连接查询

```
1   USE purchase;
```

一、笛卡尔积

交叉连接相当于关系的笛卡尔积，如表A与表B进行交叉连接:

`A` :

a	b	c
1	2	3
3	2	1
3	4	5

`B`：

c	d	f
5	6	7
7	6	5

`A CROSS JOIN B`

2*3=6

A.a	A.b	A.c	B.c	B.d	B.f
1	2	3	5	6	7
1	2	3	7	6	5
3	2	1	5	6	7
3	2	1	7	6	5
3	4	5	5	6	7
3	4	5	7	6	5

基本语法:

```
SELECT *
FROM 表1 [CROSS | INNER] JOIN 表2;

-- 或者
SELECT *
FROM 表1, 表2;
```

值得注意的是，一些商业关系型数据库的表并不等于关系，例如MySQL的表中若未设置主键约束，则可以存在重复行，而这对于关系数据库理论中的关系而言是不允许的。此外，在实际应用中一般不会使用交叉连接。原因在于交叉连接将产生大量的中间数据，占用大量内存。一般情况下，建议将交叉连接与选择条件一起构成内连接或者自然连接，以减少中间过程产生的数据量。

示例1：使用交叉连接，查询根类别表（`sort`表）和子类别表（`subsort`表）的所有数据.

```
SELECT *
FROM sort CROSS JOIN subsort limit 35;
```

```
3
4   SELECT *
5   FROM sort, subsort LIMIT 35;
6
7   SELECT *
8   FROM sort INNER JOIN subsort LIMIT 35;
9
10  SELECT *
11  FROM sort JOIN subsort LIMIT 35;
12
13  SELECT count(*) FROM sort;
14
15  SELECT count(*) FROM subsort;
16
17  SELECT count(*) FROM sort, subsort;
```

二、内连接

内连接为双目运算，定义了两表根据对应列的相等关系进行连接操作。如表A与表B进行根据列c进行等值内连接:

`A`:

a	b	c
1	2	3
3	2	1
3	4	5

`B`:

c	d	f
5	6	7
7	6	5

`A INNER JOIN B ON A.c = B.c`

A.a	A.b	A.c	B.c	B.d	B.f
3	4	5	5	6	7

基本语法:

```
SELECT 查询字段
FROM 表1 [INNER | CROSS] JOIN 表2 ON 表1.字段1 = 表2.字段2;
```

结果等价于

```
SELECT 查询字段
FROM 表1 [CROSS | INNER] JOIN 表2
WHERE 表1.字段1 = 表2.字段2;
```

> 注意，字段1可以和字段2相同

示例2：使用等值连接，查询根类别表（`sort`表）和子类别表（`subsort`表）中的根类别名称和子类别名称。

```
SELECT Sort_name, SubSort_name
FROM sort INNER JOIN subsort ON sort.Sort_ID = subsort.Sort_ID;
```

示例3：使用交叉连接，结合`WHERE`条件语句实现上例的内连接查询`sort`表和`subsort`表中的根类别名称和子类别名称。

```
SELECT *
FROM sort, subsort
WHERE sort.Sort_ID = subsort.Sort_ID;
```

三、外连接：`LEFT | RIGHT [OUTER] JOIN`

外连接操作可以视为在内连接的结果基础上，加上左表(left join)或右表(right join)的未包含在内连接结果中的行。

> 注意: MySQL尚未支持全外连接操作。可以利用左外连接和右外连接的并集来替代。

基本语法:

```
SELECT 所查字段
FROM 表1 LEFT | RIGHT [OUTER] JOIN 表2 ON 表1.字段 = 表2.字段
WHERE 条件;
```

`A`：

a	b		c
1	2		3
3	2		1
3	4		5

`B`：

	c	d	f
	5	6	7
	7	6	5

左外连接

```
SELECT * FROM A LEFT JOIN B ON A.C = B.C;
```

A.a	A.b	A.c	B.c	B.d	B.f
1	2	3	NULL	NULL	NULL
3	2	1	NULL	NULL	NULL
3	4	5	5	6	7

右外连接

```
SELECT * FROM A RIGHT JOIN B ON A.C = B.C;
```

A.a	A.b	A.c	B.c	B.d	B.f
3	4	5	5	6	7
NULL	NULL	NULL	7	6	5

全外连接

```
SELECT * FROM A LEFT JOIN B ON A.C = B.C
UNION
SELECT * FROM A RIGHT JOIN B ON A.C = B.C;
```

A.a	A.b	A.c	B.c	B.d	B.f
1	2	3	NULL	NULL	NULL
3	2	1	NULL	NULL	NULL
3	4	5	5	6	7
NULL	NULL	NULL	7	6	5

示例4：对 `sort` 表和 `subsort` 表之进行左连接查询

```
SELECT *
FROM sort LEFT JOIN subsort ON sort.Sort_ID = subsort.Sort_ID
ORDER BY SubSort_name;
```

