

SQL 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
- CREATE INDEX - Used to create and retrieve data from the database very quickly

SQL 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.

```
CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255) NOT NULL,  
    Age int
```

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.

```
CREATE TABLE Persons (  
    ID int NOT NULL UNIQUE,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255) NOT NULL,  
    Age int
```

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:

```
SQLQuery1.sql - lo...HWAK\viswak (54))*  X
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int CHECK (Age>=18)
);
```

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

```
SQLQuery1.sql - lo...HWAK\viswak (54))*  X
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255) DEFAULT 'Sandnes'
);
```

SQL 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.

CREATE INDEX Syntax

Creates an index on a table. Duplicate values are allowed:

```
CREATE INDEX index_name  
ON table_name (column1, column2, ...);
```

CREATE UNIQUE INDEX Syntax

Creates a unique index on a table. Duplicate values are not allowed:

```
CREATE UNIQUE INDEX index_name  
ON table_name (column1, column2, ...);
```

Data Manipulation Language

- DML commands are used to modify the database. It is responsible for all form of changes in the database.
- The command of DML is not auto-committed that means it can't permanently save all the changes in the database. They can be rollback. ◦
INSERT ◦ UPDATE ◦ DELETE

a. INSERT: The INSERT statement is a SQL query. It is used to insert data into the row of a table.

	name	email	DOB	address
1	vishwak	vsenan827@gmail.com	2001-07-17	london

b. UPDATE: This command is used to update or modify the value of a column in the table

	name	email
.1	senan	senan@gmail.com

c. DELETE: It is used to remove one or more row from a table.

Syntax:

DELETE FROM table_name [WHERE condition];

Data Control Language

- DCL stands for **Data Control Language**.
- DCL is used to control user access in a database.
- This command is related to the security issues.
- Using DCL command, it allows or restricts the user from accessing data in database schema.
- **DCL commands are as follows,**
 - 1. GRANT**
 - 2. REVOKE**
- It is used to grant or revoke access permissions from any database user.

1. GRANT COMMAND

- **GRANT command** gives user's access privileges to the database.
- This command allows specified users to perform specific tasks.

Syntax:

GRANT <privilege list>
ON <relation name or view name> TO
<user/role list>;

2. REVOKE COMMAND

- **REVOKE command** is used to cancel previously granted or denied permissions.
- This command withdraw access privileges given with the GRANT command.
- It takes back permissions from user.

Syntax:

REVOKE <privilege list>
ON <relation name or view name>
FROM <user name>;

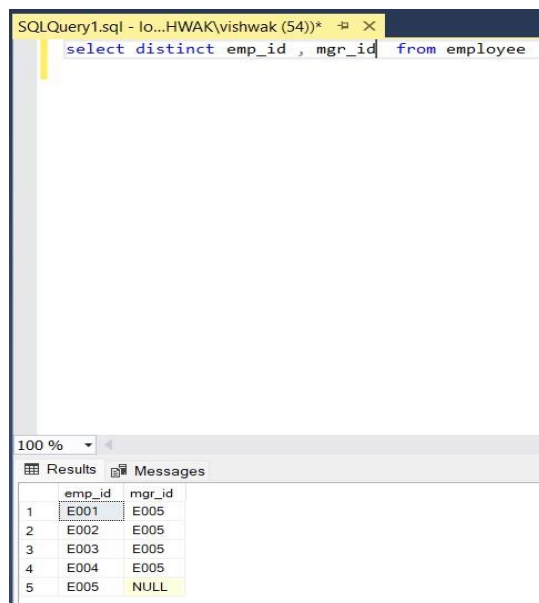
ORDER BY

- 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.

ID	EMPLOYEE	MANAGER	DATE OF BIRTH	DATE OF JOINING	SALARY	DEPARTMENT	ADDRESS	PHONE	ZIP	SEX
4	E002	Raja	2016-07-03	20500.00	E005	D001	7745781446	10	10	5
5	E001	Gopi	2016-03-03	30000.00	E005	D001	7845781446	10	10	5

SQL SELECT DISTINCT

- The SELECT DISTINCT statement is used to return only distinct (different) values.
- Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values



The screenshot shows a SQL query editor window titled "SQLQuery1.sql - Io...HWAK\viswak (54)". The query entered is "select distinct emp_id , mgr_id from employee". Below the query, the "Results" tab is active, displaying the following data:

	emp_id	mgr_id
1	E001	E005
2	E002	E005
3	E003	E005
4	E004	E005
5	E005	NULL

SQL ROW_NUMBER() Function

- The ROW_NUMBER() is a window function that assigns a sequential integer number to each row in the query's result set.

The following illustrates the syntax of the ROW_NUMBER() function:

- ROW_NUMBER() OVER ([PARTITION BY expr1, expr2,...] ORDER BY expr1 [ASC | DESC], expr2,...)

In this syntax,

- First, the PARTITION BY clause divides the result set returned from the FROM clause into partitions. The PARTITION BY clause is optional. If you omit it, the whole result set is treated as a single partition.
- Then, the ORDER BY clause sorts the rows in each partition. Because the ROW_NUMBER() is an order sensitive function, the ORDER BY clause is required.
- Finally, each row in each partition is assigned a sequential integer number called a row number. The row number is reset whenever the partition boundary is crossed.

SQL ROW_NUMBER() examples

We will use the employees and departments tables from the sample database for the demonstration:

