, comm, sal + comm, sal + NVL(comm, 0)
  2  from employees
  3  where deptno = 30;

ENAME             SAL       COMM   SAL+COMM SAL+NVL(COMM,0)
---------- ---------- ---------- ---------- ---------------
高橋           300000      30000     330000          330000
田中           355000      50000     405000          405000
伊藤           300000     140000     440000          440000
山本           285000                                285000
斉藤           150000          0     150000          150000
吉田           295000                                295000

6行が選択されました。
```

➤NVL2関数
引数の「式1」に指定された値がNULL値以外の場合は「式2」を、NULL値の場合は、「式3」を戻す関数です。
戻り値のデータ型は常に式2のデータ型と同じになります。(必要に応じて暗黙の型変換が行われ、変換出来ない場合はエラーになります)

```
SQL> select ename, sal, comm, NVL2(comm, sal+comm, sal)
  2  from employees
  3  where deptno = 30;

ENAME              SAL       COMM NVL2(COMM,SAL+COMM,SAL)
---------- ---------- ---------- ------------------------
高橋           300000      30000                   330000
田中           355000      50000                   405000
伊藤           300000     140000                   440000
山本           285000                              285000
斉藤           150000          0                   150000
吉田           295000                              295000

6行が選択されました。
```

➤NULLIF関数
引数に指定された2つの値を比較して、等しい場合は「NULL値」を戻し、等しくない場合は「式1」を戻す関数
式1にはリテラルのNULL値以外を指定する必要があります！！

●NULLIF関数
NULLIF(式1,式2)

```
SQL> select ename, sal, comm, NULLIF(comm, sal/10)
  2  from employees
  3  where comm IS NOT NULL;

ENAME              SAL       COMM NULLIF(COMM,SAL/10)
---------- ---------- ---------- --------------------
高橋           300000      30000
田中           355000      50000                50000
伊藤           300000     140000               140000
斉藤           150000          0                    0
```

➤COALESCE関数
引数に指定された式リストを先頭(左側)からチェックし、「最初に見つかったNULL値以外の値」を戻す関数
(すべての値がNULL値の場合はNULL値を戻す)
式リストに指定するデータは、すべて同じデータ型(数値、文字値、日付値)にする必要があります
(数値、文字、日付をまたがる暗黙的な型変換は行われません。)

```
SQL> select comm, mgr, ename, coalesce(comm, mgr, ename)
  2  from employees
  3  where deptno IN(10,30);
select comm, mgr, ename, coalesce(comm, mgr, ename)
                                  *
行1でエラーが発生しました。:
```

```
ORA-00932: データ型が一致しません: NUMBERが予想されましたがCHARです。


SQL>
SQL> select comm, mgr, ename, coalesce(to_char(comm), to_char(mgr), ename)
  2  from employees
  3  where deptno IN(10,30);

      COMM         MGR ENAME      COALESCE(TO_CHAR(COMM),TO_CHAR(MGR),ENAM
---------- ---------- ---------- ----------------------------------------
                      佐藤       佐藤
     30000       1007 高橋       30000
     50000       1007 田中       50000
    140000       1007 伊藤       140000
                 1001 山本       1001
                 1001 中村       1001
         0       1007 斉藤       0
                 1007 吉田       1007
                 1008 佐々木     1008

9行が選択されました。
```

【重要】
▶NVL関数は引数の「式1」に指定された値がNULL値以外の場合は「式1」を、NULL値の場合は式2を戻す
▶NVL2関数は、引数の「式1」に指定された値がNULL値以外の場合は「式2」を、NULL値の場合は「式3」を戻す
▶NULLIF関数は引数に指定された2つの値を比較して、等しい場合は「NULL値」を戻し、等しくない場合は「式1」を戻します
▶COALESCE関数は、引数に指定された式リストを先頭からチェックし、「最初に見つかったNULL値以外の値を戻す」
▶COALESCE関数の式リストには、同じデータ型(数値、文字値または、日付値)のデータを指定する必要がある。

■条件式とDECODE関数

分岐処理を行うDECODE関数。
同時にSQL文の中に「IF-THEN-ELSEロジック」を指定する方法についても習得する

➤CASE式
IF-THEN-ELSEロジックを実装出来ます
CASE式はANSI SQLに準拠しています

```
SQL> select deptno, ename, sal,
  2         case deptno when 10 then sal * 1.1
  3                     when 20 then sal * 1.2
  4                     else sal * 1.5
  5         end new_sal
  6  from employees
  7  where sal >= 300000
  8  order by deptno;

    DEPTNO ENAME            SAL     NEW_SAL
```

```
---------- ---------- ---------- ----------
        10 佐藤             500000     550000
        20 小林             300000     360000
        30 伊藤             300000     450000
        30 高橋             300000     450000
        30 田中             355000     532500
```

CASE式の中でリスト化された条件を順番に評価し、最初にTRUEを戻す条件に対応したデータを戻すCASE式のことを「検索CASE式」と呼びます。

```
SQL> select ename, sal,
  2             (case when sal < 230000 then 'A'
  3                   when sal < 380000 then 'B'
  4                   when sal < 480000 then 'C'
  5                   else 'D'
  6              end) SAL_LEVEL
  7  from employees
  8  order by sal_level;

ENAME                SAL S
---------- ---------- -
斉藤               150000 A
加藤               110000 A
鈴木               200000 A
伊藤               300000 B
佐々木              230000 B
中村               245000 B
小林               300000 B
渡辺               280000 B
田中               355000 B
高橋               300000 B
山本               285000 B
吉田               295000 B
山田               280000 B
佐藤               500000 D

14行が選択されました。
```

➤DECODE関数

> 今までの使用してきた、ほぼすべての DECODE は CASE 式で表現できる…
> しかし、重大な違いは NULL の扱いの違いにある。
>
> 単純 CASE 式は NULL との評価はすべて NULL になる。また ret_expr, default_expr に NULL の記述が許されていない、とある。(※)
>
> 一方 DECODE は NULL = NULL は True となる。
> NULL を直接評価、比較できるのは、NVL シリーズ、COALESCE などの、ごく限られた関数だけである。以下の書式は DECODE だけに許された記述である(検索 CASE 式 Oracle 9i ならば 書き換え可能である)。
>
> DECODE(nullable_col, 1, ‘ONE’, 2, ‘TWO’, NULL, ‘EMPTY’, ‘OTHERS’);
> DECODE(nullable_col, 1, ‘ONE’, 2, ‘TWO’, NULL);

http://www.shift-the-oracle.com/sql/case-decode-difference.html

ほぼほぼCASE式と同様の動作が可能だが上記の部分のみ異なっている！

```
SQL> select deptno, ename, sal,
  2             decode(deptno,10 ,sal * 1.1
  3                            ,20 ,sal * 1.2
  4                            ,sal * 1.5) NEW_ALL
  5  from employees
  6  where sal >= 300000
  7  order by deptno;

    DEPTNO ENAME             SAL    NEW_ALL
---------- ---------- ---------- ----------
        10 佐藤           500000     550000
        20 小林           300000     360000
        30 伊藤           300000     450000
        30 高橋           300000     450000
        30 田中           355000     532500
```

第6章 グループ関数とデータの集計

6.1 グループ関数(複数行関数)

グループ関数は「複数件の入力データをグループ化して、集計処理を行った結果を1つだけ戻す関数」です

■グループ関数の基本構文とタイプ

●グループ関数の基本構文
関数名([DISTINCT | ALL] {列名 | 式})

【重要】
▶グループ関数は、行のグループごとに集計した結果を1つだけ戻す
▶グループ関数はSELECT句、ORDER BY句、HAVING句で使用できる
▶グループ関数はWHERE句では使用できない
▶DISTINCTを指定すると重複した値は一回だけ処理される

グループ関数

➤COUNT関数

```
SQL> select count(*), count(comm), count(job), count(distinct job)
  2  from employees;

  COUNT(*) COUNT(COMM) COUNT(JOB) COUNT(DISTINCTJOB)
---------- ----------- ---------- ------------------
        14           4         14                  5
```

```
SQL> select count(distinct *)
  2  from employees;
select count(distinct *)
                     *
行1でエラーが発生しました。:
ORA-00936: 式がありません。


SQL> select count(ALL *)
  2  from employees;

COUNT(ALL*)
-----------
         14
```

➤MAX関数とMIN関数
最大値最小値を戻す関数。
引数には、数値型、文字型、日付型を戻す式または列を指定できます

```
SQL> select max(sal), min(sal)
  2  from employees;

  MAX(SAL)   MIN(SAL)
---------- ----------
    500000     110000
```

➤AVG関数とSUM関数
平均値と合計値を戻す関数。
引数には、数値型を戻す式または列のみ指定できます。

```
SQL> select avg(sal), sum(sal)
  2  from employees;

  AVG(SAL)   SUM(SAL)
---------- ----------
273571.429    3830000
```

❒Column　STDDEV関数、VARIANCE関数、LISTAGG関数

STDDEV関数：標準偏差を求める
VARIANCE関数：分散を求める
LISTAGG関数：各グループ内でデータを順序付けてデリミタを指定し、連結した結果を戻す

```
SQL> select deptno, avg(sal),
  2             listagg(ename, ';')
  3             within group (order by sal desc) member_list
  4  from employees
  5  group by deptno;

    DEPTNO   AVG(SAL) MEMBER_LIST
---------- ---------- ------------------------------------
        10     325000 佐藤;中村;佐々木
```

```
        20       234000 小林;山田;渡辺;鈴木;加藤
        30 280833.333 田中;伊藤;高橋;吉田;山本;斉藤
```

区切り文字を指定しないと、区切り文字無しで連結される！！

グループ関数のNULL値の扱い

基本的にグループ関数はNULL値を含めずに集計する

```
SQL> select ename, comm
  2  from employees;

ENAME             COMM
---------- ----------
佐藤
鈴木
高橋              30000
田中              50000
渡辺
伊藤             140000
山本
中村
小林
斉藤                  0
加藤
吉田
山田
佐々木

14行が選択されました。

SQL>
SQL>
SQL> select avg(comm), sum(comm)
  2  from employees;

 AVG(COMM)   SUM(COMM)
---------- ----------
     55000      220000
```

❐Column　Null値を「0」に置き換えて集計する
汎用関数を使用すればNULL値を0に置き換えて集計することができる

```
SQL> select avg(NVL(comm,0)), sum(NVL(comm,0))
  2  from employees;

AVG(NVL(COMM,0)) SUM(NVL(COMM,0))
---------------- ----------------
       15714.2857           220000
```

■グループ関数を使用した条件の指定

【重要】
▶グループ関数はWHERE句では指定できない

6.2 データのグループ化と取り出すグループの制限

- GROUP BY句には1つ以上の列を指定する必要がある
- 列別名は指定できない
- SELECT句の選択リストには「GROUP BY句で指定した列」と「グループ関数」のみ指定できる
- ORDER BY句とGROUP BY句を併用する場合、ORDER BY句には「GROUP BY句で指定した列」と「グループ関数」のみ指定できる

```
SQL> select deptno, count(*), avg(sal)
  2  from employees
  3  group by deptno;

    DEPTNO   COUNT(*)   AVG(SAL)
---------- ---------- ----------
        30          6 280833.333
        20          5      234000
        10          3      325000
```

▼2列以上の指定

```
SQL> select deptno, job, count(*), avg(sal)
  2  from employees
  3  group by deptno, job;

    DEPTNO JOB         COUNT(*)   AVG(SAL)
---------- --------- ---------- ----------
        10 社長               1     500000
        10 事務               1     230000
        20 事務               2     155000
        30 部長               1     285000
        10 部長               1     245000
        20 主任               2     290000
        20 部長               1     280000
        30 事務               1     295000
        30 営業               4     276250

9行が選択されました。
```

♪Tips
順序を指定しない場合はどのような順番で出力されるかは保証されない

➤グループ関数のネストとGROUP BY句

SELECT文にGROUP BY句の指定がある場合は、グループ関数は2レベルまでネストすることが出来ます

```
SQL> select max(avg(sal))
  2  from employees
  3  group  by deptno;
```

```
MAX(AVG(SAL))
-------------
       325000

-- これは指定出来ないみたい…
SQL> select deptno, max(avg(sal))
  2  from employees
  3  group  by deptno;
select deptno, max(avg(sal))
       *
行1でエラーが発生しました。:
ORA-00937: 単一グループのグループ関数ではありません。
```

➤GROUP BY句を指定したSELECT文のエラー
よくあるミスについてまとめ！！
●列別名は指定できない
GROUP BY句には列別名は指定できない！！
ORDER BY句には列別名を指定できるので合わせて覚えておく！

●SELECT句の選択リストにはGROUP BY句で指定した列とグループ関数のみ指定できる！

GROUP BY句に指定した列をSELECT句の選択リストに指定しなくてもエラーにはなりません

●ORDER BY句とGROUP BY句を併用する場合、ORDER BY句にはGROUP BY句で指定した列とグループ関数のみ指定できる

【重要】
▶
▶
P194

■HAVING句による取り出すグループの制限

HAVING句を使用すると、取り出すグループを制限することができる
順序はWHERE句の後ろ、かつORDER BY句の前に指定。
GROUP BY句とHAVING句は順序不同！！

GROUP BY句を指定せずにHAVING句のみを指定した場合は
選択された行全体が1つのグループとして処理されます

