MySQL

Tushar B. Kute,
http://tusharkute.com

Sorting

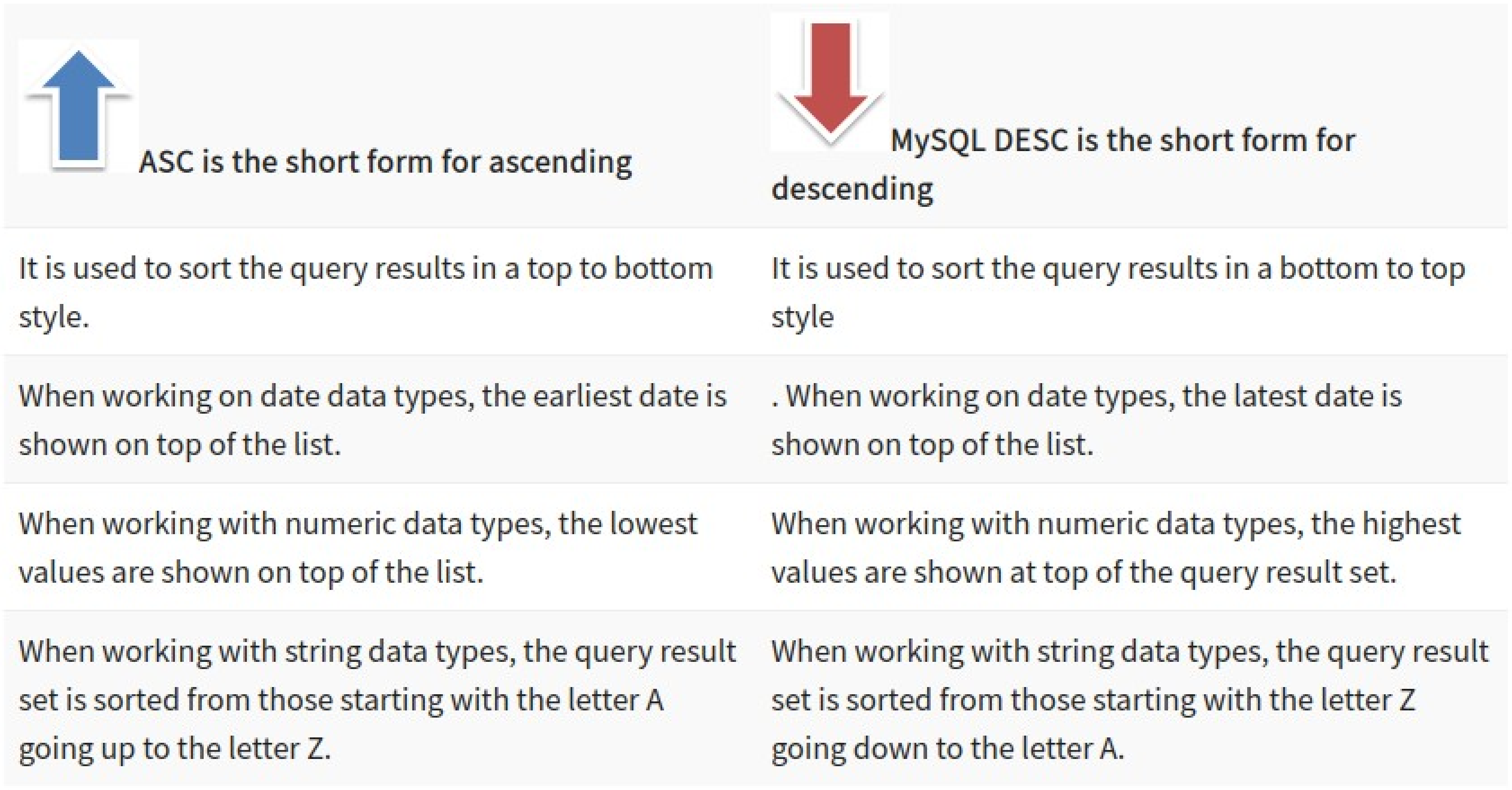
- Using the SELECT command, results were returned in the same order the records were added into the database.
- This is the default sort order. In this section, we will be looking at how we can sort our query results.
- Sorting is simply re-arranging our query results in a specified way. Sorting can be performed on a single column or on more than one column.
- It can be done on number, strings as well as date data types.

Order by

- MySQL ORDER BY is used in conjunction with the SELECT query to sort data in an orderly manner.
- The MySQL ORDER BY clause is used to sort the query result sets in either ascending or descending order.

SELECT statement... [WHERE condition | GROUP BY 'field_name(s)' HAVING condition] ORDER BY 'field_name(s)' [ASC | DESC];

Order by



ASC is the short form for ascending	MySQL DESC is the short form for descending
It is used to sort the query results in a top to bottom style.	It is used to sort the query results in a bottom to top style
When working on date data types, the earliest date is shown on top of the list.	. When working on date types, the latest date is shown on top of the list.
When working with numeric data types, the lowest values are shown on top of the list.	When working with numeric data types, the highest values are shown at top of the query result set.
When working with string data types, the query result set is sorted from those starting with the letter A going up to the letter Z.	When working with string data types, the query result set is sorted from those starting with the letter Z going down to the letter A.

Example:

- SELECT * FROM members ORDER BY date_of_birth DESC;

- SELECT * FROM 'members' ORDER BY 'gender';

- SELECT * FROM 'members' ORDER BY 'gender','date_of_birth' DESC;

The group by clause

- The GROUP BY clause is a SQL command that is used to group rows that have the same values.
- The GROUP BY clause is used in the SELECT statement. Optionally it is used in conjunction with aggregate functions to produce summary reports from the database.
- That's what it does, summarizing data from the database.
- The queries that contain the GROUP BY clause are called grouped queries and only return a single row for every grouped item.

The group by clause

- Now that we know what the SQL GROUP BY clause is, let's look at the syntax for a basic group by query.

