MySQL is a very popular open-source relational database management system (RDBMS).

**Some of The Most Important SQL Commands**

SELECT - extracts data from a database

UPDATE - updates data in a database

DELETE - deletes data from a database

INSERT INTO - inserts new data into a database

CREATE DATABASE - creates a new database

ALTER DATABASE - modifies a database

CREATE TABLE - creates a new table

ALTER TABLE - modifies a table

DROP TABLE - deletes a table

CREATE INDEX - creates an index (search key)

DROP INDEX - deletes an index

**SELECT statement:**

The SELECT statement is used to select data from a database.

The data returned is stored in a result table, called the result-set.

***SELECT Syntax***

SELECT column1, column2, ... FROM table\_name;

SELECT DISTINCT Country FROM Customers; -- selects only the DISTINCT values from the "Country" column in the "Customers" table

SELECT COUNT(DISTINCT Country) FROM Customers; -- counts and returns the number of different (distinct) countries in the "Customers" table

***WHERE clause***

SELECT column1, column2, ... FROM table\_name WHERE condition;

***Operators in The WHERE Clause***

The following operators can be used in the WHERE clause:

= Equal

> Greater than

< Less than

>= Greater than or equal

<= Less than or equal

<> Not equal. Note: In some versions of SQL this operator may be written as !=

BETWEEN Between a certain range : BETWEEN 20 AND 30

LIKE Search for a pattern : LIKE '%s'

IN To specify multiple possible values for a column : IN ('Paris','London');

***ORDER BY keyword***

ORDER BY keyword is used to sort the result-set in ascending or descending order. By default, it is ascending order, To sort the records in descending order, use the DESC keyword.

**The MySQL INSERT INTO Statement**

The INSERT INTO statement is used to insert new records in a table.

***INSERT INTO Syntax***

It is possible to write the INSERT INTO statement in two ways:

1. Specify both the column names and the values to be inserted:

INSERT INTO table\_name (column1, column2, column3, ...) VALUES (value1, value2, value3, ...);

\* Column1, column2...... column names are optional if all the values are inserted.

\* If only few columns are to be inserted then specify the particular column names.

**The MySQL UPDATE Statement**

The UPDATE statement is used to modify the existing records in a table.

***UPDATE Syntax***

UPDATE table\_name SET column1 = value1, column2 = value2, ... WHERE condition;

Be careful when updating records. If you omit the WHERE clause, ALL records will be updated!

**The MySQL DELETE Statement**

The DELETE statement is used to delete existing records in a table.

***DELETE Syntax***

DELETE FROM table\_name WHERE condition;

DELETE FROM table\_name; -- delete all rows in a table without deleting the table.

**The MySQL LIMIT Clause**

The LIMIT clause is used to specify the number of records to return.

***LIMIT Syntax***

SELECT column\_name(s) FROM table\_name WHERE condition LIMIT number;

**MySQL MIN() and MAX() Functions**

The MIN() function returns the smallest value of the selected column.

The MAX() function returns the largest value of the selected column.

***MIN() Syntax***

SELECT MIN(column\_name) FROM table\_name WHERE condition;

***MAX() Syntax***

SELECT MAX(column\_name) FROM table\_name WHERE condition;

**MySQL COUNT(), AVG() and SUM() Functions**

The COUNT() function returns the number of rows that matches a specified criterion.

***COUNT() Syntax***

SELECT COUNT(column\_name) FROM table\_name WHERE condition;

The AVG() function returns the average value of a numeric column.

***AVG() Syntax***

SELECT AVG(column\_name) FROM table\_name WHERE condition;

The SUM() function returns the total sum of a numeric column.

***SUM() Syntax***

SELECT SUM(column\_name) FROM table\_name WHERE condition;

**The MySQL LIKE Operator**

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!

