

SQL基礎/応用



2012/02/22 Ver.1.0

はじめに

□ 本研修の目的

- 本研修は、講義と演習を通して、「SQLによるプログラミングができること」を目的とする。

□ 到達目標

- SELECT句を利用した様々な問合せができる。
- テーブルの定義を行うことができる。
- テーブルの追加、削除、修正を行うことができる。

1. RDBMS



- 概要

RDBMSの概要とテーブルを構成する要素、データ型について学習する。

- 学習内容

- RDBMS
- テーブルの構造
- データ型

1.1 RDBMSとは

□ RDBMSとは

- Relational DataBase Management Systemの略
- データベース管理システムのひとつ
- データベースを運用・管理するためのソフトウェア
- データを操作する手段や操作の整合性管理、バックアップや障害対策など、さまざまな機能を有している。

1.2 テーブルの構造

□ テーブル

- データを関連する項目でまとめたデータのかたまりを「表」または「テーブル」と呼ぶ。

社員番号	社員氏名
0001	山田
0002	田中
0003	山本
0004	鈴木
0005	石井
0006	佐藤

社員テーブル

部署番号	部署名
100	総務部
200	開発部
300	営業部
400	人事部
500	経理部

部署テーブル

商品番号	商品名
001	リンゴ
002	ミカン
003	イチゴ
004	パイナップル
005	キウイ
006	バナナ

商品テーブル

1.2 テーブルの構造

□ レコードとカラム

- テーブル中の行を「行」「レコード」「row」と呼ぶ。
- テーブル中の列を「列」「カラム」「column」と呼ぶ。

注文テーブル		カラム			
レコード	注文番号	商品番号	数量	金額	納期
	1	1001	30	3600	2012/4/25
	2	1003	20	2400	2012/4/19
	3	1001	45	5400	2012/5/8

1.2 テーブルの構造

□ NULL

- 値が何も格納されていない状態。

注文テーブル

注文番号	商品番号	数量	金額	納期
1	1001	30	3600	2012/4/25
2	1003	20	1800	2012/4/19
3	1001	45	5400	

NULL

1.2 テーブルの構造

□ リレーション

- テーブルとテーブルの間に関連を持たせることができる。

注文テーブル

注文番号	商品番号	数量	金額	納期
1	1001	30	3600	2012/4/25
2	1003	20	2400	2012/4/19
3	1001	45	5400	

商品番号を使用して
テーブル間に関連を
持たせている。

商品テーブル

商品番号	商品名	単価
1001	ミカン	120
1002	リンゴ	90
1003	バナナ	120

1.2 テーブルの構造

□ プライマリキー (PRIMARY KEY)

- テーブル内のレコードを一意(*)に識別するためのカラムのこと。
- 「主キー」ともいう。
- プライマリキーの値は同一テーブル内で重複してはいけない。
- 値がNULLであってははいけない。
- 複数のカラムを組み合わせでプライマリキーにすることもできる。

*値が重複していないこと

1.2 テーブルの構造

□ プライマリキーの例(1)

- 商品テーブル内のレコードは、プライマリキーである「商品番号」によって一意に識別される。
- プライマリキー以外のカラムは値が重複していても良い。
(例えば単価が同じレコードが存在する、など)

商品 テーブル

商品番号	商品名	単価
1001	ミカン	120
1002	リンゴ	90
1003	バナナ	120

プライマリキー

1.2 テーブルの構造

□ プライマリキーの例(2)

- 以下の納品テーブルの場合、商品番号と顧客番号を組み合わせでプライマリキーにしている(複合プライマリキー)。

納品 テーブル

商品番号	顧客番号	納品数量
0001	01	100
0001	02	30
0002	03	200
0003	01	150
0003	02	500

複合プライマリキー

1.3 データ型

□ PostgreSQL で使用可能な主なデータ型

分類	データ型	説明	例
文字型	CHAR(n)	固定長文字データ nは、1～2000バイトまで指定可能	CHAR(10)
	VARCHAR(n)	可変長文字データ nは、1～4000バイトまで指定可能	VARCHAR(10)
数値型	NUMERIC(p,s)	可変長数値データ Pは最大精度38桁まで使用可能 sは小数点以下の桁数を指定可能	NUMERIC(3,4)
日付型	DATE	4桁年、月、日、時、分、秒 固定長7バイト (YYYYMMDDhhmmss)	
	TIMESTAMP[(p)]	年、月、日、時、分、秒、少数秒 pは少数秒の桁数を0～9で指定 デフォルトは6	TIMESTAMP(5)

使用するRDBMSにより利用可能なデータ型は異なる。

1.3 データ型

□ データ型の使用例(1)

■ 文字型(Char/Varchar)

値を指定する場合は、'(シングルクォーテーション)で囲む。

□ 例)CHAR(12)

'あいうえお' を入力 → 'あいうえお '(空白を2バイト)

※指定した桁数に満たない部分は空白を格納する。

□ 例)VARCHAR(12)

'あいうえお' を入力 → 'あいうえお'

※10バイト分を格納する。

■ 数値型(Numeric)

□ 例)NUMERIC(3,4) → 整数桁数3桁、小数以下桁数4桁

0 ~ ±999.9999の値を格納する。

1.3 データ型

□ データ型の使用例(2)

■ 日付型(DATE/TIMESTAMP)

値を指定する場合は、'(シングルクォーテーション)で囲む。

□ 例) DATE

2012年1月1日を入力する場合

'20120101', '120101', '2012-01-01'の指定が可能

□ 例) TIMESTAMP

'12-01-01 01:59:59.000000'

※本演習環境では、日付のデフォルトの表示書式はYY-MM-DDのため、
2012年1月1日のデータは、12-01-01と表示される。

1.3 データ型

□ 注文テーブルのテーブル構造

物理テーブル名 論理テーブル名
| |
T_ORER (注文テーブル)

物理 カラム名	データ型	論理 カラム名	備考
ORDER_NO	NUMERIC(3)	注文番号	プライマリキー
ITEM_NO	NUMERIC(4)	商品番号	
QUANTITY	NUMERIC(3)	数量	
AMOUNT	NUMERIC(7)	金額	
DELIVERY_DATE	DATE	納期	

1.3 データ型

□ 商品テーブルのテーブル構造

物理テーブル名 論理テーブル名
 | |
 T_ITEM (商品テーブル)

物理 カラム名	データ型	論理 カラム名	備考
ITEM_NO	NUMERIC(4)	商品番号	プライマリキー
ITEM_NAME	VARCHAR(20)	商品名	
PRICE	NUMERIC(4)	単価	



Memo

2. SQL



- 概要

SQLの概要とその歴史や最新バージョンについて学習する。

- 学習内容

- SQL
- SQLの歴史
- 最新バージョン

2.1 SQL

□ SQLとは

- Structured Query Languageの略
- データベースのデータにアクセスするための言語
- ANSI(American National Standard Institute)やISO(国際標準化機構)、JIS(日本工業規格)によって標準化されているため、特定のDBMS製品に依存せず、データを扱うことができる

2.2 SQL

□ SQLの歴史

- IBM社が開発した世界初のリレーショナルデータベース管理システム「System R」に実装されていた、データベース言語「SEQUEL(シークエル)」がSQLの元となっている
- 最新バージョンはSQL2008(ISO/IEC 9075)、日本ではJISX3005

年	規格
1986	SQL86(ANSI)
1987	SQL87(ISO)、JISX3005-1987(JIS)
1989	SQL89(ISO)
1992	SQL92/SQL2(ISO)
1999	SQL99(ISO/IEC)
2003	SQL2003(ISO/IEC)、JISX3005-2003(JIS)
2008	SQL2008(ISO/IEC)

2.3 SQL

□ SQLの策定者

- SQLは1986年にANSI(American National Standard Institute)によって標準化された。
- ANSIは、1918年に設立されたアメリカ国内の工業製品の規格を策定する団体。
- 日本のJISにあたる。



Memo

3. 参照系SQL

- 概要

SELECTを使用した基本的なデータベースへの問い合わせを学習する。

- 学習内容

- SELECT文の基本構文
- 重複レコードの排除
- レコードの指定
- WHERE句で使用する条件
- LIKE比較受験とワイルドカード
- 並べ替え
- 変換関数
- 集合関数
- カラム別名
- グループ化

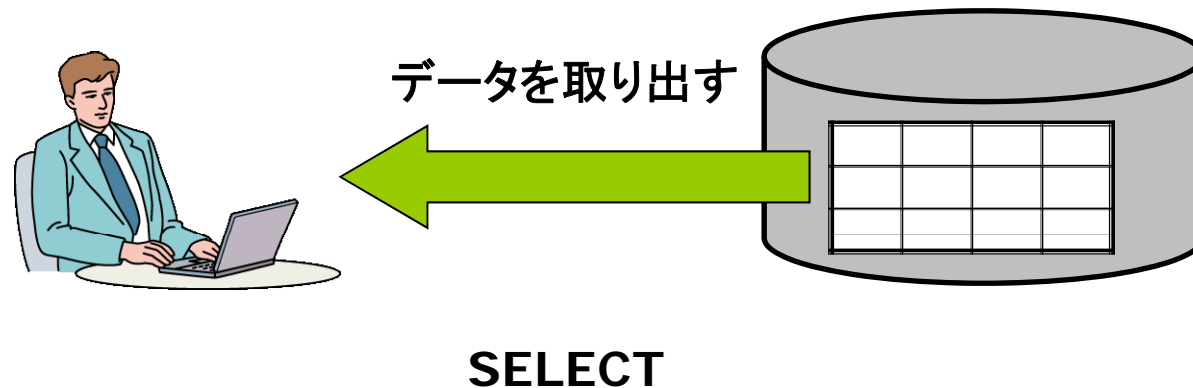
3.1 SELECT文の基本構文

□ 「参照系SQL」

『問い合わせ』に該当。SELECT文のことをいう。

□ SELECT文

- テーブルからデータを取り出す。
- 検索文(クエリ)とも呼ばれる。



3.1 SELECT文の基本構文

基本構文

SELECT [取得したいカラム名]
FROM [対象とするテーブル名] ;

- 取得したいカラム名 (SELECT句)
 - 対象とするテーブルの中に存在するカラム名や、演算式を指定する。
 - 複数のカラムを取得する場合は、カラム名を「,(カンマ)」で区切る。
 - 取得したいカラム名に「*(アスタリスク)」を指定すると、対象とするテーブルの全てのカラムを取り出す。
- 対象とするテーブル名 (FROM句)
 - データベースの中のどのテーブルを対象としてデータを取出したいかを指定する。

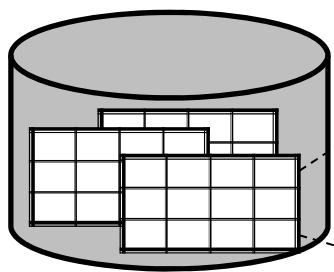
3.1 SELECT文の基本構文

□ SELECT文の考え方

SELECT 商品名 FROM 商品;

② ①

①データベースの中から、商品テーブルを取り出す。



商品テーブル

商品番号	商品名	単価
1001	ミカン	120
1002	リンゴ	90
1003	バナナ	120



②商品名のカラムだけを取り出す。

商品テーブル

商品番号	商品名	単価
1001	ミカン	120
1002	リンゴ	90
1003	バナナ	120

3.1 SELECT文の基本構文

□ 全カラム検索の例

SELECT * FROM 注文;

注文テーブル

注文番号	商品番号	数量	金額	納期
1	1001	30	3600	2012/4/25
2	1003	20	2400	2012/4/19
3	1001	45	5400	



注文番号	商品番号	数量	金額	納期
1	1001	30	3600	2012/4/25
2	1003	20	2400	2012/4/19
3	1001	45	5400	

3.1 SELECT文の基本構文

□ 指定カラム検索の例

SELECT 注文番号, 商品番号 FROM 注文;

注文テーブル

注文番号	商品番号	数量	金額	納期
1	1001	30	3600	2012/4/25
2	1003	20	2400	2012/4/19
3	1001	45	5400	



注文番号	商品番号
1	1001
2	1003
3	1001

3.2 重複レコードの排除

□ DISTINCT

- 重複するレコードを1レコードにまとめる。

基本構文

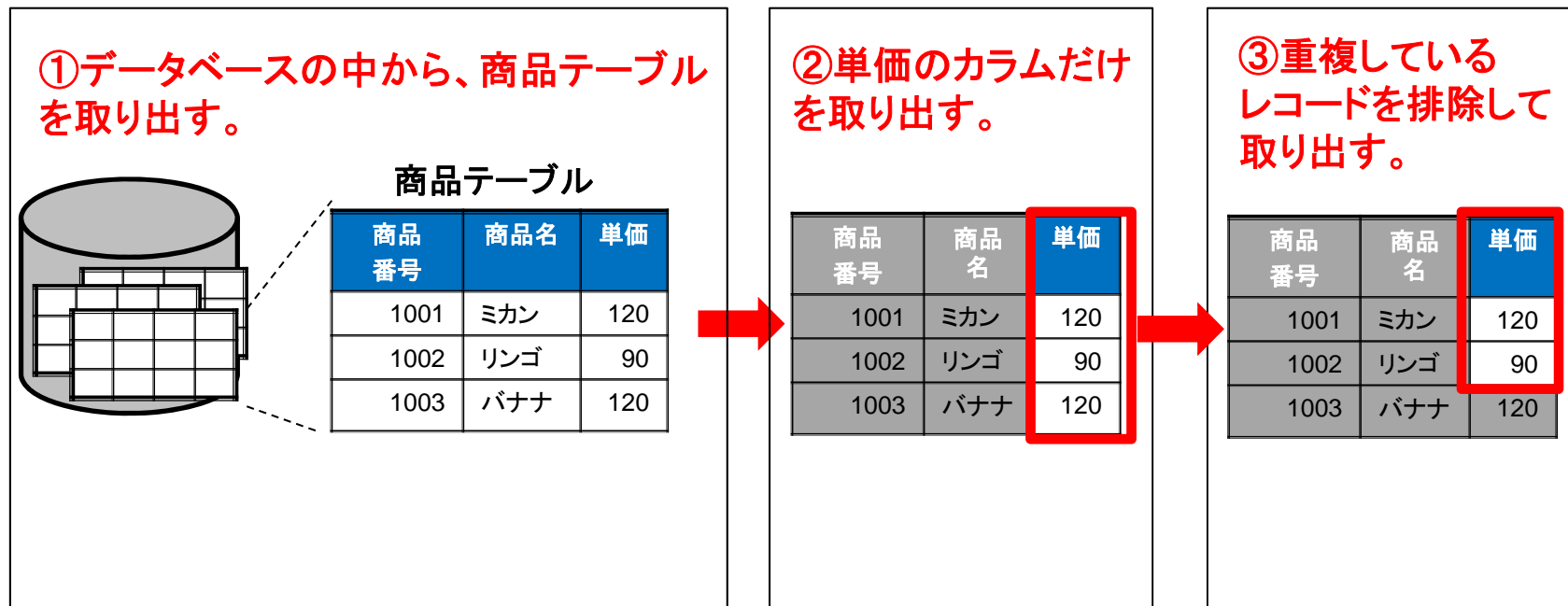
```
SELECT DISTINCT [取得したいカラム名]  
FROM [対象とするテーブル名];
```

3.2 重複レコードの排除

□ DISTINCTの考え方

SELECT DISTINCT 単価 FROM 商品;

③ ② ①



3.2 重複レコードの排除

□ DISTINCTの例

SELECT DISTINCT 商品番号 FROM 注文;

注文テーブル

注文番号	商品番号	数量	金額	納期
1	1001	30	3600	2012/4/25
2	1003	20	2400	2012/4/19
3	1001	45	5400	



