東京ITスクール
TOKYO IT SCHOOL

集合演算

目次

1. はじめに _____ 1
2. テーブルの足し算と引き算 _____ 1
3. 結合 _____ 10

1. はじめに

これまでは、主に 1 つだけのテーブルを使う SQL 文を書いてきました。ここでは、2 つ以上のテーブルを使いたい場合の SQL 文について学びます。テーブルに対し行方向（縦）に作用する集合演算子と、列方向（横）に作用する結合を覚えることで、複数のテーブルに分散しているデータを組み合わせて望む結果を選択することが出来るようになります。

2. テーブルの足し算と引き算

1　集合演算とは

集合演算とは、レコード同士を足したり引いたりする、いわばレコードの「四則演算」です。集合演算を行なうことで、2 つのテーブルにあるレコードを集めた結果や、共通するレコードを集めた結果、片方のテーブルだけにあるレコードを集めた結果などを得ることが出来ます。そして、このような集合演算を行なうための演算子を「集合演算子」と呼びます。

2　テーブルの足し算 UNION

最初に紹介する集合演算子は、レコードの足し算を行なう UNION（和）です。以下に構文を示します。

（構文）UNION

```
SELECT 文
UNION
SELECT 文
```

実際に使い方を見る前に、サンプルのテーブルを 1 つ追加しましょう。次のような、これまで使ってきた Item（商品）テーブルと同じレイアウトで、テーブル名だけが異なる「Item2（商品 2）」というテーブルを作ります。

(サンプルコード) Item2 テーブル

```
CREATE TABLE Item2
(item_id          CHAR(4)          NOT NULL,
 item_name        VARCHAR(100)     NOT NULL,
 category_id      INTEGER          NOT NULL,
 sel_price        INTEGER          ,
 pur_price        INTEGER          ,
 reg_date         DATE             ,
 PRIMARY KEY (item_id));
```

Item2 テーブルには、以下の 5 レコードを登録します。商品 ID（item_id）の「0001」～「0003」までは、これまでの Item テーブルと同じ商品のデータを持っていますが、ID「0009」のマフラーと「0010」のは、Item テーブルに存在しない商品です。

(サンプルコード) Item2 テーブルへの登録

```
INSERT INTO Item2 VALUES ('0001', 'シャツ', 1, 1000, 500, '2009-09-20');
INSERT INTO Item2 VALUES ('0002', 'ホッチキス', 2, 500, 320, '2009-09-11');
INSERT INTO Item2 VALUES ('0003', 'セーター', 1, 4000, 2800, NULL);
INSERT INTO Item2 VALUES ('0009', 'マフラー', 1, 800, 500, NULL);
INSERT INTO Item2 VALUES ('0010', '鍋', 3, 2000, 1700, '2009-09-20');

COMMIT;
```

それでは、準備が出来たところで、さっそくこの 2 つのテーブルを「Item テーブル＋Item2 テーブル」というように足し算してみましょう。以下に例を示します。

(サンプルコード) UNION

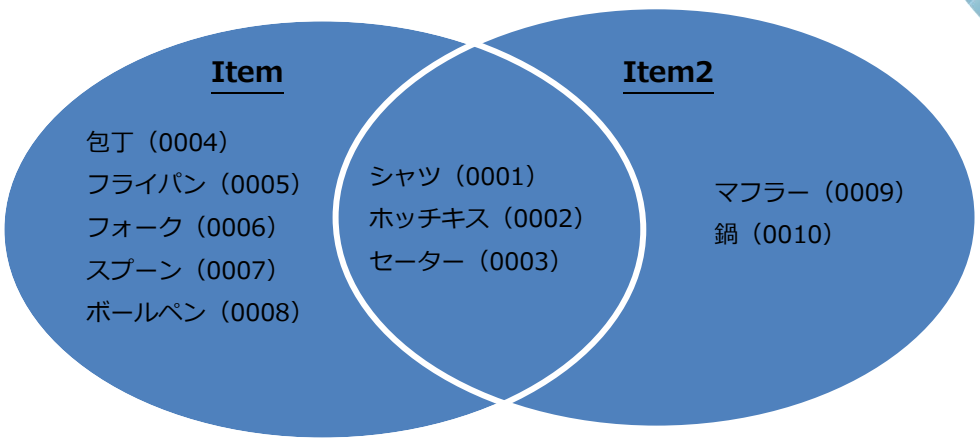
```
SELECT item_id, item_name
    FROM Item
UNION
SELECT item_id, item_name
    FROM Item2;
```

(実行結果)

```
ITEM_ID   ITEM_NAME
---------- --------------------
0001       シャツ
0002       ホッチキス
0003       セーター
0004       包丁
0005       フライパン
0006       フォーク
0007       スプーン
0008       ボールペン
0009       マフラー
0010       鍋

10 行選択されました
```