***LIKE Syntax***

SELECT column1, column2, ... FROM table\_name WHERE columnN LIKE pattern;

LIKE Operator Description

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"

**The MySQL IN Operator**

IN Operator Syntax:

SELECT column\_name(s) FROM table\_name WHERE column\_name IN (value1, value2, ...);

or:

SELECT column\_name(s) FROM table\_name WHERE column\_name IN (SELECT STATEMENT);

**The MySQL BETWEEN Operator**

The BETWEEN operator is inclusive: begin and end values are included.

***Syntax***:

SELECT column\_name(s) FROM table\_name WHERE column\_name BETWEEN value1 AND value2;

***NOT BETWEEN syntax:***

SELECT column\_name(s) FROM table\_name WHERE column\_name NOT BETWEEN value1 AND value2;

**MySQL Aliases**

Alias Column Syntax

SELECT column\_name AS alias\_name FROM table\_name;

\* alias\_name is used in complete query. column\_name can't be used when alias\_name is used.

MySQL Joins

**MySQL Joining Tables**

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

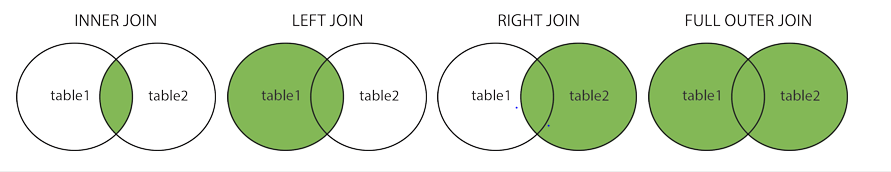
Supported Types of Joins in MySQL

INNER JOIN: Returns records that have matching values in both tables

LEFT JOIN: Returns all records from the left table, and the matched records from the right table

RIGHT JOIN: Returns all records from the right table, and the matched records from the left table

CROSS JOIN: Returns all records from both tables



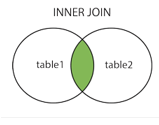
\* LEFT JOIN & RIGHT JOINS & CROSS ( FULL OUTER ) JOINS are called as OUTER JOINS.

\* The INNER JOIN keyword selects records that have matching values in both tables. ( Default JOIN keyword is INNER JOIN )

( Both JOIN & INNER JOIN can be used )

**MySQL INNER JOIN Keyword**

The INNER JOIN keyword selects records that have matching values in both tables.



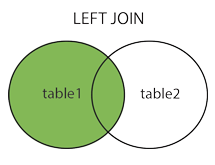
***INNER JOIN Syntax***

SELECT column\_name(s) FROM table1 INNER JOIN table2 ON table1.column\_name = table2.column\_name;

The INNER JOIN keyword selects all rows from both tables as long as there is a match between the columns.

**MySQL LEFT JOIN Keyword**

The LEFT JOIN keyword returns all records from the left table (table1), and the matching records (if any) from the right table (table2)



***LEFT JOIN Syntax***

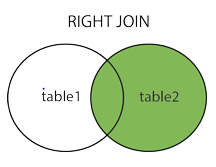
SELECT column\_name(s) FROM table1 LEFT JOIN table2 ON table1.column\_name = table2.column\_name;

The LEFT JOIN keyword returns all records from the left table (Customers), even if there are no matches in the right table (Orders).

**MySQL RIGHT JOIN Keyword**

The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records (if any) from the left table (table1).

MySQL RIGHT JOIN



***RIGHT JOIN Syntax***

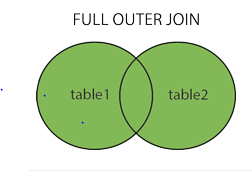
SELECT column\_name(s) FROM table1 RIGHT JOIN table2 ON table1.column\_name = table2.column\_name;

The RIGHT JOIN keyword returns all records from the right table (Employees), even if there are no matches in the left table (Orders).

**SQL CROSS JOIN Keyword**

The CROSS JOIN keyword returns all records from both tables (table1 and table2).

