『これだけは知っておきたいデータベースの常識』（藤本壱　技術評論社）より

第3章

リレーショナルデータベースをコントロールする「SQL」

現在のリレーショナルデータベースでは、データにアクセスする際に、「SQL」という統一的な言語を使うことが多いです。従って、リレーショナルデータベースを実際に使う上では、SQL の知識が必要になります。

そこで本章では、SQL の基本的な書き方を解説します。データの読み込み、条件設定、集計、またデータの保存、削除など、様々な処理を SQL で行う方法を紹介します。

3.1 SQLの概要

SQLの実際の書き方に入る前に、まずSQLの概要についてお話しします。

SQLの概要

リレーショナルデータベースのデータにアクセスするには、**リレーショナルデータベース**のシステム（以後「**RDBMS**」（Relational Database Management System）と略）に対して、「○○の条件を満たすデータを取り出したい」といったことを命令する必要があります。

現在の大半のRDBMSでは、「**SQL**」という統一的な言語を使って、命令を出すようになっています。例えば、「address」というテーブルのすべてのデータを読み込みたい場合は、RDBMSに対して次のようなSQLで命令を出します。

```
select * from address
```

プログラムを書くための言語は、難しいことが多いです。一方のSQLは、主に「データを取り出す」といった大まかな操作を行うための命令で、細かな処理は他のプログラム言語と組み合わせて行うことが一般的です。そのため、基本的な処理であれば、書き方はそれほど難しくありません。

SQLの生い立ち

SQLの原型となったのは、「**SEQUEL**」（Structured English Query Languageの略）という言語です。SEQUELは、IBMの「**System R**」というRDBMSで使われていた言語です。

RDBMSの初期のころは、それぞれのRDBMSが独自言語でデータベースを操作していました。その後、ANSI[用語]やISO[用語]によって、RDBMSを操作するための言語が、「SQL」として規格化されていきました。本書執筆時点では、2003年に制定された「**SQL:2003**」が最新の規格です。

もっとも、RDBMSごとに仕様や機能に違いがありますので、現在のRDBMSでも、標準のSQLに完全に従っているわけではありません。RDBMSごとに、様々な独自構文があります。

DML／DDL／DCL

SQLには多くの命令がありますが、大きく分けて「**DML**」「**DDL**」「**DCL**」の3つのグループに分けることができます。

DML

DMLは「**データ操作言語**」(Data Manipulation Language)の略で、その名前の通り、行の読み込みなどを行う命令です。**表3.1**のような命令があります。RDBMSを操作する際には、DMLを使った操作を行うことが、かなりの割合を占めます。

表3.1 DMLに属する命令

命令	内容
select	行を読み込む
insert	新たな行を追加する
update	既存の行を編集する
delete	行を削除する

DDL

DDLは「**データ定義言語**」(Data Definition Language)の略で、データベース全体に関する作成／削除／変更等を行う命令群です。テーブルの作成などを行う命令が、DDLに含まれます(**表3.2**)。

なお、**表3.2**の中に「**ユーザ**」という用語が出ていますが、これについては第4章で解説します。

注 ANSIとISO

ANSIは「American National Standards Institute」(米国規格協会)の略で、アメリカの工業分野の標準化団体です。また、ISOは「International Organization for Standardization」(国際標準化機構)の略で、国際的な工業分野の標準化団体です。

表 3.2　DDL に属する命令

命令	内容
create table	テーブルを作成する
alter table	テーブルの構造を変更する
drop table	テーブルを削除する
create user	ユーザを削除する
drop user	ユーザを削除する

DCL

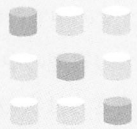
DCL は「**データ制御言語**」(Data Control Language) の略で、データベースの動作の制御に関係する命令群です。**表 3.3** のような命令があります。ただ、DCL は RDBMS の機能に依存する命令が多く、RDBMS による違いが大きいです。

なお、**表 3.3** の中に「**権限**」「**トランザクション**」「**コミット**」「**ロールバック**」といった用語が出ていますが、これらは第 4 章で解説します。

表 3.3　DCL に属する命令

命令	内容
grant	ユーザに権限を与える
revoke	ユーザの権限を削除する
begin	トランザクションを開始する
commit	トランザクションをコミットする
rollback	トランザクションをロールバックする

3.2 行を取り出す



SQL を使った操作の中では、「**行を取り出す**」という操作の使用頻度が最も高いです。様々な方法で行を取り出すことができますが、その中で特に基本的な書き方を紹介します。

select 命令でテーブルの行をすべて取り出す

行の取り出し方には、いろいろな手法があります。単純に行を取り出すだけでなく、条件を指定したり、複数のテーブルを結合したり、また行をグループ化して集計したりすることが考えられます。

SQL では、これらの行の取り出しを、すべて「**select**」という 1 つの命令で行うようになっています。select に続けて書く内容によって、様々な取り出し方ができるようになっています。

select 命令の最も単純な使い方は、1 つのテーブルから、すべての行（のすべての列）を取り出すことです。これは次のように書きます。「*」が「**すべての列**」を表します。

```
select * from テーブル名
```

例えば、「address」という名前のテーブルからすべての行を取り出すには、次のように書きます。

```
select * from address
```

射影を行う（テーブルから一部の列を取り出す）

リレーショナルデータベースの特徴として、データを取り出す際に、「**選択**」「**射影**」「**結合**」の 3 つの処理ができることがあります（43 ページ参照）。これらも、すべて

select 文の中で行います。

3つの処理の中では、**射影**（テーブルから一部の列を取り出す）が最も簡単です。その書き方は次のようになります。select の後に、**取り出したい列の名前**をカンマで区切って並べます。

```
select 列名 , 列名 , …… from テーブル名
```

例えば、「address」という名前のテーブルから、「name」と「phone」という列だけを取り出すには、次のように書きます。

```
select name, phone from address
```

◆図 3.1　「select name, phone from address」の select 文の動作

addressテーブル

name	pref	address1	phone	sex	age
山田太郎	東京都	新宿区市谷左内町XX-XX	03-1234-XXXX	男	32
鈴木花子	神奈川県	横浜市西区中央XX-XX	045-234-XXXX	女	28
高橋次郎	埼玉県	さいたま市浦和区常盤XX-XX	048-345-XXXX	男	41
斎藤智子	千葉県	千葉市中央区千葉港XX-XX	043-456-XXXX	女	33
田中浩太	大阪府	大阪市北区中之島XX-XX	06-5678-XXXX	男	36

↓ `select name, phone from address`

name	phone
山田太郎	03-1234-XXXX
鈴木花子	045-234-XXXX
高橋次郎	048-345-XXXX
斎藤智子	043-456-XXXX
田中浩太	06-5678-XXXX

選択を行う（一部の行を取り出す）

テーブルの中から、**条件を指定して一部の行だけを取り出したい**ことも、非常によくあります。このような操作を「**選択**」と呼びます（43 ページ参照）。選択も select 文で行うことができます。

where 句を指定する

選択を行う場合には、選択の条件を指定するために、select 文に「**where**」という部分を追加します。この場合の select 文の書き方は、次のようになります。

```
select 列名 , 列名 , … from テーブル名 where 条件
```

「条件」の箇所に、選択の条件を指定します。条件の書き方はいくつかあり、それらを組み合わせることで、多彩な条件を指定することができます。

なお、「列名 , 列名 , …」の部分では、テーブルから取り出す列を指定することができます（射影）。また、すべての列を取り出したい場合は、この部分に「*」を指定します。

「列の値が○○に等しい」という条件

もっとも単純な条件は、「**列の値が○○に等しい**」というものです。この条件は、「列名 = 値」のように書きます。値が文字の場合は、通常はその値の前後を「'」や「"」で囲みます。

例えば、「『address』テーブルの行の中で、『pref』列の値が『東京都』に等しい」という行だけを取り出したいとします。また、列はすべて取り出すとします。この場合の書き方は、次のようになります。

```
select * from address where pref = '東京都'
```

図 3.2 は、上記の select 文の例です。上半分に address テーブルがあり、その中で「pref」列の値が「東京都」になっているのは、「山田太郎」さんの行だけです（「pref」列に網掛けがしてある行）。したがって、上記の select 文の結果には、「山田太郎」さんの行だけが取り出されます。

❤図 3.2　「select * from address where pref = '東京都'」の select 文の動作

addressテーブル

name	pref	address1	phone	sex	age
山田太郎	東京都	新宿区市谷左内町XX-XX	03-1234-XXXX	男	32
鈴木花子	神奈川県	横浜市西区中央XX-XX	045-234-XXXX	女	28
高橋次郎	埼玉県	さいたま市浦和区常盤XX-XX	048-345-XXXX	男	41
斎藤智子	千葉県	千葉市中央区千葉港XX-XX	043-456-XXXX	女	33
田中浩太	大阪府	大阪市北区中之島XX-XX	06-5678-XXXX	男	36

```
select * from address from pref = '東京都'
```

name	pref	address1	phone	sex	age
山田太郎	東京都	新宿区市谷左内町XX-XX	03-1234-XXXX	男	32

「列の値が○○に等しくない」という条件

「列の値が○○に等しい」という条件の逆に、「**列の値が○○に等しくない**」という条件で行を取り出すことができます。この条件は、「**列名 <> 値**」や「**列名 != 値**」のように書きます。「<>」「!=」のどちらを使うかは、RDBMS ごとに異なります。

例えば、「『address』テーブルの行の中で、『pref』列の値が『埼玉』に等しくない」という行だけを取り出したいとします。また、列はすべて取り出すとします。この場合の書き方は、次のようになります。

```
select * from address where pref <> '埼玉'
```

「列の値が○○より大きい」などの条件

「列の値が○○より大きい」など、**大小関係の条件**で行を取り出すことができます(**表 3.4**)。

例えば、「『address』テーブルの行の中で、『age』列の値が 30 以上」という行だけを取り出したいとします。また、列のうち、「name」列と「age」列だけを取り出したいとします。この場合の書き方は、次のようになります。

```
select name, age from address where age >= 30
```

また、「列の値が○○以上□□以下」という条件もよく使います。この条件は、「列名 between ○○ and □□」と表すことができます。

例えば、先ほどの例で、「『address』テーブルの行の中で、『age』列の値が30以上39以下」という行だけを取り出したいとします。この場合、select文は次のように書きます。

```
select name, age from address where age between 30 and 39
```

♥表3.4 「列の値が○○より大きい」などの条件の表し方

条件	条件の表し方
列の値が○○より大きい	列名 > ○○
列の値が○○より小さい	列名 < ○○
列の値が○○以上	列名 >= ○○
列の値が○○以下	列名 <= ○○

♥図3.3 「select name, age from address where age >= 30」のselect文の動作

addressテーブル

name	pref	address1	phone	sex	age
山田太郎	東京都	新宿区市谷左内町XX-XX	03-1234-XXXX	男	32
鈴木花子	神奈川県	横浜市西区中央XX-XX	045-234-XXXX	女	28
高橋次郎	埼玉県	さいたま市浦和区常盤XX-XX	048-345-XXXX	男	41
斎藤智子	千葉県	千葉市中央区千葉港XX-XX	043-456-XXXX	女	33
田中浩太	大阪府	大阪市北区中之島XX-XX	06-5678-XXXX	男	36

```
select name, age from address where age >= 30
```

name	age
山田太郎	32
高橋次郎	41
田中浩太	36
斎藤智子	33

「列の値に○○を含む(含まない)」などの条件

「住所の先頭が『新宿区』で始まる」「名前の中に『田』が含まれる」というように、「**列の値に『○○』を含む**」といった条件で検索したい場合もあります。このときは、条件の中で「**like**」を使います(**表3.5**)。

例えば、「『address』テーブルの行の中で、『address1』列の**値の先頭**が『新宿区』」という条件で、行を取り出したいとします(列はすべて取り出す)。この場合、select文は次のように書きます(**図3.4**)。

```
select * from address where address1 like '新宿区%'
```

なお、likeの代わりに「**not like**」を使うと、「**列の値に○○を含まない**」といった条件を指定することもできます。例えば、上の例を次のように変えると、「『address』テーブルの行の中で、『address1』列の値の先頭が『新宿区』でない」という条件で、行を取り出すことができます。

```
select * from address where address1 not like '新宿区%'
```

❤表 3.5　「列の値に○○を含む」などの条件の表し方

条件	条件の表し方
列の値の先頭が○○	列名 like '○○%'
列の値の最後が○○	列名 like '%○○'
列の値に○○を含む	列名 like '%○○%'