「和集合」のイメージを以下に示します。



※（）内の数字は商品 ID を表わします。

商品 ID「0001」～「0003」の 3 つのレコードはどちらのテーブルにも存在していたので、素直に考えると重複して結果に出てくるように思うかもしれませんが、UNION に限らず集合演算子は、通常は重複行が排除されます。

3　集合演算の注意事項

この重複行を結果に出すことも可能ですが、その前に、集合演算子を使うときの一般的な注意事項を学んでおきましょう。これは UNION に限らず、この後で学習する全ての演算子に当てはまる注意事項です。

■注意事項①　__ 演算対象となるレコードの列数は同じであること
例えば、次のように、片方の列数が 2 列なのに、片方が 3 列という足し算を行なうことは出来ません。

■注意事項②　__ 足し算の対象となるレコードの列のデータ型が一致していること
左から数えて同じ位置にある列は、同じデータ型である必要があります。また、どうしても違うデータ型の列を使いたい場合は、型変換の関数 CAST を使うことで集合演算子を使用可能です。

■注意事項③　__ ORDER BY 句は 1 つだけ
UNION で足せる SELECT 文は、どんなものでもかまいません。これまでに学んだ WHERE、GROUP BY、HAVING といった句を使うことも出来ます。ただし、ORDER BY 句だけは、全体として 1 つ最後につけられるだけです。

4　重複行を残す集合演算 ALL オプション

UNION の結果から重複行を排除したくない場合は、ALL オプションを使用します。ALL オプションは UNION の後ろに「ALL」というキーワードを追加するのみです。この ALL オプションは、UNION 以外の集合演算子でも同様に使えます。以下に例を示します。

（サンプルコード）重複行を排除しない UNION

```
SELECT item_id, item_name
    FROM Item
UNION ALL
SELECT item_id, item_name
    FROM Item2;
```

（実行結果）

```
ITEM_ID   ITEM_NAME
--------- --------------------
0001      シャツ
0002      ホッチキス
0003      セーター
0004      包丁
0005      フライパン
0006      フォーク
0007      スプーン
0008      ボールペン
0001      シャツ
0002      ホッチキス
0003      セーター
0009      マフラー
0010      鍋

13 行選択されました
```

5　テーブルの共通部分の選択 INTERSECT

2 つのレコード集合の共通部分を選択する場合は、INTERSECT を使用します。構文の形式は、UNION と同じです。

（構文）INTERSECT

```
SELECT 文
INTERSECT
SELECT 文
```

それでは、Item テーブルと Item2 テーブルの共通部分を選択してみましょう。以下に例を示します。

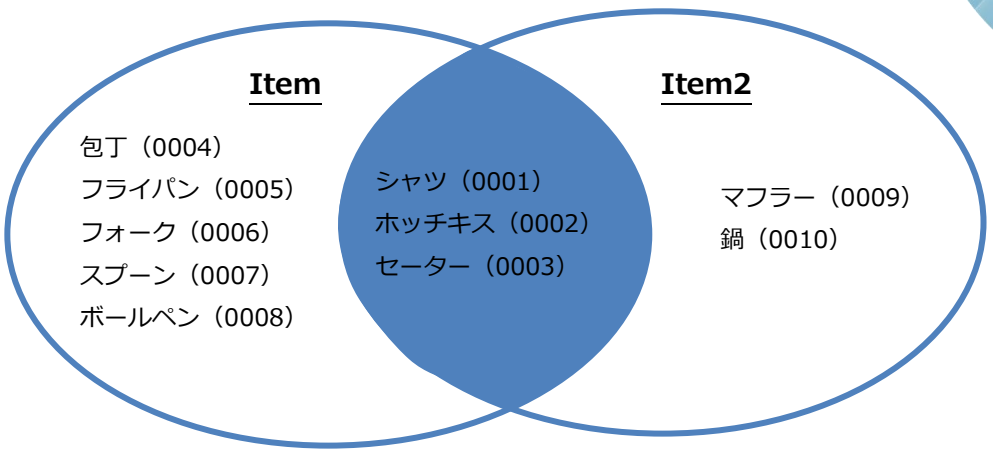
（サンプルコード）INTERSECT

```
SELECT item_id, item_name
    FROM Item
INTERSECT
SELECT item_id, item_name
    FROM Item2;
```

（実行結果）

```
ITEM_ID   ITEM_NAME
---------- ------------------
0001       シャツ
0002       ホッチキス
0003       セーター
```