  SELECT statements… GROUP BY column_name1[,column_name2,…] [HAVING condition];

The group by clause

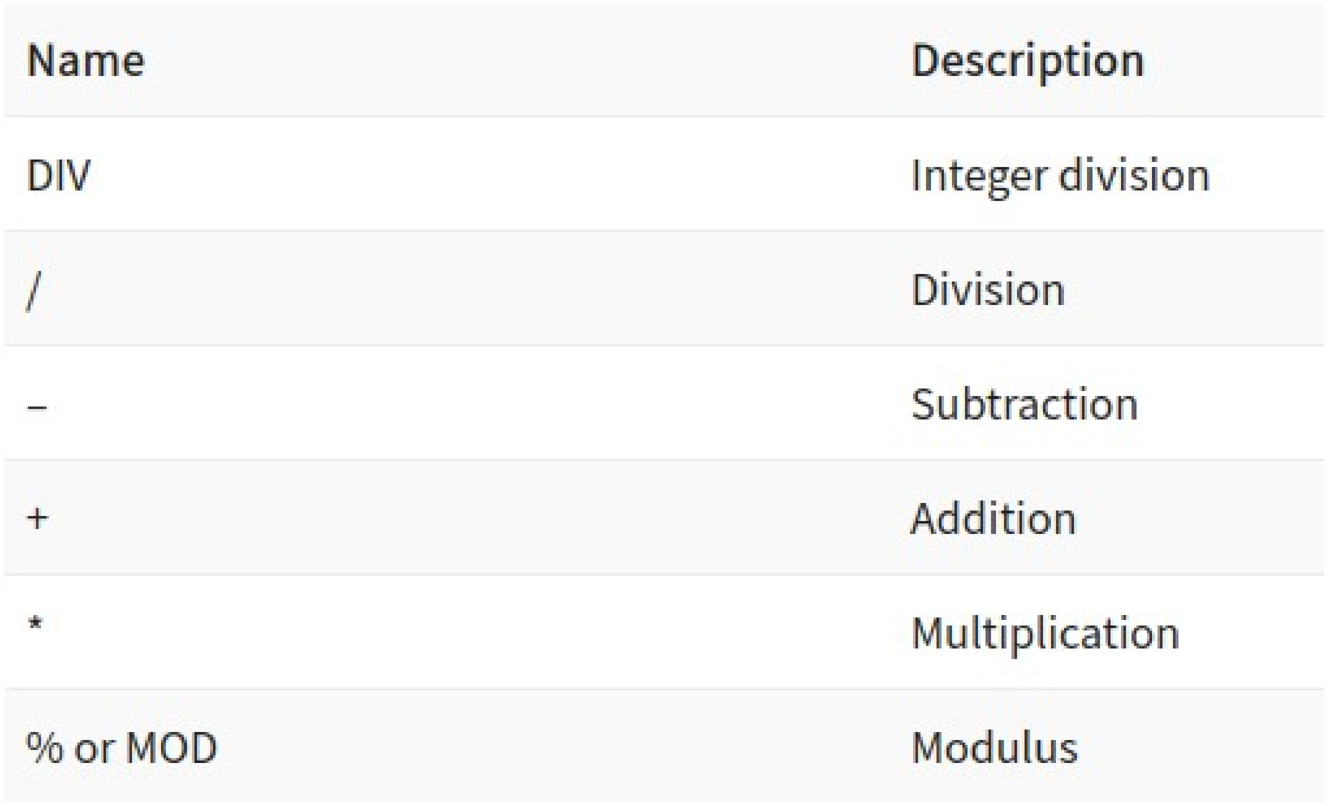
- Suppose we want to get the unique values for genders. We can use a following query –

  SELECT 'gender' FROM 'members' GROUP BY 'gender';

The group by clause

- SELECT 'category_id','year_released' FROM 'movies' GROUP BY 'category_id', 'year_released';

Numeric Functions



Name	Description
DIV	Integer division
/	Division
–	Subtraction
+	Addition
*	Multiplication
% or MOD	Modulus

Numeric Functions

- Integer Division (DIV)

  SELECT 23 DIV 6 ;

  Executing the above script gives us the following results.

  3
- Division operator (/)

  Let's now look at the division operator example. We will modify the DIV example.

  SELECT 23 / 6 ;
- Executing the above script gives us the following results.

  3.8333

Numeric Functions

- Subtraction operator (-)

  Let’s now look at the subtraction operator example. We will use the same values as in the previous two examples

  SELECT 23 - 6 ;

  Executing the above script gives us 17
- Addition operator (+)

  Let’s now look at the addition operator example. We will modify the previous example.

  SELECT 23 + 6 ;

  Executing the above script gives us 29

Numeric Functions

- Multiplication operator (*)

  Let’s now look at the multiplication operator example. We will use the same values as in the previous examples.

  SELECT 23 * 6 AS 'multiplication_result';

  Executing the above script gives us the following results.

  multiplication_result

  138

Numeric Functions

- Modulo operator (%)
  - The modulo operator divides N by M and gives us the remainder. Let’s now look at the modulo operator example. We will use the same values as in the previous examples.

SELECT 23 % 6 ;

OR

SELECT 23 MOD 6 ;

Having Clause

- MySQL HAVING Clause is used with GROUP BY clause. It always returns the rows where condition is TRUE.
- Syntax:

```
SELECT expression1, expression2, ... expression_n,
aggregate_function (expression)
FROM tables
[WHERE conditions]
GROUP BY expression1, expression2, ... expression_n
HAVING condition;
```

Having Clause

```
SELECT emp_name, SUM(working_hours) AS "Tot work hrs"

FROM employees
GROUP BY emp_name
HAVING SUM(working_hours) > 5;
```

The like clause

- The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.
- There are two wildcards often used in conjunction with the LIKE operator:
  - The percent sign (%) represents zero, one, or multiple characters
  - The underscore sign (_) represents one, single character
- The percent sign and the underscore can also be used in combinations!

Syntax:

- SELECT column1, column2, ...
  FROM table_name
  WHERE columnN LIKE pattern;
- Tip: You can also combine any number of conditions using AND or OR operators.

