Structured Query Language

## Intro

SQL – Structured Query Language.

* Let’s you access and manipulate databases.
* Is an ANSI/ISO standard

SQL can:

* execute queries against a database
* retrieve data from a database
* insert and update records in a database
* delete records from a database
* create new databases
* create new tables in a database
* create views in a database
* create stores procedures in a database
* set permissions on tables, procedures and views.

Major commands:

* SELECT
* UPDATE
* DELETE
* INSERT
* WHERE

RDBMS:

* Stands for Relational Database Management System.
* Is the basis of SQL
* The data in RDBMS is stored database objects called tables
* Table is collection of data entries and consists of columns and rows.
* Every table is broken into smaller entities called fields -> columns that is designed to maintain specific information about every record in the table. Is a vertical entity in a table.
* Record -> rows, each individual entry that exists in a table. Is a horizontal entity in a table.

## SQL Syntax

Database Tables:

* Contains one or more tables.
* Each table is identified by name.
* Each table contains records (row) with data.

SQL Statements:

* SELECT \* FROM Customers – select all the records in the Customers table.

Semicolon after SQL Statements:

* Standard way to separate each SQL statements in a database system so that it allows more than one SQL statements to be executed in same call to the server.

Important SQL Commands:

* SELECT – extracts data from a database
* UPDATE – updates data in a database
* DELETE – deletes data from a database
* INSERT INTO – inserts new data into a database
* CREATE DATABASE – creates new database
* ALTER DATABASE – modifies a database
* 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

## SQL SELECT Statement

* Used to select data from a database.

SELECT Syntax:

SELECT *column1, column2, …*

FROM *table\_name*;

Example: SELECT CustomerName, City FROM Customers;

If want to select all the fields available in the table:

SELECT \* FROM *table\_name*;

Example: SELECT \* FROM *Customers*;

## SQL SELECT DISTINCT Statement

* Used to return only distinct (different) values. A column often contains duplicate values, and sometimes you only want different values.

SELECT DISTINCT Syntax:

SELECT DISTINCT *column 1, column 2, …*

FROM *table\_name*;

SELECT example without DISTINCT: SELECT *Country* FROM *Customers*;

SELECT DISTINCT example: SELECT DISTINCT *Country* FROM *Customers*;

SELECT COUNT (DISTINCT *Country*) FROM *Customers*; (this statement lists the number of different customer countries)

SELECT COUNT (\*) AS DistinctCountries

FROM (SELECT DISTINCT Country FROM Customers); (workaround for MSAcess)

## SQL WHERE clause

* Is used to extract only those records that fulfil specified conditions
* Note: WHERE clause is not only used on SELECT statement but can be used on DELETE, UPDATE, etc.

WHERE Syntax:

SELECT *column 1, column 2, …*

FROM *table\_name*

WHERE *condition*;

Example: SELECT \* FROM *Customers*

WHERE *Country = ‘Mexico’*; (this statement selects all the customers from the Mexico)

SELECT COUNT (\*) FROM *Customers*

WHERE *Country = ‘Mexico’*; (this statement lists the number of customers from the Mexico)

SELECT COUNT (\*) AS *Mexico*

FROM (SELECT *Country* FROM *Customers* WHERE *Country = ‘Mexico’*;

Example: SELECT \* FROM *Customers*

WHERE *City = ‘London’*;

SELECT COUNT (\*) FROM *Customers*

WHERE *City = ‘London’*;

SELECT COUNT (\*) AS *CityLondon*

FROM (SELECT *City* FROM *Customers* WHERE *City = ‘London*;

### Text Fields VS Numeric Fields

* requires single quotes around text values (double quotes can also be used for most database systems) but not around numeric values.