演算イメージを以下に示します。



※（）内の数字は商品 ID を表わします。

AND が 1 つのテーブルに対して、複数の条件の共通部分を選択するのに対し、INTERSECT は、必ず 2 つのテーブルを使用し、その共通するレコードを選択します。

注意事項は UNION と同じで、「集合演算の注意事項」や「重複行を残す集合演算」の項で説明した通りです。重複行を残したい場合に「INTERSECT ALL」とするのも同じです。

6　レコードの引き算 MINUS

レコードの引き算を行なう場合は MINUS を使用します。こちらも構文の形式は UNION と同じです。

（構文）MINUS

```
SELECT 文
MINUS
SELECT 文
```

それでは、Item テーブルから Item2 テーブルを引いたレコードを選択してみましょう。以下に例を示します。

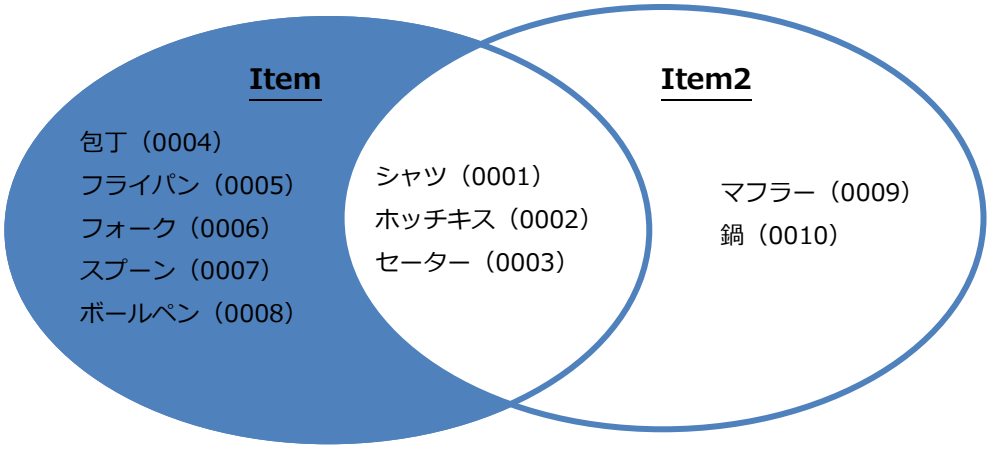
（サンプルコード）MINUS

```
SELECT item_id, item_name
    FROM Item
MINUS
SELECT item_id, item_name
    FROM Item2;
```

（実行結果）

```
ITEM_ID   ITEM_NAME
---------- --------------------
0004       包丁
0005       フライパン
0006       フォーク
0007       スプーン
0008       ボールペン
```

結果では、Item テーブルのレコードから Item2 テーブルのレコードを引いた残りが選択されています。演算イメージを以下に示します。



※（）内の数字は商品 ID を表わします。

MINUS には UNION と INTERSECT にはない、特有の注意点があります。それは、引き算という性質から見れば当前のことですが、どちらからどちらを引くかによって、結果が異なるということです。これは数の引き算でも同じですね。「4＋2」と「2＋4」の結果は同じですが、「4－2」と「2－4」は違う結果になります。従って、先ほどの SQL の Item と Item2 を入れ替えると、以下のようになります。

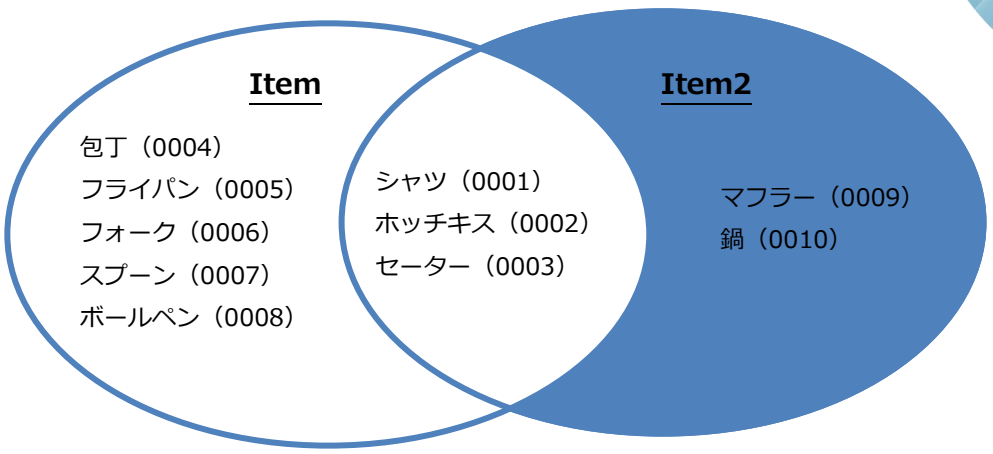
（サンプルコード）MINUS

```
SELECT item_id, item_name
    FROM Item2
MINUS
SELECT item_id, item_name
    FROM Item;
```