❤図 3.4　「select * from address where address1 like '新宿区%'」の select 文の動作

addressテーブル

name	pref	address1	phone	sex	age
山田太郎	東京都	新宿区市谷左内町XX-XX	03-1234-XXXX	男	32
鈴木花子	神奈川県	横浜市西区中央XX-XX	045-234-XXXX	女	28
高橋次郎	埼玉県	さいたま市浦和区常盤XX-XX	048-345-XXXX	男	41
斎藤智子	千葉県	千葉市中央区千葉港XX-XX	043-456-XXXX	女	33
田中浩太	大阪府	大阪市北区中之島XX-XX	06-5678-XXXX	男	36

```
select * from address where address1 like '新宿区%'
```

name	pref	address1	phone	sex	age
山田太郎	東京都	新宿区市谷左内町XX-XX	03-1234-XXXX	男	32

「列の値がリストの中のどれか 1 つに等しい」という条件

「都道府県名が『東京都』『神奈川県』『埼玉県』のいずれか」など、**列の値がリストの中のどれか1つの値と等しい**ことを、条件にしたい場合もあります。このときは、条件の中で「**in**」を使います。

```
select … from テーブル名 where 列名 in ( 値1, 値2, …, 値n)
```

例えば、上で挙げた例で、都道府県名が「pref」列にあり、テーブル名が「address」だとすると、select 文は次のように書きます（図 3.5）。

```
select * from address where pref in ('東京都', '神奈川県', '埼玉県')
```

なお、「**列の値がリストの中のどれとも等しくない**」という条件を指定することもできます。その場合は、次のように条件の中で「**not in**」を使います。

```
select … from テーブル名 where 列名 not in (値1, 値2, …, 値n)
```

❤図 3.5 「select * from address where pref in ('東京都', '神奈川県', '埼玉県')」の select 文の動作

addressテーブル

name	pref	address1	phone	sex	age
山田太郎	東京都	新宿区市谷左内町XX-XX	03-1234-XXXX	男	32
鈴木花子	神奈川県	横浜市西区中央XX-XX	045-234-XXXX	女	28
高橋次郎	埼玉県	さいたま市浦和区常盤XX-XX	048-345-XXXX	男	41
斎藤智子	千葉県	千葉市中央区千葉港XX-XX	043-456-XXXX	女	33
田中浩太	大阪府	大阪市北区中之島XX-XX	06-5678-XXXX	男	36

↓ `select * from address where pref in ('東京都', '神奈川県', '埼玉県')`

name	pref	address1	phone	sex	age
山田太郎	東京都	新宿区市谷左内町XX-XX	03-1234-XXXX	男	32
鈴木花子	神奈川県	横浜市西区中央XX-XX	045-234-XXXX	女	28
高橋次郎	埼玉県	さいたま市浦和区常盤XX-XX	048-345-XXXX	男	41

「列に値が入力されていない行を取り出す（除外する）」という条件

テーブルによっては、列に値を入力しないこともあります。例えば、住所録のテーブルに電話番号と携帯番号の列がある場合、携帯電話しか持っていない人のデータを入力するときには、電話番号の列には値を入力しません。

「**列に値が入力されていない**」ということは、テーブルの中では「**null**」で表します（45 ページ参照）。また、「列に値が入力されていない」ことを条件に指定する場合、「**列名 is null**」と書きます。

例えば、「address」テーブルの行の中から、「phone」列が null である（「phone」列に値が入力されていない）行を取り出したい場合、select 文を次のように書きま

す（図 3.6）。

```
select * from address where phone is null
```

なお、上記の逆に、「「phone」列の値が null でない（「phone」列に値が入力されている）」という条件で行を取り出したい場合、次のように「**列名 is not null**」の条件を指定します。

```
select * from address where phone is not null
```

図 3.6　「select * from address where phone is null」の select 文の動作

addressテーブル

name	pref	address1	phone	mobile
山田太郎	東京都	新宿区市谷左内町XX-XX	03-1234-XXXX	03-1234-XXXX
鈴木花子	神奈川県	横浜市西区中央XX-XX	null	045-234-XXXX
高橋次郎	埼玉県	さいたま市浦和区常盤XX-XX	048-345-XXXX	048-345-XXXX
斎藤智子	千葉県	千葉市中央区千葉港XX-XX	043-456-XXXX	null
田中浩太	大阪府	大阪市北区中之島XX-XX	06-5678-XXXX	06-5678-XXXX

```
select * from address where phone is null
```

name	pref	address1	phone	mobile
鈴木花子	神奈川県	横浜市西区中央XX-XX	null	045-234-XXXX

複数の条件を組み合わせて選択する

ここまでに挙げた例では、行を選択する際の条件は 1 つだけでした。しかし、より複雑な選択を行う場合、複数の条件を指定することもあります。この場合、条件を「and」「or」「not」で結びます。

2 つの条件がともに成立する——and

2 つの条件を指定し、それらの**条件の両方を満たす行だけ**を取り出したい場合、その 2 つの条件を「and」で結びます。

```
select … from テーブル名 where 条件1 and 条件2
```

例えば、「address」テーブルの中から、「pref」列の値が「東京都」で、かつ「age」列の値が30以上の行を取り出したいとします。この場合、select文を次のように書きます。

```
select * from address where pref = '東京都' and age >= 30
```

図3.7の例だと、「pref」列の値が「東京都」の行は2件あります(図中の「pref」列が網掛けになっている行)。また、「age」列の値が30以上の行は4件あります(図中の「age」列が網掛けになっている行)。

そして、この2つの条件を両方満たしているのは、1行目の「山田太郎」さんの行だけです。したがって、上記のselect文では、「山田太郎」さんの行だけが取り出されます。

◆図3.7 「select * from address where pref = '東京都' and age >= 30」のselect文の動作

addressテーブル

name	pref	address1	phone	sex	age
山田太郎	東京都	新宿区市谷左内町XX-XX	03-1234-XXXX	男	32
鈴木花子	神奈川県	横浜市西区中央XX-XX	045-234-XXXX	女	28
高橋次郎	埼玉県	さいたま市浦和区常盤XX-XX	048-345-XXXX	男	41
斎藤智子	千葉県	千葉市中央区千葉港XX-XX	043-456-XXXX	女	33
田中浩太	大阪府	大阪市北区中之島XX-XX	06-5678-XXXX	男	36
杉山恵美	東京都	東京都渋谷区宇田川町X-XX	03-2345-XXXX	女	27

```
select * from address where pref = '東京都' and age >= 30
```

name	pref	address1	phone	sex	age
山田太郎	東京都	新宿区市谷左内町XX-XX	03-1234-XXXX	男	32

2つの条件のどちらかが成立する——or

2つの条件を指定し、それらの**条件のどちらかを満たす行**を取り出すこともできます。この場合、2つの条件を「or」で結びます。

```
select … from テーブル名 where 条件1 or 条件2
```

例えば、「address」テーブルの中から、「pref」列の値が「東京都」か、または「sex」

列の値が「男」の行を取り出したいとします。この場合、select 文を次のように書きます。

```
select * from address where pref = '東京都' or sex = '男'
```

図 3.8 の例だと、「pref」列の値が「東京都」の行は 2 件あります（図中の「pref」列が網掛けになっている行）。また、「sex」列の値が「男」の行は 3 件あります（図中の「sex」列が網掛けになっている行）。

そして、この 2 つの条件のどちらかを満たしているのは、全部で 4 件あります。上記の select 文では、それらの 4 件の行が取り出されます。

❤図 3.8　「select * from address where pref = '東京都' or sex = '男'」の select 文の動作

addressテーブル

name	pref	address1	phone	sex	age
山田太郎	東京都	新宿区市谷左内町XX-XX	03-1234-XXXX	男	32
鈴木花子	神奈川県	横浜市西区中央XX-XX	045-234-XXXX	女	28
高橋次郎	埼玉県	さいたま市浦和区常盤XX-XX	048-345-XXXX	男	41
斎藤智子	千葉県	千葉市中央区千葉港XX-XX	043-456-XXXX	女	33
田中浩太	大阪府	大阪市北区中之島XX-XX	06-5678-XXXX	男	36
杉山恵美	東京都	東京都渋谷区宇田川町X-XX	03-2345-XXXX	女	27

```
select * from address where pref = '東京都' or sex = '男'
```

name	pref	address1	phone	sex	age
山田太郎	東京都	新宿区市谷左内町XX-XX	03-1234-XXXX	男	32
高橋次郎	埼玉県	さいたま市浦和区常盤XX-XX	048-345-XXXX	男	41
田中浩太	大阪府	大阪市北区中之島XX-XX	06-5678-XXXX	男	36
杉山恵美	東京都	東京都渋谷区宇田川町X-XX	03-2345-XXXX	女	27

条件を否定する――not

条件によっては、「○○である」という書き方をするよりも、「○○**でない**」という書き方をした方が分かりやすいこともあります。その場合、条件を括弧で囲み、その前に「**not**」を書いて、条件を否定するように書くことができます。

```
select … from テーブル名 where not( 条件 )
```

例えば、「『address』テーブルから、『pref』列の値が『東京都』と『神奈川県』の行を除く」という条件で、行を取り出したいとします。この条件は「『pref』列の値が『東京都』か『神奈川県』である」という条件を否定した形になっています。

したがって、not を使ってこの処理の select 文を書くと、次のように書くことができます[注]。

```
select * from address where not(pref in ('東京都', '神奈川県'))
```

「pref in ('東京都', '神奈川県')」が、「『pref』列の値が『東京都』か『神奈川県』である」という条件を表しています。そして、この条件全体を括弧で囲み、「not」をつけているので、「『pref』列の値が『東京都』でも『神奈川県』でもない」という条件を表すことになります(図 3.9)。

図 3.9 「select * from address where not(pref in ('東京都', '神奈川県'))」の select 文の動作

addressテーブル

name	pref	address1	phone	sex	age
山田太郎	東京都	新宿区市谷左内町XX-XX	03-1234-XXXX	男	32
鈴木花子	神奈川県	横浜市西区中央XX-XX	045-234-XXXX	女	28
高橋次郎	埼玉県	さいたま市浦和区常盤XX-XX	048-345-XXXX	男	41
斎藤智子	千葉県	千葉市中央区千葉港XX-XX	043-456-XXXX	女	33
田中浩太	大阪府	大阪市北区中之島XX-XX	06-5678-XXXX	男	36
杉山恵美	東京都	東京都渋谷区宇田川町X-XX	03-2345-XXXX	女	27

```
select * from address where not(pref in ('東京都', '神奈川県'))
```

name	pref	address1	phone	sex	age
高橋次郎	埼玉県	さいたま市浦和区常盤XX-XX	048-345-XXXX	男	41
斎藤智子	千葉県	千葉市中央区千葉港XX-XX	043-456-XXXX	女	33
田中浩太	大阪府	大阪市北区中之島XX-XX	06-5678-XXXX	男	36

注

「select * from address where pref not in ('東京都', '神奈川県')」とも書くことができますが、ここでは not を使う条件の例として、本文中に挙げた書き方を紹介しています。

3つ以上の条件を組み合わせる

and／or／notを使って、3つ以上の条件を組み合わせることもできます。ただし、その場合は**and／or／notの優先順位**に注意する必要があります。notの優先順位がもっとも高く、その次がandで、orは最も順位が低いです。

例えば、次のようなSQLを実行するとします。

```
select * from address where pref = '東京都' and sex = '男' or pref = ↵
'埼玉県' and sex = '女'
```

注

↵記号は行が続いていることを表します。

この場合、orよりもandの方が優先順位が高いので、「pref = '東京都' and sex = '男'」と「pref = '埼玉県' and sex = '女'」が先に判断されます。そして、その後に「or」が判断されます。

