

# SQL COMMANDS

## Data Definition Language (DDL)

### 1. The SQL CREATE Command:

Utilized for the creation of databases and various database objects such as tables, views, indexes, stored procedures, functions, and triggers.

**Syntax** - CREATE table new\_table( id int , name varchar(20), age int);

**Output :**

✓ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0107 seconds.)

```
CREATE table new_table( id int , name varchar(20), age int);
```

### 2. The SQL ALTER Command:

Used to modify the design or structure of an existing database. Changing attributes of database elements, like altering table schemas or modifying column properties.

**Syntax** - alter TABLE new\_table add surname varchar(30);

**Output :**

✓ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0214 seconds.)

```
alter TABLE new_table add surname varchar(30);
```

### 3. The SQL TRUNCATE Command:

Employed to delete all records from a table, including freeing up space allocated for those records. Quickly remove all data from a table while retaining the table structure for future use.

**Syntax** - TRUNCATE TABLE new\_table ;

**Output :**

✓ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0010 seconds.)

```
truncate TABLE new_table;
```

#### 4. The SQL DROP Command:

Employed to remove objects from a database. Deleting tables, views, indexes, or other database structures when they are no longer needed.

**Syntax** - DROP TABLE new\_table ;

**Output :**

✔ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0048 seconds.)

```
drop TABLE new_table;
```

#### 5. The SQL RENAME Command :

Utilized for changing the name of an existing database object. Updating names of database elements to better reflect their purpose or to comply with naming conventions.

**Syntax** – ALTER TABLE new\_table RENAME to changed\_name ;

**Output:**

✔ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0105 seconds.)

```
ALTER TABLE new_table RENAME to changed_name;
```

## Data Query Language (DQL)

#### 1. The SQL SELECT Command :

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

**Syntax** – SELECT \* FROM accounts;

```
SELECT account, year_established, revenue FROM `accounts`
```

## Output:

account	sector	year_established	revenue	employees	office_location	subsidiary_of
Acme Corporation	technology	1996	1100.04	2822	United States	
Betasoloin	medical	1999	251.41	495	United States	
Betatech	medical	1986	647.18	1185	Kenya	
Bioholding	medical	2012	587.34	1356	Philippines	
Bioplex	medical	1991	326.82	1016	United States	
Blackzim	retail	2009	497.11	1588	United States	
Bluth Company	technology	1993	1242.32	3027	United States	Acme Corporation

account	year_established	revenue
Acme Corporation	1996	1100.04
Betasoloin	1999	251.41
Betatech	1986	647.18
Bioholding	2012	587.34
Bioplex	1991	326.82
Blackzim	2009	497.11
Bluth Company	1993	1242.32

## Data Manipulation Language (DML)

### 1. The SQL INSERT Command :

Used to add new rows (records) to a table.

**Syntax** – INSERT INTO products (product, series, sales\_price) VALUES

("sample1","RTX",5999);

### Output :

product	series	sales_price
GTX Basic	GTX	550
GTX Pro	GTX	4821
MG Special	MG	55
MG Advanced	MG	3393
GTX Plus Pro	GTX	5482
GTX Plus Basic	GTX	1096
GTK 500	GTK	26768
sample1	RTX	5999

## 2. The SQL UPDATE Command :

Modifies existing data within a table. It can update one or more columns for all rows that meet the condition specified.

**Syntax** - UPDATE products set product = "new\_card" WHERE series= "RTX";

**Output :**

product	series	sales_price
GTX Basic	GTX	550
GTX Pro	GTX	4821
MG Special	MG	55
MG Advanced	MG	3393
GTX Plus Pro	GTX	5482
GTX Plus Basic	GTX	1096
GTK 500	GTK	26768
new_card	RTX	5999

## 3. The SQL DELETE Command:

Removes one or more rows from a table. Without a specific condition, it will delete all rows in the table.