商品番号
1003
1001

3.3 レコードの指定

□ WHERE句

- データベースから、特定のレコードを取得する。

基本構文

```
SELECT [取得したいカラム名]  
FROM [対象とするテーブル名]  
WHERE [取得したいレコードの条件] ;
```

□ 取得したいレコードの条件

- 対象とするテーブルの中で、どのレコードを取得するかを指定する。
- 省略した場合は全レコードが取得対象となる。
- 複数の条件を指定することもできる。

3.3 レコードの指定

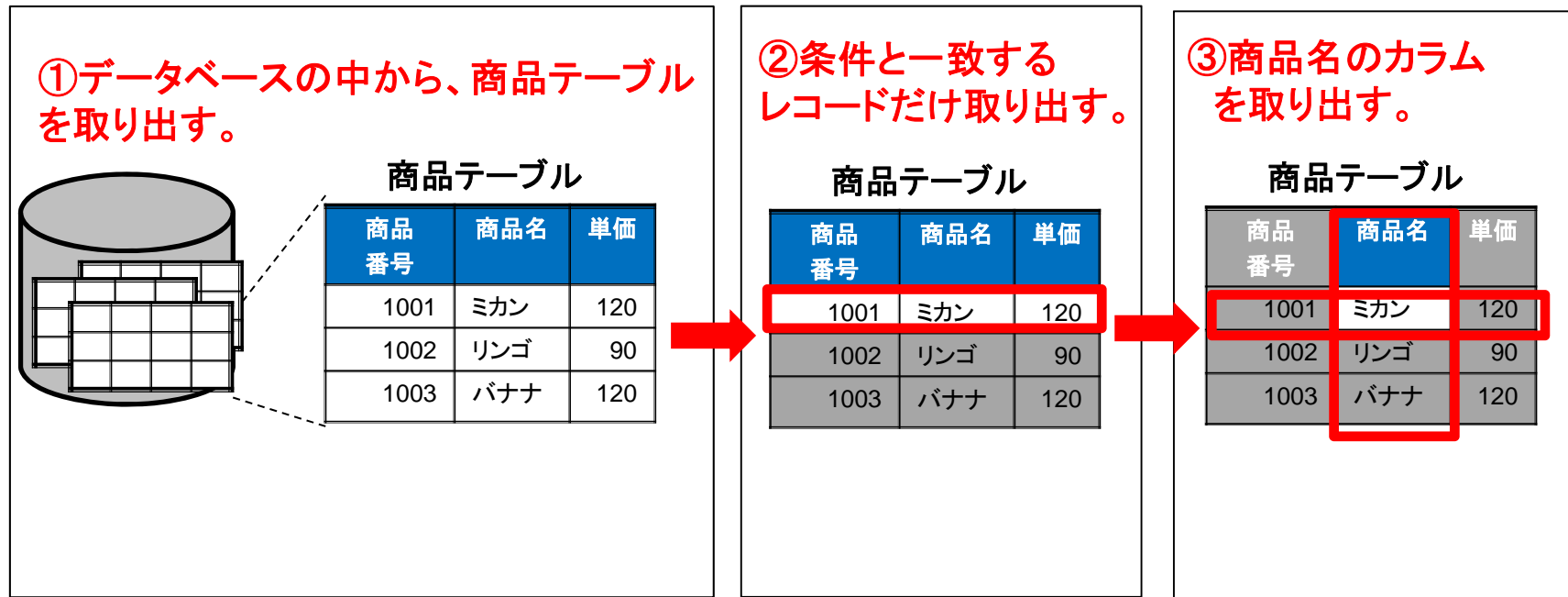
□ WHEREの考え方

SELECT 商品名 FROM 商品 WHERE 商品番号 = 1001;

③

①

②



3.4 WHERE句で使用する条件

□ 比較条件(1)

比較条件	意味
=	等しい
!=、<>、^=	等しくない
>=	以上
<=	以下
>	より大きい
<	より小さい
BETWEEN a AND b	a以上b以下 ※下限を先に指定
NOT BETWEEN a AND b	a以上b以下の範囲外

3.4 WHERE句で使用する条件

□ 比較条件(2)

比較条件	意味
IN(リスト)	リスト内のいずれかと等しい カラム名1 IN(A,B) = 「カラム名1 = A OR カラム名1 = B」
NOT IN(リスト)	リスト内のいずれとも等しくない カラム名1 NOT IN(A,B) = 「カラム名1 <> A AND カラム名2 <> B」
IS NULL	NULL値である
IS NOT NULL	NULL値でない
LIKE	文字パターンと一致する
NOT LIKE	文字パターンと一致しない

「=NULL」、「!=NULL」ではNULLを判定できない。
NULLを判定する場合は、「IS NULL」、「IS NOT NULL」を使用する。

3.4 WHERE句で使用する条件

□ 比較条件の例(1)

```
SELECT 注文番号, 数量 FROM 注文  
WHERE 数量 >= 30;
```

注文テーブル

注文番号	商品番号	数量	金額	納期
1	1001	30	3600	2012/4/25
2	1003	20	2400	2012/4/19
3	1001	45	5400	



注文番号	数量
1	30
3	45

3.4 WHERE句で使用する条件

□ 比較条件の例 (2)

SELECT 注文番号, 納期 FROM 注文
WHERE 納期 IS NULL;

注文テーブル

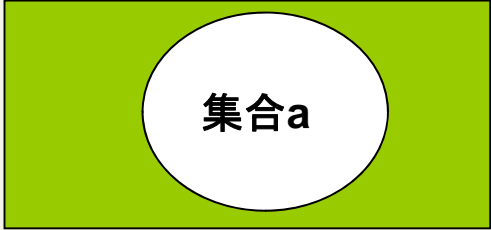
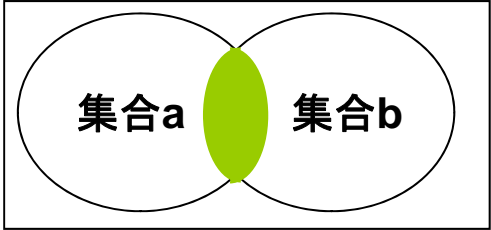
注文番号	商品番号	数量	金額	納期
1	1001	30	3600	2012/4/25
2	1003	20	2400	2012/4/19
3	1001	45	5400	



注文番号	納期
3	

3.4 WHERE句で使用する条件

□ 論理条件

論理条件	意味	ベン図
NOT	否定 NOT a (aでない)	
AND	論理積 a AND b (aかつb)	
OR	論理和 a OR b (aまたはb)	

3.4 WHERE句で使用する条件

□ 論理条件の例

```
SELECT 注文番号, 数量, 金額 FROM 注文  
WHERE 数量 > 30 AND 金額 > 3000;
```

注文テーブル

注文番号	商品番号	数量	金額	納期
1	1001	30	3600	2012/4/25
2	1003	20	2400	2012/4/19
3	1001	45	5400	



注文番号	数量	金額
3	45	5400

3.4 WHERE句で使用する条件

□ 条件の優先順位

条件	優先順位
=、>、>=、<、<=	 高 ↑ ↓ 低
IS [NOT] NULL、 [NOT] LIKE、 [NOT] IN	
[NOT] BETWEEN	
<>	
NOT	
AND	
OR	

上記の優先順位を変えたい場合は()をつける。

3.4 WHERE句で使用する条件

□ 条件の優先順位の例(1)

SELECT 商品名, 単価 FROM 商品

WHERE 商品名 = 'ミカン' OR

商品名 = 'リンゴ' AND 単価 = 90;

条件1

条件2

商品テーブル

商品番号	商品名	単価
1001	ミカン	120
1002	リンゴ	90
1003	バナナ	120



商品名	単価
ミカン	120
リンゴ	90

3.4 WHERE句で使用する条件

□ 条件の優先順位の例(2)

SELECT 商品名, 単価 FROM 商品

WHERE (商品名 = 'ミカン' OR 商品名 = 'リンゴ')

条件1

AND 単価 = 90;

条件2

商品テーブル

商品番号	商品名	単価
1001	ミカン	120
1002	リンゴ	90
1003	バナナ	120



商品名	単価
リンゴ	90

3.5 LIKE比較条件とワイルドカード

□ LIKE比較条件

- 文字パターンと部分的に一致するかどうかを判定する。
- 文字パターンは「ワイルドカード」を使って表現する。

□ ワイルドカード

ワイルドカード	内容
%	0文字以上の任意の文字
_(アンダースコア)	任意の1文字

3.5 LIKE比較条件とワイルドカード

□ LIKE比較条件の例

```
SELECT 商品名 FROM 商品  
WHERE 商品名 LIKE '%ン';
```

商品テーブル

商品番号	商品名	単価
1001	ミカン	120
1002	リンゴ	90
1003	バナナ	120



商品名
ミカン

SQLで文字を表記する際には
「'(シングルクォーテーション)」で前後を
囲みます。

3.5 LIKE比較条件とワイルドカード

□ LIKE比較条件の種類

■ 前方一致

- カラムの値の前方が条件と一致する。

例) 商品名 LIKE 'バ%'

商品名が「バ」で始まる → 「バナナ」が一致

■ 後方一致

- カラムの値の後方が条件と一致する。

例) 商品名 LIKE '%ン'

商品名が「ン」で終わる → 「ミカン」が一致

■ 部分一致

- カラムの値が条件を含んでいる。

例) 商品名 LIKE '%ン%'

商品名に「ン」を含む → 「ミカン」、「リンゴ」が一致

例) 商品名 LIKE '_ン_'

商品名が3文字で2文字目が「ン」 → 「リンゴ」が一致

3.5 LIKE比較条件とワイルドカード

□ ESCAPEオプション

- ワイルドカード(_や%)自身をLIKE条件で検索したい場合、ESCAPEオプションを使用する。
- 任意の記号をエスケープ文字として指定する。

□ ESCAPEオプションの例

- 「¥」をエスケープ文字として、「ミカン_生産地名」のパターンの文字列を検索したい場合

商品名 LIKE 'ミカン¥_%' ESCAPE '¥';

例)「ミカン_愛媛」が一致

3.6 並べ替え

□ ORDER BY句

- レコードを並べ替える。

基本構文

```
SELECT [取得したいカラム名] FROM [対象とするテーブル名]  
WHERE [取得したいレコードの条件]  
ORDER BY [並べ替えるカラム名][キーワード] ;
```

□ 並べ替えるカラム名/キーワード

- 並べ替えの基準にしたいカラム名とキーワードを指定する。
 - ASC(ASCENDING)キーワード/キーワード省略 : 昇順に並べ替える。
 - DESC(DESCENDING)キーワード : 降順に並べ替える。
- カラム名は複数記述することができる。
 - 記述したカラム名の順に並べ替えを行う。
 - カラム名ごとに昇順、降順を指定できる。 例:カラムAは昇順、カラムBは降順

3.6 並べ替え

□ データの種類とソートの順序

■ 数値

- 昇順 : 数値の小さい値から大きな値に並べ替える

■ 文字

- 昇順 : 文字コード順に並べ替える

※NULL値が含まれるデータを昇順で並べ替える場合、NULL値は一番最後に表示される。

■ 日付

- 昇順 : 古い日付から新しい日付の順に並べ替える

3.6 並べ替え

□ ORDER BYの考え方

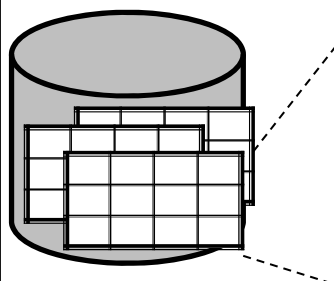
SELECT 商品名, 単価 FROM 商品 ORDER BY 単価 ASC;

②

①

③

①データベースの中から、商品テーブルを取り出す。



商品テーブル

商品番号	商品名	単価
1001	ミカン	120
1002	リンゴ	90
1003	バナナ	120

②商品名、単価を取り出す。

商品テーブル

商品番号	商品名	単価
1001	ミカン	120
1002	リンゴ	90
1003	バナナ	120

③単価をキーにレコードを昇順に並べ替える。

商品テーブル

商品番号	商品名	単価
1002	リンゴ	90
1003	バナナ	120
1001	ミカン	120

3.6 並べ替え

□ ORDER BYの例(1)

SELECT 注文番号, 商品番号, 数量 FROM 注文
ORDER BY 数量 DESC;

注文テーブル

注文番号	商品番号	数量	金額	納期
1	1001	30	3600	2012/4/25
2	1003	20	2400	2012/4/19
3	1001	45	5400	



注文番号	商品番号	数量
3	1001	45
1	1001	30
2	1003	20

3.6 並べ替え

□ ORDER BYの例(2)

```
SELECT 注文番号, 商品番号, 数量 FROM 注文  
ORDER BY 商品番号 DESC, 数量 ASC;
```

注文テーブル

注文番号	商品番号	数量	金額	納期
1	1001	30	3600	2012/4/25
2	1003	20	2400	2012/4/19
3	1001	45	5400	



注文番号	商品番号	数量
2	1003	20
1	1001	30
3	1001	45

商品番号が同じレコードは、
数量を昇順に並べ替える。

3.7 変換関数

- 変換関数とは
 - データ型の変換を行う。

関数	説明
TO_CHAR(数値 [,書式])	数値を書式で指定した文字列 <u>に</u> 変換して戻す
TO_CHAR(日付 [,書式])	日付を書式で指定した文字列 <u>に</u> 変換して戻す
TO_NUMBER(文字列 [,書式])	書式で指定した文字列 <u>を</u> 数値型に変換して戻す
TO_DATE(文字列 [,書式])	書式で指定した文字列 <u>を</u> 日付型に変換して戻す

・日付⇔数値を直接変換することはできない。

3.7 変換関数

□ 主な数値書式

要素	説明
9	9の数が有効な桁数を表す 例) 999999 → 12345
0	桁数に満たない場合、先頭に0を付けた数値を表す 例) 099999 → 012345
\$	\$記号を数値の前につけて表す 例) \$99999 → \$12345
L	ローカル通貨記号を数値の前につけて表す 例) L99999 → ¥12345
,	指定した位置に桁区切りのカンマをつけて表す 例) 99,999 → 12,345