Examples:

- WHERE CustomerName LIKE 'a%' Finds any values that start with "a"
- WHERE CustomerName LIKE '%a' Finds any values that end with "a"
- WHERE CustomerName LIKE '%or%' Finds any values that have "or" in any position
- WHERE CustomerName LIKE '_r%' Finds any values that have "r" in the second position
- WHERE CustomerName LIKE 'a_%' Finds any values that start with "a" and are at least 2 characters in length
- WHERE CustomerName LIKE 'a__%' Finds any values that start with "a" and are at least 3 characters in length
- WHERE ContactName LIKE 'a%o' Finds any values that start with "a" and ends with "o"

Distinct

- The DISTINCT keyword that allows us to omit duplicates from our results.
- This is achieved by grouping similar values together .

```
SELECT DISTINCT 'movie_id' FROM
'movierentals';
```

The Between operator

- The BETWEEN operator selects values within a given range.
- The values can be numbers, text, or dates.
- The BETWEEN operator is inclusive: begin and end values are included.

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

The Between operator

- SELECT * FROM Products

  WHERE Price NOT BETWEEN 10 AND 20;

- SELECT * FROM student WHERE name BETWEEN 'Chintoo' AND 'Jenny'

Subquery

- A subquery in MySQL is a query, which is nested into another SQL query and embedded with SELECT, INSERT, UPDATE or DELETE statement along with the various operators.
- We can also nest the subquery with another subquery.
- A subquery is known as the inner query, and the query that contains subquery is known as the outer query.

Subquery

- The inner query executed first gives the result to the outer query, and then the main/outer query will be performed.
- MySQL allows us to use subquery anywhere, but it must be closed within parenthesis.
- All subquery forms and operations supported by the SQL standard will be supported in MySQL also.

Subquery: Rules

- Subqueries should always use in parentheses.
- If the main query does not have multiple columns for subquery, then a subquery can have only one column in the SELECT command.
- We can use various comparison operators with the subquery, such as >, <, =, IN, ANY, SOME, and ALL. A multiple-row operator is very useful when the subquery returns more than one row.
- We cannot use the ORDER BY clause in a subquery, although it can be used inside the main query.
- If we use a subquery in a set function, it cannot be immediately enclosed in a set function.

Subquery: Advantages

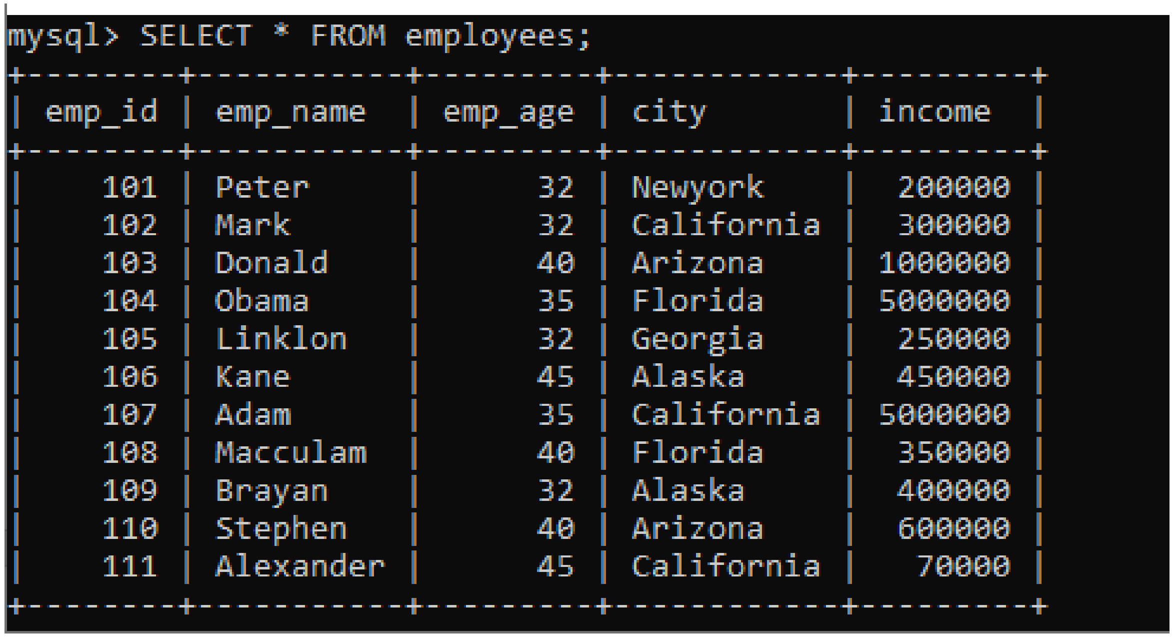
- The subqueries make the queries in a structured form that allows us to isolate each part of a statement.
- The subqueries provide alternative ways to query the data from the table; otherwise, we need to use complex joins and unions.
- The subqueries are more readable than complex join or union statements.

Subquery: Syntax

```
SELECT column_list (s) FROM  table_name
WHERE  column_name OPERATOR (SELECT
column_list (s)   FROM table_name
[WHERE])
```

Subquery: Syntax

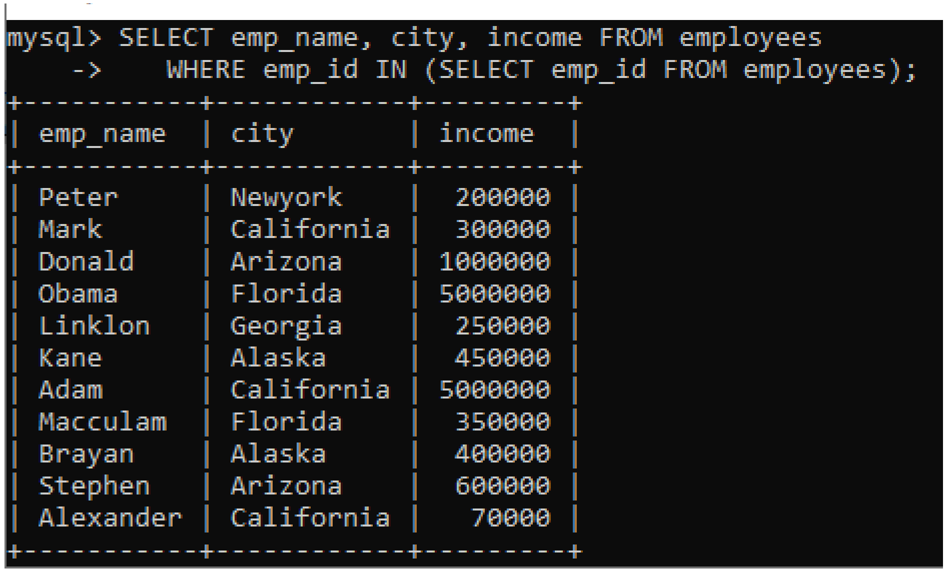
- Let us understand it with the help of an example.
- Suppose we have a table named "employees" that contains the following data:



```
mysql> SELECT * FROM employees;
+--------+-----------+---------+------------+---------+
| emp_id | emp_name  | emp_age | city       | income  |
+--------+-----------+---------+------------+---------+
|    101 | Peter     |      32 | Newyork    |  200000 |
|    102 | Mark      |      32 | California |  300000 |
|    103 | Donald    |      40 | Arizona    | 1000000 |
|    104 | Obama     |      35 | Florida    | 5000000 |
|    105 | Linklon   |      32 | Georgia    |  250000 |
|    106 | Kane      |      45 | Alaska     |  450000 |
|    107 | Adam      |      35 | California | 5000000 |
|    108 | Macculam  |      40 | Florida    |  350000 |
|    109 | Brayan    |      32 | Alaska     |  400000 |
|    110 | Stephen   |      40 | Arizona    |  600000 |
|    111 | Alexander |      45 | California |   70000 |
+--------+-----------+---------+------------+---------+
```