```
SQL> select count(*)
  2  from employees
  3  having count(*) > 1;

  COUNT(*)
----------
        14
```

▼検証用
重複行を削除した後平均を求める動きになる！！！

```
SQL> select avg(distinct sal)
  2  from employees;

AVG(DISTINCTSAL)
----------------
      268181.818
```

第7章 複数の表からのデータの取り出し

7.1 複数の表の結合

■結合とは？

➤基本的な表の結合方法
FROM句に表の名前を「,」(カンマ)で区切って指定して表を結合する方法
(この結合法はOracle独自の結合表の一つ)

```
SQL> select empno, ename, dname
  2  from employees, departments;

     EMPNO ENAME      DNAME
---------- ---------- --------------
      1001 佐藤       管理
      1002 鈴木       管理
      1003 高橋       管理
      1004 田中       管理
      1005 渡辺       管理
      1006 伊藤       管理
      1007 山本       管理
      1008 中村       管理
      1009 小林       管理
      1010 斉藤       管理
      1011 加藤       管理
      1012 吉田       管理
      1013 山田       管理
      1014 佐々木     管理
      1001 佐藤       研究開発
      1002 鈴木       研究開発
      1003 高橋       研究開発

     EMPNO ENAME      DNAME
---------- ---------- --------------
      1004 田中       研究開発
      1005 渡辺       研究開発
      1006 伊藤       研究開発
      1007 山本       研究開発
      1008 中村       研究開発
      1009 小林       研究開発
      1010 斉藤       研究開発
      1011 加藤       研究開発
      1012 吉田       研究開発
      1013 山田       研究開発
```

```
      1014 佐々木      研究開発
      1001 佐藤        営業
      1002 鈴木        営業
      1003 高橋        営業
      1004 田中        営業
      1005 渡辺        営業
      1006 伊藤        営業

     EMPNO ENAME      DNAME
---------- ---------- --------------
      1007 山本        営業
      1008 中村        営業
      1009 小林        営業
      1010 斉藤        営業
      1011 加藤        営業
      1012 吉田        営業
      1013 山田        営業
      1014 佐々木      営業
      1001 佐藤        財務
      1002 鈴木        財務
      1003 高橋        財務
      1004 田中        財務
      1005 渡辺        財務
      1006 伊藤        財務
      1007 山本        財務
      1008 中村        財務
      1009 小林        財務

     EMPNO ENAME      DNAME
---------- ---------- --------------
      1010 斉藤        財務
      1011 加藤        財務
      1012 吉田        財務
      1013 山田        財務
      1014 佐々木      財務

56行が選択されました。
```

※SELECT句に指定している列が表のどちらにも存在する場合エラーになる。

➤接頭辞を使用した列名の修飾

●表接頭辞を使用した列名の修飾
表名.列名

表接頭辞は重複する列名がなくても使用可能で、
使用すると、Oracleサーバーが SQL文の構文を解析する際に列名の重複の有無をチェックする必要がなくなる為、
僅かですが**SQL文のパフォーマンスが向上**します！
またSELECT文がわかりやすくなるというメリットもあります。

```
SQL> select employees.empno, employees.ename,
  2         departments.dname, employees.deptno
  3  from employees, departments;

     EMPNO ENAME      DNAME              DEPTNO
```

```
---------- ---------- -------------- ----------
      1001 佐藤       管理                   10
      1002 鈴木       管理                   20
      1003 高橋       管理                   30
      1004 田中       管理                   30
      1005 渡辺       管理                   20
      1006 伊藤       管理                   30
      1007 山本       管理                   30
      1008 中村       管理                   10
      1009 小林       管理                   20
      1010 斉藤       管理                   30
      1011 加藤       管理                   20
      1012 吉田       管理                   30
      1013 山田       管理                   20
      1014 佐々木     管理                   10
      1001 佐藤       研究開発               10
      1002 鈴木       研究開発               20
      1003 高橋       研究開発               30

     EMPNO ENAME      DNAME          DEPTNO
---------- ---------- -------------- ----------
      1004 田中       研究開発               30
      1005 渡辺       研究開発               20
      1006 伊藤       研究開発               30
      1007 山本       研究開発               30
      1008 中村       研究開発               10
      1009 小林       研究開発               20
      1010 斉藤       研究開発               30
      1011 加藤       研究開発               20
      1012 吉田       研究開発               30
      1013 山田       研究開発               20
      1014 佐々木     研究開発               10
      1001 佐藤       営業                   10
      1002 鈴木       営業                   20
      1003 高橋       営業                   30
      1004 田中       営業                   30
      1005 渡辺       営業                   20
      1006 伊藤       営業                   30

     EMPNO ENAME      DNAME          DEPTNO
---------- ---------- -------------- ----------
      1007 山本       営業                   30
      1008 中村       営業                   10
      1009 小林       営業                   20
      1010 斉藤       営業                   30
      1011 加藤       営業                   20
      1012 吉田       営業                   30
      1013 山田       営業                   20
      1014 佐々木     営業                   10
      1001 佐藤       財務                   10
      1002 鈴木       財務                   20
      1003 高橋       財務                   30
      1004 田中       財務                   30
      1005 渡辺       財務                   20
      1006 伊藤       財務                   30
      1007 山本       財務                   30
      1008 中村       財務                   10
      1009 小林       財務                   20
```

```
     EMPNO ENAME      DNAME          DEPTNO
---------- ---------- -------------- ----------
      1010 斉藤        財務                30
      1011 加藤        財務                20
      1012 吉田        財務                30
      1013 山田        財務                20
      1014 佐々木      財務                10

56行が選択されました。
```

➤表別名の使用
FROM句内で表別名を指定できる
「ASキーワード」は使用できないので注意
以下の点に注意して使用すること！！

- 表別名は３０バイト以下
- 一般的に表別名はわかりやすく、短い名前を指定する
- 表接頭辞に短い表別名を使用するとSQL文が短くなるため、メモリーの使用料を削減できる
- 表別名は、表別名を指定したSQL文内でのみ有効

なお表別名を指定すると、SQL文全体で元の表名は無効になっているので注意！！！

【重要】
▶結合する表に重複する列名がある場合は、表接頭辞を指定する
▶表接頭辞を使用すると、SQL文のパフォーマンスが向上する
▶表接頭辞には表別名が指定できる
▶表別名を指定すると、SQL文全体で元の表名は無効になる

■結合構文の種類

Oracleサーバーでは以下の２種類の結合構文に含まれるいろいろな方法で表を結合できる

- SQL：１９９９規格に準拠した結合構文
- Oracle独自の結合構文

➤SQL:1999結合構文の結合のタイプ

●SQL:1999結合構文
SELECT [表名.]列名 [,[表名.]列名…]
FROM 表名1
{
NATURAL JOIN 表名2 |
JOIN 表名2 USING(列名) |
JOIN 表名2 ON 結合条件 |
{LEFT | RIGHT | FULL} [OUTER] JOIN 表名2 ON 結合条件 |
CROSS JOIN 表名2
}

7.2 等価結合

等価結合：結合する2つのひょうの「**特定の列の値が等しいデータだけを取り出す結合**」
「内部結合」や「単純結合」と呼ばれることもある

厳密には「内部結合」=「等価結合」ではない
内部結合は後述の「非等価結合」が含まれる

■自然結合

自然結合：結合する2つの表に共通して存在する同じ列名かつ同じデータ型の列に基づいて表を結合する等価結合
「NATURAL JOIN句」の前後に結合する表名を指定します

特徴

- 自然結合では結合列が自動で判断される為、明示的に結合条件を指定する必要はない
- 共通して存在する列が複数ある場合は、**すべての列が結合条件として使用**される
- 自然結合では、**データ型の異なる同名の列があるとエラー**になる

```
SQL> select empno, ename dname
  2  from employees natural join departments;

     EMPNO DNAME
---------- ---------
      1001 佐藤
      1002 鈴木
      1003 高橋
      1004 田中
      1005 渡辺
      1006 伊藤
      1007 山本
      1008 中村
      1009 小林
      1010 斉藤
      1011 加藤
      1012 吉田
      1013 山田
      1014 佐々木

14行が選択されました。
```

➤共通して存在する列が複数ある場合の自然結合
それらのすべての値が等しいデータのみが取り出されます

➤自然結合で接頭辞を使用する際の注意点
SQL1999結合構文でも多くの場合で表接頭辞を使用して列名を修飾出来ますが、自然結合の結合列には表接頭辞を使用できません。

```
SQL> select e.empno, e.ename, d.deptno, d.dname
  2  from employees e natural join departments d;
select e.empno, e.ename, d.deptno, d.dname
                         *
行1でエラーが発生しました。:
ORA-25155: NATURAL結合で使用される列は修飾子を持てません。
```

表接頭辞（d）を削除すれば正常に実行出来ます！

またWHERE句に結合列を指定する場合も同様です。
WHERE句に指定した結合列に表接頭辞を指定するとエラーになります。

```
SQL> select e.empno, e.name, d.name
  2  from employees e natural join departments d
  3  where e.deptno IN(10,20);
where e.deptno IN(10,20)
      *
行3でエラーが発生しました。:
ORA-25155: NATURAL結合で使用される列は修飾子を持てません。
```

【重要】
▶自然結合とは、結合する2つの表に対して共通して存在するすべての列にもとづいて表を結合する等価結合である
▶自然結合では、明示的に結合条件を指定する必要はない
▶自然結合では、**データ型の異なる同名の列があるとエラーになる！！**
▶自然結合の**結合列には、表接頭辞を指定できない！！！**

■USING句を使用した結合

結合列を明示的に指定できる
USING句は以下のように場合に使用する

- 結合列を明示的に指定してSQL文をわかりやすくした場合
- 結合する2つの表に共通して存在する列が複数ある場合に、そのいずれかを結合列として使用した場合
- 列名が同じで**データ型が異なる列を結合列として使用する**場合

```
SQL> select empno, ename, dname
  2  from employees join departments using(deptno);

     EMPNO ENAME      DNAME
---------- ---------- --------------
      1001 佐藤       管理
      1014 佐々木     管理
      1008 中村       管理
      1013 山田       研究開発
      1005 渡辺       研究開発
      1011 加藤       研究開発
      1002 鈴木       研究開発
      1009 小林       研究開発
      1012 吉田       営業
      1010 斉藤       営業
      1006 伊藤       営業
      1004 田中       営業
      1003 高橋       営業
      1007 山本       営業

14行が選択されました。
```

➤共通して存在する列が複数ある場合のUSING句を使用した結合

➤表接頭辞を使用した列名の修飾
自然結合の場合と同様に、USING句を使用した結合でも、

結合列に表接頭辞を使用するとエラーになるので覚えておく

➤NATURAL JOIN句とUSING句は同時に使用できない

【重要】
▶USING句を使用すると、結合列を明示的に指定できる
▶結合する2つの表に共通して存在する列が複数ある場合も、USING句で結合する列を指定できる
▶USING句の結合列には表接頭辞は指定できない（自然結合と同様）
▶1つの結合に対して、NATURAL JOIN句（自然結合）とUSING句を同時に使用することは出来ない

■ON句を使用した結合

異なる名前の列を使用して表を結合することができる！！

♪Tips
NATURAL JOIN句やUSING句は等価結合でしか使用されませんが、ON句は後述する
「非等価結合」や「自己結合」などでも使用される汎用的かつ重要な構文！！

```
SQL> select empno, ename, dname
  2  from employees join departments
  3  on employees.deptno = departments.deptno
  4  ;

     EMPNO ENAME      DNAME
---------- ---------- --------------
      1001 佐藤       管理
      1014 佐々木     管理
      1008 中村       管理
      1013 山田       研究開発
      1005 渡辺       研究開発
      1011 加藤       研究開発
      1002 鈴木       研究開発
      1009 小林       研究開発
      1012 吉田       営業
      1010 斉藤       営業
      1006 伊藤       営業
      1004 田中       営業
      1003 高橋       営業
      1007 山本       営業

14行が選択されました。
```

➤接頭辞を使用した同じ名前の列の修飾
自然結合やUSING句を使用した結合では結合列に表接頭辞を使用できませんでしたが、
ON句を使用した場合では、2つのひょうにある同じ名前の列をSEELCT句やWHERE句などで使用する場合は、表接頭辞を使用して列名を修飾する必要があります。
この点もON句を使用した結合の特徴なので覚えておく！！

```
SQL> select e.empno, e.ename, d.dname, e.deptno
  2  from employees e join departments d
  3  on e.deptno = d.deptno
  4  where e.deptno IN(10,20);
```

```
     EMPNO ENAME      DNAME          DEPTNO
---------- ---------- -------------- ----------
      1001 佐藤       管理               10
      1002 鈴木       研究開発           20
      1005 渡辺       研究開発           20
      1008 中村       管理               10
      1009 小林       研究開発           20
      1011 加藤       研究開発           20
      1013 山田       研究開発           20
      1014 佐々木     管理               10

8行が選択されました。
```

【重要】
▶ON句を使用した結合では、結合条件はON句に指定する
▶ON句を使用した結合では、名前の異なる列を使用して表を結合することができる
▶ON句を使用した結合では、名前が同じ列がある場合は、表接頭辞で修飾する必要がある

Oralce独自結合構文による等価結合

●結合条件はWHERE句に指定する
●2つの表に共通して存在する列は、表接頭辞を使用して修飾する必要がある