MySQL CROSS JOIN



***CROSS JOIN Syntax***

SELECT column\_name(s) FROM table1 CROSS JOIN table2;

The CROSS JOIN keyword returns all matching records from both tables whether the other table matches or not. So, if there are rows in "Customers" that do not have matches in "Orders", or if there are rows in "Orders" that do not have matches in "Customers", those rows will be listed as well.

If you add a WHERE clause (if table1 and table2 has a relationship), the CROSS JOIN will produce the same result as the INNER JOIN clause:

\* CROSS JOIN is applicable where different sizes and diff colours are to be combined.

**MySQL Self Join**

A self join is a regular join, but the table is joined with itself.

***Self Join Syntax***

SELECT column\_name(s) FROM table1 T1, table1 T2 WHERE condition;

T1 and T2 are different table aliases for the same table.

**The MySQL UNION Operator**

The UNION operator is used to combine the result-set of two or more SELECT statements.

Every SELECT statement within UNION must have the same number of columns.

The columns must also have similar data types

The columns in every SELECT statement must also be in the same order

***UNION Syntax***

SELECT column\_name(s) FROM table1

UNION

SELECT column\_name(s) FROM table2;

***UNION ALL Syntax***

The UNION operator selects only distinct values by default. To allow duplicate values, use UNION ALL:

SELECT column\_name(s) FROM table1

UNION ALL

SELECT column\_name(s) FROM table2;

**The MySQL GROUP BY Statement**

The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

The GROUP BY statement is often used with aggregate functions (COUNT(), MAX(), MIN(), SUM(), AVG()) to group the result-set by one or more columns.

***GROUP BY Syntax***

SELECT column\_name(s) FROM table\_name WHERE condition GROUP BY column\_name(s)

ORDER BY column\_name(s);

**The MySQL HAVING Clause**

The HAVING clause was added to SQL because the WHERE keyword cannot be used with aggregate functions.

***HAVING Syntax***

SELECT column\_name(s) FROM table\_name WHERE condition GROUP BY column\_name(s)

HAVING condition ORDER BY column\_name(s);

***The MySQL USING Clause***

The USING keyword is used in JOINS instead of ON statement

SELECT \* FROM order\_items AS oi

JOIN order\_item\_notes AS oin

ON oi.order\_id = oin.order\_id AND oi.product\_id = oin.product\_id

Can be written as

SELECT \* FROM order\_items AS oi

JOIN order\_item\_notes AS oin

USING (order\_id,product\_id)

\* USING keyword has to be used only if column names in two tables are same. If they differ USING keyword can't be used.

***USING Syntax***

SELECT column\_names(s) FROM table\_name JOIN USING (column name)

**The MySQL EXISTS Operator**

The EXISTS operator is used to test for the existence of any record in a subquery.

The EXISTS operator returns TRUE if the subquery returns one or more records.

***EXISTS******Syntax***

SELECT column\_name(s) FROM table\_name WHERE EXISTS (SELECT column\_name FROM table\_name WHERE condition);

**The MySQL ANY and ALL Operators**

The ANY and ALL operators allow you to perform a comparison between a single column value and a range of other values.

***ANY Operator***

The ANY Operator returns a boolean value as a result.

returns TRUE if ANY of the subquery values meet the condition.

ANY means that the condition will be true if the operation is true for any of the values in the range.

***ANY Syntax***

SELECT column\_name(s) FROM table\_name WHERE column\_name operator ANY (SELECT column\_name FROM table\_name WHERE condition);

Note: The operator must be a standard comparison operator (=, <>, !=, >, >=, <, or <=).

***ALL Operator***

The ALL Operator returns a boolean value as a result

returns TRUE if ALL of the subquery values meet the condition

is used with SELECT, WHERE and HAVING statements

ALL means that the condition will be true only if the operation is true for all values in the range.

***ALL Syntax With SELECT***

SELECT ALL column\_name(s) FROM table\_name WHERE condition;

**The MySQL INSERT INTO SELECT Statement**

The INSERT INTO SELECT statement copies data from one table and inserts it into another table.

