MySQLTUTORIAL

MySQL CROSS JOIN

Summary: in this tutorial, you will learn about the MySQL `CROSS JOIN` clause and how to use it more effectively.

Introduction to MySQL CROSS JOIN clause

Suppose you join two tables using the `CROSS JOIN` clause. The result set will include all rows from both tables, where each row is the combination of the row in the first table with the row in the second table. In general, if each table has `n` and `m` rows respectively, the result set will have `nxm` rows.

In other words, the `CROSS JOIN` clause returns a **Cartesian product** of rows from the joined tables.

The following illustrates the syntax of the `CROSS JOIN` clause that joins two tables `t1` and `t2` :

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

Note that different from the `INNER JOIN` , `LEFT JOIN` , and `RIGHT JOIN` clauses, the `CROSS JOIN` clause does not have a join predicate. In other words, it does not have the `ON` or `USING` clause.

If you add a `WHERE` clause, in case table `t1` and `t2` has a relationship, the `CROSS JOIN` works like the `INNER JOIN` clause as shown in the following query:

```
SELECT select_list
FROM t1
CROSS JOIN t2
WHERE t1.id = t2.id;
```

MySQL CROSS JOIN clause examples

Let's take some examples to understand how the cross join works.

1) Simple cross join example

We'll use a cross join to create a deck of 52 playing cards.

First, create a table that stores suits:

```
CREATE TABLE suits (
    suit_id INT,
    suit_name VARCHAR(10)
);
```

Second, create a table to store ranks:

```
CREATE TABLE ranks (
    rank_id INT,
    rank_name VARCHAR(5)
);
```

Third, insert data into the suits and ranks table:

```
INSERT INTO suits (suit_id, suit_name) VALUES
    (1, 'Hearts'),
    (2, 'Diamonds'),
    (3, 'Clubs'),
    (4, 'Spades');

INSERT INTO ranks (rank_id, rank_name) VALUES
    (1, 'Ace'),
    (2, '2'),
    (3, '3'),
    (4, '4'),
    (5, '5'),
    (6, '6'),
    (7, '7'),
    (8, '8'),
    (9, '9'),
    (10, '10'),
    (11, 'Jack'),
```

```
    (12, 'Queen'),
    (13, 'King');
```

Finally, use a cross to combine the suits and ranks to create a deck of 52 playing cards:

```
SELECT
  suit_name,
  rank_name
FROM
  suits CROSS
  JOIN ranks
ORDER BY
  suit_name,
  rank_name;
```

Output:

```
+-----------+-----------+
| suit_name | rank_name |
+-----------+-----------+
| Clubs     | 10        |
| Clubs     | 2         |
| Clubs     | 3         |
| Clubs     | 4         |
| Clubs     | 5         |
| Clubs     | 6         |
| Clubs     | 7         |
| Clubs     | 8         |
| Clubs     | 9         |
| Clubs     | Ace       |
| Clubs     | Jack      |
| Clubs     | King      |
| Clubs     | Queen     |
| Diamonds  | 10        |
| Diamonds  | 2         |
| Diamonds  | 3         |
| Diamonds  | 4         |
| Diamonds  | 5         |
| Diamonds  | 6         |
| Diamonds  | 7         |
| Diamonds  | 8         |
```

```
| Diamonds  | 9         |
| Diamonds  | Ace       |
| Diamonds  | Jack      |
| Diamonds  | King      |
| Diamonds  | Queen     |
| Hearts    | 10        |
| Hearts    | 2         |
| Hearts    | 3         |
| Hearts    | 4         |
| Hearts    | 5         |
| Hearts    | 6         |
| Hearts    | 7         |
| Hearts    | 8         |
| Hearts    | 9         |
| Hearts    | Ace       |
| Hearts    | Jack      |
| Hearts    | King      |
| Hearts    | Queen     |
| Spades    | 10        |
| Spades    | 2         |
| Spades    | 3         |
| Spades    | 4         |
| Spades    | 5         |
| Spades    | 6         |
| Spades    | 7         |
| Spades    | 8         |
| Spades    | 9         |
| Spades    | Ace       |
| Spades    | Jack      |
| Spades    | King      |
| Spades    | Queen     |
+-----------+-----------+
52 rows in set (0.00 sec)
```

In this query, we use a cross join to combine each suit from the suits table with each rank from the ranks table, resulting in a Cartesian product that pairs every suit with every rank.

