**SQL IFNULL(),IS NULL COALESCE(), and NVL() Functions, Data tpes, SQL Constraints, Primary Key, Foreign key, Check, Default**

---------------------------------------------------------------------------------------------

-- Suppose that the "UnitsOnOrder" column is optional, and may contain NULL values.

-- Look at the following SELECT statement:

SELECT ProductName, price

FROM Products where ProductName is not null;

select \* from products;

-- In the example above, if any of the "UnitsOnOrder" values are NULL, the result will be NULL.

-- The MySQL IFNULL() function lets you return an alternative value if an expression is NULL:

SELECT IFNULL(ProductName,'NA'), SUM(IFNULL(price,0))

FROM Products group by ProductName;

-- CCOALESCE

select \* from customers

where customerid is null;

select coalesce(customerid::varchar,contactname,country,address,postalcode::varchar,customername) as first\_not\_null\_value from customers

where customerid is null;

-- The MS Access IsNull() function returns TRUE (-1) if the expression is a null value, otherwise FALSE (0):

SELECT ProductName, UnitPrice \* (UnitsInStock + IIF(IsNull(UnitsOnOrder), 0, UnitsOnOrder))

FROM Products;

-- NVL

select NVL('A',null);

select NVL(null,'A');

select NVL('B','A');

-- CCOALESCE

select COALESCE(null,'A',null,null,'D','B','C',null);

-- NVL2

select NVL2('A','B','C');

select NVL2(null,'B','C');

select NVL2(null,null,'C');

select NVL2(null,null,null);

select NVL2('A',null,null);

---------------------------------------------------------------------------------------------

-- SQL Data Types

---------------------------------------------------------------------------------------------

/\*

The data type of a column defines what value the column can hold: integer, character, money, date and time, binary, and so on.

SQL Data Types

Each column in a database table is required to have a name and a data type.

An SQL developer must decide what type of data that will be stored inside each column when creating a table. The data type is a guideline for SQL to understand what type of data is expected inside of each column, and it also identifies how SQL will interact with the stored data.

Note: Data types might have different names in different database. And even if the name is the same, the size and other details may be different! Always check the documentation!

VARCHAR(TEXT/NUMBERS and NUMBERS+TEXT)-16mb(10000)

\*/

-- String Data Types:-

---------------------------------------------------------------------------------------------

Data type Description

CHAR(size) A FIXED length string (can contain letters, numbers, and special characters). The size parameter specifies the column length in characters - can be from 0 to 255. Default is 1

VARCHAR(size) A VARIABLE length string (can contain letters, numbers, and special characters). The size parameter specifies the maximum string length in characters - can be from 0 to 65535

BINARY(size) Equal to CHAR(), but stores binary byte strings. The size parameter specifies the column length in bytes. Default is 1

TINYBLOB For BLOBs (Binary Large Objects). Max length: 255 bytes

TINYTEXT Holds a string with a maximum length of 255 characters

TEXT(size) Holds a string with a maximum length of 65,535 bytes

BLOB(size) For BLOBs (Binary Large Objects). Holds up to 65,535 bytes of data

STRING/TEXT/VARCHAR

-- Numeric Data Types:-

---------------------------------------------------------------------------------------------

Data type Description

BOOLEAN Equal to BOOL

INT(size) A medium integer. Signed range is from -2147483648 to 2147483647. Unsigned range is from 0 to 4294967295. The size parameter specifies the maximum display width (which is 255)

INTEGER(size) Equal to INT(size)

BIGINT(size) A large integer. Signed range is from -9223372036854775808 to 9223372036854775807. Unsigned range is from 0 to 18446744073709551615. The size parameter specifies the maximum display width (which is 255)

FLOAT(size, d) A floating point number. The total number of digits is specified in size. The number of digits after the decimal point is specified in the d parameter. This syntax is deprecated in MySQL 8.0.17, and it will be removed in future MySQL versions

FLOAT(p) A floating point number. MySQL uses the p value to determine whether to use FLOAT or DOUBLE for the resulting data type. If p is from 0 to 24, the data type becomes FLOAT(). If p is from 25 to 53, the data type becomes DOUBLE()

DOUBLE(size, d) A normal-size floating point number. The total number of digits is specified in size. The number of digits after the decimal point is specified in the d parameter

DOUBLE PRECISION(size, d)

DECIMAL(size, d) An exact fixed-point number. The total number of digits is specified in size. The number of digits after the decimal point is specified in the d parameter. The maximum number for size is 65. The maximum number for d is 30. The default value for size is 10. The default value for d is 0.