Subquery: Syntax

- Following is a simple SQL statement that returns the employee detail whose id matches in a subquery:
  - SELECT emp_name, city, income FROM employees WHERE emp_id IN (SELECT emp_id FROM employees);



```
mysql> SELECT emp_name, city, income FROM employees
    ->    WHERE emp_id IN (SELECT emp_id FROM employees);
+-----------+------------+---------+
| emp_name  | city       | income  |
+-----------+------------+---------+
| Peter     | Newyork    |  200000 |
| Mark      | California |  300000 |
| Donald    | Arizona    | 1000000 |
| Obama     | Florida    | 5000000 |
| Linklon   | Georgia    |  250000 |
| Kane      | Alaska     |  450000 |
| Adam      | California | 5000000 |
| Macculam  | Florida    |  350000 |
| Brayan    | Alaska     |  400000 |
| Stephen   | Arizona    |  600000 |
| Alexander | California |   70000 |
+-----------+------------+---------+
```

Subquery: With comparison

- A comparison operator is an operator used to compare values and returns the result, either true or false.
- The following comparison operators are used in MySQL <, >, =, <>, <=>, etc. We can use the subquery before or after the comparison operators that return a single value.
- The returned value can be the arithmetic expression or a column function.
- After that, SQL compares the subquery results with the value on the other side of the comparison operator.

Subquery: Example

- Following is a simple SQL statement that returns the employee detail whose income is more than 350000 with the help of subquery:
  - SELECT * FROM employees   WHERE emp_id IN (SELECT emp_id FROM employees WHERE income > 350000);
- Let us see an example of another comparison operator, such as equality (=) to find employee details with maximum income using a subquery.
  - SELECT emp_name, city, income FROM employees WHERE income = (SELECT MAX(income) FROM employees);

Correlated Subquery

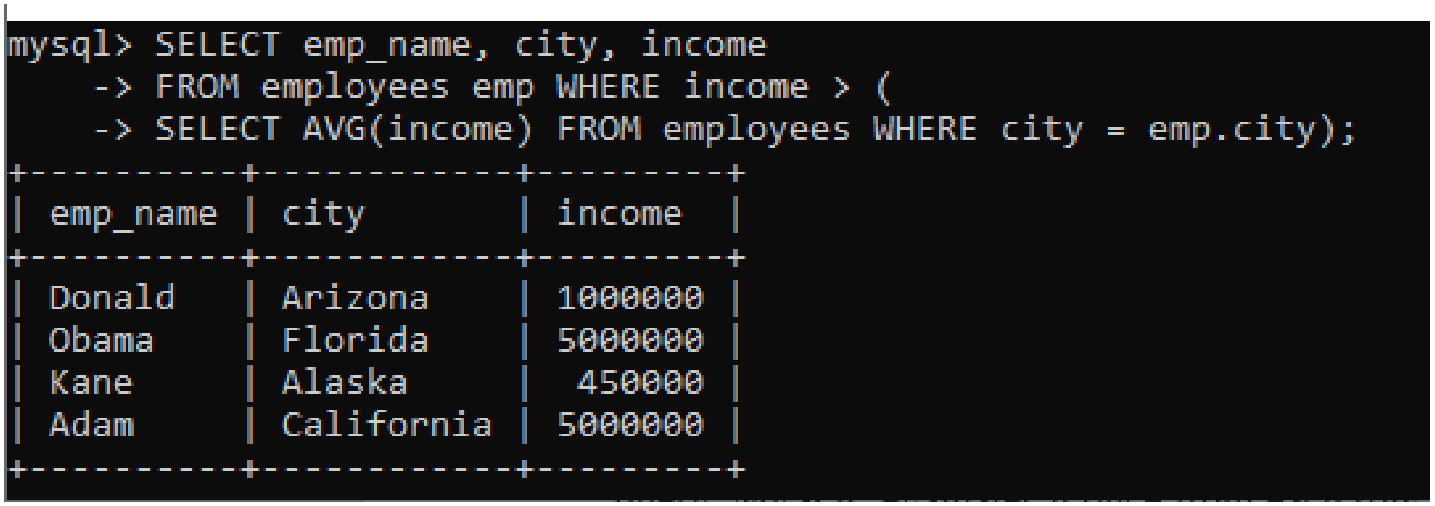
- A correlated subquery in MySQL is a subquery that depends on the outer query.
- It uses the data from the outer query or contains a reference to a parent query that also appears in the outer query.
- MySQL evaluates it once from each row in the outer query.

Correlated Subquery

```
SELECT emp_name, city, income  FROM
employees emp WHERE income > (SELECT
AVG(income) FROM employees WHERE city =
emp.city);
```

- In the above query, we select an employee name and city whose income is higher than the average income of all employees in each city.