```
SQL> select e.empno, e.ename, d.dname
  2  from employees e, departments d
  3  where e.deptno = d.deptno
  4  and d.deptno IN(10,20);

     EMPNO ENAME      DNAME
---------- ---------- --------------
      1001 佐藤       管理
      1002 鈴木       研究開発
      1005 渡辺       研究開発
      1008 中村       管理
      1009 小林       研究開発
      1011 加藤       研究開発
      1013 山田       研究開発
      1014 佐々木     管理

8行が選択されました。
```

【重要】
▶SQL:1999結合構文とOracle独自結合構文に**パフォーマンス上の差はない**

■3つ以上の表の結合

■自然結合やUSING句を使用した3つ以上の表の結合

【重要】
▶Oracleサーバーでは3つ以上の表を結合できる
▶結合列の名前が同じで、データ型も同じ場合は、自然結合（NATURAL JOIN句）やUSING句を使用した結合でも、3つ以上の表を結合できる
▶3つ以上の表を結合する場合は、1つのSQL文にNATURAL JOIN句とUSING句、ON句を同時に使用できる！！

7.3 その他の結合構文

SQL:1999結合構文を使用した「非等価結合」「自己結合」「外部結合」「クロス結合」について解説。

■非等価結合

非等価結合：結合条件に「＝」（等価演算子）以外の演算子を使用して、条件を満たすデータを取り出す結合です。

```
SQL> select e.empno, e.ename, e.sal, sg.grade
  2  from employees e join salgrades sg
  3  on e.sal between sg.losal and sg.hisal;

     EMPNO ENAME              SAL G
---------- ---------- ---------- -
      1011 加藤           110000 A
      1010 斉藤           150000 A
      1002 鈴木           200000 B
      1014 佐々木         230000 B
      1008 中村           245000 B
      1013 山田           280000 B
      1005 渡辺           280000 B
      1007 山本           285000 C
      1012 吉田           295000 C
      1009 小林           300000 C
      1003 高橋           300000 C
      1006 伊藤           300000 C
      1004 田中           355000 C
      1001 佐藤           500000 E

14行が選択されました。
```

【重要】
▶等価結合とは、結合条件に「＝」(等価演算子)以外の演算子を使用して、条件を満たすデータを取り出す結合である

■自己結合

同一のひょうに2つの表別名を指定することで1つの表を2つの表に見立ててデータを取り出す結合

♪Tips
自然結合(NATURAL JOIN句)やUSING句を使用した結合では、同じ名前の列を結合列に指定する必要がある為、前述のような自己結合は出来ません。

```
SQL> select w.empno, w.ename, m.empno, m.ename
  2  from employees w join employees m
  3  on w.mgr = m.empno
  4  order by w.empno;

     EMPNO ENAME          EMPNO ENAME
---------- ---------- ---------- ----------
      1002 鈴木             1013 山田
      1003 高橋             1007 山本
      1004 田中             1007 山本
```

```
      1005 渡辺             1001 佐藤
      1006 伊藤             1007 山本
      1007 山本             1001 佐藤
      1008 中村             1001 佐藤
      1009 小林             1005 渡辺
      1010 斉藤             1007 山本
      1011 加藤             1009 小林
      1012 吉田             1007 山本
      1013 山田             1005 渡辺
      1014 佐々木           1008 中村

13行が選択されました。
```

【重要】
▶自己結合とは、同一の表に2つの表別名を指定することで1つの表を2つの表に見立ててデータを取り出す結合である。

■外部結合

結合条件を満たしたデータだけでなく、結合条件を満たしていないデータも取り出す結合のこと。

「内部結合」：結合条件を満たしたデータのみを取り出す結合
「外部結合」：条件を満たしていないデータも取り出す結合
OUTERキーワードは省略可能

構文	種類
LEFT [OUTER] JOIN	左側外部結合
RIGHT [OUTER] JOIN	右側外部結合
FULL [OUTER] JOIN	完全外部結合

➤左側外部結合

➤右側外部結合

➤完全外部結合
両テーブルに存在するデータすべてが表示対象

■Column　Oracle独自結合構文による外部結合
Oracle独自結合構文で外部結合を行うには、「(+)」(外部結合演算子)を使用します
「(+)」をWHERE句の条件の左側につけると右側外部結合と同様の結果に、
右側につけると左側外部結合と同様の結果になります。

```
SQL> select e.empno, e.ename, d.deptno, d.dname
  2  from employees e, departments d
  3  where e.deptno = d.deptno(+);

     EMPNO ENAME          DEPTNO DNAME
---------- ---------- ---------- --------------
      1014 佐々木             10 管理
      1008 中村               10 管理
      1001 佐藤               10 管理
      1013 山田               20 研究開発
```

```
      1011 加藤                  20 研究開発
      1009 小林                  20 研究開発
      1005 渡辺                  20 研究開発
      1002 鈴木                  20 研究開発
      1012 吉田                  30 営業
      1010 斉藤                  30 営業
      1007 山本                  30 営業
      1006 伊藤                  30 営業
      1004 田中                  30 営業
      1003 高橋                  30 営業

14行が選択されました。
```

外部結合演算子をつけなかった方のデータがすべて表示されること！！！

また、Oracle独自結合構文には完全外部結合に相当する結合構文はない

【重要】
▶外部結合とは、結合条件を満たしたデータだけでなく、条件を満たしていないデータも取り出す結合である
▶外部結合には「左側外部結合」「右側外部結合」「完全外部結合」の3つの結合方法がある

■クロス結合

クロス結合：デカルト積(結合する表に格納されているデータのすべての組み合わせ。直積ともいう)
クロス結合の実行結果には多くの「意味のないデータ」が含まれる為、通常はあまり使用しませんが、大量のサンプル・データを生成する際などに使用することがありますので覚えておく

「CROSS JOINキーワード」を使用する
Tip
Oracle独自結合構文では、FROM句に複数の表名を記述し、WHERE句に結合条件を記述しない場合にデカルト積(直積)が戻されます。

【重要】
▶クロス結合とは、デカルト積(直積)を戻す結合
▶デカルト積とは、結合する表に格納されているデータのすべての組み合わせである

```
SQL> select e.empno, e.ename, d.dname
  2  from employees e cross join departments d;

     EMPNO ENAME      DNAME
---------- ---------- --------------
      1015 山口       管理
      1001 佐藤       管理
      1002 鈴木       管理
      1003 高橋       管理
      1004 田中       管理
      1005 渡辺       管理
      1006 伊藤       管理
      1007 山本       管理
      1008 中村       管理
      1009 小林       管理
      1010 斉藤       管理
      1011 加藤       管理
      1012 吉田       管理
```

```
      1013 山田         管理
      1014 佐々木       管理
      1015 山口         研究開発
      1001 佐藤         研究開発

     EMPNO ENAME      DNAME
---------- ---------- --------------
      1002 鈴木         研究開発
      1003 高橋         研究開発
      1004 田中         研究開発
      1005 渡辺         研究開発
      1006 伊藤         研究開発
      1007 山本         研究開発
      1008 中村         研究開発
      1009 小林         研究開発
      1010 斉藤         研究開発
      1011 加藤         研究開発
      1012 吉田         研究開発
      1013 山田         研究開発
      1014 佐々木       研究開発
      1015 山口         営業
      1001 佐藤         営業
      1002 鈴木         営業
      1003 高橋         営業

     EMPNO ENAME      DNAME
---------- ---------- --------------
      1004 田中         営業
      1005 渡辺         営業
      1006 伊藤         営業
      1007 山本         営業
      1008 中村         営業
      1009 小林         営業
      1010 斉藤         営業
      1011 加藤         営業
      1012 吉田         営業
      1013 山田         営業
      1014 佐々木       営業
      1015 山口         財務
      1001 佐藤         財務
      1002 鈴木         財務
      1003 高橋         財務
      1004 田中         財務
      1005 渡辺         財務

     EMPNO ENAME      DNAME
---------- ---------- --------------
      1006 伊藤         財務
      1007 山本         財務
      1008 中村         財務
      1009 小林         財務
      1010 斉藤         財務
      1011 加藤         財務
      1012 吉田         財務
      1013 山田         財務
      1014 佐々木       財務

60行が選択されました。
```

第8章 副問い合わせによる問い合わせの解決方法

8.1 副問い合わせの基本

➤副問い合わせのメリット
1回のSQL文でほしい情報が取り出せる

```
SQL> select empno, ename, sal
  2  from employees
  3  where sal >= ( select sal from employees where empno = 1003);

     EMPNO ENAME              SAL
---------- ---------- ----------
      1001 佐藤           500000
      1003 高橋           300000
      1004 田中           355000
      1006 伊藤           300000
      1009 小林           300000
```

上記ではWHERE句に副問い合わせをしてしていますが、
同様の構文でHAVING句やFROM句に指定することも出来ます。
通常は見やすくするために比較演算子の右辺に記述しますが、
左辺に記述することも可能です。

【重要】
▶副問い合わせは「()」(丸括弧)で囲む

副問合せのいろいろな使用方法

➤主問合せと異なる表を問い合わせる副問合せ

➤複数の副問合せを使用する主問合せ

➤副問合せのWHERE句以外で使用

```
SQL> select e.deptno, d.dname, avg(sal)
  2  from employees e join departments d
  3  on e.deptno = d.deptno
  4  group by e.deptno, d.dname
  5  having avg(sal) >= (select avg(sal) from employees)
  6  order by e.deptno;

    DEPTNO DNAME          AVG(SAL)
---------- ---------- ----------
        10 管理           325000
        30 営業        280833.333
```

➤FROM句での副問合せの使用

```
SQL> select empno, ename, sal
  2  from (select * from employees where deptno = 30);

     EMPNO ENAME              SAL
---------- ---------- ----------
      1003 高橋           300000
      1004 田中           355000
      1006 伊藤           300000
      1007 山本           285000
      1010 斉藤           150000
      1012 吉田           295000

6行が選択されました。
```

この方法はデータベースオブジェクトの1つである「ビュー」(SELECT文に名前をつけてOracleデータベースに保存したもの)と似ているため、「**インラインビュー**」と呼ばれることもある

♪Tips
副問合せはSQL分の様々な句で使用できますが、**GROUP BY句では使用できません**のでしっかりとおぼえておくこと！！

➤データを1件も戻さない副問合せ
副問合せの実行結果が0件の場合、主問合せにはNULL値が戻される為、主問合せの実行結果も0件になる

❐Column　副問合せのネスト
WHERE句では255レベルまでネストでき、FROM句では無限にネスト出来ます。

一般的にネストレベルが深くなるとSQL文がわかりにくくなり、またパフォーマンスも悪くなる為、副問合せのネストは可能であれば表の結合などに変換することが推奨されています。！

8.2 単一行副問合せと複数行副問合せ

■単一行副問合せ

データを1件だけ戻す副問合せです。

等しくないの表現
<>, !=, ^=

■複数行副問合せ

➤IN演算子を使用した複数行副問合せ
比較対象の列の値がリスト内のいずれかの値と等しい場合にTRUEを戻します。

```
SQL> select empno, ename, job, mgr, deptno
  2  from employees
  3  where empno IN(select mgr from employees where ename in('山田', '伊藤'));

     EMPNO ENAME      JOB              MGR     DEPTNO
---------- ---------- --------- ---------- ----------
      1005 渡辺       部長            1001         20
      1007 山本       部長            1001         30
```

➤ANY演算子を使用した複数行副問合せ
リスト内のいずれかの値が条件を満たす場合にTRUEを戻します。
ANY演算子は必ず単一行演算子とセットで使用します。
なお「＝ANY(値のリスト)」は「IN(値のリスト)」と同じ意味になります。
「＞ANY(値のリスト)」はリスト内の値のいずれか1つよりも大きければTRUEを戻すので、
ここでは「リスト内の最小値よりも大きい場合」という意味になります

♪Tips
ANY演算子の代わりにSOME演算子を使用することも出来ます。

```
SQL> select empno, ename, sal, deptno
  2  from employees
  3  where sal > ANY(select avg(sal) from employees group by deptno);

     EMPNO ENAME              SAL     DEPTNO
---------- ---------- ---------- ----------
      1001 佐藤           500000         10
      1004 田中           355000         30
      1009 小林           300000         20
      1003 高橋           300000         30
      1006 伊藤           300000         30
      1012 吉田           295000         30
      1007 山本           285000         30
      1013 山田           280000         20
      1005 渡辺           280000         20
      1008 中村           245000         10
      1014 佐々木          230000         10
      1002 鈴木           200000         20
      1010 斉藤           150000         30
      1011 加藤           110000         20

14行が選択されました。
```

♪Tips
グループ関数は2レベルまでネストできるので、上記のSQL文は以下のように書き換えることができる

```
SQL> select empno, ename, sal, deptno
  2  from employees
  3  where sal > (select min(avg(sal)) from employees group by deptno);

     EMPNO ENAME              SAL     DEPTNO
---------- ---------- ---------- ----------
      1001 佐藤           500000         10
      1002 鈴木           200000         20
      1003 高橋           300000         30
      1004 田中           355000         30
      1005 渡辺           280000         20
      1006 伊藤           300000         30
      1007 山本           285000         30
      1008 中村           245000         10
      1009 小林           300000         20
      1010 斉藤           150000         30
      1011 加藤           110000         20
      1012 吉田           295000         30
```

