Constraints are the rules enforced on the data columns of a table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database.

Constraints could be either on a column level or a table level. The column level constraints are applied only +to one column, whereas the table level constraints are applied to the whole table.

Following are some of the most commonly used constraints available in SQL. These constraints have already been discussed in [SQL - RDBMS Concepts](https://www.tutorialspoint.com/sql/sql-rdbms-concepts.htm)chapter, but it’s worth to revise them at this point.

* [CHECK Constraint](https://www.tutorialspoint.com/sql/sql-check.htm) − The CHECK constraint ensures that all the values in a column satisfies certain conditions.
* [INDEX](https://www.tutorialspoint.com/sql/sql-index.htm) − Used to create and retrieve data from the database very quickly.

Constraints can be specified when a table is created with the CREATE TABLE statement or you can use the ALTER TABLE statement to create constraints even after the table is created.

Dropping Constraints

Any constraint that you have defined can be dropped using the ALTER TABLE command with the DROP CONSTRAINT option.

For example, to drop the primary key constraint in the EMPLOYEES table, you can use the following command.

ALTER TABLE EMPLOYEES DROP CONSTRAINT EMPLOYEES\_PK;

Some implementations may provide shortcuts for dropping certain constraints. For example, to drop the primary key constraint for a table in Oracle, you can use the following command.

ALTER TABLE EMPLOYEES DROP PRIMARY KEY;

Some implementations allow you to disable constraints. Instead of permanently dropping a constraint from the database, you may want to temporarily disable the constraint and then enable it later.

Integrity Constraints

Integrity constraints are used to ensure accuracy and consistency of the data in a relational database. Data integrity is handled in a relational database through the concept of referential integrity.

There are many types of integrity constraints that play a role in **Referential Integrity (RI)**. These constraints include Primary Key, Foreign Key, Unique Constraints and other constraints which are mentioned above.

By default, a column can hold NULL values. If you do not want a column to have a NULL value, then you need to define such a constraint on this column specifying that NULL is now not allowed for that column.

A NULL is not the same as no data, rather, it represents unknown data.

## Example

For example, the following SQL query creates a new table called CUSTOMERS and adds five columns, three of which, are ID NAME and AGE, In this we specify not to accept NULLs −

CREATE TABLE CUSTOMERS(

ID INT NOT NULL,

NAME VARCHAR (20) NOT NULL,

AGE INT NOT NULL,

ADDRESS CHAR (25) ,

SALARY DECIMAL (18, 2),

PRIMARY KEY (ID)

);

If CUSTOMERS table has already been created, then to add a NOT NULL constraint to the SALARY column in Oracle and MySQL, you would write a query like the one that is shown in the following code block.

ALTER TABLE CUSTOMERS

MODIFY SALARY DECIMAL (18, 2) NOT NULL;

The DEFAULT constraint provides a default value to a column when the INSERT INTO statement does not provide a specific value.

## Example

For example, the following SQL creates a new table called CUSTOMERS and adds five columns. Here, the SALARY column is set to 5000.00 by default, so in case the INSERT INTO statement does not provide a value for this column, then by default this column would be set to 5000.00.

CREATE TABLE CUSTOMERS(

ID INT NOT NULL,

NAME VARCHAR (20) NOT NULL,

AGE INT NOT NULL,

ADDRESS CHAR (25) ,

SALARY DECIMAL (18, 2) DEFAULT 5000.00,

PRIMARY KEY (ID)

);

If the CUSTOMERS table has already been created, then to add a DEFAULT constraint to the SALARY column, you would write a query like the one which is shown in the code block below.

## ALTER TABLE CUSTOMERS

MODIFY SALARY DECIMAL (18, 2) DEFAULT 5000.00;

## Drop Default Constraint

To drop a DEFAULT constraint, use the following SQL query.

ALTER TABLE CUSTOMERS

ALTER COLUMN SALARY DROP DEFAULT;

The UNIQUE Constraint prevents two records from having identical values in a column. In the CUSTOMERS table, for example, you might want to prevent two or more people from having an identical age.

### **Example**

For example, the following SQL query creates a new table called CUSTOMERS and adds five columns. Here, the AGE column is set to UNIQUE, so that you cannot have two records with the same age.

CREATE TABLE CUSTOMERS(

ID INT NOT NULL,

NAME VARCHAR (20) NOT NULL,

AGE INT NOT NULL UNIQUE,

ADDRESS CHAR (25) ,

SALARY DECIMAL (18, 2),

PRIMARY KEY (ID)

);

If the CUSTOMERS table has already been created, then to add a UNIQUE constraint to the AGE column. You would write a statement like the query that is given in the code block below.

ALTER TABLE CUSTOMERS

MODIFY AGE INT NOT NULL UNIQUE;

You can also use the following syntax, which supports naming the constraint in multiple columns as well.

ALTER TABLE CUSTOMERS

ADD CONSTRAINT myUniqueConstraint UNIQUE(AGE, SALARY);

## DROP a UNIQUE Constraint

To drop a UNIQUE constraint, use the following SQL query.

ALTER TABLE CUSTOMERS

DROP CONSTRAINT myUniqueConstraint;

If you are using MySQL, then you can use the following syntax −

ALTER TABLE CUSTOMERS

DROP INDEX myUniqueConstraint;

A primary key is a field in a table which uniquely identifies each row/record in a database table. Primary keys must contain unique values. A primary key column cannot have NULL values.

A table can have only one primary key, which may consist of single or multiple fields. When multiple fields are used as a primary key, they are called a composite key.

If a table has a primary key defined on any field(s), then you cannot have two records having the same value of that field(s).

**Note** − You would use these concepts while creating database tables.

## Create Primary Key

Here is the syntax to define the ID attribute as a primary key in a CUSTOMERS table.

CREATE TABLE CUSTOMERS(

ID INT NOT NULL,

NAME VARCHAR (20) NOT NULL,

AGE INT NOT NULL,

ADDRESS CHAR (25) ,

SALARY DECIMAL (18, 2),

PRIMARY KEY (ID)

);

To create a PRIMARY KEY constraint on the "ID" column when the CUSTOMERS table already exists, use the following SQL syntax −

ALTER TABLE CUSTOMER ADD PRIMARY KEY (ID);

**NOTE** − If you use the ALTER TABLE statement to add a primary key, the primary key column(s) should have already been declared to not contain NULL values (when the table was first created).

For defining a PRIMARY KEY constraint on multiple columns, use the SQL syntax given below.

CREATE TABLE CUSTOMERS(

ID INT NOT NULL,

NAME VARCHAR (20) NOT NULL,

AGE INT NOT NULL,

ADDRESS CHAR (25) ,

SALARY DECIMAL (18, 2),

PRIMARY KEY (ID, NAME)

);

To create a PRIMARY KEY constraint on the "ID" and "NAMES" columns when CUSTOMERS table already exists, use the following SQL syntax.

ALTER TABLE CUSTOMERS

ADD CONSTRAINT PK\_CUSTID PRIMARY KEY (ID, NAME);

## Delete Primary Key

You can clear the primary key constraints from the table with the syntax given below.

ALTER TABLE CUSTOMERS DROP PRIMARY KEY ;

A foreign key is a key used to link two tables together. This is sometimes also called as a referencing key.

A Foreign Key is a column or a combination of columns whose values match a Primary Key in a different table.

**The relationship between 2 tables matches the Primary Key in one of the tables with a Foreign Key in the second table.**

If a table has a primary key defined on any field(s), then you cannot have two records having the same value of that field(s).

### **Example**

Consider the structure of the following two tables.

**CUSTOMERS table**

CREATE TABLE CUSTOMERS(

ID INT NOT NULL,

NAME VARCHAR (20) NOT NULL,

AGE INT NOT NULL,

ADDRESS CHAR (25) ,

SALARY DECIMAL (18, 2),

PRIMARY KEY (ID)

);

**ORDERS table**

CREATE TABLE ORDERS (

ID INT NOT NULL,

DATE DATETIME,

CUSTOMER\_ID INT references CUSTOMERS(ID),

AMOUNT double,

PRIMARY KEY (ID)

);

If the ORDERS table has already been created and the foreign key has not yet been set, the use the syntax for specifying a foreign key by altering a table.

