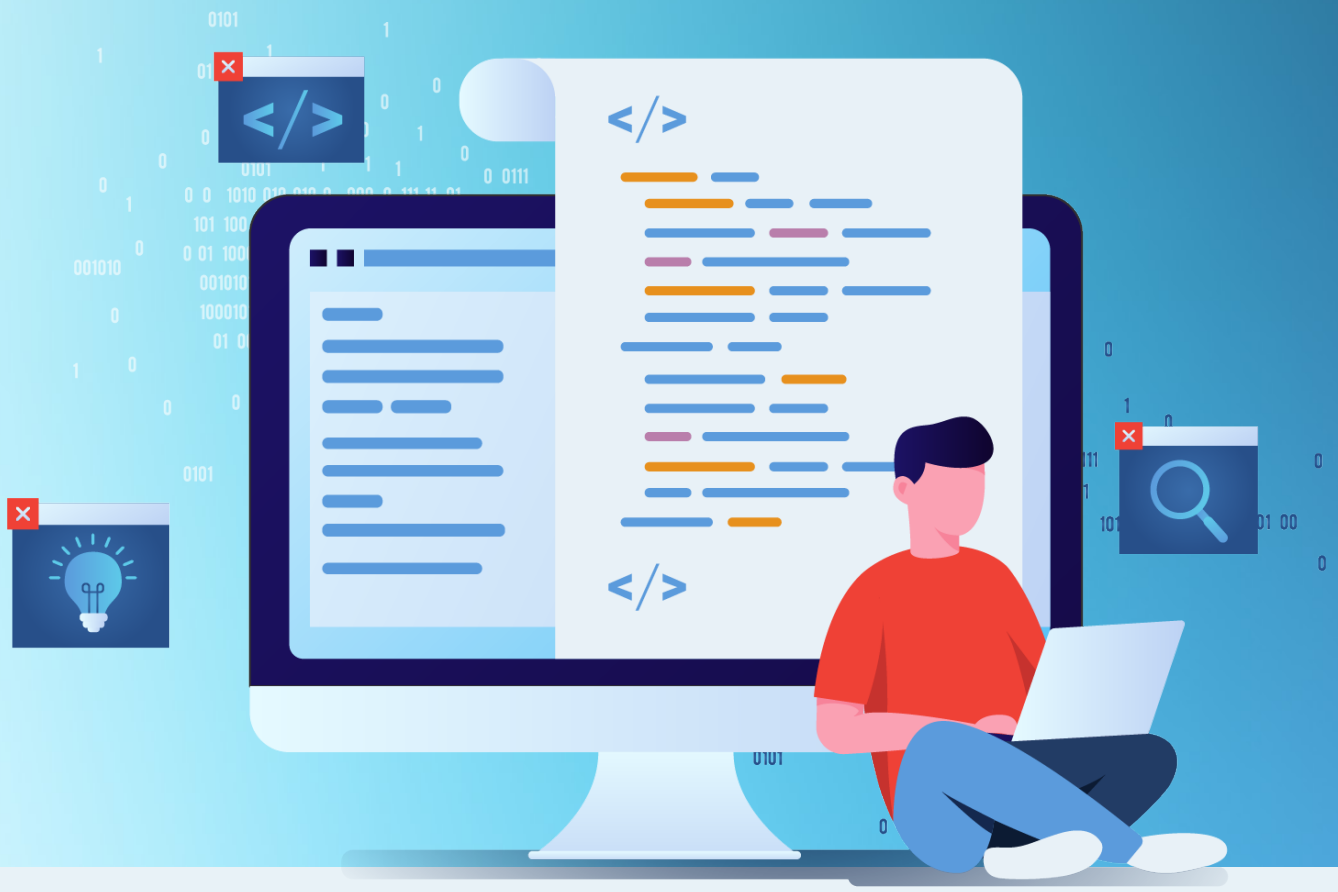


SQL Cheat Sheet

In this guide, you'll find a useful cheat sheet that documents some of the more commonly used elements of SQL, and even a few of the less common. Hopefully, it will help developers – both beginner and experienced level – become more proficient in their understanding of the SQL language.

Use this as a quick reference during development, a learning aid, or even print it out and bind it if you'd prefer (whatever works!).

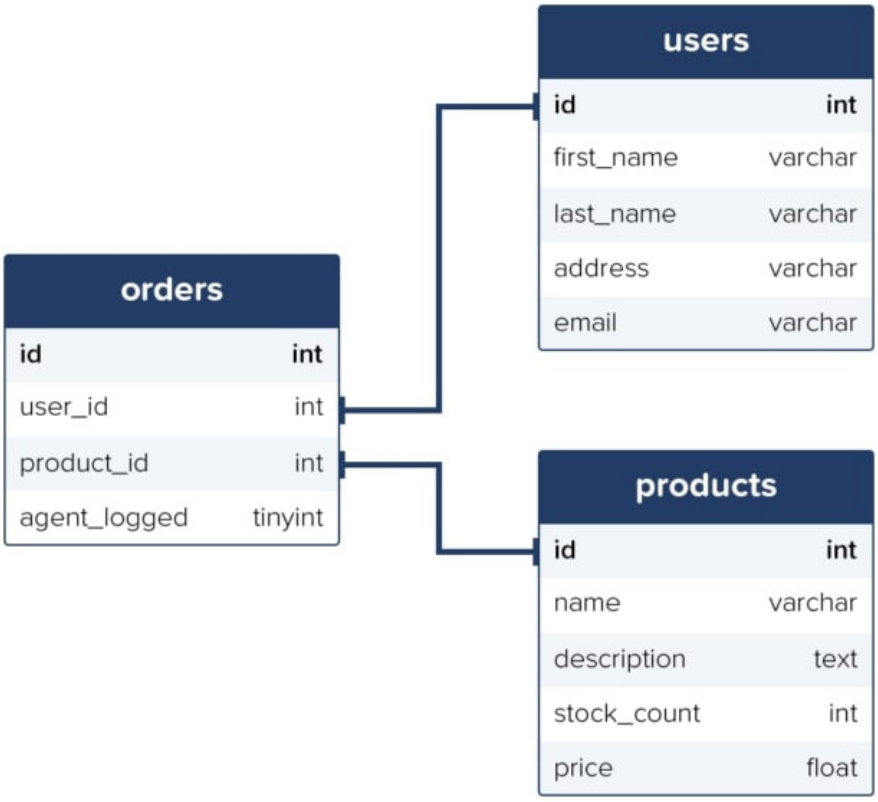
But before we get to the cheat sheet itself, for developers who may not be familiar with SQL, let's start with...

Table of Contents

What is SQL

SQL stands for Structured Query Language. It's the language of choice on today's web for storing, manipulating and retrieving data within relational databases. Most, if not all of the websites you visit will use it in some way, including this one.

Here's what a basic relational database looks like. This example in particular stores e-commerce information, specifically the products on sale, the users who buy them, and records of these orders which link these 2 entities.



Using SQL, you are able to interact with the database by writing queries, which when executed, return any results which meet its criteria.

Here's an example query:-

```
SELECT * FROM users;
```

Using this SELECT statement, the query selects all data from all columns in the user's table. It would then return data like the below, which is typically called a results set:-

users				
id	first_name	last_name	address	email
1	Luke	Harrison	1640 Rivers...	luke@lukeh...
2	Heather	Reynolds	742 Evergr...	heza@hot...
3	Simon	Clarkson	7 Peterbou...	smr@yaho...
4	Claire	Simpson	15 Musgra...	claire@hot...
5	Oliver	Harrison	1640 Rivers...	oliver@ya...
6	James	Gilbert	598 Firshil...	jgill@appl...
7	Michael	Johnson	12 Redmire...	mj@yahoo...
8	Thomas	Smith	342 Brown...	t.smith@al...
9	Robyn	Gilbert	598 Firshil...	summer@d...
10	Bryony	Brown	165 South...	bryony@h...
11	Tester	Jester	123 Fake S...	test@luke...

If we were to replace the asterisk wildcard character (*) with specific column names instead, only the data from these columns would be returned from the query.

```
SELECT first_name, last_name FROM users;
```

users	
first_name	last_name
Luke	Harrison
Heather	Reynolds
Simon	Clarkson
Claire	Simpson
Oliver	Harrison
James	Gilbert
Michael	Johnson
Thomas	Smith
Robyn	Gilbert
Bryony	Brown

We can add a bit of complexity to a standard SELECT statement by adding a WHERE clause, which allows you to filter what gets returned.

```
SELECT * FROM products WHERE stock_count <= 10 ORDER BY stock_count ASC;
```

This query would return all data from the products table with a stock_count value of less than 10 in its results set. The use of the ORDER BY keyword means the results will be ordered using the stock_count column, lowest values to highest.

products				
id	name	description	stock_count	price
192	Carton Do...	Whether y...	0	14.99
23	Cardboar...	Declutter...	1	3.49
3	SmartMo...	NULL	1	24.99
32	TripLast 33...	Cost effec...	4	16.50
875	A4 Storag...	Dimensio...	5	4.99
456	Pack of 50...	Date first a...	5	12.99
341	Set of 2 S...	5 year gua...	8	4.99
67	Large Car...	Need som...	10	12.99
196	10 X Plasti...	Pack of 10...	10	15.99
310	StorePac 5...	High qual...	10	9.99

Using the INSERT INTO statement, we can add new data to a table. Here's a basic example adding a new user to the users table:-

```
INSERT INTO users (first_name, last_name, address, email)
VALUES ('Tester', 'Jester', '123 Fake Street, Sheffield, United
Kingdom', 'test@lukeharrison.dev');
```

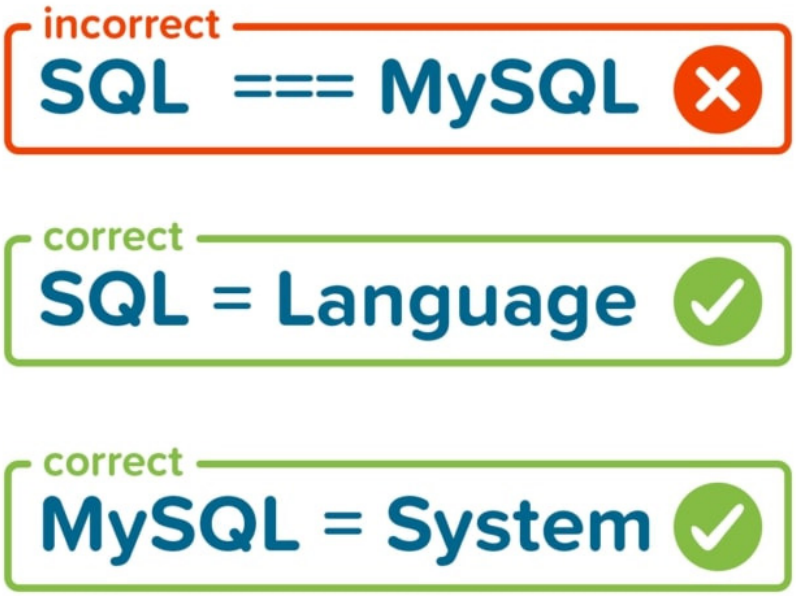
Then if you were to rerun the query to return all data from the user's table, the results set would look like this:

users				
id	first_name	last_name	address	email
1	Luke	Harrison	1640 Rivers...	luke@lukeh...
2	Heather	Reynolds	742 Evergr...	heza@hot...
3	Simon	Clarkson	7 Peterbou...	smr@yaho...
4	Claire	Simpson	15 Musgra...	claire@hot...
5	Oliver	Harrison	1640 Rivers...	oliver@ya...
6	James	Gilbert	598 Firshil...	jgill@appl...
7	Michael	Johnson	12 Redmire...	mj@yahoo...
8	Thomas	Smith	342 Brown...	t.smith@al...
9	Robyn	Gilbert	598 Firshil...	summer@d...
10	Bryony	Brown	165 South...	bryony@h...
11	Tester	Jester	123 Fake S...	test@luke...

Of course, these examples demonstrate only a very small selection of what the SQL language is capable of.

SQL vs MySQL

You may have heard of MySQL before. It's important that you don't confuse this with SQL itself, as there's a clear difference.



SQL is the language. It outlines syntax that allows you to write queries that manage relational databases. Nothing more.

MySQL meanwhile is a database system that runs on a server. It implements the SQL language, allowing you to write queries using its syntax to manage MySQL databases.