```
      1013 山田            280000          20
      1014 佐々木          230000          10

14行が選択されました。
```

➤ALL演算子を使用した複数行副問合せ
すべての値が条件を満たす場合にTRUEを戻します。

```
SQL> select empno, ename, sal, deptno
  2  from employees
  3  where sal > all(select avg(sal) from employees group by deptno);

     EMPNO ENAME              SAL     DEPTNO
---------- --------- ---------- ----------
      1004 田中            355000          30
      1001 佐藤            500000          10
```

♪Tips
複数行副問合せであれば単一行演算子を使用可能
事前に副問合せの戻す検数が把握出来ない場合は、複数行副問合せを使用しておけば、
副問合せが戻すデータ件数に依存しない汎用的なSQL文を記述出来ます。

■副問合せにおけるIN演算子とNULL値

副問合せがNULL値を戻した場合、主問合せの実行結果は0件になると解説したが、
主問合せの条件式の比較演算子に「NOT IN」を使用する場合は注意が必要
「NOT IN(値リスト)」は「<>ALL(値のリスト)」と同等となります。
つまり戻されたすべての値に対してFALSEにならないかぎり主問合せの実行結果は0件になります
しかし列値とNULL値を比較するとFALSEではなくNULL値。
なので副問合せでNULL値を取り除いて置く必要がある

```
SQL> select empno, ename
  2  from employees
  3  where empno not in(select mgr from employees where mgr is not null);

     EMPNO ENAME
---------- ---------
      1002 鈴木
      1003 高橋
      1004 田中
      1006 伊藤
      1010 斉藤
      1012 吉田
      1014 佐々木
      1015 山口

8行が選択されました。
```

【重要】
▶「NOT IN(リスト)」は「<>ALL(リスト)」と同じである
▶「NOT IN(リスト)」はリスト内のすべての値と等しく内場合にTRUEを戻す
▶「NOT IN(リスト)」はリストにNULL値が含まれるとデータは1件も戻されない

第9章 集合演算子の使用方法

9.1 集合演算子の種類と使用方法

複数の問い合わせ結果を1つに纏めることが出来ます。
集合演算子を含む問い合わせのことを「複合問い合わせ」と呼びます。
Oracleのサーバーでは以下の4つの集合演算子を使用できます。

集合演算子	説明
UNION	2つの問い合わせ結果を連結し、重複した行を排除して戻す
UNION ALL	2つの問い合わせ結果を連結し、重複した行も含めて戻す
INTERSECT	2つの問い合わせ結果のうち、共通する行だけ戻す
MINUS	1つ目の問い合わせ結果のうち、2つ目の問い合わせ結果にない行を戻す

【重要】
▶集合演算子は複数の問い合わせの結果を1つに纏める為に使用する
▶集合演算子を含む問い合わせを「複合問い合わせ」と呼ぶ

■UNION演算子

【重要】
▶UNION演算子は2つの問い合わせ結果を連結し、重複した行を排除して戻す
▶UNION ALL演算子以外の集合演算子を使用すると、問い合わせ結果はSELECT句に指定されている
列の値で昇順にソートされる(デフォルトの場合)

■UNION ALL演算子

＊ポイント❣
UNION ALL演算子はソート処理を行いません！！
(重複した行を排除しないのでデータをソートする必要が無いため)
ORDER BY句を使用していない限り、1つ目の問い合わせ結果に、2つ目の問い合わせ結果を追加したものになる。

【重要】
▶UNION ALL演算子は、2つの問い合わせ結果を連結し、重複した行も含めて戻す
▶UNION ALL演算子は、自動的に問い合わせ結果をソートしない

■INTERSECT演算子

2つの問い合わせ結果のうち、共通する行だけを戻します。
INTERSECT演算子はUNION演算子と同様で、内部的にデータをソートしたうえで重複した行を戻す動きなるので実行結果もSELECT句の先頭に指定されている列の値でソートされます。

```
SQL> select deptno, empno, ename
  2  from employees
  3  where deptno in(10,20)
  4  intersect
  5  select deptno, empno, ename
```

```
  6  from employees
  7  where deptno in(20,30);

    DEPTNO      EMPNO ENAME
---------- ---------- ----------
        20       1002 鈴木
        20       1005 渡辺
        20       1009 小林
        20       1011 加藤
        20       1013 山田
```

【重要】
▶INTERSECT演算子は2つの問い合わせ結果のうち、共通する行だけを戻す

■MINUS演算子

1つ目の問い合わせ結果のうち、2つ目の問い合わせ結果にない行を返します。
内部的にデータをソートしている。

```
SQL> select deptno, empno, ename
  2  from employees
  3  where deptno in(10,20)
  4  minus
  5  select deptno, empno, ename
  6  from employees
  7  where deptno in(20,30);

    DEPTNO      EMPNO ENAME
---------- ---------- ----------
        10       1001 佐藤
        10       1008 中村
        10       1014 佐々木
```

【重要】
▶MINUS演算子は1つ目の問い合わせ結果のうち、2つ目の問い合わせ結果にない行を戻す
▶MINUS演算子を使用している場合は、1つ目の問い合わせ結果と2つ目の問い合わせの順番を入れ替えると問い合わせ結果が変わる

9.2 集合演算子の使用に関するガイドライン

■SELECT句に指定する列の個数とデータ型

```
SQL> -- 集合演算子の複合問い合わせ
SQL> select empno from employees
  2  union
  3  select deptno from employees;

     EMPNO
----------
        10
        20
        30
      1001
```

```
      1002
      1003
      1004
      1005
      1006
      1007
      1008
      1009
      1010
      1011
      1012
      1013
      1014

17行が選択されました。
```

【重要】
▶複合問い合わせではSELECT句に指定する列(または式)の個数を2つの問い合わせで同数にする必要がある
▶2つ目の問い合わせのSELECT句に指定する列のデータ型は1つ目の問い合わせのSELECT句に指定されている列のデータ型と同じデータ型グループである必要がある。
▶それぞれに指定する列の名前は異なっていても構わないが、問い合わせ結果の列見出しには、1つ目の問い合わせに指定されている列名が表示される

❐Column　変換関数を使用した問い合わせ結果の連結

```
SQL> select empno, ename, hiredate, to_char(null)
  2  from employees
  3  union
  4  select deptno, dname, to_date(null), loc
  5  from departments;

     EMPNO ENAME          HIREDATE TO_CHAR(NU
---------- -------------- -------- ----------
        10 管理                    大手町
        20 研究開発                横浜
        30 営業                    品川
        40 財務                    東京
      1001 佐藤           01-02-25
      1002 鈴木           00-03-26
      1003 高橋           00-05-30
      1004 田中           02-06-02
      1005 渡辺           02-07-11
      1006 伊藤           08-01-06
      1007 山本           00-08-09
      1008 中村           00-09-17
      1009 小林           06-10-21
      1010 斉藤           01-12-17
      1011 加藤           06-10-21
      1012 吉田           09-03-13
      1013 山田           01-03-13

     EMPNO ENAME          HIREDATE TO_CHAR(NU
---------- -------------- -------- ----------
      1014 佐々木          04-05-02
```

```
18行が選択されました。
```

■複合問い合わせでのORDER BY句の使用

- ORDER BY句は、複合問い合わせの最後の問い合わせに指定する必要がある
- ORDER BY句には、最初の問い合わせのSELECTに指定されている列または列別名を指定する必要がある

```
SQL> select empno from employees
  2  union
  3  select deptno from employees
  4  order by empno desc;

     EMPNO
----------
      1014
      1013
      1012
      1011
      1010
      1009
      1008
      1007
      1006
      1005
      1004
      1003
      1002
      1001
        30
        20
        10
```

【重要】
▶ORDER BY句は、複合問い合わせの最後の問い合わせに指定する必要がある
▶ORDER BY句には、最初の問い合わせのSELECTに指定されている列または列別名を指定する必要がある

■複合問い合わせとNULL値

複合問合せではNULL値は無視されません

```
SQL> select mgr from employees
  2  union
  3  select comm from employees;

       MGR
----------
         0
      1001
      1005
      1007
```

```
      1008
      1009
      1013
     30000
     50000
    140000
                    ←ここはNULL値が1つ入っている

11行が選択されました。
```

■集合演算子の優先順位

集合演算子の優先順位はすべて同じです。
そのため1つのSQL文に複数の集合演算子が使用された場合は前にしてされている複合問合せから順番に実行されます。
「()」(丸括弧)を使用して優先順位を明示的に指定することも出来ます。

```
SQL> select deptno, empno, ename
  2  from employees
  3  where deptno in(20,30)
  4  union all
  5  select deptno, empno, ename
  6  from employees
  7  where deptno in(10,20)
  8  intersect
  9  select deptno, empno, ename
 10  from employees
 11  where deptno in(20,30);

    DEPTNO      EMPNO ENAME
---------- ---------- ----------
        20       1002 鈴木
        20       1005 渡辺
        20       1009 小林
        20       1011 加藤
        20       1013 山田
        30       1003 高橋
        30       1004 田中
        30       1006 伊藤
        30       1007 山本
        30       1010 斉藤
        30       1012 吉田

11行が選択されました。
```

P291参照

第10章 データ操作とトランザクション制御

10.1 DML文によるデータの追加・更新・削除

■事前準備

テーブルのコピー作業
create table emp_copy as select * from employees;
create table dept_copy as select * from departments;

■INSERT文によるデータの追加

➤列名の指定を省略したINSERT文

➤列名を明示的に指定したINSERT文

```
SQL> insert into dept_copy values(50, '教育', '大手町');

1行が作成されました。

SQL>
SQL>
SQL>
SQL> insert into dept_copy(dname, deptno, loc) values('システム', 60, '横浜');

1行が作成されました。

SQL> select * from dept_copy
  2  ;

    DEPTNO DNAME          LOC
---------- -------------- ----------
        10 管理           大手町
        20 研究開発       横浜
        30 営業           品川
        40 財務           東京
        50 教育           大手町
        60 システム       横浜

6行が選択されました。
```

【重要】
▶INSERT句に指定する列名を省略する場合は、VALUES句には、表に定義されている列と同数の値を、表の列構成と同じ順番で指定する
▶INSERT句に列名を指定する場合は、VALUES句には、列名のリストと同数の値を同じ順番で指定する

■NULL値を含む行の追加

➤暗黙的手法
列のリストから省略することによってNULL値を格納する

```
SQL> insert into dept_copy(deptno, dname) values(70, '経理');

1行が作成されました。
```

➤明示的手法
・列に対して「NULLキーワード」を指定する
・文字値や日付値を格納する列に対して、「''」(空の文字列)を指定する

```
SQL> insert into dept_copy values(80, '品質管理', NULL);

1行が作成されました。

SQL> insert into dept_copy values(90, '生産管理', '');

1行が作成されました。

SQL> select * from dept_copy;

    DEPTNO DNAME          LOC
---------- -------------- ----------
        10 管理           大手町
        20 研究開発       横浜
        30 営業           品川
        40 財務           東京
        50 教育           大手町
        60 システム       横浜
        70 経理
        80 品質管理
        90 生産管理

9行が選択されました。
```

➤「NOT NULL」が定義されている列に対するNULL値の指定
not nullが定義されている列に対するNULLのinsertはエラーになる。

```
SQL> desc departments;
 名前                                                                    NULL?    型
 ----------------------------------------------------------------------- -------- -----
----------------------------------------
 DEPTNO                                                                  NOT NULL NUMBE
R(2)
 DNAME                                                                            VARCH
AR2(14)
 LOC                                                                              VARCH
AR2(10)

SQL> insert into departments values(null, '生産管理', '横浜');
insert into departments values(null, '生産管理', '横浜')
                               *
行1でエラーが発生しました。:
ORA-01400: ("ORA01"."DEPARTMENTS"."DEPTNO")にはNULLは挿入できません。
```

■関数を使用したデータの追加

```
SQL> insert into emp_copy(empno, ename, hiredate) values(1015, '山口', sysdate);

1行が作成されました。
```

```
SQL> select empno, ename, to_char(hiredate, 'YYYY-MM-DD HH24:MI:SS') from emp_cop
y;

     EMPNO ENAME      TO_CHAR(HIREDATE,'Y
---------- ---------- -------------------
      1001 佐藤       2001-02-25 00:00:00
      1002 鈴木       2000-03-26 00:00:00
      1003 高橋       2000-05-30 00:00:00
      1004 田中       2002-06-02 00:00:00
      1005 渡辺       2002-07-11 00:00:00
      1006 伊藤       2008-01-06 00:00:00
      1007 山本       2000-08-09 00:00:00
      1008 中村       2000-09-17 00:00:00
      1009 小林       2006-10-21 00:00:00
      1010 斉藤       2001-12-17 00:00:00
      1011 加藤       2006-10-21 00:00:00
      1012 吉田       2009-03-13 00:00:00
      1013 山田       2001-03-13 00:00:00
      1014 佐々木     2004-05-02 00:00:00
      1015 山口       2016-03-16 23:24:53

15行が選択されました。
```

❐Column &置換変数を使用したデータの追加
INSERT文のVALUES句では、「&置換変数」を使用することも可能

```
SQL> set verify on
SQL> insert into emp_copy(empno, ename) values(&empno, '&ename');
empnoに値を入力してください: 1017
enameに値を入力してください: 井上
旧   1: insert into emp_copy(empno, ename) values(&empno, '&ename')
新   1: insert into emp_copy(empno, ename) values(1017, '井上')

