

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

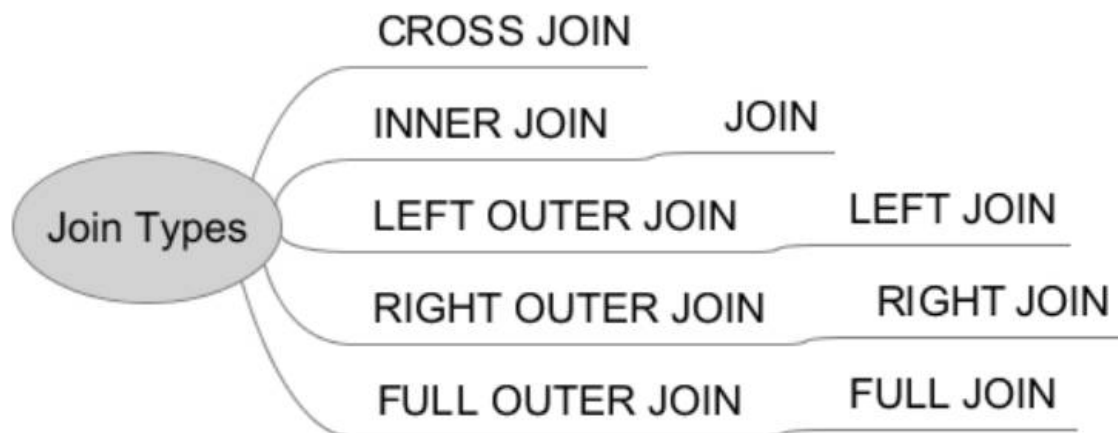
```
T1 join_type T2 [join_condition]
```

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 T_1 and T_2 (i.e., a Cartesian product), the joined table will contain a row consisting of all columns in T_1 followed by all columns in T_2 . If the tables have N and M rows respectively, the joined table will have $N * M$ rows.

Experiment 1:

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

num	name
1	a
2	b
6	c
7	d

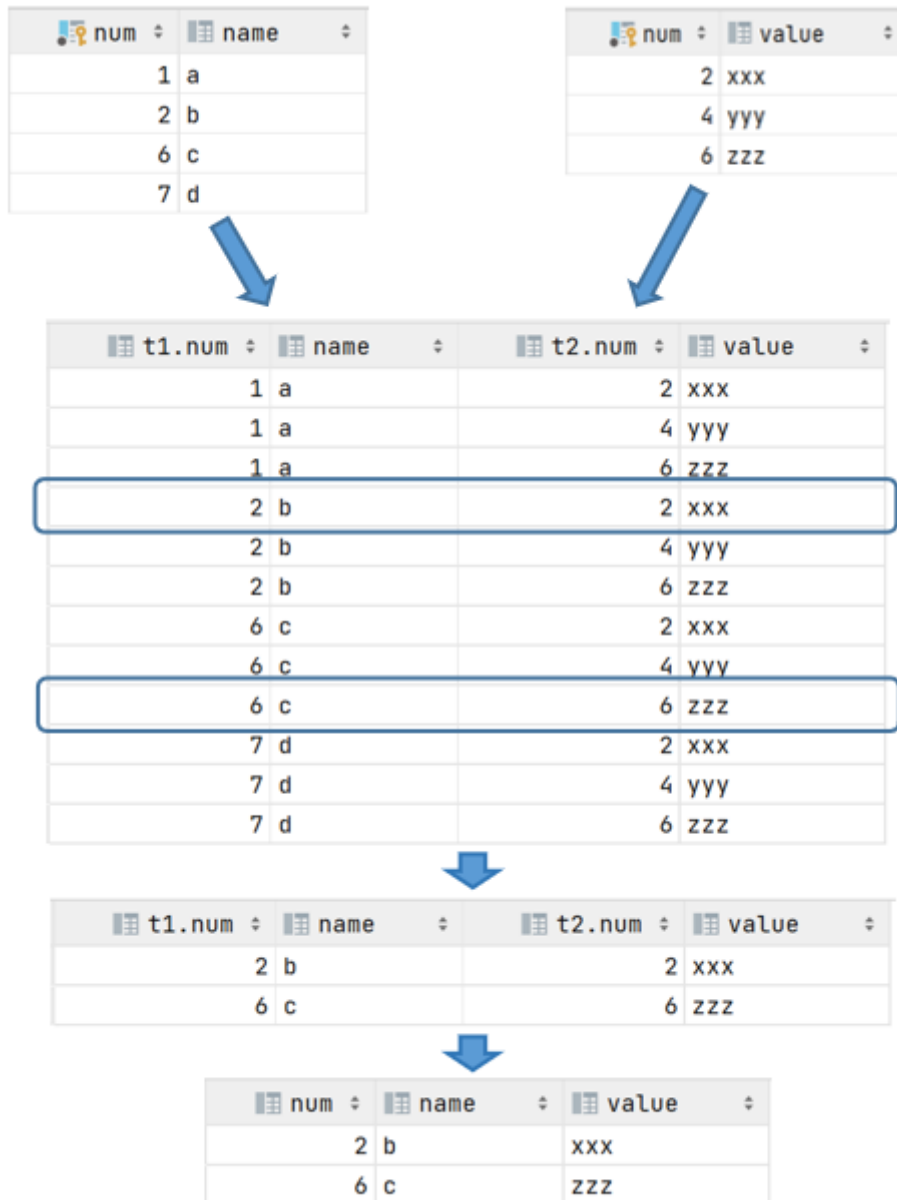
num	value
2	xxx
4	yyy
6	zzz



t1.num	name	t2.num	value
1	a	2	xxx
1	a	4	yyy
1	a	6	zzz
2	b	2	xxx
2	b	4	yyy
2	b	6	zzz
6	c	2	xxx
6	c	4	yyy
6	c	6	zzz
7	d	2	xxx
7	d	4	yyy
7	d	6	zzz

INNER JOIN-->JOIN

For each row R_1 of T_1 , the joined table has a row for each row in T_2 that satisfies the join condition with R_1 .



Experiment 2: two join conditions

```
SELECT * FROM t1 INNER JOIN t2 on t1.num=t2.num;
```

```
SELECT * FROM t1 INNER JOIN t2 USING (num);
```

```
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;

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;

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

```

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

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.

*Left
join*

num	name
1	a
2	b
6	c
7	d

num	value
2	xxx
4	yyy
6	zzz

t1.num	name	t2.num	value
1	a	2	xxx
1	a	4	yyy
1	a	6	zzz
2	b	2	xxx
2	b	4	yyy
2	b	6	zzz
6	c	2	xxx
6	c	4	yyy
6	c	6	zzz
7	d	2	xxx
7	d	4	yyy
7	d	6	zzz

t1.num	name	t2.num	value
1	a	<null>	<null>
2	b	2	xxx
6	c	6	zzz
7	d	<null>	<null>

Experiment 5:

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

```
SELECT * FROM t1 LEFT JOIN t2 USING (num);
```

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.

num	name
1	a
2	b
6	c
7	d

num	value
2	xxx
4	yyy
6	zzz

*Right
join*

t1.num	name	t2.num	value
1	a	2	xxx
1	a	4	yyy
1	a	6	zzz
2	b	2	xxx
2	b	4	yyy
2	b	6	zzz
6	c	2	xxx
6	c	4	yyy
6	c	6	zzz
7	d	2	xxx
7	d	4	yyy
7	d	6	zzz