The INSERT INTO SELECT statement requires that the data types in source and target tables matches.

Note: The existing records in the target table are unaffected.

***INSERT INTO SELECT Syntax***

Copy all columns from one table to another table:

INSERT INTO table2 SELECT \* FROM table1 WHERE condition;

Copy only some columns from one table into another table:

INSERT INTO table2 (column1, column2, column3, ...) SELECT column1, column2, column3, ...

FROM table1 WHERE condition;

**The MySQL CASE Statement**

The CASE statement goes through conditions and returns a value when the first condition is met (like an if-then-else statement). So, once a condition is true, it will stop reading and return the result. If no conditions are true, it returns the value in the ELSE clause.

If there is no ELSE part and no conditions are true, it returns NULL.

***CASE Syntax***

CASE

WHEN condition1 THEN result1

WHEN condition2 THEN result2

WHEN conditionN THEN resultN

ELSE result

END;

**MySQL IFNULL() Function**

The MySQL IFNULL() function lets you return an alternative value if an expression is NULL.

The example below returns 0 if the value is NULL:

SELECT ProductName, UnitPrice \* (UnitsInStock + IFNULL(UnitsOnOrder, 0)) FROM Products;

**MySQL COALESCE() Function**

we can use the COALESCE() function, like this:

SELECT ProductName, UnitPrice \* (UnitsInStock + COALESCE(UnitsOnOrder, 0)) FROM Products;

**MySQL Comments**

Comments are used to explain sections of SQL statements, or to prevent execution of SQL statements.

Single line comments start with --.

Any text between -- and the end of the line will be ignored (will not be executed).

Multi-line comments start with /\* and end with \*/.

Any text between /\* and \*/ will be ignored.

**MySQL Arithmetic Operators**

Operator Description

+ Add

- Subtract

\* Multiply

/ Divide

% Modulo

**MySQL Bitwise Operators**

Operator Description

& Bitwise AND

| Bitwise OR

^ Bitwise exclusive OR

**MySQL Comparison Operators**

Operator Description

= Equal to

> Greater than

< Less than

>= Greater than or equal to

<= Less than or equal to

<> Not equal to

**MySQL Compound Operators**

Operator Description

+= Add equals

-= Subtract equals

\*= Multiply equals

/= Divide equals

%= Modulo equals

&= Bitwise AND equals

^-= Bitwise exclusive equals

|\*= Bitwise OR equals

**MySQL Logical Operators**

Operator Description

ALL TRUE if all of the subquery values meet the condition

AND TRUE if all the conditions separated by AND is TRUE

ANY TRUE if any of the subquery values meet the condition

BETWEEN TRUE if the operand is within the range of comparisons

EXISTS TRUE if the subquery returns one or more records

IN TRUE if the operand is equal to one of a list of expressions

LIKE TRUE if the operand matches a pattern

NOT Displays a record if the condition(s) is NOT TRUE

OR TRUE if any of the conditions separated by OR is TRUE

SOME TRUE if any of the subquery values meet the condition

**NATURAL JOINS**

NATURAL JOIN is a join that performs the same task as an INNER or LEFT JOIN, in which the ON or USING clause refers to all columns that the tables to be joined have in common.

The MySQL NATURAL JOIN is structured in such a way that, columns with the same name of associate tables will appear once only.

\* Don't use ON clause in a NATURAL JOIN.

**CHAR vs VARCHAR**

\* CHAR(50) stores 50 bytes of memory even if the value is of 5 bytes. Whereas, VARCHAR(50) allows a value of maximum length 50 and only store the size of value assigned. If a value of 5 bytes is assigned, only 5 bytes is allocated remaining 45 bytes is not reserved.

\* VARCHAR is memory efficient than CHAR

**LAST\_INSERT\_ID()**

The LAST\_INSERT\_ID() function returns the AUTO\_INCREMENT id of the last row that has been inserted or updated in a table.