（実行結果）

```
ITEM_ID   ITEM_NAME
---------- --------------------
0009       マフラー
0010       鍋
```

演算イメージを以下に示します。



※（）内の数字は商品 ID を表わします。

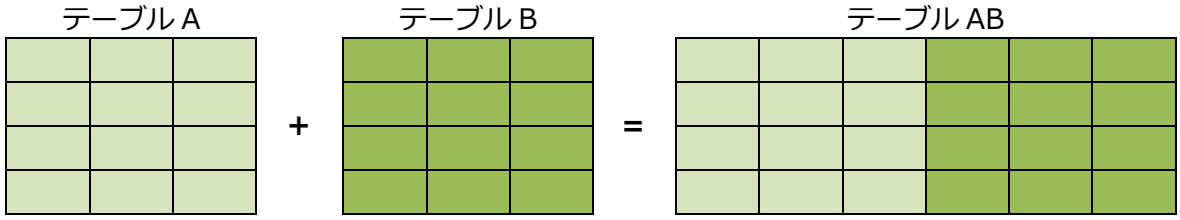
3. 結合

1　結合とは

結合（JOIN）という演算は、分かりやすく言うと、別のテーブルから列を持ってきて「列を増やす」操作です。この操作が役に立つのは、欲しいデータ（列）が 1 つのテーブルだけからでは選択出来ない場合です。これまでは、基本的に 1 つのテーブルからデータを取り出すことが多かったのですが、実際には、欲しいデータが複数のテーブルに分散されていることが頻繁にあります。そのような場合は、複数のテーブル（3 つ以上でも構いません）からデータを選択するということが可能です。

また、意外かもしれませんが同じテーブル同士を結合させることも可能です。その場合は、1 つのテーブルがあたかも別々のテーブルのように振る舞います。

結合のイメージは以下のようになります。



2　学習用テーブルの確認

まずは学習用のテーブルを確認しましょう。使用するテーブルは Item テーブルと ShopItem テーブルです。ShopItem テーブルの作成用の SQL を以下に示します。

Item テーブル

item_id（商品 ID）	item_name（商品名）	category_id（商品分類）	sel_price（販売単価）	pur_price（仕入単価）	reg_date（登録日）
0001	シャツ	1	1000	500	2009-09-20
0002	ホッチキス	2	500	320	2009-09-11
0003	セーター	1	4000	2800	NULL
0004	包丁	3	3000	2800	2009-09-20
0005	フライパン	3	6800	5000	2009-01-15
0006	フォーク	3	500	NULL	2009-09-20
0007	スプーン	3	880	790	2008-04-28
0008	ボールペン	2	100	NULL	2009-11-11

ShopItem テーブル

shop_id（店舗 ID）	shop_name（店舗名）	item_id（商品 ID）	quantity（数量）
000A	東京	0001	30
000A	東京	0002	50
000A	東京	0003	15
000B	仙台	0002	30
000B	仙台	0003	120
000B	仙台	0004	20
000B	仙台	0006	10
000B	仙台	0007	40
000C	大阪	0003	20
000C	大阪	0004	50
000C	大阪	0006	90
000C	大阪	0007	70
000D	福岡	0001	100

ShopItem テーブル作成用の SQL 文を以下に示します。

（サンプルコード）ShoItem テーブル

```
CREATE TABLE ShopItem
(shop_id          CHAR(4)          NOT NULL,
 shop_name        VARCHAR(200)     NOT NULL,
 item_id          CHAR(4)          NOT NULL,
 quantity         INTEGER          NOT NULL,
PRIMARY KEY (shop_id, item_id));
```