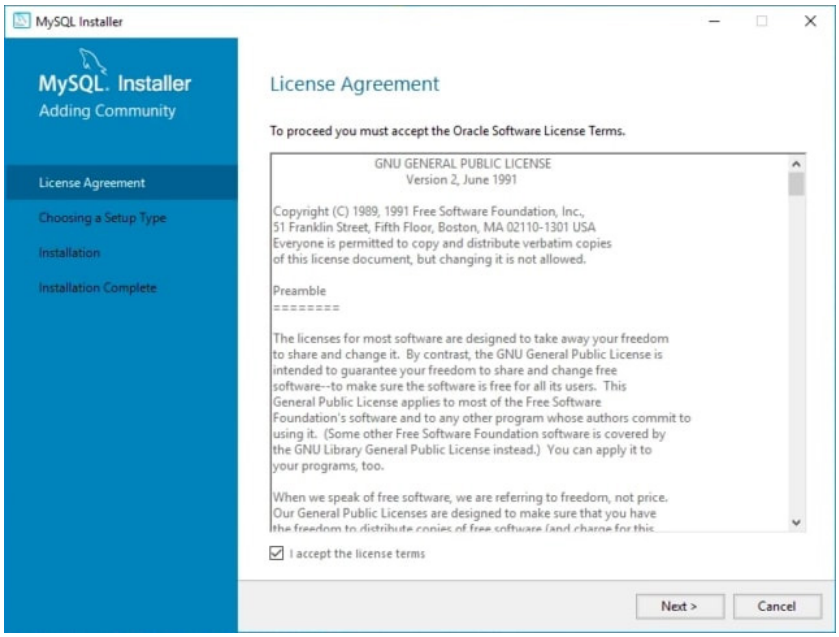
In addition to MySQL, there are other systems that implement SQL. Some of the more popular ones include:

- PostgreSQL
- SQLite
- Oracle Database
- Microsoft SQL Server

Installing MySQL

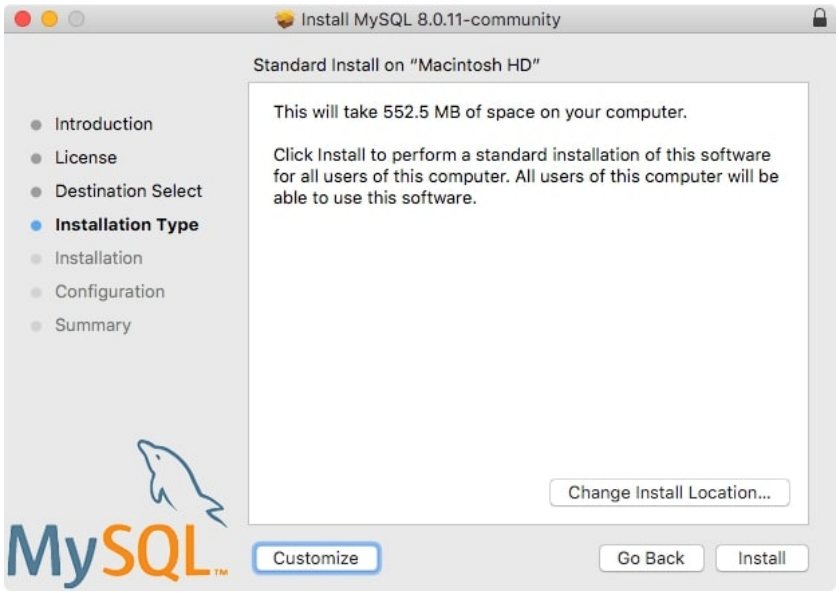
Windows

The recommended way to install MySQL on Windows is by using the installer you can download from the MySQL website.



MacOS

On macOS, the recommended way to install MySQL is using native packages, which sounds a lot more complicated than it actually is. Essentially, it also involves just downloading an installer.



Alternatively, If you prefer to use package managers such as Homebrew, you can install MySQL like so:

```
brew install mysql
```

Whilst if you need to install the older MySQL version 5.7, which is still widely used today on the web, you can:

```
brew install mysql@5.7
```

Using MySQL

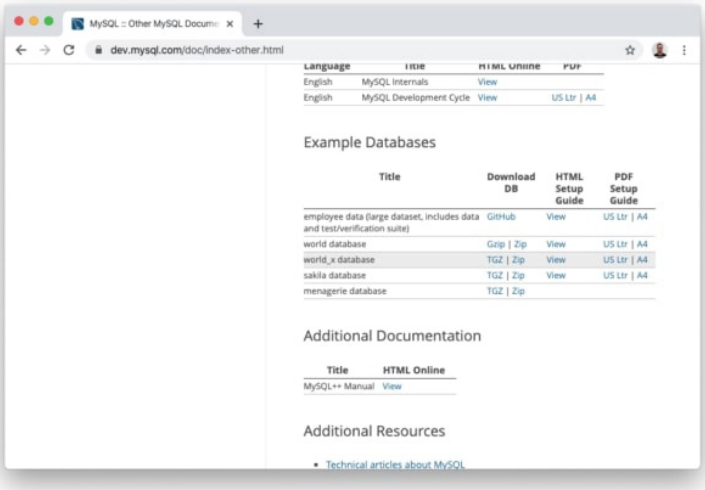
With MySQL now installed on your system, to get up and going as quickly as possible writing SQL queries, it's recommended that you use an SQL management application to make managing your databases a much simpler, easier process.

There are lots of apps to choose from which largely do the same job, so it's down to your own personal preference on which one to use:

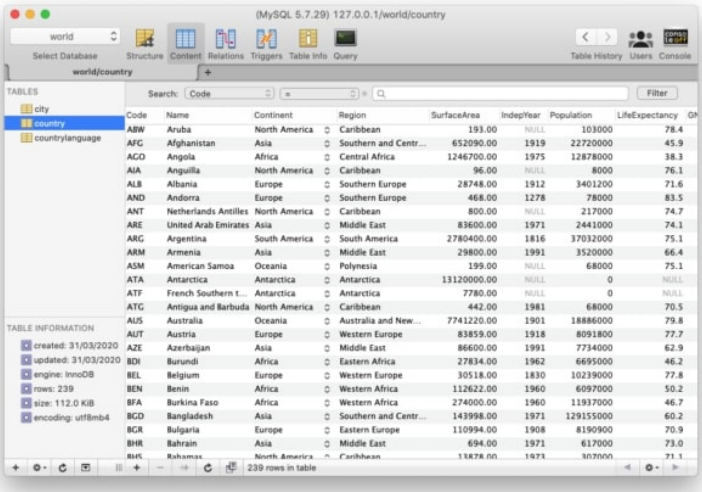
- MySQL Workbench is developed by Oracle, the owner of MySQL.
- HeidiSQL (Recommended Windows) is a free, open-source app for Windows. For macOS and Linux users, Wine is first required as a prerequisite.
- phpMyAdmin is a very popular alternative that operates in the web browser.
- Sequel Pro (Recommended macOS) is a macOS' only alternative and our favorite thanks to its clear and easy to use interface.

When you're ready to start writing your own SQL queries, rather than spending time creating your own database, consider importing dummy data instead.

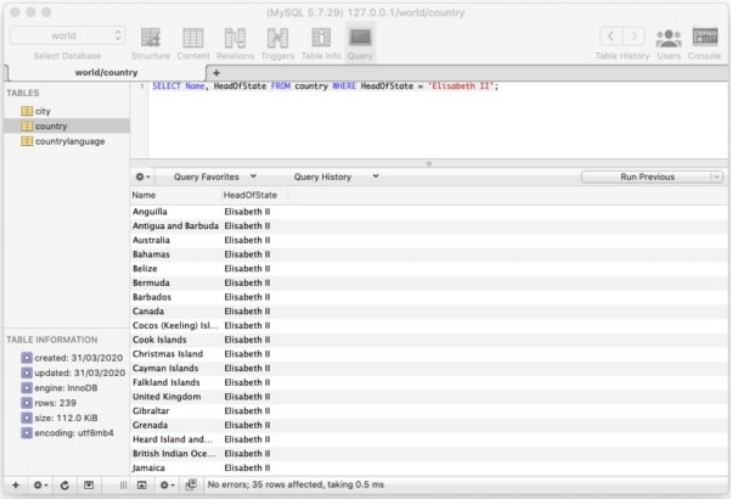
The MySQL website provides a number of dummy databases that you can download free of charge and then import into your SQL app.



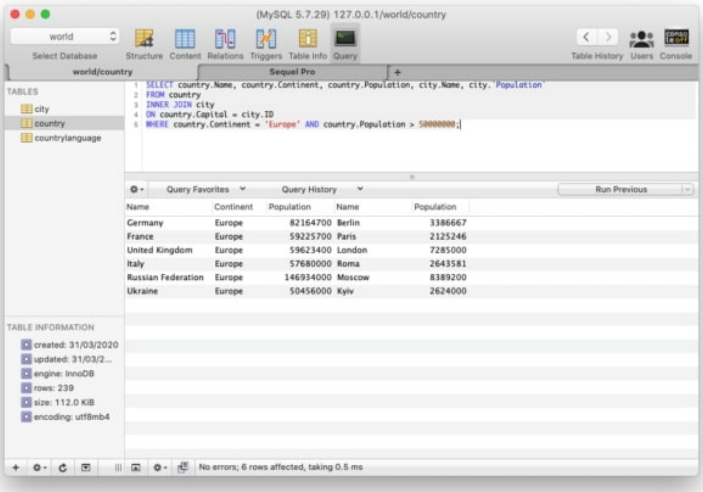
Our favorite of these is the world database, which provides some interesting data to practice writing SQL queries for. Here's a screenshot of its country table within Sequel Pro.



This example query returns all countries with Queen Elizabeth II as their head of state.



Whilst this one returns all European countries with a population of over 50million along with their capital city and its population.



Cheat Sheet

Keywords

A collection of keywords used in SQL statements, a description, and where appropriate an example. Some of the more advanced keywords have their own dedicated section later in the cheat sheet.

Where MySQL is mentioned next to an example, this means this example is only applicable to MySQL databases (as opposed to any other database system).

SQL Keywords	
Keyword	**Description**

ADD	Adds a new column to an existing table.
	Example: Adds a new column named 'email_address' to a table named 'users'.
	`ALTER TABLE users`
	`ADD email_address varchar(255);`

ADD CONSTRAINT	It creates a new constraint on an existing table, which is used to specify rules for any data in the table.
	Example: Adds a new PRIMARY KEY constraint named 'user' on columns ID and SURNAME.
	`ALTER TABLE users`
	`ADD CONSTRAINT user PRIMARY KEY (ID, SURNAME);`

ALTER TABLE	Adds, deletes or edits columns in a table. It can also be used to add and delete constraints in a table, as per the above.
	Example: Adds a new boolean column called 'approved' to a table named 'deals'.
	`ALTER TABLE deals`
	`ADD approved boolean;`
	Example 2: Deletes the 'approved' column from the 'deals' table
	`ALTER TABLE deals`
	`DROP COLUMN approved;`

SQL Keywords	
Keyword	**Description**

ALTER COLUMN	Changes the data type of a table's column.
	Example: In the 'users' table, make the column 'incept_date' into a 'datetime' type.
	`ALTER TABLE users`
	`ALTER COLUMN incept_date datetime;`

ALL	Returns true if all of the subquery values meet the passed condition.
	Example: Returns the users with a higher number of tasks than the user with the highest number of tasks in the HR department (id 2)
	`SELECT first_name, surname, tasks_no`
	`FROM users`
	`WHERE tasks_no > ALL (SELECT tasks FROM user WHERE department_id = 2);`

AND	Used to join separate conditions within a WHERE clause.
	Example: Returns events located in London, United Kingdom
	`SELECT * FROM events`
	`WHERE host_country='United Kingdom' AND host_city='London';`

ANY	Returns true if any of the subquery values meet the given condition.
	Example: Returns products from the products table which have received orders – stored in the orders table – with a quantity of more than 5.
	`SELECT name`
	`FROM products`
	`WHERE productId = ANY (SELECT productId FROM orders WHERE quantity > 5);`

AS	Renames a table or column with an alias value which only exists for the duration of the query.
	Example: Aliases north_east_user_subscriptions column
	`SELECT north_east_user_subscriptions AS ne_subs`
	`FROM users`
	`WHERE ne_subs > 5;`

ASC	Used with ORDER BY to return the data in ascending order. **Example**: Apples, Bananas, Peaches, Raddish

SQL Keywords	
Keyword	**Description**

BETWEEN	Selects values within the given range.
	
	Example 1: Selects stock with a quantity between 100 and 150.
	
	`SELECT * FROM stock`
	`WHERE quantity BETWEEN 100 AND 150;`
	
	Example 2: Selects stock with a quantity NOT between 100 and 150. Alternatively, using the NOT keyword here reverses the logic and selects values outside the given range.
	
	`SELECT * FROM stock`
	`WHERE quantity NOT BETWEEN 100 AND 150;`

CASE	Change query output depending on conditions.
	Example: Returns users and their subscriptions, along with a new column called activity_levels that makes a judgement based on the number of subscriptions.
	
	`SELECT first_name, surname, subscriptions`
	`CASE WHEN subscriptions > 10 THEN 'Very active'`
	`WHEN Quantity BETWEEN 3 AND 10 THEN 'Active'`
	`ELSE 'Inactive'`
	`END AS activity_levels`
	`FROM users;`

CHECK	Adds a constraint that limits the value which can be added to a column.
	Example 1 (MySQL): Makes sure any users added to the users table are 18 or over.
	
	`CREATE TABLE users (`
	`first_name varchar(255),`
	`age int,`
	`CHECK (age>=18)`
	`);`
	
	Example 2 (MySQL): Adds a check after the table has already been created.
	
