FROM: https://www.w3schools.com/sql/sql\_case.asp  
-**SQL** is a standard language for accessing and manipulating databases.   
-SQL stands for Structured Query Language  
-SQL lets you access and manipulate databases  
-To be compliant with the ANSI standard, all SQL’s support at least the major commands (such as SELECT, UPDATE, DELETE, INSERT, WHERE) in a similar manner.   
-**NOTE**: Most of the SQL database programs also have their own proprietary extensions in addition to the SQL standard.  
A picture containing text, indoor, screenshot

Description automatically generatedText

Description automatically generated  
  
**SQL Syntax**  
> Every table is broken up into smaller entities called fields. **A field** is a column in a table that is designed to maintain specific information about every record in the table.   
> **A record**, also called a row, is each individual entry that exists in a table. A record is a horizontal entity in a table.   
> A column is a vertical entity in a table that contains all information associated with a specific field in a table.   
> Keep in Mind That: SQL keywords are NOT case sensitive – **select** is the same as **SELECT**> Semicolon is the standard way to separate each SQL statement in database system that allow more than one SQL statement to be executed in the same call to the server.  
Text

Description automatically generated  
  
>Demo Tables (some)  
Graphical user interface, text, application, chat or text message

Description automatically generated  
  
A screenshot of a computer

Description automatically generated

|  |  |
| --- | --- |
| **SELECT** | > SELECT, is used to select data from a database.  > The data returned is stored in a result table, called the result-set. |
| Graphical user interface, text, application, Teams  Description automatically generated |
| **SELECT DISTINCT** | The **SELECT DISTINCT** statement is used to return only distinct(different) values. |
| Graphical user interface  Description automatically generated with low confidence > **SELECT without DISTINCT** \* The statement selects all (including the duplicates) values from the ‘Country’ column in the ‘Customers’ table.  > **SELECT DISTINCT** (with) \* The following SQL statement selects only the DISTINCT values from the ‘Country’ column in the ‘Customers’ table:  \* The following SQL statement lists the number of different (distinct) customer countries: |
| **WHERE** | > The **WHERE** clause is used to filter records. > It is used to extract only those records that fulfill a specified condition. |
| > It is used to extract only those records that fulfill a specified condition.  Graphical user interface, text, application  Description automatically generated > NOTE: The WHERE clauses is not only used in **SELECT** statements, it is also used in **UPDATE** , **DELETE** , etc. ! (Example) Graphical user interface  Description automatically generated with medium confidence  > Text Fields vs. Numeric Fields > SQL requires single quotes around text values (most database systems will also allow double quotes).  > However, numeric fields should not be enclosed in quotes: A picture containing text  Description automatically generated > Operators in the **WHERE** Clause Graphical user interface, application  Description automatically generated |
| **AND , OR , NOT** | > The WHERE clause can be combined with AND, OR, and NOT operators.  > The AND and OR operators are used to filter based on more than one conditions:  \* The AND operator displays a record if all the conditions separated by AND are TRUE.  \* The OR operator displays a record if any of the conditions separated by OR is TRUE.  > The NOT operator displays a record if the condition(s) is NOT TRUE. |
| Graphical user interface, application, Teams  Description automatically generated > AND example Text  Description automatically generated > OR example Text  Description automatically generated Graphical user interface, text  Description automatically generated with medium confidence > NOT example A picture containing graphical user interface  Description automatically generated > Combining AND, OR and NOT  \* You can also combine the AND, OR and NOT operators.  A picture containing text  Description automatically generated Graphical user interface, text  Description automatically generated > More combinations possible… |
| **ORDER BY  \* DESC** : descending **\* ASC** : ascending | > The **ORDER BY** keyword is used to sort the result-set in ascending or descending order.  > The **ORDER BY** keyword sorts the records in ascending order by default. To sort the records in descending order, use the **DESC** keyword. |
| Graphical user interface  Description automatically generated with medium confidence > Examples Graphical user interface  Description automatically generated  \* sorted DESCENDING by the ‘Country’ column:  Graphical user interface, application  Description automatically generated  > Multiple Columns Graphical user interface  Description automatically generated > sorted ascending by the ‘Country’ and descending by the ‘customername’ column: Graphical user interface  Description automatically generated with low confidence |
| **INSERT INTO** | >The INSERT INTO statement is used to insert records in a table.  >It is possible to write the INSERT INTO statement in two ways: |
| >(1) Specify both the column names and the values to be inserted: Graphical user interface, text, application, Word  Description automatically generated >(2) If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. Here, the INSERT INTO syntax would be as follows: Graphical user interface, text  Description automatically generated with medium confidence   >Example    >Inserting Data Only in Specified Columns > It is also possible to only insert data in specific columns. |