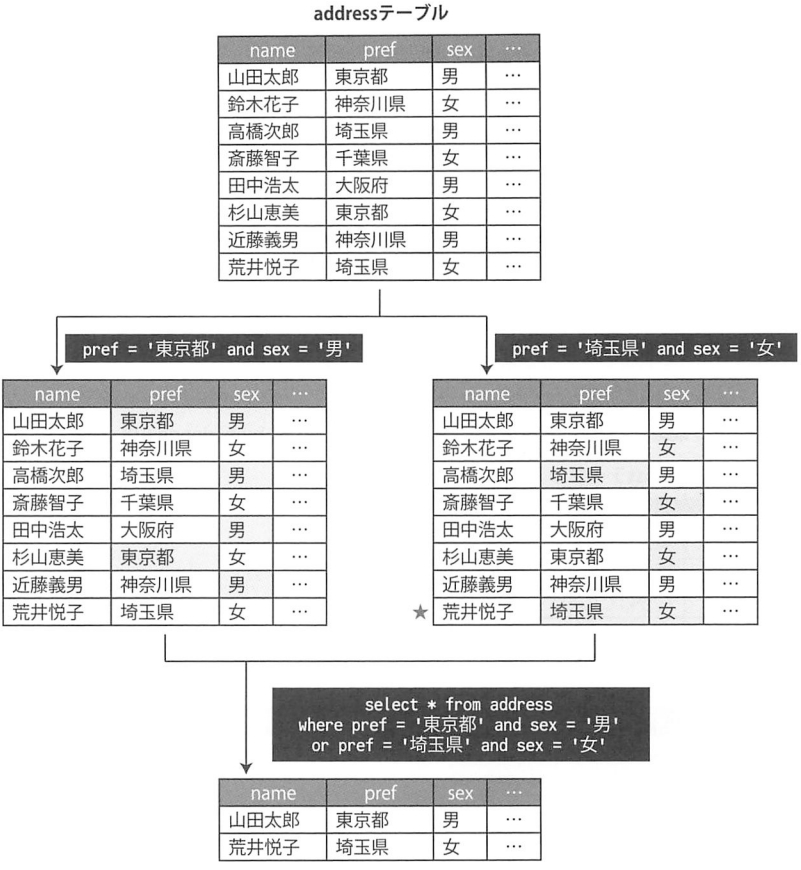
したがって、上のSQLを実行すると、「『pref』列の値が『東京都』で、かつ『sex』列の値が『男』」の行か、または「『pref』列の値が『埼玉県』で、かつ『sex』列の値が『女』」の行が取り出されます。

例えば、addressテーブルが**図3.10**の一番上のようになっていたとします。まず、「pref = '東京都' and sex = '男'」の条件を満たす行を探すと、**図3.10**の中央左のようになります。「pref = '東京都'」を満たす行が2行あり、「sex = '男'」を満たす行が4行ありますが、両方の条件を満たすのは「山田太郎」さんの行だけです（行の先頭に「★」をつけた行）。

同様に、「pref = '埼玉県' and sex = '女'」の条件を満たす行を探すと、**図3.10**の中央右のように、「荒井悦子」さんの行だけです。

そして、最後にこれらの条件の「or」が判断されます。したがって、select文の結果は、**図3.10**の一番下のようになります。

◆図 3.10 「select * from address where pref = '東京都' and sex = '男' or pref = '埼玉県' and sex = '女'」の select 文の動作



行の並べ替え

テーブルから行を取り出す際に、並べ替えを行いたい場合もよくあります。例えば、住所録のデータベースであれば、名前をあいうえお順に並べ替えることがよくあります。**行を並べ替える**には、select 文の最後に「**order by**」という句を追加します。

order by の書き方

「**order by 列名**」を追加すると、その列の値をキーにして、昇順で並べ替えることができます。「**昇順**」とは、数値なら値が小さいものから、文字なら文字コードの小さいものから、順に出力する方法です。

また、「**order by 列名 desc**」とすると、**降順**（昇順の逆）に並べ替えることができます。「desc」は「descend」（「下る」という意味）の略です。

例えば、**図 3.11** の上半分のような「address」テーブルがあるとします。このテーブルから、「sex」列の値が「男」の行を取り出し、「furigana」列の値の順に並べ替えたいとします（**図 3.11** の下半分）。この場合の select 文は、次のように書きます。

```
select * from address where sex = '男' order by furigana
```

図 3.11　「select * from address where sex = '男' order by furigana」の select 文の動作

addressテーブル

name	furigana	sex
山田太郎	やまだたろう	男
鈴木花子	すずきはなこ	女
高橋次郎	たかはしじろう	男
斎藤智子	さいとうともこ	女
田中浩太	たなかこうた	男

```
select * from address where sex = '男' order by furigana
```

name	furigana	sex
高橋次郎	たかはしじろう	男
田中浩太	たなかこうた	男
山田太郎	やまだたろう	男

複数の列をキーにして並べ替える

並べ替えの際に、キーの列の値が同じ行が、複数行存在することもあります。例えば、年齢を表す「age」という列があって、その列をキーにして並べ替えたいとします。この場合、「age」列の値が同じ行が複数ある（つまり、同じ年齢の人が複数いる）こともあり得ます。

そこで、**並べ替えのキーにする列を複数指定する**こともできます。例えば、「年齢順に並べ替えて、年齢が同じ人は名前のあいうえお順で並べ替える」といったことができます。

並べ替えのキーを複数指定するには、次のように、**order by** の後に列名を順に書きます。

```
select …… order by 列名1, 列名2, ……
```

例えば、次の select 文を実行すると、「address」テーブルからすべての行が取り出され、まず「age」列の値の昇順に並べ替えます。そして、「age」列の値が同じ行が複数ある場合は、それらの行を「furigana」列の値の昇順に並べ替えます。

```
select * from address where sex = '男' order by age, furigana
```

また、キーの列ごとに、昇順／降順を指定することもできます。降順で並べ替える列は、列名の後に「**desc**」を追加します。

例えば、先ほど挙げた select 文を次のように変えると、まず「age」列の値の降順に並べ替え、「age」列の値が同じ行が複数ある場合は、それらの行を「furigana」列の値の昇順に並べ替えます。

```
select * from address where sex = '男' order by age desc, furigana
```

値が重複する行を除外する

select 文の内容によっては、結果の複数の行で、各列の値が全く同じ値になる（重複する）こともあります。例えば、全く同じ内容の行を2行入力しているときに、テーブルからすべての行を取り出すと、その2行は重複します。

重複する行がある場合、それらの行を1回だけ取り出すこともできます。それには、select 文の列名を指定する部分の前に、「**distinct**」を追加します。

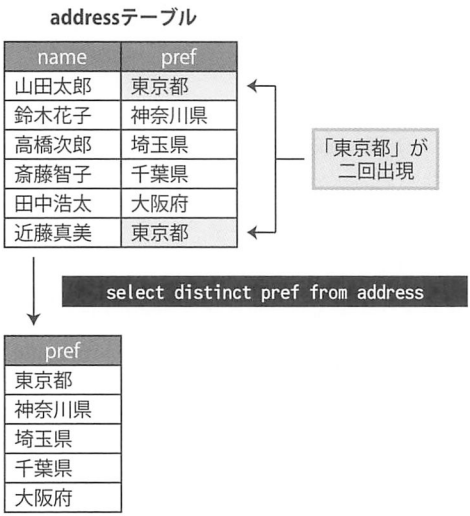
```
select distinct 列名 from ……
```

例えば、**図 3.12** の上半分の「address」テーブルから、「pref」列だけを取り出すとします。単に「select pref from address」すると、1行目と6行目の「東京都」がどちらも結果に出力されます。

ここで、select 文を次のように書くと、重複する「東京都」は 1 回だけ出力され、図 3.12 の下半分のような結果が得られます。

```
select distinct pref from address
```

図 3.12 「select distinct pref from address」の select 文の動作



3.3 テーブル内での計算や集計

select 文では、テーブル内の列同士を計算したり、行を集計することもできます。また、集計の際に行をグループ化することもできます。

列同士の計算

例えば、テーブルに商品の販売の情報を保存するとします。この場合、テーブルに商品の単価と個数の列を作り、さらにそれらを掛け算した値を金額の列に保存する、といった手法をとることが考えられます。

ただ、金額は単価と個数から計算で求めることができます。この例のように、他の列の値を元に、何らかの計算を行って求められる列のことを、「**導出列**」または「**導出項目**」と呼びます。

導出列を作るのも、select 文で行うことができます。この場合、select 文で、列名を指定する部分に、導出列の値を求めるための式を書きます。

例えば、「sales」というテーブルに、「price」(単価)と「count」(個数)という列があるとします。そして、これら 2 つの列を行ごとに掛け算して、その金額も得たいとします。この場合、select 文を次のように書きます(**図 3.13**)。

```
select *, price * count from sales
```

なお、「price」と「count」の間にある「*」は、これら 2 つの列の値を掛け算することを意味します。掛け算以外に、**表 3.6** のような計算を行うことができます。

また、RDBMS によっては、各種の「**関数**」を使って、特殊な計算を行うこともできます。関数とは、入力に対して何らかの計算を行って、その結果を出力するものです。例えば、「数値からプラスマイナスの符号を取った値を求める」「文字列の長さを求める」といった関数が考えられます。

❤図 3.13　price 列と count 列を掛け算して金額を求める

salesテーブル

item_no	price	count
1	500	20
2	800	15
3	600	10
1	500	10
3	600	25

```
select *, price * count from sales
```

item_no	price	count	price * count	
1	500	20	10000	← 500 * 20
2	800	15	12000	← 800 * 15
3	600	10	6000	← 600 * 10
1	500	10	5000	← 500 * 10
3	600	25	15000	← 600 * 25

❤表 3.6　計算を行う際の記号（演算子）

演算子	意味
+	足し算
-	引き算
*	掛け算
/	割り算

列に名前をつける

式を使って導出列を作る際に、その列に名前をつけたい場合もあります。そのときは、式の後に「as 列名」という部分を追加して、列の名前を指定します。

例えば、先ほどの例で、price 列と count 列を掛け算した結果の列に、「subtotal」という名前をつけたいとします。この場合、select 文を次のように書きます。

```
select *, price * count as subtotal from sales
```

なお、導出列だけでなく、通常の列にも名前をつけることができます。

集計を行う

select 文では、テーブル内の行に対して集計を行って、**個数／合計／平均／最大値／最小値**を求めることができます。この場合、select 文は次のように書きます。

```
select 関数（列名）from テーブル名 where 条件
```

「**関数**」で、集計する方法を指定します。集計方法と、それに応じた関数の対応は、**表 3.7** のようになっています。また、「列名」で、集計の対象にする列を指定します。ただし、**個数（count）**を求める場合は、列名に「*」を指定します。

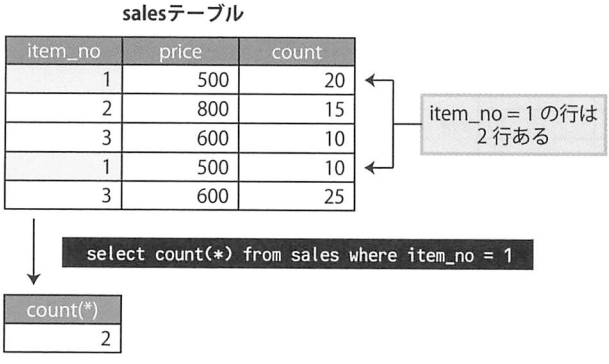
例えば、次の select 文を実行すると、「sales」テーブルの中で、「item_no」列の値が 1 になっている行の数が求められます。**図 3.14** のような場合だと、この select 文の結果は 2 になります。

```
select count(*) from sales where item_no = 1
```

表 3.7　**集計を行う関数**

関数	集計方法
count	個数
sum	合計
avg	平均
max	最大値
min	最小値

図 3.14　「select count(*) from sales where item_no = 1」の動作の例



列どうしの計算結果を集計する

集計を行う際に、テーブルに元からある列だけでなく、**列同士の計算結果を集計**することもできます。

例えば、「sales」テーブルの中で、「item_no」列の値が1番の商品について、売上の合計を求めたいとします。この処理は、次のように考えることができます。

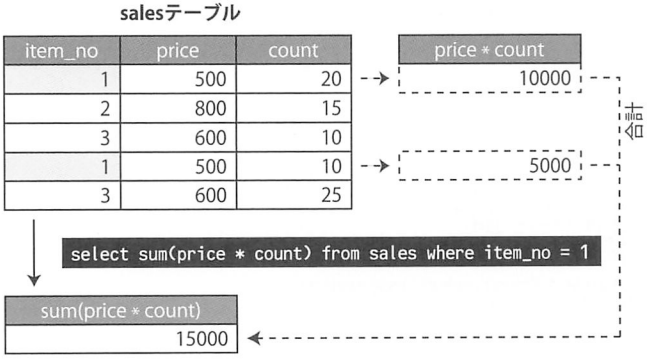
「sales」テーブルの中で、「item_no」列の値が1になっている行を対象に、「price」列と「count」列を掛け算して各行の小計を求め、それらの小計をすべて足し算する

この場合、合計したいのは、各行の「price」列と「count」列を掛け算した結果です。これは「sum(price * count)」のように書くことができます。また、「item_no」列の値が1になっている行を対象にしますので、where句の部分は「where item_no = 1」と書くことができます。これらの考え方を組み合わせてselect文を書くと、次のようになります。

```
select sum(price * count) from sales where item_no = 1
```

例えば、salesテーブルが**図3.15**の上半分のようになっていたとします。このテーブルを見ると、「item_id」列の値が1になっている行が2行あります。そこで、それら2行のそれぞれで「price」列と「count」列を掛け算した値が求められます（結果は10000と5000）。そして、それらの合計（sum）が求められ、結果は15000になります。