	`ALTER TABLE users`
	`ADD CHECK (age>=18);`

SQL Keywords	
Keyword	**Description**
CREATE DATABASE	Creates a new database. **Example**: Creates a new database named 'websitesetup'. `CREATE DATABASE websitesetup;`

CREATE TABLE	Creates a new table .
	Example: Creates a new table called 'users' in the 'websitesetup' database.
	
	`CREATE TABLE users (`
	`id int,`
	`first_name varchar(255),`
	`surname varchar(255),`
	`address varchar(255),`
	`contact_number int`
	`);`

DEFAULT	Sets a default value for a column;
	Example 1 (MySQL): Creates a new table called Products which has a name column with a default value of 'Placeholder Name' and an available_ from column with a default value of today's date.
	
	`CREATE TABLE products (`
	`id int,`
	`name varchar(255) DEFAULT 'Placeholder Name',`
	`available_from date DEFAULT GETDATE()`
	`);`
	
	Example 2 (MySQL): The same as above, but editing an existing table.
	
	`ALTER TABLE products`
	`ALTER name SET DEFAULT 'Placeholder Name',`
	`ALTER available_from SET DEFAULT GETDATE();`

DELETE	Delete data from a table. **Example**: Removes a user with a user_id of 674. `DELETE FROM users WHERE user_id = 674;`
DESC	Used with ORDER BY to return the data in descending order. **Example**: Raddish, Peaches, Bananas, Apples

SQL Keywords	
Keyword	**Description**

DROP COLUMN	Deletes a column from a table.
	Example: Removes the first_name column from the users table.
	`ALTER TABLE users`
	`DROP COLUMN first_name`

DROP DATABASE	Deletes the entire database. **Example**: Deletes a database named 'websitesetup'. `DROP DATABASE websitesetup;`

DROP DEFAULT	Removes a default value for a column.
	Example (MySQL): Removes the default value from the 'name' column in the 'products' table.
	`ALTER TABLE products`
	`ALTER COLUMN name DROP DEFAULT;`

DROP TABLE	Deletes a table from a database. **Example**: Removes the users table. `DROP TABLE users;`

EXISTS	Checks for the existence of any record within the subquery, returning true if one or more records are returned.
	Example: Lists any dealerships with a deal finance percentage less than 10.
	`SELECT dealership_name`
	`FROM dealerships`
	`WHERE EXISTS (SELECT deal_name FROM deals WHERE dealership_id = deals.dealership_id AND finance_percentage < 10);`

FROM	Specifies which table to select or delete data from.
	Example: Selects data from the users table.
	`SELECT area_manager`
	`FROM area_managers`
	`WHERE EXISTS (SELECT ProductName FROM Products WHERE area_manager_id = deals.area_manager_id AND Price < 20);`

SQL Keywords	
Keyword	**Description**

IN	Used alongside a WHERE clause as a shorthand for multiple OR conditions. So instead of:
	`SELECT * FROM users WHERE country = 'USA' OR country = 'United Kingdom' OR country = 'Russia' OR country = 'Australia';`
	You can use:
	`SELECT * FROM users WHERE country IN ('USA', 'United Kingdom', 'Russia', 'Australia');`

INSERT INTO	Add new rows to a table. **Example**: Adds a new vehicle. `INSERT INTO cars (make, model, mileage, year) VALUES ('Audi', 'A3', 30000, 2016);`
IS NULL	Tests for empty (NULL) values. **Example**: Returns users that haven't given a contact number. `SELECT * FROM users WHERE contact_number IS NULL;`
IS NOT NULL	The reverse of NULL. Tests for values that aren't empty / NULL.
LIKE	Returns true if the operand value matches a pattern. **Example**: Returns true if the user's first_name ends with 'son'. `SELECT * FROM users WHERE first_name LIKE '%son';`
NOT	Returns true if a record DOESN'T meet the condition. **Example**: Returns true if the user's first_name doesn't end with 'son'. `SELECT * FROM users WHERE first_name NOT LIKE '%son';`
OR	Used alongside WHERE to include data when either condition is true. **Example**: Returns users that live in either Sheffield or Manchester. `SELECT * FROM users WHERE city = 'Sheffield' OR 'Manchester';`

SQL Keywords	
Keyword	**Description**

ORDER BY	Used to sort the result data in ascending (default) or descending order through the use of ASC or DESC keywords.
	Example: Returns countries in alphabetical order.
	`SELECT * FROM countries`
	`ORDER BY name;`

ROWNUM	Returns results where the row number meets the passed condition.
	Example: Returns the top 10 countries from the countries table.
	`SELECT * FROM countries`
	`WHERE ROWNUM <= 10;`

SELECT	Used to select data from a database, which is then returned in a results set.
	Example 1: Selects all columns from all users.
	`SELECT * FROM users;`
	`Example 2: Selects the first_name and surname columns from all users.xx`
	`SELECT first_name, surname FROM users;`

SELECT DISTINCT	Sames as SELECT, except duplicate values are excluded.
	Example: Creates a backup table using data from the users table.
	`SELECT * INTO usersBackup2020`
	`FROM users;`

SELECT INTO	Copies data from one table and inserts it into another. **Example**: Returns all countries from the users table, removing any duplicate values (which would be highly likely) `SELECT DISTINCT country from users;`
SELECT TOP	Allows you to return a set number of records to return from a table. **Example**: Returns the top 3 cars from the cars table. `SELECT TOP 3 * FROM cars;`

SQL Keywords	
Keyword	**Description**

SET	Used alongside UPDATE to update existing data in a table.
	Example: Updates the value and quantity values for an order with an id of 642 in the orders table.
	`UPDATE orders`
	`SET value = 19.49, quantity = 2`
	`WHERE id = 642;`

SOME	Identical to ANY.
TOP	Used alongside SELECT to return a set number of records from a table. **Example**: Returns the top 5 users from the users table. `SELECT TOP 5 * FROM users;`
TRUNCATE TABLE	Similar to DROP, but instead of deleting the table and its data, this deletes only the data. **Example**: Empties the sessions table, but leaves the table itself intact. `TRUNCATE TABLE sessions;`

UNION	Combines the results from 2 or more SELECT statements and returns only distinct values.
	Example: Returns the cities from the events and subscribers tables.
	`SELECT city FROM events`
	`UNION`
	`SELECT city from subscribers;`

UNION ALL	The same as UNION, but includes duplicate values.

SQL Keywords	
Keyword	**Description**

UNIQUE	This constraint ensures all values in a column are unique.
	Example 1 (MySQL): Adds a unique constraint to the id column when creating a new users table.
	`CREATE TABLE users (`
	`id int NOT NULL,`
	`name varchar(255) NOT NULL,`
	`UNIQUE (id)`
	`);`
	Example 2 (MySQL): Alters an existing column to add a UNIQUE constraint.
	`ALTER TABLE users`
	`ADD UNIQUE (id);`

UPDATE	Updates existing data in a table.
	Example: Updates the mileage and serviceDue values for a vehicle with an id of 45 in the cars table.
	`UPDATE cars`
	`SET mileage = 23500, serviceDue = 0`
	`WHERE id = 45;`

VALUES	Used alongside the INSERT INTO keyword to add new values to a table.
	Example: Adds a new car to the cars table.
	`INSERT INTO cars (name, model, year)`
	`VALUES ('Ford', 'Fiesta', 2010);`

WHERE	Filters results to only include data which meets the given condition.
	Example: Returns orders with a quantity of more than 1 item.
	`SELECT * FROM orders`
	`WHERE quantity > 1;`

Comments

Comments allow you to explain sections of your SQL statements, or to comment out code and prevent its execution.

In SQL, there are 2 types of comments, single line and multiline.

Single Line Comments

Single line comments start with –. Any text after these 2 characters to the end of the line will be ignored.

```
-- My Select query
SELECT * FROM users;
```

Multiline Comments

Multiline comments start with /* and end with */. They stretch across multiple lines until the closing characters have been found.

```
/*
This is my select query.
It grabs all rows of data from the users table
*/
SELECT * FROM users;

/*
This is another select query, which I don't want to execute yet

SELECT * FROM tasks;
*/
```

MySQL Data Types

When creating a new table or editing an existing one, you must specify the type of data that each column accepts.

In the below example, data passed to the id column must be an int, whilst the first_name column has a VARCHAR data type with a maximum of 255 characters.

```
CREATE TABLE users (
    id int,
    first_name varchar(255)
);
```

String Data Types

String Data Types	
Data Type	**Description**
CHAR(size)	Fixed length string which can contain letters, numbers and special characters. The size parameter sets the maximum string length, from 0 – 255 with a default of 1.
VARCHAR(size)	Variable length string similar to CHAR(), but with a maximum string length range from 0 to 65535.
BINARY(size)	Similar to CHAR() but stores binary byte strings.
VARBINARY(size)	Similar to VARCHAR() but for binary byte strings.
TINYBLOB	Holds Binary Large Objects (BLOBs) with a max length of 255 bytes.
TINYTEXT	Holds a string with a maximum length of 255 characters. Use VARCHAR() instead, as it's fetched much faster.
TEXT(size)	Holds a string with a maximum length of 65535 bytes. Again, better to use VARCHAR().
BLOB(size)	Holds Binary Large Objects (BLOBs) with a max length of 65535 bytes.
MEDIUMTEXT	Holds a string with a maximum length of 16,777,215 characters.

String Data Types	
Data Type	**Description**
MEDIUMBLOB	Holds Binary Large Objects (BLOBs) with a max length of 16,777,215 bytes.
LONGTEXT	Holds a string with a maximum length of 4,294,967,295 characters.
LONGBLOB	Holds Binary Large Objects (BLOBs) with a max length of 4,294,967,295 bytes.
ENUM(a, b, c, etc...)	A string object that only has one value, which is chosen from a list of values which you define, up to a maximum of 65535 values. If a value is added which isn't on this list, it's replaced with a blank value instead. Think of ENUM being similar to HTML radio boxes in this regard. `CREATE TABLE tshirts (color ENUM('red', 'green', 'blue', 'yellow', 'purple'));`
SET(a, b, c, etc...)	A string object that can have 0 or more values, which is chosen from a list of values which you define, up to a maximum of 64 values. Think of SET being similar to HTML checkboxes in this regard.

Numeric Data Types

Numeric Data Types	
Data Type	**Description**
BIT(size)	A bit-value type with a default of 1. The allowed number of bits in a value is set via the size parameter, which can hold values from 1 to 64.
TINYINT(size)	A very small integer with a signed range of -128 to 127, and an unsigned range of 0 to 255. Here, the size parameter specifies the maximum allowed display width, which is 255.
BOOL	Essentially a quick way of setting the column to TINYINT with a size of 1. 0 is considered false, whilst 1 is considered true.
BOOLEAN	Same as BOOL.
SMALLINT(size)	A small integer with a signed range of -32768 to 32767, and an unsigned range from 0 to 65535. Here, the size parameter specifies the maximum allowed display width, which is 255.

Numeric Data Types	
Data Type	**Description**
MEDIUMINT(size)	A medium integer with a signed range of -8388608 to 8388607, and an unsigned range from 0 to 16777215. Here, the size parameter specifies the maximum allowed display width, which is 255.
INT(size)	A medium integer with a signed range of -2147483648 to 2147483647, and an unsigned range from 0 to 4294967295. Here, the size parameter specifies the maximum allowed display width, which is 255.
INTEGER(size)	Same as INT.
BIGINT(size)	A medium integer with a signed range of -9223372036854775808 to 9223372036854775807, and an unsigned range from 0 to 18446744073709551615. Here, the size parameter specifies the maximum allowed display width, which is 255.
FLOAT(p)	A floating point number value. If the precision (p) parameter is between 0 to 24, then the data type is set to FLOAT(), whilst if its from 25 to 53, the data type is set to DOUBLE(). This behaviour is to make the storage of values more efficient.
DOUBLE(size, d)	A floating point number value where the total digits are set by the size parameter, and the number of digits after the decimal point is set by the d parameter.
DECIMAL(size, d)	An exact fixed point number where the total number of digits is set by the size parameters, and the total number of digits after the decimal point is set by the d parameter. For size, the maximum number is 65 and the default is 10, whilst for d, the maximum number is 30 and the default is 10.
DEC(size, d)	Same as DECIMAL.

Date / Time Data Types

Date / Time Data Types	
Data Type	**Description**
DATE	A simple date in YYYY-MM-DD format, with a supported range from '1000-01-01' to '9999-12-31'.
DATETIME(fsp)	A date time in YYYY-MM-DD hh:mm:ss format, with a supported range from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'. By adding DEFAULT and ON UPDATE to the column definition, it automatically sets to the current date/time.
TIMESTAMP(fsp)	A Unix Timestamp, which is a value relative to the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). This has a supported range from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC. By adding DEFAULT CURRENT_TIMESTAMP and ON UPDATE CURRENT TIMESTAMP to the column definition, it automatically sets to current date/time.
TIME(fsp)	A time in hh:mm:ss format, with a supported range from '-838:59:59' to '838:59:59'.
YEAR	A year, with a supported range of '1901' to '2155'.