| **NULL** values  \* testing null values **IS NOT NULL** or **NOT NULL IS NULL** | >NULL values – filed with no value >If a field in a table is optional, it is possible to insert a new record or update a record without adding a value to this field. Then, the field will be saved with a **NULL** value.   Testing >It is not possible to test for **NULL** values with comparison operators, such as =, <, or >.  >We will have to use the **IS NULL** and **IS NOT NULL** operators instead. |
| >IS NULL   >IS NOT NULL |
| **UPDATE** | >The **UPDATE** statement is used to modify the existing records in a table.  >Note: Be careful when updating records in a table! Notice the **WHERE** clause in the **UPDATE** statement. The **WHERE** clause specifies which record(s) that should be updated. If you omit the **WHERE** clause, all records in the table will be updated! |
| >It is the **WHERE** clause that determines how many records will be updated.  >The following SQL statement will update the ContactName to ‘Juan’ for all records where country is ‘Mexico’.     >Update Warning: Be careful when updating records. If you omit the **WHERE** clause, ALL records will be updated! |
| **DELETE** | >The **DELETE** statement is used to delete existing records in a table. >Note: Be careful when deleting records in a table! Notice the **WHERE** clause in the **DELETE** statement. The **WHERE** clause specifies which record(s) should be deleted. If you omit the WHERE clause, all records in the table will be deleted! |
| >The following statement deletes the customers ‘Alfreds Futterkiste’ from the ‘Customers’ table:   > Delete All Records > It is possible to delete all rows in a able without deleting the table. This means that the table structure, attributes, and indexes will be intact:   >The following SQL statement deletes all rows in the ‘Customers’ table, without deleting the table: |
| **SELECT TOP** | >The **SELECT TOP** clause is used to specify the number of records to return. >The **SELECT TOP** clause is useful on large tables with thousands of records. Returning a large number of records can impact performance.  >Note: Not all databases systems support the **SELECT TOP** clause. Mysql supports the LIMIT clause to select a limited number of records, while Oracle uses **FETCH FIRST n ROW ONLY** and **ROWNUM**. |
| >Example  -SQL TOP, LIMIT and FETCH FIRST examples >The following SQL stamen selects the first three records from the table above.(for SQL Server/MC Access)  >The following SQL statement shows the equivalent example for MySQL  >The following SQL statement shows the equivalent example for Oracle:  >The following SQL statement selects the first 50% of the records from the "Customers" table (for SQL Server/MS Access):  TOP PERCENT example >The following SQL statement selects the first 50% of the records from the "Customers" table (for SQL Server/MS Access):  >The following SQL statement shows the equivalent example for Oracle:   ADD a WHERE CLAUSE >The following SQL statement selects the first three records from the "Customers" table, where the country is "Germany" (for SQL Server/MS Access):  >The following SQL statement shows the equivalent example for MySQL:  >The following SQL statement shows the equivalent example for Oracle: |
| **MIN()** and **MAX()** | >The MIN() function return the smallest value of the selected column.  >The MAX() function returns the largest value of the selected column. |
|  | Examples >The following SQL statement finds the price of the cheapest product:  >The following SQL statement finds the price of the most expensive product: |
| **COUNT() , AVG() ,** and **SUM()** | >The COUNT() function returns the number of rows that matches a specified criterion.  >The AVG() function returns the average value of a numeric column.  >The SUM() function returns the total sum of a numeric column. |
| >COUNT()  >AVG()  >SUM()   Example    NULL values are ignored. |
| **LIKE** operator | >The LIKE operator is used in a WHERE clause to search for a specified pattern in a column. >There are two wildcards often used in conjunction with the LIKE operator:  \* The percent sign(5) represents zero, one or multiple characters.   \* The underscore sign(\_) represents one, single character. |
| More Operators   Examples >The following SQL statement selects all customers with a CustomerName starting with "a":  >The following SQL statement selects all customers with a CustomerName ending with "a":  >The following SQL statement selects all customers with a CustomerName that have "or" in any position:  >The following SQL statement selects all customers with a CustomerName that have "r" in the second position:  >The following SQL statement selects all customers with a CustomerName that starts with "a" and are at least 3 characters in length:  >The following SQL statement selects all customers with a ContactName that starts with "a" and ends with "o":  >The following SQL statement selects all customers with a CustomerName that does NOT start with "a": |
| Wildcard Characters | >A wildcard character is used to substitute one ore more characters in a string.  >Wildcard characters are used with the LIKE operator. The LIKE operator is used in a WHERE clause to search for a specified pattern in a column. |
| >All the wildcards can also be used in combinations!   > Review this as necessary: <https://www.w3schools.com/sql/sql_wildcards.asp> |
| **IN** | >The IN operator allows you to specify multiple values in a WHERE clause.  >The IN operator is a shorthand for multiple OR conditions. |
| or    Example  >The following SQL statement selects all customers that are located in "Germany", "France" or "UK":  >The following SQL statement selects all customers that are NOT located in "Germany", "France" or "UK":  >The following SQL statement selects all customers that are from the same countries as the suppliers: |
| **BETWEEN** | >The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates. >The BETWEEN operator is inclusive: begin and end values are included. |
| Example  >The following SQL statement selects all products with a price between 10 and 20:  >To display the products outside the range of the previous example, use NOT BETWEEN: |