図3.15　「select sum(price * count) from sales where item_no = 1」の動作の例



複数の集計を一度に行う

select 文を使って集計する際に、「関数 (列名)」の部分を 2 回以上書いて、複数の集計を一度に行うこともできます。

例えば、**図 3.15** の例で、金額の合計だけでなく、個数(「count」列)の合計も合わせて求めたいとします。この場合、次のように select 文を書けば、求めることができます。

```
select sum(count), sum(price * count) from sales where item_no = 1
```

グループ化して集計する

先ほどの例では、「item_no」列の値が 1 の行を対象に、合計を求めました。ただ、実際の処理では、「すべての商品を対象に、商品ごとの金額を集計したい」といったことも多々あります。

select 文を使った集計では、列の値によって行をグループ化して、グループごとに集計することもできます。前述の**図 3.15** のようなテーブルの場合だと、「item_no」列の値で行をグループ化して、商品ごとの金額を集計する、といったことができます。

グループ化を行うには、select 文に「**group by**」という句を追加します。書き方は次のようになります。「group by」の後の「グループ化の列名」で、グループ化に使う列を指定します。

```
select グループ化の列名 , 関数 ( 集計対象の列名 ) from テーブル名 group by
グループ化の列名
```

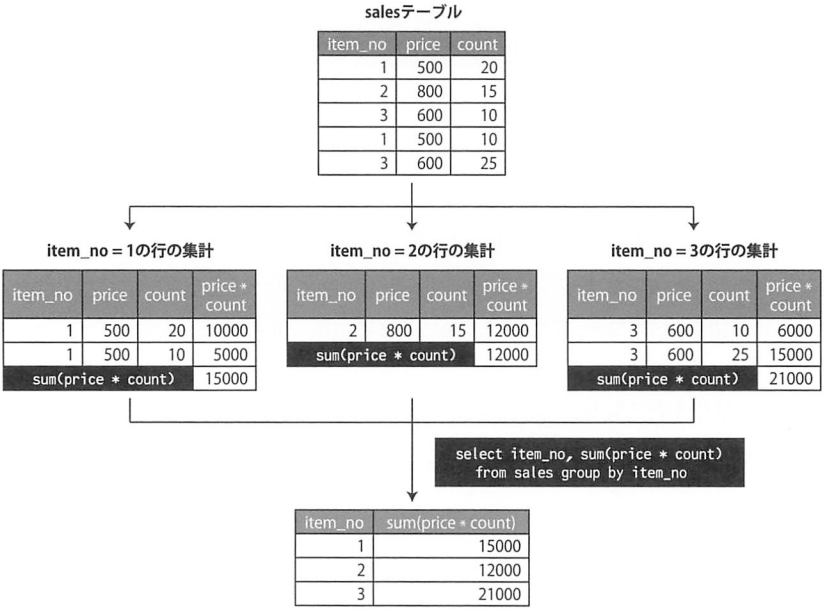
例えば、上で述べたように、**図 3.15** の「sales」テーブルを対象に、「item_no」列の値で行をグループ化して、それぞれの商品ごとの金額を求めるには、次のような select 文を実行します。

```
select item_no, sum(price * count) from sales group by item_no
```

この select 文は、次のようなイメージで処理されます。

①「item_no」列の値によって、行がグループに分けられます。
② それぞれのグループで、各行の「price」列と「count」列を掛けた値が求められます。
③ それぞれのグループで、各行の②で求めた値を合計します（図の中段部分）。
④ ③の集計結果をまとめて、「item_no」列と合計の列からなるテーブルのように出力します（図の下段部分）。

図 3.16　「select item_no, sum(price * count) from sales group by item_no」文の動作の例



salesテーブル

item_no	price	count
1	500	20
2	800	15
3	600	10
1	500	10
3	600	25

item_no = 1の行の集計

item_no	price	count	price * count
1	500	20	10000
1	500	10	5000
sum(price * count)			15000

item_no = 2の行の集計

item_no	price	count	price * count
2	800	15	12000
sum(price * count)			12000

item_no = 3の行の集計

item_no	price	count	price * count
3	600	10	6000
3	600	25	15000
sum(price * count)			21000

```
select item_no, sum(price * count)
from sales group by item_no
```

item_no	sum(price * count)
1	15000
2	12000
3	21000

複数の列でグループ化する

先ほどの例では、1つの列（「item_no」列）の値で行をグループ化しました。それだけでなく、複数の列の値を組み合わせて、行をグループ化することもできます。その場合、select文の「group by」の後に、グループ化に使う列の名前をコンマで区切って並べます。

例えば、日々のゲームのスコアを記録するために、「year」「month」「day」「score」の4つの列からなるテーブルを作り、テーブル名を「game」にしたとします。そ

して、このテーブルを元に、年月別のゲームのプレイ回数と、平均点を求めたいとします。この処理は、次のような select 文で行うことができます。

```
select year, month, count(*), avg(score) from game group by year, month
```

集計結果に条件を指定する

group by 句を使ってグループごとに集計し、その集計結果に対して条件を指定して、結果の一部だけを取り出したい、といったこともあります。

例えば、前ページの**図 3.16** のように、商品ごとの売り上げを求めたとします。この結果に対して、「売上が 2 万円以上の商品だけを取り出したい」という条件を指定して、結果の一部だけを取り出す、といったことが考えられます。

このように、集計結果に対して条件を指定したい場合は、select 文に「**having**」という句を追加します。

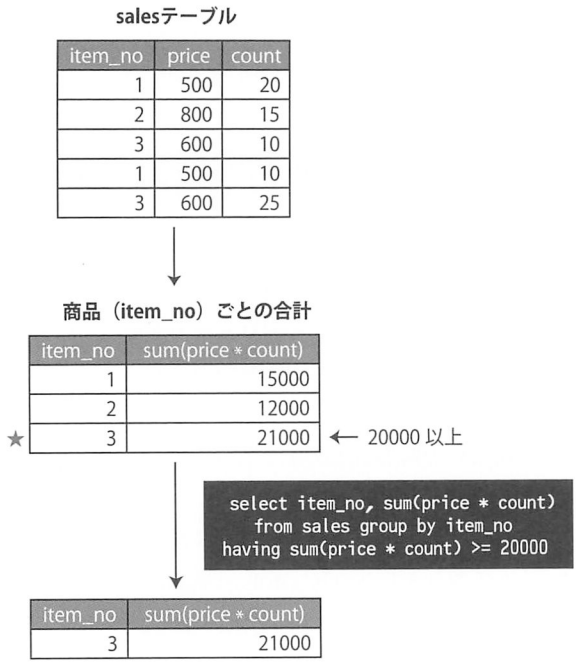
例えば、前述したように、前ページの**図 3.16** の集計を行って、「売上が 2 万円以上の商品だけを取り出す」という処理をしたいとします。「商品ごとの売上」は、前ページの select 文では、「sum(price * count)」によって求められます。

したがって、having 句では、「『sum(price * count)』の値が 20000 以上」のように条件を書きます。具体的な select 文は次のようになります。

```
select item_no, sum(price * count) from sales group by item_no having sum(price
* count) >= 20000
```

図 3.16 では、売上が 2 万円以上の商品は、「item_no」列の値が 3 の商品だけです。したがって、上の select 文の結果は、**図 3.17** の下段のようになります（図の中段の「★」の行）。

❤図 3.17　「select item_no, sum(price * count) from sales group by item_no having sum(price * count) >= 20000」の select 文の実行例



集計結果を並べ替える

sum 等の関数で**集計した結果を、並べ替えて出力**することもできます。

並べ替えには **order by** 句を使います。ただし、関数の集計結果で並べ替える場合は、「order by 列名」ではなく、「**order by 列番号**」のような書き方をします。select の後に列名や関数を書きますが、その左端の列や関数を 1 番の列として、列に番号を付けます。

例えば、100 ページの**図 3.16** のように商品ごとの売り上げを集計した後、売り上げの多い順に並べ替えて出力したいとします。

99 ページでは、select 文は次のように書きました。これを、sum 関数の集計結果で並べ替えるように変えます。

```
select item_no, sum(price * count) from sales group by item_no
```

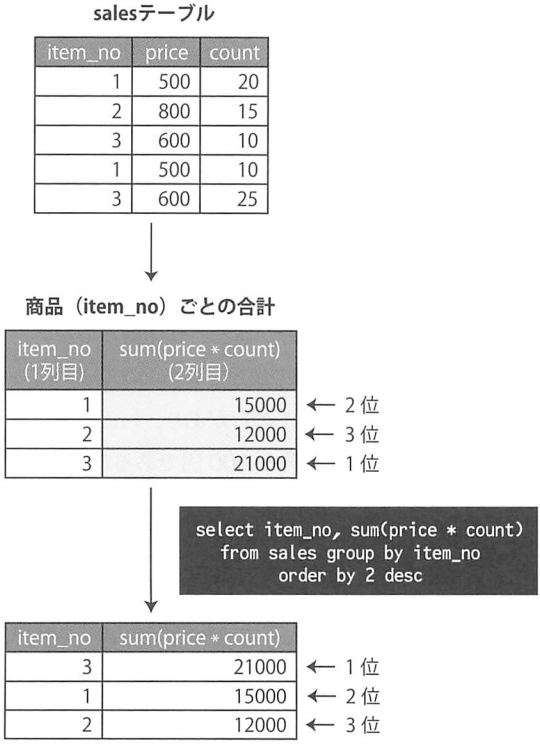
この select 文では、列として「item_no」と「sum(price * count)」の 2 つを指定していて、sum 関数は 2 番目の列にあたります。したがって、select 文を次のように書くことで、売り上げの多い順に並べ替えることができます(図 3.18)。

```
select item_no, sum(price * count) from sales group by item_no order by 2 desc
```

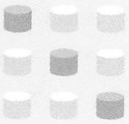
注「order by 関数」のように書く

現在の RDBMS では、関数の集計結果で並べ替える際に、「order by sum(price * count)」のように、関数の式を指定できる製品もあります。

❤図 3.18 「select item_no, sum(price * count) from sales group by item_no order by 2 desc」の select 文の動作の例



3.4 テーブルを結合する



リレーショナルデータベースの大きな特徴の1つとして、複数のテーブルを結合することが挙げられます。テーブルを結合してデータを取り出す場合も、select文を使います。

内部結合

テーブル間の結合には、いくつかのパターンがあります。その中で最もよく使うのが、「**内部結合**」という結合です。

内部結合とは？

内部結合は、**2つのテーブルを、「列の値が等しい」という条件で結びつけて、1つのテーブルのように扱う手法**です。

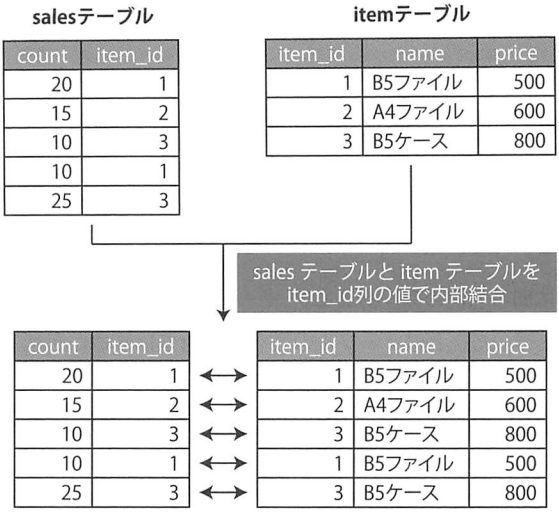
例えば、**図3.19**の上段のように、「sales」(売上)と「item」(商品)という2つのテーブルがあるとします。「sales」「item」のどちらのテーブルにも「item_id」という列があり、この列の値によって、販売した商品が分かるという仕組みです。

「sales」テーブルの1番上の行では、「item_id」列の値が1になっています。一方、「item」テーブルを見ると、「item_id」列の値が1の行があり、「name」列と「price」列から、その商品が「B5ファイル」で、値段が500円であることが分かります。

つまり、「sales」テーブルの1番上の行からは、「500円のB5ファイルを20冊販売した」という情報を得ることができます。

同様の手順で、「sales」テーブル内の各行と、「item」テーブル内の各行を、「item_id」列の値で結合していきます。その結果は**図3.19**の下段のようになります。

◆図 3.19 内部結合の例



内部結合を表す select 文

内部結合でテーブルから取り出す場合も、select 文を使います。次のように、**from 句にテーブル名を 2 つ書き、また where 句で結合する列を指定する**、という書き方をよく使います。

```
select 列名 , … from テーブル名 1, テーブル名 2 where テーブル名 1. 結合列名 =
テーブル名 2. 結合列名
```