-- Note: All the numeric data types may have an extra option: UNSIGNED or ZEROFILL. If you add the UNSIGNED option, MySQL disallows negative values for the column. If you add the ZEROFILL option, MySQL automatically also adds the UNSIGNED attribute to the column.

DECIMAL/FLOAT/NUMBER(P,S)

-- Date and Time Data Types:-

---------------------------------------------------------------------------------------------

Data type Description

DATE A date. Format: YYYY-MM-DD. The supported range is from '1000-01-01' to '9999-12-31'

DATETIME(fsp) A date and time combination. Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'. Adding DEFAULT and ON UPDATE in the column definition to get automatic initialization and updating to the current date and time

TIMESTAMP(fsp) A timestamp. TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD hh:mm:ss. The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC. Automatic initialization and updating to the current date and time can be specified using DEFAULT CURRENT\_TIMESTAMP and ON UPDATE CURRENT\_TIMESTAMP in the column definition

TIME(fsp) A time. Format: hh:mm:ss. The supported range is from '-838:59:59' to '838:59:59'

YEAR A year in four-digit format. Values allowed in four-digit format: 1901 to 2155, and 0000.

---------------------------------------------------------------------------------------------

-- SQL Constraints

---------------------------------------------------------------------------------------------

/\*

SQL constraints are used to specify rules for data in a table.

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 datatype constraint,

column2 datatype constraint,

column3 datatype constraint,

....

);

SQL Constraints

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.

The following constraints 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 - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table

FOREIGN KEY - Prevents actions that would destroy links between tables

CHECK - Ensures that the values in a column satisfies a specific condition

DEFAULT - Sets a default value for a column if no value is specified

\*/

COMMENT TEXT DEFAULT 'No comments'

AGE INT CHECK (AGE>=18)

-- Not null:-

---------------------------------------------------------------------------------------------

-- SQL NOT NULL on CREATE TABLE

-- The following SQL ensures that the "ID", "LastName", and "FirstName" columns will NOT accept NULL values when the "Persons" table is created:

CREATE TABLE Persons (

ID int NOT NULL,

LastName varchar(255) NOT NULL,

FirstName varchar(255) NOT NULL,

Age int

);

insert into persons values(null,null,null,22);

-- SQL 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:

ALTER TABLE Persons

MODIFY Age int NOT NULL;

-- UNIQUE :-

---------------------------------------------------------------------------------------------

/\*

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.

SQL UNIQUE Constraint on CREATE TABLE

The following SQL creates a UNIQUE constraint on the "ID" column when the "Persons" table is created:

\*/

CREATE TABLE Persons (

ID int NOT NULL UNIQUE,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Age int

);

-- To name a UNIQUE constraint, and to define a UNIQUE constraint on multiple columns, use the following SQL syntax:

--MySQL / SQL Server / Oracle / MS Access:

CREATE TABLE Persons (

ID int NOT NULL,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Age int,

CONSTRAINT UC\_Person UNIQUE (ID,LastName)

);

--SQL UNIQUE Constraint on ALTER TABLE

--To create a UNIQUE constraint on the "ID" column when the table is already created, use the following SQL:

MySQL / SQL Server / Oracle / MS Access:

ALTER TABLE Persons

ADD UNIQUE (ID);

To name a UNIQUE constraint, and to define a UNIQUE constraint on multiple columns, use the following SQL syntax:

MySQL / SQL Server / Oracle / MS Access:

ALTER TABLE Persons

ADD CONSTRAINT UC\_Person UNIQUE (ID,LastName);

DROP a UNIQUE Constraint

To drop a UNIQUE constraint, use the following SQL:

MySQL:

ALTER TABLE Persons

DROP INDEX UC\_Person;

SQL Server / Oracle / MS Access:

ALTER TABLE Persons

DROP CONSTRAINT UC\_Person;

-- PRIMARY KEY :-

---------------------------------------------------------------------------------------------

/\*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).