| Aliases | >SQL Aliases are used to give a table, or a column in a table, a temporary name. >Aliases are often used to make column names more readable. >An alias only exists for the duration of that query. >An alias is created with the AS keyword. |
| > Alias Column Syntax   >Alias Table Syntax   Example  >The following SQL statement creates two aliases, one for the CustomerID column and one for the CustomerName column  >The following SQL statement creates two aliases, one for the CustomerName column and one for the ContactName column. Note: It requires double quotation marks or square brackets if the alias name contains spaces:  >The following SQL statement creates an alias named "Address" that combine four columns (Address, PostalCode, City and Country):  > To get the SQL statement above to work in MySQL use the following:   Alias for Tables Example >The following SQL statement selects all the orders from the customer with CustomerID=4 (Around the Horn). We use the "Customers" and "Orders" tables, and give them the table aliases of "c" and "o" respectively (Here we use aliases to make the SQL shorter): >The following SQL statement is the same as above, but without aliases:    Aliases cam be useful when:  \* There are more than one table involved in a query.   \* Functions are used in the query.   \* Column names are big or not very readable.   \* Two or more columns are combined together. |
| **JOIN** | >A JOIN clause is used to combine rows from two or more tables, based on a related column between them. |
| Example   >The relationship between the two tables above is the ‘CustomerID’ column. >Then, we can create the following SQL statement(that contains INNER JOIN), that selects records that have matching values in both tables:  this will produce something like this: |
| **INNER JOIN** | >The INNER JOIN keyword selects records that have matching values in both tables. |
| Example   >The following SQL statement selects all orders with customer information:  >Note: The INNER JOIN keyword selects all rows from both tables as long as there is a match between the columns. If there are records in the ‘Orders’ table that do not have matches in ‘Customers’, these orders will not be shown!  >JOIN three tables |
| **LEFT JOIN** | >The LEFT JOIN keyword returns all records the left table(table1), and the matching records from the right table(table2). The result is 0 records from the right side, if there is no match. |
| >Note: In some databases LEFT JOIN is called LEFT OUTER JOIN.  >Using the same above Demo Example Data   >Note: The LEFT JOIN keyword returns all records from the left table(Customers), even if there are no matches in the right table(Orders). |
| **RIGHT JOIN** | >The RIGHT JOIN keyword returns all records from the right table(table2), and the matching records from the left table(table1). The results is 0 records from the left side, if there is no match. |
| >Note: In some databases RIGHT JOIN is called RIGHT OUTER JOIN.   Example >Using a similar database structure as above:   >The RIGHT JOIN keyword returns all records from the right table(Employees), even if there are no matches in the left table(Orders). |
| **FULL JOIN** | >The FULL OUTER JOIN keyword returns all records when there is a match in left(table1) or right(table2) table records.  >Tip: FULL OUTER JOIN and FULL JOIN are the same. |
| >FULL OUTER can potentially return very large result-sets!  Example   >A selection from the result set may look like this:   >The FULL OUTER JOIN keyword returns all matching records from both tables whether the other tables matches or not. So, if there are rows in ‘Customers’ that do not have matches in ‘Orders’, or if there are rows in “Orders’ that do not have matches in ‘Customers’, those rows will be listed as well. |
| **SELF JOIN** | >A SELF JOIN is a regular join, but the table is joined with itself. |
| >T1 and T2 are different table aliases for the same table.  Example > The following statement matches customers that are from the same city: |
| **UNION** | >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. |
| UNION syntax  UNION ALL syntax  >The UNION operator selects only distinct values by default. To allow duplicates values, use UNION ALL. > Note: The columns names in the result-set are usually equal to the column names in the first SELECT.    UNION example >The following statement returns the cities(only distinct values) from both the ‘Customers’ and the ‘Suppliers’ table:  >Note: If some customers or suppliers have the same city, each city will only be listed once, because UNION selects only distinct values. Use UNION ALL to also select duplicate values!  UNION ALL example >The following SQL statement returns the cities (duplicate values also) from both the "Customers" and the "Suppliers" table:   UNION with WHERE >The following statement returns the German cities(only distinct values) from both the ‘Customers’ and the ‘Suppliers’ table:   UNION ALL with WHERE >The following SQL statement returns the German cities (duplicate values also) from both the "Customers" and the "Suppliers" table:   Another UNION example >The following statement lists all customers and suppliers:  >Notice the "AS Type" above - it is an alias. SQL Aliases are used to give a table or a column a temporary name. An alias only exists for the duration of the query. So, here we have created a temporary column named "Type", that list whether the contact person is a "Customer" or a "Supplier". |
| **GROUP BY** | >The GROUP BY statement groups rows that have the same values into summary rows, like ‘find the number of customers in each country’ >The GROUP BY statement is often used with aggregate functions(COUNT(), MAX(), MIN(), SUM(), AVG()) to group the result-set by one or more columns. |
| Example  > The following SQL statement lists the number of customers in each country:  >The following SQL statement lists the number of customers in each country, sorted high to low:    >The following SQL Statement lists the number of orders sent by each shipper: |