**The MySQL CREATE DATABASE Statement**

The CREATE DATABASE statement is used to create a new SQL database.

***Syntax***

CREATE DATABASE databasename;

\* Make sure you have admin privilege before creating any database. Once a database is created, you can check it in the list of databases with the following SQL command: SHOW DATABASES;

**The MySQL DROP DATABASE Statement**

The DROP DATABASE statement is used to drop an existing SQL database.

***Syntax***

DROP DATABASE databasename;

\* Be careful before dropping a database. Deleting a database will result in loss of complete information stored in the database!

**The MySQL CREATE TABLE Statement**

The CREATE TABLE statement is used to create a new table in a database.

***Syntax***

CREATE TABLE table\_name (

column1 datatype,

column2 datatype,

column3 datatype,

....

);

The column parameters specify the names of the columns of the table.

The datatype parameter specifies the type of data the column can hold (e.g. varchar, integer, date, etc.).

***Create Table Using Another Table***

A copy of an existing table can also be created using CREATE TABLE.

The new table gets the same column definitions. All columns or specific columns can be selected.

If you create a new table using an existing table, the new table will be filled with the existing values from the old table.

***Syntax***

CREATE TABLE new\_table\_name AS

SELECT column1, column2,...

FROM existing\_table\_name

WHERE ....;

**The MySQL DROP TABLE Statement**

The DROP TABLE statement is used to drop an existing table in a database.

***Syntax***

DROP TABLE table\_name;

**MySQL TRUNCATE TABLE**

The TRUNCATE TABLE statement is used to delete the data inside a table, but not the table itself.

***Syntax***

TRUNCATE TABLE table\_name;

**MySQL ALTER TABLE Statement**

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.

The ALTER TABLE statement is also used to add and drop various constraints on an existing table.

***ALTER TABLE - ADD Column***

To add a column in a table, use the following syntax:

ALTER TABLE table\_name ADD column\_name datatype;

***ALTER TABLE - DROP COLUMN***

To delete a column in a table, use the following syntax (notice that some database systems don't allow deleting a column):

ALTER TABLE table\_name DROP COLUMN column\_name;

***ALTER TABLE - MODIFY COLUMN***

To change the data type of a column in a table, use the following syntax:

ALTER TABLE table\_name MODIFY COLUMN column\_name datatype;

***Create Constraints***

Constraints can be specified when the table is created with the CREATE TABLE statement, or after the table is created with the ALTER TABLE statement.

***Syntax***

CREATE TABLE table\_name (

column1 datatype constraint,

column2 datatype constraint,

column3 datatype constraint,

....

);

**MySQL Constraints**

SQL constraints are used to specify rules for the data in a table.

Constraints 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 table. If there is any violation between the constraint and the data action, the action is aborted.

Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

The following constraints are commonly used in SQL:

NOT NULL - Ensures that a column cannot have a NULL value

UNIQUE - Ensures that all values in a column are different

PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table

FOREIGN KEY - Prevents actions that would destroy links between tables

CHECK - Ensures that the values in a column satisfies a specific condition

DEFAULT - Sets a default value for a column if no value is specified

CREATE INDEX - Used to create and retrieve data from the database very quickly

**MySQL UNIQUE Constraint**

The UNIQUE constraint ensures that all values in a column are different.

Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns.

A PRIMARY KEY constraint automatically has a UNIQUE constraint.

However, you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

**MySQL PRIMARY KEY Constraint**

The PRIMARY KEY constraint uniquely identifies each record in a table.

Primary keys must contain UNIQUE values, and cannot contain NULL values.

A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).

***PRIMARY KEY on CREATE TABLE***

The following SQL creates a PRIMARY KEY on the "ID" column when the "Persons" table is created:

CREATE TABLE Persons (

ID int NOT NULL,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Age int,

PRIMARY KEY (ID)

);

**MySQL FOREIGN KEY Constraint**

The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.