例えば、前述の**図 3.19** の例だと、「sales」テーブルの「item_id」列と、「item」テーブルの「item_id」列が等しい、という条件で結合を行います。したがって、次のような select 文を書きます。

```
select * from sales, item where sales.item_id = item.item_id
```

また、次のように「inner join」を使った書き方をすることもできます。

```
select 列名 , … from テーブル名 1 inner join テーブル名 2 on テーブル名 1.
結合列名 = テーブル名 2. 結合列名
```

例えば、前述の select 文を inner join を使う書き方に変えると、次のようになります。

```
select * from sales inner join item on sales.item_id = item.item_id
```

なお、「inner join」は単に「**join**」とだけ書いてもかまいません。

テーブルに別名を付ける

結合する 2 つのテーブルに同じ名前の列がある場合、その列名を指定する際には、「**テーブル名 . 列名**」のように書いて、どのテーブルの列を操作対象にするかを指定する必要があります。

例えば、105 ページの**図 3.19** では、「sales」／「item」の両テーブルに「item_id」という列があります。したがって、「sales」テーブルの「item_id」列は、「sales.item_id」のように書く必要があります。

ただ、複雑な処理になると、「テーブル名 . 列名」の書き方では select 文が長くなり、見通しが悪くなります。

そこで、**テーブル名に別名を付ける**こともできます。それには、select 文の from 句を次のように書きます。

```
from テーブル名 1 別名 1, テーブル名 2 別名 2, ……
```

また、「inner join テーブル名 on」の書き方を使う場合は、「inner join テーブル名 別名 on」のように書きます。

さらに、テーブル名もつけて列名を指定する箇所では、「**別名 . 列名**」のように、テーブル名の代わりに別名を使うことができます。

例えば、次のような select 文があるとします。

```
select * from sales, item where sales.item_id = item.item_id
```

この select 文で、sales ／ item のテーブルに、別名として「s」と「i」を付けるとすると、次のように書きます。

```
select * from sales s, item i where s.item_id = i.item_id
```

また、これを inner join on を使って書きなおすと、次のようになります。

```
select * from sales s inner join item i on s.item_id = i.item_id
```

外部結合

内部結合に対して、「**外部結合**」という結合もあります。

結合する列の値が一致しない行があるとどうなる？

まず、**図 3.20** の上半分を見てください。これは、105 ページの**図 3.19** の「sales」／「item」テーブルに、それぞれ 1 行ずつ追加した形になっています。

これら 2 つのテーブルを見ると、「item_id」列の値が 1 から 3 の行は、両方のテーブルに存在しています。しかし、「item_id」列の値が 4 の行は、「item」テーブルにしかありません。また、「item_id」列の値が 5 の行は、「sales」テーブルにしかありません。

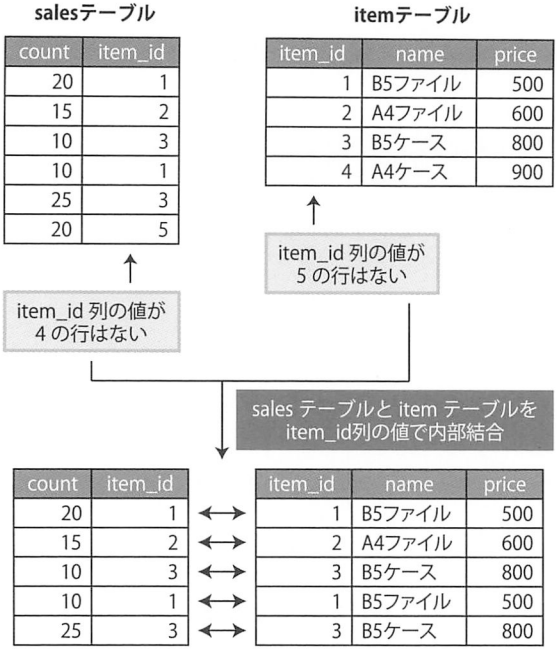
この例のように、**2 つのテーブルを結合する際に、結合に使う列の値が、どちらか 1 つのテーブルの行にしかない**こともあり得ます。

図 3.20 の場合だと、「item」テーブルの「item_id」列が 4 の行は、新たに扱うようにした商品の情報を追加したことを意味します。また、「sales」テーブルに「item_id」列の値が 4 の行はありませんが、これはまだその商品の販売実績がないことを意味しています。

一方、「sales」テーブルには、「item_id」列の値が 5 の行があります。これは、「item」テーブルにまだ存在しない商品の販売データを、間違えて入力したように見受けられます。

この状況で、「sales」テーブルと「item」テーブルを「item_id」列で結合すると、結果は**図 3.20** の下半分のようになります。「sales」テーブルの「item_id」列が 5 の行や、「item」テーブルの「item_id」列が 4 の行のように、結合する列の値が一致しない行は、内部結合では取り出されません。

図 3.20　結合する列の値が一致しない行がある場合の内部結合



左外部結合

図 3.20 のように、結合する列の値が一致しない行がある場合に、その行も含めて取り出したいこともあります。このような場合には、内部結合ではなく、「**外部結合**」という結合を使います。

外部結合には、「**左外部結合**」「**右外部結合**」「**完全外部結合**」の 3 種類があります。まず、左外部結合を紹介します。

左外部結合は、結合する 2 つのテーブルを左右に並べたとして、**左側のテーブルの行はすべて取り出し、その各行に右側のテーブルの行を結合する**、という処理を行います。

ただし、左側のテーブルの行の中で、右側のテーブルに対応する行がない場合、その行の右側のテーブル部分の各列は、値がない（null）状態になります。また、右側のテーブルにしか結合列に値がない行がある場合は、その行は取りだされません。

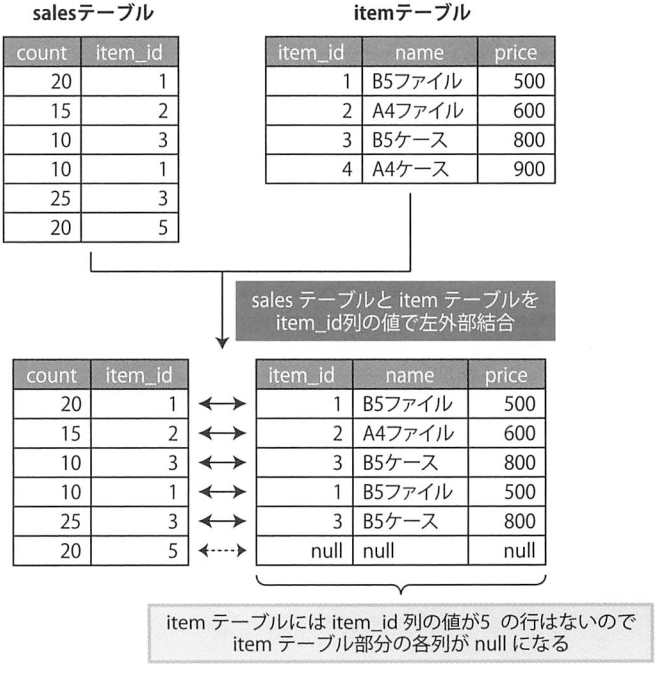
例えば、**図 3.21** の上半分のように、「sales」テーブルと「item」テーブルがあるものとします。そして、これら 2 つのテーブルを、「item_id」列で左外部結合す

るとします。「sales」テーブルを左、「item」テーブルを右と考えます。

この図では、「sales」テーブルには、「item_id」列の値が 5 の行があります。一方、「item」テーブルには「item_id」列の値が 5 の行はありません。そのため、「sales」テーブルの「item_id」列の値が 5 の行では、それに対応する「item」テーブル部分の各列は値がない状態になります(**図 3.21** の下半分)。

また、「item」テーブルには「item_id」列の値が 4 の行がありますが、「sales」テーブルにはそのような行はありません。そのため、「item」テーブルの「item_id」列の値が 4 の行は、select 文の結果には出力されません。

◆図 3.21　左外部結合の例



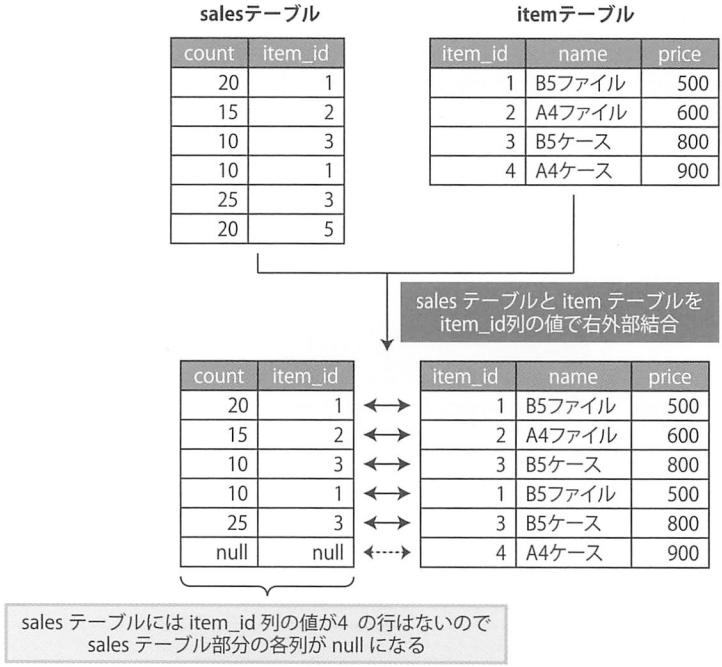
右外部結合

左外部結合は、左右 2 つのテーブルのうち、左側のテーブルの行をすべて取り出す手法でした。それと同様に、**右側のテーブルの行をすべて取り出す外部結合**もあります。これを「**右外部結合**」と呼びます。

例えば、**図 3.22** の上半分のように、「sales」テーブルと「item」テーブルがあるものとします。そして、これら 2 つのテーブルを、「item_id」列で右外部結合するとします。

「item」テーブルには、「item_id」列の値が 4 の行があります。しかし、「sales」テーブルには、「item_id」列の値が 4 の行はありません。したがって、「item」テーブルの「item_id」列の値が 4 の行を取り出す際には、対応する「sales」テーブル部分の列は null になります(**図 3.22** の下半分)。

❤図 3.22　右外部結合の例



完全外部結合

完全外部結合は、左／右外部結合を組み合わせた結合で、次のような動作をします。

① 左右両方のテーブルから、すべての行を取り出します。

② 左側のテーブルにしかない行では、それに対応する右側のテーブルの各列は、null

になります。

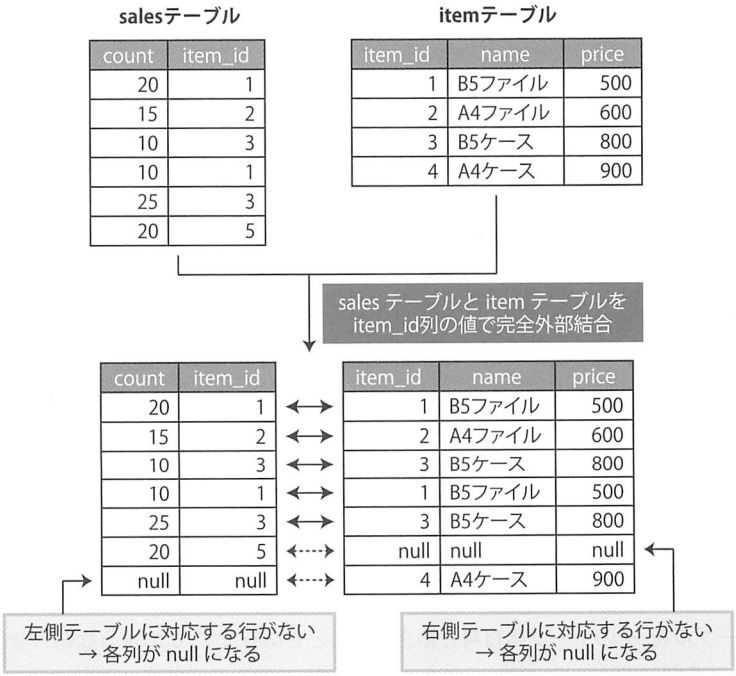
③右側のテーブルにしかない行では、それに対応する左側のテーブルの各列は、null になります。

例えば、**図 3.23** の上半分のように、「sales」テーブルと「item」テーブルがあるものとします。そして、これら 2 つのテーブルを、「item_id」列で完全外部結合するとします。

「item_id」列の値が 5 の行は、左側のテーブル（sales）にしかありません。したがって、この行の右側テーブル部分の各列は null になります。

一方、「item_id」列の値が 4 の行は、右側のテーブル（item）にしかありません。したがって、この行の左側テーブル部分の各列は null になります（**図 3.23** の下半分）。

