Some of The Most Important SQL Commands:

* SELECT - extracts data from a database
* SELECT DISTINCT - statement is used to return only distinct (different) values
  + SELECT COUNT(DISTINCT Country) FROM Customers;
* WHERE Clause –
  + <> Not equal. Note: In some versions of SQL this operator may be written as !=
  + SELECT \* FROM Customers WHERE NOT Country='Germany' AND NOT Country='USA';
* LIKE operator –
  + SELECT \* FROM Customers WHERE CustomerName LIKE 'a%'; - selects all customers with a CustomerName starting with "a"
  + SELECT \* FROM Customers WHERE CustomerName LIKE '%a'; - selects all customers with a CustomerName ending with "a"
  + SELECT \* FROM Customers WHERE CustomerName LIKE '%or%'; - selects all customers with a CustomerName that have “or” in any position
  + SELECT \* FROM Customers WHERE CustomerName LIKE '\_r%'; - selects all customers with a CustomerName that have “r” in second position
  + SELECT \* FROM Customers WHERE CustomerName LIKE 'a\_\_%'; - selects all customers with a CustomerName that starts with “a” and are atleast 3 chars in length
  + SELECT \* FROM Customers WHERE ContactName LIKE 'a%o'; - selects all customers with a ContactName that starts with "a" and ends with "o"
  + SELECT \* FROM employees WHERE length(emp\_name)=6 AND emp\_name LIKE '\_\_R%';
  + SELECT \* FROM Customers WHERE City LIKE '[bsp]%'; - selects all customers with a City starting with "b", "s", or "p"
  + SELECT \* FROM Customers WHERE City LIKE '[a-c]%'; - selects all customers with a City starting with "a", "b", or "c"
  + SELECT \* FROM Customers WHERE City LIKE '[!bsp]%'; - select all customers with a City NOT starting with "b", "s", or "p". NOT LIKE ‘[bsp]%’ is also ok.
* IS NULL / IS NOT NULL – identify null and not null values
* IN / NOT IN Keyword - specify multiple values in a WHERE clause
* BETWEEN operator –
  + SELECT \* FROM Products WHERE Price BETWEEN 10 AND 20 AND CategoryID NOT IN (1,2,3);
  + SELECT \* FROM Products WHERE ProductName BETWEEN 'Carnarvon Tigers' AND 'Mozzarella di Giovanni' ORDER BY ProductName;
  + SELECT \* FROM Orders WHERE OrderDate BETWEEN #07/01/1996# AND #07/31/1996#; or SELECT \* FROM Orders WHERE OrderDate BETWEEN '1996-07-01' AND '1996-07-31';
* SQL Aliases –
  + SELECT CustomerID AS ID, CustomerName AS [Customer Name] FROM Customers; or SELECT CustomerID AS ID, CustomerName AS “Customer Name” FROM Customers;
  + SELECT CustomerName, CONCAT(Address , ', ' , PostalCode , ' ,' , City , ', ' , Country AS Address FROM Customers;
  + SELECT o.OrderID, o.OrderDate, c.CustomerName FROM Customers AS c, Orders AS o WHERE c.CustomerName='Around the Horn' AND c.CustomerID=o.CustomerID;
* ORDER BY Keyword –
  + SELECT \* FROM Customers ORDER BY Country ASC, CustomerName DESC;
* GROUP BY Keyword – The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country"
  + SELECT Shippers.ShipperName, COUNT(Orders.OrderID) AS NumberOfOrders FROM Orders LEFT JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID GROUP BY ShipperName; - this will display the number of orders from each shipper.
* HAVING Clause - The HAVING clause was added to SQL because the WHERE keyword cannot be used with aggregate functions like count, sum, min, max and avg.
  + SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders FROM Orders INNER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID WHERE LastName = 'Davolio' OR LastName = 'Fuller' GROUP BY LastName HAVING COUNT(Orders.OrderID) > 25;
* EXISTS Operator - The EXISTS operator is used to test for the existence of any record in a subquery. The EXISTS operator returns TRUE if the subquery returns one or more records.
  + SELECT SupplierName FROM Suppliers WHERE EXISTS (SELECT ProductName FROM Products WHERE Products.SupplierID = Suppliers.supplierID AND Price = 22);
* ANY / ALL / SOME Operator – ANY, ALL and SOME returns a boolean value as a result. ALL is used with SELECT, WHERE and HAVING statements
  + SELECT ProductName FROM Products WHERE ProductID = ANY (SELECT ProductID FROM OrderDetails WHERE Quantity > 1000);
  + SELECT ProductName FROM Products WHERE ProductID = ALL (SELECT ProductID FROM OrderDetails WHERE Quantity = 10);
  + SELECT \* FROM Products WHERE Price > SOME (SELECT Price FROM Products WHERE Price > 20);
* UPDATE - updates data in a database
  + UPDATE Customers SET ContactName = 'Alfred Schmidt', City= 'Frankfurt' WHERE CustomerID = 1;
* DELETE - deletes data from a database
  + DELETE FROM Customers; - delete all records
  + DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';
* INSERT INTO - inserts new data into a database
  + INSERT INTO Customers (CustomerName, City, Country) VALUES ('Cardinal', 'Stavanger', 'Norway');
* LIMIT –
  + SELECT \* FROM Customers LIMIT 3;
* MIN() and MAX() –
  + SELECT MIN(Price) AS SmallestPrice FROM Products;
  + SELECT MAX(Price) AS LargestPrice FROM Products;
* COUNT(), AVG() and SUM() –
  + SELECT COUNT(ProductID) FROM Products;
  + SELECT AVG(Price) FROM Products;
  + SELECT SUM(Quantity) FROM OrderDetails;
* SELECT INTO - The SELECT INTO statement copies data from one table into a new table. IN clause to copy the table into a new table in another database.
