JOIN

A joined table is a table derived from two other (real or derived) tables according to the rules of the particular join type. `Inner`, `outer`, and `cross-joins` are available. The general syntax of a joined table is

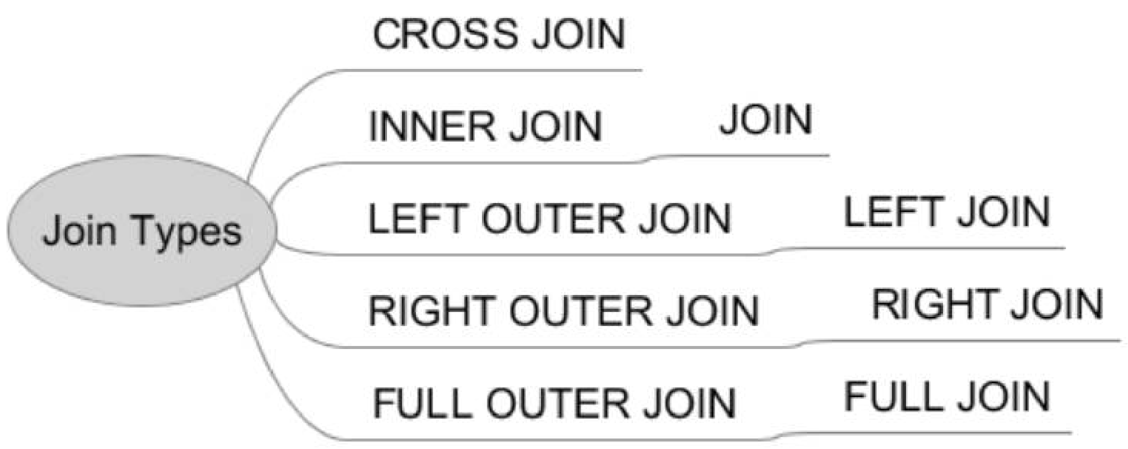
> *T1 join_type T2* [*join_condition*]

Here we use the following tables as example

```
--table T1
create table T1(
 num int primary key ,
 name varchar(10)
);
insert into T1(num, name) values
(1, 'a'),(2, 'b'),(6,'c'),(7,'d');

--table T2
create table T2(
 num int primary key ,
 value varchar(10)
);
insert into T2(num,value) values
(2, 'xxx'),(4,'yyy'),(6,'zzz');

--table T3
create table T3(
 num int primary key ,
 name varchar(10)
);
insert into T3(num,name) values
(1, 'pp'),(2,'qq'),(7,'kk');
```



CROSS JOIN

> *T1 CROSS JOIN T2*

For every possible combination of rows from `T1` and `T2` (i.e., a Cartesian product), the joined table will contain a row consisting of all columns in `T1` followed by all columns in `T2`. If the tables have `N` and `M` rows respectively, the joined table will have **N * M** rows.

包含由T1中的所有列和T2中的所有列组成的行

Experiment 1:

```
SELECT * FROM t1 CROSS JOIN t2;
```

INNER JOIN-->JOIN

For each row R1 of T1, the joined table has a row for each row in T2 that satisfies the join condition with R1. 对于T1中的每一行R1，连接表对T2中的每一行具有满足R1的连接条件的行。

Experiment 2: two join conditions

```
SELECT * FROM t1 INNER JOIN t2 on t1.num=t2.num;   t1.num t2.num（共4列）

SELECT * FROM t1 INNER JOIN t2 USING (num);
                                            num（共3列）
SELECT * FROM t1 NATURAL JOIN t2;
SELECT * FROM t1 NATURAL JOIN t3;
```

Tips: Notice the number of columns in out -tables. `NATURAL` is a shorthand form of `USING`: it forms a `USING` list consisting of all column names that appear in both input tables.

Experiment 3:

```
-- versus group 1
select *
from T1
cross join T2;

select *
from T1 inner join T2 on TRUE; 无条件连接--即形成一个cross join

select *
from T1, T2;

-- versus group 2
SELECT * FROM T1, T2 WHERE T1.num = T2.num AND T2.value='zzz';

SELECT * FROM T1 INNER JOIN T2 on T1.num = T2.num where T2.value='zzz';

SELECT * FROM T1 NATURAL JOIN T2 where T2.value='zzz';
```

Tips: These above table expressions of each group are equivalent

Experiment 4:

```
-- versus group1
select * FROM T1
CROSS JOIN T2
INNER JOIN T3 ON T2.num=T3.num;

select * FROM T1, T2
INNER JOIN T3 ON T2.num=T3.num;

--versus group2
select * FROM T1
CROSS JOIN T2
INNER JOIN T3 ON T1.num=T3.num;      cross join和,在一定情况下可以互相替换
                                     但是优先级有前后之分

select * FROM T1, T2
INNER JOIN T3 ON T1.num=T3.num;--wrong syntax
          此处T1的操作不能识别
```

Tips: In versus group 2, the `condition` "T1.num=T3.num" can reference `T1` in the first case but not the second.