Example – Text Fields: SELECT \* FROM *Customers*

WHERE *Country = ‘Mexico’*;

Example – Numeric Fields: SELECT \* FROM *Customers*

WHERE *CustomerID = 1*;

### Operators in the WHERE Clause

|  |  |
| --- | --- |
| **Operator** | **Description** |
| = | Equal |
| > | Greater than |
| < | Less than |
| >= | Greater than or equal |
| >= | Less than or equal |
| < > or != | Not equal |
| BETWEEN | Between a certain range |
| LIKE | Search for a pattern |
| IN | To specify multiple possible values for a column |

Remember: select is used to select data or extracts data from a database

select distinct is used to return only distinct values

where is used to extract those records that fulfil specified conditions

## SQL AND, OR and NOT Operators

* WHERE clause can be combined with AND, OR, and NOT Operators.
* The AND operator displays a record if all the conditions separated by AND are TRUE

AND Syntax:

SELECT *column1*, *column2*, *column3*, ...

FROM *table\_name*

WHERE *condition1* AND *condition2* AND *condition 3* …;

Example: SELECT *\** FROM *Customers*

WHERE *Country = ‘UK’* AND *City = ‘London’*;

SELECT *\** FROM *Customers*

WHERE *Country* IN(‘*UK’, ‘Spain’)* AND *City* IN (*‘London’, ‘Barcelona’, ‘Madrid’*);

* The OR operator displays a record if any of the conditions separated by OR are TRUE

OR Syntax:

SELECT *column1*, *column2*, *column3*, ...

FROM *table\_name*

WHERE *condition1* OR *condition2* OR *condition 3* …;

Example: SELECT *\** FROM *Customers*

WHERE *City = ‘Berlin’* OR ‘*London’*;

* The NOT operator displays a record if the conditions are NOT TRUE.

NOT Syntax:

SELECT *column1*, *column2*, *column3*, ...

FROM *table\_name*

WHERE NOT *condition*;

Example: SELECT \* FROM *customers*

WHERE NOT *Country = ‘Spain’*;

Example: SELECT \* FROM *customers*

WHERE NOT *Country* IN (*‘Spain’*, ‘Germany’)

Example: SELECT COUNT (\*) FROM *customers*

WHERE NOT *Country* IN (*‘Spain’*, ‘Germany’)

Example: SELECT COUNT (\*) AS *countriesotherthanUKandSpain*

FROM (SELECT *Country* FROM *Customers* WHERE NOT *Country* IN (*‘Spain’, ‘UK’*));

* Combining AND, OR and NOT:

Example: : SELECT \* FROM *customers*

WHERE *Country = ‘Spain’* AND (*City = ‘Barcelona’* OR *City = ‘Madrid’*);

Or

Example: SELECT \* FROM *customers*

WHERE *Country = ‘Spain’* AND *City* IN (*‘Spain’, Barcelona’);*

Example: SELECT COUNT (\*) AS *Spain*

FROM (SELECT (*Country, City)* FROM *customers* WHERE *Country = ‘Spain’* AND *City* IN (*‘Barcelona’, ‘Madrid’));*

Example: SELECT \* FROM *customers*

WHERE NOT *Country = ‘Spain’* AND NOT *Country = ‘Germany’;*

Or

Example: SELECT \* FROM *customers*

WHERE NOT *Country* IN *(‘Spain’, ‘Germany’);*

Example: SELECT COUNT (\*) AS *countriesotherthanGermanyandSpain*

FROM (SELECT *Country* FROM *Customers* WHERE NOT *Country* IN *(‘Spain’, ‘Germany’));*

## SQL ORDER BY Keyword

* Used to sort the result set in ascending – descending order.
* Default – ascending. If you want to change it to descending – DESC

ORDER BY Syntax:

SELECT *column1, column 2, …*

FROM *table\_name*

ORDER BY *column 1, column 2, … ASC|DESC;*

Example: SELECT \* FROM *customers,*

ORDER BY *Country;* (by default it’s been sorted out in ascending order)

Example: SELECT \* FROM *customers,*

ORDER BY *Country* DESC; (it’s been sorted out in descending order)

Example: SELECT \* FROM *customers,*

ORDER BY *Country, City;* (by defaults it been sorted out in ascending order)

Example: SELECT \* FROM *customers,*

ORDER BY *Country* ASC, *City* DESC; (the Country is sorted in ascending order and City in descending order)

## SQL INSERT INTO Statement:

* Used to insert new records in a table.