**Syntax** - DELETE FROM products where series = "RTX";

**Output :**

product	series	sales_price
GTX Basic	GTX	550
GTX Pro	GTX	4821
MG Special	MG	55
MG Advanced	MG	3393
GTX Plus Pro	GTX	5482
GTX Plus Basic	GTX	1096
GTK 500	GTK	26768

## 4. The SQL LOCK Command :

the LOCK TABLE command is used to apply a lock to a table to restrict other users from accessing it in a specific way, based on the type of lock applied.

**Syntax** – LOCK TABLE products READ;

## Output :

✓ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0004 seconds.)

```
LOCK TABLES products READ;
```

[ [Edit inline](#) ] [ [Edit](#) ] [ [Create PHP code](#) ]

⚠ Current selection does not contain a unique column. Grid edit, checkbox, Edit, Copy and Delete features are not available. [?](#)

✓ Showing rows 0 - 6 (7 total, Query took 0.0259 seconds.)

```
SELECT * from products;
```

☐ Profiling [ [Edit inline](#) ] [ [Edit](#) ] [ [Explain SQL](#) ] [ [Create PHP code](#) ] [ [Refresh](#) ]

☐ Show all | Number of rows:  Filter rows:

Extra options

product	series	sales_price
GTX Basic	GTX	550
GTX Pro	GTX	4821
MG Special	MG	55
MG Advanced	MG	3393
GTX Plus Pro	GTX	5482
GTX Plus Basic	GTX	1096
GTK 500	GTK	26768

## Data Control Language (DCL)

### 1. The SQL GRANT command :

This command is used to give users access privileges to the database. These privileges include SELECT, INSERT, UPDATE, DELETE, and more, which can be granted to users or roles.


**Syntax** - GRANT SELECT, INSERT, UPDATE

ON **Syntax** sampledatabase.products

TO root@localhost;



SHOW GRANTS FOR root@localhost ;

## Output:

 MySQL returned an empty result set (i.e. zero rows). (Query took 0.0083 seconds.)

```
GRANT SELECT, INSERT, UPDATE ON sampledatabase.products TO root@localhost;
```

[\[ Edit inline \]](#) [\[ Edit \]](#) [\[ Create PHP code \]](#)

 Current selection does not contain a unique column. Grid edit, checkbox, Edit, Copy and Delete features are not available. 

Your SQL query has been executed successfully.

```
SHOW GRANTS FOR root@localhost;
```

☐ Profiling [\[ Edit inline \]](#) [\[ Edit \]](#) [\[ Create PHP code \]](#) [\[ Refresh \]](#)

Extra options

**Grants for root@localhost**

```
GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, DROP...
GRANT ALLOW_NONEXISTENT_DEFINER,APPLICATION_PASSWO...
GRANT SELECT, INSERT, UPDATE ON `sampledatabase`.`...
GRANT PROXY ON ``@`` TO `root`@`localhost` WITH GR...
```

## 2. The SQL REVOKE Command:

This command is used to remove previously granted privileges from a user or role. It's the opposite of GRANT and ensures that unauthorized users can't access sensitive data.

### Syntax –

REVOKE ALL PRIVILEGES, GRANT OPTION

FROM root@localhost;

SHOW GRANTS FOR root@localhost ; Output :

## Output :

 MySQL returned an empty result set (i.e. zero rows). (Query took 0.0102 seconds.)

```
REVOKE ALL PRIVILEGES, GRANT OPTION FROM root@localhost;
```

[\[ Edit inline \]](#) [\[ Edit \]](#) [\[ Create PHP code \]](#)

 Current selection does not contain a unique column. Grid edit, checkbox, Edit, Copy and Delete features are not available. 