OUTER JOIN

LEFT OUTER JOIN-->LEFT JOIN

T1：1 2 6 7
T2：2 4 6

First, an inner join is performed. Then, for each row in T1 that does not satisfy the join condition with any row in T2, a joined row is added with null values in columns of T2. Thus, the joined table always has at least one row for each row in T1.

首先，执行一个内部连接。然后，对于T1中的任何一行不满足连接条件的每一行，在T2的列中添加一个带有空值的连接行。因此，所连接的表对于T1中的每一行总是至少有一行。

Experiment 5:

```
SELECT * FROM t1 LEFT JOIN t2 ON t1.num = t2.num;

SELECT * FROM t1 LEFT JOIN t2 USING (num);        1 2 6 7
```

RIGHT OUTER JOIN-->RIGHT JOIN

First, an inner join is performed. Then, for each row in T2 that does not satisfy the join condition with any row in T1, a joined row is added with null values in columns of T1. This is the converse of a left join: the result table will always have a row for each row in T2.

Experiment 6:

```
SELECT * FROM t1 RIGHT JOIN t2 ON t1.num = t2.num;
                                                 2 4 6
SELECT * FROM t1 RIGHT JOIN t2 USING (num);
```

FULL OUTER JOIN-->FULL JOIN

First, an inner join is performed. Then, for each row in T1 that does not satisfy the join condition with any row in T2, a joined row is added with null values in columns of T2. Also, for each row of T2 that does not satisfy the join condition with any row in T1, a joined row with null values in the columns of T1 is added.

Experiment 7:

FULL JOIN vs LEFT JOIN vs RIGHT JOIN

```
SELECT * FROM t1 FULL JOIN t2 ON t1.num = t2.num;          1 2 6 7 4
```

Tips: The result of Experiment 7 could compare to that of Experiment 5 and 6.

Experiment 8:

```
SELECT * FROM t1 LEFT JOIN t2 ON t1.num = t2.num AND t2.value = 'xxx';

SELECT * FROM t1 LEFT JOIN t2 ON t1.num = t2.num WHERE t2.value = 'xxx';
```

Tips: The condition of left join in first case is "(t1.num = t2.num AND t2.value = 'xxx')".

With

`WITH` provides a way to write auxiliary statements for use in a larger query. These statements, which are often referred to as Common Table Expressions or CTEs, can be thought of as defining temporary tables that exist just for one query. Each auxiliary statement in a `WITH` clause can be a `SELECT`, `INSERT`, `UPDATE`, or `DELETE`; and the `WITH` clause itself is attached to a primary statement that can also be a `SELECT`, `INSERT`, `UPDATE`, or `DELETE`.

Experiment 9: Iteration by 'with'

可以被认为是定义了仅为一个查询而存在的临时表。

WITH子句中的每个辅助语句都可以是选择、插入、更新或删除；并且WITH子句本身被附加到主语句上，主语句也可以是选择、插入、更新或删除。

```
WITH RECURSIVE t(n) AS (
VALUES (1)
UNION ALL
SELECT n+1 FROM t WHERE n < 100
)
SELECT sum(n) FROM t LIMIT 100;
```

先执行select * from 把结果放到一个临时表中，作为全局使用
一次分析，多次使用：便于构造多个子查询

5050

```
WITH RECURSIVE t(n) AS (
SELECT 1
UNION ALL
SELECT n+1 FROM t
)
SELECT n FROM t LIMIT 100;
```

Experiment 10:

film(film(Jackie Chan act).runtime>avg(film(Jackie Chan act).runtime)).detail

```
WITH jc_film as (
    select m.movieid id, m.title tl, m.runtime rt, m.year_released yr
    from movies m
             join credits c on m.movieid = c.movieid
             join people p on c.peopleid = p.peopleid
    where p.first_name = 'Jackie'
      and p.surname = 'Chan'
      and c.credited_as = 'A'
)
select *
from jc_film
where jc_film.rt >
      (select avg(t1.rt) from jc_film t1);
```