| **HAVING** | >The HAVING clause was added to SQL because the WHERE keyword cannot be used with aggregate functions. |
| Example >Using similar demo database as above >The following SQL statement lists the number of customers in each country. Only include countries with more than 5 customers:  >The following SQL statement lists the number of customers in each country, sorted high to low(Only include countries with more than 5 customers):    >The following SQL statement lists the employees that have registered more than 10 orders:  >The following SQL statement lists if the employees ‘Davolio’ or ‘Fuller’ have registered more than 25 orders: |
| **EXISTS** | >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 ore more records. |
| Example  >The following SQL statement returns TRUE and lists the suppliers with a product price less than 20:  >The following statement returns TRUE and lists the suppliers with a produce price equal to 22 |
| **ANY , ALL** | >The ANY and ALL operators allow you to perform a comparison between a single column values and a range of other values.   >The ANY operator:  \* Returns a Boolean values as a result  \* Returns TRUE if ANY of the subquery values meet the condition >ANY means that the condition will be true if the operation is true for any of the values in the range.   >The ALL operator:  \*Returns a Boolean values as a result  \*Returns TUE if ALL of the subquery values meet the condition  \*Is used with SELECT, WHERE, and HAVING statements >All means that the condition will be true if the operation is true for all values in the range. |
| ANY  > Note: The operator must be a standard comparison operator (=, <>, !=, >, >=, <, or <=).    > Note: The operator must be a standard comparison operator (=, <>, !=, >, >=, <, or <=).  Example   ANY > The following SQL statement lists the ProductName if it finds ANY records in the OrderDetails table has Quantity equal to 10 (this will return TRUE because the Quantity column has some values of 10):   > The following SQL statement lists the ProductName if it finds ANY records in the OrderDetails table has Quantity larger than 99 (this will return TRUE because the Quantity column has some values larger than 99):   > The following SQL statement lists the ProductName if it finds ANY records in the OrderDetails table has Quantity larger than 1000 (this will return FALSE because the Quantity column has no values larger than 1000):    ALL > The following SQL statement lists ALL the product names:   > The following SQL statement lists the ProductName if ALL the records in the OrderDetails table has Quantity equal to 10. This will of course return FALSE because the Quantity column has many different values (not only the value of 10): |
| **SELECT INTO** | >The SELECT INTO statement copies data from one table into a new table. |
| >The new table will be created with the column-names and types as defined in the old table. You can create new column names using the AS clause.  Examples >The following SQL statement creates a backup copy of Customers:   >The following SQL statement uses in the IN clause to copy the table into a new table in another database:   >The following SQL statement copies only a few columns into a new table:   >The following SQL statement copies only the German customers into a new table   >The following SQL statement copies data from more than one table into a new table:   > Tip: 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: |
| **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 matches. >Note: The existing records in the target table are unaffected. |
| Examples  > The following SQL statement copies "Suppliers" into "Customers" (the columns that are not filled with data, will contain NULL):    > The following SQL statement copies "Suppliers" into "Customers" (fill all columns):  > The following SQL statement copies only the German suppliers into "Customers": |
| **CASE** | > The CASE statement 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. |
| Examples   >The following SQL goes through conditions and returns a value when the first condition is met:   >The following SQL will order the customers by City. However, if City is NULL, then order by Country: |
| **NULL** functions  **\* IFNULL()  \* ISNULL()  \* COALESCE()  \* NVL()** | >Review this section as necessary >https://www.w3schools.com/sql/sql\_isnull.asp |
|  |
| **Stored Procedures** > This is an awesome way to save time, by constructing one single procedure and applying it multiple times if necessary. | >A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again.  >So if you have an SQL query that you write over and over again, save it as a stored procedure, and then just call it to execute it.  >You can also pass parameters to a stored procedure, so that the stored procedure can act based on the parameter value(s) that is passed. |
| Examples  Stored Procedure Example >The following SQL statement creates a stored procedure named ‘SelectAllCustoemrs’ that selects all records from the ‘Customers’ table:  Stored Procedure With One Parameter>The following SQL statement creates a stored procedure that selects Customers from a particular City from the ‘Customers’ table: Stored Procedure With Multiple Parameters >Setting up multiple parameters is very easy. Just list each parameter and the data type separated by a comma as shown below.  >The following SQL statement creates a stored procedure that selects Customers from a particular City with a particular PostalCode from the "Customers" table: |
| **Comments** | >Comments are used to example sections of SQL statements, or to prevent execution of SQL statements.  Single-line Comments   Example   Multiple-line Comments |
| **Operators** |  |
| |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| | |  |
| SQL Database | |
| |||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||| | |
| **CREATE DATABASE** | >The CREATE DATABASE statement is used to create a new SQL database. |