1行が作成されました。

SQL> /
empnoに値を入力してください: 1018
enameに値を入力してください: 木村
旧   1: insert into emp_copy(empno, ename) values(&empno, '&ename')
新   1: insert into emp_copy(empno, ename) values(1018, '木村')

1行が作成されました。

SQL> select * from emp_copy where empno >= 1017;

     EMPNO ENAME      YOMI                 JOB              MGR HIREDATE         SA
L       COMM     DEPTNO
---------- ---------- -------------------- --------- ---------- --------- ---------
-          ---------- ----------
      1017 井上
      1018 木村
```

「/」（スラッシュ）コマンドを使用すれば再実行できる。

■副問合せを使用したデータの追加

【重要】
▶「INSERT INTO 表名 SELECT…」には、VALUES句は指定しない

```
SQL> insert into dept_copy select deptno + 1, dname, loc from departments;

4行が作成されました。

SQL> select * from dept_copy;

    DEPTNO DNAME          LOC
---------- -------------- ----------
        10 管理           大手町
        20 研究開発       横浜
        30 営業           品川
        40 財務           東京
        50 教育           大手町
        60 システム       横浜
        70 経理
        80 品質管理
        90 生産管理
        11 管理           大手町
        21 研究開発       横浜
        31 営業           品川
        41 財務           東京

13行が選択されました。
```

■UPDATE文によるデータの更新

```
SQL> select empno, ename, deptno from employees where deptno is null;

レコードが選択されませんでした。

SQL> select empno, ename, deptno from emp_copy where deptno is null;

     EMPNO ENAME          DEPTNO
---------- ---------- ----------
      1015 山口
      1017 井上
      1018 木村

SQL> update emp_copy set deptno = 10 where deptno is null;

3行が更新されました。
```

➤複数の列の更新
～略～

【重要】
UPDATE文でWHERE句を省略すると、表内のすべての行が更新される

■副問合せを使用したデータの更新

SET句で副問合せを使用すれば、副問合せで取り出したデータを使用して表を更新できます。

```
SQL> update emp_copy
  2  set sal = (select sal from emp_copy where empno = 1014)
  3  where empno = 1016;

1行が更新されました。
```

●複数の副問合せを使用したUPDATE文

```
SQL> select empno, ename, job, sal from employees where empno = 1010;

     EMPNO ENAME      JOB              SAL
---------- ---------- -------- ----------
      1010 斉藤       営業         150000
SQL> update emp_copy
  2  set job = (select job from employees where empno = 1010),
  3      sal = (select sal from employees where empno = 1010)
  4  where empno = (select empno from employees where ename = '加藤');

1行が更新されました。
SQL> select empno, ename, job, sal from emp_copy where ename = '加藤';

     EMPNO ENAME      JOB              SAL
---------- ---------- -------- ----------
      1011 加藤       営業         150000
```

複数の副問合せを含む蒸気のUPDATE文は、複数列副問合せを使用したUPDATE文に書き換えることが出来ます。
「**複数列副問合せ**」という名前は覚えておいてください！

```
SQL> update emp_copy set (job, sal) = (select job, sal from employees where empno
 = 1010)
  2  where empno = (select empno from employees where ename = '加藤');

1行が更新されました。
```

❒Tips
表名の代わりに副問合せを使用できる。
【重要】
▶1つのUPDATE文で複数の副問合せを使用できる
▶WHERE句、SET句および表名の代わりに副問合せを指定できる
▶複数列副問合せを使用して、同時に同一行の複数列を更新することもできる

■DELETE文によるデータの削除

```
SQL> select empno, ename, deptno from emp_copy where empno = 1018;

     EMPNO ENAME          DEPTNO
---------- ---------- ----------
      1018 林                 20
```

```
SQL> delete from emp_copy where empno = 1018;

1行が削除されました。

SQL> select empno, ename, deptno from emp_copy where empno = 1018;

レコードが選択されませんでした。

-- 複数行削除
SQL> delete from emp_copy where empno >= 1010;
```

6行が削除されました。

■副問合せを使用したデータの削除

副問合せで取り出したデータを元にして削除する行を特定出来ます。

```
SQL> select empno, ename from emp_copy where deptno = (select deptno from employe
es where empno = 1009);

     EMPNO ENAME
---------- ----------
      1002 鈴木
      1005 渡辺
      1009 小林

SQL>
SQL> delete from emp_copy where deptno = (select deptno from employees where empn
o = 1009);

3行が削除されました。

SQL> select empno, ename from emp_copy where deptno = (select deptno from employe
es where empno = 1009);

レコードが選択されませんでした。
```

10.2 トランザクションの制御

■トランザクションとは

1つまたは複数の操作によって完結する、一連のデータ操作をまとめた「論理的な処理単位」です

- すべての操作を確定する(コミット)
- すべての操作を取り消す(ロールバック)

➤トランザクションの構成要素
トランザクションは以下のいづれかの要素で構成されます。
トランザクションの構成要素

種類	構成要素
DML(データ操作言語)	トランザクションは論理的な最小作業単位として扱われる1つ

種類	構成要素
DDL(データ定義言語)	トランザクションは1つのDDL文で構成
DCL(データ制御言語)	トランザクションは1つのDCL文のみで構成

【重要】
▶トランザクションとは、論理的な処理単位である
▶DDL文とDCL文は1文で1トランザクションになる
▶DML文を含むトランザクションは論理的な最小作業単位として扱われる1つまたは複数のDML文で構成される

➤トランザクションの開始と終了

- COMMIT文またはROLLBACK文の実行
- DDL文の実行(自動コミット)
- DCL文の実行(自動コミット)
- ユーザーによるSQL DeveloperやSQL*PLUSの終了(自動コミット/自動ロールバック)
- マシン障害やシステムクラッシュの発生(自動ロールバック)

■明示的なトランザクション制御

トランザクション制御文	説明
COMMIT	一連の処理を確定し、終了する
SAVEPOINT セーブポイント名	セーブポイントを作成、トランザクションは継続
ROLLBACK	一連のをすべて取り消し、トランザクションを終了
ROLLBACK TO [SAVEPOINT] セーブポイント名	指定されたセーブポイントまでの処理を取り消す

➤ROLLBACK文による処理の取り消し

```
SQL> create table dept_copy2
  2  as
  3  select  * from departments;

表が作成されました。

SQL> select * from dept_copy2;

    DEPTNO DNAME          LOC
---------- -------------- ----------
        10 管理           大手町
        20 研究開発       横浜
        30 営業           品川
        40 財務           東京
```

```
SQL> insert into dept_copy2
  2  values(50, '教育', '大手町');
```

```
1行が作成されました。

SQL> insert into dept_copy2
  2  values(60,'システム', '横浜');

1行が作成されました。

SQL>
SQL> select * from dept_copy2
  2  where deptno >= 50;

    DEPTNO DNAME          LOC
---------- -------------- ---------
        50 教育           大手町
        60 システム       横浜

SQL>
SQL> rollback;

ロールバックが完了しました。

SQL> select * from dept_copy2
  2  where deptno >= 50;

レコードが選択されませんでした。
```

➤COMMIT文による処理の確定
確定した処理はROLLBACK文を実行しても取り消すことは出来ませんので覚えておく

```
SQL> insert into dept_copy2
  2  values(50, '教育', '大手町');

1行が作成されました。

SQL>
SQL>
SQL> insert into dept_copy2
  2  values(60,'システム', '横浜');

1行が作成されました。

SQL>
SQL>
SQL> commit;

コミットが完了しました。

SQL> rollback;

ロールバックが完了しました。

SQL> select * from dept_copy2
  2  where deptno >= 50;

    DEPTNO DNAME          LOC
```

```
---------- -------------- ----------
        50 教育           大手町
        60 システム       横浜
```

➤セーブポイントの作成と利用

「**ROLLBACK文で指定されたセーブポイントよりも後に作成したセーブポイントは破棄される**」点に注意

```
SQL> update dept_copy2
  2  set loc = 'POINT A'
  3  where deptno = 60;

1行が更新されました。

SQL> savepoint a;

セーブ・ポイントが作成されました。

SQL> update dept_copy2
  2  set loc = 'POINT B'
  3  where deptno = 60;

1行が更新されました。

SQL> savepoint b;

セーブ・ポイントが作成されました。

SQL> update dept_copy2
  2  set loc = 'POINT C'
  3  where deptno  60;
where deptno  60
              *
行3でエラーが発生しました。:
ORA-00920: 関係演算子が無効です。


SQL> update dept_copy2
  2  set loc = 'POINT C'
  3  where deptno = 60;

1行が更新されました。

SQL> rollback a;
rollback a
         *
行1でエラーが発生しました。:
ORA-02181: ROLLBACK WORKのオプションが無効です。


SQL> rollback to a;

ロールバックが完了しました。
```

```
SQL> select * from dept_copy2
  2  where deptno = 60;

    DEPTNO DNAME          LOC
---------- -------------- ----------
        60 システム       POINT A

SQL>
SQL> rollback to b;
rollback to b
*
行1でエラーが発生しました。:
ORA-01086: セーブポイント'B'はこのセッションで設定されていないか、無効です


SQL> rollback;

ロールバックが完了しました。

SQL> select * from dept_copy
  2
SQL> select * from dept_copy2
  2  where deptno = 60;

    DEPTNO DNAME          LOC
---------- -------------- ----------
        60 システム       横浜

SQL>
```

♪Tips
トランザクション内のDML文でエラーが発生した場合は、そのDML文だけが自動的に取り消されます(文レベルのロールバック)
トランザクション全体はロールバックされませんし、トランザクションも終了しません。

【重要】
～略～

■暗黙的なトランザクション処理

トランザクション制御文を実行していない場合でトランザクション処理が発生するケース

暗黙的なトランザクション処理

種類	トランザクション処理が実行されるタイミング
自動コミット	DDL文実行時、DML文実行時、SQL DeveloperまたはSQL*Plusの正常終了時(※１)
自動ロールバック	SQL DeveloperまたはSQL*Plusの以上終了時、システム障害発生時(Oracleのインスタンスの以上停止時)

※１SQL*Plusのウィンドウの「閉じる」ボタンを押すと、異常終了になります。SQL*Plusを正常終了する場合は「EXIT」コマンドを実行すること

❐Column　SQL*PlusのAUTOCOMMIT機能
SQL*PlusのAUTOCOMMIT機能を有効にすると、各DML文の実行時に
自動的にコミット処理が行われます。(AUTOCOMMIT機能はデフォルト値はOFFです)

```
SQL> update dept_copy2
  2  set loc = 'さいたま'
  3  where deptno = 60;

1行が更新されました。

コミットが完了しました。
```

■TRUNCATE文

●TRUNCATE文
TRUNCATE TABLE 表名;

- TRUNCATE文では削除するデータを指定できない
- TRUNCATE文はDDLなので実行時に自動コミットが実行される(DELETEはDML文)
- 自動コミットが実行されるので処理をロールバック出来ない
- TRUNCATE文ではロールバック用のデータを生成する必要がないため、DELTE文よりも短時間でデータを削除できる

```
SQL> select count(*) from emp_copy;

  COUNT(*)
----------
         6

SQL>
SQL> delete from emp_copy;

6行が削除されました。

コミットが完了しました。
SQL>
SQL> truncate table emp_copy;

表が切り捨てられました。
```

♪Tips
TRUNCATE文を実行した場合、操作対象の表に**DELETEトリガー**が定義されていても、
実行されないので注意しましょう。
例えばDELTEトリガーを使用すれば、製品情報の表からデータが削除された場合に、そのデータを
自動的に旧製品情報の表にコピーすることが出来ます。

10.3 同時実行制御

読み取り一貫性

読み取り一貫性機能の目的
「DML操作開始時点での最新のコミット済みデータが常に各セッションに戻されるように保証すること」

➤読み取り一貫性の仕組み
1.Aさんが読み取り操作(DML文)を開始する
2.Bさんがデータの変更操作を実行してコミット
3.データ変更操作を受け付けたOracleサーバーは変更前のデータを「UNDOセグメント」にコピーしてからデータを変更
4.AさんにはUNDOセグメント内のデータを戻す

➤未確定データの扱い

- 読み取り操作は書き込み操作と無関係に実行できる
- 書き込み操作は読み取り操作と無関係に実行できる

➤読み取り一貫性の確認

【重要】
▶未コミットのデータは他のセッションからは参照出来ない
▶未コミットのデータは変更を行ったセッションでだけ確認できる
▶コミット後はすべてのセッションで変更後のデータを参照できる
▶読み取り操作は書き込み操作によって待機させられない
▶同じユーザーでも、セッションが異なれば、他のユーザーと同じように変更途中の未コミットデータは参照出来ない

■ロック

Oracleサーバーでは書き込み処理に対してロックのメカニズムを使用している。
変更対象の行ごとに排他ロックをかけてデータを変更します。
変更対象の行の変更が完了するまで待機する動き

■FOR UPDATE句による排他ロック

FOR UPDATE句を使用すると、SELECT文の実行時に対象の行に行レベルの排他ロックをかけることが出来ます。

●FOR UPDATE句を使用するSELECT文の基本構文

```
SELECT 列名 [ ,列名…]
FROM   表名
[WHERE 条件]
FOR UPDATE [OF 表名.列名] [NOWAIT  |  WAIT  秒数]
[ORDER BY  列名  [, 列名...]]
```