Your SQL query has been executed successfully.

```
SHOW GRANTS FOR root@localhost;
```

☐ Profiling [\[ Edit inline \]](#) [\[ Edit \]](#) [\[ Create PHP code \]](#) [\[ Refresh \]](#)

Extra options

**Grants for root@localhost**

```
GRANT USAGE ON *.* TO 'root'@'localhost'
```

```
GRANT PROXY ON ``@`` TO 'root'@'localhost' WITH GR...
```

## Transaction Control Language (TCL)


### 1. The SQL COMMIT Command :

This command is used to permanently save all changes made during the current transaction. Once a transaction is committed, it cannot be undone.

**Syntax** - INSERT into products(product, series , sales\_price) VALUES ("sample ", "RTX", 8999) ;


COMMIT;

## Output :

 1 row inserted. (Query took 0.0005 seconds.)

```
INSERT into products(product, series , sales_price) VALUES ("sample ", "RTX", 8999);
```

[\[ Edit inline \]](#) [\[ Edit \]](#) [\[ Create PHP code \]](#)

 MySQL returned an empty result set (i.e. zero rows). (Query took 0.0004 seconds.)

```
COMMIT;
```

[\[ Edit inline \]](#) [\[ Edit \]](#) [\[ Create PHP code \]](#)

product	series	sales_price
GTX Basic	GTX	550
GTX Pro	GTX	4821
MG Special	MG	55
MG Advanced	MG	3393
GTX Plus Pro	GTX	5482
GTX Plus Basic	GTX	1096
GTK 500	GTK	26768
sample	RTX	8999

## 2. The SQL ROLLBACK Command :


Used to undo transactions that have not yet been committed. It can revert the database to the last committed state. ROLLBACK is useful in case of an error or if the transaction needs to be aborted.

Syntax –INSERT INTO products ( product, series, sales\_price)

VALUES ( 'Bluetooth Speaker', 'BTS300', 1299);


ROLLBACK;


Output :

 MySQL returned an empty result set (i.e. zero rows). (Query took 0.0003 seconds.)

```
ROLLBACK;
```

[\[ Edit inline \]](#) [\[ Edit \]](#) [\[ Create PHP code \]](#)

 Current selection does not contain a unique column. Grid edit, checkbox, Edit, Copy and Delete features are not available.

 Showing rows 0 - 14 (15 total, Query took 0.0004 seconds.)

```
SELECT * from products where sales_price > 1;
```

☐ Profiling [\[ Edit inline \]](#) [\[ Edit \]](#) [\[ Explain SQL \]](#) [\[ Create PHP code \]](#) [\[ Refresh \]](#)



product	series	sales_price
GTX Basic	GTX	550
GTX Pro	GTX	4821
MG Special	MG	55
MG Advanced	MG	3393
GTX Plus Pro	GTX	5482
GTX Plus Basic	GTX	1096
GTK 500	GTK	26768
sample	RTX	8999

### 3. The SQL SAVEPOINT Command:

This command allows you to set a savepoint within a transaction, which is a point to which you can later roll back. It helps create a partial rollback point within a long transaction.

#### Syntax –

```
INSERT INTO products ( product, series, sales_price)
VALUES ( ' Speaker', 'B00', 1299);
```

```
SAVEPOINT Ins_1 ;
```

```
INSERT INTO products ( product, series, sales_price)
VALUES ( ' headset', 'T00', 4899);
```

```
ROLLBACK to Ins_1;
```

#### Output:

<p>✓ 1 row inserted. (Query took 0.0004 seconds.)</p> <pre>INSERT INTO products ( product, series, sales_price) VALUES ( ' Speaker', 'B00', 1299);</pre> <p>[ Edit inline ] [ Edit ] [ Create PHP code ]</p>
<p>✓ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0003 seconds.)</p> <pre>SAVEPOINT Ins_1;</pre> <p>[ Edit inline ] [ Edit ] [ Create PHP code ]</p>
<p>✓ 1 row inserted. (Query took 0.0001 seconds.)</p> <pre>INSERT INTO products ( product, series, sales_price) VALUES ( ' headset', 'T00', 4899);</pre> <p>[ Edit inline ] [ Edit ] [ Create PHP code ]</p>
<p>✓ MySQL returned an empty result set (i.e. zero rows). (Query took 0.0003 seconds.)</p> <pre>ROLLBACK to Ins_1;</pre> <p>[ Edit inline ] [ Edit ] [ Create PHP code ]</p>