SQL PRIMARY KEY on CREATE TABLE

The following SQL creates a PRIMARY KEY on the "ID" column when the "Persons" table is created:

\*/

MySQL:

CREATE TABLE Persons (

ID int NOT NULL,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Age int,

PRIMARY KEY (ID)

);

SQL Server / Oracle / MS Access:

CREATE TABLE Persons (

ID int NOT NULL PRIMARY KEY,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Age int

);

--To allow naming of a PRIMARY KEY constraint, and for defining a PRIMARY KEY constraint on multiple columns, use the following SQL syntax:

--MySQL / SQL Server / Oracle / MS Access:

CREATE TABLE Persons (

ID int NOT NULL,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Age int,

CONSTRAINT PK\_Person PRIMARY KEY (ID,LastName)

);

--Note: In the example above there is only ONE PRIMARY KEY (PK\_Person). However, the VALUE of the primary key is made up of TWO COLUMNS (ID + LastName).

SQL 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:

--MySQL / SQL Server / Oracle / MS Access:

ALTER TABLE Persons

ADD PRIMARY KEY (ID);

--To allow naming of a PRIMARY KEY constraint, and for defining a PRIMARY KEY constraint on multiple columns, use the following SQL syntax:

--MySQL / SQL Server / Oracle / MS Access:

ALTER TABLE Persons

ADD CONSTRAINT PK\_Person PRIMARY KEY (ID,LastName);

--Note: If you use ALTER TABLE to add a primary key, the primary key column(s) must have been declared to not contain NULL values (when the table was first created).

DROP a PRIMARY KEY Constraint

To drop a PRIMARY KEY constraint, use the following SQL:

--MySQL:

ALTER TABLE Persons

DROP PRIMARY KEY;

SQL Server / Oracle / MS Access:

ALTER TABLE Persons

DROP CONSTRAINT PK\_Person;

-- FOREIGN KEY :-

---------------------------------------------------------------------------------------------

/\*

SQL 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.

Look at the following two tables:

Persons Table

PersonID LastName FirstName Age

1 Hansen Ola 30

2 Svendson Tove 23

3 Pettersen Kari 20

Orders Table

OrderID OrderNumber PersonID

1 77895 3

2 44678 3

3 22456 2

4 24562 1

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.

SQL FOREIGN KEY on CREATE TABLE

The following SQL creates a FOREIGN KEY on the "PersonID" column when the "Orders" table is created:

\*/

-- MySQL:

CREATE TABLE Orders (

OrderID int NOT NULL,

OrderNumber int NOT NULL,

PersonID int,

PRIMARY KEY (OrderID),

FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)

);

--SQL Server / Oracle / MS Access:

CREATE TABLE Orders (

OrderID int NOT NULL PRIMARY KEY,

OrderNumber int NOT NULL,

PersonID int FOREIGN KEY REFERENCES Persons(PersonID)

);

--To allow naming of a FOREIGN KEY constraint, and for defining a FOREIGN KEY constraint on multiple columns, use the following SQL syntax:

--MySQL / SQL Server / Oracle / MS Access:

CREATE TABLE Orders (

OrderID int NOT NULL,

OrderNumber int NOT NULL,

PersonID int,

PRIMARY KEY (OrderID),

CONSTRAINT FK\_PersonOrder FOREIGN KEY (PersonID)

REFERENCES Persons(PersonID)

);