```
SQL> select e.empno, ename, e.sal, d.dname, d.loc
  2  from employees e join departments d using(deptno)
  3  where e.empno = 1014
  4  for update of e.empno;

     EMPNO ENAME             SAL DNAME          LOC
---------- ---------- ---------- -------------- ----------
      1014 佐々木         230000 管理           大手町

SQL>
```

```
SQL>
SQL> commit;

コミットが完了しました。
```

```
SQL> select * from employees
  2  where empno = 1014
  3  for update nowait;
select * from employees
              *
行1でエラーが発生しました。:
ORA-00054: リソース・ビジー。NOWAITが指定されているか、タイムアウトしました


SQL> select * from employees
  2  where empno = 1014
  3  for update wait;
for update wait
              *
行3でエラーが発生しました。:
ORA-30005: WAIT間隔が指定されていないか無効です。


SQL> select * from employees
  2  where empno = 1014
  3  for update wait 5;
select * from employees
              *
行1でエラーが発生しました。:
ORA-30006: リソース・ビジー; WAITタイムアウトの期限に達しました。


SQL>
SQL> select * from departments
  2  where deptno = 10
  3  for update nowait;

    DEPTNO DNAME          LOC
---------- -------------- ----------
        10 管理           大手町

SQL> rollback;

ロールバックが完了しました。
```

【重要】

▶ロックはトランザクション終了時まで保持されます

▶FOR UPDATE句を指定すると、SELECT文でも読み取り操作の対象行がロックされます

要素	説明
NOWAIT	待機せずすぐにエラーを戻す
WAIT 秒数	指定した秒数だけ待機し、その間にロックが解除されない場合はエラーを戻す

要素	説明
OF 表名.列名	指定した列を含む表の行のみにロックを限定する

FOR UPDATE同士が重なると後で実行されたSQL文が待たされる！

第11章 DDLを使用した表の作成と管理

11.1 表の作成と削除

■スキーマとは

データベースオブジェクトを管理する為に使用される「論理的な概念」
Oracleサーバーは各ユーザーがユーザー名と同じ名前のスキーマを1つ所有しており、各ユーザーが作成したオブジェクトはそのユーザーが所有するスキーマに格納される
スキーマはあくまでも論理的な概念であるため、実際に領域が割り当てられるわけではないので注意

➤別のユーザーが所有するオブジェクトを参照する方法
「ora01.employees」のように指定する必要がある。
【重要】
▶ユーザーはユーザー名と同じ名前のスキーマを1つ所有する
▶他のユーザーが所有するスキーマ内のオブジェクトを参照するには「スキーマ名.オブジェクト名」を指定する必要がある。

❐Column　スキーマオブジェクト
スキーマオブジェクト：各ユーザーが所有するスキーマに含まれるオブジェクトのこと
代表的なもの

- 表
- ビュー
- 順序
- 索引
- シノニム

●ビュー
1つまたは複数の表やビューを元に作成された「論理的な表」
SELECT文に名前をつけてデータベースに保存したもの
このようにビューはあたかも表であるかのように使用できる為「仮想表」とも呼ばれます。

```
SQL> create view v_dept30
  2  as
  3  select e.empno, e.ename, e.job, d.loc
  4  from employees e join departments d
  5  on e.deptno = d.deptno
  6  where e.deptno = 30;

ビューが作成されました。

SQL> desc v_dept30
 名前                                                      NULL?    型
 --------------------------------------------------------- -------- -----
```

```
 EMPNO                                                     NOT NULL NUMBE
R(4)
 ENAME                                                              VARCH
AR2(10)
 JOB                                                                VARCH
AR2(8)
 LOC                                                                VARCH
AR2(10)

SQL> select * from v_dept30;

     EMPNO ENAME      JOB      LOC
---------- ---------- -------- ----------
      1003 高橋       営業     品川
      1004 田中       営業     品川
      1006 伊藤       営業     品川
      1007 山本       部長     品川
      1010 斉藤       営業     品川
      1012 吉田       事務     品川

6行が選択されました。

SQL>
```

複雑なSELECT文を簡略化出来ます。
また、ユーザーごとに見せたくない列があれば、ビューを作成して、表自体を隠し、
ビューを参照させることでセキュリティの機能を実装出来ます。

●順序
指定された規則にしたがって**一意な番号(順序値)を自動的に生成する**スキーマオブジェクトです。

```
SQL> select max(ordno) from orders;

MAX(ORDNO)
----------
        13

SQL> create sequence s_ord_ordno start with 14 increment by 1;

順序が作成されました。

SQL> insert into orders(ordno, custno) values(s_ord_ordno.nextval, 1000);

1行が作成されました。

SQL> select s_ord_ordno.currval from dual;

   CURRVAL
----------
        14

SQL> insert into orders(ordno, custno) values(s_ord_ordno.nextval, 1001);

1行が作成されました。
```

```
SQL> select s_ord_ordno.currval from dual;

   CURRVAL
----------
        15

SQL> select ordno, custno from orders
  2  where ordno >= 14;

     ORDNO     CUSTNO
---------- ----------
        14       1000
        15       1001

SQL> rollback;

ロールバックが完了しました。
```

オプション	説明
START WITH	最初に生成される順序の初期値
INCREMENT BY	順序の増分値(いくつずつ増やしていくか)

●索引
検索のパフォーマンスを向上させるために使用するスキーマオブジェクトです。
また、データの一意性(重複がないこと)をチェックする際にも使用されます。

➤索引の作成

```
SQL> create index ind_prod_pname on products(pname);

索引が作成されました。

SQL> create index od_prodno_quantity on ord_details(prodno, quantity);

索引が作成されました。
```

索引を使用するかしないかは、Oracleサーバーのオプティマイザーという機能が自動的に判断します

●シノニム
オブジェクトの別名を表すスキーマオブジェクトです。
名前が長いオブジェクトには短い名前のシノニムを設定しておくと便利

```
SQL> create synonym dept for departments;

シノニムが作成されました。

SQL> select * from dept;

    DEPTNO DNAME          LOC
---------- -------------- ----------
        10 管理           大手町
```

```
        20 研究開発        横浜
        30 営業            品川
        40 財務            東京
```

DUAL表はパブリックシノニムとして設定されている。

■オブジェクトの命名規則

- 長和は30バイト以下(文字数ではなくバイト数)
- 先頭の文字は数字以外の文字(数字及び記号は不可能)
- 使用できる文字は、英数字(A~Z, a~z, 0~9)および漢字、カタカナ、ひらがな(日本語環境の場合)
- 使用できる記号は「_」「$」「＃」の3種類のみ
- 同一のスキーマ内で重複する名前は定義出来ない
- Oracleサーバーの予約後(SELECT, FROM, ORDERなど)は使用できない
- アルファベットの大文字小文字は区別されない

【重要】
上記の中でも重要なものになる
▶オブジェクト名の先頭文字は数字以外の文字
▶オブジェクト名のアルファベットの大文字小文字は区別されない
▶オブジェクト名に使用できる記号は「_」「$」「＃」のみ

■表の作成

表を作成するためにはCREATE TABLE権限が必要です。
また自分が所有しているスキーマ以外に表を作成するにはCREATE ANY TABLE権限が必要です。

●CREATE TABLE構文

```
CREATE TABLE [スキーマ名].表名
(
    列名   データ型
    [,列名   データ型…]
)
```

スキーマ名を省略した場合は、実行ユーザーが所有するスキーマに作成される。

❐Column DDL(Data Definition Language: データ定義言語)
DDLを実行すると「データ・ディクショナリ」と呼ばれるOracleの管理情報を格納するために使用される特別なひょうにオブジェクトの定義情報が登録されたり、登録されている定義情報が変更・削除されたりします。

➤DEFAULTオプション
●DEFAULTオプションを使用した列の定義
列名 データがた[DEFAULT 式]

```
SQL> create table emp2
  2  (
  3     empno    number(4),
  4     ename    varchar2(10),
  5     hiredate date default sysdate
```

```
  6  );

表が作成されました。

SQL>
SQL> insert into emp2(empno, ename)
  2  values(1001,'佐藤');

1行が作成されました。
SQL>
SQL> select * from emp2;

     EMPNO ENAME      HIREDATE
---------- ---------- --------
      1001 佐藤       16-03-18
```

♪Tips
デフォルト値には、リテラル値、式またはSQL関数(SYSDATEやUSERなど)を指定できます。
また、Oracle12cからは、順序オブジェクトを参照するNEXTVAL擬似列や
CURRVAL擬似列も指定できるようになりました。一方別の列の名前は指定できません。
試験では消去法で回答すること！！

■表の削除

表を削除できるのは表の所有者またはDROP ANY TABLE権限を持つユーザーのみです。
●DROP TABLE文の基本構文
DROP TABLE 表名 [PURGE];

表に定義されている制約や索引も削除されます。
一方、表を参照するビューやシノニムは削除されません。(無効になります)
なおDROP TABLEを実行すると表はゴミ箱に移動されます。
完全に削除されるわけではない。
表を完全に削除するにはPURGE句を指定します。

```
SQL> drop table emp1 purge;

表が削除されました。
```

♪Tips
削除された表をゴミ箱から復元するにはFLASHBACK TABLE文を使用します。(試験範囲！！)

【重要】
▶表を削除すると、表内のすべてのデータと表に定義されている制約、索引も削除される
▶表を削除しても、表を参照するビューとシノニムは削除されない(無効になる)
▶表の所有者またはDROP ANY TABLE権限を持つユーザーだけが表を削除できる

11.2 データ型

■データ型とは

Oracleサーバーで扱うデータの形式
●データ型の分類と主なデータ型

分類	説明	主なデータ型
文字型	列に文字データを格納できるようにするために使用する	・CHAR型 ・VARCHAR2型 ・LONG型 ・CLOB型 ・NCLOB型
数値型	列に数値データを格納できるようにするために使用する	・NUMBER型
日付型	列に日付データを格納できるようにするために使用する	・DATE型
バイナリ型	列にバイナリデータを格納できるようにするために使用する	・RAW型 ・LONG RAW型 ・BLOB型 ・BFILE型
ROWID	列にROWIDを格納できるようにするために使用する	・ROWID型

➤文字型

データ型	説明
VARCHAR2型	最大4000バイトまでの文字データを格納できる。可変長データ型
CHAR型	最大2000バイトまでの文字データを格納できる、固定長データ型
LONG型	最大2GBまでの文字データを格納できる可変長データ型
CLOB型	最大4GBまでの文字データを格納できるデータ型
NCLOB型	最大4GBまでのUnicode文字データを格納できるデータ型

●VARCHAR2型とCHAR型の違い

	VARCHAR2型	CHAR型
最大サイズ	4000バイト	2000バイト
最大サイズ指定の省略	不可(省略するとエラー)	可(省略した場合デフォルト値は「1」)
確保するデータサイズ	可変長(格納するデータのサイズに応じて変わる)	固定長(格納するデータのサイズにかかわらず表作成時に定義したサイズで一定)

●LONG型の制限
LONG型は、VARCHAR2型やCHAR型の列には格納出来ない、大きな文字データを格納できるデータ型(最大2GB)ですが、
制限があるので注意。。

- 副問合せを使用した表の作成時に、LONG列はコピー出来ない(エラーが発生する)
- LONG列はGROUP BY句とORDER BY句に指定できない
- LONG列は、1つの表に1つだけ定義できる(LONG列またはLONG RAW列のどちらか1つだけ)
- LONG列には制約を定義出来ない

➤数値型
●NUMBER型の定義方法

列名 NUMBER[最大精度 [,位取り]]

オプション	説明
最大精度	格納する数値データの最大精度を指定する。最大38桁
位取り	格納する数値データの小数点以下の桁数を指定する

最大精度は少数点以下の数値も含めた桁数
位取りを省略した場合、最大精度に指定した桁数の整数値を格納出来ます。

➤日付型
●DATE型の定義方法
列名 DATE

固定長で7バイト
世紀、年、月、日、時、分、秒が内部的な数値書式の形式で格納されます。

➤バイナリデータ型
●バイナリデータを格納する主なデータ型

データ型	説明
RAW型	最大2000バイトまでのバイナリデータが格納できる、可変長データ型。最大サイズの指定は省略できない
LONG RAW型	最大2GBまでのバイナリデータを格納できる、可変長データ型。LONG型と同様の制限がある。
BLOB型	最大4GBまでのバイナリデータを格納できるデータ型
BFILE型	最大4GBまでのバイナリデータを格納できる、読み取り専用のデータ型。BFILE型が定義された列に格納したデータはOracleサーバーのデータファイル上ではなく、OS上のファイル(イメージファイルや動画ファイルなど)に保存される

それぞれのデータ型の定義方法は以下のとおり
LONG RAW型
BLOB型
BFILE型　には最大サイズを指定できないので注意してください

➤バイナリデータ型の定義方法
列名 RAW(最大サイズ)
列名 LONG RAW
列名 BLOB
列名 BFILE

♪Tips　**LOB型について**
CLOB, BLOB, BFILEはLOB型(Large OBject型)と呼ばれます。
その名の通り大きなオブジェクトを格納出来ます。LOB型は表とは異なる場所に実際のデータを格納し、表の中には実際にデータを格納した場所へのポインタ情報のみを格納して使うことが出来ます。
CLOBの「C」はCharacter(文字)のC、

BLOBとBFILEの「B」はBinary(バイナリ)のBです。
CLOBとBLOBはOracleサーバーが管理するデータファイルにデータとポインタ情報を格納しますが、
BFILEはOS上のイメージファイルや動画ファイルなどのポインタ情報のみを格納します。
LONG列やLONG RAW列と異なり、1つの表に複数のLOB型の列を定義出来ます。

➤ROWID型
●ROWID型の定義方法
列名 ROWID

表の各行に割り当てられている一意なアドレス(BASE64文字列)です。