ShopItem テーブルには、以下のレコードを登録します。

（サンプルコード）ShopItem テーブルへの登録

```
INSERT INTO ShopItem (shop_id, shop_name, item_id, quantity) VALUES ('000A', '東京', '0001', 30);
INSERT INTO ShopItem (shop_id, shop_name, item_id, quantity) VALUES ('000A', '東京', '0002', 50);
INSERT INTO ShopItem (shop_id, shop_name, item_id, quantity) VALUES ('000A', '東京', '0003', 15);
INSERT INTO ShopItem (shop_id, shop_name, item_id, quantity) VALUES ('000B', '仙台', '0002', 30);
INSERT INTO ShopItem (shop_id, shop_name, item_id, quantity) VALUES ('000B', '仙台', '0003', 120);
INSERT INTO ShopItem (shop_id, shop_name, item_id, quantity) VALUES ('000B', '仙台', '0004', 20);
INSERT INTO ShopItem (shop_id, shop_name, item_id, quantity) VALUES ('000B', '仙台', '0006', 10);
INSERT INTO ShopItem (shop_id, shop_name, item_id, quantity) VALUES ('000B', '仙台', '0007', 40);
INSERT INTO ShopItem (shop_id, shop_name, item_id, quantity) VALUES ('000C', '大阪', '0003', 20);
INSERT INTO ShopItem (shop_id, shop_name, item_id, quantity) VALUES ('000C', '大阪', '0004', 50);
INSERT INTO ShopItem (shop_id, shop_name, item_id, quantity) VALUES ('000C', '大阪', '0006', 90);
INSERT INTO ShopItem (shop_id, shop_name, item_id, quantity) VALUES ('000C', '大阪', '0007', 70);
INSERT INTO ShopItem (shop_id, shop_name, item_id, quantity) VALUES ('000D', '福岡', '0001', 100);

COMMIT;
```

3　結合の考え方

それでは、内部結合を学習していきます。まずは、2 つのテーブルに存在する列をもう一度整理してみましょう。

	Item	ShopItem
商品 ID	○	○
商品名	○	
商品分類	○	
販売単価	○	
仕入単価	○	
登録日	○	
店舗 ID		○
店舗名		○
数量		○

この表を見ると商品ID列だけがどちらのテーブルにも存在することが分かります。結合とは、この共通列を使って 2 つのテーブルを関連付けることです。

これによって、ShopItem テーブルのみでは分からない商品の名前や単価を Item テーブルから取得することが出来ます。

ShopItem テーブル

shop_id	shop_name	item_id	quantity
000A	東京	0001	30
000A	東京	0002	50
000A	東京	0003	15
000B	仙台	0002	30
000B	仙台	0003	120
000B	仙台	0004	20
000B	仙台	0006	10
000B	仙台	0007	40
000C	大阪	0003	20
000C	大阪	0004	50
000C	大阪	0006	90
000C	大阪	0007	70
000D	福岡	0001	100

＋

Item テーブル

item_id	item_name	sel_price
0001	シャツ	1000
0002	ホッチキス	500
0003	セーター	4000
0002	ホッチキス	500
0003	セーター	4000
0004	包丁	3000
0006	フォーク	500
0007	スプーン	880
0003	セーター	4000
0004	包丁	3000
0006	フォーク	500
0007	スプーン	880
0001	シャツ	1000

4　内部結合（INNER JOIN）

それでは、内部結合を学習していきます。内部結合では、商品 ID のような**結合のキーになる列の値が両方のテーブルに存在する行**を取得します。

まずは、結合の構文を学ぶ前にこのような結果となる SQL 文を実行してみましょう。SELECT 文は以下の通りです。

(サンプルコード) ShopItem テーブルと Item テーブルの内部結合

```
SELECT
   SI.shop_id,
   SI.shop_name,
   SI.item_id,
   I.item_name,
   I.sel_price,
   SI.quantity
FROM ShopItem SI INNER JOIN Item I
ON SI.item_id = I.item_id;
```

(実行結果)

SHOP_ID	SHOP_NAME	ITEM_ID	ITEM_NAME	SEL_PRICE	QUANTITY
000A	東京	0001	シャツ	1000	30
000A	東京	0002	ホッチキス	500	50
000A	東京	0003	セーター	4000	15
000B	仙台	0002	ホッチキス	500	30
000B	仙台	0003	セーター	4000	120
000B	仙台	0004	包丁	3000	20
000B	仙台	0006	フォーク		10
000B	仙台	0007	スプーン	790	40
000C	大阪	0003	セーター	4000	20
000C	大阪	0004	包丁	3000	50
000C	大阪	0006	フォーク		90
000C	大阪	0007	スプーン	790	70
000D	福岡	0001	シャツ	1000	100

13 行選択されました

それでは SQL 文の中身を見ていきましょう

●FROM 句

```
FROM ShopItem SI INNER JOIN Item I
```

FROM 句に、ShopItem と Item という 2 つのテーブルを記述しています。内部結合をする場合は「INNER JOIN」キーワードの後に結合するテーブル名を記述します。結合するテーブルが 3 つ以上ある場合は、更に「INNER JOIN」キーワードの後に結合するテーブル名を記述します。

また、テーブル名の後の「SI」や「I」は別名です。必須ではありませんが、結合の際は別名を付けるのが一般的です。

●ON 句

```
ON SI.item_id = I.item_id;
```

ON 句では、結合するテーブルを結びつける列（結合キー）を指定します。この場合は商品 ID（item_id）が結合キーです。いわば ON は、結合条件専用の WHERE のような役割だといえます。WHERE 句と同じように、複数の結合キーを指定するために、AND、OR を使うことも出来ます。