INSERT INTO Syntax:

INSERT INTO *table\_name (column1, column2, …)*

VALUES (*values1, values2, …);*

INSERT INTO *table\_name*

VALUES (*values1, values2, …);*

Example: INSERT INTO *Customers (CustomerName, ContactName, Address, City, PostalCode, Country)*

VALUES *(‘Cardinal’, ‘Tom B. Erichson’, ‘Skagen 21’, ‘Stavanger’, ‘4006’, ‘Norway’)*

Example: INSERT INTO *Customers (CustomerName, City, Country)*

VALUES *(‘Cardinal’, ‘Stavanger’, ‘Norway’)* (only insert data in specified columns)

## SQL NULL Value:

* Is a field with no value.
* Note: A NULL value is not a zero value or a field that contains space. A field with NULL value is the one that has been left during record

IS NULL Syntax

SELECT *column names*

FROM *table\_name*

WHERE *condition* IS NULL*;*

IS NO NULL Syntax:

SELECT *column names*

FROM *table\_name*

WHERE *condition* IS NOT NULL*;*

Example: SELECT *CustomerName, ContactName, Address*

FROM *Customers*

WHERE *Address* IS NULL*;*

Example: SELECT *CustomerName, ContactName, Address*

FROM *Customers*

WHERE *condition* IS NOT NULL*;*

## SQL UPDATE Statement

* Used to modify the existing records

UPDATE Syntax

UPDATE *table\_name*

SET *column1 = value1, column2 = value2, …*

WHERE *condition;*

Example: UPDATE *Products*

SET *SupplierID = 2, CategoryID = 3*

WHERE *ProductID = 1;*

Example: UPDATE *Customers*

SET *ContactName = ‘Juan’*

WHERE *Country = ‘UK’;* (this SQL statement will update the contact name Juan for all the records where the country is UK)

## SQL DELETE Statement:

* Is used to delete existing records in a table.

DELETE Syntax:

DELETE FROM *table\_name*

WHERE *condition;*

DELETE FROM *table\_name* ( this statement is used for deleting all the records in the table)

* Note: The WHERE Clause in the DELETE syntax. WHERE clause specifies which records should be deleted. If omitted, then all the records in the table will be deleted.

Example: DELETE FROM *Customers*

WHERE *CustomerName = ‘Around the Horn’;*

Example: DELETE FROM *Customers*

## SQL TOP, LIMIT or ROWNUM Clause

* Is used to specify the number of records to return.
* Note: Not all database support SELECT TOP clause. MYSQL supports LIMIT, whereas Oracle supports ROWNUM.

SELECT TOP Clause Syntax: MySQL Syntax

SELECT *column\_name(s)*

FROM *table\_name*

WHERE *condition*

LIMIT *number;*

SELECT TOP Clause Syntax: MS Access Syntax

SELECT TOP *number| percent columnname(s)*

FROM *table\_name*

WHERE *condition;*

SELECT TOP Clause Syntax: Oracle Syntax

SELECT *column\_name(s)*

FROM *table\_name*

WHERE *ROWNUM <= number;*

Example: SELECT *\** FROM *Products*

LIMIT 3*;*

Example: SELECT *\** FROM *Products*

WHERE *Price = 21;*

Example: SELECT *\** FROM *Products*

WHERE *Price = 21*

LIMIT *3;*

## SQL MIN () and MAX () Functions:

* The MIN () function returns the smallest value of the selected column.
* The MAX () function returns the largest value of the selected column.