A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the PRIMARY KEY in another table.

**MySQL CHECK Constraint**

The CHECK constraint is used to limit the value range that can be placed in a column.

If you define a CHECK constraint on a column it will allow only certain values for this column.

If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

***CHECK on CREATE TABLE***

The following SQL creates a CHECK constraint on the "Age" column when the "Persons" table is created. The CHECK constraint ensures that the age of a person must be 18, or older:

CREATE TABLE Persons (

ID int NOT NULL,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Age int,

CHECK (Age>=18)

);

**MySQL DEFAULT Constraint**

The DEFAULT constraint is used to set a default value for a column.

The default value will be added to all new records, if no other value is specified.

***DEFAULT on CREATE TABLE***

The following SQL sets a DEFAULT value for the "City" column when the "Persons" table is created:

CREATE TABLE Persons (

ID int NOT NULL,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Age int,

City varchar(255) DEFAULT 'Sandnes'

);

**MySQL CREATE INDEX Statement**

The CREATE INDEX statement is used to create indexes in tables.

Indexes are used to retrieve data from the database more quickly than otherwise. The users cannot see the indexes, they are just used to speed up searches/queries.

Note: Updating a table with indexes takes more time than updating a table without (because the indexes also need an update). So, only create indexes on columns that will be frequently searched against.

***CREATE INDEX Syntax***

Creates an index on a table. Duplicate values are allowed:

CREATE INDEX index\_name

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

CREATE UNIQUE INDEX Syntax

Creates a unique index on a table. Duplicate values are not allowed:

CREATE UNIQUE INDEX index\_name

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

***MySQL CREATE INDEX Example***

The SQL statement below creates an index named "idx\_lastname" on the "LastName" column in the "Persons" table:

CREATE INDEX idx\_lastname ON Persons (LastName);

If you want to create an index on a combination of columns, you can list the column names within the parentheses, separated by commas:

CREATE INDEX idx\_pname

ON Persons (LastName, FirstName);

***DROP INDEX Statement***

The DROP INDEX statement is used to delete an index in a table.

ALTER TABLE table\_name

DROP INDEX index\_name;

**AUTO INCREMENT**

What is an AUTO INCREMENT Field?

Auto-increment allows a unique number to be generated automatically when a new record is inserted into a table.

Often this is the primary key field that we would like to be created automatically every time a new record is inserted.

**MySQL AUTO\_INCREMENT Keyword**

MySQL uses the AUTO\_INCREMENT keyword to perform an auto-increment feature.

By default, the starting value for AUTO\_INCREMENT is 1, and it will increment by 1 for each new record.

The following SQL statement defines the "Personid" column to be an auto-increment primary key field in the "Persons" table:

CREATE TABLE Persons (

Personid int NOT NULL AUTO\_INCREMENT,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Age int,

PRIMARY KEY (Personid)

);

To let the AUTO\_INCREMENT sequence start with another value, use the following SQL statement:

ALTER TABLE Persons AUTO\_INCREMENT=100;

**MySQL Dates**

The most difficult part when working with dates is to be sure that the format of the date you are trying to insert, matches the format of the date column in the database.

As long as your data contains only the date portion, your queries will work as expected. However, if a time portion is involved, it gets more complicated.

**MySQL Date Data Types**

MySQL comes with the following data types for storing a date or a date/time value in the database:

DATE - format YYYY-MM-DD

DATETIME - format: YYYY-MM-DD HH:MI:SS

TIMESTAMP - format: YYYY-MM-DD HH:MI:SS

YEAR - format YYYY or YY

**MySQL CREATE VIEW Statement**

In SQL, a view is a virtual table based on the result-set of an SQL statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

You can add SQL statements and functions to a view and present the data as if the data were coming from one single table.

A view is created with the CREATE VIEW statement.

***CREATE VIEW Syntax***

CREATE VIEW view\_name AS

SELECT column1, column2, ...