Correlated Subquery

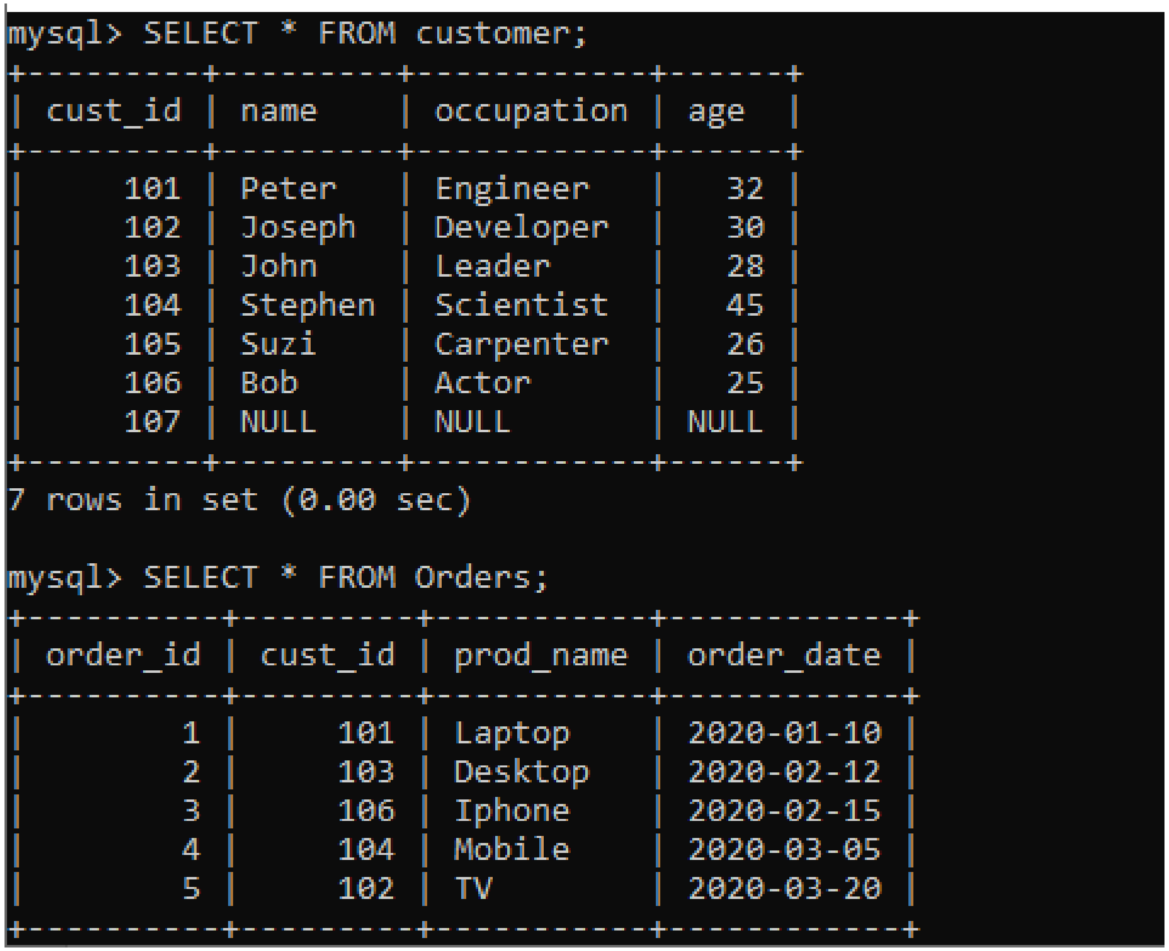


```
mysql> SELECT emp_name, city, income
    -> FROM employees emp WHERE income > (
    -> SELECT AVG(income) FROM employees WHERE city = emp.city);
+----------+------------+---------+
| emp_name | city       | income  |
+----------+------------+---------+
| Donald   | Arizona    | 1000000 |
| Obama    | Florida    | 5000000 |
| Kane     | Alaska     |  450000 |
| Adam     | California | 5000000 |
+----------+------------+---------+
```

Subquery with EXISTS or NOT EXISTS

- The EXISTS operator is a Boolean operator that returns either true or false result.
- It is used with a subquery and checks the existence of data in a subquery.
- If a subquery returns any record at all, this operator returns true. Otherwise, it will return false. The NOT EXISTS operator used for negation that gives true value when the subquery does not return any row.
- Otherwise, it returns false. Both EXISTS and NOT EXISTS used with correlated subqueries. The following example illustrates it more clearly.

Subquery with EXISTS or NOT EXISTS



```
mysql> SELECT * FROM customer;
+---------+---------+------------+------+
| cust_id | name    | occupation | age  |
+---------+---------+------------+------+
|     101 | Peter   | Engineer   |   32 |
|     102 | Joseph  | Developer  |   30 |
|     103 | John    | Leader     |   28 |
|     104 | Stephen | Scientist  |   45 |
|     105 | Suzi    | Carpenter  |   26 |
|     106 | Bob     | Actor      |   25 |
|     107 | NULL    | NULL       | NULL |
+---------+---------+------------+------+
7 rows in set (0.00 sec)

mysql> SELECT * FROM Orders;
+----------+---------+-----------+------------+
| order_id | cust_id | prod_name | order_date |
+----------+---------+-----------+------------+
|        1 |     101 | Laptop    | 2020-01-10 |
|        2 |     103 | Desktop   | 2020-02-12 |
|        3 |     106 | Iphone    | 2020-02-15 |
|        4 |     104 | Mobile    | 2020-03-05 |
|        5 |     102 | TV        | 2020-03-20 |
+----------+---------+-----------+------------+
```

Subquery with EXISTS or NOT EXISTS

- The below SQL statements uses EXISTS operator to find the name, occupation, and age of the customer who has placed at least one order.