MIN () Syntax:

SELECT MIN (*column\_name)*

FROM *table\_name*

WHERE *condition;*

Example: SELECT MIN *(Price) AS Smallest Price*

FROM *Products*

MAX () Syntax:

SELECT MAX (*column\_name)*

FROM *table\_name*

WHERE *condition;*

Example: SELECT MAX *(Price) AS LargestPrice*

FROM *Products*

## SQL COUNT (), AVG (), and SUM () Statement:

* COUNT () - returns the number of rows that matches the specified criteria.

COUNT () Syntax:

SELECT COUNT(*column\_name)*

FROM *table\_name*

WHERE *condition;*

Example: SELECT COUNT (\*)

FROM *Product*

WHERE *Quantity = 10;*

Example: SELECT COUNT (\*)

FROM *Product;*

* AVG () - returns the average value of a numeric column.

AVERAGE () Syntax:

SELECT AVG(*column\_name)*

FROM *table\_name*

WHERE *condition;*

Example: SELECT AVG (Price)

FROM *Product;*

* SUM () – returns the total sum of a numeric column.

SUM () Syntax:

SELECT SUM(*column\_name)*

FROM *table\_name*

WHERE *condition;*

Example: SELECT SUM (Price)

FROM *Product;*

## SQL LIKE Operator:

* Used in a WHERE clause to search for a specified pattern in a column.

% - the percent sign represents zero, one or multiple characters

\_ - the underscore sign represents a single character.

LIKE Operator Syntax:

SELECT *column1, column2, ...*

FROM *table\_name*

WHERE *columnN* LIKE *pattern;*

* Note: You can combine any number of conditions using AND or OR operators.

|  |  |
| --- | --- |
| **LIKE Operator** | **Description** |
| Where CustomerName LIKE ‘a%’ | Any values that starts with “a” |
| Where CustomerName LIKE ‘%a’ | Any values that ends with “a” |
| Where CustomerName LIKE ‘%or%’ | Any values that have “or” in any position |
| Where CustomerName LIKE ‘\_r%’ | Any values that have “r” in a second position |
| Where CustomerName LIKE ‘a\_%’ | Any values that start with “a” and are at least 3 characters in length |
| Where CustomerName LIKE ‘a%o’ | Any values that start with “a” and ends with “o” |

Example: SELECT *\** FROM *Customers*

WHERE *CustomerName* LIKE *‘a%’;* (gives values that starts with “a”)

Example: SELECT \*

FROM *Customers*

WHERE *CustomerName* NOT LIKE *‘a%’;* (gives values that does not start with “a”)

## SQL WILDCARDS characters:

* Is used to include one or more characters in a string.
* Wildcards characters are used with the LIKE Operators. The LIKE operator uses WHERE clause to search for a specified pattern in a column

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Description** | **Example** |
| % | Represents zero or more characters | bl% finds bl, black, blue and blob |
| \_ | Represents single character | h\_t finds hot, hat, and hit |
| [] | Represents any single character within the brackets | h[oa] finds hot, hat, but not hit |
| ^ | Represents any character not in the brackets | h[^oa] finds hit, but not hot and hat |
| - | Represents a range of characters | c[a-b]t finds cat, cbt |

## SQL IN Operator:

* Allows you to specify multiple values in a WHERE clause.
* Is a shorthand version of multiple OR conditions.

IN Syntax:

SELECT *column names(s)*

FROM *table\_name*

WHERE *column name* IN *(value1, value2, …);*

Or

SELECT *column names(s)*

FROM *table\_name*

WHERE *column name* IN *(*SELECT *STATEMENT);*

Example: SELECT \* FROM *customers*

WHERE *Country* IN *(‘Germany’, ‘France’, ‘UK’)*

Example: SELECT \* FROM *customers*

WHERE *Country* NOTIN *(‘Germany’, ‘France’, ‘UK’)*

Example: SELECT \* FROM *customers*

WHEOT IRE *Country* IN *(*SELECT *Country* FROM *Suppliers)*

## SQL BETWEEN Operator:

* Selects values within the given range. The values can be text, numbers or dates.
* BETWEEN operator is inclusive – begin and end values are included.

