Access monitor: mysql -u [username] -p; (will prompt for password)

Show all databases: show databases;

Access database: mysql -u [username] -p [database] (will prompt for password)

Create new database: create database [database];

Select database: use [database];

Determine what database is in use: select database();

Show all tables: show tables;

Show table structure: describe [table];

List all indexes on a table: show index from [table];

Create new table with columns: CREATE TABLE [table] ([column] VARCHAR(120), [another-column] DATETIME);

Adding a column: ALTER TABLE [table] ADD COLUMN [column] VARCHAR(120);

Adding a column with an unique, auto-incrementing ID: ALTER TABLE [table] ADD COLUMN [column] int NOT NULL AUTO\_INCREMENT PRIMARY KEY;

Inserting a record: INSERT INTO [table] ([column], [column]) VALUES ('[value]', '[value]');

MySQL function for datetime input: NOW()

Selecting records: SELECT \* FROM [table];

Explain records: EXPLAIN SELECT \* FROM [table];

Selecting parts of records: SELECT [column], [another-column] FROM [table];

Counting records: SELECT COUNT([column]) FROM [table];

Counting and selecting grouped records: SELECT \*, (SELECT COUNT([column]) FROM [table]) AS count FROM [table] GROUP BY [column];

Selecting specific records: SELECT \* FROM [table] WHERE [column] = [value]; (Selectors: <, >, !=; combine multiple selectors with AND, OR)

Select records containing [value]: SELECT \* FROM [table] WHERE [column] LIKE '%[value]%';

Select records starting with [value]: SELECT \* FROM [table] WHERE [column] LIKE '[value]%';

Select records starting with val and ending with ue: SELECT \* FROM [table] WHERE [column] LIKE '[val\_ue]';

Select a range: SELECT \* FROM [table] WHERE [column] BETWEEN [value1] and [value2];

Select with custom order and only limit: SELECT \* FROM [table] WHERE [column] ORDER BY [column] ASC LIMIT [value]; (Order: DESC, ASC)

Updating records: UPDATE [table] SET [column] = '[updated-value]' WHERE [column] = [value];

Deleting records: DELETE FROM [table] WHERE [column] = [value];

Delete *all records* from a table (without dropping the table itself): DELETE FROM [table]; (This also resets the incrementing counter for auto generated columns like an id column.)

Delete all records in a table: truncate table [table];

Removing table columns: ALTER TABLE [table] DROP COLUMN [column];

Deleting tables: DROP TABLE [table];

Deleting databases: DROP DATABASE [database];

Custom column output names: SELECT [column] AS [custom-column] FROM [table];

Export a database dump (more info [here](http://stackoverflow.com/a/21091197/1815847)): mysqldump -u [username] -p [database] > db\_backup.sql

Use --lock-tables=false option for locked tables (more info [here](http://stackoverflow.com/a/104628/1815847)).

Import a database dump (more info [here](http://stackoverflow.com/a/21091197/1815847)): mysql -u [username] -p -h localhost [database] < db\_backup.sql

Logout: exit;

**Aggregate functions**

Select but without duplicates: SELECT distinct name, email, acception FROM owners WHERE acception = 1 AND date >= 2015-01-01 00:00:00

Calculate total number of records: SELECT SUM([column]) FROM [table];

Count total number of [column] and group by [category-column]: SELECT [category-column], SUM([column]) FROM [table] GROUP BY [category-column];

Get largest value in [column]: SELECT MAX([column]) FROM [table];

Get smallest value: SELECT MIN([column]) FROM [table];

Get average value: SELECT AVG([column]) FROM [table];

Get rounded average value and group by [category-column]: SELECT [category-column], ROUND(AVG([column]), 2) FROM [table] GROUP BY [category-column];

**Multiple tables**

Select from multiple tables: SELECT [table1].[column], [table1].[another-column], [table2].[column] FROM [table1], [table2];

Combine rows from different tables: SELECT \* FROM [table1] INNER JOIN [table2] ON [table1].[column] = [table2].[column];

Combine rows from different tables but do not require the join condition: SELECT \* FROM [table1] LEFT OUTER JOIN [table2] ON [table1].[column] = [table2].[column]; (The left table is the first table that appears in the statement.)