```
SQL> select rowid, deptno, dname from departments;

ROWID                  DEPTNO DNAME
------------------ ---------- --------------
AAAXHAAAGAAAAEXAAA         10 管理
AAAXHAAAGAAAAEXAAB         20 研究開発
AAAXHAAAGAAAAEXAAC         30 営業
AAAXHAAAGAAAAEXAAD         40 財務
```

【重要】
▶VARCHAR2型の最大のサイズは省略出来ない
▶CHAR型の最大サイズの指定を省略するとデフォルト値の「1」が使用される
▶LONG列は副問合せを使用した表の作成時にコピー出来ない(エラーが発生する)
▶LONG列はGROUP BY句とORDER BY句に指定できない
▶LONG列は、1つの表に1つだけ定義できる
▶LONG列には、制約を定義出来ない
▶1つの表に複数のLOB型の列を定義できる

■その他のデータ型

上記に加えて、Oracle Database 9i以降からは以下の
「日付時刻及び期間を格納するデータ型」がサポートされています。

- TIMESTAMP型
- INTERVAL YEAR TO MONTH型
- INTERVAL DAY TO SECOND型

➤TIMESTAMP型
DATE型を拡張したデータ型です。
DATE型の列が格納できる年、月、日、時、分、秒に加えて、秒の小数点以下の値も格納出来ます。
小数点以下の桁数に０～９の範囲を指定できます。(省略した場合のデフォルト値は「６」です)
●TIMESTAMP型の定義方法
列名　TIMESTAMP[(小数点以下の桁数)]

```
SQL> create table time1
  2  (
  3     date1   date,
  4     timestamp1 timestamp
  5  );

表が作成されました。

SQL> insert into time1 values(sysdate, systimestamp);
```

```
1行が作成されました。

SQL> column timestamp1 format a30
SQL> select * from time1;

DATE1     TIMESTAMP1
--------- ------------------------------
16-03-18 16-03-18 20:17:54.715000
```

●TIMESTAMP型のバリエーション

バリエーション	説明
TIMESTAMP WITH TIME ZONE	タイムゾーンの時差を含むことができる。タイムゾーンの時差は列の一部として格納され、表示される
TIMESTAMP WITH LOCAL TIMEZONE	タイムゾーンの時差を含むことができる。タイムゾーンの時差は列の一部として格納されず、データ取得時にローカルセッションのタイムゾーンの値で表示される

※タイムゾーンの時差とは、ローカル時間とUTC(協定世界時)の差(時間と分)です。日本のタイムゾーンの時差は「+9:00」

```
SQL> create table time2
  2  ( timeA timestamp with time zone,
  3    timeB timestamp with local time zone
  4  );

表が作成されました。

SQL>
SQL> insert into time2
  2  values(systimestamp, systimestamp);

1行が作成されました。

SQL> select * from time2;

TIMEA
---------------------------------------------------------------------------
TIMEB
---------------------------------------------------------------------------
16-03-18 20:29:23.381000 +09:00
16-03-18 20:29:23.381000


SQL> column timeA format a35
SQL> column timeB format a35
SQL> select * from time2;

TIMEA                               TIMEB
----------------------------------- -----------------------------------
16-03-18 20:29:23.381000 +09:00     16-03-18 20:29:23.381000
```

```
SQL> alter session set time_zone='US/Hawaii';

セッションが変更されました。

SQL> select * from time2;

TIMEA                                TIMEB
---------------------------------- ----------------------------------
16-03-18 20:29:23.381000 +09:00      16-03-18 01:29:23.381000

-- セッションのタイムゾーンを元に戻す
SQL> alter session set time_zone='Asia/Tokyo';

セッションが変更されました。
```

➤INTERVAL YEAR TO MONTH型とINTERVAL DAY TO SECOND型
それぞれ指定した２つの日付、時刻の間隔を格納できるデータ型

データ型	説明
INTERVAL YEAR TO MONTH	２つの日付。時刻の間隔を**年**および**月の単位**で格納する
INTERVAL DAY TO SECOND	２つの日付。時刻の間隔を**日、時、分、秒の単位**で格納する

```
SQL> create table time3
  2  (
  3     timeA   interval year to month,
  4     timeB   interval day to second
  5  );

表が作成されました。

SQL> insert into time3
  2  values(interval '1-2' year to month, interval '10 12:30' day to minute);

1行が作成されました。

SQL> select to_char(sysdate, 'YYYY-MM-DD HH24:MI'),
  2         to_char(sysdate + timeA, 'YYYY-MM-DD'),
  3         to_char(sysdate + timeB, 'YYYY-MM-DD HH24:MI')
  4  from time3;

TO_CHAR(SYSDATE, TO_CHAR(SY TO_CHAR(SYSDATE+
---------------- ---------- ----------------
2016-03-18 20:44 2017-05-18 2016-03-29 09:14
```

【重要】
▶TIMESTAMP型は、秒の小数点以下の値も格納できる
▶INTERVAL YEAR TO MONTH型とINTERVAL DAY TO SECOND型は２つの日付・時刻の間隔を格納できる

11.3 制約の種類と指定方法

■制約とは

➤制約の定義方法
列レベルまたは表レベルで定義可能。
2つの違いは主に構文による違いだけで、作成された制約の機能は同じ。

- 「CONSTRAINT制約名」は省略可能(省略時はOracleサーバーが「SYS_Cn」の形式で名前を作成する)
- 表レベル制約の構文では「()」(丸括弧)内に制約を定義する列を1つ以上指定する
- NOT NULL制約は列レベルでのみ定義できる
- 複数の列の組み合わせからなる複合の制約は表レベルでのみ定義できる
- 1つの列に複数の列レベル制約を定義する場合は、改行またはスペースで区切る(カンマではない。カンマだと次のカラムに行ってまう)
- 1つの表に複数の表レベル制約を定義する場合は、「,」(カンマ)で区切る(改行やスペースではない(次にカラムが来る可能性はないからカンマで区切っても大丈夫))

既存の表にデータが格納されている場合でも、
「格納されているデータが、追加する制約のルールに従っている場合は、制約を追加できる」ということは覚えておきましょう

❐Column
上記の通り、「CONSTRAINT制約名」は省略可能ですが、省略すると制約名が「SYS_Cn」(nは制約の名前を一意に識別する整数：SYS_C0000123など)になります。しかし制約名は制約の管理(削除や無効化)を行う際に指定する必要があり、また制約に反する変更処理を行った時に発生するエラーの「エラーメッセージ」の中に表示されることもあるので、明示的にわかりやすい名前を設定することをオススメする。

```
SQL> create table emp1
  2  (
  3     empno number(4) constraint emp1_empno_pk primary key,
  4     ename varchar2(20)
  5  );

表が作成されました。

SQL> create table emp2
  2  (
  3     empno number(4),
  4     ename varchar(20),
  5     constraint emp2_empno_pk primary key(empno)
  6  );

表が作成されました。
```

■NOT NULL制約

定義した列にはNULL値を設定できなくなる
また列レベルでのみ定義出来ます。

```
SQL> create table emp3
  2  (
  3     empno    number(4)       constraint       emp3_empno_nn not null,
  4     ename    varchar2(20)
```

```
  5  );

表が作成されました。

SQL> desc emp3;
 名前                                                              NULL?    型
 ----------------------------------------------------------------- -------- -----
---------------------------------------
 EMPNO                                                             NOT NULL NUMBE
R(4)
 ENAME                                                                      VARCH
AR2(20)

SQL> insert into emp3 values(null, '佐藤');
insert into emp3 values(null, '佐藤')
                        *
行1でエラーが発生しました。:
ORA-01400: ("ORA01"."EMP3"."EMPNO")にはNULLは挿入できません。
```

■UNIQUE制約(一意キー制約)

重複した値を格納できなくなる。
ただしNULL値を含めることは可能で、複数行にNULL値を含めることも出来ます。
UNIQUE制約を定義すると、自動的に制約と同じ名前の一意索引(重複値が許可されない索引)が作成されます。
(この索引はOracleサーバーがデータの一意制を効率的に保証するために使用します。)

```
SQL> create table emp4
  2  (
  3      empno    number(4)       constraint      emp4_empno_uk unique,
  4      ename    varchar2(20)
  5  );

表が作成されました。

SQL> desc emp4;
 名前                                                              NULL?    型
 ----------------------------------------------------------------- -------- -----
---------------------------------------
 EMPNO                                                                      NUMBE
R(4)
 ENAME                                                                      VARCH
AR2(20)

SQL> insert into emp4 values(1001,'佐藤');

1行が作成されました。

SQL> insert into emp4 values(1001,'鈴木');
insert into emp4 values(1001,'鈴木')
*
行1でエラーが発生しました。:
ORA-00001: 一意制約(ORA01.EMP4_EMPNO_UK)に反しています
```

```
SQL> create table emp5
  2  (
  3     empno    number(4)       constraint      emp5_empno_nn not null,
  4     ename    varchar2(20),
  5     constraint emp5_empno_uk unique(empno)
  6  );

表が作成されました。

SQL> desc table emp5
使用方法: DESCRIBE [スキーマ.]オブジェクト[@db_link]
SQL> desc emp5
 名前                                                                    NULL?    型
 ----------------------------------------------------------------------- -------- -----
------------------------------------
 EMPNO                                                                   NOT NULL NUMBE
R(4)
 ENAME                                                                            VARCH
AR2(20)

SQL> insert into emp4 values(1001,'佐藤');
insert into emp4 values(1001,'佐藤')
*
行1でエラーが発生しました。:
ORA-00001: 一意制約(ORA01.EMP4_EMPNO_UK)に反しています


SQL> insert into emp5 values(1001,'佐藤');

1行が作成されました。

SQL> insert into emp5 values(1001,'鈴木');
insert into emp5 values(1001,'鈴木')
*
行1でエラーが発生しました。:
ORA-00001: 一意制約(ORA01.EMP5_EMPNO_UK)に反しています
```

複合のユニーク制約も可能
→表レベルで制約を定義する
～略～

■PRIMARY KEY制約(主キー制約)

その列(または列の組み合わせ)には表内の各行を一意に識別できる値しか格納できなくなる。
重複値やNULL値は格納出来ない
組み合わせの場合は表レベル定義じゃないとだめ

【重要】
▶PRIMARY KEY制約を定義すると、その列(または列の組み合わせ)には、重複した値やNULL値を格納できない
▶PRIMARY KEY制約を定義すると自動的に制約と同じ名前の一意索引が作成される
▶列の組み合わせに対してPRIMARY KEY制約を定義する場合は表レベル制約の構文を使用する必要がある。

■FOREIGN KEY制約(外部キー制約)

その列には参照先の列に存在する値しか格納できなくなる
NULL値を含めることは可能

●FOREIGN KEY制約の列レベル制約の構文
[CONSTRAINT 制約名] REFERENCES　親表名(参照する列名 [,参照する列名…])

●FOREIGN KEY制約の表レベル制約の構文
[CONSTRAINT 制約名] FOREIGN KEY(列名 [,列名…]) REFERENCES 親表名(参照する列名[, 参照する列名])

指定値	説明
親表名	参照先の表の名前。FOREIGN KEY制約で参照する表を「親表」、FOREIGN KEY制約が定義されている表を「子表」と呼ぶ。なお、親表には別の表だけでなく、FOREIGN KEY制約を定義する表(つまり同じ表)を指定することもできる。
参照する列名	親表に定義されている、参照先の名前。ただし、FOREIGN KEY制約は**UNIQUE制約**または**PRIMARY KEY制約**が定義されている列しか参照できない

♪Tips
FOREIGN KEY制約は、他の列の値を参照して整合性をチェックすることから「参照整合性制約」と呼ばれることもあります。

```
SQL> create table emp7
  2  (
  3     empno    number(4)       constraint emp7_empno_pk primary key,
  4     ename    varchar2(10)    constraint emp7_ename_nn not null,
  5     deptno   number(4)       constraint emp7_dept1_deptno_fk references dept1
(deptno)
  6  );

表が作成されました。

SQL> insert into emp7 values(1001, '佐藤', 10);

1行が作成されました。

SQL> insert into emp7 values(1002, '鈴木',30);

1行が作成されました。

SQL> insert into emp7 values(1002, '田中',60);
insert into emp7 values(1002, '田中',60)
*
行1でエラーが発生しました。:
ORA-00001: 一意制約(ORA01.EMP7_EMPNO_PK)に反しています

SQL> insert into emp7 values(1003, '田中',60);
```

```
insert into emp7 values(1003, '田中' ,60)
*
行1でエラーが発生しました。:
ORA-02291: 整合性制約(ORA01.EMP7_DEPT1_DEPTNO_FK)に違反しました - 親キーがありま
せん
```

子表に親表を参照している行が存在する場合は、親表の行を削除出来ない
FOREGIN KEY制約を定義すると依存する行がなくても親表は削除出来ない。
子表を先に削除してから親表を削除するようにする。