❤図 3.23　完全外部結合の例



外部結合を select 文で表す

左／右／完全の各外部結合は、select 文ではそれぞれ「left outer join」「right outer join」「full outer join」を使って、次のように書きます。

①左外部結合

```
select 列名 , … from 左テーブル名 left outer join 右テーブル名 on 左テーブル名
. 結合列名 = 右テーブル名 . 結合列名
```

②右外部結合

```
select 列名 , … from 左テーブル名 right outer join 右テーブル名 on 左テーブル名
. 結合列名 = 右テーブル名 . 結合列名
```

③完全外部結合

```
select 列名 , … from 左テーブル名 full outer join 右テーブル名 on 左テーブル名
. 結合列名 = 右テーブル名 . 結合列名
```

例えば、これまでの例で述べてきたように、「sales」テーブルと「item」テーブルを「item_id」列で外部結合するには、次のように書きます。

①左外部結合

```
select * from sales left outer join item on sales.item_id = item.item_id
```

②右外部結合

```
select * from sales right outer join item on sales.item_id = item.item_id
```

③完全外部結合

```
select * from sales full outer join item on sales.item_id = item.item_id
```

なお、「outer join」の「outer」は、省略することができます。例えば、「left outer join」は、「left join」と書くことができます。

自己結合

あるテーブルを、そのテーブル自身と結合することもあります。このような結合を、「**自己結合**」と呼びます。

自己結合の例

例えば、社員の情報を管理するために、「employee」というテーブルを作るとします。このテーブルには、**図 3.24** のように、社員番号（id）や名前（name）などの列を作ります。

また、トップ以外の社員には上司がいますので、その上司の社員番号も保存します。列名は「superior_id」とすることにします。トップの人は、superior_id 列に値を入れないことにします（null）。

例えば、**図 3.24** を見ると、社員番号 1 番の「山田太郎」さんの行は、superior_id 列が null になっています。つまり、山田太郎さんが、この会社のトップであることを意味します。

ここで、各社員の名前と、それぞれの上司の名前を、一覧で見たいとします。この場合、**図 3.25** の上半分のように、「employee」テーブルを 2 つ並べて、左のテーブルの「superior_id」列と、右のテーブルの「id」列とを内部結合し、左右それぞれのテーブルから「name」列の値を取り出します（**図 3.25** の下半分）。

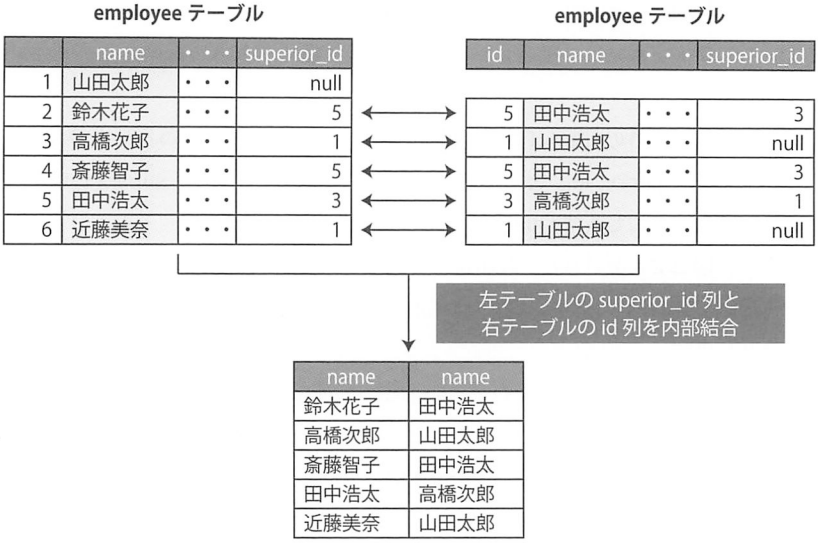
この例のように、あるテーブルを、そのテーブル自身と結合したい場合もあります。このような結合を、「**自己結合**」と呼びます。

図 3.24　employee テーブル

employeeテーブル

id	name	・・	superior_id
1	山田太郎	・・	null
2	鈴木花子	・・	5
3	高橋次郎	・・	1
4	斎藤智子	・・	5
5	田中浩太	・・	3
6	近藤美奈	・・	1

❤図 3.25　各社員の名前とそれぞれの上司の名前を得る手順



	name	・・・	superior_id
1	山田太郎	・・・	null
2	鈴木花子	・・・	5
3	高橋次郎	・・・	1
4	斎藤智子	・・・	5
5	田中浩太	・・・	3
6	近藤美奈	・・・	1

id	name	・・・	superior_id
5	田中浩太	・・・	3
1	山田太郎	・・・	null
5	田中浩太	・・・	3
3	高橋次郎	・・・	1
1	山田太郎	・・・	null

name	name
鈴木花子	田中浩太
高橋次郎	山田太郎
斎藤智子	田中浩太
田中浩太	高橋次郎
近藤美奈	山田太郎

自己結合を行う select 文

select 文で from 句を書く際に、**テーブル名に別名を付ける**ことができます（106 ページ参照）。**自己結合を行うには、この機能を利用**します。左右 2 つのテーブルが同じテーブルなので、左／右それぞれに違う別名を付けて、両者を区別できるようにします。

例えば、**図 3.25** の左右の employee テーブルに、それぞれ「l」と「r」の別名を付けるとします（「l」「r」は「left」と「right」の略）。すると、両テーブルを結合する条件は、「l テーブルの superior_id 列と、r テーブルの id 列が等しい」と表すことができます。また、取り出す列は、l テーブルの name 列と、r テーブルの name 列になります。

ここまでの考え方に沿って select 文を書くと、次のようになります。

```
select l.name, r.name from employee l, employee r where l.superior_id = r.id
```

なお、**図 3.25** の例では、左右の employee テーブルを内部結合したので、上司がいない「山田太郎」さんの行は、結果には出力されませんでした。これを、次のように**左外部結合**に変えれば、山田太郎さんの行も結果に出力され、上司の名前の

列は null になります。

```
select l.name, r.name from employee l left outer join employee r on
l.superior_id = r.id
```

結合の select 文の様々な書き方

ここまでは、select 文で結合する方法を書いてきましたが、結合した結果から一部の行を取り出すなど、様々な書き方をすることができます。

他の条件も指定する

単純に 2 つのテーブルを結合するだけだと、両者のテーブルで結合列の値が等しい行は、すべて結果に出力されます。しかし、2 つのテーブルを結合した上で、一部の行だけを選択したい場合もあります。

選択のための条件を追加すれば、一部の行だけを取り出すことができます。 inner join を使わない書き方だと、次のように、結合の条件の後に「and」で他の条件を追加します。

```
select 列名 , … from テーブル名 1, テーブル名 2 where テーブル名 1. 結合列名 =
テーブル名 2. 結合列名 and 選択の条件
```

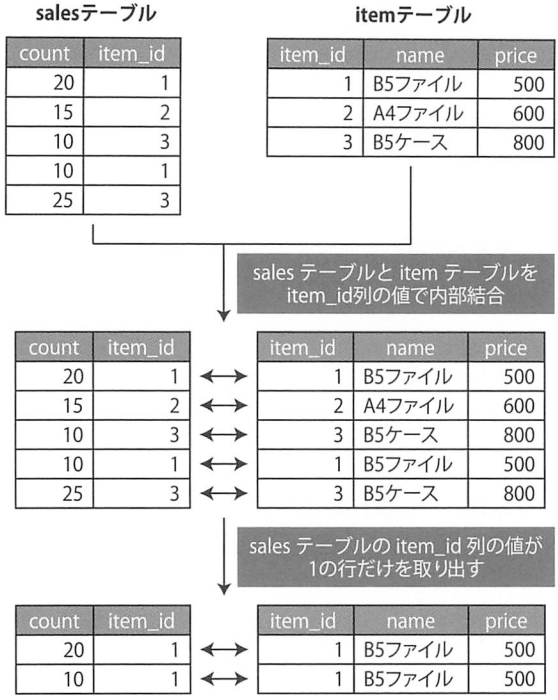
また、inner join や outer join を使う書き方なら、「on テーブル名 1. 結合列名 = テーブル名 2. 結合列名」の後に where 句を追加します。

```
select 列名 , … from テーブル名 1 [inner | left outer | right outer |
full outer ] join テーブル名 on テーブル名 1. 結合列名 = テーブル名 2. 結合列名
where 選択の条件
```

例えば、**図 3.26** の上段のように「sales」テーブルと「item」テーブルがある時に、両テーブルを「item_id」列で結合し（**図 3.26** の中段）、「sales」テーブルの「item_id」列の値が 1 の行だけを取り出すとします（**図 3.26** の下段）。この場合の select 文は、次のように書きます。

```
select * from sales, item where sales.item_id = item.item_id and sales.item_id = 1
```

❤図 3.26　「select * from sales, item where sales.item_id = item.item_id and sales.item_id = 1」の select 文の動作



集計やグループ化を組み合わせる

テーブルを結合した上で、集計を行うこともできます。また、集計の際に行をグループ化することもできます。

例えば、図 3.27 の上段のように、「sales」と「item」の 2 つのテーブルがあるとします。これらのテーブルから、図 3.27 の下段のように、商品ごとの売り上げ合計を求めたいとします。

この処理は、次のような手順で行うことができます。

①「sales」テーブルと「item」テーブルを、「item_id」列の値で内部結合します。
②「sales」テーブルの「item_id」列の値で、①の結果をグループ化します。
③②のグループ毎に、「sales」テーブルの「count」列の値と「item」テーブルの「price」列の値を掛けた値を合計します。
④③の結果から、各グループの「sales」テーブルの「item_id」列、「item」テーブルの「name」列、そしてグループ毎の金額の合計を出力します。

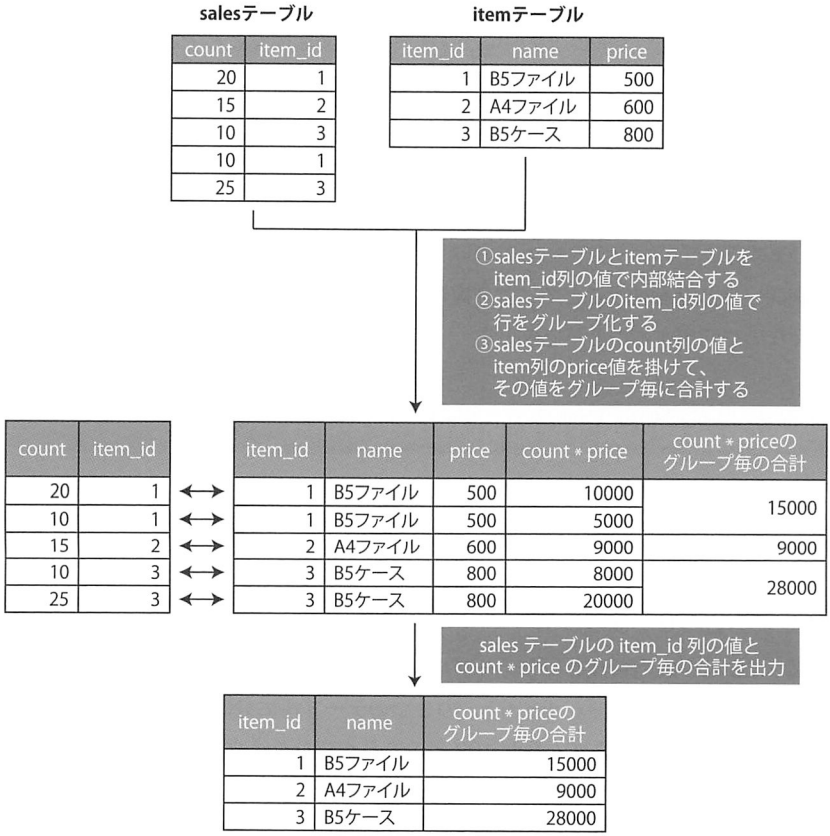
上記の 4 つから、次のように select 文の各部分を組み立てます。

①から、結合の条件は「sales.item_id = item.item_id」になります。
②から、グループ化の方法は「group by sales.item_id」になります。
③から、集計の方法は「sum(sales.count * item.price)」になります。
④から、取り出す列は「sales.item_id」「item.name」「sum(sales.count * item.price)」の 3 つになります。

これらを組み合わせて、select 文は次のようになります。

```
select sales.item_id, item.name, sum(sales.count * item.price) from sales, item
where sales.item_id = item.item_id group by sales.item_id
```