2) A complex cross join example

First, create a new database `salesdb` :

```
CREATE DATABASE IF NOT EXISTS salesdb;
```

Second, switch the current data to the new database `salesdb` :

```
USE salesdb;
```

Third, create new tables in the `salesdb` database:

```
CREATE TABLE products (
    id INT PRIMARY KEY AUTO_INCREMENT,
    product_name VARCHAR(100),
    price DECIMAL(13,2 )
);

CREATE TABLE stores (
    id INT PRIMARY KEY AUTO_INCREMENT,
    store_name VARCHAR(100)
);

CREATE TABLE sales (
    product_id INT,
    store_id INT,
    quantity DECIMAL(13 , 2 ) NOT NULL,
    sales_date DATE NOT NULL,
    PRIMARY KEY (product_id , store_id),
    FOREIGN KEY (product_id)
        REFERENCES products (id)
        ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY (store_id)
        REFERENCES stores (id)
        ON DELETE CASCADE ON UPDATE CASCADE
);
```

Here are the descriptions of the tables:

- The table `products` contains the product master data that includes product id, product name, and sales price.
- The table `stores` contains the stores where the products are sold.

- The table `sales` contains the products that are sold in a particular store by quantity and date.

Fourth, insert data into the three tables.

Suppose that we have three products `iPhone` , `iPad` and `Macbook Pro` which are sold in two stores `North` and `South` .

```
INSERT INTO products(product_name, price)
VALUES('iPhone', 699),
      ('iPad',599),
      ('Macbook Pro',1299);

INSERT INTO stores(store_name)
VALUES('North'),
      ('South');

INSERT INTO sales(store_id,product_id,quantity,sales_date)
VALUES(1,1,20,'2017-01-02'),
      (1,2,15,'2017-01-05'),
      (1,3,25,'2017-01-05'),
      (2,1,30,'2017-01-02'),
      (2,2,35,'2017-01-05');
```

Fifth, the following statement returns the total sales for each product in each store:

```
SELECT
    store_name,
    product_name,
    SUM(quantity * price) AS revenue
FROM
    sales
        INNER JOIN
    products ON products.id = sales.product_id
        INNER JOIN
    stores ON stores.id = sales.store_id
GROUP BY store_name , product_name;
```

store_name	product_name	revenue
North	iPad	8985.0000
North	iPhone	13980.0000
North	Macbook Pro	32475.0000
South	iPad	20965.0000
South	iPhone	20970.0000

Now, what if you wish to determine which store had no sales of a particular product? The previously mentioned statement is unable to address this query.

To solve the problem, you can use the `CROSS JOIN` clause.

Sixth, use the `CROSS JOIN` clause to get the combination of all stores and products:

```
SELECT
    store_name, product_name
FROM
    stores AS a
        CROSS JOIN
    products AS b;
```

store_name	product_name
North	iPhone
South	iPhone
North	iPad
South	iPad
North	Macbook Pro
South	Macbook Pro

Next, join the result of the query above with a query that returns the total of sales by store and product:

```
SELECT
    b.store_name,
    a.product_name,
    IFNULL(c.revenue, 0) AS revenue
FROM
    products AS a
        CROSS JOIN
    stores AS b
        LEFT JOIN
    (SELECT
```

```
        stores.id AS store_id,
        products.id AS product_id,
        store_name,
            product_name,
            ROUND(SUM(quantity * price), 0) AS revenue
    FROM
        sales
    INNER JOIN products ON products.id = sales.product_id
    INNER JOIN stores ON stores.id = sales.store_id
    GROUP BY stores.id, products.id, store_name , product_name) AS c ON c.store_id
        AND c.product_id= a.id
ORDER BY b.store_name;
```

store_name	product_name	revenue
North	Macbook Pro	32475
North	iPad	8985
North	iPhone	13980
South	iPhone	20970
South	Macbook Pro	0
South	iPad	20965

Note that the query used the `IFNULL` function to return 0 if the revenue is `NULL` (in case the store had no sales).

By using the `CROSS JOIN` clause this way, you can answer a wide range of questions e.g., find the sales revenue by salesman, month even if the salesman has no sales in a particular month.

Summary

- A cross join combines each row from a table with each row from another table, resulting in a Cartesian product.
- Use the `CROSS JOIN` clause to perform a cross join.