3.7 変換関数

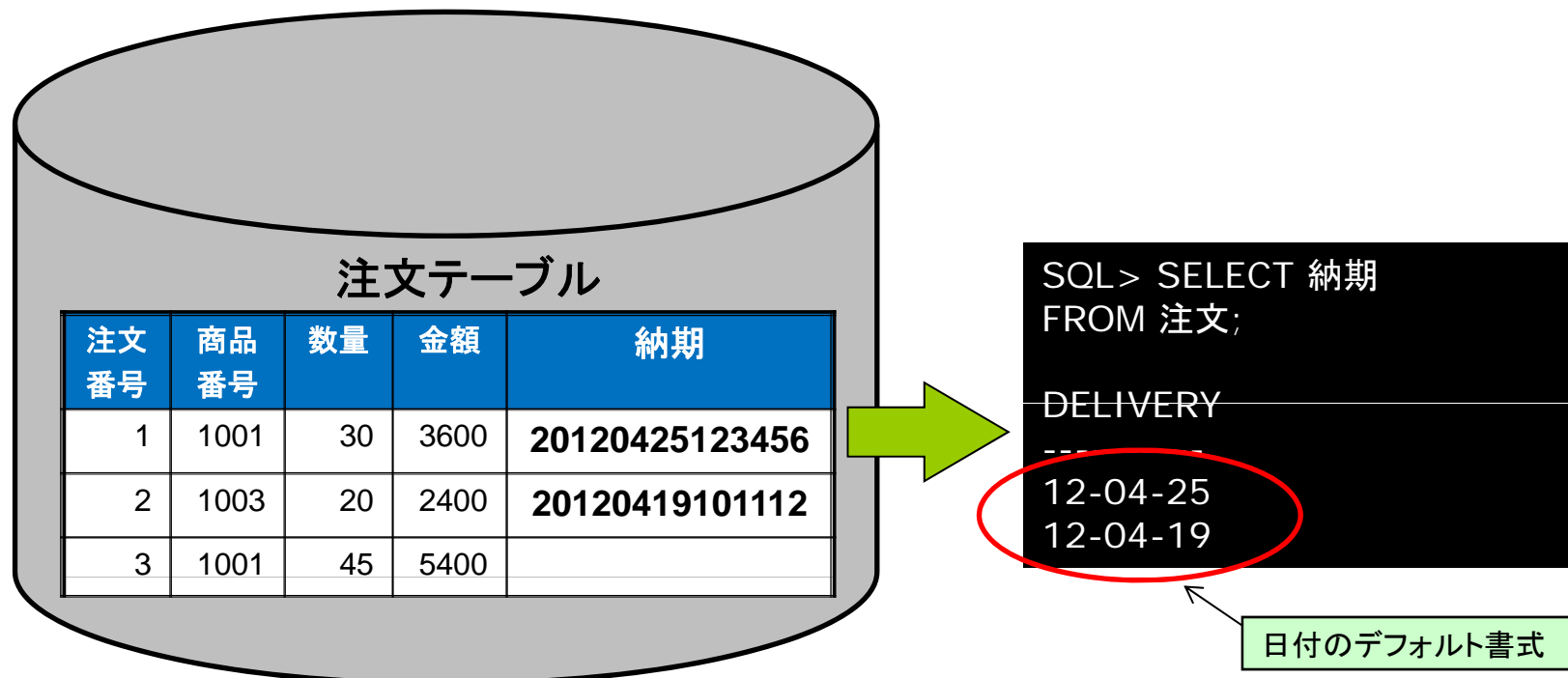
□ 主な日付書式

要素	説明
YYYY	年を4桁で表す
YY	年を下2桁で表す
MM	月を2桁で表す
MONTH	月(1月～12月)を表す(英語の場合、空白が埋め込まれた9文字の長さの月の名前 例)MAY△△△△△△
MON	月の名前(英語の場合、3文字の省略形)
DD	日(01～31日)を表す
DAY	曜日を表す(英語の場合、空白が埋め込まれた9文字の長さの曜日) 例)TUESDAY△△
HH24	時間(0～23)を表す
MI	分(00～59)を表す
SS	秒(00～59)を表す

3.7 変換関数

□ 日付データについて

- 実際に格納されている日付データはYYYYMMDDhhmmssの形式である。デフォルトの表示書式(YY-MM-DD)以外の書式で表示したい場合は、書式の指定が必要となる。



3.7 変換関数

□ 変換関数の例(1)

```
SELECT 注文番号, TO_CHAR(納期, 'YYYY-MM-DD')  
FROM 注文;
```

注文テーブル

注文番号	商品番号	数量	金額	納期
1	1001	30	3600	2012/4/25
2	1003	20	2400	2012/4/19
3	1001	45	5400	



注文番号	TO_CHAR(納期, 'YYYY-MM-DD')
1	2012-04-25
2	2012-04-19
3	

3.7 変換関数

□ 変換関数の例(2)

```
SELECT 注文番号, TO_CHAR(金額, 'L9, 999')  
FROM 注文;
```

注文テーブル

注文番号	商品番号	数量	金額	納期
1	1001	30	3600	2012/4/25
2	1003	20	2400	2012/4/19
3	1001	45	5400	



注文番号	TO_CHAR(金額, 'L9999')
1	¥3,600
2	¥2,400
3	¥5,400

3.8 集合関数

□ 集合関数とは

- レコードのグループ単位で1つの結果を返す。

関数名	説明
AVG	カラムの平均値を返す。(NULL値は無視)
COUNT	カラムがNULLでないレコード数を返す。 カラム名に*を指定すると、重複値およびNULL値を含めたレコード数を返す。COUNT(DISTINCT カラム名)で重複値を排除したレコードを返す。
MAX	カラムの最大値を返す。(NULL値は無視)
MIN	カラムの最小値を返す。(NULL値は無視)
SUM	カラムの合計値を返す。(NULL値は無視)

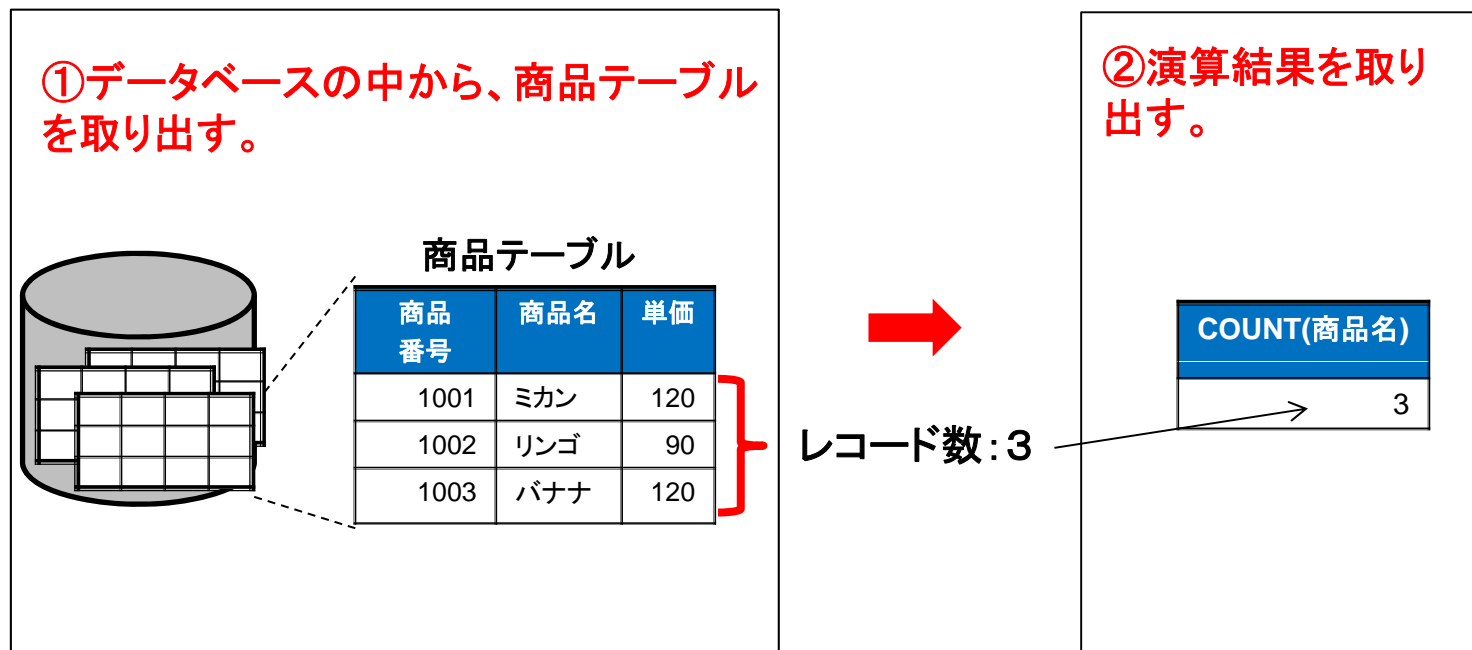
基本構文

```
SELECT 集合関数(カラム名) FROM [対象とするテーブル名];
```

3.8 集合関数

□ 集合関数の考え方

SELECT COUNT(商品名) FROM 商品;



3.8 集合関数

□ 集合関数の例

SELECT MAX(数量), AVG(金額) FROM 注文;

注文テーブル

注文番号	商品番号	数量	金額	納期
1	1001	30	3600	2012/4/25
2	1003	20	2400	2012/4/19
3	1001	45	5400	



MAX(数量)	AVG(金額)
45	3800

3.9 カラム別名

□ カラム別名とは

- 問合せの結果を取り出す際に、カラムの別名を指定して、結果を読みやすくする。

□ カラム別名の指定方法

- ① カラム名の後ろに1つ以上の空白を入れる。
- ② カラム名の後ろにASキーワードを指定する。
- ③ 命名規則に違反する別名を使用したい場合（先頭の文字だけ大文字にしたいなど）は、カラム名の後ろに「"（ダブルクォーテーション）」で囲んで指定する。

□ カラム別名の考え方

SELECT	<u>COUNT(商品名)</u>	<u>合計</u>	FROM	商品;
SELECT	<u>COUNT(商品名)</u>	<u>AS 合計</u>	FROM	商品;
SELECT	<u>COUNT(商品名)</u>	<u>"合計"</u>	FROM	商品;
	②	③		①

①データベースの中から、商品テーブルを取り出す。

商品テーブル

商品番号	商品名	単価
1001	ミカン	120
1002	リンゴ	90
1003	バナナ	120

②演算結果を別名で取り出す。

合計	3
----	---

レコード数: 3

3.9 カラム別名

□ カラム別名の例

SELECT

MAX(数量) AS 最大個数, AVG(金額) AS 平均金額

FROM 注文;

注文テーブル

注文番号	商品番号	数量	金額	納期
1	1001	30	3600	2012/4/25
2	1003	20	2400	2012/4/19
3	1001	45	5400	

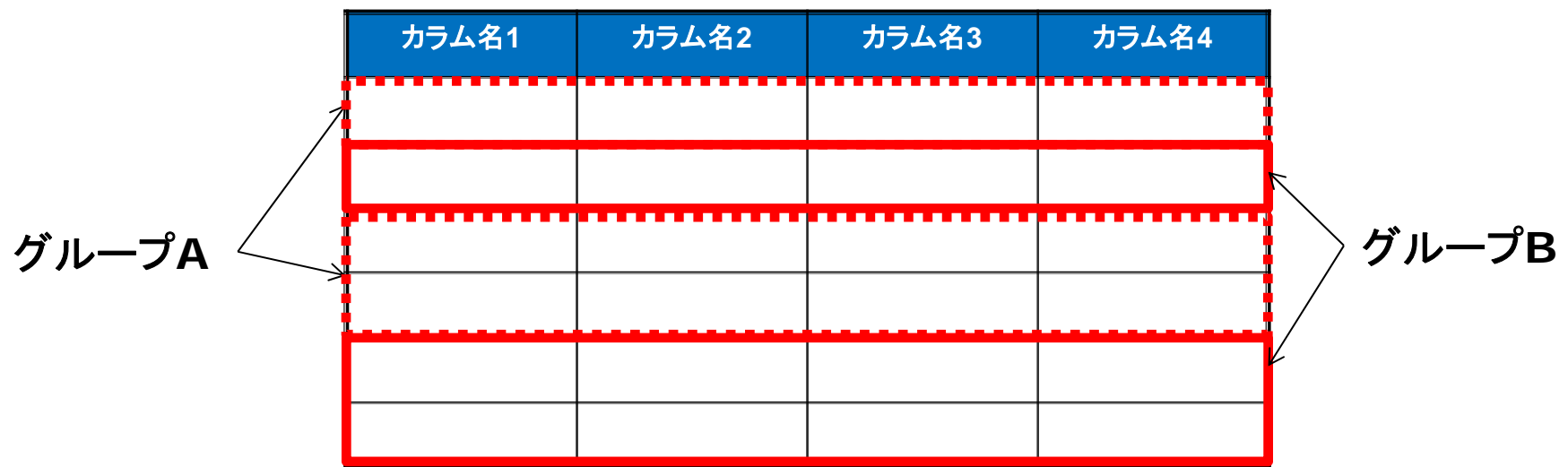


最大個数	平均金額
45	3800

3.10 グループ化

□ グループ化とは

- 1つのテーブルをレコード単位で複数のグループに分割すること。



3.10 グループ化

基本構文

```
SELECT [取得したいカラム名]  
FROM [対象とするテーブル名]  
WHERE [取得したいレコードの条件]  
GROUP BY [グループ化したいカラム名] ;
```

□ グループ化したいカラム名

- どのカラムの値を基にグループ化するか、カラム名を指定する。
- GROUP BYを使う場合、SELECT句で指定できるカラム名はGROUP BYで指定したカラム名か、集合関数のみ。
- 記述の順番はWHERE句の後に記述する。記述順に注意。

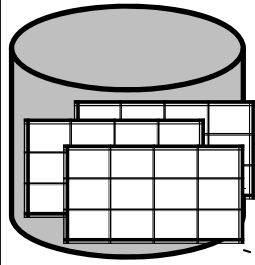
3.10 グループ化

□ GROUP BYの考え方

SELECT 商品番号, COUNT(商品番号) AS 注文件数
FROM 注文 GROUP BY 商品番号;

① ② ③

①データベースの中から、注文テーブルを取り出す。



注文テーブル

注文番号	商品番号	...	納期
1	1001	...	2012/4/25
2	1003	...	2012/4/19
3	1001	...	2012/4/25

②商品番号ごとにグループ化。

注文テーブル

注文番号	商品番号	...	納期
1	1001	...	2012/4/25
2	1003	...	2012/4/19
3	1001	...	2012/4/25

グループA グループB

③商品番号、演算結果を別名で取り出す。

商品番号	注文件数
1003	1
1001	2

3.10 グループ化

□ GROUP BYの例

SELECT 商品番号, SUM(金額) AS 合計金額
FROM 注文 GROUP BY 商品番号;

注文テーブル

注文番号	商品番号	数量	金額	納期
1	1001	30	3600	2012/4/25
2	1003	20	2400	2012/4/19
3	1001	45	5400	



商品番号	合計金額
1003	2400
1001	9000

3.10 グループ化

□ HAVING句

- グループ化されたレコードに対して条件を与える。

基本構文

```
SELECT [取得したいカラム名]  
FROM [対象とするテーブル名]  
WHERE [取得したいレコードの条件]  
GROUP BY [グループ化したいカラム名]  
HAVING [取得したいグループの条件] ;
```

□ 取得したいグループの条件

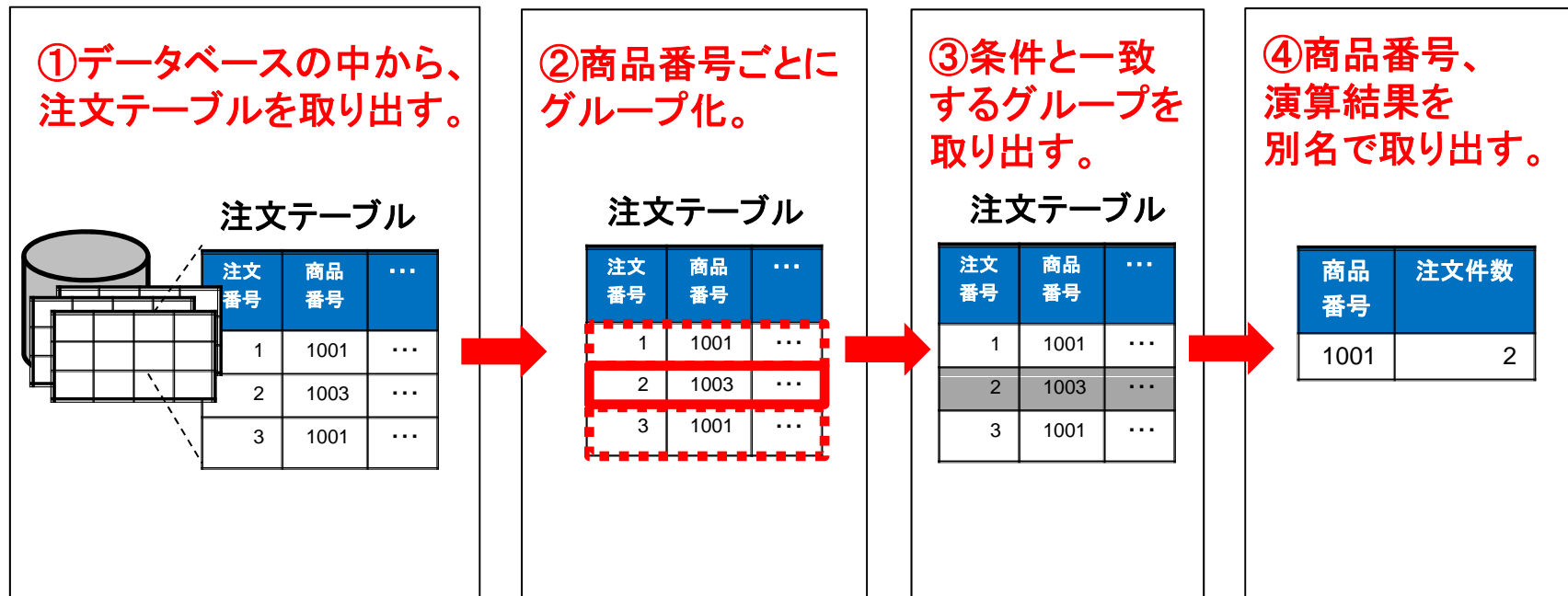
- 集合関数を条件に含めることができる(WHERE句には集合関数は含めることができない)。
- GROUP BY句がなくても使用できる。その場合、問い合わせ結果全体を1つのグループとみなして処理する。