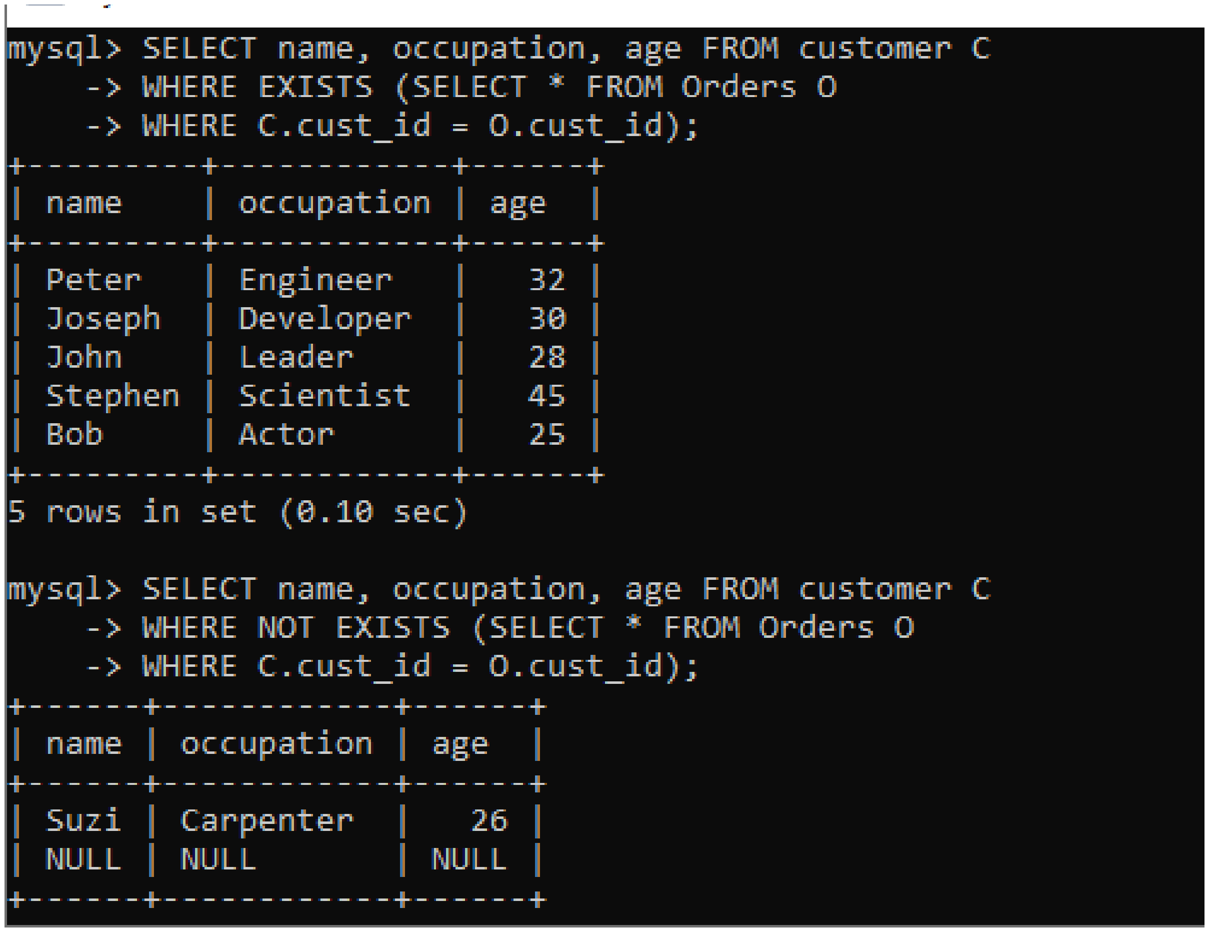
```
SELECT name, occupation, age FROM
customer C  WHERE EXISTS (SELECT * FROM
Orders O WHERE C.cust_id = O.cust_id);
```

Subquery with EXISTS or NOT EXISTS

- This statement uses NOT EXISTS operator that returns the customer details who have not placed an order.

```
SELECT name, occupation, age FROM customer
C   WHERE NOT EXISTS (SELECT * FROM Orders
O   WHERE C.cust_id = O.cust_id);
```

Subquery with EXISTS or NOT EXISTS



```
mysql> SELECT name, occupation, age FROM customer C
    -> WHERE EXISTS (SELECT * FROM Orders O
    -> WHERE C.cust_id = O.cust_id);
+---------+------------+------+
| name    | occupation | age  |
+---------+------------+------+
| Peter   | Engineer   |   32 |
| Joseph  | Developer  |   30 |
| John    | Leader     |   28 |
| Stephen | Scientist  |   45 |
| Bob     | Actor      |   25 |
+---------+------------+------+
5 rows in set (0.10 sec)

mysql> SELECT name, occupation, age FROM customer C
    -> WHERE NOT EXISTS (SELECT * FROM Orders O
    -> WHERE C.cust_id = O.cust_id);
+------+------------+------+
| name | occupation | age  |
+------+------------+------+
| Suzi | Carpenter  |   26 |
| NULL | NULL       | NULL |
+------+------------+------+
```

MySQL JOINS

- MySQL JOINS are used with SELECT statement. It is used to retrieve data from multiple tables. It is performed whenever you need to fetch records from two or more tables.
- There are three types of MySQL joins:
  - MySQL INNER JOIN (or sometimes called simple join)
  - MySQL LEFT OUTER JOIN (or sometimes called LEFT JOIN)
  - MySQL RIGHT OUTER JOIN (or sometimes called RIGHT JOIN)

MySQL Inner JOIN (Simple Join)

- The MySQL INNER JOIN is used to return all rows from multiple tables where the join condition is satisfied. It is the most common type of join.
- Syntax:

```
SELECT columns
FROM table1
INNER JOIN table2
ON table1.column = table2.column;
```

MySQL Inner JOIN (Simple Join)

- Let's take an example:
- Consider two tables "officers" and "students", having the following data.

```
    SELECT officers.officer_name, officers.address,
students.course_name
    FROM officers
    INNER JOIN students
    ON officers.officer_id = students.student_id;
```

MySQL Left Outer Join

- The LEFT OUTER JOIN returns all rows from the left hand table specified in the ON condition and only those rows from the other table where the join condition is fulfilled.
- Syntax:

```
SELECT columns
FROM table1
LEFT [OUTER] JOIN table2
ON table1.column = table2.column;
```

MySQL Left Outer Join

- Let's take an example:
- Consider two tables "officers" and "students", having the following data.

```
 SELECT  officers.officer_name, officers.address,
                    students.course_name
FROM officers
LEFT JOIN students
ON officers.officer_id = students.student_id;
```