--SQL FOREIGN KEY on ALTER TABLE

--To create a FOREIGN KEY constraint on the "PersonID" column when the "Orders" table is already created, use the following SQL:

--MySQL / SQL Server / Oracle / MS Access:

ALTER TABLE Orders

ADD FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);

--To allow naming of a FOREIGN KEY constraint, and for defining a FOREIGN KEY constraint on multiple columns, use the following SQL syntax:

--MySQL / SQL Server / Oracle / MS Access:

ALTER TABLE Orders

ADD CONSTRAINT FK\_PersonOrder

FOREIGN KEY (PersonID) REFERENCES Persons(PersonID);

DROP a FOREIGN KEY Constraint

--To drop a FOREIGN KEY constraint, use the following SQL:

--MySQL:

ALTER TABLE Orders

DROP FOREIGN KEY FK\_PersonOrder;

SQL Server / Oracle / MS Access:

ALTER TABLE Orders

DROP CONSTRAINT FK\_PersonOrder;

-- CHECK :-

---------------------------------------------------------------------------------------------

/\*

SQL 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.

SQL 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:

\*/

--MySQL:

CREATE TABLE Persons (

ID int NOT NULL,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Age int,

CHECK (Age>=18)

);

--SQL Server / Oracle / MS Access:

CREATE TABLE Persons (

ID int NOT NULL,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Age int CHECK (Age>=18)

);

--To allow naming of a CHECK constraint, and for defining a CHECK constraint on multiple columns, use the following SQL syntax:

--MySQL / SQL Server / Oracle / MS Access:

CREATE TABLE Persons (

ID int NOT NULL,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Age int,

City varchar(255),

CONSTRAINT CHK\_Person CHECK (Age>=18 AND City='Sandnes')

);

--SQL CHECK on ALTER TABLE

--To create a CHECK constraint on the "Age" column when the table is already created, use the following SQL:

--MySQL / SQL Server / Oracle / MS Access:

ALTER TABLE Persons

ADD CHECK (Age>=18);

--To allow naming of a CHECK constraint, and for defining a CHECK constraint on multiple columns, use the following SQL syntax:

--MySQL / SQL Server / Oracle / MS Access:

ALTER TABLE Persons

ADD CONSTRAINT CHK\_PersonAge CHECK (Age>=18 AND City='Sandnes');

DROP a CHECK Constraint

To drop a CHECK constraint, use the following SQL:

--SQL Server / Oracle / MS Access:

ALTER TABLE Persons

DROP CONSTRAINT CHK\_PersonAge;

-- MySQL:

ALTER TABLE Persons

DROP CHECK CHK\_PersonAge;

-- DEFAULT :-

---------------------------------------------------------------------------------------------

/\*SQL 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 value is specified.

SQL DEFAULT on CREATE TABLE

The following SQL sets a DEFAULT value for the "City" column when the "Persons" table is created:

\*/

CREATE TABLE Persons (

ID int NOT NULL,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Age int,

City varchar(255) DEFAULT 'Sandnes'

);

--The DEFAULT constraint can also be used to insert system values, by using functions like GETDATE():

CREATE TABLE Orders (

ID int NOT NULL,

OrderNumber int NOT NULL,

OrderDate date DEFAULT GETDATE()

);

--SQL DEFAULT on ALTER TABLE

--To create a DEFAULT constraint on the "City" column when the table is already created, use the following SQL:

ALTER TABLE Persons

ALTER City SET DEFAULT 'Sandnes';

SQL Server:

ALTER TABLE Persons

ADD CONSTRAINT df\_City

DEFAULT 'Sandnes' FOR City;

ALTER TABLE Persons

ALTER COLUMN City SET DEFAULT 'Sandnes';

ALTER TABLE Persons

MODIFY City DEFAULT 'Sandnes';

--DROP a DEFAULT Constraint

--To drop a DEFAULT constraint, use the following SQL:

ALTER TABLE Persons

ALTER City DROP DEFAULT;

ALTER TABLE Persons

ALTER COLUMN City DROP DEFAULT;