ON 句は、内部結合を行なう場合は記述が必須です。かつ、書く場所は FROM と WHERE の間でなくてはなりません。

●SELECT 句

```
SELECT
  SI.shop_id,
  SI.shop_name,
  SI.item_id,
  S.item_name,
  S.sel_price,
  SI.quantity
```

SELECT 句では、SI.shop_id や I.item_name のように、<テーブルの別名>.<列名>という記述の仕方をしています。これは、テーブルが 1 つだけの場合と違って、結合の場合、どの列をどのテーブルから持ってきているかが不明瞭になりやすいため、それを防ぐための措置です。構文としては、この記述をしなければならないのは、2 つのテーブルに存在している列（ここでは item_id）のみで、他の列は「shop_id」のように列名だけ書いてもエラーにはなりません。しかし、前述のように混乱を避けるという理由から、結合の場合は SELECT 句の全ての列を<テーブルの別名>.<列名>の書式で書くようにすることが望ましいでしょう。

以上のことをまとめると、構文としては以下のようになります。

(構文) 内部結合

```
SELECT <列名>
FROM <テーブル名> INNER JOIN <結合するテーブル名>
ON <テーブル名>.<列名> = <テーブル名>.<列名>
```

また、内部結合と WHERE 句を組み合わせることで、結合結果から更に絞り込むことが出来ます。その場合、WHERE 句は ON 句の後に記述します。WHERE 句以外にも GROUP BY 句、HAVING 句、OREDER BY 句なども同様に使用出来ます。

5　外部結合（LEFT OUTER JOIN、RIGHT OUTER JOIN）

続いて外部結合を学習しましょう。内部結合が、「商品 ID のような結合のキーになる列の値が両方のテーブルに存在する行を取得」するのみ対し、外部結合では、結合キーの列の値の有無に関わらず、**基準となっているテーブルの全行を取得**します。その際、表示不可能なデータ（紐づけ先が無い）は NULL で表示されます。

構文は基本的に内部結合のものとほぼ変わりません。違う点は、FROM 句の中身です。「INNER JOIN」が内部結合であったの対し、外部結合では、「LEFT OUTER JOIN」、または「RIGHT OUT JOIN」と記述します。LEFT の場合は、左外部結合と呼び、結合元のテーブルが基準のテーブルとなり必ず全行選択されます。（WHERE 句がある場合は、そこから絞り込まれます。）RIGHT の場合は、右外部結合と呼び、逆に結合先のテーブルが基準となります。

(構文) 外部結合

```
SELECT <列名>
FROM <テーブル名> LEFT[RIGHT] OUTER JOIN <結合するテーブル名>
ON <テーブル名>.<列名> = <テーブル名>.<列名>
```

それでは、Item テーブルを基準に ShopItem テーブルを左外部結合してみましょう。（ちなみにこれは、ShopItem を基準に Item テーブルを右外部結合することと同じです。）

(サンプルコード) ShopItem テーブルと Item テーブルの外部結合

```
SELECT
   SI.shop_id,
   SI.shop_name,
   SI.item_id,
   I.item_name,
   I.sel_price,
   SI.quantity
FROM Item I LEFT OUTER JOIN ShopItem SI
ON SI.item_id = I.item_id;
```

(実行結果)

SHOP_ID	SHOP_NAME	ITEM_ID	ITEM_NAME	SEL_PRICE	QUANTITY
000A	東京	0001	シャツ	1000	30
000A	東京	0002	ホッチキス	500	50
000A	東京	0003	セーター	4000	15
000B	仙台	0002	ホッチキス	500	30
000B	仙台	0003	セーター	4000	120
000B	仙台	0004	包丁	3000	20
000B	仙台	0006	フォーク		10
000B	仙台	0007	スプーン	790	40
000C	大阪	0003	セーター	4000	20
000C	大阪	0004	包丁	3000	50
000C	大阪	0006	フォーク		90
000C	大阪	0007	スプーン	790	70
000D	福岡	0001	シャツ	1000	100
			フライパン	5000	
			ボールペン		

15 行選択されました

6　3 つ以上のテーブルを使った結合

結合の基本的な形は 2 つのテーブルですが、別に 3 つ以上のテーブルを同時に結出来ない訳ではありません。結合出来るテーブルの数に原理的な制限はありません。

構文としては、FROM 句に JOIN 句、ON 句のセットを結合するテーブルの分だけ記述していくことで、3 つ以上のテーブルの結合が可能です。

次のような商品の在庫を管理するテーブルを作ります。「S001」と「S002」という 2 つの倉庫に商品を保管しているとします。

StockItem（在庫商品）テーブル

house_id（倉庫 ID）	item_id（商品 ID）	Stock（在庫数量）
S001	0001	0
S001	0002	120
S001	0003	200
S001	0004	3
S001	0005	0
S001	0006	99
S001	0007	999
S001	0008	200
S002	0001	10
S002	0002	25
S002	0003	34
S002	0004	19
S002	0005	99
S002	0006	0
S002	0007	0
S002	0008	18