MySQL Right Outer Join

- The MySQL Right Outer Join returns all rows from the RIGHT-hand table specified in the ON condition and only those rows from the other table where he join condition is fulfilled.
- Syntax:

```
SELECT columns
FROM table1
RIGHT [OUTER] JOIN table2
ON table1.column = table2.column;
```

MySQL Right Outer Join

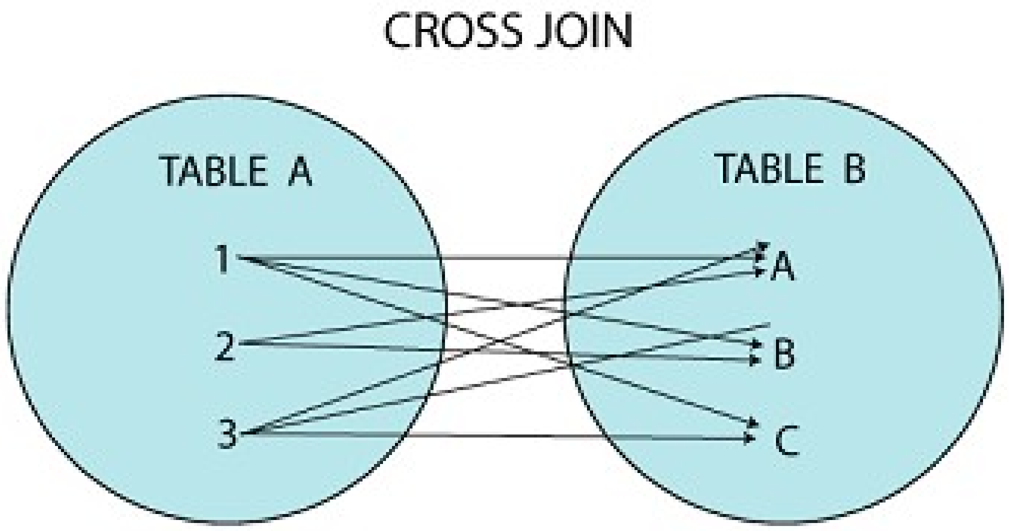
- Let's take an example:
- Consider two tables "officers" and "students", having the following data.

```
 SELECT officers.officer_name, officers.address,
          students.course_name, students.student_name
 FROM officers
 RIGHT JOIN students
 ON officers.officer_id = students.student_id;
```

MySQL CROSS JOIN

- MySQL CROSS JOIN is used to combine all possibilities of the two or more tables and returns the result that contains every row from all contributing tables.
- The CROSS JOIN is also known as CARTESIAN JOIN, which provides the Cartesian product of all associated tables.
- The Cartesian product can be explained as all rows present in the first table multiplied by all rows present in the second table.
- It is similar to the Inner Join, where the join condition is not available with this clause.

MySQL CROSS JOIN



MySQL CROSS JOIN

- The CROSS JOIN keyword is always used with the SELECT statement and must be written after the FROM clause.
- The following syntax fetches all records from both joining tables:

```
SELECT column-lists
FROM table1
CROSS JOIN table2;
```

- In the above syntax, the column-lists is the name of the column or field that you want to return and table1 and table2 is the table name from which you fetch the records.

MySQL EquiJoin

- The process is called joining when we combine two or more tables based on some common columns and a join condition.
- An equijoin is an operation that combines multiple tables based on equality or matching column values in the associated tables.
- We can use the equal sign (=) comparison operator to refer to equality in the WHERE clause.
- This joining operation returns the same result when we use the JOIN keyword with the ON clause and then specifying the column names and their associated tables.

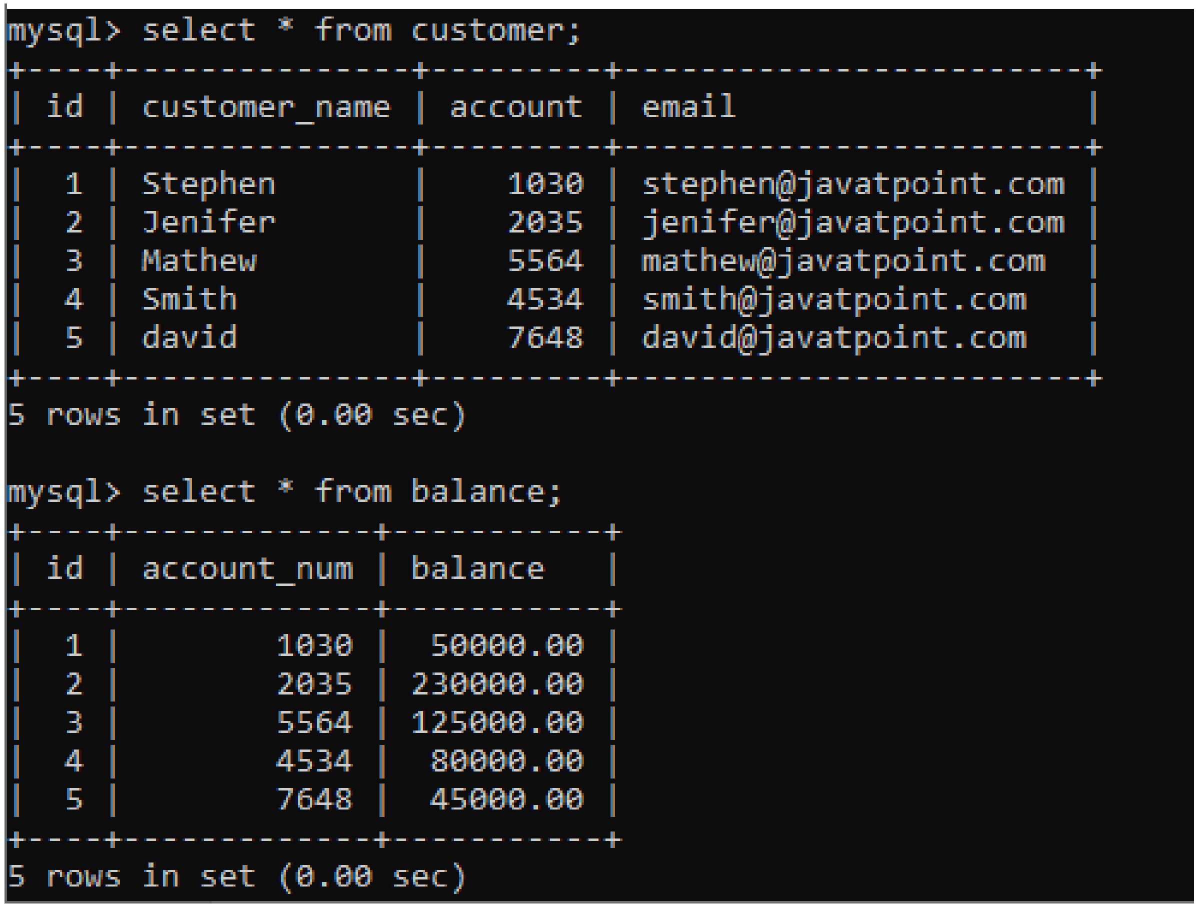
MySQL EquiJoin

```
SELECT column_name (s)
FROM table_name1, table_name2, ...., table_nameN
WHERE table_name1.column_name = table_name2.column_name;
```

