15 Top SQL Commands for Beginners - A Ramp-Up Guide to Learn SQL Fast

SQL is the simplest, and probably the first database you've heard about.
You `SELECT` your SQL stars, `CREATE` a table here, `DROP` a table there.
Viewing the output of your new SQL table is not enough SQL commands learnt.
Where do you go from here?

If you're just getting started, it's hard to know what exactly to learn in SQL.

At SQLPlay, we've tried to make your experience a tad bit easier with a roadmap.
When you're building for the real world, we get a chance to visualize SQL.
In this ramp-up guide, you'll learn SQL with the objective of creating your own Recipe Database using the 15 most important SQL commands:

- How to Create a table in SQL
- How to Insert into SQL table
- How to Select SQL rows
- SQL Where
- SQL Order By
- SQL And, Or, NOT
- SQL Between
- Update table in SQL
- SQL Like
- SQL Group By
- SQL Delete
- SQL Not Equal
- SQL Count, Avg, Sum

- SQL Having
- SQL Joins

Some bonuses you'll learn if you read till the end:

- What is a Primary Key in SQL
- SQL Injections and how to prevent it

What is SQL?

Think of SQL database as a container for storing different items — your clothes, cutlery, currency, computer.

SQL stands for Structured Query Language.
Just as the name suggests, the database can store items not carelessly, but in a structured manner.

Having SQL is no less than having a managed inventory.
From clothes to actual wholesale products, it can take it all.
No sweat.

In SQL, you have a database
A database is a collection of tables.
A table further is divided into rows and columns.

Row → Record of an item (all details about a particular item, eg. its id, name and quantity)
Column → Field or Attribute of items (item_name describes what kind of items you're storing in your table, eg. is it a jacket, socks, or graphic tees?)

id	item_name	quantity
1	Jacket	4
2	Socks	2
3	Graphic Tees	15

This is the kind of data you can store, and also intelligently retrieve.
Let's see how to use SQL!

Why to use SQL or a Database?

Why not store data in text files?
Why even use a database?

In a database, your data is:

- searchable
- sortable
- constrained
- structured

You can’t do those in a text file.

Imagine a bank employee going through all the paper records to find out how much balance you have in your bank account.

bank-employee-smiling-and-checking-bank-balance

It is so tedious that every time you go and ask them, you wish you didn’t.

Having a database helps the bank employee to find out your bank balance in the fraction of a second, using just your account number and running a query on SQL.

> Just that, your banker doesn’t write SQL, the developers and DBAs (database administrator) do.

Let's *not* talk about systems that need big management.
Let's talk about simpler systems that make us happy — food recipes.

1. How to Create a Table in SQL

Creating a table in SQL is as simple as pie.

For example: How do we define a recipe?

We need to know what ingredients to use, in the right order and the time needed to cook the dish.

To create a table that you can store your recipes in, run this SQL query:

```
CREATE TABLE recipes
(id SERIAL PRIMARY KEY,
title varchar(40),
description varchar(120),
servings int,
cook_time int)
```

⚠️ For SQLite: replace `SERIAL PRIMARY KEY` with `INTEGER PRIMARY KEY AUTOINCREMENT`.

SQL needs instructions so here we tell it to create a table using `CREATE TABLE` command.
Now we need to our SQL table a name right after this command, example: **recipes**.
As we learnt earlier, SQL is all about structuring.
So we define what to store and what kind they are.

Let's learn about constraints

- Primary Key (PK) - It's the key which is unique in your table and can't be empty (null).
- Not Null - If you define a field as not null, it can't have empty values.
- Autoincrement - If you define a field with autoincrement, it will also need to be an integer which will start from value 1 and with subsequent row insertions increase the value of the field. Eg. 1,2,3,…
- Serial - Works just like autoincrement but has built-in data type of `BIGINT`

Let’s learn about data types

- Integer - Just the usual numbers. Eg. 1, 22, 66
- String - You can put multiple words and phrases. Eg. `'Bakery'`, `'https://sqlplay.net'`
- Varchar - Just like string but we limit with `VARCHAR(n)`, where **n** is the number of characters.

2. How to Insert into SQL Table

Now that we have created a table, we need to add our item details to it.

table-with-empty-plate-and-cutllery

> “***An empty table serveth none.”***

To insert your newfound recipes into the SQL table, run this command.

```
INSERT INTO recipes
VALUES (1, 'Spaghetti with Meatballs',
'Perfect pasta for busy weeknights with juicy meatballs', 2, 45)
```

```
INSERT INTO recipes
VALUES (2, 'Meat Bolognese',
'Perfect, cheesy meat and eggs for the soul', 4, 40)
```

```
INSERT INTO recipes
VALUES (3, 'Beef Wellington',
'Only healthy cow meat steak and butter. Nothing else', 3, 60)
```

```
INSERT INTO recipes
VALUES (4, 'Shakshuka',
'A truly one-pot meal with exotic veggies and five eggs', 2, 60)
```


It contains the values for the columns in order.