3.10 グループ化

□ HAVINGの考え方

SELECT 商品番号, COUNT(商品番号) AS 注文件数
FROM 注文 GROUP BY 商品番号 HAVING 商品番号 = 1001;

① ② ③ ④



3.10 グループ化

□ HAVINGの例

```
SELECT 商品番号, SUM(金額) AS 合計金額  
FROM 注文 GROUP BY 商品番号 HAVING SUM(金額) >= 5000;
```

注文テーブル

注文番号	商品番号	数量	金額	納期
1	1001	30	3600	2012/4/25
2	1003	20	2400	2012/4/19
3	1001	45	5400	



商品番号	合計金額
1001	9000



Memo

4.更新系SQL

- 概要

UPDATE,INSERT,DELETEを使用した基本的なデータベースの更新について学習する。

- 学習内容

- 更新系SQLの種類
- UPDATE文の基本構文
- INSERT文の基本構文
- DELETE文の基本構文
- DELETEの物理削除と論理削除

4.1 更新系SQLの種類

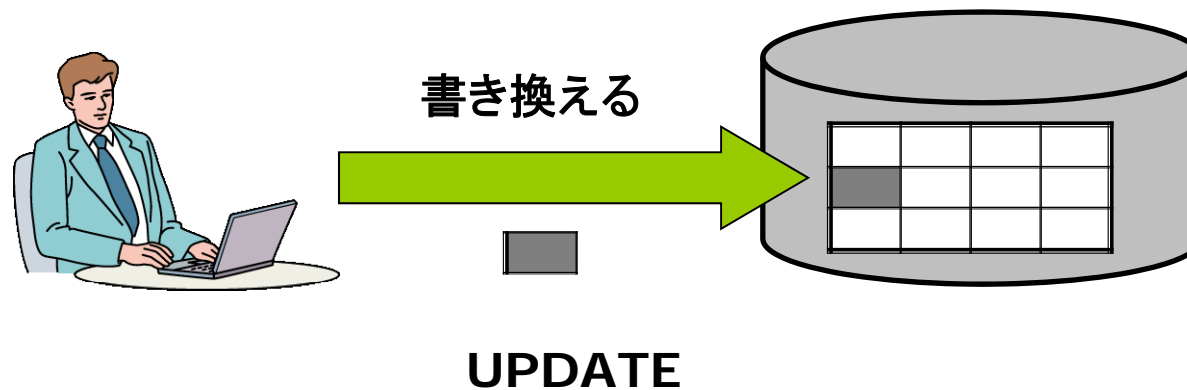
□ 「更新系SQL」は以下の3種類がある。

- レコードの値を変更 → UPDATE文
- レコードの追加 → INSERT文
- レコードの削除 → DELETE文

4.2 UPDATE文の基本構文

□ UPDATE文

- テーブルのデータを書き換える。



4.2 UPDATE文の基本構文

基本構文

```
UPDATE [対象とするテーブル名]  
SET [書き換えたいカラム名]=[値],...  
WHERE [書き換えたいレコードの条件];
```

- 対象とするテーブル内の書き換えたいカラムを「値」で書き換える。
- 書き換えたいカラム名は、複数記述することも可能。
- WHEREを省略すると全レコードが更新対象となる。

4.2 UPDATE文の基本構文

□ UPDATEの考え方(全件更新)

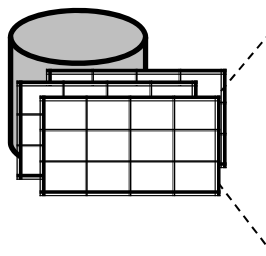
UPDATE 商品 SET 単価 = 100;

①

②

①データベースの中から、
商品テーブルを取り出す。

商品テーブル



商品 番号	商品名	単価
1001	ミカン	120
1002	リンゴ	90
1003	バナナ	120



②単価を「100」に
書き換える。

商品テーブル

商品 番号	商品名	単価
1001	ミカン	100
1002	リンゴ	100
1003	バナナ	100

4.2 UPDATE文の基本構文

□ UPDATEの例(全件更新)

UPDATE 注文 SET 納期 = '20120501';

注文テーブル

注文番号	商品番号	数量	金額	納期
1	1001	30	3600	2012/4/25
2	1003	20	2400	2012/4/19
3	1001	45	5400	



注文番号	商品番号	数量	金額	納期
1	1001	30	3600	2012/5/1
2	1003	20	2400	2012/5/1
3	1001	45	5400	2012/5/1

4.2 UPDATE文の基本構文

□ UPDATEの考え方(指定レコード更新)

UPDATE 商品

①

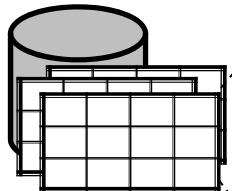
SET 商品名 = 'オレンジ', 単価 = 100

③

WHERE 商品番号 = 1001;

②

①データベースの中から、
商品テーブルを取り出す。



商品テーブル

商品番号	商品名	単価
1001	ミカン	120
1002	リンゴ	90
1003	バナナ	120

②商品番号が1001の
レコードを取り出す。

商品テーブル

商品番号	商品名	単価
1001	ミカン	120
1002	リンゴ	90
1003	バナナ	120

③商品名を「オレンジ」に、
単価を「100」に書き換える。

商品テーブル

商品番号	商品名	単価
1001	オレンジ	100
1002	リンゴ	90
1003	バナナ	120

4.2 UPDATE文の基本構文

□ UPDATEの例(指定レコード更新)

UPDATE 注文 SET 納期 = '20120501'

WHERE 注文番号 = 3;

注文テーブル

注文番号	商品番号	数量	金額	納期
1	1001	30	3600	2012/4/25
2	1003	20	2400	2012/4/19
3	1001	45	5400	

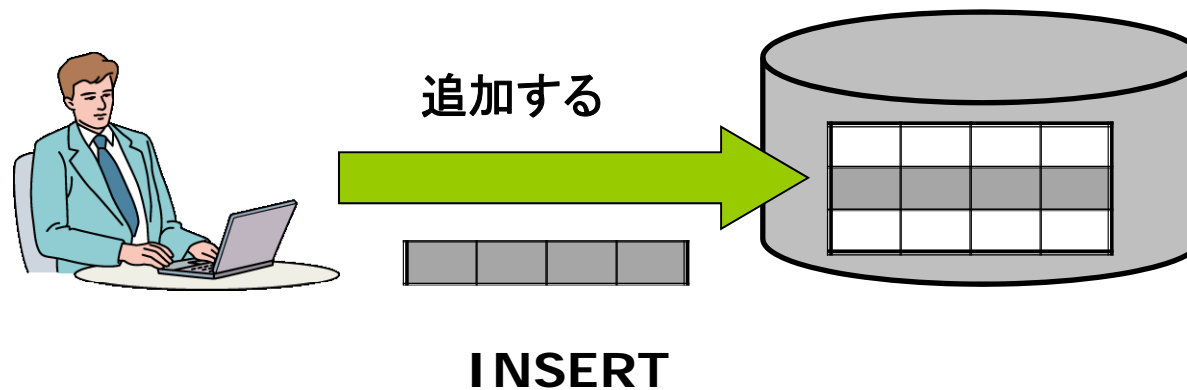


注文番号	商品番号	数量	金額	納期
1	1001	30	3600	2012/4/25
2	1003	20	2400	2012/4/19
3	1003	45	5400	2012/5/1

4.3 INSERT文の基本構文

□ INSERT文

- テーブルにデータを追加(挿入)する。



4.3 INSERT文の基本構文

基本構文

```
INSERT INTO [対象とするテーブル名]  
([カラム名1],[カラム名2],...)  
VALUES ([値1],[値2],...);
```

- 対象とするテーブルの指定したカラムに指定した値を持つレコードを追加する。
- カラム名を指定する部分を記述した場合、カラム名と値の記述順が一致していないといけない。
- カラム名を指定する部分を省略することも可能だが、その場合は値を記述する順序をデータベース上で定義されている順序にあわせる必要がある。

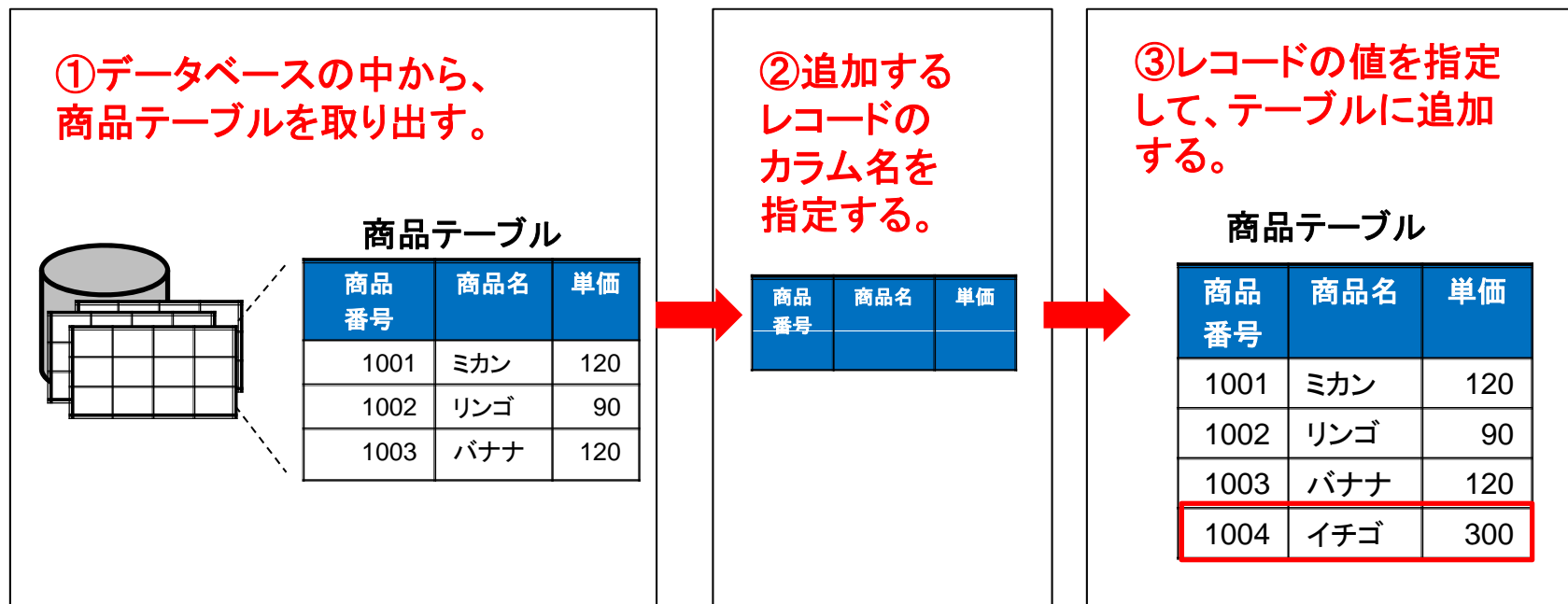
4.3 INSERT文の基本構文

□ INSERTの考え方

INSERT INTO 商品

(商品番号, 商品名, 単価)

VALUES (1004, 'イチゴ', 300);



4.3 INSERT文の基本構文

□ INSERTの例(カラム名指定あり)

INSERT INTO 注文

(注文番号, 商品番号, 数量, 金額, 納期)

VALUES (4, 1004, 25, 7500, '20120525');

注文テーブル

注文番号	商品番号	数量	金額	納期
1	1001	30	3600	2012/4/25
2	1003	20	2400	2012/4/19
3	1001	45	5400	



注文番号	商品番号	数量	金額	納期
1	1001	35	3600	2012/4/25
2	1003	35	2400	2012/4/19
3	1001	35	5400	
4	1004	25	7500	2012/5/25

4.3 INSERT文の基本構文

□ INSERTの例(カラム名指定なし)

INSERT INTO 注文

VALUES (4, 1004, 25, 7500, '201205025');

注文テーブル

注文番号	商品番号	数量	金額	納期
1	1001	30	3600	2012/4/25
2	1003	20	2400	2012/4/19
3	1001	45	5400	

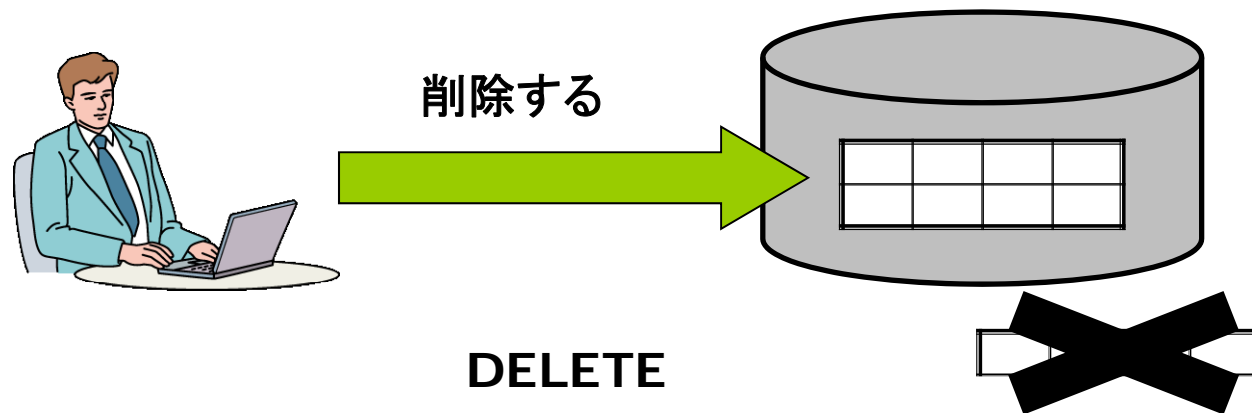


注文番号	商品番号	数量	金額	納期
1	1001	35	3600	2012/4/25
2	1003	35	2400	2012/4/19
3	1001	35	5400	
4	1004	25	7500	2012/5/25

4.4 DELETE文の基本構文

□ DELETE文

- テーブルからデータを削除する。



4.4 DELETE文の基本構文

基本構文

**DELETE FROM [対象とするテーブル名]
WHERE [削除したいレコードの条件] ;**

- 対象とするテーブルの削除したいレコードの条件に一致するレコードを削除する。
- WHERE句は省略可能。省略した場合は全てのレコードを削除する(全件削除)。

4.4 DELETE文の基本構文

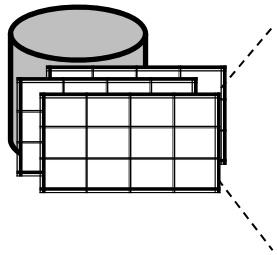
□ DELETEの考え方(全件削除)

DELETE FROM 商品;