| Example >The following SQL statement creates a database called ‘testDB’:  >Tip: Make sure you have admin privilege before creating any database. Once a database is created, you can check in in the list of databases with the following SQL Command:   SHOW DATABASE; |
| **DROP DATABASE** | >The DROP DATABASE statement is used to drop an existing SQL database. |
| >Note: Be careful before dropping a database. Deleting a database will result in loss of complete information stored in the database!  Example  >Tip: Make sure you have admin privilege before dropping any database. Once a database is dropped, you can check it in the list of databases with the following SQL statement:  SHOW DATABASES; |
| **BACKUP DATABASE** | >The BACKUP DATABASE statement is used in SQL Server to create a full back up of an existing SQL database. |
| The BACKUP WITH DIFFERENTIAL Statement > A differential back up only backs up the parts of the database that have changed since the last full database backup:   Example >The following SQL statement creates a full back up of the existing database "testDB" to the D disk:  >Tip: Always back up the database to a different drive than the actual database. Then, if you get a disk crash, you will not lose your backup file along with the database.  BACKUP WITH DIFFERENTIAL Example >The following SQL statement creates a differential back up of the database ‘testDB’  >Tip: A differential back up reduces the back up time(since only the changes are backed up). |
| **CREATE TABLE** | >The CREATE TABLE statement is used to create a new table in a database. |
| >The column parameters specify the names of the columns of the table.  >The datatype parameter specifies the type of data the column can hold (e.g. varchar, integer, date, etc.). >Tip: For an overview of the available data types. Check this page out: <https://www.w3schools.com/sql/sql_datatypes.asp>  Example >The following example creates a table called ‘Person’ that contains five columns: PerdonID, LastName, FirstName, Address, and City:  (Output)  >Tip: The empty ‘Persons’ table can now be filled with data with INSERT INTO statement.  Creating a Table using Another Table >A copy of an existing table can also be created using CREATE TABLE.  >The new table gets the same column definitions. All columns or specific columns can be selected.  >If you create a new table using an existing table, the new table will be filled with the existing values from the old table.   Example > The following SQL statement creates a new table called ‘TestTables’ (which is a copy of the ‘Customers’ table): |
| **DROP TABLE** | >The DROP TABLE statement is used to drop an existing table in a database. |
| >Note: Be careful before dropping a table. Deleting a table will result in a loss of complete information stored in the table!  Example >The following statement drops the existing table ‘Shippers’:  >The TRUNCATE TABLE statement is used to delete the data inside a table, but not the table itself. |
| **ALTER TABLE** | >The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.  >The ALTER TABLE statement is also used to add and drop various constraints on an existing table. |
| Adding Column  Example >The following adds an ‘Email’ column   Dropping Column  >The following deltes the ‘Email’ column from the ‘Customers’ table:   Altering/Modifying Columns   Example  >We want to add a column named ‘DateOfBirth’ in the ‘Persons’ table. >We use the following SQL statement:   >>I skipped some parts of this. Please review the following as necessary: https://www.w3schools.com/sql/sql\_alter.asp |
| Creating Constraints | >SQL constraints are used to specify rules for data in a table. >Constraints can be specified when the table is created with the CREATE TABLE statement, or after the table is created with the ALTER TABLE statement. |
| >SQL constraints are used to specify rules for the data in a table. >Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted. >Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table. |
| **NOT NULL** Constraint | >By default, a column can hold NULL values.  >The NOT NULL constraint enforces a column to NOT accept NULL values.  >This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field. |
| >The following SQL ensures that the ‘ID’, ‘’Lastname’, and ‘Firstname’ columns will NOT accept NULL values when the ‘Persons’ table is created:   NOT NULL on ALTER TABLE >To create A NOT NULL constraint on the ‘Age’ column when the ‘Persons’ table is already created, use the following SQL: |
| **UNIQUE** Constraint | >The UNIQUE constraint ensures that all values in a column are different. >Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns. >A PRIMARY KEY constraint automatically has a UNIQUE constraint. >However, you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table. |
| UNIQUE Constraint on CREATE TABLE >The following SQL creates a UNIQUE constraint on the ‘ID’ column when the ‘Persons’ table is created:   UNIQUE Constraint on ALTER TABLE >To create a UNIQUE constraint on the ‘ID’ column when the table is already created, use the following:   DROP a UNIQUE Constraint >To drop a UNIQUE constraint, use the following SQL: |
| **PRIMARY KEY** Constraint | >The PRIMARY KEY constraint uniquely identifies each record in a table. >Primary keys must contain UNIQUE values, and cannot contain NULL values.  >A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields). |
| > Note: In the example there is only one PRIMARY KEY (Pk\_Person). However, the VALUE of the primary key is made up of TWO COLUMNS (ID + LastName)  Primary KEY on ALTER TABLE >To create a PRIMARY KEY constraint on the ‘ID’ column when the table is already created, use the following SQL:   DROP a PRIMARY KEY Constraint  Resources 1. <https://stackoverflow.com/questions/840162/should-each-and-every-table-have-a-primary-key>  2. <https://www.sqlshack.com/learn-sql-primary-key/> |