  + SELECT \* INTO CustomersBackup2017 FROM Customers; - creates a backup copy of customers
  + SELECT \* INTO CustomersBackup2017 IN 'Backup.mdb' FROM Customers;
  + SELECT INTO can also be used to create a new, empty table using the schema of another. Just add a WHERE clause that causes the query to return no data
  + SELECT \* INTO newtable FROM oldtable WHERE 1 = 0;
* INSERT INTO SELECT - The INSERT INTO SELECT statement copies data from one table and inserts it into another table. The INSERT INTO SELECT statement requires that the data types in source and target tables match.
  + INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country) SELECT SupplierName, ContactName, Address, City, PostalCode, Country FROM Suppliers;
* CASE Expression - The CASE expression goes through conditions and returns a value when the first condition is met (like an if-then-else statement). So, once a condition is true, it will stop reading and return the result. If no conditions are true, it returns the value in the ELSE clause. If there is no ELSE part and no conditions are true, it returns NULL.
  + SELECT CustomerName, City, Country  
    FROM Customers  
    ORDER BY  
    (CASE  
        WHEN City IS NULL THEN Country  
        ELSE City  
    END);
  + SELECT OrderID, Quantity,  
    CASE  
        WHEN Quantity > 30 THEN 'The quantity is greater than 30'  
        WHEN Quantity = 30 THEN 'The quantity is 30'  
        ELSE 'The quantity is under 30'  
    END AS QuantityText  
    FROM OrderDetails;
* IFNULL(), ISNULL(), COALESCE() and NVL() Functions –
  + Return the specified value IF the expression is NULL, otherwise return the expression. Ex- SELECT IFNULL(NULL, "W3Schools.com");
  + The ISNULL() function returns 1 or 0 depending on whether an expression is NULL. Ex- SELECT ISNULL(""); -- 0
  + COALESCE() returns the first non-null value in a list. Ex: SELECT COALESCE(NULL, NULL, NULL, 'W3Schools.com', NULL, 'Example.com');
  + NVL() lets you replace null (returned as a blank) with a string in the results of a query. Ex- SELECT last\_name, NVL(TO\_CHAR(commission\_pct), 'Not Applicable') commission FROM employees WHERE last\_name LIKE 'B%' ORDER BY last\_name;
* Stored procedures – A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again by executing it. We can also use parameters to execute the procedure.
  + CREATE PROCEDURE SelectAllCustomers  
    AS  
    SELECT \* FROM Customers  
    GO; -- EXEC SelectAllCustomers; - to execute it
  + CREATE PROCEDURE SelectAllCustomers @City nvarchar(30), @PostalCode nvarchar(10)  
    AS  
    SELECT \* FROM Customers WHERE City = @City AND PostalCode = @PostalCode  
    GO; -- EXEC SelectAllCustomers @City = 'London', @PostalCode = 'WA1 1DP'; - to execute
* (INNER) JOIN: Returns records that have matching values in both tables
  + SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate FROM Orders INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;
* LEFT (OUTER) JOIN: Returns all records from the left table, and the matched records from the right table
  + SELECT Customers.CustomerName, Orders.OrderID FROM Customers LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID ORDER BY Customers.CustomerName;
* RIGHT (OUTER) JOIN: Returns all records from the right table, and the matched records from the left table
  + SELECT Orders.OrderID, Employees.LastName, Employees.FirstName FROM Orders RIGHT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID ORDER BY Orders.OrderID;
* FULL (OUTER) JOIN: Returns all records when there is a match in either left or right table
  + SELECT Customers.CustomerName, Orders.OrderID FROM Customers FULL OUTER JOIN Orders ON Customers.CustomerID=Orders.CustomerID ORDER BY Customers.CustomerName;
* SELF JOIN: A table is joined with itself.
  + SELECT A.CustomerName AS CustomerName1, B.CustomerName AS CustomerName2, A.City FROM Customers A, Customers B WHERE A.CustomerID <> B.CustomerID AND A.City = B.City ORDER BY A.City;
* UNION / UNION ALL: The UNION operator is used to combine the result-set of two or more SELECT statements.
  + Every SELECT statement within UNION must have the same number of columns
  + The columns must also have similar data types
  + The columns in every SELECT statement must also be in the same order
  + The UNION operator selects only distinct values by default. To allow duplicate values, use UNION ALL
* CREATE DATABASE - creates a new database
  + CREATE DATABASE testDB;
* DROP DATABASE - drops an existing database
  + DROP DATABASE testDB;
  + You need admin priviledges to drop a dabase. Use SHOW DATABASES; to see if its dropped.
* BACKUP DATABASE - creates a full backup of an existing database
  + BACKUP DATABASE testDB  
    TO DISK = 'D:\backups\testDB.bak';
  + A differential back up only backs up the parts of the database that have changed since the last full database backup.
  + BACKUP DATABASE testDB  
    TO DISK = 'D:\backups\testDB.bak'  
    WITH DIFFERENTIAL;
* ALTER DATABASE - modifies a database
  + ALTER DATABASE name [ [ WITH ] option [ ... ] ] where option can be: CONNECTION LIMIT connlimit
* CREATE TABLE - creates a new table
* ALTER TABLE - modifies a table
* DROP TABLE - deletes a table
* CREATE INDEX - creates an index (search key)
* DROP INDEX - deletes an index