示例5：在 `sort` 表和 `subsort` 表之间使用右连接查询

```
SELECT *
FROM sort RIGHT JOIN subsort ON sort.Sort_ID = subsort.Sort_ID
ORDER BY Sort_name;
```

示例6：利用左外连接和右外连接实现 `sort` 表和 `subsort` 表的全连接查询

```
SELECT *
FROM sort RIGHT JOIN subsort ON sort.Sort_ID = subsort.Sort_ID
UNION
SELECT *
FROM sort LEFT JOIN subsort ON sort.Sort_ID = subsort.Sort_ID;

SELECT *
FROM sort JOIN subsort
WHERE sort.sort_id = subsor.sort_id OR sort.sort_id IS NULL OR subsort.sort_id IS NULL
```

> `UNION`，`UNION ALL` 都可以实现查询结构的并操作，不同之处在于 `UNION` 严格执行了集合并的概念，即剔除重复行；`UNION ALL` 不剔除重复行，保留两个表中的所有数据，例如：
>
> ```
> SELECT * FROM SORT
> UNION
> SELECT * FROM SORT; -- 去重
>
> SELECT * FROM SORT
> UNION ALL
> SELECT * FROM SORT;  -- 不去重
>
> select a.sort_id, count(*)
> from (SELECT * FROM SORT
>   UNION ALL
>   SELECT * FROM SORT) a
> group by a.sort_id
> order by a.sort_id;
> ```

- 思考：如何在 `MYSQL` 中实现减和交操作?

示例7：在 `sort` 表和 `subsort` 表之间使用内连接查询，然后查询类别名称中有 `'文件'` 的记录，将查询结果按照 `Subsort_ID` 降序排列。

```
SELECT Sort_name, Subsort_ID, SubSort_name
FROM sort INNER JOIN subsort ON sort.Sort_ID = subsort.Sort_ID
WHERE sort_name LIKE '%文件%'
```

```
ORDER BY Subsort_ID DESC;

SELECT a.sort_name, COUNT(b.subsort_id)
FROM sort a JOIN subsort b ON a.sort_id = b.sort_id
WHERE sort_name LIKE '%文件%'
GROUP BY a.sort_name
HAVING COUNT(b.subsort_id) > 5
ORDER BY COUNT(b.subsort_id) DESC
LIMIT 5;

-- 等价于
SELECT Sort_name, Subsort_ID, SubSort_name
FROM sort, subsort
WHERE sort.Sort_ID = subsort.Sort_ID AND sort_name LIKE '%文件%'
ORDER BY Subsort_ID DESC;
```

四、自然连接+多表连接

1. 自然连接:寻找两表中相同的字段进行等值连接，去除重复字段

基本语法:

```
SELECT 字段列表
FROM 表1 NATURAL JOIN 表2
WHERE 条件表达式
GROUP BY 分组字段 HAVING 二次过滤条件
ORDER BY 排序字段1 ASC | DESC
LIMIT [m,] n;
```

等价于

```
SELECT 字段列表
FROM 表1 JOIN 表2 USING(连接字段)
WHERE 条件表达式
GROUP BY 分组字段 HAVING 二次过滤条件
ORDER BY 排序字段1 ASC | DESC
LIMIT [m,] n;
```

A :

a	b	c
1	2	3
3	2	1
3	4	5

B :

c	d	f
5	6	7
7	6	5

`A NATURAL JOIN B`

```
SELECT *
FROM A NATURAL JOIN B;

-- 结果等价于
SELECT a, b, a.c, d, f
FROM A JOIN B ON A.c = A.c

-- 结果等价于（指定作自然连接的字段）
SELECT *
FROM A JOIN B USING(c);
```

a	b	c	d	f
3	4	5	6	7

示例8: 对表 `sort` 和 `subsort` 进行自然连接

```
SELECT *
FROM sort NATURAL JOIN subsort LIMIT 5;

-- 等价于
SELECT sort.sort_id, sort_name, subsort_id, subsort_name
FROM sort CROSS JOIN subsort
WHERE sort.sort_id = subsort.sort_id LIMIT 5;

-- 等价于
SELECT *
FROM sort JOIN subsort USING(sort_id) LIMIT 5;

--
SELECT *
FROM sort NATURAL LEFT JOIN subsort LIMIT 5;
```