このテーブルを作る SQL 文と、テーブルにデータを登録する SQL 文は以下の通りです。

(サンプルコード) StockItem テーブルの作成とデータ登録

```
--DDL:テーブル作成
CREATE TABLE StockItem
(house_id                CHAR(4)                NOT NULL,
 item_id                 CHAR(4)                NOT NULL,
 stock                   INTEGER                NOT NULL,
 PRIMARY KEY             (house_id, item_id));

--DML:データ登録
INSERT INTO StockItem  (house_id, item_id, stock) VALUES('S001', '0001', 0);
INSERT INTO StockItem  (house_id, item_id, stock) VALUES('S001', '0002',120);
INSERT INTO StockItem  (house_id, item_id, stock) VALUES('S001', '0003',200);
INSERT INTO StockItem  (house_id, item_id, stock) VALUES('S001', '0004',3);
INSERT INTO StockItem  (house_id, item_id, stock) VALUES('S001', '0005',0);
INSERT INTO StockItem  (house_id, item_id, stock) VALUES('S001', '0006',99);
INSERT INTO StockItem  (house_id, item_id, stock) VALUES('S001', '0007',999);
INSERT INTO StockItem  (house_id, item_id, stock) VALUES('S001', '0008',200);
INSERT INTO StockItem  (house_id, item_id, stock) VALUES('S002', '0001',10);
INSERT INTO StockItem  (house_id, item_id, stock) VALUES('S002', '0002',25);
INSERT INTO StockItem  (house_id, item_id, stock) VALUES('S002', '0003',34);
INSERT INTO StockItem  (house_id, item_id, stock) VALUES('S002', '0004',19);
INSERT INTO StockItem  (house_id, item_id, stock) VALUES('S002', '0005',99);
INSERT INTO StockItem  (house_id, item_id, stock) VALUES('S002', '0006',0);
INSERT INTO StockItem  (house_id, item_id, stock) VALUES('S002', '0007',0);
INSERT INTO StockItem  (house_id, item_id, stock) VALUES('S002', '0008',18);

COMMIT;
```

内部結合を行なった SQL 文の FROM 句に、再度 INNER JOIN によって StockItem テーブルを追加しています。

(サンプルコード) 3 テーブルの内部結合

```
SELECT SI.shop_id, SI.shop_name, SI.item_id, I.item_name, I.sel_price,
   SI.quantity, StI.stock
FROM ShopItem SI INNER JOIN Item I ON SI.item_id = I.item_id
INNER JOIN StockItem StI ON SI.item_id = StI.item_id
WHERE StI.house_id = 'S001';
```

(実行結果)

SHOP_ID	SHOP_NAME	ITEM_ID	ITEM_NAME	SEL_PRICE	QUANTITY	STOCK
000D	福岡	0001	シャツ	1000	100	0
000A	東京	0001	シャツ	1000	30	0
000B	仙台	0002	ホッチキス	500	30	120
000A	東京	0002	ホッチキス	500	50	120
000C	大阪	0003	セーター	4000	20	200
000B	仙台	0003	セーター	4000	120	200
000A	東京	0003	セーター	4000	15	200
000C	大阪	0004	包丁	3000	50	3
000B	仙台	0004	包丁	3000	20	3
000C	大阪	0006	フォーク		90	99
000B	仙台	0006	フォーク		10	99
000C	大阪	0007	スプーン	790	70	999
000B	仙台	0007	スプーン	790	40	999

13 行選択されました

7　同じテーブル同士を結合

結合は基本的に異なるテーブル同士で行いますが、1 つのテーブルを 2 つ（またはそれ以上）のテーブルに見立てて結合させることも出来ます。
まずは、以下の検索結果を見てください。