❖図 3.27 結合とグループ化を組み合わせる例



count	item_id
20	1
15	2
10	3
10	1
25	3

item_id	name	price
1	B5ファイル	500
2	A4ファイル	600
3	B5ケース	800

count	item_id	item_id	name	price	count * price	count * priceのグループ毎の合計
20	1	1	B5ファイル	500	10000	15000
10	1	1	B5ファイル	500	5000	
15	2	2	A4ファイル	600	9000	9000
10	3	3	B5ケース	800	8000	28000
25	3	3	B5ケース	800	20000	

item_id	name	count * priceのグループ毎の合計
1	B5ファイル	15000
2	A4ファイル	9000
3	B5ケース	28000

3.5 副問い合わせと相関副問い合わせ

1 つの select 文の中に、別の select 文を入れ子に書いて、複雑な条件で行を取り出すこともできます。本節では、select 文を入れ子にする「**副問い合わせ**」や「**相関副問い合わせ**」を紹介します。

副問い合わせの概要

場合によっては、「**select 文で取り出した結果を、他の select 文の中で使いたい**」というようなことがあります。

このような場合、select 文の中にさらに select 文を書くことで、1 つの select 文にまとめることができます。この「**select 文の中にさらに select 文がある**」という形のことを、「**副問い合わせ**」(英語では「Sub Query」) と呼びます。

副問い合わせでは、「select 文の中に select 文がある」という形になりますが、内側の select 文が副問い合わせです。一方、外側の select 文は「**主問い合わせ**」と呼びます。

副問い合わせを使うことで、より複雑な問い合わせを行うことができ、データを様々な形で活用することができます。

where 句の条件に副問い合わせを使う

副問い合わせの使い方はいくつかあります。中でも、「**where 句の条件に副問い合わせを使う**」というパターンをよく使います。

副問い合わせを使う事例

図 3.28 は、販売管理を行うテーブルの例です。**表 3.8** のようなデータを正規化して、「order_main」と「order_sub」の 2 つのテーブルに分けた形になっています。

ここで、「商品番号が 101 番か 102 番の商品が売れた注文の、注文番号と注文日を取り出す」という処理を行いたいとします。

表 3.8 で見ると、注文番号 1 番の注文では、商品番号 101 番と 102 番の商品が売れていて、「101 番の商品が売れた」という条件を満たしていますので、この注文の行は結果に取り出されます。同様に、注文番号 3 番の注文でも、商品番号 101 番の商品が売れていますので、この注文の行も結果に取り出されます。

一方、注文番号が 2 番の商品では、商品番号 103 番の商品しか売れていません。したがって、この注文の行は結果に取り出されません。

❤図 3.28　販売管理を行うテーブルの例

order_mainテーブル

order_id	order_date
1	2009/4/1
2	2009/4/12
3	2009/4/25

order_subテーブル

order_id	item_id	count
1	101	20
1	102	15
2	103	10
3	101	10
3	103	25

❤表 3.8　図 3.28 の元となったデータ

注文番号 (order_id)	注文日 (order_date)	商品番号 (item_id)	個数 (count)
1	2009/4/1	101	20
		102	15
2	2009/4/12	103	10
3	2009/4/25	101	10
		103	25

事例の考え方

この事例は、次のような考え方で処理することができます。

まず、「order_sub」テーブルから、「item_id」列の値が 101 か 102 になっている行を取り出し、各行の「order_id」列の値を取り出します。ただし、101 番と 102 番の商品が両方売れた注文では、「order_id」列に同じ値が 2 回取り出されますので、それらの重複は取り除きます。

例えば、**図 3.28** の「order_sub」テーブルだと、「order_id」列が 1 番の行が 2 行あり、それぞれの「item_id」列の値は 101 と 102 です。したがって、この 2 行はどちらも条件に合致しますので、「order_id」列が 1 の行が 2 回取り出されます。

そこで、これらの重複を取り除きます。

この処理によって、商品番号 101 番か 102 番の商品が売れた注文の、注文番号が分かります。この処理を select 文で書くと、次のようになります（**図 3.29** の中段）。

```
select distinct order_id from order_sub where item_id = 101 or item_id = 102
```

ここまでで、「『商品番号 101 番か 102 番の商品が売れた注文』の注文番号」が分かりました。次に、その情報を利用して、「order_main」テーブルから、それらの注文の注文番号と注文日（「order_id」列と「order_date」列を取り出せば、求めたい結果を得ることができます。

図 3.29 の中段を見ると、「『商品番号 101 番か 102 番の商品が売れた注文』の注文番号」は、1 番と 3 番であることが分かります。したがって、次の select 文を実行すると、それらの注文の注文番号と注文日を得ることができます（**図 3.29** の下段）。

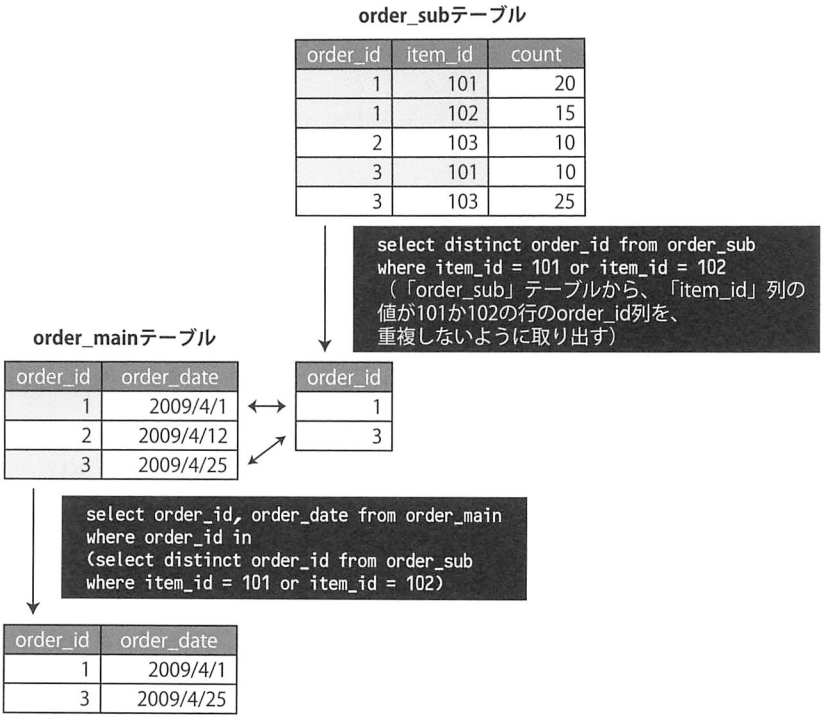
```
select order_id, order_date from order_main where order_id in (1, 3)
```

この select 文の「1, 3」は、1 つ目の select 文の結果によって得られたものです。そこで、この「1, 3」を、1 つ目の select 文に丸ごと置き換え、次の select 文を得ます。

```
select order_id, order_date from order_main where order_id in
(select distinct order_id from order_sub where item_id = 101 or item_id = 102)
```

この select 文は、where 句で条件を指定する箇所に、他の select 文の結果を使う形になっていて、副問い合わせの一例になっています。

◆図 3.29 事例①を処理する際の考え方



副問い合わせの結果をテーブルのように扱う

ここまでは、「select 文の where 句の条件を、別の select 文の結果で表す」という副問い合わせを取り上げました。ただ、副問い合わせは他の箇所でも使うことができます。

例えば、「**副問い合わせの結果を、1 つのテーブルのように扱う**」ということが考えられます。

通常の select 文は、次のように、from 句にテーブル名を書きます。

```
select 列名 , 列名 , … from テーブル名 …
```

ここで、「テーブル名」の箇所を、次のように副問い合わせに置き換えます。すると、副問い合わせの結果をテーブルとみなし、そのテーブルに対してさらに条件

を指定して、行を取り出すことができます。

```
select 列名 , 列名 , … from (
  select 列名 , 列名 , … from テーブル名 …
) …
```

118ページの事例を書き換える

ここで、118ページの事例を、**「副問い合わせの結果を、1つのテーブルのように扱う」**という考え方で捉えなおしてみます。

121ページの**図3.29**では、副問い合わせによって、「order_id」列だけのテーブルができています。このテーブルと「order_main」テーブルとから下段の結果を得る処理は、「『order_main』テーブルと、副問い合わせの結果のテーブルとを、『order_id』列で内部結合する」というように考えることもできます(**図3.30**)。

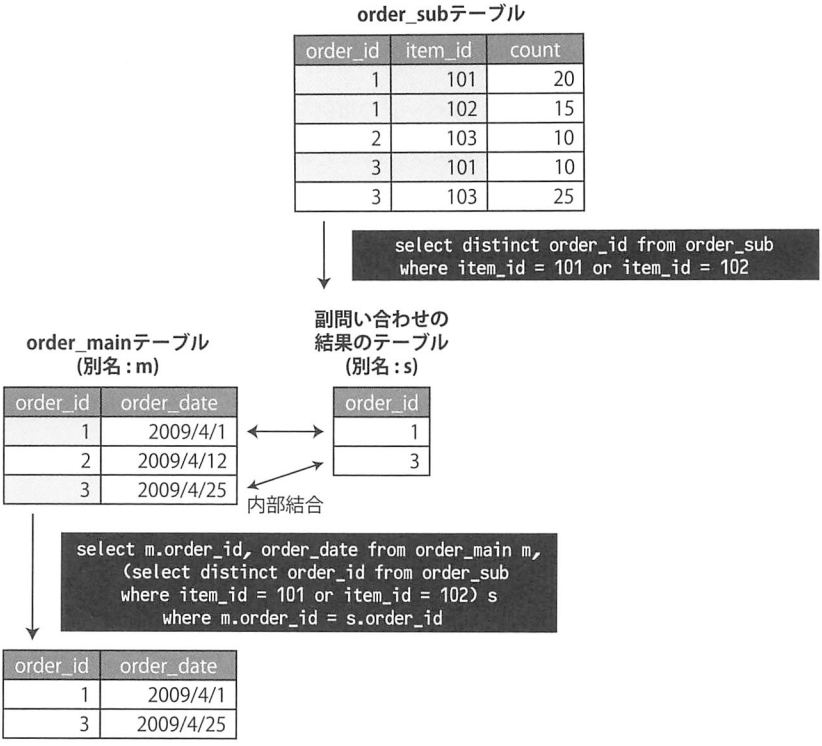
副問い合わせの結果のテーブルに、「s」という名前を付けるとします。また、「order_main」テーブルに「m」という別名を付けるとします。すると、**図3.30**の中段から下段への処理は、「『m』(order_main)テーブルと『s』テーブルを、「order_id」列で内部結合する」と表すことができます。したがって、この処理をselect文で書くと、次のようになります。

```
select order_id, order_date from order_main m, s where m.order_id = s.order_id
```

ここで、from句の「s」の部分を、元の副問い合わせに置き換えます。これで、118ページの事例は、次のselect文でも処理できることが分かります。

```
select m.order_id, order_date from order_main m,
  (select distinct order_id from order_sub
   where item_id = 101 or item_id = 102) s
where m.order_id = s.order_id
```

◆図 3.30 118 ページの事例を、「副問い合わせの結果を、1 つのテーブルのように扱う」という考え方で捉えなおす



相関副問い合わせの概要

副問い合わせの一種として、「**相関副問い合わせ**」があります。

相関副問い合わせも、通常の副問い合わせと同様に、select 文の中に別の select 文がある形になります。ただ、select 文の処理を行う順序が異なります。

通常の副問い合わせでは、次の手順で処理が行われます。

① 副問い合わせの処理を行って、結果を求めます。
② ①の結果を、主問い合わせの中で利用します。

一方、相関副問い合わせは、次のような手順で処理を行います。

① 主問い合わせから、1 行ずつ行を取り出します。
② ①の各行に対して、副問い合わせを実行します。

相関副問い合わせの例

相関副問い合わせの例として、118 ページの事例を相関副問い合わせに置き換えることを考えます。この事例は、次のように処理することもできます。

①「order_main」テーブルから、1 行ずつ順に取り出します。
②①で取り出した行から、「order_id」列の値を得ます。
③「order_sub」テーブルの中で、「order_id」列の値が②と等しく、かつ「item_id」列の値が 101 か 102 の行を探します（この処理が副問い合わせにあたります）。
④③で行が見つかれば、①で取り出した行の「order_id」列と「order_date」列を、結果として出力します（この処理が主問い合わせにあたります）。
⑤③で行が見つからなかったら、①で取り出した行を結果に出力しません。