```
SQL> delete from dept1
  2  where deptno = 10;
delete from dept1
*
行1でエラーが発生しました。:
ORA-02292: 整合性制約(ORA01.EMP7_DEPT1_DEPTNO_FK)に違反しました - 子レコードがあ
ります


SQL> drop table dept1 purge;
drop table dept1 purge
           *
行1でエラーが発生しました。:
ORA-02449: 表の一意キーまたは主キーが外部キーに参照されています。
```

➤ON DELETE CASCADE と ON DELETE SET NULL
親表を削除した場合のデフォルトの動作を変更出来ます。

キーワード	説明
指定なし	子表に親表を参照する行がある場合、親表の参照されている行は削除出来ない
ON DELETE CASCADE	子表に親表を参照する行がある場合、親表の参照されている行を削除すると、子表の参照している行も削除される。列レベル構文は使用できない
ON DELETE SET NULL	子表に親表を参照する行がある場合、親表の参照されている行を削除すると、子表の参照している行にNULL値が設定される。列レベル構文は使用できない

```
SQL> alter table dept2 modify(
  2     deptno number(4) constraint dept2_pk primary key
  3  );

表が変更されました。

SQL> create table emp9
  2  (
  3     empno    number(4)       constraint emp9_empno_pk primary key,
  4     ename    varchar2(10)    constraint emp9_ename_nn not null,
  5     deptno   number(4)       constraint emp9_dept2_deptno_fk
  6                                        references dept2(deptno)
  7                                        on delete cascade
```

```
  8  );

表が作成されました。

SQL>
SQL>
SQL>
SQL> insert into emp9 values(1001, '佐藤', 10);

1行が作成されました。

SQL>
SQL> select * from emp9;

     EMPNO ENAME          DEPTNO
---------- ---------- ----------
      1001 佐藤               10

SQL>
SQL> delete from dept2
  2  where deptno = 10;

1行が削除されました。

SQL> delete from dept2
  2
SQL> select * from emp9;

レコードが選択されませんでした。
```

~ON DELETE SET NULLは略~

【重要】
▶FOREIGN KEY制約(外部キー制約)を定義すると、その列には参照先の列に存在する値しか格納出来ない
▶FOREIGN KEY制約(外部キー制約)を定義しても、NULL値を含めることができる
▶FOREIGN KEY制約はUNIQUE制約またはPRIMARY KEY制約が定義されている列しか参照出来ない
▶FOREIGN KEY制約を定義すると、親表は削除出来ない

■CHECK制約

その列には指定した条件に対して、TRUEまたはNULLを戻す値しか格納できなくなります。
●CHECK制約の列レベル制約及び表レベル制約の構文

```
[CONSTRAINT 制約名] CHECK(条件)
```

CHECK制約の条件にはWHERE句に指定できる条件と同様の指定が行えますが、
以下の指定は許可されていないので、注意してください。

- CURRVAL，NEXTVAL，LEVEL，ROWNUM擬似列の参照
- SYSDATE，UID，USER，USERENV関数の呼び出し
- 他の行の値を参照する問合せ

【重要】
▶CHECK制約を定義すると、その列には指定した条件に対してTRUEまたはNULLを戻す値歯科格納出来ない
▶CHECK制約は1つの列に複数定義できる
▶CHECK制約の定義には次の式は使用できない

- CURRVAL，NEXTVAL，LEVEL，ROWNUM擬似列の参照
- SYSDATE，UID，USER，USERENV関数の呼び出し
- 他の行の値を参照する問合せ
  ▶CHECK制約の条件には一部の指定を除き、基本的にはWHERE句に指定できる条件と同様の指定が行える

11.4 副問合せを使用した表の作成と表構造の変更

■副問合せを使用した表の作成

```
SQL> create table dept_copy
  2  as
  3  select * from departments;

表が作成されました。

SQL> desc departments;
 名前                                                                    NULL?    型
 ----------------------------------------------------------------------- -------- -----
----------------------------------------
 DEPTNO                                                                  NOT NULL NUMBE
R(2)
 DNAME                                                                            VARCH
AR2(14)
 LOC                                                                              VARCH
AR2(10)

SQL> desc dept_copy;
 名前                                                                    NULL?    型
 ----------------------------------------------------------------------- -------- -----
----------------------------------------
 DEPTNO                                                                           NUMBE
R(2)
 DNAME                                                                            VARCH
AR2(14)
 LOC                                                                              VARCH
AR2(10)
```

- 副問合せのSELECT句で計算式や関数を使用している場合は、CREATE TABLE文で別名を指定するか、副問合せのSELECT句で列別名を指定して新しい列名を指定する必要がある
- 列名を指定した場合、副問合せのSELECT句のリストと同じ個数の列名を指定する必要がある
- デフォルト値や制約を指定する事もできる。ただしデータ型は指定できない。データ型は元の列のデータ型が自動的に定義されます
- NOT NULL制約以外の制約はコピーされない。PRIMARY KEY制約の暗黙的なNOT NULL制約もコピーされない

```
SQL> create table emp30
  2  as
  3  select empno, ename, sal, sal*12
  4  from employees
  5  where deptno = 30;
select empno, ename, sal, sal*12
                          *
行3でエラーが発生しました。:
ORA-00998: 式に列の別名を指定する必要があります。


SQL> create table emp30(empno, ename, sal, annsal)
  2  as
  3  select empno, ename, sal, sal*12
  4  from employees
  5  where deptno = 30;

表が作成されました。

SQL> select * from emp30;

     EMPNO ENAME              SAL      ANNSAL
---------- ---------- ---------- ----------
      1003 高橋           300000     3600000
      1004 田中           355000     4260000
      1006 伊藤           300000     3600000
      1007 山本           285000     3420000
      1010 斉藤           150000     1800000
      1012 吉田           295000     3540000

6行が選択されました。
```

❐Column　副問合せで列別名を指定した表の作成

```
SQL> create table emp30_2
  2  as
  3  select empno, ename, sal, sal*12
  4  as annsal from employees
  5  where deptno = 30;

表が作成されました。

SQL>
SQL> select * from emp30_2;

     EMPNO ENAME              SAL      ANNSAL
---------- ---------- ---------- ----------
      1003 高橋           300000     3600000
      1004 田中           355000     4260000
      1006 伊藤           300000     3600000
      1007 山本           285000     3420000
      1010 斉藤           150000     1800000
      1012 吉田           295000     3540000

6行が選択されました。
```

```
SQL> -- 表構造のコピー
SQL> create table emp_copy2
  2  as
  3  select * from employees
  4  where 1 = 2;

表が作成されました。

SQL> desc emp_copy2;
 名前                                                                                  NULL?    型
 ------------------------------------------------------------------------------------ -------- -----
------------------------------------
 EMPNO                                                                                          NUMBE
R(4)
 ENAME                                                                                          VARCH
AR2(10)
 YOMI                                                                                           VARCH
AR2(20)
 JOB                                                                                            VARCH
AR2(8)
 MGR                                                                                            NUMBE
R(4)
 HIREDATE                                                                                       DATE
 SAL                                                                                            NUMBE
R(7)
 COMM                                                                                           NUMBE
R(7)
 DEPTNO                                                                                         NUMBE
R(2)

SQL> select * from emp_copy2;

レコードが選択されませんでした。
```

WEHRE 1 = 2のように絶対にTRUEにならない条件を指定することで、
表構造だけをコピーすることができる

【重要】
▶データ型とNOT NULL制約は、新しい表にコピーされる
▶NOT NULL制約以外の制約はコピーされない
▶副問合せのSELECT句で駅算式や関数を使用している場合は、明示的に列名(列別名)を指定する必要がある。

■表構造の変更

- 表に新しい列を追加する
- 既存の列のデータ型を変更する
- 既存の列にデフォルト値を設定する
- 既存の列を削除する
- 既存の列の名前を変更する
- 表を読み取り/書き込みモードにする
- 表を読み取り専用モードにする

➤列の追加

```
SQL> create table emp14
  2  (empno number(4),
  3   ename varchar2(10));

表が作成されました。

SQL> desc emp14
 名前                                                                                                  NULL?    型
 ----------------------------------------------------------------------------------------------------- -------- ------
----------------------------------------------

 EMPNO                                                                                                 NUMBE
R(4)
 ENAME                                                                                                 VARCH
AR2(10)

SQL> insert into emp14 values(1, '佐藤');

1行が作成されました。

SQL> insert into emp14 values(2, '鈴木');

1行が作成されました。

SQL> alter table emp14
  2  add (deptno number(3));

表が変更されました。

SQL> desc emp14
 名前                                                                                                  NULL?    型
 ----------------------------------------------------------------------------------------------------- -------- ------
----------------------------------------

 EMPNO                                                                                                 NUMBE
R(4)
 ENAME                                                                                                 VARCH
AR2(10)
 DEPTNO                                                                                                NUMBE
R(3)

SQL> select * from emp14;

     EMPNO ENAME          DEPTNO
---------- ---------- ----------
         1 佐藤
         2 鈴木
```

NOT NULL制約を定義する際の注意事項
DEFAULTオプションを使用すると、行があってもNOT NULL制約が定義で切る

```
SQL> alter table emp14
  2  add (sal number(8) not null);
alter table emp14
            *
行1でエラーが発生しました。:
```

```
ORA-01758: 必須列(NOT NULL)を追加するには、表を空にする必要があります。


SQL> alter table emp14
  2  add (sal number(4) default 200000 not null);
add (sal number(4) default 200000 not null)
                           *
行2でエラーが発生しました。:
ORA-01438: この列に許容される指定精度より大きな値です


SQL> alter table emp14
  2  add (sal number(8) default 200000 not null);

表が変更されました。

SQL> desc emp14
 名前                                                                    NULL?    型
 ----------------------------------------------------------------------- -------- ------
----------------------------------------
 EMPNO                                                                            NUMBE
R(4)
 ENAME                                                                            VARCH
AR2(10)
 DEPTNO                                                                           NUMBE
R(3)
 SAL                                                                     NOT NULL NUMBE
R(8)

SQL> select * from emp14;

     EMPNO ENAME          DEPTNO        SAL
---------- ---------- ---------- ----------
         1 佐藤                      200000
         2 鈴木                      200000
```

【重要】
▶既存の表に列を追加できる(ALTER TABLE文でADDキーワードを使う)
▶既存のデータがある表に列を追加すると、既存の行の新しい列の値はDEFAULTオプションで指定した値か、NULLになる
▶既存のデータが制約のルールを満たしていれば、既存の表に制約を追加できる
▶新しく追加する列にNOT NULL制約を追加できるのは、表が空の場合か、DEFAULTオプションが指定されている場合

➤列の変更
既存列のデータ型、サイズ、デフォルト値を変更出来ます。

- サイズまたは精度の増加はいつでもできる
- 列のサイズは次の場合のみ減少できる
    - 列にNULL値のみが含まれている
    - 表に行がない
    - 列のサイズは、既存の列の値未満には減少できない

- 列にNULL値のみが含まれている場合は、データ型を変更できる

- 列にNULL値以外のデータが含まれていても、サイズを変更しない場合は、CHAR型からVARCHAR2型、またはVARCHAR2型からCHAR型へはデータ型を変更できる
- 列のデフォルト値の変更は、以後の表への挿入のみに適用される

●既存の列を変更する構文

```
ALTER TABLE 表名
MODIFY (列名 [データ型] [DEFAULT 式]
              [, 列名 [データ型] [DEFAULT 式]…])
```

➤列の削除
●

```
ALTER TABLE 表名 DROP (列名 [, 列名…]);
```

❒Column　UNUSEDマーク
列にUNUSEDマークを設定します。UNUSEDマークが設定されている列は削除された列と同等の扱いになるため、同じ名前の列を作成出来ます。そのため実際のデータ削除処理を、利用ユーザーの少ない夜間などに回すことが出来ます。
UNUSEDマークが設定されると、DESCコマンドでも列名やデータ型を確認できなくなるので注意してください。
●UNUSEDマークを設定する構文

```
ALTER TABLE 表名
SET UNUSED(列名 [,列名…])

ALTER TABLE 表名
SET UNUSED COLUMN(列名);
```

●UNUSEDマーク付きの列を削除する構文

```
ALTER TABLE 表名 DROP UNUSED COLUMNS;
```

【重要】
▶列は削除できる
▶列を削除した後も、表には1つ以上の列を残す必要がある
▶列の削除は元に戻せない
▶別の列から参照される主キーは、CASCADEオプションを指定しないかぎり削除出来ない
▶1度に1つの列のみを削除する場合は、ALTER TABLE DROP COLUMN文も使用できる
▶UNUSEDマークを設定すると、削除された列と同等の扱いになり、マークをつけた後は同じ名前の列を作成できる
▶UNUSEDマークが設定されている列の名前の確認や取り消しは出来ない
▶UNUSEDマークが設定されている列は、DROP UNUSED COLUMNSを指定したALTER TABLE文で削除する

➤表モードの変更
●表のモードを変更する構文

```
ALTER TABLE 表名 {READ WRITE | READ ONLY};
```

モード	説明
READ WRITE	読み取り書き込みモード。
READ ONLY	読み取り専用モード。表のデータを変更することは出来ない。FOR UPDATE句を使用したSELECT文も実行出来ない。ただし表自体を削除することはできる