(実行結果)

```
SELECT * FROM Emp;

EMPNO ENAME      JOB              MGR HIREDATE       SAL       COMM       DEPTNO
----- ---------- --------- ---------- -------------- --------- ---------- ----------
 7369 SMITH      CLERK           7902 80-12-17            800                    20
 7499 ALLEN      SALESMAN        7698 81-02-20           1600        300         30
 7521 WARD       SALESMAN        7698 81-02-22           1250        500         30
 7566 JONES      MANAGER         7839 81-04-02           2975                    20
 7654 MARTIN     SALESMAN        7698 81-09-28           1250       1400         30
 7698 BLAKE      MANAGER         7839 81-05-01           2850                    30
 7782 CLARK      MANAGER         7839 81-06-09           2450                    10
 7839 KING       PRESIDENT            81-11-17           5000                    10
 7844 TURNER     SALESMAN        7698 81-09-08           1500          0         30
 7900 JAMES      CLERK           7698 81-12-03            950                    30
 7902 FORD       ANALYST         7566 81-12-03           3000                    20
 7934 MILLER     CLERK           7782 82-01-23           1300                    10

12 行選択されました
```

こちらは、Oracle のサンプルテーブルです。MGR 列には直属上司の EMPNO を格納しています。ただ、EMPNO だけだと誰が上司かが分かりづらいです。恐らく、名前も同時に出したいと思うことでしょう。
このような場合には、自分のテーブルと結合をして上司の名前を表示させると良いでしょう。自分のテーブル同士の結合は特別な記述が必要な訳ではなく、これまでの学習の応用です。
以下に例を示します。

(サンプルコード) 同じテーブル同士の左外部結合

```
SELECT e1.empno, e1.ename, e1.job, e1.mgr, e2.ename mgr_name, e2.job mgr_job
FROM Emp e1 LEFT OUTER JOIN Emp e2 ON e1.mgr = e2.empno
ORDER BY empno;
```

(実行結果)

EMPNO	ENAME	JOB	MGR	MGR_NAME	MGR_JOB
7369	SMITH	CLERK	7902	FORD	ANALYST
7499	ALLEN	SALESMAN	7698	BLAKE	MANAGER
7521	WARD	SALESMAN	7698	BLAKE	MANAGER
7566	JONES	MANAGER	7839	KING	PRESIDENT
7654	MARTIN	SALESMAN	7698	BLAKE	MANAGER
7698	BLAKE	MANAGER	7839	KING	PRESIDENT
7782	CLARK	MANAGER	7839	KING	PRESIDENT
7839	KING	PRESIDENT			
7844	TURNER	SALESMAN	7698	BLAKE	MANAGER
7900	JAMES	CLERK	7698	BLAKE	MANAGER
7902	FORD	ANALYST	7566	JONES	MANAGER
7934	MILLER	CLERK	7782	CLARK	MANAGER

12 行選択されました

このように、同じテーブルに別名を付与して結合させることで、1 つの Emp テーブルがあたかも 2 つあるかのように扱うことが出来るのです。

8　クロス結合（CROSS JOIN）

クロス結合では、結合したテーブルの全ての組み合わせが選択されます。従って、ON 句も使用しません。（指定することが出来ません）例えば結合するテーブルが 13 行のものと 8 行のものであれば、13×8＝104 行の結果が作られるのがクロス結合です。構文は以下の通りです。

(構文) CROSS JOIN

```
SELECT <列名> FROM <テーブル名> CROSS JOIN <結合するテーブル名>
```

9　結合の古い構文

ここまでで紹介した内部結合及び外部結合の構文は、標準 SQL で定められている正式なものです。ただ、結合は古い構文も現場レベルでは未だに使用されているのが現状です。

例えば、内部結合の SELECT 文を古い構文で書き換えると以下のようになります。

(サンプルコード) 古い構文を使った内部結合

```
SELECT SI.shop_id, SI.shop_name, SI.item_id, I.item_name, I.sel_price, SI.quantity
FROM ShopItem SI, Item I
WHERE SI.item_id = I.item_id;
```

(実行結果)

```
SHOP_ID  SHOP_NAME            ITEM_ID  ITEM_NAME              SEL_PRICE            QUANTITY
-------- -------------------- -------- ---------------------- -------------------- --------------------
000A     東京                 0001     シャツ                 1000                 30
000A     東京                 0002     ホッチキス             500                  50
000A     東京                 0003     セーター               4000                 15
000B     仙台                 0002     ホッチキス             500                  30
000B     仙台                 0003     セーター               4000                 120
000B     仙台                 0004     包丁                   3000                 20
000B     仙台                 0006     フォーク                                    10
000B     仙台                 0007     スプーン               790                  40
000C     大阪                 0003     セーター               4000                 20
000C     大阪                 0004     包丁                   3000                 50
000C     大阪                 0006     フォーク                                    90
000C     大阪                 0007     スプーン               790                  70
000D     福岡                 0001     シャツ                 1000                 100

13 行選択されました
```

この書き方でも、結果は先程と全く同じになります。しかも、この構文は一応全ての DBMS で利用出来るので、その意味で方言という訳ではありません。ただ「古い」だけです。