> Note that this method of inserting into SQL table needs all column values in order, including the painstaking ID.
> 1, 2, or 3, what do I know?

To avoid putting in IDs, field values in exact order every time, we can use the syntax which specifies the field names:

```
INSERT INTO recipes (title, description, servings, cook_time)
VALUES ('Spaghetti with Meatballs',
'Perfect pasta for busy weeknights with juicy meatballs', 2, 45)
```

You can skip some fields as well.
Example: If you don’t have an idea of how many people it will serve, you can simply skip it.

```
INSERT INTO recipes (title, description, cook_time)
VALUES ('Spaghetti with Meatballs',
'Perfect pasta for busy weeknights with juicy meatballs', 45)
```

3. How to get data from table — Select SQL rows

We just ran our fresh SQL INSERT command.
But, where is our data?
How do we verify that our data is safe in the table? Psst, it's not!

We check its contents by running a SQL `SELECT` command.

```
SELECT * from recipes;
```

Here `SELECT` does the job of, well, *selecting* or gathering items.
By using `*`, you select every column.
`FROM recipes` lets you select which table to view data *from*.

Select specific columns in SQL

In case you just want specific columns from your table, you can replace `*` with your column names.

You can also change the column name and put a shorter alias using the keyword `as` followed by your new alias name.
Here we made an alias of description as detail.

```
SELECT title, description as detail from recipes;
```

Output from the `SELECT` command:

title	detail
Spaghetti with Meatballs	Perfect pasta for busy weeknights with juicy meatballs

4. Filter rows — SQL Where

filter-coffee-to-filter-sql-rows

“Tring Tring” — 7 AM.
Snooze.
”Tring Tring” — 8AM.
Stop.

You wake up but don’t accept how 5 minutes of snooze turned into an hour.
No time for a shower.
Meeting at 9 AM.

Don’t miss your breakfast.
Grab a quick bite with this SQL filter query.

```
SELECT * from recipes WHERE cook_time <= 20;
```

This ensures you get a recipe using SQL `WHERE` that takes lesser than (<) or equal to (=) 20 minutes.
20 minutes.
No more.

To get the recipes which serves at least two people, try on this SQL query:

```
SELECT * from recipes WHERE servings >= 2;
```

With servings >= 2, you can ensure enough food for two.

5. Sorting rows A→Z — SQL Order By

Want to browse through recipes which take the least time to highest?
Sort them in order of ascending cook time using `ORDER BY`!

```
SELECT * from recipes ORDER BY cook_time ASC;
```

You can use `ORDER BY` command with `ASC` for ascending (lowest to highest) or `DESC` descending (highest to lowest).

6. Searching in SQL for text — SQL Like

Sometimes, after a soul-sucking 9-5 — you come to your dog at home.
Your dog isn't that well a chef.
Both bellies need to be fed though.

Today, search for everything whose description includes pasta.

Your job had you chained to a chair, not to the radius of a supermarket.
Now all you have in your pantry is … pasta.
Find a comforting recipe that'll help you get some dog food today.

```
SELECT * from recipes WHERE description LIKE '%pasta%';
```

% represents any number of characters.
Examples:

- `app%` will select the rows which begin with **app**, like **app**le, **app**s, **app**ly
- `%ap` will select the rows which end with **ap**, like m**ap**, l**ap**, c**ap**.
- `%uck%` will select the rows with **uck** in between, like l**uck**y, fire tr**uck**s, ch**uck**le.
- `f%y` will select the rows which begin with **f** and ends with **y**, like **f**irefl**y**, **f**ish**y**, **f**air**y**.

Limit search to specific characters

When you're sure of how many characters your word will have, use the `_` (underscore) wildcard selector.

```
SELECT * from recipes WHERE cook_time LIKE '4_';
```

This will give us all the recipes which has cook time of 40 to 49 minutes.
It's similar to SQL Between command in this use case.

How to use the `%` and `_` selectors together

Now that you've already learnt both the SQL LIKE wildcard selectors, let's see how we can use them together.

You were quite busy with your work, so you asked your son to enter some recipes for you.
Accidentally he misspelled **spaghetti** as **spaghetty** in some instances, so you're pulling your hair out trying to find the recipes.