Operators

Arithmetic Operators

Arithmetic Operators	
Operator	**Description**
+	Add
-	Subtract
*	Multiply
/	Divide
%	Modulo

Bitwise Operator

Bitwise Operator	
Operator	**Description**
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR

Comparison Operators

Comparison Operators	
Operator	**Description**
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to

Compound Operators

Compound Operators	
Operator	**Description**
+=	Add equals
-=	Subtract equals
*=	Multiply equals
/=	Divide equals
%=	Modulo equals
&=	Bitwise AND equals
^-=	Bitwise exclusive equals
|*=	Bitwise OR equals

Functions

String Functions

String Functions	
Name	**Description**
ASCII	Returns the equivalent ASCII value for a specific character.
CHAR_LENGTH	Returns the character length of a string.
CHARACTER_ LENGTH	Same as CHAR_LENGTH.
CONCAT	Adds expressions together, with a minimum of 2.
CONCAT_WS	Adds expressions together, but with a separator between each value.
FIELD	Returns an index value relative to the position of a value within a list of values.
FIND IN SET	Returns the position of a string in a list of strings.
FORMAT	When passed a number, returns that number formatted to include commas (eg 3,400,000).
INSERT	Allows you to insert one string into another at a certain point, for a certain number of characters.
INSTR	Returns the position of the first time one string appears within another.
LCASE	Convert a string to lowercase.
LEFT	Starting from the left, extract the given number of characters from a string and return them as another.
LENGTH	Returns the length of a string, but in bytes.
LOCATE	Returns the first occurrence of one string within another,
LOWER	Same as LCASE.
LPAD	Left pads one string with another, to a specific length.
LTRIM	Remove any leading spaces from the given string.

String Functions	
Name	**Description**
MID	Extracts one string from another, starting from any position.
POSITION	Returns the position of the first time one substring appears within another.
REPEAT	Allows you to repeat a string
REPLACE	Allows you to replace any instances of a substring within a string, with a new substring.
REVERSE	Reverses the string.
RIGHT	Starting from the right, extract the given number of characters from a string and return them as another.
RPAD	Right pads one string with another, to a specific length.
RTRIM	Removes any trailing spaces from the given string.
SPACE	Returns a string full of spaces equal to the amount you pass it.
STRCMP	Compares 2 strings for differences
SUBSTR	Extracts one substring from another, starting from any position.
SUBSTRING	Same as SUBSTR
SUBSTRING_INDEX	Returns a substring from a string before the passed substring is found the number of times equals to the passed number.
TRIM	Removes trailing and leading spaces from the given string. Same as if you were to run LTRIM and RTRIM together.
UCASE	Convert a string to uppercase.
UPPER	Same as UCASE.

Numeric Functions

Numeric Functions	
Name	**Description**
ABS	Returns the absolute value of the given number.
ACOS	Returns the arc cosine of the given number.
ASIN	Returns the arc sine of the given number.
ATAN	Returns the arc tangent of one or 2 given numbers.
ATAN2	Return the arc tangent of 2 given numbers.
AVG	Returns the average value of the given expression.
CEIL	Returns the closest whole number (integer) upwards from a given decimal point number.
CEILING	Same as CEIL.
COS	Returns the cosine of a given number.
COT	Returns the cotangent of a given number.
COUNT	Returns the amount of records that are returned by a SELECT query.
DEGREES	Converts a radians value to degrees.
DIV	Allows you to divide integers.
EXP	Returns e to the power of the given number.
FLOOR	Returns the closest whole number (integer) downwards from a given decimal point number.
GREATEST	Returns the highest value in a list of arguments.
LEAST	Returns the smallest value in a list of arguments.
LN	Returns the natural logarithm of the given number
LOG	Returns the natural logarithm of the given number, or the logarithm of the given number to the given base
LOG10	Does the same as LOG, but to base 10.

Numeric Functions	
Name	**Description**
LOG2	Does the same as LOG, but to base 2.
MAX	Returns the highest value from a set of values.
MIN	Returns the lowest value from a set of values.
MOD	Returns the remainder of the given number divided by the other given number.
PI	Returns PI.
POW	Returns the value of the given number raised to the power of the other given number.
POWER	Same as POW.
RADIANS	Converts a degrees value to radians.
RAND	Returns a random number.
ROUND	Round the given number to the given amount of decimal places.
SIGN	Returns the sign of the given number.
SIN	Returns the sine of the given number.
SQRT	Returns the square root of the given number.
SUM	Returns the value of the given set of values combined.
TAN	Returns the tangent of the given number.
TRUNCATE	Returns a number truncated to the given number of decimal places.

Date Functions

Numeric Functions	
Name	**Description**
ADDDATE	Add a date interval (eg: 10 DAY) to a date (eg: 20/01/20) and return the result (eg: 20/01/30).
ADDTIME	Add a time interval (eg: 02:00) to a time or datetime (05:00) and return the result (07:00).
CURDATE	Get the current date.
CURRENT_DATE	Same as CURDATE.
CURRENT_TIME	Get the current time.
CURRENT_ TIMESTAMP	Get the current date and time.
CURTIME	Same as CURRENT_TIME.
DATE	Extracts the date from a datetime expression.
DATEDIFF	Returns the number of days between the 2 given dates.
DATE_ADD	Same as ADDDATE.
DATE_FORMAT	Formats the date to the given pattern.
DATE_SUB	Subtract a date interval (eg: 10 DAY) to a date (eg: 20/01/20) and return the result (eg: 20/01/10).
DAY	Returns the day for the given date.
DAYNAME	Returns the weekday name for the given date.
DAYOFWEEK	Returns the index for the weekday for the given date.
DAYOFYEAR	Returns the day of the year for the given date.
EXTRACT	Extract from the date the given part (eg MONTH for 20/01/20 = 01).
FROM DAYS	Return the date from the given numeric date value.
HOUR	Return the hour from the given date.

Numeric Functions	
Name	**Description**
LAST DAY	Get the last day of the month for the given date.
LOCALTIME	Gets the current local date and time.
LOCALTIMESTAMP	Same as LOCALTIME.
MAKEDATE	Creates a date and returns it, based on the given year and number of days values.
MAKETIME	Creates a time and returns it, based on the given hour, minute and second values.
MICROSECOND	Returns the microsecond of a given time or datetime.
MINUTE	Returns the minute of the given time or datetime.
MONTH	Returns the month of the given date.
MONTHNAME	Returns the name of the month of the given date.
NOW	Same as LOCALTIME.
PERIOD_ADD	Adds the given number of months to the given period.
PERIOD_DIFF	Returns the difference between 2 given periods.
QUARTER	Returns the year quarter for the given date.
SECOND	Returns the second of a given time or datetime.
SEC_TO_TIME	Returns a time based on the given seconds.
STR_TO_DATE	Creates a date and returns it based on the given string and format.
SUBDATE	Same as DATE_SUB.
SUBTIME	Subtracts a time interval (eg: 02:00) to a time or datetime (05:00) and return the result (03:00).
SYSDATE	Same as LOCALTIME.
TIME	Returns the time from a given time or datetime.
TIME_FORMAT	Returns the given time in the given format.

Numeric Functions	
Name	**Description**
TIME_TO_SEC	Converts and returns a time into seconds.
TIMEDIFF	Returns the difference between 2 given time/datetime expressions.
TIMESTAMP	Returns the datetime value of the given date or datetime.
TO_DAYS	Returns the total number of days that have passed from '00-00-0000' to the given date.
WEEK	Returns the week number for the given date.
WEEKDAY	Returns the weekday number for the given date.
WEEKOFYEAR	Returns the week number for the given date.
YEAR	Returns the year from the given date.
YEARWEEK	Returns the year and week number for the given date.

Misc Functions

Numeric Functions	
Name	**Description**
IN	Returns the given number in binary.
BINARY	Returns the given value as a binary string.
CAST	Convert one type into another.
COALESCE	From a list of values, return the first non-null value.
CONNECTION_ID	For the current connection, return the unique connection ID.
CONV	Convert the given number from one numeric base system into another.
CONVERT	Convert the given value into the given datatype or character set.
CURRENT_USER	Return the user and hostname which was used to authenticate with the server.
DATABASE	Get the name of the current database.
GROUP BY	Used alongside aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the results. **Example: Lists the number of users with active orders.** SELECT COUNT(user_id), active_orders FROM users GROUP BY active_orders;
HAVING	It's used in the place of WHERE with aggregate functions. **Example: Lists the number of users with active orders, but only include users with more than 3 active orders.** SELECT COUNT(user_id), active_orders FROM users GROUP BY active_orders HAVING COUNT(user_id) > 3;
IF	If the condition is true return a value, otherwise return another value.
IFNULL	If the given expression equates to null, return the given value.

Numeric Functions	
Name	**Description**
ISNULL	If the expression is null, return 1, otherwise return 0.
LAST_INSERT_ID	For the last row which was added or updated in a table, return the auto increment ID.
NULLIF	Compares the 2 given expressions. If they are equal, NULL is returned, otherwise the first expression is returned.
SESSION_USER	Return the current user and hostnames.
SYSTEM_USER	Same as SESSION_USER.
USER	Same as SESSION_USER.
VERSION	Returns the current version of the MySQL powering the database.

Wildcard Characters

In SQL, Wildcards are special characters used with the LIKE and NOT LIKE keywords which allow us to search data with sophisticated patterns much more efficiently

Wildcards	
Name	**Description**

%	Equates to zero or more characters.
	Example 1: Find all users with surnames ending in 'son'.
	`SELECT * FROM users`
	`WHERE surname LIKE '%son';`
	
	Example 2: Find all users living in cities containing the pattern 'che'
	`SELECT * FROM users`
	`WHERE city LIKE '%che%';`

_	Equates to any single character.
	Example: Find all users living in cities beginning with any 3 characters, followed by 'chester'.
	`SELECT * FROM users`
	`WHERE city LIKE '___chester';`

[charlist]	Equates to any single character in the list.
	Example 1: Find all users with first names beginning with J, H or M.
	`SELECT * FROM users`
	`WHERE first_name LIKE '[jhm]%';`
	
	Example 2: Find all users with first names beginning letters between A-L.
	`SELECT * FROM users`
	`WHERE first_name LIKE '[a-l]%';`
	
	Example 3: Find all users with first names not ending with letters between n-s.
	`SELECT * FROM users`
	`WHERE first_name LIKE '%[!n-s]';`

Keys

In relational databases, there is a concept of primary and foreign keys. In SQL tables, these are included as constraints, where a table can have a primary key, a foreign key, or both.

Primary Key

A primary key allows each record in a table to be uniquely identified. There can only be one primary key per table, and you can assign this constraint to any single or combination of columns. However, this means each value within this column(s) must be unique.

Typically in a table, the primary key is an ID column, and is usually paired with the AUTO_INCREMENT keyword. This means the value increases automatically as new records are created.

Example 1 (MySQL)

Create a new table and set the primary key to the ID column.

```
CREATE TABLE users (
id int NOT NULL AUTO_INCREMENT,
first_name varchar(255),
last_name varchar(255) NOT NULL,
address varchar(255),
email varchar(255),
PRIMARY KEY (id)
);
```

Example 2 (MySQL)

Alter an existing table and set the primary key to the first_name column.

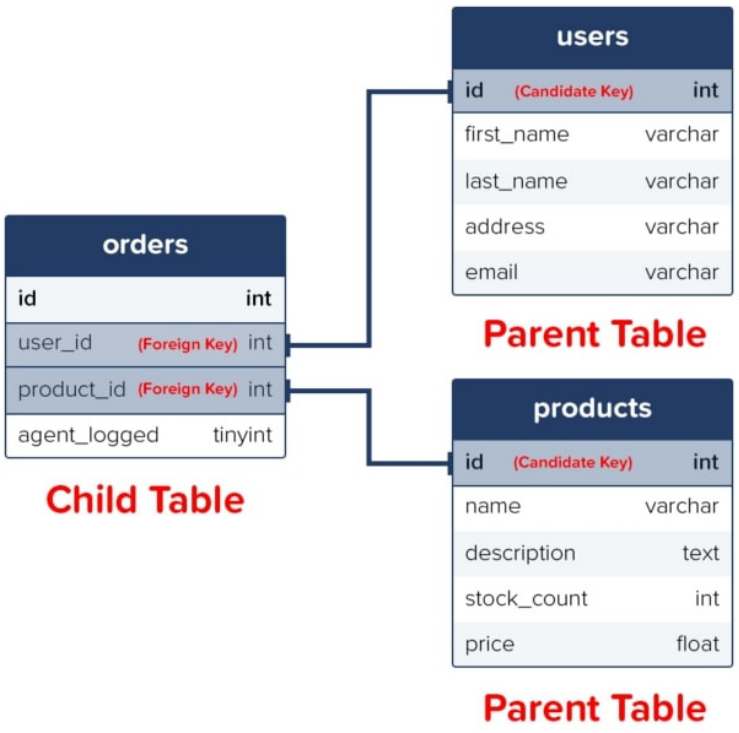
```
ALTER TABLE users
ADD PRIMARY KEY (first_name);
```

Foreign Key

A foreign key can be applied to one column or many and is used to link 2 tables together in a relational database.

