

# SQLITE - JOINS

[http://www.tutorialspoint.com/sqlite/sqlite\\_using\\_joins.htm](http://www.tutorialspoint.com/sqlite/sqlite_using_joins.htm)

Copyright © tutorialspoint.com

The SQLite **Joins** clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each.

SQL defines three major types of joins:

- The CROSS JOIN
- The INNER JOIN
- The OUTER JOIN

Before we proceed, let's consider two tables COMPANY and DEPARTMENT. We already have seen INSERT statements to populate COMPANY table. So just let's assume the list of records available in COMPANY table:

ID	NAME	AGE	ADDRESS	SALARY
1	Paul	32	California	20000.0
2	Allen	25	Texas	15000.0
3	Teddy	23	Norway	20000.0
4	Mark	25	Rich-Mond	65000.0
5	David	27	Texas	85000.0
6	Kim	22	South-Hall	45000.0
7	James	24	Houston	10000.0

Another table is DEPARTMENT has the following definition:

```
CREATE TABLE DEPARTMENT(  
  ID INT PRIMARY KEY      NOT NULL,  
  DEPT CHAR(50) NOT NULL,  
  EMP_ID INT NOT NULL  
);
```

Here is the list of INSERT statements to populate DEPARTMENT table:

```
INSERT INTO DEPARTMENT (ID, DEPT, EMP_ID)  
VALUES (1, 'IT Billing', 1 );  
  
INSERT INTO DEPARTMENT (ID, DEPT, EMP_ID)  
VALUES (2, 'Engineering', 2 );  
  
INSERT INTO DEPARTMENT (ID, DEPT, EMP_ID)  
VALUES (3, 'Finance', 7 );
```

Finally, we have the following list of records available in DEPARTMENT table:

ID	DEPT	EMP_ID
1	IT Billing	1
2	Engineerin	2
3	Finance	7

## The CROSS JOIN

A CROSS JOIN matches every row of the first table with every row of the second table. If the input tables have x and y columns, respectively, the resulting table will have x+y columns. Because CROSS JOINS have the potential to generate extremely large tables, care must be taken to only use them when appropriate.

Following is the syntax of CROSS JOIN:

```
SELECT ... FROM table1 CROSS JOIN table2 ...
```

Based on the above tables, we can write a cross join as follows:

```
sqlite> SELECT EMP_ID, NAME, DEPT FROM COMPANY CROSS JOIN DEPARTMENT;
```

Above query will produce the following result:

EMP_ID	NAME	DEPT
1	Paul	IT Billing
2	Paul	Engineerin
7	Paul	Finance
1	Allen	IT Billing
2	Allen	Engineerin
7	Allen	Finance
1	Teddy	IT Billing
2	Teddy	Engineerin
7	Teddy	Finance
1	Mark	IT Billing
2	Mark	Engineerin
7	Mark	Finance
1	David	IT Billing
2	David	Engineerin
7	David	Finance
1	Kim	IT Billing
2	Kim	Engineerin
7	Kim	Finance
1	James	IT Billing
2	James	Engineerin
7	James	Finance

## The INNER JOIN

A INNER JOIN creates a new result table by combining column values of two tables *table1* and *table2* based upon the join-predicate. The query compares each row of table1 with each row of table2 to find all pairs of rows which satisfy the join-predicate. When the join-predicate is satisfied, column values for each matched pair of rows of A and B are combined into a result row.

An INNER JOIN is the most common type of join and is the default type of join. You can use INNER keyword optionally.

Following is the syntax of INNER JOIN:

```
SELECT ... FROM table1 [INNER] JOIN table2 ON conditional_expression ...
```

To avoid redundancy and keep the phrasing shorter, INNER JOIN conditions can be declared with a **USING** expression. This expression specifies a list of one or more columns:

```
SELECT ... FROM table1 JOIN table2 USING ( column1 ,... ) ...
```

A NATURAL JOIN is similar to a **JOIN...USING**, only it automatically tests for equality between the values of every column that exists in both tables:

```
SELECT ... FROM table1 NATURAL JOIN table2...
```

Based on the above tables, we can write a INNER JOIN as follows:

```
sqlite> SELECT EMP_ID, NAME, DEPT FROM COMPANY INNER JOIN DEPARTMENT
        ON COMPANY.ID = DEPARTMENT.EMP_ID;
```

Above query will produce the following result:

EMP_ID	NAME	DEPT
1	Paul	IT Billing
2	Allen	Engineerin
7	James	Finance

## The OUTER JOIN

The OUTER JOIN is an extension of the INNER JOIN. Though SQL standard defines three types of OUTER JOINS: LEFT, RIGHT, and FULL but SQLite only supports the **LEFT OUTER JOIN**.

The OUTER JOINS have a condition that is identical to INNER JOINS, expressed using an ON, USING, or NATURAL keyword. The initial results table is calculated the same way. Once the primary JOIN is calculated, an OUTER join will take any unjoined rows from one or both tables, pad them out with NULLs, and append them to the resulting table.

Following is the syntax of LEFT OUTER JOIN:

```
SELECT ... FROM table1 LEFT OUTER JOIN table2 ON conditional_expression ...
```

To avoid redundancy and keep the phrasing shorter, OUTER JOIN conditions can be declared with a USING expression. This expression specifies a list of one or more columns:

```
SELECT ... FROM table1 LEFT OUTER JOIN table2 USING ( column1 ,... ) ...
```

Based on the above tables, we can write a inner join as follows:

```
sqlite> SELECT EMP_ID, NAME, DEPT FROM COMPANY LEFT OUTER JOIN DEPARTMENT
        ON COMPANY.ID = DEPARTMENT.EMP_ID;
```

Above query will produce the following result:

EMP_ID	NAME	DEPT
1	Paul	IT Billing
2	Allen	Engineerin
	Teddy	
	Mark	
	David	
	Kim	
7	James	Finance

Loading [MathJax]/jax/output/HTML-CSS/fonts/TeX/fontdata.js