Use `_` to find the words ending in **i** or **y** (or anything)!

```
SELECT * from recipes WHERE title LIKE 'spaghett_ %';
```

7. Logical operator — SQL AND, OR, NOT

SQL NOT

You did get up to make yourself pasta — but not enough to go to the pet store.
Your dog recently passed away … definitely not due to starvation.
As a token of forgiveness, you decide to never even think of meat.

Use the SQL NOT operator to filter out all the recipes which have 'meat' in their description.

```
SELECT * from recipes WHERE NOT description LIKE '%meat%';
```

SQL AND

You decide to go vegan.
Again, not for the dog.

Use the SQL AND operator to filter out multiple recipes which contain even the slightest amount of ‘meat’ and ‘egg’.

```
SELECT * from recipes WHERE NOT description LIKE '%meat%' AND '%egg%';
```

Combine AND with NOT.

Get yourself guilt-free pasta recipes, without the kill.

```
SELECT * from recipes WHERE NOT description LIKE '%meat%' AND description LIKE '%pasta%';
```

SQL OR

You’re vegan.
Your family isn’t.

Let’s get every recipe which contains either **meat** or **egg**.

```
SELECT * from recipes WHERE description LIKE '%meat%' OR '%egg%';
```

This is the output which you'll see:

id	title	description	servings	cook_time
1	Spaghetti with Meatballs	Perfect pasta for busy weeknights with juicy meatballs	2	45
2	Meat Bolognese	Perfect, cheesy meat and eggs for the soul	2	40
3	Beef Wellington	Only healthy cow meat steak and butter, nothing else	2	60

8. SQL Not Equal

Turning vegan?
Nuke those recipes which dare remind you of me…meat!

Learning the SQL Delete command will help in this.

```
DELETE FROM recipes WHERE title LIKE '%meat%';
```

If you're ***not*** too sure, the SQL NOT operator comes in handy and dandy.
Avoid all meat recipes with this clever SQL NOT hack:

```
SELECT * FROM recipes WHERE NOT title LIKE '%meat%';
```

9. SQL Between

It's Thanksgiving.
Nothing's changed.
Just this time, you're inviting your parents over.

The turkey’s warm - sitting in the oven. But not for long.
Your parents will be home in 2 hours.
Find a recipe that will help you quickly set the table — with the big turkey as the main dish.

```
SELECT * from recipes WHERE cook_time BETWEEN 20 AND 60;
```

This will give you all the recipes which have cook time ranging from 20 to 60. No more, no less.

10. Update rows in SQL table

You are happy after your Thanksgiving went great.

But then, you had a new finding — maybe the turkey could use more time in the oven.

Use the SQL Update command to update your cook time findings in the recipe:

```
UPDATE recipes
SET cook_time=60
WHERE title LIKE '%Turkey%'
```

With the `SET` command you can set the field value to an exact value.
Example: Set the cook_time to be 60 minutes.

> ⚠️ If you don't put a SQL WHERE clause, your query will update every single row of your table.

11. Remove items from your SQL table — SQL Delete

So I heard you wanted to delete a recipe that's too long?
I agree.
Let's delete those recipes which demand over 2 hours.

Here's your SQL Delete command that'll take care of that for you:

```
DELETE FROM recipes WHERE cook_time > 120;
```

12. SQL Count, Avg, Sum

Months flew by.
Dishes came by, recipes started pouring in on your recipes table with INSERTs.
You're an expert now.

But it makes you wonder —

> ***"How many recipes are actually there in my table?"***

With SQL Count command, you will know the exact number of items in your table. Run this:

```
SELECT COUNT(*) FROM recipes;
```

Want to know what's the average cook time for all your recipes? Use the SQL AVG command:

```
SELECT AVG(cook_time) FROM recipes;
```

13. Join Multiple Tables on Common Row — SQL Joins

Right now your SQL table for recipes only has the title, description, cook time and servings.
It doesn't tell you about the ingredients to be used in order and the quantity of each ingredient.

Let's create another SQL table — `ingredients`.

```
CREATE TABLE ingredients
(
  id SERIAL PRIMARY KEY,
  recipe_id BIGINT UNSIGNED,
  name varchar(30),
  quantity int,
  FOREIGN KEY (recipe_id) REFERENCES recipes(id)
)
```

⚠️ For SQLite: replace `SERIAL PRIMARY KEY` with `INTEGER PRIMARY KEY AUTOINCREMENT`.

Notice that `recipe_id` here is the common key between both our tables **recipes** and **ingredients**. This is called a foreign key — a link to join two tables in SQL.