As seen in the diagram below, the table containing the foreign key is called the child key, whilst the table which contains the referenced key, or candidate key, is called the parent table.

This essentially means that the column data is shared between 2 tables, as a foreign key also prevents invalid data from being inserted which isn't also present in the parent table.



Example 1 (MySQL)

Create a new table and turn any columns that reference IDs in other tables into foreign keys.

```
CREATE TABLE orders (
id int NOT NULL,
user_id int,
product_id int,
PRIMARY KEY (id),
FOREIGN KEY (user_id) REFERENCES users(id),
FOREIGN KEY (product_id) REFERENCES products(id)
);
```

Example 2 (MySQL)

Alter an existing table and create a foreign key.

```
ALTER TABLE orders
ADD FOREIGN KEY (user_id) REFERENCES users(id);
```

Indexes

Indexes are attributes that can be assigned to columns that are frequently searched against to make data retrieval a quicker and more efficient process.

This doesn't mean each column should be made into an index though, as it takes longer for a column with an index to be updated than a column without. This is because when indexed columns are updated, the index itself must also be updated.

Wildcards	
Name	**Description**

CREATE INDEX	Creates an index named 'idx_test' on the first_name and surname columns of the users table. In this instance, duplicate values are allowed.
	`CREATE INDEX idx_test`
	`ON users (first_name, surname);`

CREATE UNIQUE INDEX	Creates an index named 'idx_test' on the first_name and surname columns of the users table. In this instance, duplicate values are allowed.
	`CREATE UNIQUE INDEX idx_test`
	`ON users (first_name, surname);`

DROP INDEX	Creates an index named 'idx_test' on the first_name and surname columns of the users table. In this instance, duplicate values are allowed.
	`ALTER TABLE users`
	`DROP INDEX idx_test;`

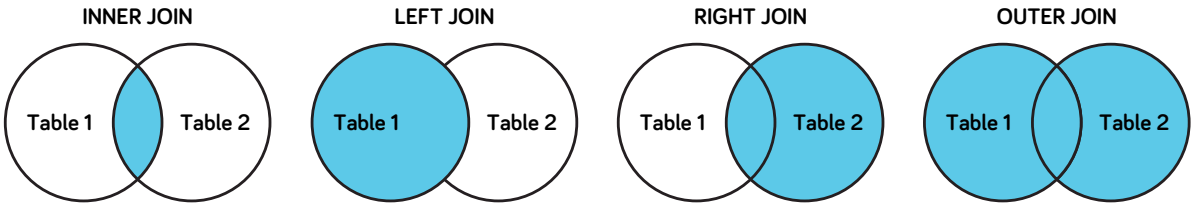
Joins

In SQL, a JOIN clause is used to return a results set which combines data from multiple tables, based on a common column which is featured in both of them

There are a number of different joins available for you to use:

- Inner Join (Default): Returns any records which have matching values in both tables.
- Left Join: Returns all of the records from the first table, along with any matching records from the second table.
- Right Join: Returns all of the records from the second table, along with any matching records from the first.
- Full Join: Returns all records from both tables when there is a match.

A common way of visualising how joins work is like this:



In the following example, an inner join will be used to create a new unifying view combining the orders table and then 3 different tables

We'll replace the user_id and product_id with the first_name and surname columns of the user who placed the order, along with the name of the item which was purchased.

orders

id	user_id	product_id	agent_logged
1	5	196	0
2	4	32	1
3	6	310	0
4	10	196	1
5	1	67	1
6	1	341	1
7	1	875	0
8	9	3	1
9	5	23	1
10	8	196	1

users

id	first_name	last_name	address	email
1	Luke	Harrison	1640 Rivers...	luke@lukeh...
2	Heather	Reynolds	742 Evergr...	heza@hot...
3	Simon	Clarkson	7 Peterbou...	smr@yaho...
4	Claire	Simpson	15 Musgra...	claire@hot...
5	Oliver	Harrison	1640 Rivers...	oliver@ya...
6	James	Gilbert	598 Firshil...	jgill@appl...
7	Michael	Johnson	12 Redmire...	mj@yahoo...
8	Thomas	Smith	342 Brown...	t.smith@al...
9	Robyn	Gilbert	598 Firshil...	summer@d...
10	Bryony	Brown	165 South...	bryony@h...

products

id	name	description	stock_count	price
192	Carton Do...	Whether y...	0	14.99
23	Cardboar...	Declutter...	1	3.49
3	SmartMo...	NULL	1	24.99
32	TripLast 33...	Cost effec...	4	16.50
875	A4 Storag...	Dimensio...	5	4.99
456	Pack of 50...	Date first a...	5	12.99
341	Set of 2 S...	5 year gua...	8	4.99
67	Large Car...	Need som...	10	12.99
196	10 X Plasti...	Pack of 10...	10	15.99
310	StorePac 5...	High qual...	10	9.99

```
SELECT orders.id, users.first_name, users.surname, products.name as 'product name'
FROM orders
INNER JOIN users on orders.user_id = users.id
INNER JOIN products on orders.product_id = products.id;
```

Would return a results set which looks like:

Inner Join Result Set

id	first_name	surname	product name
1	Oliver	Harrison	10 X Plasti...
2	Claire	Simpson	TripLast 33...
3	James	Gilbert	StorePac 5...
4	Bryony	Brown	10 X Plasti...
5	Luke	Harison	Large Car...
6	Luke	Harrison	Set of 2 S...
7	Luke	Harrison	A4 Storag...
8	Robyn	Gilbert	SmartMo...
9	Oliver	Harrison	Cardboar...
10	Thomas	Smith	10 X Plasti...

View

A view is essentially a SQL results set that get stored in the database under a label, so you can return to it later, without having to rerun the query. These are especially useful when you have a costly SQL query which may be needed a number of times, so instead of running it over and over to generate the same results set, you can just do it once and save it as a view.

Creating Views

To create a view, you can do so like this:

```
CREATE VIEW priority_users AS
SELECT * FROM users
WHERE country = 'United Kingdom';
```

Then in future, if you need to access the stored result set, you can do so like this:

```
SELECT * FROM [priority_users];
```

Replacing Views

With the CREATE OR REPLACE command, a view can be updated.

```
CREATE OR REPLACE VIEW [priority_users] AS
SELECT * FROM users
WHERE country = 'United Kingdom' OR country='USA';
```

Deleting Views

To delete a view, simply use the DROP VIEW command.

```
DROP VIEW priority_users;
```

SQL Basics Cheat Sheet

LearnSQL.com

SQL

SQL, or *Structured Query Language*, is a language to talk to databases. It allows you to select specific data and to build complex reports. Today, SQL is a universal language of data. It is used in practically all technologies that process data.

SAMPLE DATA

COUNTRY

id	name	population	area
1	France	66600000	640680
2	Germany	80700000	357000
...	...	...	...

CITY

id	name	country_id	population	rating
1	Paris	1	2243000	5
2	Berlin	2	3460000	3
...	...	...	...	...

QUERYING SINGLE TABLE

Fetch all columns from the `country` table:

```
SELECT *
FROM country;
```

Fetch `id` and `name` columns from the `city` table:

```
SELECT id, name
FROM city;
```

Fetch city names sorted by the `rating` column in the default ASCending order:

```
SELECT name
FROM city
ORDER BY rating [ASC];
```

Fetch city names sorted by the `rating` column in the DESCending order:

```
SELECT name
FROM city
ORDER BY rating DESC;
```

ALIASES

COLUMNS

```
SELECT name AS city_name
FROM city;
```

TABLES

```
SELECT co.name, ci.name
FROM city AS ci
JOIN country AS co
  ON ci.country_id = co.id;
```

FILTERING THE OUTPUT

COMPARISON OPERATORS

Fetch names of cities that have a rating above 3:

```
SELECT name
FROM city
WHERE rating > 3;
```

Fetch names of cities that are neither Berlin nor Madrid:

```
SELECT name
FROM city
WHERE name != 'Berlin'
  AND name != 'Madrid';
```

TEXT OPERATORS

Fetch names of cities that start with a 'P' or end with an 's':

```
SELECT name
FROM city
WHERE name LIKE 'P%'
   OR name LIKE '%s';
```

Fetch names of cities that start with any letter followed by 'ublin' (like Dublin in Ireland or Lublin in Poland):

```
SELECT name
FROM city
WHERE name LIKE '_ublin';
```

OTHER OPERATORS

Fetch names of cities that have a population between 500K and 5M:

```
SELECT name
FROM city
WHERE population BETWEEN 500000 AND 5000000;
```

Fetch names of cities that don't miss a rating value:

```
SELECT name
FROM city
WHERE rating IS NOT NULL;
```

Fetch names of cities that are in countries with IDs 1, 4, 7, or 8:

```
SELECT name
FROM city
WHERE country_id IN (1, 4, 7, 8);
```

QUERYING MULTIPLE TABLES

INNER JOIN

`JOIN` (or explicitly **`INNER JOIN`**) returns rows that have matching values in both tables.

```
SELECT city.name, country.name
FROM city
[INNER] JOIN country
  ON city.country_id = country.id;
```

CITY			COUNTRY	
id	name	country_id	id	name
1	Paris	1	1	France
2	Berlin	2	2	Germany
3	Warsaw	4	3	Iceland

LEFT JOIN

`LEFT JOIN` returns all rows from the left table with corresponding rows from the right table. If there's no matching row, **`NULL`**s are returned as values from the second table.

```
SELECT city.name, country.name
FROM city
LEFT JOIN country
  ON city.country_id = country.id;
```

CITY			COUNTRY	
id	name	country_id	id	name
1	Paris	1	1	France
2	Berlin	2	2	Germany
3	Warsaw	4	NULL	NULL

RIGHT JOIN

`RIGHT JOIN` returns all rows from the right table with corresponding rows from the left table. If there's no matching row, **`NULL`**s are returned as values from the left table.

```
SELECT city.name, country.name
FROM city
RIGHT JOIN country
  ON city.country_id = country.id;
```

CITY			COUNTRY	
id	name	country_id	id	name
1	Paris	1	1	France
2	Berlin	2	2	Germany
NULL	NULL	NULL	3	Iceland

FULL JOIN

`FULL JOIN` (or explicitly **`FULL OUTER JOIN`**) returns all rows from both tables – if there's no matching row in the second table, **`NULL`**s are returned.

```
SELECT city.name, country.name
FROM city
FULL [OUTER] JOIN country
  ON city.country_id = country.id;
```

CITY			COUNTRY	
id	name	country_id	id	name
1	Paris	1	1	France
2	Berlin	2	2	Germany
3	Warsaw	4	NULL	NULL
NULL	NULL	NULL	3	Iceland

CROSS JOIN

`CROSS JOIN` returns all possible combinations of rows from both tables. There are two syntaxes available.

```
SELECT city.name, country.name
FROM city
CROSS JOIN country;
```

```
SELECT city.name, country.name
FROM city, country;
```

CITY			COUNTRY	
id	name	country_id	id	name
1	Paris	1	1	France
1	Paris	1	2	Germany
2	Berlin	2	1	France
2	Berlin	2	2	Germany

NATURAL JOIN

`NATURAL JOIN` will join tables by all columns with the same name.

```
SELECT city.name, country.name
FROM city
NATURAL JOIN country;
```

CITY			COUNTRY	
country_id	id	name	name	id
6	6	San Marino	San Marino	6
7	7	Vatican City	Vatican City	7
5	9	Greece	Greece	9
10	11	Monaco	Monaco	10

`NATURAL JOIN` used these columns to match rows: **`city.id, city.name, country.id, country.name`**
`NATURAL JOIN` is very rarely used in practice.