Rename column or table using an *alias*: SELECT [table1].[column] AS '[value]', [table2].[column] AS '[value]' FROM [table1], [table2];

## Basic Commands in SQL:

|  |  |  |
| --- | --- | --- |
| Command | Syntax | Description |
| ALTER table | ALTER TABLE table\_name ADD column\_name datatype; | It is used to add columns to a table in a database |
| AND | SELECT column\_name(s)FROM table\_nameWHERE column\_1 = value\_1  AND column\_2 = value\_2; | It is an operator that is used to combine two conditions |
| AS | SELECT column\_name AS ‘Alias’FROM table\_name; | It is a keyword in SQL that is used to rename a column or table using an alias name |
| AVG | SELECT AVG(column\_name)FROM table\_name; | It is used to aggregate a numeric column and return its average |
| BETWEEN | SELECT column\_name(s)FROM table\_nameWHERE column\_name BETWEEN value\_1 AND value\_2; | It is an operation used to filter the result within a certain range |
| CASE | SELECT column\_name,CASEWHEN condition THEN ‘Result\_1’WHEN condition THEN ‘Result\_2’ELSE ‘Result\_3’ENDFROM table\_name; | It is a statement used to create different outputs inside a SELECT statement |
| COUNT | SELECT COUNT(column\_name)FROM table\_name; | It is a function that takes the name of a column as an argument and counts the number of rows when the column is not NULL |
| Create TABLE | CREATE TABLE table\_name (  column\_1 datatype,   column\_2 datatype,   column\_3 datatype); | It is used to create a new table in a database and specify the name of the table and columns inside it |
| DELETE | DELETE FROM table\_nameWHERE some\_column = some\_value; | It is used to remove the rows from a table |
| GROUP BY | SELECT column\_name, COUNT(\*)FROM table\_nameGROUP BY column\_name; | It is a clause in SQL used for [aggregate functions](https://intellipaat.com/blog/tutorial/sql-tutorial/sql-functions/) in collaboration with the SELECT statement |
| HAVING | SELECT column\_name, COUNT(\*)FROM table\_nameGROUP BY column\_nameHAVING COUNT(\*) > value; | It is used in SQL because the WHERE keyword cannot be used in aggregating functions |
| [INNER JOIN](https://intellipaat.com/blog/tutorial/sql-tutorial/inner-join-in-sql/) | SELECT column\_name(s)FROM table\_1JOIN table\_2  ON table\_1.column\_name = table\_2.column\_name; | It is used to combine rows from different tables if the Join condition goes TRUE |
| INSERT | INSERT INTO table\_name (column\_1, column\_2, column\_3) VALUES (value\_1, ‘value\_2’, value\_3); | It is used to add new rows to a table |
| IS NULL/ IS NOT NULL | SELECT column\_name(s)FROM table\_nameWHERE column\_name IS NULL; | It is an operator used with the WHERE clause to check for the empty values |
| [LIKE](https://intellipaat.com/blog/tutorial/sql-tutorial/like-and-between-operators-in-sql/) | SELECT column\_name(s)FROM table\_nameWHERE column\_name LIKE pattern; | It is a special operator used with the WHERE clause to search for a specific pattern in a column |
| LIMIT | SELECT column\_name(s)FROM table\_nameLIMIT number; | It is a clause to specify the maximum number of rows the result set must have |
| MAX | SELECT MAX(column\_name)FROM table\_name; | It is a function that takes a number of columns as an argument and returns the largest value among them |
| MIN | SELECT MIN(column\_name)FROM table\_name; | It is a function that takes a number of columns as an argument and returns the smallest value among them |
| OR | SELECT column\_nameFROM table\_nameWHERE column\_name = value\_1   OR column\_name = value\_2; | It is an operator that is used to filter the result set to contain only the rows where either condition is TRUE |
| ORDER BY | SELECT column\_nameFROM table\_nameORDER BY column\_name ASC | DESC; | It is a clause used to sort the result set by a particular column either numerically or alphabetically |
| [OUTER JOIN](https://intellipaat.com/blog/tutorial/sql-tutorial/full-join-sql/) | SELECT column\_name(s)FROM table\_1LEFT JOIN table\_2  ON table\_1.column\_name = table\_2.column\_name; | It issued to combine rows from different tables even if the condition is NOT TRUE |
| ROUND | SELECT ROUND(column\_name, integer)FROM table\_name; | It is a function that takes the column name and an integer as an argument and rounds the values in a column to the number of decimal places specified by an integer |
| SELECT | SELECT column\_name FROM table\_name; | It is a statement that is used to fetch data from a database |
| [SELECT DISTINCT](https://intellipaat.com/blog/tutorial/sql-tutorial/distinct/) | SELECT DISTINCT column\_nameFROM table\_name; | It is used to specify that the statement is a query that returns unique values in specified columns |
| SUM | SELECT SUM(column\_name)FROM table\_name; | It is a function used to return the sum of values from a particular column |
| [UPDATE](https://intellipaat.com/blog/tutorial/sql-tutorial/update-query/) | UPDATE table\_nameSET some\_column = some\_valueWHERE some\_column = some\_value; | It is used to edit rows in a table |
| [WHERE](https://intellipaat.com/blog/tutorial/sql-tutorial/where-clause/) | SELECT column\_name(s)FROM table\_nameWHERE column\_name operator value; | It is a clause used to filter the result set to include the rows where the condition is TRUE |
| WITH | WITH temporary\_name AS (SELECT \*FROM table\_name)SELECT \*FROM temporary\_nameWHERE column\_name operator value; | It is used to store the result of a particular query in a temporary table using an alias |