FROM table\_name

WHERE condition;

**MySQL Updating a View**

A view can be updated with the CREATE OR REPLACE VIEW statement.

***CREATE OR REPLACE VIEW Syntax***

CREATE OR REPLACE VIEW view\_name AS

SELECT column1, column2, ...

FROM table\_name

WHERE condition;

**MySQL Dropping a View**

A view is deleted with the DROP VIEW statement.

***DROP VIEW Syntax***

DROP VIEW view\_name;

**MySQL String Functions**

Function Description

ASCII Returns the ASCII value for the specific character

CHAR\_LENGTH Returns the length of a string (in characters)

CHARACTER\_LENGTH Returns the length of a string (in characters)

CONCAT Adds two or more expressions together

CONCAT\_WS Adds two or more expressions together with a separator

FIELD Returns the index position of a value in a list of values

FIND\_IN\_SET Returns the position of a string within a list of strings

FORMAT Formats a number to a format like "#,###,###.##", rounded to a specified number of decimal places

INSERT Inserts a string within a string at the specified position and for a certain number of characters

INSTR Returns the position of the first occurrence of a string in another string

LCASE Converts a string to lower-case

LEFT Extracts a number of characters from a string (starting from left)

LENGTH Returns the length of a string (in bytes)

LOCATE m Returns the position of the first occurrence of a substring in a string

LOWER Converts a string to lower-case

LPAD Left-pads a string with another string, to a certain length

LTRIM Removes leading spaces from a string

MID Extracts a substring from a string (starting at any position)

POSITION Returns the position of the first occurrence of a substring in a string

REPEAT Repeats a string as many times as specified

REPLACE Replaces all occurrences of a substring within a string, with a new substring

REVERSE Reverses a string and returns the result

RIGHT Extracts a number of characters from a string (starting from right)

RPAD Right-pads a string with another string, to a certain length

RTRIM Removes trailing spaces from a string

SPACE Returns a string of the specified number of space characters

STRCMP Compares two strings

SUBSTR Extracts a substring from a string (starting at any position)

SUBSTRING Extracts a substring from a string (starting at any position)

SUBSTRING\_INDEX Returns a substring of a string before a specified number of delimiter occurs

TRIM Removes leading and trailing spaces from a string

UCASE Converts a string to upper-case

UPPER Converts a string to upper-case

**MySQL Numeric Functions**

Function Description

ABS Returns the absolute value of a number

ACOS Returns the arc cosine of a number

ASIN Returns the arc sine of a number

ATAN Returns the arc tangent of one or two numbers

ATAN2 Returns the arc tangent of two numbers

AVG Returns the average value of an expression

CEIL Returns the smallest integer value that is >= to a number

CEILING Returns the smallest integer value that is >= to a number

COS Returns the cosine of a number

COT Returns the cotangent of a number

COUNT Returns the number of records returned by a select query

DEGREES Converts a value in radians to degrees

DIV Used for integer division

EXP Returns e raised to the power of a specified number

FLOOR Returns the largest integer value that is <= to a number

GREATEST Returns the greatest value of the list of arguments

LEAST Returns the smallest value of the list of arguments

LN Returns the natural logarithm of a number

LOG Returns the natural logarithm of a number, or the logarithm of a number to a specified base

LOG10 Returns the natural logarithm of a number to base 10

LOG2 Returns the natural logarithm of a number to base 2

MAX Returns the maximum value in a set of values

MIN Returns the minimum value in a set of values

MOD Returns the remainder of a number divided by another number

PI Returns the value of PI

POW Returns the value of a number raised to the power of another number

POWER Returns the value of a number raised to the power of another number

RADIANS Converts a degree value into radians

RAND Returns a random number

ROUND Rounds a number to a specified number of decimal places

SIGN Returns the sign of a number

SIN Returns the sine of a number

SQRT Returns the square root of a number

SUM Calculates the sum of a set of values

TAN Returns the tangent of a number

TRUNCATE Truncates a number to the specified number of decimal places

**MySQL Date Functions**

Function Description

ADDDATE Adds a time/date interval to a date and then returns the date

ADDTIME Adds a time interval to a time/datetime and then returns the time/datetime