BETWEEN Operator Syntax:

SELECT *column names(s)*

FROM *table\_name*

WHERE *column name* BETWEEN *value1* AND *value2;*

Example: SELECT *\**FROM *Products*

WHERE *Price* BETWEEN *10* AND *20;*

Example: SELECT *\**FROM *Products*

WHERE *Price NOT* BETWEEN *10* AND *20;*

Example: SELECT *\**FROM *Products*

WHERE *Price* BETWEEN *10* AND *20*

AND *Category ID* NOT IN *(1,2,3);* (This example -> BETWEEN with IN Example)

Example: SELECT *\**FROM *Products*

WHERE *ProductName* BETWEEN *‘Carnarvon Tigers’* AND *‘Mozzarella di Giovanni’*

ORDER BY *ProductName* (This example -> BETWEEN with text values example)

Example: SELECT *\**FROM *Products*

WHERE *ProductName* NOT BETWEEN *‘Carnarvon Tigers’* AND *‘Mozzarella di Giovanni’*

ORDER BY *ProductName*

Example: SELECT *\**FROM *Orders*

WHERE *OrderDate NOT* BETWEEN *#01/07/1996#* AND *#31/07/1996#;* (This example -> BETWEEN Dates example)

OR

Example: SELECT *\**FROM *Orders*

WHERE *OrderDate NOT* BETWEEN *’01-07-1996’* AND *‘31-07-1996’;*

## SQL ALIASES:

* Is used to give a table, or a column in a table a temporary name.
* Only exists for a duration of the query.

ALIAS COLUMN Syntax:

SELECT *column name* AS *alias\_name*

FROM *table\_name;*

Example: SELECT *CustomerID* AS *ID,* SELECT *Customer Name* AS *Customers*

FROM *Customers;*

ALIAS TABLE Syntax:

SELECT *column name(s)*

FROM *table\_name* AS *alias\_name;*

Example: SELECT *o.OrderID, o.OrderDate, c.CustomerName*

FROM *Customers* AS *c, Orders* AS *o*

WHERE *c.CustomersName = “Around the Horn”* AND *c.CustomerID = o.CustomerID;*

* Note: Always use quotation marks or squared brackets if alias name contains space.

Example: SELECT *SupplierID* AS *ID, SupplierName* AS *[Supplier Name], ContactName* AS *[Contact Name]*

FROM *Suppliers;*

STILL NEED TO DO SQL JOINS



## SQL UNION Operator:

* Is used to combine result-set of two or more SELECT statements.
* Each SELECT statement within UNION must have the same number of columns.
* The columns must have similar data types
* The columns in each SELECT statement must also be in the same order.

UNION Syntax:

SELECT *column name (s)* FROM *table\_name1*

UNION

SELECT *column name (s)* FROM *table\_name2;*

Example: SELECT *City, Country* FROM *Customers*

FROM *Country = ‘Germany’*

UNION

SELECT *City, Country* FROM *Suppliers*

WHERE *Country = ‘Germany’*

ORDER BY *City;* (the result returns the German cities (only distinct (different) values from both the Customer and the Suppliers table)

UNION ALL Syntax:

SELECT *column name (s)* FROM *table\_name1*

UNION ALL

SELECT *column name (s)* FROM *table\_name2;*

Example: SELECT *City, Country* FROM *Customers*

FROM *Country = ‘Germany’*

UNION ALL

SELECT *City, Country* FROM *Suppliers*

WHERE *Country = ‘Germany’*

ORDER BY *City;* (the result returns the German cities (duplicate values also) from both the Customer and the Suppliers table)

Example: SELECT *‘Customer’* AS *Type, ContactName, City, Country*

FROM *Customers*

UNION

SELECT *‘Suppliers’, ContactName, City, Country’*

FROM *Suppliers;* (the result returns the German cities (only distinct (different) values from both the Customer and the Suppliers table)

## SQL GROUP BY Statement:

* The GROUP BY statement group rows that have the same values into similar rows.
* Is often used to aggregate with SUM, AVG, COUNT, MAX, MIN functions to group the result – set by one or more columns.