| **FOREIGN KEY** Constraint | > The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.  >A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the PRIMARY KEY in another table.  >The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table. |
| > Notice that the "PersonID" column in the "Orders" table points to the "PersonID" column in the "Persons" table.  > The "PersonID" column in the "Persons" table is the PRIMARY KEY in the "Persons" table.  > The "PersonID" column in the "Orders" table is a FOREIGN KEY in the "Orders" table.  > The FOREIGN KEY constraint prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the parent table.  FOREIGN KEY on CREATE TABLE > The following SQL creates a FOREIGN KEY on the ‘PersonID’ column when the ‘Orders’ table is created:   FOREIGN KEY on ALTER TABLE   DROP a FOREIGN KEY Constraint   > <https://www.youtube.com/watch?v=obVZhLu62pk> |
| **CHECK** Constraint | > The CHECK constraint is used to limit the value range that can be placed in a column.  > If you define a CHECK constraint on a column it will allow only certain values for this column.  > If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row |
| CHECK on CREATE TABLE >The following SQL creates a CHECK constraint on the "Age" column when the "Persons" table is created. The CHECK constraint ensures that the age of a person must be 18, or older:   CHECK on ALTER TABLE  DROP a CHECK Constraint |
| **DEFAULT** Constraint | > The DEFAULT constraint is used to set a default value for a column. > The default value will be added to all new records, If no other values is specified. |
| DEFAULT on CREATE TABLE   DEFAULT on ALTER TABLE   DROP a DEFAULT Constraint |
| **CREATE INDEX** Statement | > The CREATE INDEX statement is used to create indexes in tables.  > Indexes are used to retrieve data from the database more quickly than otherwise. The users cannot see the indexes, they are just used to speed up searches/queries.  > Note: Updating a table with indexes takes more time than updating a table without (because the indexes also need an update). So, only create indexes on columns that will be frequently searched against. |
| >Creates an Index on a table. Duplicate values allowed:  >Create unique index on a table. Duplicate values are not allowed:  >Note: The syntax for creating indexes varies among different databases. Therefore: Check the syntax for creating indexes in your database.  Example >The SQL statement below creates an index named ‘idx\_lastname’ on the ‘LastName’ column in the ‘Persons’ table:   DROP INDEX Statement > The DROP INDEX statement is used to delete an index in a table: |
| **AUTO** INCREMENT | >Auto-increment allows a unique number to be generated automatically when a new record is inserted into a table. >Often this is the primary key field that we would like to be created automatically every time a new record is inserted. |
| >The following SQL statement defines the ‘Personid’ column to be auto-increment primary key field in the ‘Persons’ table:   >MySQL uses the AUTO\_INCREMENT keyword to perform an auto-increment feature.  >By default, the starting value for AUTO\_INCREMENT is 1, and it will increment by 1 for each new record. >To let the AUTO\_INCREMENT sequence start with another value, use the following SQL stamen:    >MySQL uses the AUTO\_INCREMENT keyword to perform an auto-increment feature.  >By default, the starting value for AUTO\_INCREMENT is 1, and it will increment by 1 for each new record.  >To let the AUTO\_INCREMENT sequence start with another value, use the following SQL statement:    >To insert a new record into the ‘Persons’ table, we will NOT have to specify a value for the ‘Persoid’ column (a unique value will be added automatically).   > The SQL statement above would insert a new record into the "Persons" table. The "Personid" column would be assigned a unique value. The "FirstName" column would be set to "Lars" and the "LastName" column would be set to "Monsen".  > The following SQL statement defines the "Personid" column to be an auto-increment primary key field in the "Persons" table:   >The MS SQL Server uses the IDENTITY keyword to perform an auto-increment feature.  >In the example above, the starting value for IDENTITY is 1, and it will increment by 1 for each new record.  >Tip: To specify that the "Personid" column should start at value 10 and increment by 5, change it to IDENTITY(10,5).  > To insert a new record into the "Persons" table, we will NOT have to specify a value for the "Personid" column (a unique value will be added automatically):   > The SQL statement above would insert a new record into the "Persons" table. The "Personid" column would be assigned a unique value. The "FirstName" column would be set to "Lars" and the "LastName" column would be set to "Monsen".  >>>Syntax for Access and Syntax for Oracle.  >>>Review as necessary: https://www.w3schools.com/sql/sql\_autoincrement.asp |
| **DATES** | >The most difficult part when working with dates is to be sure that the format of the date you are trying to insert, matches the format of the date column in the database.  >As long as your data contains only the date portion, your queries will work as expected. However, if a time portion is involved, it gets more complicated. |
|  |
| **VIEW** | >In SQL, a view is a virtual table based on the result-set of an SQL statement.  >A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.  >You can add SQL statements and functions to a view and present the data as if the data were coming from one single table.  >A view is created with the CREATE VIEW statement. |
| >Examples: <https://www.w3schools.com/sql/sql_view.asp>  >Updating a View >A view can be updated with the CREATE OR REPLACE VIEW statement.      >Dropping a View >A view is deleted with the DROP VIEW statement. |
| **SQL Injection** \* Review this as necessary \* This includes going over a lot of detail. Beware! | <https://www.w3schools.com/sql/sql_injection.asp> |
| SQL Hosting | <https://www.w3schools.com/sql/sql_hosting.asp> |
| **Data Types** | <https://www.w3schools.com/sql/sql_datatypes.asp> |
|  |  |