②

①

①データベースの中から、
商品テーブルを取り出す。



商品テーブル

商品 番号	商品名	単価
1001	ミカン	120
1002	リンゴ	90
1003	バナナ	120

②全レコードを
削除する。

商品テーブル

商品 番号	商品名	単価
----------	-----	----

4.4 DELETE文の基本構文

□ DELETEの例(全件)

DELETE FROM 注文;

注文テーブル

注文番号	商品番号	数量	金額	納期
1	1001	30	3600	2012/4/25
2	1003	20	2400	2012/4/19
3	1001	45	5400	



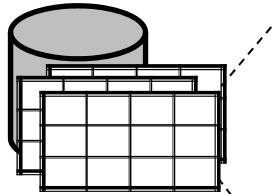
注文番号	商品番号	数量	金額	納期
------	------	----	----	----

4.4 DELETE文の基本構文

□ DELETEの考え方(指定レコード削除)

DELETE FROM 商品
③ WHERE 商品番号 = 1001;
②

①データベースの中から、
商品テーブルを取り出す。



商品テーブル

商品番号	商品名	単価
1001	ミカン	120
1002	リンゴ	90
1003	バナナ	120

②条件と一致する
レコードだけ
取り出す。

商品テーブル

商品番号	商品名	単価
1001	ミカン	120
1002	リンゴ	90
1003	バナナ	120

③②のレコードを
削除する。

商品テーブル

商品番号	商品名	単価
1002	リンゴ	90
1003	バナナ	120

4.4 DELETE文の基本構文

□ DELETEの例(指定レコード削除)

DELETE FROM 注文

WHERE 注文番号 = 3;

注文テーブル

注文番号	商品番号	数量	金額	納期
1	1001	30	3600	2012/4/25
2	1003	20	2400	2012/4/19
3	1001	45	5400	

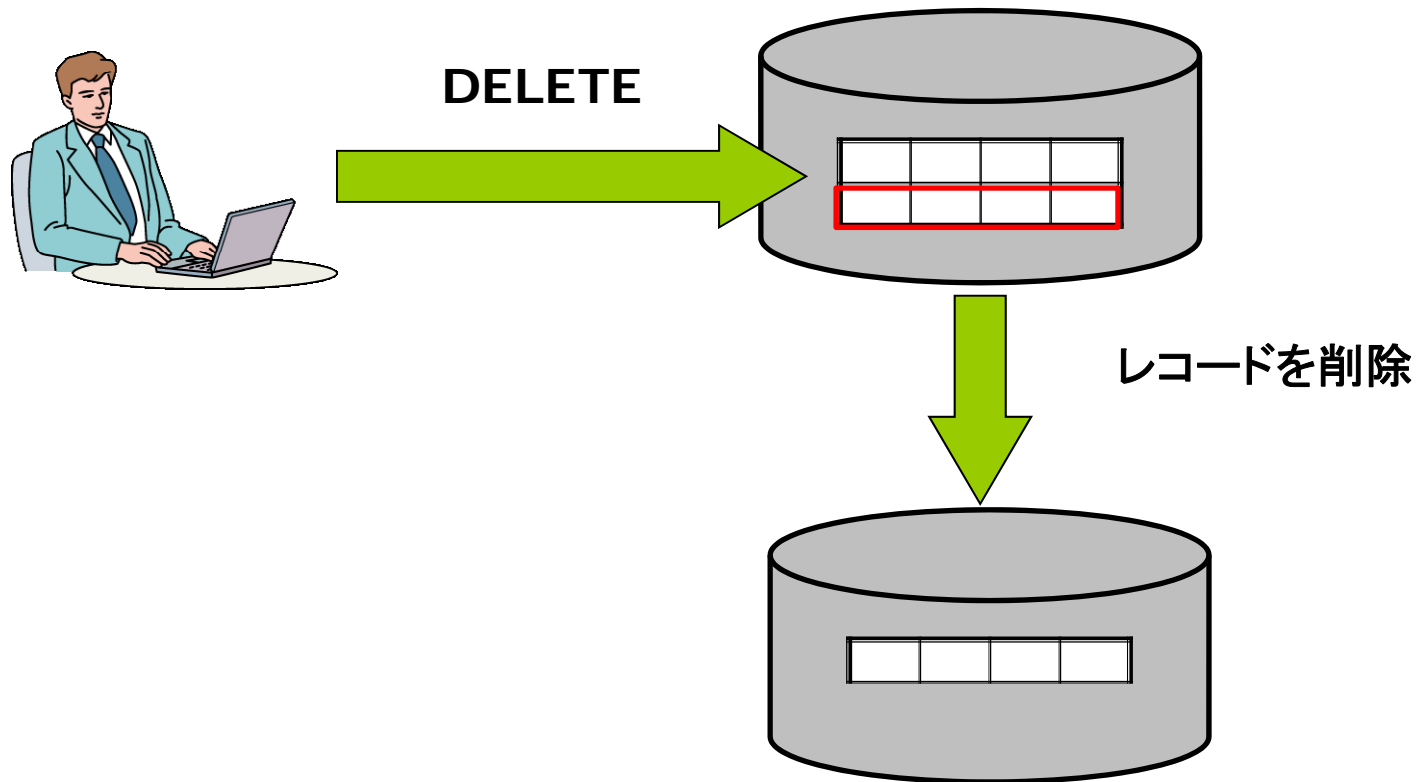


注文番号	商品番号	数量	金額	納期
1	1001	30	3600	2012/4/25
2	1003	20	2400	2012/4/19

4.5 DELETEの物理削除と論理削除

□ 物理削除

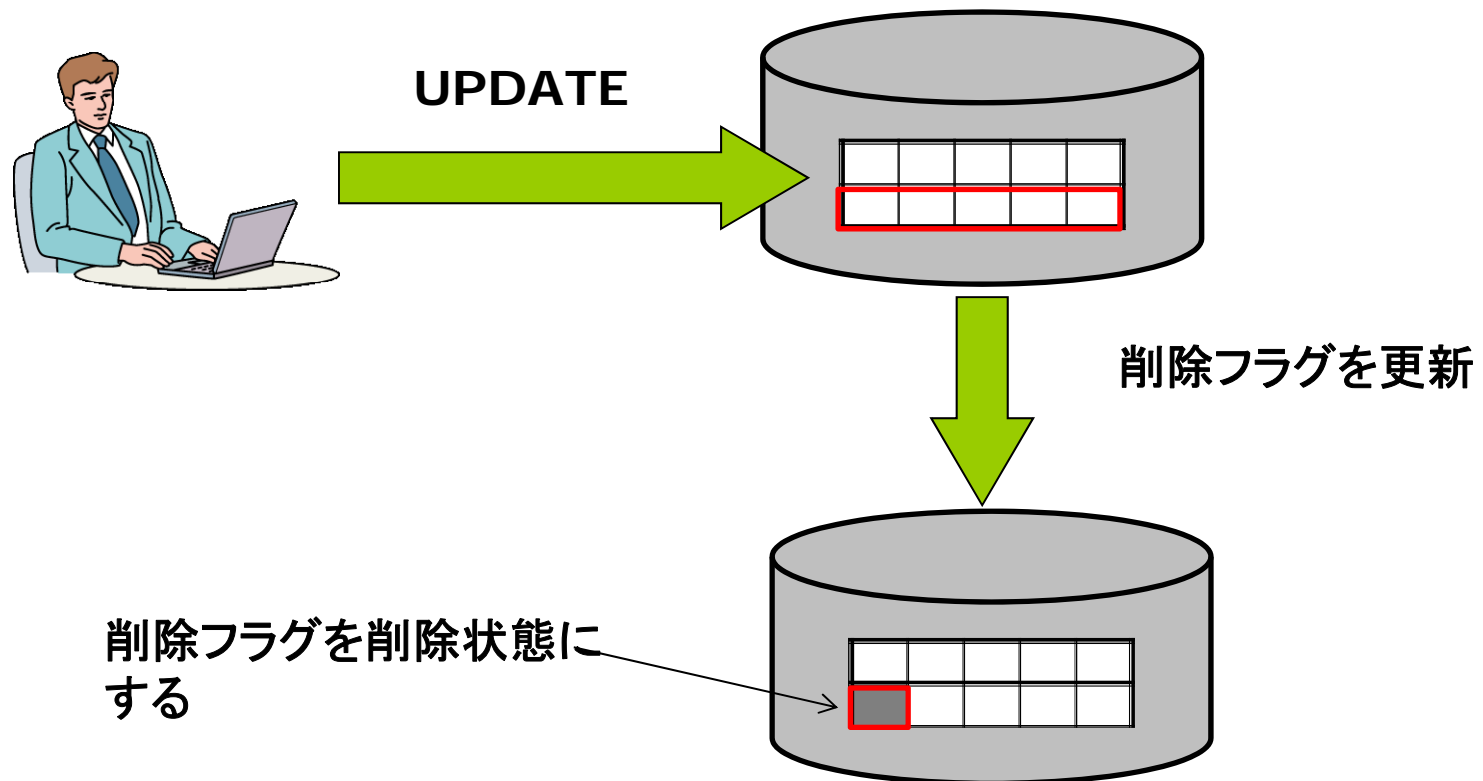
- テーブルからレコードを削除する。



4.5 DELETEの物理削除と論理削除

□ 論理削除

- テーブルに削除フラグという領域を用意し、削除時は削除フラグを削除状態にする。
- 削除フラグが削除状態のレコードは、実際には存在しているが、業務上は削除扱いとする。



4.5 DELETEの物理削除と論理削除

□ 論理削除のメリット

- 誤ってレコードを削除してしまった場合に、簡単に元に戻せる。
- レコードの削除、挿入の繰り返しによる、データベース内のデータが断片化することを抑えられる。

□ 論理削除のデメリット

- データの取り出し時にWHERE句の条件で、削除フラグの確認が必要になる。
- データに個人情報が含まれる場合、データの扱いには十分な注意が必要となる。



Memo

5. DML 【Data Manipulation Language】

- ・概要

DML 【Data Manipulation Language】の基本を学習する。

- ・学習内容

- 複数テーブルからの取得(結合)
- WHERE句を使用した結合
- 直積結合(CROSS JOIN)
- 内部結合(JOIN)
- 外部結合(OUTER JOIN)
- 自然結合(NATURAL JOIN)
- 副問い合わせ
- 副問い合わせを使用したUPDATE
- 副問い合わせを使用したINSERT
- 副問い合わせを使用したDELETE
- 和集合(UNION)

5.1 複数テーブルからの取得(結合)

□ 結合とは

- 複数のテーブルをあるカラムの値を基に連結し、1つの結果を取り出すこと。

□ テーブルを結合する理由

- テーブルは一般的に正規化されているため、必要とするデータを取得するためには、複数のテーブルを結合する処理が必要になる。

5.1 複数テーブルからの取得(結合)

□ カラム名の修飾

- 2つ以上のテーブルを利用するとき、それぞれのテーブルに同じカラム名が存在する場合、どちらのテーブルのカラムなのかを指定する必要がある。
- 「テーブル名.カラム名」でどのテーブルのカラムかを指定する。
- 1つのテーブルにしか存在しないカラム名もテーブル名で修飾することで、テーブルを検索する負荷が軽減され、パフォーマンスが向上する。

□ カラム名の修飾の例

```
SELECT 商品.商品番号, 商品.商品名 FROM 商品;
```

5.1 複数テーブルからの取得(結合)

□ テーブル別名

- テーブルに別名をつけることができる。
- テーブル別名をつけることで長いテーブル名を指定しなくてすむ。
- テーブル名の後に空白を置き、テーブル別名を指定する。
- テーブル別名を指定した場合、そのSELECT文の中では、テーブル名でカラム名を修飾する。

□ テーブル別名の例

- 注文テーブルに「T」、商品テーブルに「S」という別名をつけた場合

```
SELECT T. 注文番号, S. 商品番号, S. 商品名, T. 数量  
FROM 注文 T, 商品 S;
```

5.1 複数テーブルからの取得(結合)

□ 結合の種類

- WHERE句を使用した結合
- JOIN句を使用した結合
- 直積結合(CROSS JOIN)
- 内部結合(INNER JOIN)
- 外部結合(OUTER JOIN)
 - ・ 左側外部結合(LEFT OUTER JOIN)
 - ・ 右側外部結合(RIGHT OUTER JOIN)
- 自然結合(NATURAL JOIN)

5.2 WHERE句を使用した結合

基本構文

```
SELECT [カラム名] FROM [テーブル名1,テーブル名2]  
WHERE [結合条件];
```

- WHERE句を使用した結合
 - WHERE句へテーブル名1、テーブル名2に存在する同じカラム名を結合条件として記述する。

5.2 WHERE句を使用した結合

□ WHERE句を使用した結合の例

- 注文テーブルと商品テーブルに対して、商品番号で結合をおこなう。

SELECT

 O.注文番号,O.商品番号,
 I.商品名,I.単価,O.数量,
 O.金額,O.納期

FROM

 注文 O,商品 I

WHERE

 O.商品番号 = I.商品番号;

5.2 WHERE句を使用した結合

□ WHERE句を使用した結合イメージ

注文テーブル

注文番号	商品番号	数量	金額	納期
1	1001	30	3600	2012/4/25
2	1003	20	2400	2012/4/19
3	1001	45	5400	

商品テーブル

商品番号	商品名	単価
1001	ミカン	120
1002	リンゴ	90
1003	バナナ	120



注文番号	商品番号	商品名	単価	数量	金額	納期
1	1001	ミカン	120	30	3600	2012/4/25
2	1003	バナナ	120	20	2400	2012/4/19
3	1001	ミカン	120	45	5400	

5.3 直積結合(CROSS JOIN)

基本構文

```
SELECT  
    [カラム名]  
FROM  
    [テーブル名1]  
CROSS JOIN  
    [テーブル名2];
```

- 直積結合(CROSS JOIN)
 - すべてのレコードの組み合わせを取り出す。
 - 直積演算とも呼ばれる。

5.3 直積結合 (CROSS JOIN)

□ CROSS JOINの例

- 注文テーブルに対して、商品テーブルを直積結合で結合する。

```
SELECT
    O.注文番号,O.商品番号,
    O.数量,O.金額,
    O.納期,
    I.商品番号,I.商品名,I.単価
FROM
    注文 O
CROSS JOIN
    商品 I;
```


5.3 直積結合 (CROSS JOIN)

□ CROSS JOINのイメージ

注文テーブル

注文番号	商品番号	数量	金額	納期
1	1001	30	3600	2012/4/25
2	1003	20	2400	2012/4/19
3	1001	45	5400	

商品テーブル

商品番号	商品名	単価
1001	ミカン	120
1002	リンゴ	90
1003	バナナ	120



注文番号	商品番号	数量	金額	納期	商品番号	商品名	単価
1	1001	30	3600	2012/4/25	1001	ミカン	120
1	1001	30	3600	2012/4/25	1002	リンゴ	90
1	1001	30	3600	2012/4/25	1003	バナナ	120
2	1003	20	2400	2012/4/19	1001	ミカン	120
2	1003	20	2400	2012/4/19	1002	リンゴ	90
2	1003	20	2400	2012/4/19	1003	バナナ	120
3	1001	45	5400		1001	ミカン	120
3	1001	45	5400		1002	リンゴ	90
3	1001	45	5400		1003	バナナ	120

5.4 内部結合 (INNER JOIN)

基本構文

```
SELECT  
    [カラム名]  
FROM  
    [テーブル名 1]  
INNER JOIN  
    [テーブル名 2]  
ON  
    [結合条件] ;
```

5.4 内部結合 (INNER JOIN)

□ 内部結合 (INNER JOIN)

- 結合条件に一致するレコードのみを取り出す。
- テーブルを結合するための条件を指定する。
- 1つ目のテーブル(テーブル名1)はFROM句に記述し、2つ目以降のテーブル(テーブル名2)は、「INNER JOIN 句」に記述する。
- 「ON句」に結合条件を記述する。
- FROM句に指定したテーブル名1に存在するレコードのみが対象。