CURDATE Returns the current date

CURRENT\_DATE Returns the current date

CURRENT\_TIME Returns the current time

CURRENT\_TIMESTAMP Returns the current date and time

CURTIME Returns the current time

DATE Extracts the date part from a datetime expression

DATEDIFF Returns the number of days between two date values

DATE\_ADD Adds a time/date interval to a date and then returns the date

DATE\_FORMAT Formats a date

DATE\_SUB Subtracts a time/date interval from a date and then returns the date

DAY Returns the day of the month for a given date

DAYNAME Returns the weekday name for a given date

DAYOFMONTH Returns the day of the month for a given date

DAYOFWEEK Returns the weekday index for a given date

DAYOFYEAR Returns the day of the year for a given date

EXTRACT Extracts a part from a given date

FROM\_DAYS Returns a date from a numeric datevalue

HOUR Returns the hour part for a given date

LAST\_DAY Extracts the last day of the month for a given date

LOCALTIME Returns the current date and time

LOCALTIMESTAMP Returns the current date and time

MAKEDATE Creates and returns a date based on a year and a number of days value

MAKETIME Creates and returns a time based on an hour, minute, and second value

MICROSECOND Returns the microsecond part of a time/datetime

MINUTE Returns the minute part of a time/datetime

MONTH Returns the month part for a given date

MONTHNAME Returns the name of the month for a given date

NOW Returns the current date and time

PERIOD\_ADD Adds a specified number of months to a period

PERIOD\_DIFF Returns the difference between two periods

QUARTER Returns the quarter of the year for a given date value

SECOND Returns the seconds part of a time/datetime

SEC\_TO\_TIME Returns a time value based on the specified seconds

STR\_TO\_DATE Returns a date based on a string and a format

SUBDATE Subtracts a time/date interval from a date and then returns the date

SUBTIME Subtracts a time interval from a datetime and then returns the time/datetime

SYSDATE Returns the current date and time

TIME Extracts the time part from a given time/datetime

TIME\_FORMAT Formats a time by a specified format

TIME\_TO\_SEC Converts a time value into seconds

TIMEDIFF Returns the difference between two time/datetime expressions

TIMESTAMP Returns a datetime value based on a date or datetime value

TO\_DAYS Returns the number of days between a date and date "0000-00-00"

WEEK Returns the week number for a given date

WEEKDAY Returns the weekday number for a given date

WEEKOFYEAR Returns the week number for a given date

YEAR Returns the year part for a given date

YEARWEEK Returns the year and week number for a given date

**MySQL Advanced Functions**

Function Description

BIN Returns a binary representation of a number

BINARY Converts a value to a binary string

CASE Goes through conditions and return a value when the first condition is met

CAST Converts a value (of any type) into a specified datatype

COALESCE Returns the first non-null value in a list

CONNECTION\_ID Returns the unique connection ID for the current connection

CONV Converts a number from one numeric base system to another

CONVERT Converts a value into the specified datatype or character set

CURRENT\_USER Returns the user name and host name for the MySQL account that the server used to authenticate the current client

DATABASE Returns the name of the current database

IF Returns a value if a condition is TRUE, or another value if a condition is FALSE

IFNULL Return a specified value if the expression is NULL, otherwise return the expression

ISNULL Returns 1 or 0 depending on whether an expression is NULL

LAST\_INSERT\_ID Returns the AUTO\_INCREMENT id of the last row that has been inserted or updated in a table

NULLIF Compares two expressions and returns NULL if they are equal. Otherwise, the first expression is returned

SESSION\_USER Returns the current MySQL user name and host name

SYSTEM\_USER Returns the current MySQL user name and host name

USER Returns the current MySQL user name and host name

VERSION Returns the current version of the MySQL database