t1.num	name	t2.num	value
2	b	2	xxx
<null>	<null>	4	yyy
6	c	6	zzz

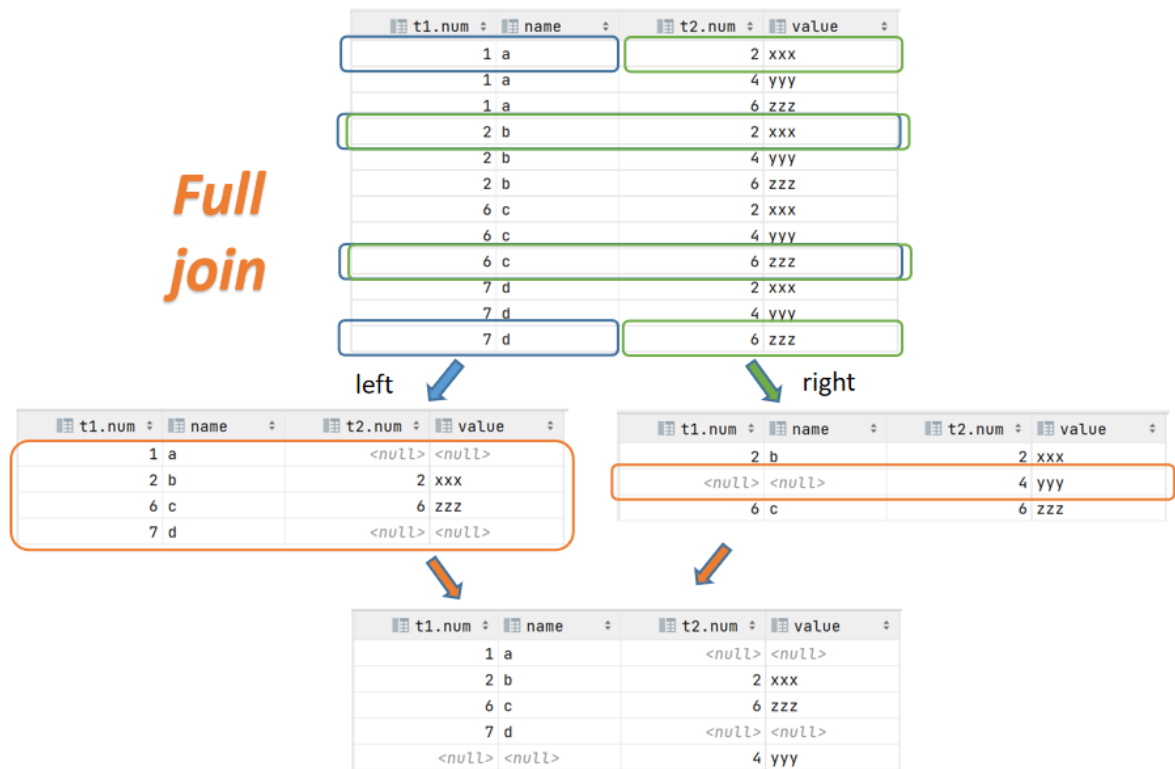
Experiment 6:

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

```
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;
```

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'

```
WITH RECURSIVE t(n) AS (  
VALUES (1)  
UNION ALL  
SELECT n+1 FROM t WHERE n < 100  
)  
SELECT sum(n) FROM t LIMIT 100;  
  
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 t1, 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);
```

Combining Queries

```
query1 UNION [ALL] query2  
query1 INTERSECT [ALL] query2  
query1 EXCEPT [ALL] query2
```

Here we use the following tables as example

```
create table T1  
(  
num int primary key,  
name varchar(10)  
);  
insert into T1(num, name)  
values (1, 'qqq');
```



```

(2, 'yyy'),
(4, 'zzz'),
(7, 'qqq'),
(8, 'zzz');

create table T2
(
    num int primary key,
    name varchar(10)
);
insert into T2(num, name)
values (2, 'yyy'),
       (4, 'qqq'),
       (6, 'qqq');
```

UNION and UNION ALL

`UNION` effectively appends the result of *query2* to the result of *query1* (although there is no guarantee that this is the order in which the rows are actually returned). Furthermore, it eliminates duplicate rows from its result, in the same way as `DISTINCT`, unless `UNION ALL` is used.

Experiment 11: Union

```

-- same attribution name
select num from T1
UNION
select num from T2;

-- attribution names are same with T1
select * from T1
UNION
select * from T2;
```

Tips: First combine all, then do the distinct.

Experiment 12: Union ALL

```

-- Union all
select num from T1
UNION all
select num from T2;

-- union all + order by
select num from T1
UNION all
select num from T2
order by num;

-- union all + order by II
```

```

select * from T1
UNION all
select * from T2
order by num;

-- union all + distinct
select distinct(U.*) from(
select * from T1
UNION all
select * from T2
) U;

```

INTERSECT and INTERSECT ALL

`INTERSECT` returns all rows that are both in the result of *query1* and in the result of *query2*. Duplicate rows are eliminated unless `INTERSECT ALL` is used.

Experiment 13: INTERSECT

```

-- intersect
select name from T1
intersect
select name from T2;

```

Tips: First intersect all, then do the distinct.

Experiment 14: INTERSECT ALL

```

-- intersect all
select name from T1
intersect all
select name from T2;

-- intersect all + distinct
select distinct (I.name)
from (
    select name
    from T1
    intersect all
    select name
    from T2
) I;

```

EXCEPT

`EXCEPT` returns all rows that are in the result of `query1` but not in the result of `query2`. (This is sometimes called the *difference* between two queries.) Again, duplicates are eliminated unless `EXCEPT ALL` is used.

Experiment 15: EXCEPT

```
-- except I
select name
from T1
except
select name
from T2;

-- except II
select name
from T2
except
select name
from T1;
```

Tips: Notice the order of tables.

Experiment 16: EXCEPT ALL

```
-- EXCEPT ALL
select name
from T1
except all
select name
from T2;

-- EXCEPT ALL + distinct
select distinct(E.name)
from (
    select name
    from T1
    except all
    select name
    from T2
) E;
```