5.4 内部結合 (INNER JOIN)

□ INNER JOINの例

- 注文テーブルに対して、商品テーブルを内部結合で結合する。

```
SELECT
    O.注文番号,O.商品番号,
    O.数量,O.金額,
    O.納期,
    I.商品番号,I.商品名,I.単価
FROM
    注文 O
INNER JOIN
    商品 I
ON
    O.商品番号 = I.商品番号;
```

5.4 内部結合 (INNER JOIN)

□ INNER JOINのイメージ

注文テーブル

注文番号	商品番号	数量	金額	納期
1	1001	30	3600	2012/4/25
2	1003	20	2400	2012/4/19
3	1001	45	5400	

商品テーブル

商品番号	商品名	単価
1001	ミカン	120
1002	リンゴ	90
1003	バナナ	120



注文番号	商品番号	数量	金額	納期	商品番号	商品名	単価
1	1001	30	3600	2012/4/25	1001	ミカン	120
2	1003	20	2400	2012/4/19	1003	バナナ	120
3	1001	45	5400		1001	ミカン	120

商品テーブルの商品番号が1002のデータは、結合条件に一致しないため、取り出されない。

5.5 外部結合 (OUTER JOIN)

基本構文

```
SELECT  
    [カラム名]  
FROM  
    [テーブル名1]  
LEFT OUTER JOIN  
RIGHT OUTER JOIN  
    [テーブル名2]  
ON  
    [結合条件] ;
```

5.5 外部結合 (OUTER JOIN)

□ 外部結合 (OUTER JOIN)

- テーブル名1またはテーブル名2の一方にしかないレコードも取り出す。
- どちらをテーブル名1に記述するかによって2種類の結合方法が存在する。
 - 左側外部結合 (LEFT OUTER JOIN)
テーブル名1を全て取り出す。
 - 右側外部結合 (RIGHT OUTER JOIN)
テーブル名2を全て取り出す。

※通常はLEFT OUTER JOINの使用が推奨されている。

5.5 外部結合 (OUTER JOIN)

□ LEFT OUTER JOINの例

- 商品テーブルに対して、注文テーブルを左側外部結合で結合する。

```
SELECT
  I.商品番号,I.商品名,I.単価,
  O.注文番号,O.商品番号,
  O.数量,O.金額,
  O.納期
FROM
  商品 I
LEFT OUTER JOIN
  注文 O
ON
  O.商品番号 = I.商品番号;
```


5.5 外部結合(OUTER JOIN)

□ LEFT OUTER JOINのイメージ

商品テーブル

商品番号	商品名	単価
1001	ミカン	120
1002	リンゴ	90
1003	バナナ	120

注文テーブル

注文番号	商品番号	数量	金額	納期
1	1001	30	3600	2012/4/25
2	1003	20	2400	2012/4/19
3	1001	45	5400	

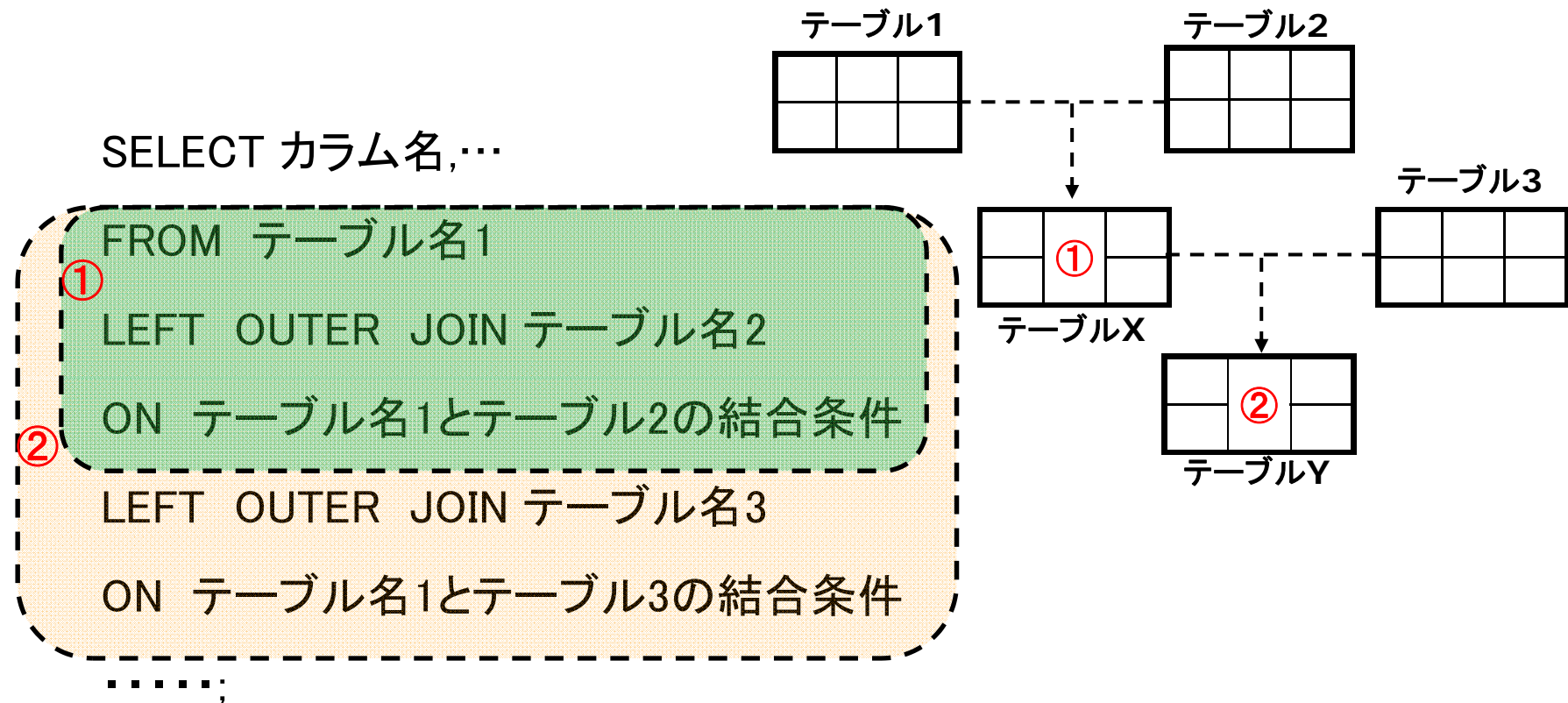
商品番号	商品名	単価	注文番号	商品番号	数量	金額	納期
1001	ミカン	120	1	1001	30	3600	2012/4/25
1003	バナナ	120	2	1003	20	2400	2012/4/19
1001	ミカン	120	3	1001	45	5400	
1002	リンゴ	90					

条件に合致するレコードが
注文テーブルに存在しない場合は、
全てNULLが取得される。

5.5 外部結合 (OUTER JOIN)

□ 左側外部結合 (LEFT OUTER JOIN) の推奨

- 3つ以上のテーブルを結合する場合、左側のテーブルを基準に結合していくため、左側外部結合を推奨とする。



5.6 自然結合 (NATURAL JOIN)

基本構文

```
SELECT  
    [カラム名]  
FROM  
    [テーブル名1]  
NATURAL JOIN  
    [テーブル名2] ;
```

5.6 自然結合 (NATURAL JOIN)

- 自然結合 (NATURAL JOIN) とは
 - 2つのテーブルで同じ名前で同じデータ型のカラムが存在すると自動的にテーブルを結合する。
 - 結合条件は不要。
 - カラムの値が一致するレコードのみを取り出す。
 - 結合するカラム名はテーブル名で修飾してはいけない。

5.6 自然結合 (NATURAL JOIN)

□ NATURAL JOINの例

- 注文テーブルに対して、商品テーブルを自然結合で結合する。

```
SELECT
    O.注文番号,商品番号,
    O.数量,O.金額,
    O.納期,
    I.商品名,I.単価
FROM
    注文 O
NATURAL JOIN
    商品 I;
```

5.6 自然結合 (NATURAL JOIN)

□ NATURAL JOINのイメージ

注文テーブル

注文番号	商品番号	数量	金額	納期
1	1001	30	3600	2012/4/25
2	1003	20	2400	2012/4/19
3	1001	45	5400	

商品テーブル

商品番号	商品名	単価
1001	ミカン	120
1002	リンゴ	90
1003	バナナ	120



注文番号	商品番号	数量	金額	納期	商品名	単価
1	1001	30	3600	2012/4/25	ミカン	120
2	1003	20	2400	2012/4/19	バナナ	120
3	1001	45	5400		ミカン	120

5.7 副問い合わせ

基本構文

```
SELECT  
    [取得したいカラム名]  
FROM  
    [テーブル名]  
WHERE  
    [条件 (副問い合わせ) ] ;
```

- 副問い合わせ(サブクエリ)とは
 - SELECT文(主問い合わせ)の中に埋め込まれたSELECT文のこと。
 - ある問い合わせの結果から取り出した値を利用して検索することができる。

5.7 副問い合わせ

□ 副問い合わせの考え方

SELECT 注文番号, 金額 FROM 注文
WHERE 金額 = (SELECT MAX(金額) FROM 注文);

④
③
②
①

①副問い合わせを実行(注文テーブルからMAX(金額)を求める)

注文テーブル

注文番号	商品番号	数量	金額	...
1	1001	30	3600	...
2	1003	20	2400	...
3	1001	45	5400	...

②データベースから注文テーブルを取り出す。

注文テーブル

注文番号	商品番号	数量	金額	...
1	1001	30	3600	...
2	1003	20	2400	...
3	1001	45	5400	...

③①の結果と一致するレコードを取り出す。

注文テーブル

注文番号	商品番号	数量	金額	...
1	1001	30	3600	...
2	1003	20	2400	...
3	1001	45	5400	...

④指定されたカラムを取り出す。

注文テーブル

注文番号	商品番号	数量	金額	...
1	1001	30	3600	...
2	1003	20	2400	...
3	1001	45	5400	...

5.7 副問い合わせ

- 副問い合わせの結果を比較する演算子
 - 条件の指定には＝、＞、＜などの比較演算子以外に以下の演算子がある。

演算子	意味
IN(副問い合わせ)	副問い合わせをした結果のレコードのいずれかの値と等しい
NOT IN(副問い合わせ)	副問い合わせをした結果のレコードすべてと等しくない ※副問い合わせが戻す複数の値の中に1つでもNULL値が含まれていると、他の値がNULL以外であったとしても主問い合わせが値を戻さない
EXISTS(副問い合わせ)	相関副問い合わせ(*) 副問い合わせをした結果、レコードが存在すれば真(TRUE)を返す
NOT EXISTS(副問い合わせ)	相関副問い合わせ(*) 副問い合わせした結果、レコードが存在しなければ真(TRUE)を返す

相関副問い合わせ:

主問い合わせで取り出される1レコードごとに、副問い合わせを実行する。

5.7 副問い合わせ

□ INを使用した例

- 注文テーブルから「バナナ」を注文した注文番号を取得する。

注文テーブル

注文番号	商品番号	数量	金額	納期
1	1001	30	3600	2012/4/25
2	1003	20	2400	2012/4/19
3	1001	45	5400	

商品テーブル

商品番号	商品名	単価
1001	ミカン	120
1002	リンゴ	90
1003	バナナ	120

```
SELECT      O.注文番号
FROM
WHERE      O.商品番号 IN (
                        SELECT
                        FROM
                        WHERE
                        )
;
```

I.商品番号
商品 I
I.商品名 = 'バナナ'



注文番号
2

5.7 副問い合わせ

□ EXISTSを使用した例

- 前ページのSQLをEXISTSで書き換える。

```
SELECT      O.注文番号
FROM        注文 O
WHERE       EXISTS (
                SELECT      I.商品番号
                FROM        商品 I
                WHERE       I.商品名 = 'バナナ'
                AND         I.商品番号 = O.商品番号
            )
;
```



注文番号
2

5.7 副問い合わせ

- 副問い合わせを使用できる箇所
 - SELECT文、SET句など
 - ・ 副問い合わせの結果をカラムとして扱う。
 - WHERE句、HAVING句など
 - ・ 副問い合わせの結果を検索条件として扱う。
 - FROM句など
 - ・ 副問い合わせの結果をテーブルとして扱う。

5.8 副問い合わせを使用したUPDATE

基本構文

```
UPDATE  
    [テーブル名]  
SET  
    [カラム名] = (副問い合わせ)  
WHERE  
    [書き換えたいレコードの条件] ;
```

- 副問い合わせを使用したUPDATE
 - SET句で更新するカラムの値を指定する代わりに、副問い合わせを使用する。

5.8 副問い合わせを使用したUPDATE

□ 副問い合わせを使用したUPDATEの例

- 注文テーブルの注文番号が3の納期をMAX(納期)に更新する。

注文テーブル

注文番号	商品番号	数量	金額	納期
1	1001	30	3600	2012/4/25
2	1003	20	2400	2012/4/19
3	1001	45	5400	

```
UPDATE 注文 O
SET     O.納期 = (
                SELECT MAX(O.納期)
                FROM 注文 O
            )
WHERE   O.注文番号 = 3;
```

注文テーブル

注文番号	商品番号	数量	金額	納期
1	1001	30	3600	2012/4/25
2	1003	20	2400	2012/4/19
3	1001	45	5400	2012/4/25

5.9 副問い合わせを使用したINSERT

基本構文

```
INSERT INTO  
    [テーブル名]  
    (カラム名,カラム名,...)  
VALUES  
    (副問い合わせ,カラム名... );
```

- 副問い合わせを使用したINSERT
 - VALUES句で値を指定する代わりに、副問い合わせを使用する。

5.9 副問い合わせを使用したINSERT

- 副問い合わせを使用したINSERTの例
 - 注文テーブルに以下のレコードを追加する。

注文番号	商品番号	数量	金額	納期
MAX(注文番号)+1	1002	25	2250	2012/5/25

```
INSERT INTO 注文 O
      (O.注文番号,O.商品番号,O.数量,
      O.金額,O.納期)
VALUES
(
      (SELECT MAX(O.注文番号)+1 FROM 注文 O),
      1002,25,2250,'20120525'
);
```

注文テーブル

注文番号	商品番号	数量	金額	納期
1	1001	30	3600	2012/4/25
2	1003	20	2400	2012/4/19
3	1001	45	5400	
4	1002	25	2250	2012/5/25

5.10 副問い合わせを使用したDELETE

基本構文

```
DELETE FROM [対象とするテーブル名]  
WHERE (副問い合わせ);
```

- 副問い合わせを使用したDELETE
 - WHERE句で削除する条件を指定する代わりに、副問い合わせを使用する。

5.10 副問い合わせを使用したDELETE

- 副問い合わせを使用したDELETEの例
 - 注文テーブルのMIN(数量)のレコードを削除する。

注文テーブル

注文番号	商品番号	数量	金額	納期
1	1001	30	3600	2012/4/25
2	1003	20	2400	2012/4/19
3	1001	45	5400	

DELETE FROM
WHERE

注文 O
O.金額 =
(SELECT MIN(O.金額)
FROM 注文 O);



注文テーブル

注文番号	商品番号	数量	金額	納期
1	1001	30	3600	2012/4/25
3	1001	45	5400	

5.11 和集合 (UNION)

基本構文

```
SELECT (カラム名) FROM [テーブル名1]  
UNION/UNION ALL  
SELECT (カラム名) FROM [テーブル名2];
```

□ 和集合 (UNION) とは

- ある問い合わせ結果と、別の問い合わせ結果を1つの結果に結合する。
- それぞれの問い合わせ結果は、同じ構造でなければならない(構造が異なるカラムが存在してはいけない)。
- UNIONは、重複を除くレコードを取得する。
- UNION ALLは、重複を含むすべてのレコードを取得する。