ALTER TABLE ORDERS

ADD FOREIGN KEY (Customer\_ID) REFERENCES CUSTOMERS (ID);

## DROP a FOREIGN KEY Constraint

To drop a FOREIGN KEY constraint, use the following SQL syntax.

ALTER TABLE ORDERS

DROP FOREIGN KEY;

The CHECK Constraint enables a condition to check the value being entered into a record. If the condition evaluates to false, the record violates the constraint and isn't entered the table.

### **Example**

For example, the following program creates a new table called CUSTOMERS and adds five columns. Here, we add a CHECK with AGE column, so that you cannot have any CUSTOMER who is below 18 years.

CREATE TABLE CUSTOMERS(

ID INT NOT NULL,

NAME VARCHAR (20) NOT NULL,

AGE INT NOT NULL CHECK (AGE >= 18),

ADDRESS CHAR (25) ,

SALARY DECIMAL (18, 2),

PRIMARY KEY (ID)

);

If the CUSTOMERS table has already been created, then to add a CHECK constraint to AGE column, you would write a statement like the one given below.

ALTER TABLE CUSTOMERS

MODIFY AGE INT NOT NULL CHECK (AGE >= 18 );

You can also use the following syntax, which supports naming the constraint in multiple columns as well −

ALTER TABLE CUSTOMERS

ADD CONSTRAINT myCheckConstraint CHECK(AGE >= 18);

## DROP a CHECK Constraint

To drop a CHECK constraint, use the following SQL syntax. This syntax does not work with MySQL.

ALTER TABLE CUSTOMERS

DROP CONSTRAINT myCheckConstraint;

he INDEX is used to create and retrieve data from the database very quickly. An Index can be created by using a single or group of columns in a table. When the index is created, it is assigned a ROWID for each row before it sorts out the data.

Proper indexes are good for performance in large databases, but you need to be careful while creating an index. A Selection of fields depends on what you are using in your SQL queries.

### **Example**

For example, the following SQL syntax creates a new table called CUSTOMERS and adds five columns in it.

CREATE TABLE CUSTOMERS(

ID INT NOT NULL,

NAME VARCHAR (20) NOT NULL,

AGE INT NOT NULL,

ADDRESS CHAR (25) ,

SALARY DECIMAL (18, 2),

PRIMARY KEY (ID)

);

Now, you can create an index on a single or multiple columns using the syntax given below.

CREATE INDEX index\_name

ON table\_name ( column1, column2.....);

To create an INDEX on the AGE column, to optimize the search on customers for a specific age, you can use the follow SQL syntax which is given below −

CREATE INDEX idx\_age

ON CUSTOMERS ( AGE );

## DROP an INDEX Constraint

To drop an INDEX constraint, use the following SQL syntax.

ALTER TABLE CUSTOMERS

DROP INDEX idx\_age;

Joins

The SQL **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.

Consider the following two tables −

**Table 1** − CUSTOMERS Table

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

**Table 2** − ORDERS Table

+-----+---------------------+-------------+--------+

|OID | DATE | CUSTOMER\_ID | AMOUNT |

+-----+---------------------+-------------+--------+

| 102 | 2009-10-08 00:00:00 | 3 | 3000 |

| 100 | 2009-10-08 00:00:00 | 3 | 1500 |

| 101 | 2009-11-20 00:00:00 | 2 | 1560 |

| 103 | 2008-05-20 00:00:00 | 4 | 2060 |

+-----+---------------------+-------------+--------+

Now, let us join these two tables in our SELECT statement as shown below.

SQL> SELECT ID, NAME, AGE, AMOUNT

FROM CUSTOMERS, ORDERS

WHERE CUSTOMERS.ID = ORDERS.CUSTOMER\_ID;

This would produce the following result.

+----+----------+-----+--------+

| ID | NAME | AGE | AMOUNT |

+----+----------+-----+--------+

| 3 | kaushik | 23 | 3000 |

| 3 | kaushik | 23 | 1500 |

| 2 | Khilan | 25 | 1560 |

| 4 | Chaitali | 25 | 2060 |

+----+----------+-----+--------+

Here, it is noticeable that the join is performed in the WHERE clause. Several operators can be used to join tables, such as =, <, >, <>, <=, >=, !=, BETWEEN, LIKE, and NOT; they can all be used to join tables. However, the most common operator is the equal to symbol.

There are different types of joins available in SQL −

* [INNER JOIN](https://www.tutorialspoint.com/sql/sql-inner-joins.htm) − returns rows when there is a match in both tables.
* [LEFT JOIN](https://www.tutorialspoint.com/sql/sql-left-joins.htm) − returns all rows from the left table, even if there are no matches in the right table.
* [RIGHT JOIN](https://www.tutorialspoint.com/sql/sql-right-joins.htm) − returns all rows from the right table, even if there are no matches in the left table.
* [FULL JOIN](https://www.tutorialspoint.com/sql/sql-full-joins.htm) − returns rows when there is a match in one of the tables.
* [SELF JOIN](https://www.tutorialspoint.com/sql/sql-self-joins.htm) − is used to join a table to itself as if the table were two tables, temporarily renaming at least one table in the SQL statement.
* [CARTESIAN JOIN](https://www.tutorialspoint.com/sql/sql-cartesian-joins.htm) − returns the Cartesian product of the sets of records from the two or more joined tables.

Let us now discuss each of these joins in detail.

The most important and frequently used of the joins is the **INNER JOIN**. They are also referred to as an **EQUIJOIN**.

The 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,

<http://www.way2tutorial.com/sql/sql_equi_join.php>

column values for each matched pair of rows of A and B are combined into a result row.

## Syntax

The basic syntax of the **INNER JOIN** is as follows.

SELECT table1.column1, table2.column2...

FROM table1

INNER JOIN table2

ON table1.common\_field = table2.common\_field;

## Example

Consider the following two tables.

**Table 1** − CUSTOMERS Table is as follows.

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

**Table 2** − ORDERS Table is as follows.

+-----+---------------------+-------------+--------+

| OID | DATE | CUSTOMER\_ID | AMOUNT |

+-----+---------------------+-------------+--------+

| 102 | 2009-10-08 00:00:00 | 3 | 3000 |

| 100 | 2009-10-08 00:00:00 | 3 | 1500 |

| 101 | 2009-11-20 00:00:00 | 2 | 1560 |

| 103 | 2008-05-20 00:00:00 | 4 | 2060 |

+-----+---------------------+-------------+--------+

Now, let us join these two tables using the INNER JOIN as follows −

SQL> SELECT ID, NAME, AMOUNT, DATE

FROM CUSTOMERS

INNER JOIN ORDERS

ON CUSTOMERS.ID = ORDERS.CUSTOMER\_ID;

This would produce the following result.

+----+----------+--------+---------------------+

| ID | NAME | AMOUNT | DATE |

+----+----------+--------+---------------------+

| 3 | kaushik | 3000 | 2009-10-08 00:00:00 |

| 3 | kaushik | 1500 | 2009-10-08 00:00:00 |

| 2 | Khilan | 1560 | 2009-11-20 00:00:00 |

| 4 | Chaitali | 2060 | 2008-05-20 00:00:00 |

+----+----------+--------+---------------------+

The SQL **LEFT JOIN** returns all rows from the left table, even if there are no matches in the right table. This means that if the ON clause matches 0 (zero) records in the right table; the join will still return a row in the result, but with NULL in each column from the right table.

This means that a left join returns all the values from the left table, plus matched values from the right table or NULL in case of no matching join predicate.

## Syntax

The basic syntax of a **LEFT JOIN** is as follows.

SELECT table1.column1, table2.column2...