SQL KEYWORDS

KEYWORDS	DESCRIPTION
ADD	Adds a new column to an existing table. **Example**: Adds a new column named email_address' to a table named 'users!. ALTER TABLE users ADD email address varchar(255);
ADD CONSTRAINT	It creates a new constraint on an existing table, which is used to specify rules for any data in the table. **Example**: Adds a new PRIMARY KEY constraint named 'user' on columns ID and SURNAME. ALTER TABLE users ADD CONSTRAINT user PRIMARY KEY (ID, SURNAME);
ALTER TABLE	Adds, deletes or edits columns in a table. It can also be used to add and delete constraints in a table, as per the above. **Example**: Adds a new boolean column called approved to a table named 'deals! ALTER TABLE deals ADD approved boolean; **Example 2**: Deletes the approved column from the 'deals' table. ALTER TABLE deals DROP COLUMN approved;
ALTER COLUMN	Changes the data type of a table's column. Example In the 'users' table, make the column 'incept_date' into a 'datetime' type. ALTER TABLE users ALTER COLUMN incept_date datetime;
ALL	Returns true if all of the subquery values meet the passed condition. **Example**: Returns the users with a higher number of tasks than the user with the highest number of tasks in the HR department (id 2). SELECT first_name, surname, tasks_no FROM users WHERE tasks_no > ALL (SELECT tasks FROM user WHERE department_id = 2);
AND	Used to join separate conditions within a WHERE clause. **Example**:Returns events located in London, United Kingdom. SELECT * FROM events WHERE host_country='United Kingdom' AND host_city='London';
ANY	Returns true if any of the subquery values meet the given condition. **Example**:Returns products from the products table which have received orders – stored in the orders table - with a quantity of more than 5. SELECT name FROM products WHERE productId = ANY (SELECT productId FROM orders WHERE Qty >
AS	Renames a table or column with an alias value which only exists for the duration of the query. **Example**:Aliases north_east_user_ subscriptions column. SELECT north_east_user_subscriptions AS ne_subs FROM users WHERE ne_subs > 5;
ASC	Used with ORDER BY to return the data in ascending order. **Example**:Apples, Bananas, Peaches, Raddish.
BETWEEN	Selects values within the given range. **Example 1**:Selects stock with a quantity between 100 and 150. SELECT * FROM stock WHERE quantity BETWEEN 100 AND 150; **Example 2**:Selects stock with a quantity NOT between 100 and 150. Alternatively, using the NOT keyword here reverses the logic and selects values outside the given range. SELECT * FROM stock WHERE quantity NOT BETWEEN 100 AND 150;
CASE	Change query output depending on conditions. **Example 1**:Returns users and their subscriptions, along with a new column called activity_levels that makes a judgement based on the number of subscriptions. SELECT first_name, surname, subscriptions CASE WHEN subscriptions > 10 THEN 'Very active' WHEN Quantity BETWEEN 3 AND 10 THEN 'Active' ELSE 'Inactive' END AS activity levels FROM users;
CHECK	Adds a constraint that limits the value which can be added to a column. **Example 1(MySQL)**: Makes sure any users added to the users table are 18 or over. CREATE TABLE users first_name varchar(255), age int, CHECK (age=18)); **Example 2(MySQL)**: Adds a check after the table has already been created. ALTER TABLE users ALTER TABLE users
CREATE DATABASE	Creates a new database. **Example 1(MySQL)**: Creates a new database named websiteset. CREATE DATABASE websitesetup;
CREATE TABLE	Creates a new table. **Example**: Creates a new table called 'users' in the websitesetup database. CREATE TABLE users id int, first_name varchar(255), surname varchar(255), address varchar(255), contact number int);
DEFAULT	Sets a default value for a column; **Example 1(MySQL)**: Creates a new table called Products which has a name column with a default value of 'Placeholder Name' and an available_from column with a default value of today's date. CREATE TABLE products id int, name varchar(255) DEFAULT 'Placeholder Name', available from date DEFAULT GETDATE()); **Example 2(MySQL)**: The same as above, but editing an existing table. ALTER TABLE products ALTER name SET DEFAULT 'Placeholder Name' ALTER available_from SET DEFAULT GETDATE();
DELETE	Delete data from a table. **Example**: Removes a user with a user_id of 674. DELETE FROM users WHERE user_id = 674;
DESC	Used with ORDER BY to return the data in descending order. **Example**: Raddish, Peaches, Bananas, Apples.

KEYWORDS	DESCRIPTION
DROP COLUMN	Deletes a column from a table. **Example**: Removes the first_name column from the users table. ALTER TABLE users DROP COLUMN first_name
DROP DATABASE	Deletes the entire database. **Example**: Deletes a database named 'websitesetup. DROP DATABASE websitesetup;
DROP DEFAULT	Removes a default value for a column. **Example 1(MySQL)**: Removes the default value from the name column in the 'products' table. ALTER TABLE products ALTER COLUMN name DROP DEFAULT;
DROP TABLE	Deletes a table from a database. **Example**: Removes the users table. DROP TABLE users;
EXISTS	Checks for the existence of any record within the subquery, returning true if one or more records are returned. **Example**: Lists any dealerships with a deal finance percentage less than 10. SELECT dealership_name FROM dealerships WHERE EXISTS (SELECT deal_name FROM deals WHERE dealership_id = deals.dealership_id AND finance_ percentage < 10);
FROM	Specifies which table to select or delete data from. **Example**:Selects data from the users table. SELECT area_manager FROM area_managers WHERE EXISTS (SELECT ProductName FROM Products WHERE area_manager_id = deals.area_manager_id AND Price < 20);
IN	Used alongside a WHERE cause as a shorthand for multiple OR conditions. So instead of: SELECT * FROM users WHERE country = 'USA' OR country = 'United Kingdom' OR country = 'Russia' OR country = 'Australia'; You can use:- SELECT * FROM users WHERE country IN (USA', 'United Kingdom', 'Russia', 'Australia');
INSERT INTO	Add new rows to a table. **Example**: Adds a new vehicle. INSERT INTO cars (make, model, mileage, year) VALUES ('Audi', 'A3', 30000, 2016);
IS NULL	Tests for empty (NULL) values. **Example**: Returns users that haven't given a contact number. SELECT * FROM users WHERE contact_number IS NULL;
IS NOT NULL	The reverse of NULL Tests for values that aren't empty / NULL.
LIKE	Returns true if the operand value matches a pattern. **Example**: Returns true if the user's first_name ends with 'son'. SELECT * FROM users WHERE first_name LIKE '%son';
NOT	Returns true if a record DOESN'T meet the condition. **Example**: Returns true if the user's first_name doesn't end with 'son'. SELECT * FROM users WHERE first_name NOT LIKE 'son';
OR	Used alongside WHERE to include data when either condition is true. **Example**: Returns users that live in either Sheffield or Manchester. SELECT * FROM users WHERE city = 'Sheffield' OR 'Manchester';
ORDER BY	Used to sort the result data in ascending (default) or descending order through the use of ASC or DESC keywords. **Example**: Returns countries in alphabetical order. SELECT * FROM countries ORDER BY name;
ROWNUM	Returns results where the row number meets the passed condition. **Example** : Returns the top 10 countries from the countries table. SELECT * FROM countries WHERE ROWNUM C= 10;
SELECT	Used to select data from a database, which is then returned in a results set. **Example** : Selects all columns from all users. SELECT * FROM users; Example 2: Selects the first_name and surname columns from all users.XX SELECT first_name, surname FROM users;
SELECT DISTINCT	Sames as SELECT, except duplicate values are excluded. **Example** : Creates a backup table using data from the users table. SELECT * INTO usersBackup2020 FROM users;
SELECT INTO	Copies data from one table and inserts it into another. **Example**: Returns all countries from the users table, removing any duplicate values (which would be highly likely). SELECT DISTINCT country from users;
SELECT TOP	Allows you to return a set number of records to return from a table. **Example**: Returns the top 3 cars from the cars table. SELECT TOP 3 * FROM cars;
SET	Used alongside UPDATE to update existing data in a table. **Example**: Updates the value and quantity values for an order with an id of 642 in the orders table. UPDATE orders SET value = 19.49, quantity = 2 WHERE id = 642;
SOME	Identical to ANY
TOP	Used alongside SELECT to return a set number of records from a table. **Example**: Returns the top 5 users from the users table. SELECT TOP 5 * FROM users;
TRUNCATE TABLE	Similar to DROP, but instead of deleting the table and its data, this deletes only the data. **Example**: Empties the sessions table, but leaves the table itself intact. TRUNCATE TABLE sessions;
UNION	Combines the results from 2 or more SELECT statements and returns only distinct values. **Example**: Returns the cities from the events and subscribers tables. SELECT city FROM events UNION SELECT city from subscribers;
UNION ALL	The same as UNION but includes duplicate values.
UNIQUE	This constraint ensures all values in a column are unique. **Example 1 (MySQL)**: Adds a unique constraint to the id column when creating a new users table. CREATE TABLE users id int NOT NULL, name varchar(255) NOT NULL, UNIQUE (id)); **Example 2 (MySQL)**: Alters an existing column to add a UNIQUE constraint. ALTER TABLE users ADD UNIQUE (id);
UPDATE	Updates existing data in a table. **Example**: Updates the mileage and service Due values for a vehicle with an UPDATE cars SET mileage = 23500, serviceDue = 8 WHERE id = 45;
VALUES	Used alongside the INSERT INTO keyword to add new values to a table. **Example**: Adds a new car to the cars table. SET mileage = 23500, serviceDue = 8 INSERT INTO cars (name, model, year) VALUES ('Ford', 'Fiesta', 2010);
WHERE	Filters results to only include data which meets the given condition. **Example**: Returns orders with a quantity of more than 1 item. SELECT * FROM orders WHERE quantity > 1;

COMMENTS

SINGLE LINE COMMENTS

Single line comments start with -. Any text after these 2 characters to the end of the line will be ignored

```
-- My Select query
SELECT * FROM users;
```

MULTILINE COMMENTS

Multiline comments start with /* and end with */. They stretch across multiple lines until the closing characters have been found.

```
/*
This is my select query.
It grabs all rows of data from the users table
SELECT * FROM users;
This is another select query, which I don't
want to execute yet
SELECT * FROM tasks;
*/
```

MySQL Data Types

```
CREATE TABLE users (
id int,
first_name varchar(255)
);
```

STRING DATA TYPES

DATA TYPE	DESCRIPTION
CHAR(SIZE)	Fixed length string which can contain letters, numbers and special characters. The size parameter sets the maximum string length, from 0 - 255 with a default of 1.
VARCHAR(SIZE)	Variable length string similar to CHARQ, but with a maximum string length range from 0 to 65535.
BINARY(SIZE)	Similar to CHARO but stores binary byte strings.
VARCHAR(SIZE)	Similar to CHARO but stores binary byte strings.
TINYBLOB	Holds Binary Large Objects (BLOBs) with a maxlength of 255 bytes.
TINYTEXT	Holds a string with a maximum length of 255 characters. Use VARCHAR() instead, as it's fetched much faster.
TEXT(size)	Holds a string with a maximum length of 65535 bytes. Again, better to use VARCHARO
BLOB(size)	Holds Binary Large Objects (BLOBS) with a max length of 65535 bytes.
MEDIUMBLOB	Holds Binary Large Objects (BLOBs) with a max length of 16,777,215 bytes
LONGTEXT	Holds a string with a maximum length of 4,294,967,295 characters.
LONGBLOB	Holds Binary Large Objects (BLOBS) with a max length of 4,294,967,295 bytes
ENUM(a, b, c, etc...)	A string object that only has one value, which is chosen from a list of values which you define, up to a maximum of 65535 values. If a value is added which isn't on this list, it's replaced with a blank value instead. Think of ENUM being similar to HTML radio boxes in this regard. CREATE TABLE tshirts (color ENUM 'red', 'green', 'blue', 'yellow', 'purple'));
SET (a, b, c, etc...)	A string object that can have 0 or more values, which is chosen from a list of values which you define, up to a maximum of 64 values. Think of SET being similar to HTML checkboxes in this regard

NUMERIC DATA TYPES

DATA TYPE	DESCRIPTION
BIT(SIZE)	A bit-value type with a default of 1. The allowed number of bits in a value is set via the size parameter, which can hold values from 1 to 64.
TINYINT(SIZE)	A very small integer with a signed range of -128 to 127, and an unsigned range of 0 to 255. Here, the size parameter specifies the maximum allowed display width, which is 255.
BOOLEAN	Same as BOOL
SMALLINT(size)	A small integer with a signed range of -32768 to 32767, and an unsigned range from 0 to 65535. Here, the size parameter specifies the maximum allowed display width, which is 255.
MEDIUMINT(size)	A medium integer with a signed range of -8388608 to 8388607,and an unsigned range from 0 to 16777215. Here, the size parameter specifies the maximum allowed display width, which is 255.
INT(size)	A medium integer with a signed range of -2147483648 to 2147483647, and an unsigned range from 0 to 4294967295. Here, the size parameter specifies the maximum allowed display width, which is 255.
INTEGER(size)	Same as INT.
BIGINT(size)	A medium integer with a signed range of -9223372036854775808 to 9223372036854775807, and an unsigned range from 0 to18446744073709551615. Here, the size parameter specifies the maximum allowed display width, which is 255.
FLOAT(p)	A floating point number value. If the precision (p) parameter is between O to 24, then the data type is set to FLOAT(), whilst if its from 25 to 53,the data type is set to DOUBLE(). This behaviour is to make the storage of values more efficient.
DOUBLE(size, d)	A floating point number value where the total digits are set by the size parameter, and the number of digits after the decimal point is set by the d parameter
DECIMAL(size, d)	An exact fixed point number where the total number of digits is set by the size parameters, and the total number of digits after the decimal point is set by the d parameter. For size, the maximum number is 65 and the default is 10, whilst ford,the maximum number is 30 and the default is 10.
DEC(sze, d)	Same as DECIMAL.