GROUP BY Syntax:

SELECT *column name (s)*

FROM *table\_name*

GROUP BY *column name (s)*

ORDER BY *column name (s);*

## SQL HAVING Clause:

* It was added because the WHERE keyword could not be used with the other aggregated functions.

HAVING Clause Syntax:

SELECT *column name (s)*

FROM *table\_name*

WHERE *condition*

GROUP BY *column name (s)*

HAVING *condition*

ORDER BY *column name (s);*

## SQL EXISTS Operator:

* Is used to test for the existence of any record in a subquery.

EXISTS Syntax:

SELECT *column\_name(s)*

FROM *table\_name*

WHERE EXISTS

(SELECT *column\_name* FROM *table\_name* WHERE *condition*);

## SQL ANY, ALL Operators:

* ANY and ALL operators are used with WHERE and HAVING clause

ANY Syntax:

SELECT *column\_name(s)*

FROM *table\_name*

WHERE *column\_name operator* ANY

(SELECT *column\_name* FROM *table\_name* WHERE *condition);*

ALL Syntax:

SELECT *column\_name(s)*

FROM *table\_name*

WHERE *column\_name operator* ANY

(SELECT *column\_name* FROM *table\_name* WHERE *condition);*

## SELECT INTO Statement:

* Copies one data from one table into a new table

SELECT INTO Syntax:

SELECT *\**

INTO new\_*table [*IN *externaldb]*

FROM *old\_table*

WHERE *condition;* (copies all columns into a new table)

SELECT *column1, column2, column3, …*

INTO new\_*table [*IN *externaldb]*

FROM *old\_table*

WHERE *condition;* (copies selected columns in a new table)

## SQL INSERT INTO SELECT Statement:

* Copiers data from one table and inserts it into another table.

INSERT INTO Syntax:

INSERT INTO *table2*

SELECT \* FROM *table 1*

WHERE *condition;* (copies all columns from one table to another)

INSERT INTO *table 2(column1, column2, …)*

SELECT *column1, column2, …*

FROM *table1*

WHERE *condition;* (copies all columns from one table to another)

## SQL Case Statement:

* The CASE statement goes through conditions and returns a value when the first condition is met (Its like an IF – THEN ELSE statement)

CASE Syntax:

CASE

WHEN *condition1* THEN *result1*

WHEN *condition2* THEN *result2*

WHEN *conditionN* THEN *result*

ELSE *results*

END;

* If there is no ELSE statement, it returns as NULL.

## SQL COALESCE () FUNCTION:

* Returns the first Non – NULL expression among its arguments. If all the expressions evaluate to null, then COALESCE function will return null.

COALESCE Syntax:

SELECT *column name (s),* COALESCE(*expression1, expressions2, … expressionsN)*

FROM *table\_name;*

Example: SELECT *ProductName, UnitPrice \* (UnitInStock +* COALESCE(*UnitOnStock, 0)*

DATABASES

## SQL CREATE DATABASE Statement:

* Create a new SQL database

CREATE Database Syntax:

CREATE DATABASE *databasename;*

Example: CREATE DATABASE *testDB;*

## SQL DROP DATABASE Statement:

* Is used to drop (delete) an existing database.

DROP database Syntax:

DROP DATABASE [IF EXISTS]

databasename (this syntax is used to drop (delete) a database)

DROP DATABASE [IF EXISTS]

databasename1, databasename2, … (this syntax is used to drop (delete) multiple databases)

The IF EXISTS allows you to conditionally delete a database only if the database already exists.

## SQL BACKUP DATABASE Statement:

* Is used to create a full backup of an existing database.