**Fill out you SQL Knowledge – Intro**  
(<https://www.youtube.com/watch?v=SMO-I2YhOdU&list=PLfycUyp06LG_WeMMfUE9jfT7oxD8stRUA&index=16> )  
> **SELECT**  
A picture containing website

Description automatically generatedGraphical user interface

Description automatically generated with low confidence> You can also add, subtract, etc.  
Table

Description automatically generatedTable

Description automatically generated> Selecting columns based on criteria. The ‘Then’ and ‘Else’ is very similar to the if else statements in R/Python.   
> A more complex query, with the same function.   
Table

Description automatically generated  
  
  
> **COALESCE( ) or IFNULL( )** ~ Similar functionalities, that deal with NULL values.   
Table

Description automatically generatedTable

Description automatically generated  
> <https://www.w3schools.com/sql/func_sqlserver_coalesce.asp>  
   
  
> Reference the previous row immediately, or column?  
Table

Description automatically generated  
  
> SUM()  
> COUNT()  
> COUNT( distinct ‘column1’)   
>STDEV  
  
> WHERE clause (Filtering the Data)  
Diagram

Description automatically generated with low confidence  
  
  
  
**SQL Server Programming Part 1 – Stored Procedure Basics**  
> <https://www.youtube.com/watch?v=fjNsRV4zLdc&list=PLfycUyp06LG_WeMMfUE9jfT7oxD8stRUA&index=18>  
> Stored Procedure Basics : Grouped SQL queries under the same heading. Useful and efficient.  
A picture containing graphical user interface

Description automatically generated  
>The above is a stored procedure call. It executes a set of SQL code to get results.  
Graphical user interface, text, application

Description automatically generated  
**Creating Procedure**  
CREATE PROC  
AS   
BEGIN  
sql cod ....  
END  
>Using the BEGIN and END are not necessary but are good practice to ensure that the block you want in the procedure is being executed.  
  
> USE statement. This can be further incorporated. Review as necessary. Not necessary to included it, but its good practice to do so.  
Graphical user interface, text, application

Description automatically generated  
> Making sure your stored procedure was indeed saved and written properly. Check it works.

Graphical user interface, text

Description automatically generated  
  
  
> Executing the Stored procedure:  
> Normally this:  
  
  
> Modifying a Stored Procedure:  
\* You can simply edit the Stored Procedure script, but instead of using CREATE PROC. You use ALTER PROC.  
Graphical user interface, text, application

Description automatically generated  
  
> Deleting a Stored Procedure.   
\* You can simply find in the object file explore and delete it or DROP PROC.  
Graphical user interface

Description automatically generated with medium confidence  
  
  
  
**SQL Server Programming Part 2 -Stored Procedure Parameters**  
(<https://www.youtube.com/watch?v=Vs-atxMs4mw&list=PLfycUyp06LG_WeMMfUE9jfT7oxD8stRUA&index=19>)   
> What are Parameters?  
> Creating Parameters?  
> Executing Procedures with Parameters  
> Optional Parameters and Default Values  
> Using Stored Procedures in Other Applications  
  
  
  
  
> **Creating Parameters**  
Graphical user interface, text, application

Description automatically generated  
> Adding the **Begin** and **End** is useful for multiple parameters. It’s a good habit to use them.   
> Specified Parameter must always begin with **@**)   
 - You must also specify the type of data. In the above case INT was used.  
  