## The SQL JOIN Commands :

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

### INNER JOIN

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

**Syntax** - SELECT \* FROM sales\_transactions

INNER JOIN product\_dim ON sales\_transactions.product\_id = product\_dim.product\_id;

### Output:

order_id	product_id	cust_id	product_quantity	order_date	product_id	product_name	product_price	effective_start_date	effective_end_date	current_ind
146740	614	121960	1	2019-01-24	614	Google Phone	610.00	2019-06-01	2019-09-30	N
146740	614	121960	1	2019-01-24	614	Google Phone	579.00	2019-10-01	2019-10-31	N
146740	614	121960	1	2019-01-24	614	Google Phone	599.00	2019-03-01	2019-05-31	N
146740	614	121960	1	2019-01-24	614	Google Phone	620.00	1900-01-01	2019-02-28	N
146740	614	121960	1	2019-01-24	614	Google Phone	589.00	2019-11-01	2019-11-30	N
146740	614	121960	1	2019-01-24	614	Google Phone	550.00	2019-12-01	9999-12-31	Y
141749	394	113809	1	2019-01-24	394	Flatscreen TV	279.00	2019-10-01	2019-11-30	N
141749	394	113809	1	2019-01-24	394	Flatscreen TV	289.00	2019-04-01	2019-07-31	N

### LEFT JOIN

The LEFT JOIN keyword returns all records from the left table (table1), and the matching records from the right table (table2). The result is 0 records from the right side, if there is no match.

**Syntax** -

SELECT \*

FROM product\_dim

LEFT JOIN sales\_transactions

ON product\_dim.product\_id = sales\_transactions.product\_id;

## Output:

product_id	product_name	product_price	effective_start_date	effective_end_date	current_ind	order_id	product_id	cust_id	product_quantity	order_date
614	Google Phone	610.00	2019-06-01	2019-09-30	N	146740	614	121960	1	2019-01-24
614	Google Phone	579.00	2019-10-01	2019-10-31	N	146740	614	121960	1	2019-01-24
614	Google Phone	599.00	2019-03-01	2019-05-31	N	146740	614	121960	1	2019-01-24
614	Google Phone	620.00	1900-01-01	2019-02-28	N	146740	614	121960	1	2019-01-24
614	Google Phone	589.00	2019-11-01	2019-11-30	N	146740	614	121960	1	2019-01-24
614	Google Phone	550.00	2019-12-01	9999-12-31	Y	146740	614	121960	1	2019-01-24
394	Flatscreen TV	279.00	2019-10-01	2019-11-30	N	141749	394	113809	1	2019-01-24

## RIGHT JOIN

The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records from the left table (table1). The result is 0 records from the left side, if there is no match.

**Syntax** - SELECT \*

FROM sales\_transactions

RIGHT JOIN customer\_dim

ON sales\_transactions.cust\_id = customer\_dim.cust\_id;