5.11 和集合 (UNION)

□ 和集合 (UNION) の例

- 資格Aテーブルから資格Aを保有している社員名を、資格Bテーブルから資格Bを保有している社員を取得し、どちらかの資格を取得している社員名を取得する。重複は排除する。

資格Aテーブル

社員番号	社員名	取得日
5001	鈴木	2009/2/20
5009	田中	2011/5/10
5023	加藤	2000/10/12

資格Bテーブル

社員番号	社員名	取得日
5041	佐藤	2007/7/20
5023	加藤	2009/12/10
5001	鈴木	2011/3/15

```
SELECT 社員名 FROM 資格A
UNION
SELECT 社員名 FROM 資格B;
```



社員名
鈴木
田中
加藤
佐藤



Memo

6. DDL 【Data Definition Language】



- ・概要

DML 【Data Definition Language】の基本を学習する。

- ・学習内容

- テーブルの作成と削除
- 制約
- インデックスの作成と削除
- ビューの作成と削除
- シーケンスの作成と削除

6.1 テーブルの作成と削除

□ ネーミングルール

- テーブルやカラム、インデックス、ビューなどをネーミングする際には以下のルールがある。

- 文字で開始する必要がある(数字で開始してはいけない)
- 長さ30バイト以下
- 英数字、記号(\$ __ #)が使用可能
- 予約語(*)は使用できない。 *SELECTやTABLEなどSQLで使用しているキーワード
- 一意なテーブル名、同一テーブル内では一意なカラム名でなければならない。
- 大文字、小文字は区別されない。
- 日本語環境においては、漢字、ひらがな、カタカナの使用も可能

6.1 テーブルの作成と削除

- テーブルの作成
 - データベースにテーブルを作成する。



6.1 テーブルの作成と削除

基本構文

```
CREATE TABLE [テーブル名]  
([カラム定義], [カラム定義]...);
```

□ カラム定義

■ カラム名

- カラムを識別するための名前

■ データ型

- カラムに格納するデータの型

■ データ長

- カラムに格納できるデータの長さ

6.1 テーブルの作成と削除

□ プライマリキーの定義

基本構文

```
CREATE TABLE [テーブル名] (  
  [カラム定義] CONSTRAINT [制約名] PRIMARY KEY,  
  [カラム定義]...);
```

複合プライマリキーの場合

```
CREATE TABLE [テーブル名](  
  [カラム定義],  
  [カラム定義],  
  [カラム定義]  
  CONSTRAINT [制約名]  
  PRIMARY KEY(カラム名1,カラム名2));
```

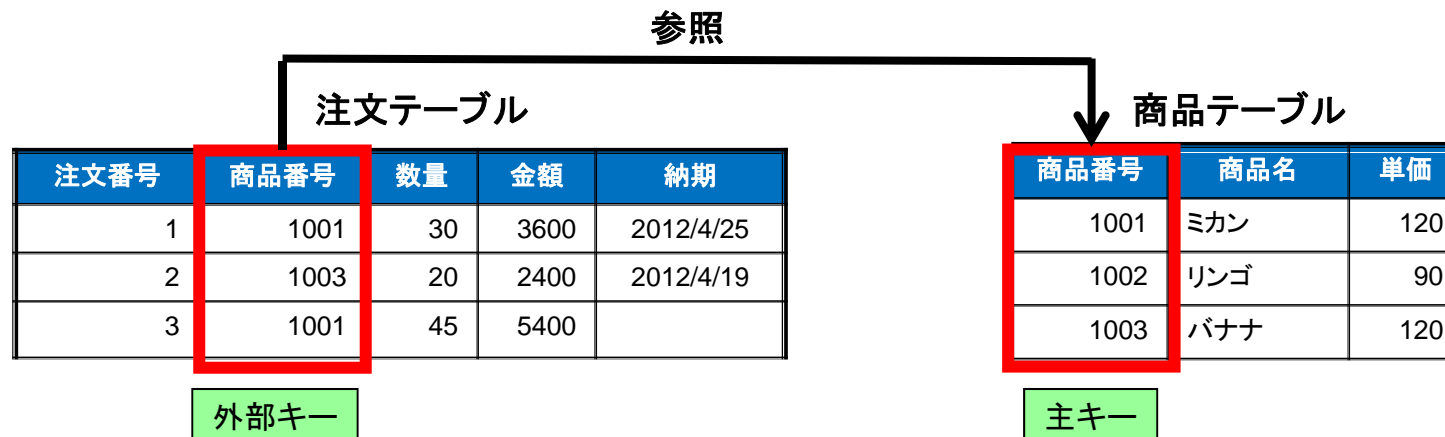
6.1 テーブルの作成と削除

□ 外部キー (FOREIGN KEY)

- 他のテーブルのプライマリキーを参照しているカラム
- 外部キーの値は参照しているキーの値または、NULLでなければならない。

□ 外部キーの例

- 注文テーブルの商品番号は外部キー
- 商品テーブルのプライマリキーである商品番号を参照している。



6.1 テーブルの作成と削除

□ 外部キーの定義

基本構文

```
CREATE TABLE [テーブル名] (  
  [カラム定義] CONSTRAINT [制約名] REFERENCES  
    参照するテーブル名(参照するカラム名),  
  [カラム定義]...);
```

6.1 テーブルの作成と削除

□ CREATE TABLE文の例

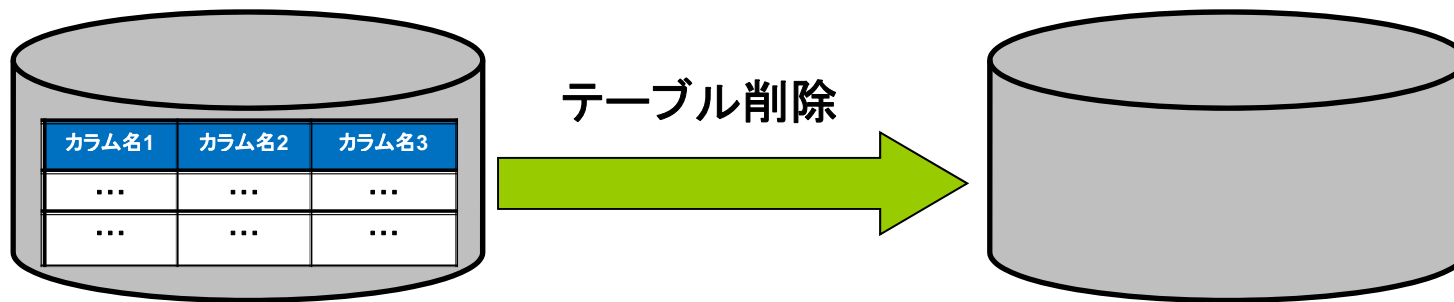
注文テーブルのCREATE TABLE文

```
CREATE TABLE 注文
( 注文番号 NUMERIC(3) CONSTRAINT
                                PK_注文番号 PRIMARY KEY,
  商品番号 NUMERIC(4),
  数量     NUMERIC(3),
  金額     NUMERIC(7),
  納期     DATE
);
```

6.1 テーブルの作成と削除

□ テーブルの削除

- データベースからテーブルを削除する。



6.1 テーブルの作成と削除

基本構文

```
DROP TABLE [テーブル名] ;
```

□ DROP TABLE文の例

```
DROP TABLE 注文;
```

6.2 制約

□ 制約とは

- テーブルに無効なデータを挿入させないための仕組み。

□ 制約の種類

制約	説明
NOT NULL	NULL値の入力を許さない(入力必須カラム)
UNIQUE	値の重複を許さない(一意カラム)
PRIMARY KEY	テーブル内のレコードを識別する、プライマリキー(NOT NULLでなおかつUNIQUE)
FOREIGN KEY	外部キー制約、参照整合性とも呼ばれる。 参照先に存在する値またはNULLでなければいけない。 (NOT NULL制約が同時に定義されていない場合)
CHECK	指定した条件を満たす値でなければいけない (在庫は0以上の値でなければならないなど)

6.2 制約

□ 制約の作成(1)

- テーブルの作成時に制約を作成する。

基本構文

NOT NULL/UNIQUE/PRIMARY KEYの場合

```
CREATE TABLE [テーブル名] (  
    [カラム定義] CONSTRAINT [制約名]  
        NOT NULL/UNIQUE/PRIMARY KEY,  
    [カラム定義] ,  
    ...);
```

6.2 制約

□ 制約の作成(2)

基本構文

複合PRIMARY KEYの場合

```
CREATE TABLE [テーブル名](  
    [カラム定義],  
    ...,  
    CONSTRAINT [制約名]  
        PRIMARY KEY(カラム名1,カラム名2));
```

複合プライマリキーは
テーブル定義の末尾に記述

6.2 制約

□ 制約の作成(3)

基本構文

FOREIGN KEYの場合

```
CREATE TABLE [テーブル名] (  
    [カラム定義]    CONSTRAINT [制約名]  
        REFERENCES 参照テーブル名(参照カラム名),  
    [カラム定義] ,  
    ...);
```

6.2 制約

□ CREATE TABLE文での制約の作成例

```
CREATE TABLE 注文 (  
    ジャンルID  VARCHAR(3)  
        CONSTRAINT NN_ジャンルID NOT NULL,  
    商品番号    VARCHAR(6)  
        CONSTRAINT NN_商品番号 NOT NULL,  
    商品名      VARCHAR(40),  
    ユーザID    NUMERIC(3)  
        CONSTRAINT FK_ユーザID  
        REFERENCES ユーザ(ユーザID),  
    単価        NUMERIC(6),  
    注文日     DATE,  
    CONSTRAINT PK_注文  
    PRIMARY KEY(ジャンルID, 商品番号)  
);
```

6.2 制約

□ 制約の追加

- すでに作成したテーブルに制約を追加する。

基本構文

NOT NULL制約以外

```
ALTER TABLE [テーブル名]  
ADD CONSTRAINT [制約名]  
UNIQUE (カラム名) /  
PRIMARY KEY (カラム名) /  
FOREIGN KEY (カラム名)  
    REFERENCES [参照テーブル名] (参照カラム) ;
```

NOT NULL制約

```
ALTER TABLE [テーブル名]  
MODIFY [カラム名] NOT NULL ;
```

6.2 制約

- 制約の削除
 - 制約を削除する。

基本構文

```
ALTER TABLE [テーブル名]  
DROP CONSTRAINT [制約名] ;
```

6.3 インデックスの作成と削除

□ インデックスとは

- データの検索速度を向上させるために、どのレコードがどこにあるかを示したもの。

```
SELECT 注文番号, 商品番号, 数量, 金額, 納期  
FROM 注文  
WHERE 商品番号 = 1003;
```

商品番号インデックス

商品番号	格納場所
1001	XXXX
1002	XXXX
1003	XXXX

注文テーブル					
注文番号	商品番号	数量	金額	納期	
1	1001	30	3600	2012/4/25	
2	1003	20	2400	2012/4/19	名3
3	1001	45	5400		
...

6.3 インデックスの作成と削除

□ インデックス作成の方法

■ 自動作成

- PRIMARY KEY制約、UNIQUE制約を定義することで、そのカラムに対して「一意索引」を自動的に作成する。
- 「一意索引」とは、キー値が一意でなければならない索引のこと。
- 一意索引を使用することで、プライマリキーや一意キーの値の一意性を実現する。

■ 手動作成

- CREATE INDEX文を使用し、「非一意索引」を作成する。
- 「非一意索引」の場合は、キー値に同じ値を持つことができる。
- 例えば、外部キーのカラムに非一意索引を作成することで、結合操作を高速に行うことができる。
- CREATE INDEX文では一意索引も作成できるが、カラム値の一意性はPRIMARY KEY制約、またはUNIQUE制約を明示的に定義することが推奨される。

6.3 インデックスの作成と削除

□ インデックス作成のガイドライン

■ インデックスが有効なカラムの条件

- カラムがWHERE句の条件や結合の条件に頻繁に指定される。
- カラムの値の種類が多い。
- 大量データから少量データ(全体の2～4パーセント)を検索することが多いカラム。
- NULL値が多く含まれていて、NULL以外の値を検索するカラム。

6.3 インデックスの作成と削除

□ インデックスの作成

- テーブル内のカラムに対してインデックスを作成する。

基本構文

```
CREATE INDEX [INDEX名]  
ON テーブル名 (カラム名);
```

6.3 インデックスの作成と削除

□ インデックスの削除

- カラムに作成されたインデックスを削除する。

基本構文

```
DROP INDEX [INDEX名] ;
```

6.4 ビューの作成と削除

□ ビューとは

- ある参照系SQLの実行結果を見かけ上ひとつのテーブルとみなす。
- 実際にデータを保持することのない仮想的なテーブル。
- 基本的には検索用として使用する。

□ ビューのメリット

- ユーザによって、アクセスできるテーブルのカラムやレコードを限定できる。
- 複数のテーブルを結合したデータを表現できる。

6.4 ビューの作成と削除

- ビューの作成
 - テーブルからビューを作成する。

基本構文

```
CREATE VIEW [ビュー名] AS 副問い合わせ;
```

6.4 ビューの作成と削除

□ ビューの定義例

注文テーブルから、商品名が「ミカン」のレコードを取り出した結果を仮想的なテーブルとする注文ビューを作成する。

```
CREATE VIEW V_注文 AS
SELECT 0. 注文番号, 0. 商品番号, 0. 数量,
       0. 金額, 0. 納期
FROM 注文 0
WHERE 0. 商品番号 IN (
    SELECT 1. 商品番号 FROM 商品 1
    WHERE 1. 商品名 = 'ミカン'
)
;
```

ビューに定義したいSQL

6.4 ビューの作成と削除

□ ビューの利用例

- 定義済みのビューを用いてSELECT文を実行することができる。

```
SELECT  
    注文番号, 商品番号, 数量, 金額, 納期  
FROM V_注文;
```

6.4 ビューの作成と削除

□ ビューの削除

- ビューを削除する。
- ビューは仮想的なテーブルのため、ビューを削除しても元のテーブルに影響はない。

基本構文

DROP VIEW [ビュー名] ;

6.5 シーケンスの作成と削除

□ シーケンスとは

- 順序のこと。
- 一意な整数値を生成する。
- プライマリキーなどに利用される。
- 現在値を取得するのに別途SQLを利用する必要がある。
- 初期値を変更することはできない。初期値を変更する場合は、いったんシーケンスを削除して再度作成する。

6.5 シーケンスの作成と削除

- シーケンスの作成
 - シーケンスを作成する。

基本構文

```
CREATE SEQUENCE [シーケンス名]  
START WITH [初期値]  
INCREMENT BY [増分]  
MAXVALUE [最大値] ;
```

6.5 シーケンスの作成と削除

□ シーケンスの定義例

- 初期値が5で1ずつ増加し、最大100までのシーケンスを作成する。

```
CREATE SEQUENCE SEQ_注文  
START WITH 5  
INCREMENT BY 1  
MAXVALUE 100;
```

6.5 シーケンスの作成と削除

□ シーケンスの利用

- 作成したシーケンスから値を生成する。

基本構文

```
nextval('シーケンス名')
```

- 現在のシーケンスの値を取得する。

基本構文

```
currval('シーケンス名')
```

6.5 シーケンスの作成と削除

□ シーケンスの利用例

- シーケンスを利用してプライマリキーを生成し、注文テーブルにレコードを追加する。

```
INSERT INTO 注文  
VALUES  
    (nextval('SEQ_注文'), , 1002, 10, 900, ' 20120510' );
```