> **Using a parameter in the WHER Clause**  
Graphical user interface, text, application

Description automatically generated  
  
Graphical user interface, application, Word

Description automatically generated  
  
> When using the stored procedure with parameters. You must enter parameter criteria. As functions to in normal programming languages.   
Graphical user interface, table

Description automatically generated with medium confidence  
  
> **Adding Multiple Parameters**  
Graphical user interface, text, application

Description automatically generated  
> How you organize your parameters is your choice. Just make sure your organize them adequately and consistently.   
Graphical user interface

Description automatically generated  
  
> Using Named Parameters  
Text

Description automatically generated  
> You can also use this format when executing your stored parameters  
  
> **Creating a Text Parameter**  
Text, timeline

Description automatically generated  
> % % are wildcard characters  
Text

Description automatically generated  
> Adding the @Title parameter into the WHERE clause, so that use can input anything they want to that parameter.  
Text

Description automatically generated  
  
> **Creating Optional Parameters**  
> Creating the parameters used above, as optional!  
> Assigning Default Values to Parameters  
(Two Options) (Quite Simple, just like I do In R in Python)  
1. Add a default value, so that parameters become an optional parameter  
Text

Description automatically generated  
  
  
  
  
  
  
2.   
Text

Description automatically generated  
> The only downside to using the NULL to make the parameters optional, is that you have to modify your WHERE clause as such.   
Graphical user interface, text, application

Description automatically generated  
> In the long run, this will be more proper. Since you don’t have to input a default value. Which can sometimes be uncertain.  
  
  
**SQL – Using Variables**  
(<https://www.youtube.com/watch?v=NmYaOlcbfZM&list=PLfycUyp06LG_WeMMfUE9jfT7oxD8stRUA&index=20> )   
> What are Variables?  
> Declaring Variables.  
> Assigning a Value to a Variable.  
> Referring to a Variable in a Query.  
  
> Storing Query Results in Variables  
> Displaying the Vale of a Variable  
> Reading a Record Into Variables  
> Accumulating Values in Variables  
> Global Variables  
  
  
  
  
  
  
  
  
  
  
> **Declaring a Variable**  
Graphical user interface, text, application

Description automatically generated  
  
> **Assigning a Variable**  
Graphical user interface, text, application

Description automatically generated  
Graphical user interface, text, application

Description automatically generated  
  
  
  
  
  
  
> Storing a Query Result in a Variable  
Graphical user interface, text, application

Description automatically generated  
> Using the SELECT query to select the values desired, instead of a fixed value(s). Doing this, you select all the counts from that column.   
  
> Selecting the Value in a Variable  
Graphical user interface, text

Description automatically generated  
Graphical user interface, text

Description automatically generated  
  
  
  
  
  
  
  
  
  
  
  
  
> Using the PRINT statement instead!!!!  
Graphical user interface, text

Description automatically generated  
Graphical user interface, text, application

Description automatically generated  
> Using the PRINT and CAST(concatenate) statements!!!  
  
Text

Description automatically generated  
  
  
  
  
  
  
  
  
  
> Displaying the results (Review this as necessary, as it is a little bit confusing)   
Graphical user interface, text, application, email

Description automatically generated  
  
> Accumulating Values in Variables  
Graphical user interface, text, application

Description automatically generated  
\* List of all the actors born in 1970.   
Graphical user interface, text, application

Description automatically generated  
\* The above syntax provides you with the actor names separated. Using the CHAR() commands. Interesting and helpful when necessary to do this!  
  
  
  
  
  
  
  
  
  
> Global variables  
> Built in global/system variables!  
> Helpful built-in variables  
Graphical user interface, text, application

Description automatically generatedGraphical user interface

Description automatically generated with medium confidence  
  
  
**Output Parameters & Return Values**  
(<https://www.youtube.com/watch?v=GvRv4V-AK70&list=PLfycUyp06LG_WeMMfUE9jfT7oxD8stRUA&index=21> )  
> Recap of Input Parameters  
> Defining Output Parameters  
> Getting the Result of an Output Parameter  
> Using Return Values in Stored Procedures  
  
>>> Skipped this one. Review this as necessary for now.   
  
  
**Null or NOT NULL constraint when Create Table – Assignment 02   
& BRACKETS**  
(<https://www.tutorialspoint.com/sql/sql-not-null.htm> )   
> By default, a column can hold NULL values. If you do not want a column to have a NULL value, then you need to define such a constraint on this column specifying that NULL is now not allowed for that column.  
> A NULL is not the same as no data, rather, it represents unknown data.  
Graphical user interface, text, application

Description automatically generated  
  
  
  
  
  
  
> **SQUARE BRACKETS**  
(<https://newbedev.com/what-is-the-use-of-the-square-brackets-in-sql-statements> )   
\* The brackets are required if you use keywords or special chars in the column names or identifiers. You could name a column [First Name] (with a space)--but then you'd need to use brackets every time you referred to that column.  
Graphical user interface, text

Description automatically generated with medium confidence