Let's put some ingredients in our new ingredients table.

Insert rows using SQL with these commands:

```
INSERT INTO ingredients
VALUES (1, 1, 'spaghetti', 200)

INSERT INTO ingredients
VALUES (2, 1, 'tomato puree', 50)

INSERT INTO ingredients
VALUES (3, 1, 'white onions', 20)

INSERT INTO ingredients
VALUES (4, 1, 'salt', 4)

INSERT INTO ingredients
VALUES (5, 1, 'feta cheese', 30)
```

Add some more recipes into the ingredients table:

```
INSERT INTO ingredients
VALUES (6, 4, 'eggs', 5);

INSERT INTO ingredients
VALUES (7, 4, 'garlic', 2);

INSERT INTO ingredients
VALUES (8, 4, 'galangal', 1);

INSERT INTO ingredients
VALUES (9, 4, 'salt', 8);

INSERT INTO ingredients
VALUES (10, 4, 'white onions', 30);
```

Perform a SELECT operation to view all the ingredients.

```
SELECT * FROM ingredients
```

id	recipe_id	name	quantity
1	1	spaghetti	200
2	1	tomato puree	50
3	1	white onions	20
4	1	salt	4
5	1	feta cheese	30

The field `recipe_id` lets you know which recipe these ingredients are from.

Let's perform the simplest inner join on these two tables — recipes and ingredients.

```
SELECT *
FROM recipes
INNER JOIN ingredients ON
ingredients.recipe_id = recipes.id
```

In SQL Inner Join returns all rows from both tables where there is a match.

id	recipe_id	name	quantity	id	titl

1	1	spaghetti	200	1	Sp Me
2	1	tomato puree	50	1	Sp Me
3	1	white onions	20	1	Sp Me
4	1	salt	4	1	Sp Me
5	1	feta cheese	30	1	Sp Me

You got all the ingredients needed for your spaghetti recipe in a sweet, simple SQL table!
To-do lists, where you at?

14. Group rows together — SQL Group By

You have learnt so much, that you became a SQL chef.
You've got a $100,000 seed funding from Gordon Ramsay as he closely saw the recipes you entered in your database.
Now you feel confident enough to open a restaurant of your own, overlooking the sea.
Today, in fact, is your first shopping day for ingredients.
Tomorrow will be your opening day.

You would want to know how many ingredients you need, and their total quantity you need for all that's on your menu.

SQL Group by command helps you do *just* that.

```
SELECT name, COUNT(*) as 'needed in recipes',
SUM(quantity) as 'total quantity'
from ingredients
GROUP BY name
ORDER BY COUNT(*) DESC
```

Shopping just got a whole lot easier.
You don't have to check if salt is in the shopping list twice or thrice.
You get the exact quantity summed up in each row for each ingredient.

You can use `AS` like an alias for your columns to make it look pretty.

name	needed in recipes	total quantity
white onions	2	50
salt	2	12
tomato puree	1	50
spaghetti	1	200

garlic	1	2
galangal	1	1
feta cheese	1	30
eggs	1	5

15. Filter GROUP BY Query — SQL Having

```
SELECT name, COUNT(*) as 'needed in recipes',
SUM(quantity) as 'total quantity' from ingredients
GROUP BY name
HAVING name = 'salt'
```

Output of query:

name	needed in recipes	total quantity
salt	2	12

Notice that we used a new command — SQL Having.
The SQL Having command is used instead of SQL WHERE when we group our rows in SQL.

Where To Go Next?

When you're working with a database, there's all sorts of operations you can do to make your database management journey a bit easier.

But there are always:

- the crazy ones — SQL Create, SQL Insert, SQL Select
- the rebels — SQL Where, SQL AND, OR, NOT, SQL Update
- the misfits — SQL Order By, SQL Not Equal, SQL Delete, SQL Count, Avg, Sum
- the troublemakers — SQL Like, SQL Joins, SQL Between, SQL Group By, SQL Having

The SQL Having clause with the SQL Group By ones, they're not fond of SQL ORDER BYs.
They have no respect for SQL key constraints.

You can use them for your queries, disagree with the outputs and shake your head, or glorify them after correcting your query.
The only thing you can't do is ignore them.

While some may see them as the crazy SQL commands, we see genius:

Genius enough to achieve that employee raise calculation, and all together, the SQL commands that can change your life too.

If you’re that genius the world is looking for, don’t waste your time setting up a huge MySQL, SQLite or PostgreSQL database on your computer.

Get started on creating those magical SQL queries that can change your world by getting the SQL Play app that’s ready for your genius.