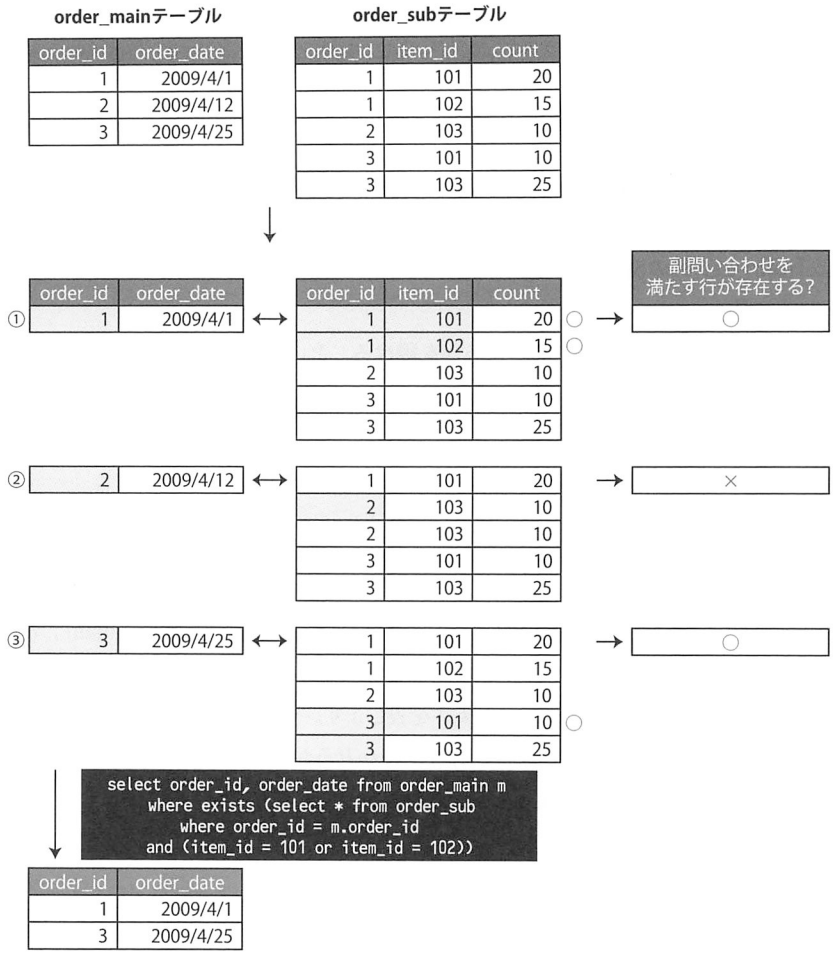
この相関副問い合わせを図で表すと、**図 3.31** のようになります。まず、「order_main」テーブルから 1 行目（「order_id」列が 1 の行）を取り出します。そして、「order_sub」テーブルの中で、「order_id」列の値が「order_main」テーブルの「order_id」（1 番）と同じで、かつ「item_id」列の値が 101 か 102 の行を探します。これが副問い合わせです。

すると、**図 3.31** の①の部分のように、「order_sub」テーブルの中に、副問い合わせの条件を満たす行が 2 行あります（「order_sub」テーブルの左端に「○」がついている行）。

同様に、「order_main」テーブルの 2 行目や 3 行目についても、副問い合わせを順に実行していきます。すると、「order_main」テーブルの 2 行目に対する副問い合わせでは、「order_sub」テーブルには条件を満たす行がありません。一方、「order_main」テーブルの 3 行目に対する副問い合わせでは、「order_sub」テーブルには条件を満たす行があります。

このようにして副問い合わせを行って、「order_sub」テーブルに条件を満たす行が存在したときだけ、「order_main」テーブルの対応する行を取り出します。その結果、**図 3.31** の最下段のような結果を得ることができます。

◆図 3.31 相関副問い合わせの例



相関副問い合わせを行う select 文

図 3.31 の処理を、実際の select 文で表してみます。

まず、主問い合わせは「order_main」テーブルから、「order_id」列と「order_date」を取り出すことです。この処理は、次のように書くことができます。

```
select order_id, order_date from order_main m
```

なお、テーブル名の後に「m」とありますが、これはテーブルに「m」という別名を付けることを意味します（106ページ参照）。

一方、副問い合わせは、「order_sub」テーブルに、次の条件をともに満たす行が存在するかどうかを調べることです。

①「order_main」テーブルの各行の「order_id」列の値と、「order_sub」テーブルの「order_id」列の値が等しい
②「item_id」列の値が101か102

「条件を満たす行が存在するかどうかを調べる」という処理を行うには、まず「条件を満たす行を取り出す」という処理を行います。これは、次のselect文で行います。

```
select * from order_sub where order_id = m.order_id and (item_id = 101 or
item_id = 102)
```

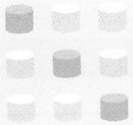
「order_id = m.order_id」の条件があります。主問い合わせで「order_main」テーブルに「m」の別名を付けましたので、この部分が①の条件を表します。また、andから後のカッコの部分が、②の条件を表します。

次に、このselect文を満たす行が存在するかどうか、ということを調べます。これは、上記のselect文全体を、「exists(……)」という演算子で囲むことで表すことができます。

最後に、主問い合わせにwhere句を作り、その条件として副問い合わせを使うようにします。結果として次のselect文が出来上がります。

```
select order_id, order_date from order_main m where exists (
  select * from order_sub where order_id = m.order_id and (item_id = 101 or
  item_id = 102)
  )
```

3.6 テーブルの作成と行の挿入等の操作



SQL には、テーブルを作成する命令や、テーブルに行を挿入したりするための命令もあります。この節では、それらの命令を紹介します。

テーブルの作成――― create table 命令

データベースに**テーブルを作成する**には、「**create table**」という命令を使います。create table 命令は、RDBMS ごとに書き方に差がありますが、大筋では**リスト 3.1** のような書き方をします。

「テーブル名」と「列名」で、作成するテーブルの名前と、その中に作成する列の名前を指定します。また、個々の列に保存するデータの種類を、「データ型」のところに指定します。

RDBMS によって、扱えるデータ型に差がありますが、基本的には数値／文字／日付／バイナリデータ（画像など）といったデータを扱うことができます。

リスト 3.1　テーブルの作成

```
create table テーブル名 (
    列名 データ型,
    列名 データ型,
    …
    列名 データ型
)
```

create table 命令の例

例えば、データベースに**表 3.9** のような構造の「people」テーブルを作りたいとします。また、このデータベースのシステムでは、データ型を**表 3.10** のように表すとします。この場合、テーブルを作るための create table 文は、**リスト 3.2** のように書きます。

❤表 3.9　作成するテーブル

列	データの種類
id	整数
name	文字列（最大 30 バイト）
birthday	日付
photo	バイナリ

❤表 3.10　このデータベースで使えるデータ型

データの種類	データ型
整数	integer
文字列	char(バイト)
日付	date
バイナリ	blob

リスト 3.2　表 3.9 のテーブルを作成する create table 文

```
create table people (
    id integer,
    name char(30),
    birthday date,
    photo blob
)
```

行の作成――― insert 命令

テーブルに行を作成するには、「insert」という命令を使います。

行を新規作成して値を設定する

行のすべての列に値を設定するには、次のように書きます。

```
insert into テーブル名 values ( 値 1, 値 2, …, 値 n)
```

また、行の一部の列に値を設定する場合、次のように列名を指定することもできます。

```
insert into テーブル名 ( 列名 1, 列名 2, …, 列名 n) values ( 値 1, 値 2, …, 値 n)
```

例えば、「address」というテーブルがあり、「id」「name」「pref」「address」の 4 つの列があるとします。このテーブルに、各列の値が**表 3.11** のような行を追加したい場合、次のように書きます(**図 3.32**)。

```
insert into address values (1, '山田太郎', '東京都', '新宿区市谷左内町 XX-XX')
```

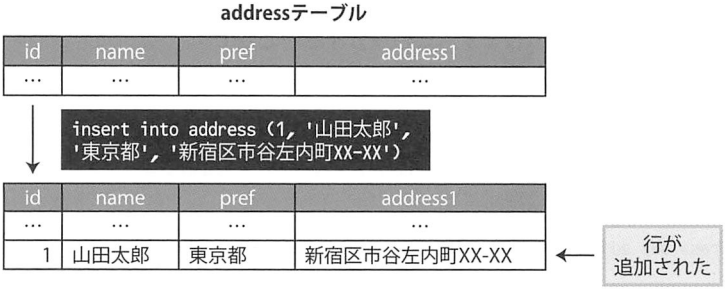
また、行を追加して、id 列以外の 3 つの列に、**表 3.11** の値を入れたいとします。この場合は、select 文を次のように書き、列名と値を対応させます。

```
insert into address (name, pref, address1) values ('山田太郎', '東京都', '
新宿区市谷左内町 XX-XX')
```

◆表 3.11　行に設定する値

列	値
id	1
name	山田太郎
pref	東京都
address1	新宿区市谷左内町 XX-XX

◆図 3.32　insert 命令のイメージ



既存のテーブルから行をコピーする

select 文を使って既存のテーブルの行を取り出し、それらの行を他のテーブルにコピーしたい場合もあります。これは、次のような insert 文で行うことができます。

①列名を指定しない場合

```
insert into テーブル名 select 文
```

②列名を指定する場合

```
insert into テーブル名 ( 列名 1, 列名 2, …, 列名 n) select 文
```

例えば、次の insert 文を実行すると、「item」テーブルから、「name」列の先頭が「B5」になっている行を読み込み、それらを「item_copy」テーブルにコピーすることができます。

```
insert into item_copy select * from item where name like 'B5%'
```

行の内容の変更――― update 命令

テーブル内の既存の行を書き換えたいこともあります。これは「**update**」という命令で行うことができます。

```
update テーブル名 set 列名 1 = 値 1, 列名 2 = 値 2, … where 条件
```

「**条件**」の部分で、書き換えの対象とする行を指定します。条件を指定しないと、すべての行が書き換えられてしまいますので、注意が必要です。また、条件の指定方法を間違うと、1 行だけ書き換えるつもりだったのに、複数行が書き換えられてしまう、といったことも起こり得ます。この点も注意が必要です。

例えば、**図 3.33** の上半分のような「address」テーブルがあるとします。ここで、山田さんが引っ越して住所が「渋谷区宇田川町 XX-XX」に変わるとします。この場合、次のような update 文を実行して、住所を変更します(**図 3.33**)。

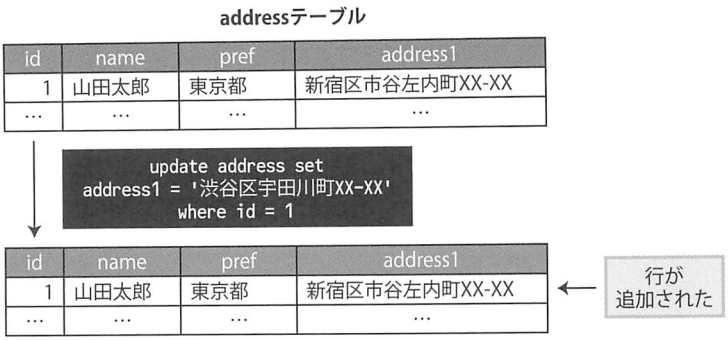
```
update address set address1 = '渋谷区宇田川町 XX-XX' where id = 1
```

この例では、書き換える行は 1 行だけですので、**その行を間違いなく選ぶ**ような条件が必要です。

ここでは、条件として「id = 1」を指定しています。一般に、id 列には個々の行を識別するための値を入れますので、特定の 1 行を間違いなく指定するのに使うこ

とができます。

◆図 3.33 update 命令の例



行の削除——delete 命令

既存の行をテーブルから削除することもできます。これは「delete」という命令で行います。

```
delete from テーブル where 条件
```

update 命令と同様に、「**条件**」の部分で、削除する対象の行を指定します。**条件を指定しないと、テーブルのすべての行が削除されてしまいます**ので、注意が必要です。また、条件を間違って指定すると、削除するつもりではなかった行まで削除されたり、逆に削除しようとしていた行が削除されなかったりすることがあり得ますので、この点も注意が必要です。

例えば、**図 3.34** の上半分のテーブルから、「山田太郎」さんの行を削除するには、次のようにすることが考えられます。update の例と同様に、削除するのは 1 行だけなので、**間違いなく削除できるように、id 列で条件を指定**しています。

```
delete from address where id = 1
```

❤図 3.34　delete 命令の例

addressテーブル

id	name	pref	address1
1	山田太郎	東京都	新宿区市谷左内町XX-XX
2	鈴木花子	神奈川県	横浜市西区中央XX-XX
…	…	…	…

↓ `delete from address where id = 1`

id	name	pref	address1
2	鈴木花子	神奈川県	横浜市西区中央XX-XX
…	…	…	…

← 行が削除された