SQL Programming Lecture 3

Department of Information Technology
Rajiv Gandhi College of Engineering and Research,
Nagpur.

SQL | Constraints

- Constraints are the rules that we can apply on the type of data in a table.
- Can specify the limit on the type of data that can be stored in a particular column in a table
- constraints in SQL are:
- **NOT NULL**
- **UNIQUE**
- **PRIMARY KEY**
- **FOREIGN KEY**
- **CHECK**
- **DEFAULT**

How to specify constraints?

- At the time of creating the table using CREATE TABLE statement.
- Also specify the constraints after creating a table using ALTER TABLE statement.
- **Syntax**:

```
CREATE TABLE sample_table(
 column1 data_type(size) constraint_name,
 column2 data_type(size) constraint_name,
 column3 data_type(size) constraint_name,....);
```

NOT NULL

- If we specify a field in a table to be NOT NULL. Then the field will never accept null value.
- You will be not allowed to insert a new row in the table without specifying any value to this field.
- For example, the below query creates a table Student with the fields ID and NAME as NOT NULL. That is, we are bound to specify values for these two fields every time we wish to insert a new row.

```
CREATE TABLE Student(
ID int(6) NOT NULL,
NAME varchar(10) NOT NULL,
ADDRESS varchar(20));
```

UNIQUE

- This constraint helps to uniquely identify each row in the table. i.e. for a particular column, all the rows should have unique values.
- We can have more than one UNIQUE columns in a table.
- For example, the below query creates a table Student where the field ID is specified as UNIQUE. i.e, no two students can have the same ID.

```
CREATE TABLE Student(
ID int(6) NOT NULL UNIQUE,
NAME varchar(10),
ADDRESS varchar(20));
```

PRIMARY KEY

- Primary Key is a field which uniquely identifies each row in the table.
- If a field in a table as primary key, then the field will not be able to contain NULL values as well as all the rows should have unique values for this field.
- So, in other words we can say that this is combination of NOT NULL and UNIQUE constraints.
- A table can have only one field as primary key.

PRIMARY KEY

- Query will create a table named Student and specifies the field ID as primary key.

```
CREATE TABLE Student(
ID int(6) NOT NULL UNIQUE,
NAME varchar(10),
ADDRESS varchar(20),
PRIMARY KEY(ID));
```

OR

```
CREATE TABLE Student(
ID int(6) NOT NULL PRIMARY KEY,
NAME varchar(10),
ADDRESS varchar(20));
```

FOREIGN KEY

- Foreign Key is a field in a table which uniquely identifies each row of a another table.
- That is, this field points to primary key of another table.
- This usually creates a kind of link between the tables.
- Consider the two tables as shown below:

FOREIGN KEY

- Consider the two tables as shown below:

Orders

O_ID	ORDER_NO	C_ID
1	2253	3
2	3325	3
3	4521	2
4	8532	1

Customers

C_ID	NAME	ADDRESS
1	RAMESH	DELHI
2	SURESH	NOIDA
3	DHARA	GURGAON

FOREIGN KEY

- The field C_ID in Orders table is the primary key in Customers table,
- It uniquely identifies each row in the Customers table. Therefore, it is a Foreign Key in Orders table.
- Syntax:

```
CREATE TABLE Orders(
 O_ID int NOT NULL,
 ORDER_NO int NOT NULL,
 C_ID  int,
 PRIMARY KEY (O_ID),
 FOREIGN KEY (C_ID) REFERENCES
 Customers(C_ID)
```

CHECK

- Using the CHECK constraint we can specify a condition for a field, which should be satisfied at the time of entering values for this field.
- For example, the below query creates a table Student and specifies the condition for the field AGE as (AGE >= 18).
- That is, the user will not be allowed to enter any record in the table with AGE < 18.

```
CREATE TABLE Student(
ID int(6) NOT NULL,
NAME varchar(10) NOT NULL,
AGE int NOT NULL CHECK (AGE >= 18));
```

DEFAULT

- This constraint is used to provide a default value for the fields.
- That is, if at the time of entering new records in the table if the user does not specify any value for these fields then the default value will be assigned to them.
- For example, the below query will create a table named Student and specify the default value for the field AGE as 18.

```
CREATE TABLE Student(
 ID int(6) NOT NULL,
 NAME varchar(10) NOT NULL,
 AGE int DEFAULT 18);
```

Operator in SQL

- An operator is a reserved word or a character used primarily in an SQL statement's WHERE clause to perform operation(s), such as comparisons and arithmetic operations.
- These Operators are used to specify conditions in an SQL statement and to serve as conjunctions for multiple conditions in a statement.
- Arithmetic operators
- Comparison operators
- Logical operators
- Operators used to negate conditions

Arithmetic Operators

- These operators are used to perform operations such as addition, multiplication, subtraction etc.

operator	Operation	Description
+	Addition	Add values on either side of the operator
–	Subtraction	Used to subtract the right hand side value from the left hand side value
*	Multiplication	Multiples the values present on each side of the operator
/	Division	Divides the left hand side value by the right hand side value
%	Modulus	Divides the left hand side value by the right hand side value; and returns the remainder

Arithmetic Operators

- **Example:**

```
SELECT 40 + 20;
SELECT 40 - 20;
SELECT 40 * 20;
SELECT 40 / 20;
SELECT 40 % 20;
```

- **Output:**

```
60
20
800
2
0
```

Comparison Operators

- These operators are used to perform operations such as equal to, greater than, less than etc.