FROM table1

LEFT JOIN table2

ON table1.common\_field = table2.common\_field;

Here, the given condition could be any given expression based on your requirement.

## Example

Consider the following two tables,

**Table 1** − CUSTOMERS Table is as follows.

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

**Table 2** − Orders Table is as follows.

+-----+---------------------+-------------+--------+

| OID | DATE | CUSTOMER\_ID | AMOUNT |

+-----+---------------------+-------------+--------+

| 102 | 2009-10-08 00:00:00 | 3 | 3000 |

| 100 | 2009-10-08 00:00:00 | 3 | 1500 |

| 101 | 2009-11-20 00:00:00 | 2 | 1560 |

| 103 | 2008-05-20 00:00:00 | 4 | 2060 |

+-----+---------------------+-------------+--------+

Now, let us join these two tables using the LEFT JOIN as follows.

SQL> SELECT ID, NAME, AMOUNT, DATE

FROM CUSTOMERS

LEFT JOIN ORDERS

ON CUSTOMERS.ID = ORDERS.CUSTOMER\_ID;

This would produce the following result −

+----+----------+--------+---------------------+

| ID | NAME | AMOUNT | DATE |

+----+----------+--------+---------------------+

| 1 | Ramesh | NULL | NULL |

| 2 | Khilan | 1560 | 2009-11-20 00:00:00 |

| 3 | kaushik | 3000 | 2009-10-08 00:00:00 |

| 3 | kaushik | 1500 | 2009-10-08 00:00:00 |

| 4 | Chaitali | 2060 | 2008-05-20 00:00:00 |

| 5 | Hardik | NULL | NULL |

| 6 | Komal | NULL | NULL |

| 7 | Muffy | NULL | NULL |

+----+----------+--------+---------------------+

The SQL **RIGHT JOIN** returns all rows from the right table, even if there are no matches in the left table. This means that if the ON clause matches 0 (zero) records in the left table; the join will still return a row in the result, but with NULL in each column from the left table.

This means that a right join returns all the values from the right table, plus matched values from the left table or NULL in case of no matching join predicate.

## Syntax

The basic syntax of a **RIGHT JOIN** is as follow.

SELECT table1.column1, table2.column2...

FROM table1

RIGHT JOIN table2

ON table1.common\_field = table2.common\_field;

## Example

Consider the following two tables,

**Table 1** − CUSTOMERS Table is as follows.

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

**Table 2** − ORDERS Table is as follows.

+-----+---------------------+-------------+--------+

|OID | DATE | CUSTOMER\_ID | AMOUNT |

+-----+---------------------+-------------+--------+

| 102 | 2009-10-08 00:00:00 | 3 | 3000 |

| 100 | 2009-10-08 00:00:00 | 3 | 1500 |

| 101 | 2009-11-20 00:00:00 | 2 | 1560 |

| 103 | 2008-05-20 00:00:00 | 4 | 2060 |

+-----+---------------------+-------------+--------+

Now, let us join these two tables using the RIGHT JOIN as follows.

SQL> SELECT ID, NAME, AMOUNT, DATE

FROM CUSTOMERS

RIGHT JOIN ORDERS

ON CUSTOMERS.ID = ORDERS.CUSTOMER\_ID;

This would produce the following result −

+------+----------+--------+---------------------+

| ID | NAME | AMOUNT | DATE |

+------+----------+--------+---------------------+

| 3 | kaushik | 3000 | 2009-10-08 00:00:00 |

| 3 | kaushik | 1500 | 2009-10-08 00:00:00 |

| 2 | Khilan | 1560 | 2009-11-20 00:00:00 |

| 4 | Chaitali | 2060 | 2008-05-20 00:00:00 |

+------+----------+--------+---------------------+

The SQL **FULL JOIN** combines the results of both left and right outer joins.

The joined table will contain all records from both the tables and fill in NULLs for missing matches on either side.

## Syntax

The basic syntax of a **FULL JOIN** is as follows −

SELECT table1.column1, table2.column2...

FROM table1

FULL JOIN table2

ON table1.common\_field = table2.common\_field;

Here, the given condition could be any given expression based on your requirement.

## Example

Consider the following two tables.

**Table 1** − CUSTOMERS Table is as follows.

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

**Table 2** − ORDERS Table is as follows.

+-----+---------------------+-------------+--------+

|OID | DATE | CUSTOMER\_ID | AMOUNT |

+-----+---------------------+-------------+--------+

| 102 | 2009-10-08 00:00:00 | 3 | 3000 |

| 100 | 2009-10-08 00:00:00 | 3 | 1500 |

| 101 | 2009-11-20 00:00:00 | 2 | 1560 |

| 103 | 2008-05-20 00:00:00 | 4 | 2060 |

+-----+---------------------+-------------+--------+

Now, let us join these two tables using FULL JOIN as follows.

SQL> SELECT ID, NAME, AMOUNT, DATE

FROM CUSTOMERS

FULL JOIN ORDERS

ON CUSTOMERS.ID = ORDERS.CUSTOMER\_ID;

This would produce the following result −

+------+----------+--------+---------------------+

| ID | NAME | AMOUNT | DATE |

+------+----------+--------+---------------------+

| 1 | Ramesh | NULL | NULL |

| 2 | Khilan | 1560 | 2009-11-20 00:00:00 |

| 3 | kaushik | 3000 | 2009-10-08 00:00:00 |

| 3 | kaushik | 1500 | 2009-10-08 00:00:00 |

| 4 | Chaitali | 2060 | 2008-05-20 00:00:00 |

| 5 | Hardik | NULL | NULL |

| 6 | Komal | NULL | NULL |

| 7 | Muffy | NULL | NULL |

| 3 | kaushik | 3000 | 2009-10-08 00:00:00 |

| 3 | kaushik | 1500 | 2009-10-08 00:00:00 |

| 2 | Khilan | 1560 | 2009-11-20 00:00:00 |

| 4 | Chaitali | 2060 | 2008-05-20 00:00:00 |

+------+----------+--------+---------------------+

If your Database does not support FULL JOIN (MySQL does not support FULL JOIN), then you can use **UNION ALL** clause to combine these two JOINS as shown below.

SQL> SELECT ID, NAME, AMOUNT, DATE

FROM CUSTOMERS

LEFT JOIN ORDERS

ON CUSTOMERS.ID = ORDERS.CUSTOMER\_ID

UNION ALL

SELECT ID, NAME, AMOUNT, DATE

FROM CUSTOMERS

RIGHT JOIN ORDERS

ON CUSTOMERS.ID = ORDERS.CUSTOMER\_ID

The SQL **SELF JOIN** is used to join a table to itself as if the table were two tables; temporarily renaming at least one table in the SQL statement.

## Syntax

The basic syntax of SELF JOIN is as follows −

SELECT a.column\_name, b.column\_name...

FROM table1 a, table1 b

WHERE a.common\_field = b.common\_field;

Here, the WHERE clause could be any given expression based on your requirement.

## Example

Consider the following table.

**CUSTOMERS Table** is as follows.

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

Now, let us join this table using SELF JOIN as follows −

SQL> SELECT a.ID, b.NAME, a.SALARY

FROM CUSTOMERS a, CUSTOMERS b

WHERE a.SALARY < b.SALARY;

This would produce the following result −

+----+----------+---------+

| ID | NAME | SALARY |

+----+----------+---------+

| 2 | Ramesh | 1500.00 |

| 2 | kaushik | 1500.00 |

| 1 | Chaitali | 2000.00 |

| 2 | Chaitali | 1500.00 |

| 3 | Chaitali | 2000.00 |

| 6 | Chaitali | 4500.00 |

| 1 | Hardik | 2000.00 |

| 2 | Hardik | 1500.00 |

| 3 | Hardik | 2000.00 |

| 4 | Hardik | 6500.00 |

| 6 | Hardik | 4500.00 |

| 1 | Komal | 2000.00 |

| 2 | Komal | 1500.00 |

| 3 | Komal | 2000.00 |

| 1 | Muffy | 2000.00 |

| 2 | Muffy | 1500.00 |

| 3 | Muffy | 2000.00 |

| 4 | Muffy | 6500.00 |

| 5 | Muffy | 8500.00 |

| 6 | Muffy | 4500.00 |

+----+----------+---------+

The CARTESIAN JOIN or CROSS JOIN returns the Cartesian product of the sets of records from two or more joined tables. Thus, it equates to an inner join where the join-condition always evaluates to either True or where the join-condition is absent from the statement.

## Syntax

The basic syntax of the **CARTESIAN JOIN** or the **CROSS JOIN** is as follows −

SELECT table1.column1, table2.column2...

FROM table1, table2 [, table3 ]

## Example

Consider the following two tables.

**Table 1** − CUSTOMERS table is as follows.

+----+----------+-----+-----------+----------+

| ID | NAME | AGE | ADDRESS | SALARY |

+----+----------+-----+-----------+----------+

| 1 | Ramesh | 32 | Ahmedabad | 2000.00 |

| 2 | Khilan | 25 | Delhi | 1500.00 |

| 3 | kaushik | 23 | Kota | 2000.00 |

| 4 | Chaitali | 25 | Mumbai | 6500.00 |

| 5 | Hardik | 27 | Bhopal | 8500.00 |

| 6 | Komal | 22 | MP | 4500.00 |

| 7 | Muffy | 24 | Indore | 10000.00 |

+----+----------+-----+-----------+----------+

Table 2: ORDERS Table is as follows −

+-----+---------------------+-------------+--------+

|OID | DATE | CUSTOMER\_ID | AMOUNT |

+-----+---------------------+-------------+--------+

| 102 | 2009-10-08 00:00:00 | 3 | 3000 |

| 100 | 2009-10-08 00:00:00 | 3 | 1500 |

| 101 | 2009-11-20 00:00:00 | 2 | 1560 |

| 103 | 2008-05-20 00:00:00 | 4 | 2060 |

+-----+---------------------+-------------+--------+

Now, let us join these two tables using CARTESIAN JOIN as follows −

SQL> SELECT ID, NAME, AMOUNT, DATE

FROM CUSTOMERS, ORDERS;

This would produce the following result −

+----+----------+--------+---------------------+

| ID | NAME | AMOUNT | DATE |

+----+----------+--------+---------------------+

| 1 | Ramesh | 3000 | 2009-10-08 00:00:00 |

| 1 | Ramesh | 1500 | 2009-10-08 00:00:00 |

| 1 | Ramesh | 1560 | 2009-11-20 00:00:00 |

| 1 | Ramesh | 2060 | 2008-05-20 00:00:00 |

| 2 | Khilan | 3000 | 2009-10-08 00:00:00 |

| 2 | Khilan | 1500 | 2009-10-08 00:00:00 |

| 2 | Khilan | 1560 | 2009-11-20 00:00:00 |

| 2 | Khilan | 2060 | 2008-05-20 00:00:00 |

| 3 | kaushik | 3000 | 2009-10-08 00:00:00 |

| 3 | kaushik | 1500 | 2009-10-08 00:00:00 |

| 3 | kaushik | 1560 | 2009-11-20 00:00:00 |

| 3 | kaushik | 2060 | 2008-05-20 00:00:00 |

| 4 | Chaitali | 3000 | 2009-10-08 00:00:00 |

| 4 | Chaitali | 1500 | 2009-10-08 00:00:00 |

| 4 | Chaitali | 1560 | 2009-11-20 00:00:00 |

| 4 | Chaitali | 2060 | 2008-05-20 00:00:00 |

| 5 | Hardik | 3000 | 2009-10-08 00:00:00 |

| 5 | Hardik | 1500 | 2009-10-08 00:00:00 |

| 5 | Hardik | 1560 | 2009-11-20 00:00:00 |

| 5 | Hardik | 2060 | 2008-05-20 00:00:00 |

| 6 | Komal | 3000 | 2009-10-08 00:00:00 |

| 6 | Komal | 1500 | 2009-10-08 00:00:00 |

| 6 | Komal | 1560 | 2009-11-20 00:00:00 |

| 6 | Komal | 2060 | 2008-05-20 00:00:00 |

| 7 | Muffy | 3000 | 2009-10-08 00:00:00 |

| 7 | Muffy | 1500 | 2009-10-08 00:00:00 |

| 7 | Muffy | 1560 | 2009-11-20 00:00:00 |

| 7 | Muffy | 2060 | 2008-05-20 00:00:00 |

+----+----------+--------+---------------------+

The following table has a list of all the important Date and Time related functions available through SQL. There are various other functions supported by your RDBMS. The given list is based on MySQL RDBMS.

|  |  |
| --- | --- |
| **Sr.No.** | **Function & Description** |
| 1 | [**ADDDATE()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_adddate)  Adds dates |
| 2 | [**ADDTIME()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_addtime)  Adds time |
| 3 | [**CONVERT\_TZ()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_convert-tz)  Converts from one timezone to another |
| 4 | [**CURDATE()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_curdate)  Returns the current date |
| 5 | [**CURRENT\_DATE(), CURRENT\_DATE**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_current-date)  Synonyms for CURDATE() |
| 6 | [**CURRENT\_TIME(), CURRENT\_TIME**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_current-time)  Synonyms for CURTIME() |
| 7 | [**CURRENT\_TIMESTAMP(), CURRENT\_TIMESTAMP**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_current-timestamp)  Synonyms for NOW() |
| 8 | [**CURTIME()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_curtime)  Returns the current time |
| 9 | [**DATE\_ADD()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_date-add)  Adds two dates |
| 10 | [**DATE\_FORMAT()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_date-format)  Formats date as specified |
| 11 | [**DATE\_SUB()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_date-sub)  Subtracts two dates |
| 12 | [**DATE()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_date)  Extracts the date part of a date or datetime expression |
| 13 | [**DATEDIFF()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_datediff)  Subtracts two dates |
| 14 | [**DAY()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_day)  Synonym for DAYOFMONTH() |
| 15 | [**DAYNAME()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_dayname)  Returns the name of the weekday |
| 16 | [**DAYOFMONTH()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_dayofmonth)  Returns the day of the month (1-31) |
| 17 | [**DAYOFWEEK()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_dayofweek)  Returns the weekday index of the argument |
| 18 | [**DAYOFYEAR()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_dayofyear)  Returns the day of the year (1-366) |
| 19 | [**EXTRACT**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_extract)  Extracts part of a date |
| 20 | [**FROM\_DAYS()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_from-days)  Converts a day number to a date |
| 21 | [**FROM\_UNIXTIME()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_from-unixtime)  Formats date as a UNIX timestamp |
| 22 | [**HOUR()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_hour)  Extracts the hour |
| 23 | [**LAST\_DAY**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_last-day)  Returns the last day of the month for the argument |
| 24 | [**LOCALTIME(), LOCALTIME**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_localtime)  Synonym for NOW() |
| 25 | [**LOCALTIMESTAMP, LOCALTIMESTAMP()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_localtimestamp)  Synonym for NOW() |
| 26 | [**MAKEDATE()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_makedate)  Creates a date from the year and day of year |
| 27 | [**MAKETIME**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_maketime)  MAKETIME() |
| 28 | [**MICROSECOND()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_microsecond)  Returns the microseconds from argument |
| 29 | [**MINUTE()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_minute)  Returns the minute from the argument |
| 30 | [**MONTH()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_month)  Return the month from the date passed |
| 31 | [**MONTHNAME()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_monthname)  Returns the name of the month |
| 32 | [**NOW()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_now)  Returns the current date and time |
| 33 | [**PERIOD\_ADD()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_period-add)  Adds a period to a year-month |
| 34 | [**PERIOD\_DIFF()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_period-diff)  Returns the number of months between periods |
| 35 | [**QUARTER()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_quarter)  Returns the quarter from a date argument |
| 36 | [**SEC\_TO\_TIME()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_sec-to-time)  Converts seconds to 'HH:MM:SS' format |
| 37 | [**SECOND()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_second)  Returns the second (0-59) |
| 38 | [**STR\_TO\_DATE()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_str-to-date)  Converts a string to a date |
| 39 | [**SUBDATE()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_subdate)  When invoked with three arguments a synonym for DATE\_SUB() |
| 40 | [**SUBTIME()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_subtime)  Subtracts times |
| 41 | [**SYSDATE()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_sysdate)  Returns the time at which the function executes |
| 42 | [**TIME\_FORMAT()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_time-format)  Formats as time |
| 43 | [**TIME\_TO\_SEC()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_time-to-sec)  Returns the argument converted to seconds |
| 44 | [**TIME()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_time)  Extracts the time portion of the expression passed |
| 45 | [**TIMEDIFF()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_timediff)  Subtracts time |
| 46 | [**TIMESTAMP()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_timestamp)  With a single argument this function returns the date or datetime expression. With two arguments, the sum of the arguments |
| 47 | [**TIMESTAMPADD()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_timestampadd)  Adds an interval to a datetime expression |
| 48 | [**TIMESTAMPDIFF()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_timestampdiff)  Subtracts an interval from a datetime expression |
| 49 | [**TO\_DAYS()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_to-days)  Returns the date argument converted to days |
| 50 | [**UNIX\_TIMESTAMP()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_unix-timestamp)  Returns a UNIX timestamp |
| 51 | [**UTC\_DATE()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_utc-date)  Returns the current UTC date |
| 52 | [**UTC\_TIME()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_utc-time)  Returns the current UTC time |
| 53 | [**UTC\_TIMESTAMP()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_utc-timestamp)  Returns the current UTC date and time |
| 54 | [**WEEK()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_week)  Returns the week number |
| 55 | [**WEEKDAY()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_weekday)  Returns the weekday index |
| 56 | [**WEEKOFYEAR()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_weekofyear)  Returns the calendar week of the date (1-53) |
| 57 | [**YEAR()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_year)  Returns the year |
| 58 | [**YEARWEEK()**](https://www.tutorialspoint.com/sql/sql-date-functions.htm#function_yearweek)  Returns the year and week |

ADDDATE(date,INTERVAL expr unit), ADDDATE(expr,days)

When invoked with the INTERVAL form of the second argument, ADDDATE() is a synonym for DATE\_ADD(). The related function SUBDATE() is a synonym for DATE\_SUB(). For information on the INTERVAL unit argument, see the discussion for DATE\_ADD().

mysql> SELECT DATE\_ADD('1998-01-02', INTERVAL 31 DAY);

+---------------------------------------------------------+

| DATE\_ADD('1998-01-02', INTERVAL 31 DAY) |

+---------------------------------------------------------+

| 1998-02-02 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

mysql> SELECT ADDDATE('1998-01-02', INTERVAL 31 DAY);

+---------------------------------------------------------+

| ADDDATE('1998-01-02', INTERVAL 31 DAY) |

+---------------------------------------------------------+

| 1998-02-02 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

When invoked with the days form of the second argument, MySQL treats it as an integer number of days to be added to expr.

mysql> SELECT ADDDATE('1998-01-02', 31);

+---------------------------------------------------------+

| DATE\_ADD('1998-01-02', INTERVAL 31 DAY) |

+---------------------------------------------------------+

| 1998-02-02 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

ADDTIME(expr1,expr2)

ADDTIME() adds expr2 to expr1 and returns the result. The expr1 is a time or datetime expression, while the expr2 is a time expression.

mysql> SELECT ADDTIME('1997-12-31 23:59:59.999999','1 1:1:1.000002');

+---------------------------------------------------------+

| DATE\_ADD('1997-12-31 23:59:59.999999','1 1:1:1.000002') |

+---------------------------------------------------------+

| 1998-01-02 01:01:01.000001 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

CONVERT\_TZ(dt,from\_tz,to\_tz)

This converts a datetime value dt from the time zone given by from\_tz to the time zone given by to\_tz and returns the resulting value. This function returns NULL if the arguments are invalid.

mysql> SELECT CONVERT\_TZ('2004-01-01 12:00:00','GMT','MET');

+---------------------------------------------------------+

| CONVERT\_TZ('2004-01-01 12:00:00','GMT','MET') |

+---------------------------------------------------------+

| 2004-01-01 13:00:00 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

mysql> SELECT CONVERT\_TZ('2004-01-01 12:00:00','+00:00','+10:00');

+---------------------------------------------------------+

| CONVERT\_TZ('2004-01-01 12:00:00','+00:00','+10:00') |

+---------------------------------------------------------+

| 2004-01-01 22:00:00 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

CURDATE()

Returns the current date as a value in 'YYYY-MM-DD' or YYYYMMDD format, depending on whether the function is used in a string or in a numeric context.

mysql> SELECT CURDATE();

+---------------------------------------------------------+

| CURDATE() |

+---------------------------------------------------------+

| 1997-12-15 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

mysql> SELECT CURDATE() + 0;

+---------------------------------------------------------+

| CURDATE() + 0 |

+---------------------------------------------------------+

| 19971215 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

CURRENT\_DATE and CURRENT\_DATE()

CURRENT\_DATE and CURRENT\_DATE() are synonyms for CURDATE()

CURTIME()

Returns the current time as a value in 'HH:MM:SS' or HHMMSS format, depending on whether the function is used in a string or in a numeric context. The value is expressed in the current time zone.

mysql> SELECT CURTIME();

+---------------------------------------------------------+

| CURTIME() |

+---------------------------------------------------------+

| 23:50:26 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

mysql> SELECT CURTIME() + 0;

+---------------------------------------------------------+

| CURTIME() + 0 |

+---------------------------------------------------------+

| 235026 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

CURRENT\_TIME and CURRENT\_TIME()

CURRENT\_TIME and CURRENT\_TIME() are synonyms for CURTIME().

CURRENT\_TIMESTAMP and CURRENT\_TIMESTAMP()

CURRENT\_TIMESTAMP and CURRENT\_TIMESTAMP() are synonyms for NOW().

DATE(expr)

Extracts the date part of the date or datetime expression expr.

mysql> SELECT DATE('2003-12-31 01:02:03');

+---------------------------------------------------------+

| DATE('2003-12-31 01:02:03') |

+---------------------------------------------------------+

| 2003-12-31 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

DATEDIFF(expr1,expr2)

DATEDIFF() returns expr1 . expr2 expressed as a value in days from one date to the other. Both expr1 and expr2 are date or date-and-time expressions. Only the date parts of the values are used in the calculation.

mysql> SELECT DATEDIFF('1997-12-31 23:59:59','1997-12-30');

+---------------------------------------------------------+

| DATEDIFF('1997-12-31 23:59:59','1997-12-30') |

+---------------------------------------------------------+

| 1 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

DATE\_ADD(date,INTERVAL expr unit), DATE\_SUB(date,INTERVAL expr unit)

These functions perform date arithmetic. The **date** is a DATETIME or DATE value specifying the starting date. The **expr** is an expression specifying the interval value to be added or subtracted from the starting date. The expr is a string; it may start with a '-' for negative intervals.

A **unit** is a keyword indicating the units in which the expression should be interpreted.

The **INTERVAL** keyword and the unit specifier are not case sensitive.

The following table shows the expected form of the expr argument for each unit value.

|  |  |
| --- | --- |
| **unit Value** | **Expected exprFormat** |
| MICROSECOND | MICROSECONDS |
| SECOND | SECONDS |
| MINUTE | MINUTES |
| HOUR | HOURS |
| DAY | DAYS |
| WEEK | WEEKS |
| MONTH | MONTHS |
| QUARTER | QUARTERS |
| YEAR | YEARS |
| SECOND\_MICROSECOND | 'SECONDS.MICROSECONDS' |
| MINUTE\_MICROSECOND | 'MINUTES.MICROSECONDS' |
| MINUTE\_SECOND | 'MINUTES:SECONDS' |
| HOUR\_MICROSECOND | 'HOURS.MICROSECONDS' |
| HOUR\_SECOND | 'HOURS:MINUTES:SECONDS' |
| HOUR\_MINUTE | 'HOURS:MINUTES' |
| DAY\_MICROSECOND | 'DAYS.MICROSECONDS' |
| DAY\_SECOND | 'DAYS HOURS:MINUTES:SECONDS' |
| DAY\_MINUTE | 'DAYS HOURS:MINUTES' |
| DAY\_HOUR | 'DAYS HOURS' |
| YEAR\_MONTH | 'YEARS-MONTHS' |

The values **QUARTER** and **WEEK** are available from the MySQL 5.0.0. version.

mysql> SELECT DATE\_ADD('1997-12-31 23:59:59',

-> INTERVAL '1:1' MINUTE\_SECOND);

+---------------------------------------------------------+

| DATE\_ADD('1997-12-31 23:59:59', INTERVAL... |

+---------------------------------------------------------+

| 1998-01-01 00:01:00 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

mysql> SELECT DATE\_ADD('1999-01-01', INTERVAL 1 HOUR);

+---------------------------------------------------------+

| DATE\_ADD('1999-01-01', INTERVAL 1 HOUR) |

+---------------------------------------------------------+

| 1999-01-01 01:00:00 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

DATE\_FORMAT(date,format)

This command formats the date value as per the format string.

The following specifiers may be used in the format string. The '%' character is required before the format specifier characters.

|  |  |
| --- | --- |
| **Sr.No.** | **Specifier & Description** |
| 1 | **%a**  Abbreviated weekday name (Sun..Sat) |
| 2 | **%b**  Abbreviated month name (Jan..Dec) |
| 3 | **%c**  Month, numeric (0..12) |
| 4 | **%D**  Day of the month with English suffix (0th, 1st, 2nd, 3rd, .) |
| 5 | **%d**  Day of the month, numeric (00..31) |
| 6 | **%e**  Day of the month, numeric (0..31) |
| 7 | **%f**  Microseconds (000000..999999) |
| 8 | **%H**  Hour (00..23) |
| 9 | **%h**  Hour (01..12) |
| 10 | **%I**  Hour (01..12) |
| 11 | **%i**  Minutes, numeric (00..59) |
| 12 | **%j**  Day of year (001..366) |
| 13 | **%k**  Hour (0..23) |
| 14 | **%l**  Hour (1..12) |
| 15 | **%M**  Month name (January..December) |
| 16 | **%m**  Month, numeric (00..12) |
| 17 | **%p**  AM or PM |
| 18 | **%r**  Time, 12-hour (hh:mm:ss followed by AM or PM) |
| 19 | **%S**  Seconds (00..59) |
| 20 | **%s**  Seconds (00..59) |
| 21 | **%T**  Time, 24-hour (hh:mm:ss) |
| 22 | **%U**  Week (00..53), where Sunday is the first day of the week |
| 23 | **%u**  Week (00..53), where Monday is the first day of the week |
| 24 | **%V**  Week (01..53), where Sunday is the first day of the week; used with %X |
| 25 | **%v**  Week (01..53), where Monday is the first day of the week; used with %x |
| 26 | **%W**  Weekday name (Sunday..Saturday) |
| 27 | **%w**  Day of the week (0=Sunday..6=Saturday) |
| 28 | **%X**  Year for the week where Sunday is the first day of the week, numeric, four digits; used with %V |
| 29 | **%x**  Year for the week, where Monday is the first day of the week, numeric, four digits; used with %v |
| 30 | **%Y**  Year, numeric, four digits |
| 31 | **%y**  Year, numeric (two digits) |
| 32 | **%%**  A literal .%. character |
| 33 | **%x**  x, for any.x. not listed above |

mysql> SELECT DATE\_FORMAT('1997-10-04 22:23:00', '%W %M %Y');

+---------------------------------------------------------+

| DATE\_FORMAT('1997-10-04 22:23:00', '%W %M %Y') |

+---------------------------------------------------------+

| Saturday October 1997 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

mysql> SELECT DATE\_FORMAT('1997-10-04 22:23:00'

-> '%H %k %I %r %T %S %w');

+---------------------------------------------------------+

| DATE\_FORMAT('1997-10-04 22:23:00....... |

+---------------------------------------------------------+

| 22 22 10 10:23:00 PM 22:23:00 00 6 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

DATE\_SUB(date,INTERVAL expr unit)

This is similar to the DATE\_ADD() function.

DAY(date)

The DAY() is a synonym for the DAYOFMONTH() function.

DAYNAME(date)

Returns the name of the weekday for date.

mysql> SELECT DAYNAME('1998-02-05');

+---------------------------------------------------------+

| DAYNAME('1998-02-05') |

+---------------------------------------------------------+

| Thursday |

+---------------------------------------------------------+

1 row in set (0.00 sec)

DAYOFMONTH(date)

Returns the day of the month for date, in the range 0 to 31.

mysql> SELECT DAYOFMONTH('1998-02-03');

+---------------------------------------------------------+

| DAYOFMONTH('1998-02-03') |

+---------------------------------------------------------+

| 3 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

DAYOFWEEK(date)

Returns the weekday index for date (1 = Sunday, 2 = Monday, ., 7 = Saturday). These index values correspond to the ODBC standard.

mysql> SELECT DAYOFWEEK('1998-02-03');

+---------------------------------------------------------+

|DAYOFWEEK('1998-02-03') |

+---------------------------------------------------------+

| 3 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

DAYOFYEAR(date)

Returns the day of the year for date, in the range 1 to 366.

mysql> SELECT DAYOFYEAR('1998-02-03');

+---------------------------------------------------------+

| DAYOFYEAR('1998-02-03') |

+---------------------------------------------------------+

| 34 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

EXTRACT(unit FROM date)

The EXTRACT() function uses the same kinds of unit specifiers as DATE\_ADD() or DATE\_SUB(), but extracts parts from the date rather than performing date arithmetic.

mysql> SELECT EXTRACT(YEAR FROM '1999-07-02');

+---------------------------------------------------------+

| EXTRACT(YEAR FROM '1999-07-02') |

+---------------------------------------------------------+

| 1999 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

mysql> SELECT EXTRACT(YEAR\_MONTH FROM '1999-07-02 01:02:03');

+---------------------------------------------------------+

| EXTRACT(YEAR\_MONTH FROM '1999-07-02 01:02:03') |

+---------------------------------------------------------+

| 199907 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

FROM\_DAYS(N)

Given a day number N, returns a DATE value.

mysql> SELECT FROM\_DAYS(729669);

+---------------------------------------------------------+

| FROM\_DAYS(729669) |

+---------------------------------------------------------+

| 1997-10-07 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

**Note** − Use FROM\_DAYS() with caution on old dates. It is not intended for use with values that precede the advent of the Gregorian calendar (1582).

FROM\_UNIXTIME(unix\_timestamp)

FROM\_UNIXTIME(unix\_timestamp,format)

Returns a representation of the **unix\_timestamp** argument as a value in 'YYYY-MM-DD HH:MM:SS or YYYYMMDDHHMMSS format, depending on whether the function is used in a string or in a numeric context. The value is expressed in the current time zone. The unix\_timestamp argument is an internal timestamp values, which are produced by the **UNIX\_TIMESTAMP()**function.

If the format is given, the result is formatted according to the format string, which is used in the same way as is listed in the entry for the **DATE\_FORMAT()** function.

mysql> SELECT FROM\_UNIXTIME(875996580);

+---------------------------------------------------------+

| FROM\_UNIXTIME(875996580) |

+---------------------------------------------------------+

| 1997-10-04 22:23:00 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

HOUR(time)

Returns the hour for time. The range of the return value is 0 to 23 for time-of-day values. However, the range of TIME values actually is much larger, so HOUR can return values greater than 23.

mysql> SELECT HOUR('10:05:03');

+---------------------------------------------------------+

| HOUR('10:05:03') |

+---------------------------------------------------------+

| 10 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

LAST\_DAY(date)

Takes a date or datetime value and returns the corresponding value for the last day of the month. Returns NULL if the argument is invalid.

mysql> SELECT LAST\_DAY('2003-02-05');

+---------------------------------------------------------+

| LAST\_DAY('2003-02-05') |

+---------------------------------------------------------+

| 2003-02-28 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

LOCALTIME and LOCALTIME()

LOCALTIME and LOCALTIME() are synonyms for NOW().

LOCALTIMESTAMP and LOCALTIMESTAMP()

LOCALTIMESTAMP and LOCALTIMESTAMP() are synonyms for NOW().

MAKEDATE(year,dayofyear)

Returns a date, given year and day-of-year values. The dayofyear value must be greater than 0 or the result will be NULL.

mysql> SELECT MAKEDATE(2001,31), MAKEDATE(2001,32);

+---------------------------------------------------------+

| MAKEDATE(2001,31), MAKEDATE(2001,32) |

+---------------------------------------------------------+

| '2001-01-31', '2001-02-01' |

+---------------------------------------------------------+

1 row in set (0.00 sec)

MAKETIME(hour,minute,second)

Returns a time value calculated from the hour, minute and second arguments.

mysql> SELECT MAKETIME(12,15,30);

+---------------------------------------------------------+

| MAKETIME(12,15,30) |

+---------------------------------------------------------+

| '12:15:30' |

+---------------------------------------------------------+

1 row in set (0.00 sec)

MICROSECOND(expr)

Returns the microseconds from the time or datetime expression (expr) as a number in the range from 0 to 999999.

mysql> SELECT MICROSECOND('12:00:00.123456');

+---------------------------------------------------------+

| MICROSECOND('12:00:00.123456') |

+---------------------------------------------------------+

| 123456 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

MINUTE(time)

Returns the minute for time, in the range 0 to 59.

mysql> SELECT MINUTE('98-02-03 10:05:03');

+---------------------------------------------------------+

| MINUTE('98-02-03 10:05:03') |

+---------------------------------------------------------+

| 5 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

MONTH(date)

Returns the month for date, in the range 0 to 12.

mysql> SELECT MONTH('1998-02-03')

+---------------------------------------------------------+

| MONTH('1998-02-03') |

+---------------------------------------------------------+

| 2 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

MONTHNAME(date)

Returns the full name of the month for a date.

mysql> SELECT MONTHNAME('1998-02-05');

+---------------------------------------------------------+

| MONTHNAME('1998-02-05') |

+---------------------------------------------------------+

| February |

+---------------------------------------------------------+

1 row in set (0.00 sec)

NOW()

Returns the current date and time as a value in 'YYYY-MM-DD HH:MM:SS' or YYYYMMDDHHMMSS format, depending on whether the function is used in a string or numeric context. This value is expressed in the current time zone.

mysql> SELECT NOW();

+---------------------------------------------------------+

| NOW() |

+---------------------------------------------------------+

| 1997-12-15 23:50:26 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

PERIOD\_ADD(P,N)

Adds N months to a period P (in the format YYMM or YYYYMM). Returns a value in the format YYYYMM. Note that the period argument P is not a date value.

mysql> SELECT PERIOD\_ADD(9801,2);

+---------------------------------------------------------+

| PERIOD\_ADD(9801,2) |

+---------------------------------------------------------+

| 199803 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

PERIOD\_DIFF(P1,P2)

Returns the number of months between periods P1 and P2. These periods P1 and P2 should be in the format YYMM or YYYYMM. Note that the period arguments P1 and P2 are not date values.

mysql> SELECT PERIOD\_DIFF(9802,199703);

+---------------------------------------------------------+

| PERIOD\_DIFF(9802,199703) |

+---------------------------------------------------------+

| 11 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

QUARTER(date)

Returns the quarter of the year for date, in the range 1 to 4.

mysql> SELECT QUARTER('98-04-01');

+---------------------------------------------------------+

| QUARTER('98-04-01') |

+---------------------------------------------------------+

| 2 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

SECOND(time)

Returns the second for time, in the range 0 to 59.

mysql> SELECT SECOND('10:05:03');

+---------------------------------------------------------+

| SECOND('10:05:03') |

+---------------------------------------------------------+

| 3 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

SEC\_TO\_TIME(seconds)

Returns the seconds argument, converted to hours, minutes and seconds, as a value in 'HH:MM:SS' or HHMMSS format, depending on whether the function is used in a string or numeric context.

mysql> SELECT SEC\_TO\_TIME(2378);

+---------------------------------------------------------+

| SEC\_TO\_TIME(2378) |

+---------------------------------------------------------+

| 00:39:38 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

STR\_TO\_DATE(str,format)

This is the inverse of the DATE\_FORMAT() function. It takes a string str and a format string format. The STR\_TO\_DATE() function returns a DATETIME value if the format string contains both date and time parts. Else, it returns a DATE or TIME value if the string contains only date or time parts.

mysql> SELECT STR\_TO\_DATE('04/31/2004', '%m/%d/%Y');

+---------------------------------------------------------+

| STR\_TO\_DATE('04/31/2004', '%m/%d/%Y') |

+---------------------------------------------------------+

| 2004-04-31 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

SUBDATE(date,INTERVAL expr unit) and SUBDATE(expr,days)

When invoked with the INTERVAL form of the second argument, SUBDATE() is a synonym for DATE\_SUB(). For information on the INTERVAL unit argument, see the discussion for DATE\_ADD().

mysql> SELECT DATE\_SUB('1998-01-02', INTERVAL 31 DAY);

+---------------------------------------------------------+

| DATE\_SUB('1998-01-02', INTERVAL 31 DAY) |

+---------------------------------------------------------+

| 1997-12-02 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

mysql> SELECT SUBDATE('1998-01-02', INTERVAL 31 DAY);

+---------------------------------------------------------+

| SUBDATE('1998-01-02', INTERVAL 31 DAY) |

+---------------------------------------------------------+

| 1997-12-02 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

SUBTIME(expr1,expr2)

The SUBTIME() function returns expr1 . expr2 expressed as a value in the same format as expr1. The expr1 value is a time or a datetime expression, while the expr2 value is a time expression.

mysql> SELECT SUBTIME('1997-12-31 23:59:59.999999',

-> '1 1:1:1.000002');

+---------------------------------------------------------+

| SUBTIME('1997-12-31 23:59:59.999999'... |

+---------------------------------------------------------+

| 1997-12-30 22:58:58.999997 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

SYSDATE()

Returns the current date and time as a value in 'YYYY-MM-DD HH:MM:SS' or YYYYMMDDHHMMSS format, depending on whether the function is used in a string or in a numeric context.

mysql> SELECT SYSDATE();

+---------------------------------------------------------+

| SYSDATE() |

+---------------------------------------------------------+

| 2006-04-12 13:47:44 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

TIME(expr)

Extracts the time part of the time or datetime expression **expr** and returns it as a string.

mysql> SELECT TIME('2003-12-31 01:02:03');

+---------------------------------------------------------+

| TIME('2003-12-31 01:02:03') |

+---------------------------------------------------------+

| 01:02:03 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

TIMEDIFF(expr1,expr2)

The TIMEDIFF() function returns expr1 . expr2 expressed as a time value. These expr1 and expr2 values are time or date-and-time expressions, but both must be of the same type.

mysql> SELECT TIMEDIFF('1997-12-31 23:59:59.000001',

-> '1997-12-30 01:01:01.000002');

+---------------------------------------------------------+

| TIMEDIFF('1997-12-31 23:59:59.000001'..... |

+---------------------------------------------------------+

| 46:58:57.999999 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

TIMESTAMP(expr), TIMESTAMP(expr1,expr2)

With a single argument, this function returns the date or datetime expression expr as a datetime value. With two arguments, it adds the time expression expr2 to the date or datetime expression **expr1** and returns the result as a datetime value.

mysql> SELECT TIMESTAMP('2003-12-31');

+---------------------------------------------------------+

| TIMESTAMP('2003-12-31') |

+---------------------------------------------------------+

| 2003-12-31 00:00:00 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

TIMESTAMPADD(unit,interval,datetime\_expr)

This function adds the integer expression interval to the date or datetime expression **datetime\_expr**. The unit for interval is given by the unit argument, which should be one of the following values −

* FRAC\_SECOND
* SECOND, MINUTE
* HOUR, DAY
* WEEK
* MONTH
* QUARTER or
* YEAR

The unit value may be specified using one of the keywords as shown or with a prefix of SQL\_TSI\_.

For example, DAY and SQL\_TSI\_DAY both are legal.

mysql> SELECT TIMESTAMPADD(MINUTE,1,'2003-01-02');

+---------------------------------------------------------+

| TIMESTAMPADD(MINUTE,1,'2003-01-02') |

+---------------------------------------------------------+

| 2003-01-02 00:01:00 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

TIMESTAMPDIFF(unit,datetime\_expr1,datetime\_expr2)

Returns the integer difference between the date or datetime expressions datetime\_expr1 and datetime\_expr2. The unit for the result is given by the unit argument. The legal values for the unit are the same as those listed in the description of the TIMESTAMPADD() function.

mysql> SELECT TIMESTAMPDIFF(MONTH,'2003-02-01','2003-05-01');

+---------------------------------------------------------+

| TIMESTAMPDIFF(MONTH,'2003-02-01','2003-05-01') |

+---------------------------------------------------------+

| 3 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

TIME\_FORMAT(time,format)

This function is used like the DATE\_FORMAT() function, but the format string may contain format specifiers only for hours, minutes and seconds.

If the time value contains an hour part that is greater than 23, the %**H** and %**k** hour format specifiers produce a value larger than the usual range of 0 to 23. The other hour format specifiers produce the hour value modulo 12.

mysql> SELECT TIME\_FORMAT('100:00:00', '%H %k %h %I %l');

+---------------------------------------------------------+

| TIME\_FORMAT('100:00:00', '%H %k %h %I %l') |

+---------------------------------------------------------+

| 100 100 04 04 4 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

TIME\_TO\_SEC(time)

Returns the time argument converted to seconds.

mysql> SELECT TIME\_TO\_SEC('22:23:00');

+---------------------------------------------------------+

| TIME\_TO\_SEC('22:23:00') |

+---------------------------------------------------------+

| 80580 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

TO\_DAYS(date)

Given a date, returns a day number (the number of days since year 0).

mysql> SELECT TO\_DAYS(950501);

+---------------------------------------------------------+

| TO\_DAYS(950501) |

+---------------------------------------------------------+

| 728779 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

UNIX\_TIMESTAMP(), UNIX\_TIMESTAMP(date)

If called with no argument, this function returns a Unix timestamp (seconds since '1970-01-01 00:00:00' UTC) as an unsigned integer. If UNIX\_TIMESTAMP() is called with a date argument, it returns the value of the argument as seconds since '1970-01-01 00:00:00' UTC. date may be a DATE string, a DATETIME string, a TIMESTAMP, or a number in the format YYMMDD or YYYYMMDD.

mysql> SELECT UNIX\_TIMESTAMP();

+---------------------------------------------------------+

| UNIX\_TIMESTAMP() |

+---------------------------------------------------------+

| 882226357 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

mysql> SELECT UNIX\_TIMESTAMP('1997-10-04 22:23:00');

+---------------------------------------------------------+

| UNIX\_TIMESTAMP('1997-10-04 22:23:00') |

+---------------------------------------------------------+

| 875996580 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

UTC\_DATE, UTC\_DATE()

Returns the current UTC date as a value in 'YYYY-MM-DD' or YYYYMMDD format, depending on whether the function is used in a string or numeric context.

mysql> SELECT UTC\_DATE(), UTC\_DATE() + 0;

+---------------------------------------------------------+

| UTC\_DATE(), UTC\_DATE() + 0 |

+---------------------------------------------------------+

| 2003-08-14, 20030814 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

UTC\_TIME, UTC\_TIME()

Returns the current UTC time as a value in 'HH:MM:SS' or HHMMSS format, depending on whether the function is used in a string or numeric context.

mysql> SELECT UTC\_TIME(), UTC\_TIME() + 0;

+---------------------------------------------------------+

| UTC\_TIME(), UTC\_TIME() + 0 |

+---------------------------------------------------------+

| 18:07:53, 180753 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

UTC\_TIMESTAMP, UTC\_TIMESTAMP()

Returns the current UTC date and time as a value in 'YYYY-MM-DD HH:MM:SS' or in a YYYYMMDDHHMMSS format, depending on whether the function is used in a string or in a numeric context.

mysql> SELECT UTC\_TIMESTAMP(), UTC\_TIMESTAMP() + 0;

+---------------------------------------------------------+

| UTC\_TIMESTAMP(), UTC\_TIMESTAMP() + 0 |

+---------------------------------------------------------+

| 2003-08-14 18:08:04, 20030814180804 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

WEEK(date[,mode])

This function returns the week number for date. The two-argument form of WEEK() allows you to specify whether the week starts on a Sunday or a Monday and whether the return value should be in the range from 0 to 53 or from 1 to 53. If the mode argument is omitted, the value of the default\_week\_format system variable is used

|  |  |  |  |
| --- | --- | --- | --- |
| **Mode** | **First Day of week** | **Range** | **Week 1 is the first week.** |
| 0 | Sunday | 0-53 | with a Sunday in this year |
| 1 | Monday | 0-53 | with more than 3 days this year |
| 2 | Sunday | 1-53 | with a Sunday in this year |
| 3 | Monday | 1-53 | with more than 3 days this year |
| 4 | Sunday | 0-53 | with more than 3 days this year |
| 5 | Monday | 0-53 | with a Monday in this year |
| 6 | Sunday | 1-53 | with more than 3 days this year |
| 7 | Monday | 1-53 | with a Monday in this year |

mysql> SELECT WEEK('1998-02-20');

+---------------------------------------------------------+

| WEEK('1998-02-20') |

+---------------------------------------------------------+

| 7 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

WEEKDAY(date)

Returns the weekday index for date (0 = Monday, 1 = Tuesday, . 6 = Sunday).

mysql> SELECT WEEKDAY('1998-02-03 22:23:00');

+---------------------------------------------------------+

| WEEKDAY('1998-02-03 22:23:00') |

+---------------------------------------------------------+

| 1 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

WEEKOFYEAR(date)

Returns the calendar week of the date as a number in the range from 1 to 53. WEEKOFYEAR() is a compatibility function that is equivalent to WEEK(date,3).

mysql> SELECT WEEKOFYEAR('1998-02-20');

+---------------------------------------------------------+

| WEEKOFYEAR('1998-02-20') |

+---------------------------------------------------------+

| 8 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

YEAR(date)

Returns the year for date, in the range 1000 to 9999, or 0 for the .zero. date.

mysql> SELECT YEAR('98-02-03');

+---------------------------------------------------------+

| YEAR('98-02-03') |

+---------------------------------------------------------+

| 1998 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

YEARWEEK(date), YEARWEEK(date,mode)

Returns the year and the week for a date. The mode argument works exactly like the mode argument to the WEEK() function. The year in the result may be different from the year in the date argument for the first and the last week of the year.

mysql> SELECT YEARWEEK('1987-01-01');

+---------------------------------------------------------+

| YEAR('98-02-03')YEARWEEK('1987-01-01') |

+---------------------------------------------------------+

| 198653 |

+---------------------------------------------------------+

1 row in set (0.00 sec)

**Note** − The week number is different from what the WEEK() function would return (0) for optional arguments 0 or 1, as WEEK() then returns the week in the context of the given year.