DATE/TIME DATA TYPES

DATA TYPE	DESCRIPTION
DATE	A simple date in YYYY-MM-DD format, with a supported range from '1000-01-01' to '9999-12-31.
DATETIME(fsp)	A date time in YYYY-MM-DD hh:mmess format, with a supported range from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'.' By adding DEFAULT and ON UPDATE to the column definition, it automatically sets to the current date/time.
TIMESTAMP(fsp)	A Unix Timestamp, which is a value relative to the number of seconds since the Unix epoch (1970-01-01 00:00:00' UTC). This has a supported range from '1970-01-01 00:00:01 UTC to 2038-01-09 03:14:07' UTC By adding DEFAULT CURRENT_TIMESTAMP and ON UPDATE CURRENT TIMESTAMP to the column definition, it automatically sets to current date/time.
TIME(fsp)	A time in hh:mmess format, with a supported range from '838:59:59 to '838:59:59.
YEAR	A year, with a supported range of '1901' to '2155.

OPERATORS

ARITHMETIC OPERATORS

OPERATOR	DESCRIPTION
+	Add
-	Subtract
*	Multiply
/	Divide
%	Modulo

BITWISE OPERATORS

OPERATOR	DESCRIPTION
&	Bitwise AND
I	Bitwise OR
^	Bitwise exclusive OR

COMPARISON OPERATORS

OPERATOR	DESCRIPTION
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to

COMPOUND OPERATORS

OPERATOR	DESCRIPTION
+=	Add equals
-=	Subtract equals
*=	Multiply equals
/=	Divide equals
%=	Modulo equals
&=	Bitwise AND equals
^=	Bitwise exclusive equals
I*=	Bitwise OR equals

FUNCTIONS

STRING FUNCTIONS

DATA TYPE	DESCRIPTION
ASCIIA	Returns the equivalent ASCII value for a specific character.
CHAR_LENGTH	Returns the character length of a string.
CHARACTER LENGTH	Same as CHAR_LENGTH.
CONCAT	Adds expressions together, with a minimum of 2.
CONCAT_WS	Adds expressions together, but with a separator between each value.
FIELD	Returns an index value relative to the position of a value within a list of values.
FORMAT	When passed a number, returns that number formatted to include commas (eg 3,400,000).
INSERT	Allows you to insert one string into another at a certain point, for a certain number of characters.
INSTR	Returns the position of the first time one string appears within another.
LCASE	Convert a string to lowercase.
LEFT	Starting from the left, extract the given number of characters from a string and return them as another.
LENGTH	Returns the length of a string, but in bytes.
LOCATE	Returns the first occurrence of one string within another,
LOWER	Same as LCASE
LPAD	Left pads one string with another, to a specific length.
LTRIM	Remove any leading spaces from the given string.
MID	Extracts one string from another, starting from any position.
POSITION	Returns the position of the first time one substring appears within another.
REPEAT	Allows you to repeat a string.
REPLACE	Allows you to replace any instances of a substring within a string, with a new substring.
REVERSE	Reverses the string.
RIGHT	Starting from the right, extract the given number of characters from a string and return them as another.
RPAD	Right pads one string with another, to a specific length.
RTRIM	Removes any trailing spaces from the given string.
SPACE	Returns a string full of spaces equal to the amount you pass it.
STRCMP	Compares 2 strings for differences.
SUBSTR	Extracts one substring from another, starting from any position.
SUBSTRING	Same as SUBSTR
SUBSTRING_ INDEX	Returns a substring from a string before the passed substring is found the number of times equals to the passed number.
TRIM	Removes trailing and leading spaces from the given string Same as if you were to run LTRIM and RTRIM together.
UCASE	Convert a string to uppercase.
UPPER	Same as UCASE.

NUMERIC FUNCTIONS

DATA TYPE	DESCRIPTION
ABS	Returns the absolute value of the given number.
ACOS	Returns the arc cosine of the given number.
ASIN	Returns the arc sine of the given number.
ATAN	Returns the arc tangent of one or 2 given numbers
ATAN2	Return the arc tangent of 2 given numbers.
AVG	Returns the average value of the given expression.
CEIL	Returns the closest whole number (integer) upwards from a given decimal point number.
CEILING	Same as CEIL
COS	Returns the cosine of a given number.
COT	Returns the cotangent of a given number.
COUNT	Returns the amount of records that are returned by a SELECT query.
DEGREES	Converts a radians value to degrees.
DIV	Allows you to divide integers.
EXP	Returns to the power of the given number.
FLOOR	Returns the closest whole number (integer) downwards from a given decimal point number.
GREATEST	Returns the highest value in a list of arguments.
LEAST	Returns the smallest value in a list of arguments.
LN	Returns the natural logarithm of the given number.
LOG	Returns the natural logarithm of the given number, or the logarithm of the given number to the given base.
LOG10	Does the same as LOG, but to base 10
LOG2	Does the same as LOG, but to base 2.
MAX	Returns the highest value from a set of values.
MIN	Returns the lowest value from a set of values.
MOD	Returns the remainder of the given number divided by the other given number.
PI	Returns PI.
POW	Returns the value of the given number raised to the power of the other given number.
POWER	Same as POW.
RADIANS	Converts a degrees value to radians.
RAND	Returns a random number.
SIGN	Returns the sign of the given number.
SIN	Returns the sine of the given number.
SQRT	Returns the square root of the given number.
SUM	Returns the value of the given set of values combined.
TAN	Returns the tangent of the given number.
TRUNCATE	Returns number truncated to the given number of decimal places.

NUMERIC FUNCTIONS

DATA TYPE	DESCRIPTION
ADDDATE	Add a date interval (eg 10 DAY) to a date (eg 20/01/20) and return the result (eg: 20/01/30).
ADDTIME	Add a time interval (eg: 02:00) to a time or datetime (05:00) and return the result (07:00).
CURDATE	Get the current date.
CURRENT_DATE	Same as CURDATE.
CURRENT_ TIMESTAMP	Get the current date and time.
CURTIME	Same as CURRENT_TIME.
DATE	Extracts the date from a datetime expression.
DATEDIFF	Returns the number of days between the 2 given dates.
DATE_ADD	Same as ADDDATE.
DATE_FORMAT	Formats the date to the given pattern.
DATE_SUB	Subtract a date interval (eg: 10 DAY) to a date (eg 20/01/20) and return the result (eg: 20/01/10).
DAY	Returns the day for the given date.
DAYNAME	Returns the weekday name for the given date.
DAYOFWEEK	Returns the index for the weekday for the given date.
DAYOFYEAR	Returns the day of the year for the given date.
EXTRACT	Extract from the date the given part (eg MONTH for 20/01/20=01).
FROM DAYS	Return the date from the given numeric date value. HOUR.
HOUR	Return the hour from the given date.
LAST DAY	Get the last day of the month for the given date.
LOCALTIME	Gets the current local date and time.
LOCALTIMESTAMP	Same as LOCALTIME.
MAKEDATE	Creates a date and returns it, based on the given year and number of days values.
MICROSECOND	Returns the microsecond of a given time or datetime.
MINUTE	Returns the minute of the given time or datetime.
MONTH	Returns the month of the given date.
MONTHNAME	Returns the name of the month of the given date.
NOW	Same as LOCALTIME.
PERIOD_ADD	Adds the given number of months to the given period.
PERIOD_DIFF	Returns the difference between 2 given periods.
QUARTER	Returns the year quarter for the given date.
SECOND	Returns the second of a given time or datetime.
SEC_TO_TIME	Returns a time based on the given seconds.
STR_TO_DATE	Creates a date and returns it based on the given string and format.
SUBDATE	Same as DATE_SUB.
SUBTIME	Subtracts a time interval (eg: 02:00) to a time or datetime (05:00) and return the result (03:00).
SYSDATE	Same as LOCALTIME.
TIME	Returns the time from a given time or datetime.
TIME_FORMAT	Returns the given time in the given format.
TIME_TO_SEC	Converts and returns a time into seconds.
TIMEDIFF	Returns the difference between 2 given time/datetime expressions.
TIMESTAMP	Returns the datetime value of the given date or datetime.
TO_DAYS	Returns the total number of days that have passed from '00-00- 0000' to the given date.
WEEK	Returns the week number for the given date.
WEEKDAY	Returns the weekday number for the given date.
WEEKOFYEAR	Returns the week number for the given date.
YEAR	Returns the year from the given date.
YEARWEEK	Returns the year and week number for the given date.

NUMERIC FUNCTIONS

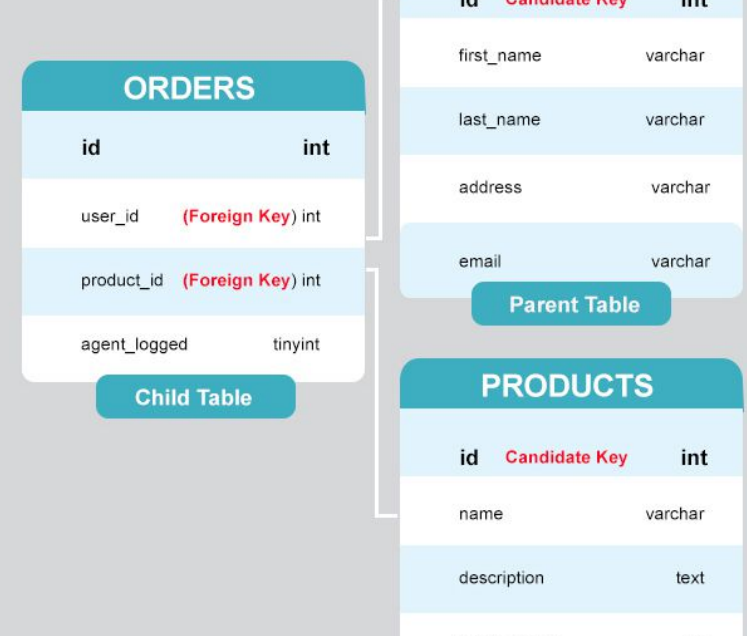
DATA TYPE	DESCRIPTION
IN	Returns the given number in binary.
BINARY	Returns the given value as a binary string
CAST	Convert one type into another.
COALESCE	From a list of values, return the first non-null value.
CONNECTION_ID	For the current connection, return the unique connection ID.
CONV	Convert the given number from one numeric base system into another.
CONVERT	Convert the given value into the given datatype or character set.
CURRENT_USER	Return the user and hostname which was used to authenticate with the server.
DATABASE	Get the name of the current database.
GROUP BY	Used alongside aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the results. **Example**: Lists the number of users with active orders. SELECT COUNT(user_id), active_orders FROM users GROUP BY active_orders;
HAVING	It's used in the place of WHERE with aggregate functions. **Example**: Lists the number of users with active orders, but only include users with more than 3 active orders. SELECT COUNT(user_id), active_orders FROM users GROUP BY active_orders HAVING COUNT(user_id) > 3;
IF	If the condition is true return a value, otherwise return another value.
IFNULL	If the expression is null, return 1 otherwise return O.
LAST_INSERT_ID	For the last row which was added or updated in a table, return the auto increment.
NULLIF	Compares the 2 given expressions. If they are equal, NULL is returned, otherwise the first expression is returned.
SESSION_USER	Return the current user and hostnames.
SYSTEM_USER	Same as SESSION_USER.
VERSION	Returns the current version of the MySQL powering the database.

WILDCARDS

NAME	DESCRIPTION
%	Equates to zero or more characters. **Example 1**: Find all users with surnames ending in 'son'. SELECT * FROM users WHERE surname LIKE 'xson'; **Example 2**: Find all users with surnames ending in 'son'. SELECT * FROM users WHERE city LIKE '%che%';
_	Equates to any single character. **Example 1**: Find all users living in cities beginning with any 3 characters, followedby 'chester'. SELECT * FROM users WHERE city LIKE '___chester'
(CHADIST)	Equates to any single character in the list. **Example 1**: Find all users with first names beginning with J, H or M. SELECT * FROM users WHERE first_name LIKE '(jhm]>'; **Example 2**: Find all users with first names beginning letters between A-L. SELECT * FROM users WHERE first_name LIKE '[a-1]>'; **Example 3**: Find all users with first names not ending with letters between n-s. SELECT * FROM users WHERE first_name LIKE '%[!n-s]);

KEYS

FOREIGN KEY



EXAMPLE 1 (MYSQL)

```
CREATE TABLE orders (
id int NOT NULL,
user_id int,
product_id int,
PRIMARY KEY (id),
FOREIGN KEY (user_id) REFERENCES users(id),
FOREIGN KEY (product_id) REFERENCES products(id)
);
```

EXAMPLE 2 (MYSQL)

```
ALTER TABLE orders
ADD FOREIGN KEY (user_id) REFERENCES users(id);
```

PRIMARY KEY

EXAMPLE 1 (MYSQL)

```
CREATE TABLE users (
id int NOT NULL AUTO_INCREMENT,
first_name varchar(255).
last_name varchar(255) NOT NULL,
address varchar(255).
email varchar(255).
PRIMARY KEY (id)
);
```

EXAMPLE 2 (MYSQL)

```
ALTER TABLE users
ADD PRIMARY KEY (first_name);
```

WILDCARDS

DATA TYPE	DESCRIPTION
CREATE INDEX	Creates an index named 'idx_test' on the first_name and surname columns of the users table. In this instance, duplicate values are allowed. CREATE INDEX idx_test ON users (first_name, surname);
CREATE UNIQUE INDEX	Creates an index named "idx_test' on the first name and surname columns of the users table. In this instance, duplicate values are allowed. CREATE UNIQUE INDEX idx_test ON users (first_name, surname);
DROP INDEX	Creates an index named "idx_test' on the first name and surname columns of the users table. In this instance, duplicate values are allowed. ALTER TABLE users DROP INDEX idx_test;

VIEWS

CREATING VIEWS

```
CREATE VIEW priority_users AS
SELECT * FROM users
WHERE country = United Kingdom;
```

```
SELECT * FROM [priority_users);
```

REPLACING VIEWS

```
CREATE OR REPLACE VIEW [priority_users] AS
SELECT * FROM users
WHERE country = 'United Kingdom' OR country="USA';
```

DELETING VIEWS

```
DROP VIEW priority_users;
```

JOINS

ORDERS

id	first_name	Last_name	address	email
1	Lube	Harison	1640, Kjetil Homme ...	Lube@153....
2	Healer	Reynolds	742, Norway, Denmark.	heal132@hot......
3	Simpson	Chekson	7, Nova Scotia.....	Simpson76@....
4	Chekson	Simpson	15, Santo Domingo....	Checkson80@....
5	Oliver	Harison	1640, San Salvador..	oliver5715@....
6	jones	Gabet	598, Caracas, 1010 ...	jones547@....
7	Micheal	Johnson	12, Western Michigan..	Micheal0017@....
8	Thomes	Smith	342, Mary Jones Station	smith098@....
9	Robyn	Gabet	598, Monte, Trigo..	robyn65478@....
10	Byony	Brown	165, First Ave. Usa..	byony8754@....