Operator	Operation	Description
=	Equal to	Used to check if the values of both operands are equal or not. If they are equal, then it returns TRUE.
>	Greater than	Returns TRUE if the value of left operand is greater than the right operand.
<	Less than	Checks whether the value of left operand is less than the right operand, if yes returns TRUE.
>=	Greater than or equal to	Used to check if the left operand is greater than or equal to the right operand, and returns TRUE, if the condition is true.

Comparison Operators

- These operators are used to perform operations such as equal to, greater than, less than etc.

Operator	Operation	Description
<=	Less than or equal to	Returns TRUE if the left operand is less than or equal to the right operand.
<> or !=	Not equal to	Used to check if values of operands are equal or not. If they are not equal then, it returns TRUE.
!>	Not greater than	Checks whether the left operand is not greater than the right operand, if yes then returns TRUE.
!<	Not less than	Returns TRUE, if the left operand is not less than the right operand.

Comparison Operators

- **Example:**

Consider the following table to perform various operations.

StudentID	FirstName	LastName	Age
1	Atul	Mishra	23
2	Priya	Kapoor	21
3	Rohan	Singhania	21
4	Akanksha	Jain	20
5	Vaibhav	Gupta	25

Comparison Operators

- **Example[Use equal to]:**

```
SELECT * FROM Students
WHERE Age = 20;
```

- Output

StudentID	FirstName	LastName	Age
4	Akanksha	Jain	20

- **Example[Use greater than]:**

```
SELECT * FROM Students
WHERE Age > 23;
```

- Output

StudentID	FirstName	LastName	Age
5	Vaibhav	Gupta	25

Comparison Operators

- **Example[Use less than or equal to]:**

```
SELECT * FROM students
WHERE Age <= 21;
```

- **Output:**

StudentID	FirstName	LastName	Age
2	Priya	Kapoor	21
3	Rohan	Singhania	21
4	Akanksha	Jain	20

Logical Operators

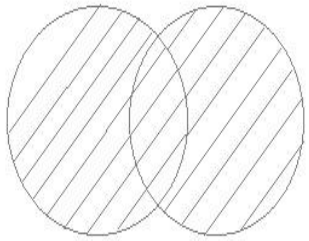
Operator	Description
IN	Used to compare a specific value to the literal values mentioned.
BETWEEN	Searches for values within the range mentioned.
AND	Allows the user to mention multiple conditions in a WHERE clause.
OR	Combines multiple conditions in a WHERE clause.
NOT	A negate operators, used to reverse the output of the logical operator.
EXISTS	Used to search for the row's presence in the table.
LIKE	Compares a pattern using wildcard operators.

Relational set operators

- SQL supports few Set operations which can be performed on the table data. These are used to get meaningful results from data stored in the table, under different special conditions.
- Different types of SET operations, along with example:
- UNION
- UNION ALL
- INTERSECT
- MINUS

UNION Operation

- UNION is used to combine the results of two or more SELECT statements.
- However it will eliminate duplicate rows from its resultset.
- In case of union, number of columns and datatype must be same in both the tables, on which UNION operation is being applied.



UNION OPERATION

- Example of UNION
- First Table

ID	Name
1	abhi
2	adam

Second Table

ID	Name
2	adam
3	Chester

•Union SQL query will be,

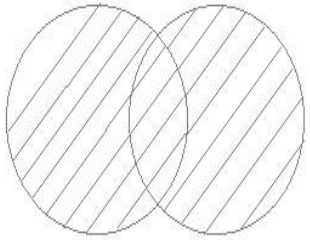
```
SELECT * FROM First
UNION
SELECT * FROM Second;
```

•The resultset table will look like

ID	NAME
1	abhi
2	adam
3	Chester

UNION ALL

- This operation is similar to Union. But it also shows the duplicate rows.



Union All query will be like,

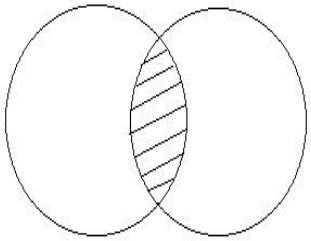
```
SELECT * FROM First
UNION ALL
SELECT * FROM Second;
```

The resultset table will look like,

ID	NAME
1	abhi
2	adam
2	adam
3	Chester

INTERSECT

- Intersect operation is used to combine two SELECT statements, but it only retuns the records which are common from both SELECT statements.
- In case of Intersect the number of columns and datatype must be same.



INTERSECT

- Example of Intersect
- Intersect query will be,

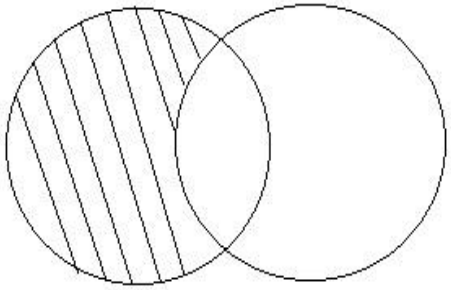
```
SELECT * FROM First
INTERSECT
SELECT * FROM Second;
```

- The resultset table will look like

ID	NAME
2	adam

MINUS

- The Minus operation combines results of two SELECT statements and return only those in the final result, which belongs to the first set of the result.



MINUS

- Example of Minus
- Minus query will be,

```
SELECT * FROM First
MINUS
SELECT * FROM Second;
```

- The resultset table will look like,

ID	NAME
1	abhi