## Output :

order_id	product_id	cust_id	product_quantity	order_date	cust_id	cust_address	cust_age	effective_start_date	effective_end_date	current_ind
150093	394	142788	1	2019-01-24	142788	289 13th St, Boston, MA 02215	15	1900-01-01	9999-12-31	Y
146354	981	135145	1	2019-01-24	135145	829 Wilson St, Portland, OR 97035	16	1900-01-01	9999-12-31	Y
147640	422	128521	1	2019-01-24	128521	400 Cedar St, Austin, TX 73301	16	1900-01-01	9999-12-31	Y
143482	715	204040	1	2019-01-24	204040	238 Cedar St, New York City, NY 10001	15	1900-01-01	9999-12-31	Y
142981	953	218041	1	2019-01-24	218041	39 Maple St, San Francisco, CA 94016	16	1900-01-01	9999-12-31	Y
141612	953	126317	1	2019-01-24	126317	892 12th St, Los Angeles, CA 90001	16	1900-01-01	9999-12-31	Y
141840	715	235573	1	2019-01-24	235573	739 11th St, Boston, MA 02215	16	1900-01-01	9999-12-31	Y
147479	406	130640	2	2019-01-24	130640	417 6th St, San Francisco, CA 94016	15	1900-01-01	9999-12-31	Y

## The SQL UNION Operator

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

### Syntax –

```
SELECT product_id, product_price, product_name AS item FROM product_dim
```

```
UNION
```

```
SELECT cust_address AS item FROM customer_dim;
```

### Output :

cust_id
121960
113809
102255
190891
108376
240579
171873
180633
228883
238405
226319
225279

### The SQL 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.

### Syntax –

```
SELECT SUM(product_price), product_name
```

```
FROM product_dim
```

```
GROUP BY product_name;
```

### Output :

SUM(product_price)	product_name
604.98	20in Monitor
1939.95	27in 4K Gaming Monitor
646.99	27in FHD Monitor
4289.98	34in Ultrawide Monitor
36.42	AA Batteries (4-pack)
25.12	AAA Batteries (4-pack)
812.00	Apple Airpods Headphones
739.96	Bose SoundSport Headphones
1437.00	Flatscreen TV
1943.00	Golf Handheld GPS Caddie
3547.00	Google Phone
3293.00	iPhone
4180.00	LG Dryer
1645.00	LG Washing Machine
125.93	Lightning Charging Cable
10397.00	Macbook Pro Laptop
5826.99	ThinkPad Laptop
152.86	USB-C Charging Cable
93.95	Wired Headphones

### The SQL HAVING Clause

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

#### Syntax –

```
SELECT COUNT(product_id), product_name
```

```
FROM product_dim
```

```
GROUP BY product_name
```

```
HAVING COUNT(product_id) > 5;
```

#### Output :

COUNT(product_id)	product_name
10	34in Ultrawide Monitor
8	AA Batteries (4-pack)
8	AAA Batteries (4-pack)
6	Bose SoundSport Headphones
6	Google Phone
7	LG Dryer
9	Lightning Charging Cable
6	Macbook Pro Laptop
6	ThinkPad Laptop
10	USB-C Charging Cable
7	Wired Headphones

### The SQL 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.

**Syntax** - `SELECT * FROM sales_transactions WHERE product_quantity > ALL (SELECT product_quantity FROM sales_transactions WHERE product_quantity < 5);`

**Output:**

order_id	product_id	cust_id	product_quantity	order_date
143416	692	163935	7	2019-01-24
147196	692	236078	5	2019-01-25
265060	406	193512	5	2019-10-08
271094	692	180976	6	2019-10-08
277471	692	238194	7	2019-10-09
269696	406	177092	5	2019-10-10
148664	692	234505	5	2019-01-28
145791	406	236744	5	2019-01-28
275070	406	110303	5	2019-10-11
144717	692	134154	5	2019-01-29
276704	692	159353	5	2019-10-12
265644	406	179542	6	2019-10-12

### The other SQL OPERATORS :

**AND** - TRUE if all the conditions separated by AND is TRUE

**Syntax** - SELECT product\_id

FROM sales\_transactions

WHERE product\_quantity > ALL (

SELECT product\_quantity

FROM sales\_transactions

WHERE product\_quantity < 5

);

### Output:

product_id
692
692
406
692
692
406
692

**BETWEEN** - TRUE if the operand is within the range of comparisons

**Syntax** – SELECT cust\_id, cust\_age

FROM customer\_dim

WHERE cust\_age BETWEEN 20 AND 40;