- 現在のシーケンスを取得する

```
SELECT currval('SEQ_注文') ;
```

6.5 シーケンスの作成と削除

- シーケンスの削除
 - シーケンスを削除する。

基本構文

```
DROP SEQUENCE [シーケンス名] ;
```



Memo

7. DCL 【Data Control Language】

- ・概要

DCL 【Data Control Language】の基本を学習する。

- ・学習内容

- 権限
- トランザクション
- ロック

7.1 権限

□ 権限

- テーブルやビューなどを利用するための権利
- ユーザごとに権限を設定する。

□ 権限の種類

- CREATE TABLE
- CREATE USER
- CREATE VIEW
- INSERT
- SELECT
- UPDATE
- DELETE など

7.1 権限

- 権限の付与 (GRANT)
 - 権限を付与する。

基本構文

```
GRANT 権限 TO ユーザー名 ;
```

7.1 権限

- 権限の削除 (REVOKE)
 - 権限を削除する。

基本構文

```
REVOKE 権限 FROM ユーザー名 ;
```

7.2 トランザクション

□ トランザクションとは

- 関連のある複数の操作をひとつにまとめたもの
- データベースを更新する単位
- 操作を確定することをコミット、破棄することをロールバックと呼ぶ。
- 操作の確定が行われるまでは、データベースに値が反映はされない。
- 一連の操作の開始から1つのトランザクションが開始され、コミットもしくはロールバックが実行されると、そのトランザクションは終了する。

7.2 トランザクション

□ トランザクションが必要な例

■ 銀行口座の振り込み処理

- Aさんの口座から、Bさんの口座に30万円振り込む。

口座テーブル

口座番号	顧客名	残高
0010001	A	120万円
0010002	B	100万円
0010003	C	94万円
0100001	D	1240万円
1009421	E	1000万円
1101493	F	3890万円
0010020	G	30万円

トランザクション

- ① Aさんの口座の残高を更新
 $120万円 - 30万 = 90万円$
- ② Bさんの口座の残高更新
 $100万円 + 30万 = 130万円$

「振込」業務が成立するには①と②の両方の処理が正しく完了していなければならない。

7.2 トランザクション

□ 更新の確定 (COMMIT)

- トランザクション内のすべての変更を確定する。
- 次のトランザクションの開始でもある。
- 一度COMMITしたトランザクションは取り消し (ROLLBACK) はできない。

口座テーブル

口座番号	顧客名	残高
0010001	A	90万円
0010002	B	130万円
0010003	C	94万円
0100001	D	1240万円
1009421	E	1000万円
1101493	F	3890万円
0010020	G	30万円

トランザクション

- ① Aさんの口座の残高を更新
 $120\text{万円} - 30\text{万} = 90\text{万円}$
- ② Bさんの口座の残高更新
 $100\text{万円} + 30\text{万} = 130\text{万円}$

COMMIT

①と②の両方の処理が正しく完了。
「振込」業務が成立。

7.2 トランザクション

□ 更新の取り消し(ROLLBACK)

- トランザクションが行った更新の取り消しを行う。
- トランザクションを開始したときの状態に戻す。

口座テーブル

口座番号	顧客名	残高
0010001	A	90万円
0010002	B	100万円
0010003	C	94万円
0100001	D	1240万円
1009421	E	1000万円
1101493	F	3890万円
0010020	G	30万円

トランザクション

① Aさんの口座の残高を更新
 $120万円 - 30万円 = 90万円$

② Bさんの口座の残高を更新
 $106万円 - 30万円 = 76万円$

失敗！

ROLLBACK

「振込」が失敗。
トランザクションが行った更新の
取り消しを行う。

7.2 トランザクション

□ 暗黙のコミット

- 次のような場合は自動COMMITが発生し、トランザクションが終了する。
 - DDL文(CREATE、ALTER、DROPなどの)発行
 - DCL文(GRANT、REVOKEなど)の発行

※暗黙のCOMMITによって終了されたトランザクションはROLLBACKできない。

6.2 トランザクション

□ トランザクションの例

BEGIN; ← 同時に新しいトランザクションが開始される。

INSERT INTO ...;

UPDATE ...;

COMMIT; ← 更新が確定し、INSERT文とUPDATE文の結果が反映される。
トランザクションが終了する。

BEGIN; ← 同時に新しいトランザクションが開始される。

DELETE FROM ...; ← DELETEが同一トランザクションであれば、削除が
反映される。
SELECT ...; ← 同一トランザクションでない場合は、削除は反映さない。

ROLLBACK; ← 更新を取り消し、DELETE文の結果は反映されず破棄される。
トランザクションが終了し、同時に新しいトランザクションが開始される。

SELECT ...; ← DELETEは無効になり、データは削除されていない。

7.3 ロック

□ ロック

- 同一のデータに対して、複数のトランザクションが同時に書き込むことを防止する仕組み。
- ロックには、他のトランザクションに対する読み書きを一切禁止する「排他ロック」と、読み出しだけを許可する「共有ロック」がある。



7.3 ロック

□ テーブルロック

- テーブル全体をロックする。
- DML操作中にテーブルに対してアクセスさせないようにするために使用される。

口座テーブル

口座番号	顧客名	残高
0010001	A	90万円
0010002	B	100万円
0010003	C	94万円
0100001	D	1240万円
1009421	E	1000万円
1101493	F	3890万円
0010020	G	30万円



7.3 ロック

□ レコードロック

- 選択されたレコードをロックする。
- 2つのトランザクションによる同じレコードの変更を防ぐ目的で使われる。

口座テーブル

口座番号	顧客名	残高
0010001	A	90万円
0010002	B	100万円
0010003	C	94万円
0100001	D	1240万円
1009421	E	1000万円
1101493	F	3890万円
0010020	G	30万円



7.3 ロック

□ 暗黙的なロック

- 基本的にロックは完全に自動化されているため、ユーザーが明示的にロックする必要はない。
- INSERT, UPDATE, DELETEは、レコードロックを発生させる。
- WHERE句の条件に一致したレコードをロックする。

7.3 ロック

□ 明示的なロック

- ユーザが明示的にロックすることもできる。
- 複数のユーザから更新される可能性のあるテーブルまたはレコードを利用する場合に行う。
- 明示的なロックは、トランザクションがCOMMITまたはROLLBACKされたときに解除される。

7.3 ロック

- 明示的なテーブルロック
 - テーブルをロックする。

基本構文

```
LOCK TABLE [テーブル名] ;
```

7.3 ロック

- 明示的なレコードロック
 - レコードをロックする。

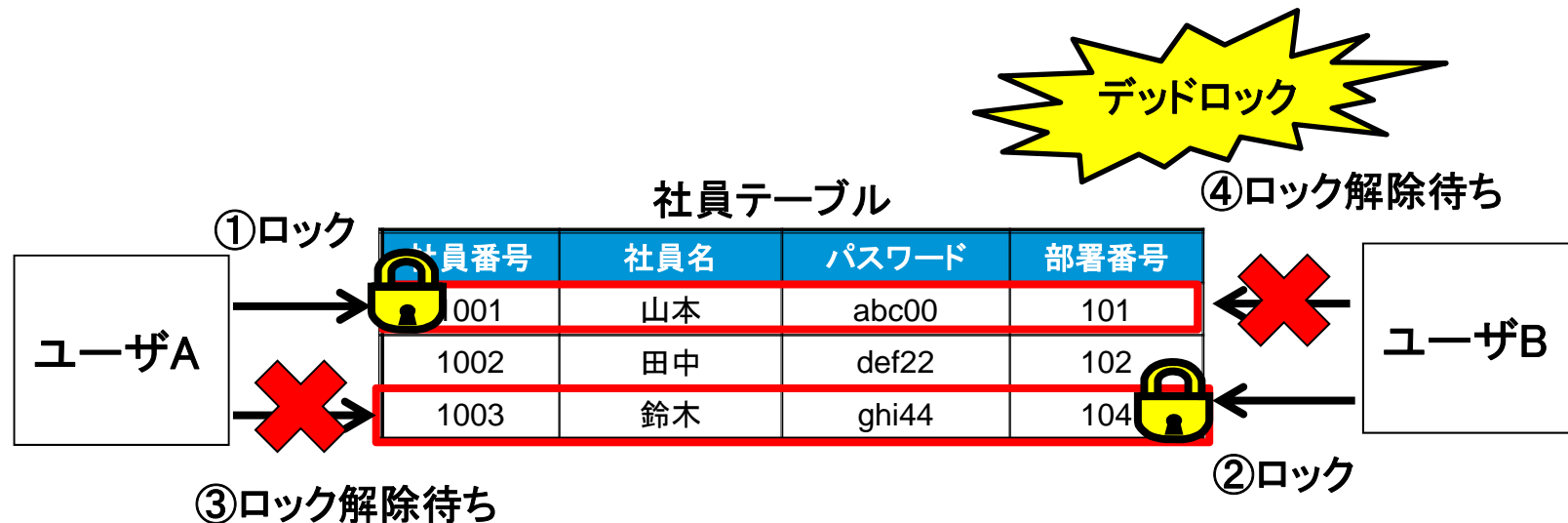
基本構文

```
SELECT [カラム名] FROM [テーブル名]  
WHERE [取得したいレコードの条件]  
FOR UPDATE ;
```


7.3 ロック

□ デッドロック

- 2人以上のユーザがお互いにロックを掛け合い、ロックの解除待ちになってしまう状態のこと。
- デッドロック状況を自動的に検出し、デッドロック中のトランザクションの1つをROLLBACKして、デッドロックを解消する。





Memo

8. その他

- ・概要
その他重要な技術要素について学習する。
- ・学習内容
 - 行番号
 - ストアドプロシージャ
 - トリガー

7.1 行番号

□ 行番号

- 問合せによって戻される各レコードの番号を戻す。

□ 行番号の利用例

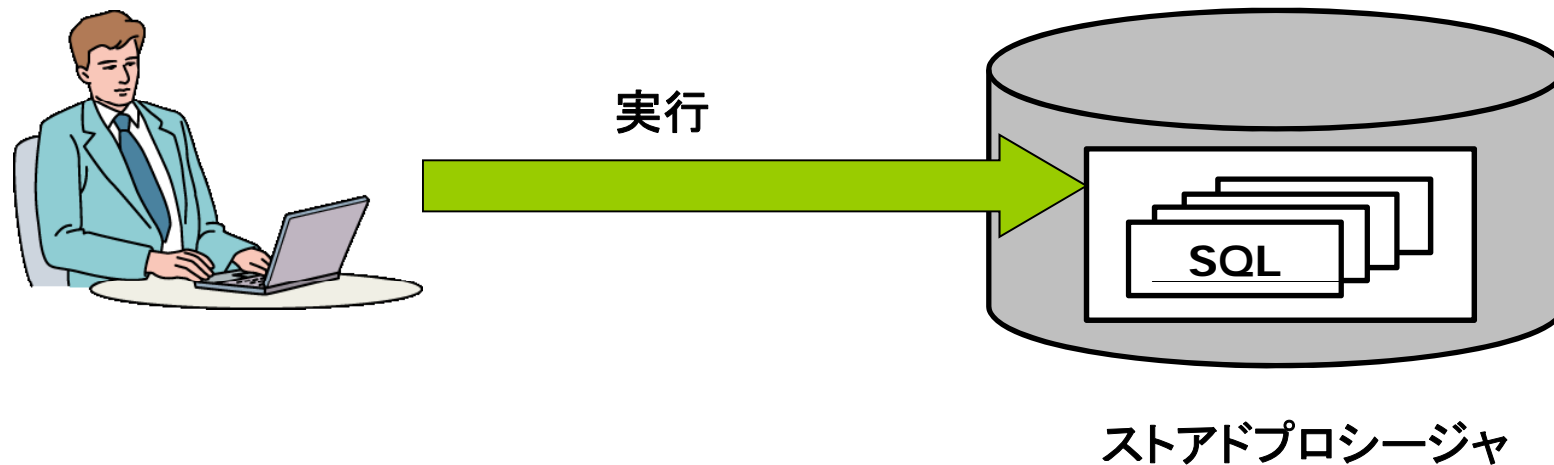
- 問合せ結果にレコード番号を付加

```
SELECT row_number() over(),I.商品番号,I.商品名,I.単価  
FROM 商品 I ;
```

8.2 ストアドプロシージャ

□ ストアドプロシージャ

- 一連のSQL文からなる手続きをデータベースに格納しておき、ユーザからの呼び出しで実行させる仕組み。
- あらかじめデータベースに登録されているため、ネットワーク上の通信を削減でき、処理の高速化を図ることができる。



8.3 トリガー

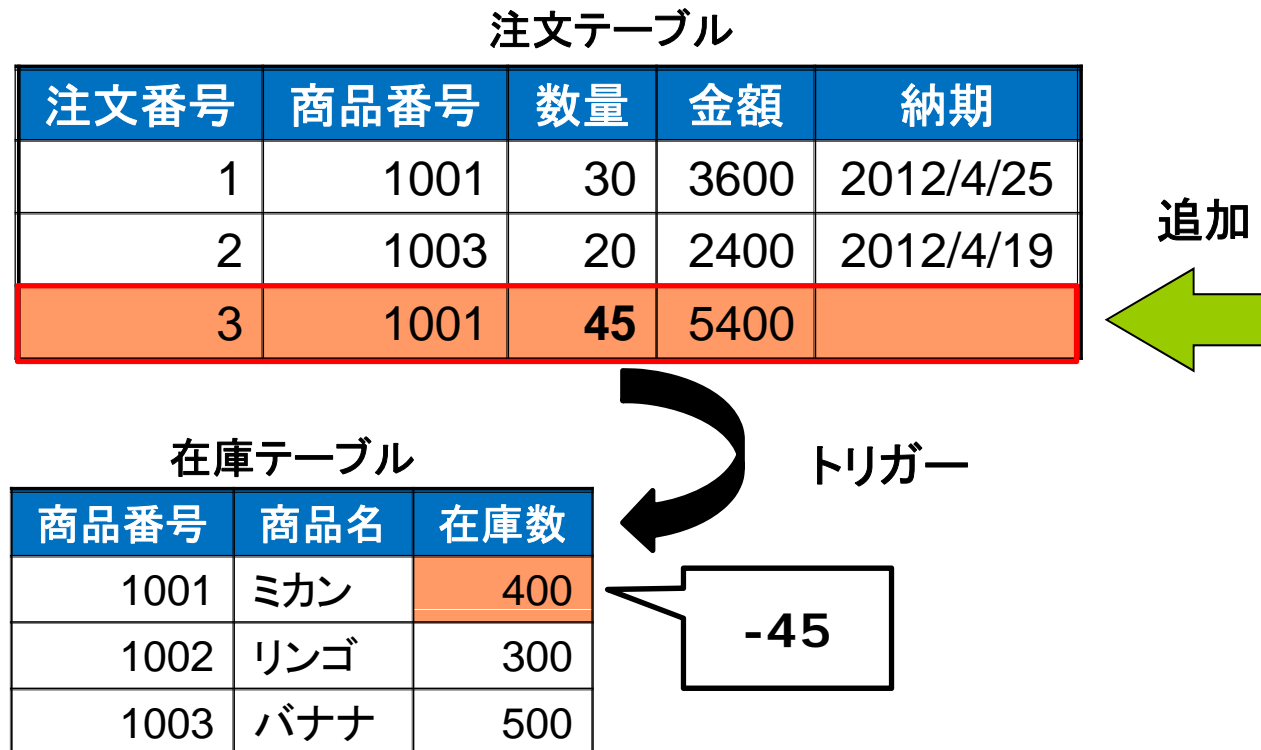
□ トリガー

- テーブルに対して追加、変更、削除などの特定の操作が発生したときに、自動で実行されるストアドプロシージャのこと。
- トリガーは常にテーブルを監視し、テーブルに対して操作が行われると、あらかじめ登録した処理を実行する。
- トリガーを使用すると、データの不整合を防ぐことができる。

8.3 トリガー

□ トリガーの例

- 注文テーブルにデータが追加されたら、在庫テーブルの在庫数を受注個数分マイナスする。





Memo