ORDERS

id	user_id	product_id	agent_logged
1	5	196	0
2	4	32	1
3	6	310	0
4	10	196	1
5	1	67	1
6	1	341	1
7	1	875	0
8	9	3	1
9	5	23	1
10	8	196	1

PROUDCTS

id	name	description	stock_count	price
102	Cartono..	Why	0	14,99
23	Cardbor..	Dec	1	3.49
3	Smart..	NULL	1	24.99
32	Troast 33..	Die	4	09.50
275	A4..	berta	5	4.99
436	Pack of 50.	you	5	12.99
341	Set of 25..	Modo	2	4.99
67	Large Car..	Pack	10	12.99
196	10, XP..	Hay que	10	15.99
310	Set of 35 ..	Cetta	10	2.99

PROUDCTS

id	name	description	stock_count	price
102	Cartono..	Why	0	14,99
23	Cardbor..	Dec	1	3.49
3	Smart..	NULL	1	24.99
32	Troast 33..	Die	4	09.50
275	A4..	berta	5	4.99
436	Pack of 50.	you	5	12.99
341	Set of 25..	Modo	2	4.99
67	Large Car..	Pack	10	12.99
196	10, XP..	Hay que	10	15.99
310	Set of 35 ..	Cetta	10	2.99

```
SELECT orders.id, users.first_name, users.surname, products.name as 'product name
FROM orders
INNER JOIN users on orders.user_id = users.id
INNER JOIN products on orders.product_id = products.id;
```

INNER JOIN RESULT SET

id	first_name	surname	product name
1	Oliver	Harison	10 X Plastic.....
2	Claire	Simpson	TripLast 33....
3	James	Gilbert	StorePac 5.....
4	Bryony	Brown	10 X Plasti.....
5	Luke	Harison	Large Care.....
6	Luke	Harison	Set of 2 S.....
7	Luke	Harison	A4 Storage....
8	Robyn	Gilbert	SmartMo....
9	Oliver	Harrison	Cardboar....
10	Thomas	Smith	10 X Plasti....

DATAQUEST

SQL Cheat Sheet: Fundamentals
Performing calculations with SQL

Performing a single calculation:

```
SELECT 1320+17;
```

Performing multiple calculations:

```
SELECT 1320+17, 1340-3, 7*191, 8022/6;
```

Performing calculations with multiple numbers:

```
SELECT 1*2*3, 1+2+3;
```

Renaming results:

```
SELECT 2*3 AS mult, 1+2+3 AS nice_sum;
```

Selecting tables, columns, and rows:

Remember: The order of clauses matters in SQL. SQL uses the following order of precedence: FROM, SELECT, LIMIT.

Display the whole table:

```
SELECT *
  FROM table_name;
```

Select specific columns from a table:

```
SELECT column_name_1, column_name_2
  FROM table_name;
```

Display the first 10 rows on a table:

```
SELECT *
  FROM table_name;
 LIMIT 10;
```

Adding comments to your SQL queries

Adding single-line comments:

```
-- First comment
SELECT column_1, column_2, column_3 -- Second comment
  FROM table_name; -- Third comment
```

Adding block comments:

```
/*
This comment
spans over
multiple lines
 */
SELECT column_1, column_2, column_3
  FROM table_name;
```

SQL Intermediate:
Joins & Complex Queries

Many of these examples use table and column names from the real SQL databases that learners work with in our interactive SQL courses. For more information, sign up for a free account and try one out!

Joining data in SQL:

Joining tables with INNER JOIN:

```
SELECT column_name_1, column_name_2 FROM table_name_1
INNER JOIN table_name_2 ON table_name_1.column_name_1
          = table_name_2.column_name_1;
```

Joining tables using a LEFT JOIN:

```
SELECT * FROM facts
LEFT JOIN cities ON cities.facts_id = facts.id;
```

Joining tables using a RIGHT JOIN:

```
SELECT f.name country, c.name city
FROM cities c
RIGHT JOIN facts f ON f.id = c.facts;
```

Joining tables using a FULL OUTER JOIN:

```
SELECT f.name country, c.name city
FROM cities c
FULL OUTER JOIN facts f ON f.id = c.facts_id;
```

Sorting a column without specifying a column name:

```
SELECT name, migration_rate FROM FACTS
ORDER BY 2 desc; -- 2 refers to migration_rate column
```

Using a join within a subquery, with a limit:

```
SELECT c.name capital_city, f.name country
FROM facts f
INNER JOIN (
        SELECT * FROM cities
                WHERE capital = 1
                ) c ON c.facts_id = f.id
INNER 10
```

Joining data from more than two tables:

```
SELECT [column_names] FROM [table_name_one]
   [join_type] JOIN [table_name_two] ON [join_constraint]
      [join_type] JOIN [table_name_three] ON [join_constraint]
      ...
      ...
      ...
      [join_type] JOIN [table_name_three] ON [join_constraint]
```

Other common SQL operations:

Combining columns into a single column:

```
SELECT
          album_id,
          artist_id,
          "album id is " || album_id col_1,
          "artist id is " || artist_id col2,
          album_id || artist_id col3
FROM album LIMIT 3;
```

Matching part of a string:

```
SELECT
     first_name,
     last_name,
     phone
FROM customer
WHERE first_name LIKE "%Jen%";
```

Using if/then logic in SQL with CASE:

```
CASE
     WHEN [comparison_1] THEN [value_1]
     WHEN [comparison_2] THEN [value_2]
     ELSE [value_3]
     END
AS [new_column_name]
```

Using the WITH clause:

```
WITH track_info AS
(
     SELECT
          t.name,
          ar.name artist,
          al.title album_name,
     FROM track t
     INNER JOIN album al ON al.album_id = t.album_id
     INNER JOIN artist ar ON ar.artist_id = al.artist_id
)
SELECT * FROM track_info
WHERE album_name = "Jagged Little Pill";
```

Creating a view:

```
CREATE VIEW chinook.customer_2 AS
SELECT * FROM chinook.customer;
```

Dropping a view

```
DROP VIEW chinook.customer_2;
```

Selecting rows that occur in one or more SELECT statements:

```
[select_statement_one]
UNION
[select_statement_two];
```

Selecting rows that occur in both SELECT statements:

```
SELECT * from customer_usa
INTERSECT
SELECT * from customer_gt_90_dollars;
```

Selecting rows that occur in the first SELECT statement but not the second SELECT statement:

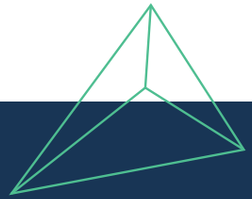
```
SELECT * from customer_usa
EXCEPT
SELECT * from customer_gt_90_dollars;
```

Chaining WITH statements:

```
WITH
usa AS
     (
     SELECT * FROM customer
     WHERE country = "USA"
     ),
last_name_g AS
     (
     SELECT * FROM usa
     WHERE last_name LIKE "G%"
     ),
state_ca AS
     (
     SELECT * FROM last_name_g
     WHERE state = "CA"
     )
SELECT
     first_name,
     last_name,
     country,
     state
FROM state_ca
```

Important Concepts and Resources:
Reserved words

Reserved words are words that cannot be used as identifiers (such as variable names or function names) in a programming language, because they have a specific meaning in the language itself. Here is a list of reserved words in SQL.



SQL Basics Cheat Sheet

LearnSQL.com

AGGREGATION AND GROUPING

GROUP BY **groups** together rows that have the same values in specified columns. It computes summaries (aggregates) for each unique combination of values.

CITY

id	name	country_id
1	Paris	1
101	Marseille	1
102	Lyon	1
2	Berlin	2
103	Hamburg	2
104	Munich	2
3	Warsaw	4
105	Cracow	4

→

CITY

country_id	count
1	3
2	3
4	2

AGGREGATE FUNCTIONS

- **avg(**expr**)** – average value for rows within the group
- **count(**expr**)** – count of values for rows within the group
- **max(**expr**)** – maximum value within the group
- **min(**expr**)** – minimum value within the group
- **sum(**expr**)** – sum of values within the group

EXAMPLE QUERIES

Find out the number of cities:

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

Find out the number of cities with non-null ratings:

```
SELECT COUNT(rating)
FROM city;
```

Find out the number of distinctive country values:

```
SELECT COUNT(DISTINCT country_id)
FROM city;
```

Find out the smallest and the greatest country populations:

```
SELECT MIN(population), MAX(population)
FROM country;
```

Find out the total population of cities in respective countries:

```
SELECT country_id, SUM(population)
FROM city
GROUP BY country_id;
```

Find out the average rating for cities in respective countries if the average is above 3.0:

```
SELECT country_id, AVG(rating)
FROM city
GROUP BY country_id
HAVING AVG(rating) > 3.0;
```

SUBQUERIES

A subquery is a query that is nested inside another query, or inside another subquery. There are different types of subqueries.

SINGLE VALUE

The simplest subquery returns exactly one column and exactly one row. It can be used with comparison operators =, <, <=, >, or >=.

This query finds cities with the same rating as Paris:

```
SELECT name FROM city
WHERE rating = (
    SELECT rating
    FROM city
    WHERE name = 'Paris'
);
```

MULTIPLE VALUES

A subquery can also return multiple columns or multiple rows. Such subqueries can be used with operators IN, EXISTS, ALL, or ANY.

This query finds cities in countries that have a population above 20M:

```
SELECT name
FROM city
WHERE country_id IN (
    SELECT country_id
    FROM country
    WHERE population > 20000000
);
```

CORRELATED

A correlated subquery refers to the tables introduced in the outer query. A correlated subquery depends on the outer query. It cannot be run independently from the outer query.

This query finds cities with a population greater than the average population in the country:

```
SELECT *
FROM city main_city
WHERE population > (
    SELECT AVG(population)
    FROM city average_city
    WHERE average_city.country_id = main_city.country_id
);
```

This query finds countries that have at least one city:

```
SELECT name
FROM country
WHERE EXISTS (
    SELECT *
    FROM city
    WHERE country_id = country.id
);
```

SET OPERATIONS

Set operations are used to combine the results of two or more queries into a single result. The combined queries must return the same number of columns and compatible data types. The names of the corresponding columns can be different.

CYCLING

id	name	country
1	YK	DE
2	ZG	DE
3	WT	PL
...	...	...

SKATING

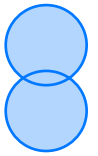
id	name	country
1	YK	DE
2	DF	DE
3	AK	PL
...	...	...

UNION

UNION combines the results of two result sets and removes duplicates. UNION ALL doesn't remove duplicate rows.

This query displays German cyclists together with German skaters:

```
SELECT name
FROM cycling
WHERE country = 'DE'
UNION / UNION ALL
SELECT name
FROM skating
WHERE country = 'DE';
```

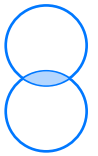


INTERSECT

INTERSECT returns only rows that appear in both result sets.

This query displays German cyclists who are also German skaters at the same time:

```
SELECT name
FROM cycling
WHERE country = 'DE'
INTERSECT
SELECT name
FROM skating
WHERE country = 'DE';
```



EXCEPT

EXCEPT returns only the rows that appear in the first result set but do not appear in the second result set.

This query displays German cyclists unless they are also German skaters at the same time:

```
SELECT name
FROM cycling
WHERE country = 'DE'
EXCEPT / MINUS
SELECT name
FROM skating
WHERE country = 'DE';
```