**Output :**

cust_id	cust_age
214237	20
231714	20
201909	20
170483	20
168788	20
202851	20
131656	20
130652	20
201527	20

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

**Syntax** – SELECT \*

FROM product\_dim

WHERE product\_id IN (582, 216, 614);

**Output :**

product_id	product_name	product_price	effective_start_date	effective_end_date	current_ind
582	iPhone	635.00	2019-10-01	2019-11-30	N
582	iPhone	689.00	2019-06-01	2019-09-30	N
582	iPhone	649.00	2019-04-01	2019-05-31	N
582	iPhone	700.00	1900-01-01	2019-03-31	N
216	LG Dryer	610.00	2019-06-01	2019-06-30	N
216	LG Dryer	590.00	2019-10-01	2019-11-30	N
216	LG Dryer	600.00	2019-07-01	2019-08-31	N
216	LG Dryer	630.00	1900-01-01	2019-03-31	N
216	LG Dryer	590.00	2019-04-01	2019-05-31	N
216	LG Dryer	610.00	2019-09-01	2019-09-30	N
614	Google Phone	610.00	2019-06-01	2019-09-30	N
614	Google Phone	579.00	2019-10-01	2019-10-31	N
614	Google Phone	599.00	2019-03-01	2019-05-31	N



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

**Syntax** – SELECT \*

FROM customer\_dim

WHERE NOT cust\_age >18;

**Output :**

cust_id	cust_address	cust_age	effective_start_date	effective_end_date	current_ind
185057	335 Meadow St, Los Angeles, CA 90001	15	1900-01-01	9999-12-31	Y
225569	753 Adams St, Portland, ME 04101	15	1900-01-01	9999-12-31	Y
134924	295 Dogwood St, New York City, NY 10001	15	1900-01-01	9999-12-31	Y
218931	40 Jefferson St, Atlanta, GA 30301	15	1900-01-01	9999-12-31	Y
140361	169 Lake St, Boston, MA 02215	15	1900-01-01	9999-12-31	Y
188518	609 Highland St, San Francisco, CA 94016	15	1900-01-01	9999-12-31	Y
191233	801 Walnut St, New York City, NY 10001	15	1900-01-01	9999-12-31	Y

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

**Syntax** – SELECT \*

FROM product\_dim

WHERE product\_price > 500 OR product\_name= 'mac%';

**Output :**

product_id	product_name	product_price	effective_start_date	effective_end_date	current_ind
582	iPhone	635.00	2019-10-01	2019-11-30	N
582	iPhone	689.00	2019-06-01	2019-09-30	N
582	iPhone	649.00	2019-04-01	2019-05-31	N
582	iPhone	700.00	1900-01-01	2019-03-31	N
216	LG Dryer	610.00	2019-06-01	2019-06-30	N
216	LG Dryer	590.00	2019-10-01	2019-11-30	N
216	LG Dryer	600.00	2019-07-01	2019-08-31	N
216	LG Dryer	630.00	1900-01-01	2019-03-31	N
216	LG Dryer	590.00	2019-04-01	2019-05-31	N

**SOME** - TRUE if any of the subquery values meet the condition

**Syntax** – SELECT \*

FROM product\_dim

WHERE product\_price > SOME (

SELECT product\_price

FROM product\_dim

WHERE product\_name LIKE '%Book%'

);

**Output :**

product_id	product_name	product_price	effective_start_date	effective_end_date	current_ind
422	Macbook Pro Laptop	1800.00	1900-01-01	2019-03-31	N
422	Macbook Pro Laptop	1699.00	2019-04-01	2019-05-31	N
422	Macbook Pro Laptop	1699.00	2019-10-01	2019-10-31	N
422	Macbook Pro Laptop	1750.00	2019-11-01	2019-11-30	N
422	Macbook Pro Laptop	1800.00	2019-06-01	2019-09-30	N