2. 多表间的连接

多表间的连接按从左往右的次序进行连接，例如，`A JOIN B JOIN C`，先运算 `A JOIN B`，然后将其结果与 `C` 进行连接。

示例9: 多表交叉连接

```
SELECT * FROM product JOIN sort JOIN subsort;

SELECT * FROM product, sort JOIN subsort;
```

示例10: 多表自然连接

```
SELECT count(*)
FROM product NATURAL JOIN sort NATURAL JOIN subsort;
```

示例11: 多表内连接

```
SELECT count(*)
FROM product JOIN sort JOIN subsort ON product.sort_id=sort.sort_id AND
sort.sort_id=subsort.sort_id;

SELECT count(*)
FROM product JOIN sort JOIN subsort
WHERE product.sort_id=sort.sort_id AND sort.sort_id=subsort.sort_id;
```

五、子查询

`SELECT` 语句的结果也为一张表，因此可以将 `SELECT` 的查询结果作为条件表达式中 `IN` 操作符的集合，或者直接作为下一次查询的起点。

示例12：使用 `IN` 关键字, 查询子类别名 `“闹钟”` 对应的根类别信息。

```
SELECT *
FROM sort
WHERE Sort_ID IN ( SELECT Sort_ID
                   FROM subsort
                   WHERE SubSort_name='闹钟');
-- 等价于
SELECT sort.Sort_ID, sort.Sort_name
FROM sort JOIN subsort ON sort.Sort_ID = subsort.Sort_ID
WHERE subsort.SubSort_name = '闹钟';
```

`EXISTS` 操作符用于判断集合是否为空，如果为空，则返回0；如果不为空，则返回1.

示例13：使用 `EXISTS` 关键字，如果存在子类别编号为 `3101` ，则查询类别表中所有的记录。

```
SELECT *
FROM sort
WHERE EXISTS (SELECT Sort_ID
              FROM subsort
              WHERE SubSort_ID=3101);
```

`ANY` 操作符用于判断标量a与集合A之间的比较，例如 `a > ANY(A)` 如果标量a大于A中的某一个元素，则返回1；否则返回0。

> 等价于 `a > MIN(A)`

示例14：使用带 `ANY` 关键字的子查询，查询满足以下条件的产地名称：对应单价大于产地为 `'大连'` 的任一产品价格。

```
SELECT distinct Product_Place
FROM product
WHERE price > ANY (SELECT price
                   FROM product
                   WHERE Product_Place = '大连');
-- 等价于
SELECT distinct Product_Place
FROM product
WHERE price > (SELECT MIN(price)
                FROM product
                WHERE Product_Place = '大连');
```

`ALL` 操作符用于判断标量a与集合A之间的比较，例如 `a > ALL(A)` 如果标量a大于A中的所有元素，则返回1；否则返回0。

> 等价于 `a > MAX(A)`

示例15：使用带 `ALL` 关键字的子查询，查询满足以下条件的产地名称：对应单价大于产地为 `'大连'` 的所有产品的产地。

```
SELECT distinct Product_Place
FROM product
WHERE price > ALL (SELECT price
                   FROM product
                   WHERE Product_Place = '大连');
-- 等价于
SELECT distinct Product_Place
FROM product
WHERE price > (SELECT MAX(price)
                FROM product
                WHERE Product_Place = '大连');
```

练习

- (1) 使用内连接，查询 `product` 表和 `orders` 表中的订单号、订单时间、商品名称和商品数量。
- (2) 在 `product` 表和 `orders` 表之间使用左连接查询商品 `id` 、商品名称、订单号、订单时间和订单数量。
- (3) 在 `product` 表和 `orders` 表之间使用右连接查询商品 `id` 、商品名称、订单号、订单时间和订单数量。

- (4) 在 `member` 表和 `orders` 表之间使用内连接查询订单号、订单时间、商品名id、商品数量和客户真实姓名，并按订单时间降序排列。
- (5) 使用 `IN` 查询商品产地为 `“广东”` 的商品订单信息。
- (6) 使用 `EXISTS` 查询是否存在产地为 `“上海”` 的商品订单信息，如果存在，查询所有订单信息。
- (7) 使用 `ANY` 查询商品价格大于任一 `sort_id` 为11的商品价格的类别编号和类别名称（使用 `product` 和 `sort` ，要求去重）。
- (8) 使用 `ALL` 查询订单信息，其中商品价格要大于所有产地为 `“珠海”` 的商品商品价格，并按 `product_id` 升序排列。（使用 `orders` 表和 `product` 表）