## Commands and syntax for querying data from a single table and multiple tables:

|  |  |
| --- | --- |
| Single Table | Multiple Table |
| SELECT c1 FROM t To select the data in Column c1 from table t | SELECT c1, c2 FROM t1 INNER JOIN t2 on conditionSelect column c1 and c2 from table t1 and perform an inner join between t1 and t2 |
| SELECT \* FROM t To select all rows and columns from table t | SELECT c1, c2 FROM t1 LEFT JOIN t2 on condition Select column c1 and c2 from table t1 and perform a left join between t1 and t2 |
| SELECT c1 FROM t WHERE c1 = ‘test’ To select data in column c1 from table t, where c1=test | SELECT c1, c2 FROM t1 RIGHT JOIN t2 on condition Select column c1 and c2 from table t1 and perform a right join between t1 and t2 |
| SELECT c1 FROM t ORDER BY c1 ASC (DESC) To select data in column c1 from table t either in ascending or descending order | SELECT c1, c2 FROM t1 FULL OUTER JOIN t2 on condition Select column c1 and c2 from table t1 and perform a full outer join between t1 and t2 |
| SELECT c1 FROM t ORDER BY c1LIMIT n OFFSET offset To skip the offset of rows and return the next n rows | SELECT c1, c2 FROM t1 CROSS JOIN t2 Select column c1 and c2 from table t1 and produce a Cartesian product of rows in a table |
| SELECT c1, aggregate(c2) FROM t GROUP BY c1 To group rows using an aggregate function | SELECT c1, c2 FROM t1, t2Select column c1 and c2 from table t1 and produce a Cartesian product of rows in a table |
| SELECT c1, aggregate(c2) FROM t GROUP BY c1HAVING condition Group rows using an aggregate function and filter these groups using ‘HAVING’ clause | SELECT c1, c2 FROM t1 A INNER JOIN t2 B on condition Select column c1 and c2 from table t1 and join it to itself using INNER JOIN clause |