OR

```
SELECT (column_list | *)
FROM table_name1
JOIN table_name2
ON table_name1.column_name = table_name2.column_name;
```

MySQL EquiJoin

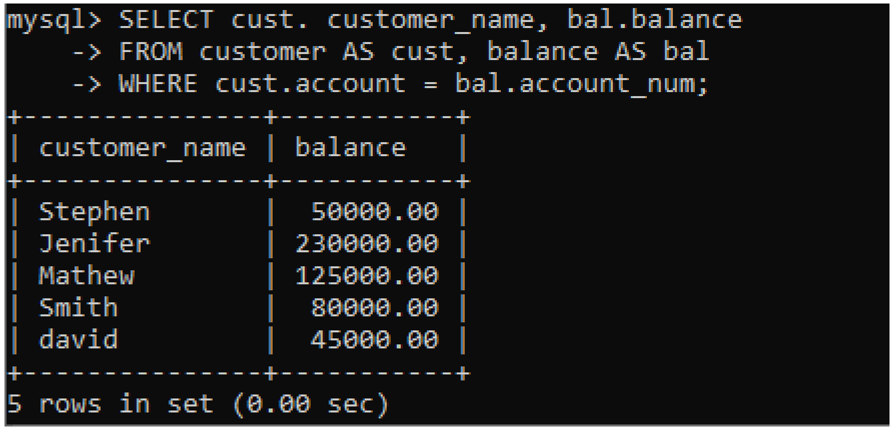


```
mysql> select * from customer;
+----+---------------+---------+-----------------------+
| id | customer_name | account | email                 |
+----+---------------+---------+-----------------------+
|  1 | Stephen       |    1030 | stephen@javatpoint.com |
|  2 | Jenifer       |    2035 | jenifer@javatpoint.com |
|  3 | Mathew        |    5564 | mathew@javatpoint.com  |
|  4 | Smith         |    4534 | smith@javatpoint.com   |
|  5 | david         |    7648 | david@javatpoint.com   |
+----+---------------+---------+-----------------------+
5 rows in set (0.00 sec)

mysql> select * from balance;
+----+-------------+-----------+
| id | account_num | balance   |
+----+-------------+-----------+
|  1 |        1030 |  50000.00 |
|  2 |        2035 | 230000.00 |
|  3 |        5564 | 125000.00 |
|  4 |        4534 |  80000.00 |
|  5 |        7648 |  45000.00 |
+----+-------------+-----------+
5 rows in set (0.00 sec)
```

MySQL EquiJoin

```
mysql> SELECT cust. customer_name, bal.balance
FROM customer AS cust, balance AS bal
WHERE cust.account = bal.account_num;
```



```
mysql> SELECT cust. customer_name, bal.balance
    -> FROM customer AS cust, balance AS bal
    -> WHERE cust.account = bal.account_num;
+---------------+-----------+
| customer_name | balance   |
+---------------+-----------+
| Stephen       |  50000.00 |
| Jenifer       | 230000.00 |
| Mathew        | 125000.00 |
| Smith         |  80000.00 |
| david         |  45000.00 |
+---------------+-----------+
5 rows in set (0.00 sec)
```

Union

- MySQL Union clause allows us to combine two or more relations using multiple SELECT queries into a single result set. By default, it has a feature to remove the duplicate rows from the result set.
- Union clause in MySQL must follow the rules given below:
  - The order and number of the columns must be the same in all tables.
  - The data type must be compatible with the corresponding positions of each select query.
  - The column name in the SELECT queries should be in the same order.

Union

```
SELECT column_name(s) FROM table_name1
UNION
SELECT column_name(s) FROM table_name2;
```

Example:

```
SELECT stud_name, subject FROM student1
UNION
SELECT stud_name, subject FROM student2;
```

MySQL Copy/Clone/Duplicate Table

- MySQL copy or clone table is a feature that allows us to create a duplicate table of an existing table, including the table structure, indexes, constraints, default values, etc.
- Copying data of an existing table into a new table is very useful in a situation like backing up data in table failure.
- It is also advantageous when we need to test or perform something without affecting the original table, for example, replicating the production data for testing.

MySQL Copy/Clone/Duplicate Table

- We can copy an existing table to a new table using the CREATE TABLE and SELECT statement, as shown below:

```
CREATE TABLE new_table_name
SELECT column1, column2, column3
FROM existing_table_name;
```

MySQL Copy/Clone/Duplicate Table

```
CREATE TABLE IF NOT EXISTS new_table_name
SELECT column1, column2, column3
FROM existing_table_name
WHERE condition;
```

MySQL Copy/Clone/Duplicate Table

- CREATE TABLE IF NOT EXISTS new_table_name LIKE existing_table_name;

- INSERT new_table_name SELECT * FROM existing_table_name;

Thank you

@mitu_skillologies

@mITuSkillologies

@mitu_group

@mitu-skillologies

@MITUSkillologies

kaggle

@mituskillologies

Web Resources
https://mitu.co.in
http://tusharkute.com

@mituskillologies

contact@mitu.co.in

tushar@tusharkute.com