BACKUP DATABASE Syntax:

BACKUP DATABASE *databasename*

TO DISK = *‘filepath’;*

BACKUP DIFFERENTIAL DATABASE Syntax:

BACKUP DATABASE *databasename*

TO DISK = *‘filepath’*

WITH *DIFFERENTIAL;* (it only backs up the part of the database that has changed since the full database backup)

## SQL CREATE TABLE Statement:

* Is used to create a new table in a database.

CREATE TABLE Syntax:

CREATE TABLE *table\_name (*

*Column1 datatype,*

*Column2 datatype,*

*Column3 datatype,*

*…*

*);*

*Columns* – specifies the name of the column of the table

*Datatype* – specifies the types of data the column can hold (varcher, integer, etc…)

Example: CREATE TABLE *Persons (*

*PersonsID INT,*

*LastName VARCHAR (255),*

*FirstName VARCHAR (255),*

*Address VARCHAR (255),*

*City VARCHAR (255));*

## SQL DROP TABLE Statement:

* The DROP table is used to drop (delete) the existing table in a database.

DROP TABLE Syntax:

DROP TABLE *table\_name;*

## SQL ALTER TABLE Statement:

* ALTER table is used to add, delete or modify columns on an existing table.
* Also used to add and drop various constraints on an existing table.

ALTER TABLE Syntax – ADD Column:

ALTER TABLE *table\_name*

ADD COLUMN *column\_name datatype*;

ALTER TABLE Syntax – DROP Column:

ALTER TABLE *table\_name*

DROP COLUMN *column\_name datatype;*

ALTER TABLE Syntax - ALTER/MODIFY (Change) Column:

ALTER TABLE *table\_name*

ALTER COLUMN *column\_name;*

## SQL CREATE CONSTRAINTS:

* 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.

Syntax:

CREATE TABLE *table\_name (*

*Column1 constraint,*

*Column2 constraint,*

*Column3 constraint,*

*…*

*);*

* Are used to specify rules for the data in a table
* Are used to limit the data that can go into the table
* Constraints can be column level or table level.

Column level applies constraints to the columns

Table level applies constraints to the whole table.

* Constraints that are commonly used in SQL:
* NOT NULL: Ensures that a column cannot have a NULL value.
* UNIQUE: Ensures that all values in a column are different.
* PRIMARY KEY: it is a combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table.
* FOREIGN KEY: Uniquely identifies a row/record in another table.
* CHECK: ensures that all values in a column uniquely satisfies a specific condition.
* DEFAULT: sets a default value for a column when no values is specified.
* INDEX: Used to create and retrieve data from the database very quickly.

### NOT NULL CONSTRAINT:

* Enforces a column not to accept null values.

Example (SQL NOT NULL on CREATE table): CREATE TABLE *Passengers (*

*ID int NOT NULL,*

*LastName varchar (255) NOT NULL,*

*FirstName varchar (255) NOT NULL,*

*Age int NOT NULL*

*);*

Example (SQL NOT NULL on ALTER table): ALTER TABLE *Passengers*

MODIFY *Age int* NOT NULL;

### UNIQUE CONSTRAINT:

* Ensures all values in a column are unique (different).
* Both UNIQUE and PRIMARY KEY constraints provide a guarantee of uniqueness for a column or sets of columns
* A PRIMARY KEY Constraint automatically has a UNIQUE constraint.
* You can have many UNIQUE constraints per table, but one PRIMARY KEY constraint per table.

Example: CREATE TABLE *Persons (*

*ID* int NOT NULL,

*LastName* varchar (255) NOT NULL,

*FirstName* varchar (255) NOT NULL,

*Age* int,

UNIQUE (*ID*)

);

Example: CREATE TABLE *Persons (*

*ID* int NOT NULL,

*LastName* varchar (255) NOT NULL,

*FirstName* varchar (255) NOT NULL,

*Age* int,

CONSTRAINT UC\_Persons UNIQUE (ID, LastName)

);

### SQL PRIMARY KEY Constraint:

* Uniquely identifies each record in a table.
* Must contain UNIQUE values and cannot contain NULL Values.
* A table can only have ONE primary key and in the table the primary key can consist of single or multiple columns.

Example: CREATE TABLE *Persons (*

*ID* int NOT NULL,

*LastName* varchar (255) NOT NULL,

*FirstName* varchar (255) NOT NULL,

*Age* int,

PRIMARY KEY (*ID*)

);

Example: CREATE TABLE *Persons (*

*ID* int NOT NULL,

*LastName* varchar (255) NOT NULL,

*FirstName* varchar (255) NOT NULL,

*Age* int,

CONSTRAINT UC\_Persons PRIMARY KEY (ID, LastName)

);

Example: ALTER TABLE *Persons*

ADD PRIMARY KEY *(ID);* ( can be used to create PRIMARY KEY constraint on the “ID” column when the table is already created)

### SQL FOREIGN KEY Constraint:

* Is a key used to link two tables together.
* Is a field in one table that refers to the PRIMARY KEY in another table.
* Is used to prevent actions that would destroy links between tables.
* Also prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the table it points to.

Example: CREATE TABLE *Orders (*

*OrderID* int NOT NULL,

*OrderNumber* int NOT NULL,

*PersonID* int NOT NULL,

PRIMARY KEY *(OrderID),*

FOREIGN KEY *(PersonID)* REFERENCES *Persons (PersonID)*

);

### SQL CHECK Constraint:

* is used to limit the value range that can be placed in a column.
* If you define a CHECK constraint on a single column it allows only certain values for this column.
* If you define a CHECK constraint on a table, it can limit the values in a certain column based on values in other columns in the row.

Example: CREATE TABLE *Persons (*

*PersonID* int NOT NULL,

*LastName* varchar (255) NOT NULL,

*FirstName* varchar (255),

*Age* int,

CHECK *(Age >= 18)*

);

Example: CREATE TABLE *Persons (*

*PersonID* int NOT NULL,

*LastName* varchar (255) NOT NULL,

*FirstName* varchar (255),

*Age* int,

*City* varchar (255),

CONSTRAINT *CHK\_Persons* CHECK *(Age >= 18, City = ‘Sandnes’)*

);

Example: ALTER TABLE *Persons*

ADD CHECK *(Age >= 18);*

### SQL DEFAULT Constraint:

* Is used to provide a default value for a column.
* The DEFAULT value will be added to all new records IF no other value is specified.

Example: CREATE TABLE *Persons (*

*PersonID* int NOT NULL,

*LastName* varchar (255) NOT NULL,

*FirstName* varchar (255),

*Age* int,

*City* varchar (255) DEFAULT *‘Sandnes’*

*);*

Example: ALTER TABLE *Persons*

ALTER COLUMN *City* DROP DEFAULT;

### SQL CREATE INDEX Statement:

* Is used to create indexes in tables.
* Are used to retrieve data from the databases more quickly.
* Note: Updating a table with indexes take more time than a table without because the indexes also need an update.

CREATE INDEX Syntax:

CREATE INDEX *index\_name*

ON *table\_name (column1, column2, …);*

CREATE UNIQUE INDEX Syntax:

CREATE UNIQUE INDEX *index\_name*

ON *table\_name (column1, column2, …);*

## AUTO INCREMENT Field:

* Allows a unique number to be generated automatically when a new record is inserted into a table.

Example: CREATE TABLE *Persons (*

*PersonID* int NOT NULL AUTO INCREMENT,

*LastName* varchar (255) NOT NULL,

*FirstName* varchar (255)

*Age* int,

PRIMARY *KEY (PersonID)*

*);*

Example: ALTER TABLE *Persons* AUTO INCREMENT = 100; (to let the AUTO – INCREMENT sequence to start with another value, we can use this above statement)