### **4. Working with Tables**

|  |  |  |
| --- | --- | --- |
| CREATE | **CREATE** **table** (<table\_name> <field1(**type**)>, <field2, (**type**)>…); | Used to create a new table |
| INSERT | **INSERT** **INTO** <table\_name>(field1, field2, …) **VALUES**(value1, value2,…) | This command is used to insert one or more rows of data into the table. |
| UPDATE | **UPDATE** <table\_name> SET field\_name1 = value1, field\_name2 = value2, [**WHERE** <condition>]; | Used to modify existing data in a table.  The WHERE command here is optional here. |
| SELECT | **SELECT** <field\_list> **FROM** <table\_name>; | Used to select list values from a table. |
| SELECT DISTINCT | **SELECT** **DISTINCT** <field\_list> **FROM** <table\_name>; | Used to select only unique values from a list. |
| WHERE | **SELECT** <filed\_list> **FROM** <table\_name> **WHERE** <condition>; | The WHERE clause allows to select values based on specified conditions. |
| ORDER BY | **SELECT** <field\_list> **FROM** <table\_name> **ORDER** **BY** <field\_name> [**ASC**|**DESC**]; | The ORDER BY clause is used to sort the queried result set in either ascending or Descending Order. By default, it will be Ascending Order. |
| AND | … expression\_1 **AND** expression\_2 | This is a logical operator. It is mainly used with the WHERE clause. Here, the query statement is executed only when both the expressions are true. |
| OR | … expression\_1 **OR** expression\_2 | This is a logical operator. It is mainly used with the WHERE clause. Here, the query statement is executed only when one or both the expressions are true. |
| IN | **SELECT** <field\_list> **FROM** <table\_name> **WHERE** <expression|column\_1> **IN** (value1, value2, …) | The IN operator is used with the WHERE clause. It enables to determine if a specified value in a list matches in another set of values. |
| BETWEEN | … **expression** [NOT] **BETWEEN** **expression\_1** **AND** **expression\_2** | This operator is also used with the WHERE clause. It is used to specify whether a value is in a specified range. |
| LIMIT | **SELECT** <field\_list> **FROM** <table\_name> **LIMIT** <first\_row\_offset> <number **of** **rows** **to** be returned>; | This clause is used with SELECT statement to specify the number of rows to be returned. |
| IS NULL | Value IS **NULL** | It is used to test if a value is NULL or not. If it is NULL, the expression returns true, else false. |
| INNER JOIN | **SELECT** <field\_list> **FROM** <table1\_name> **INNER** **JOIN** <table2\_name> **ON** <join\_condition> | This is a filter clause that matches each row in one table with every row in the other table thus enabling to query only those rows that have corresponding columns from both tables. |
| LEFT JOIN | **SELECT** <field\_names> **FROM** <table1\_name> **LEFT** **JOIN** <table2\_name> **ON** join\_condition; | It allows you to query data from multiple tables. It matches each row from the first table to each row in the second table on the join\_condition |
| RIGHT JOIN | **SELECT** <field\_names> **FROM** <table1\_name> **LEFT** **JOIN** <table2\_name> **ON** join\_condition; | Same as LEFT\_JOIN except that the table manipulation is in reverse order.  It matches each row from the second table to each row in the first table on the join\_condition |
| CROSS JOIN | **SELECT** \* **FROM** <table1\_name> **CROSS** **JOIN** <table2\_name>; | It returns the cartesian product of rows from the joined tables. |
| GROUP BY | **SELECT** <field1, field2, field3…> **FROM** <table1\_name> **WHERE** <condition/expression> **GROUP** **BY** <field1, field2, field3…> | This command enables to group rows into subgroups based on column or expression values. |
| HAVING | **SELECT** <field1, field2, field3…> **FROM** <table1\_name> **WHERE** <condition/expression> **GROUP** **BY** <field1, field2, field3…> **HAVING** <group\_condition> | It is generally used with the GROUP BY clause. It is used to specify filter conditions for group of rows. |
| ROLL UP | **SELECT** <field\_name1>, **SUM**(column\_name) <field\_name2> **FROM** <table\_name> **GROUP** **BY** <field\_name1>**WITH** **ROLLUP**; | Used to generate the subtotals as well as grandtotals of fieldvalues. |
| EXISTS | **SELECT** <field\_list> **FROM** <table\_list> **WHERE** [**NOT**] **EXISTS**(subquery); | It is a Boolean operator that returns either true or false. It is generally used to determine if a query/subquery has returned any number of rows. |
| INTERSECT | (**SELECT** <field\_list> **FROM** <table1\_name>) **INTERSECT** (**SELECT** <field\_list> **FROM** <table2\_name>) | This is a set operator which returns only distinct rows of two or more queries. |
| UNION | **SELECT** <field\_list> **UNION** [**DISTINCT** | **ALL**] **SELECT** <field\_list> **UNION** [**DISTINCT** | **ALL**] | This command combines two or more result sets from multiple SELECT queries and returns a single result set. |
| UPDATE JOIN | **UPDATE** <table1>, <table2>. [**INNER** **JOIN** | **LEFT** **JOIN**] table1.common\_field = table2.common\_field SET table1.field1 = newvalues … **WHERE** <condition> | JOIN clauses when used with the UPDATE statement is called as UPDATE JOIN. |
| DELETE | **DELETE** **FROM** <table\_name>  **WHERE** <condition> | This command is used to delete data from table. |
| DELETE JOIN | **DELETE** <field1>,<field2> **FROM** <table1> **INNER** **JOIN** <table2> **ON** table1.key = table2.key WHERE <condition>; | This command is used to delete data from multiple tables using the JOIN statement. |
| ON DELETE CASCADE | **SELECT** <table\_name> **FROM** <referential\_constraints> **WHERE** <constraint\_schema = 'database\_name'> **AND** referenced\_table\_name = 'parent\_table' **AND** delete\_rule = 'CASCADE' | This command enables deleting data from child table automatically when data from master table is deleted. |
| REPLACE | REPLACE <**table\_name**>(<**field1**>, <**field2**>,…) VALUES (<**value1**>, <**value2**>, …) | This command is used to inert or update data in a table. |