Lab 2

Constraints, Integrity and Where clause

What is NULL value?

A NULL value in a table is a value in a field that appears to be blank, which means a field with a NULL value is a field with no value.

It is very important to understand that a NULL value is different than a zero value or a field that contains spaces. A field with a NULL value is one that has been left blank during record creation.

SQL Constraints:

Constraints are the rules enforced on data columns on table. These 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 database.

Constraints could be column level or table level. Column level constraints are applied only to one column , whereas table level constraints are applied to the whole table.

Following are commonly used constraints available in SQL:

- NOT NULL Constraint: Ensures that a column cannot have NULL value.
- DEFAULT Constraint: Provides a default value for a column when none is specified.
- UNIQUE Constraint: Ensures that all values in a column are different.
- PRIMARY Key: Uniquely identified each rows/records in a database table.
- FOREIGN Key: Uniquely identified a rows/records in any another database table.
- CHECK Constraint: The CHECK constraint ensures that all values in a column satisfy certain conditions.
- INDEX: Use to create and retrieve data from the database very quickly.

NOT NULL Constraint:

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 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.

Example:

For example, the following SQL creates a new table called CUSTOMERS and adds five columns, three of which, ID and NAME and AGE, specify not to accept NULLs:

```
CREATE TABLE CUSTOMERS(
        ID   INT              NOT NULL,
        NAME VARCHAR (20)     NOT NULL,
        AGE  INT              NOT NULL,
        ADDRESS  CHAR (25) ,
        SALARY   DECIMAL (18, 2),
        PRIMARY KEY (ID)
);
```

DEFAULT Constraint:

The DEFAULT constraint provides a default value to a column when the INSERT INTO statement does not provide a specific value.

Example:

For example, the following SQL creates a new table called CUSTOMERS and adds five columns. Here, SALARY column is set to 5000.00 by default, so in case INSERT INTO statement does not provide a value for this column.then by default this column would be set to 5000.00.

```
CREATE TABLE CUSTOMERS(
        ID   INT              NOT NULL,
        NAME VARCHAR (20)     NOT NULL,
        AGE  INT              NOT NULL,
        ADDRESS  CHAR (25) ,
        SALARY   DECIMAL (18, 2) DEFAULT 5000.00,
        PRIMARY KEY (ID)
);
```

Drop Default Constraint:

To drop a DEFAULT constraint, use the following SQL:

```
ALTER TABLE CUSTOMERS
   ALTER COLUMN SALARY DROP DEFAULT;
```

UNIQUE Constraint:

The UNIQUE Constraint prevents two records from having identical values in a particular column. In the CUSTOMERS table, for example, you might want to prevent two or more people from having identical age.

Example:

For example, the following SQL creates a new table called CUSTOMERS and adds five columns. Here, AGE column is set to UNIQUE, so that you can not have two records with same age:

```
CREATE TABLE CUSTOMERS(
        ID   INT              NOT NULL,
        NAME VARCHAR (20)     NOT NULL,
        AGE  INT              NOT NULL UNIQUE,
        ADDRESS  CHAR (25) ,
        SALARY   DECIMAL (18, 2),
        PRIMARY KEY (ID)
);
```

You can also use following syntax, which supports naming the constraint in multiple columns as well:

```
ALTER TABLE CUSTOMERS
   ADD CONSTRAINT myUniqueConstraint UNIQUE(AGE, SALARY);
```

DROP a UNIQUE Constraint:

To drop a UNIQUE constraint, use the following SQL:

```
ALTER TABLE CUSTOMERS
   DROP CONSTRAINT myUniqueConstraint;
```

If you are using MySQL, then you can use the following syntax:

```
ALTER TABLE CUSTOMERS
   DROP INDEX myUniqueConstraint;
```

PRIMARY Key:

A primary key is a field in a table which uniquely identifies each row/record in a database table. Primary keys must contain unique values. A primary key column cannot have NULL values.

A table can have only one primary key, which may consist of single or multiple fields. When multiple fields are used as a primary key, they are called a composite key.

If a table has a primary key defined on any field(s), then you can not have two records having the same value of that field(s).

Note: You would use these concepts while creating database tables.

Create Primary Key:

syntax :

```
CREATE TABLE CUSTOMERS(
       ID   INT              NOT NULL,
       NAME VARCHAR (20)     NOT NULL,
       AGE  INT              NOT NULL,
       ADDRESS  CHAR (25) ,
       SALARY   DECIMAL (18, 2),
       PRIMARY KEY (ID)
);
```

For defining a PRIMARY KEY constraint on multiple columns, use the following SQL syntax:

```
CREATE TABLE CUSTOMERS(
       ID   INT              NOT NULL,
       NAME VARCHAR (20)     NOT NULL,
       AGE  INT              NOT NULL,
       ADDRESS  CHAR (25) ,
       SALARY   DECIMAL (18, 2),
       PRIMARY KEY (ID, NAME)
);
```

To create a PRIMARY KEY constraint on the "ID" and "NAMES" columns when CUSTOMERS table already exists, use the following SQL syntax:

```
ALTER TABLE CUSTOMERS
   ADD CONSTRAINT PK_CUSTID PRIMARY KEY (ID, NAME);
```

Delete Primary key:

Alter table *table name*

Drop constraint *constraint_name*

FOREIGN Key:

A foreign key is a key used to link two tables together. This is sometimes called a referencing key.

Foreign Key is a column or a combination of columns whose values match a Primary Key in a different table.

The relationship between 2 tables matches the Primary Key in one of the tables with a Foreign Key in the second table. If a table has a primary key defined on any field(s), then you can not have two records having the same value of that field(s).

Example:

Consider the structure of the two tables as follows:

CUSTOMERS table:

```
CREATE TABLE CUSTOMERS(
        ID   INT              NOT NULL,
        NAME VARCHAR (20)     NOT NULL,
        AGE  INT              NOT NULL,
        ADDRESS  CHAR (25) ,
        SALARY   DECIMAL (18, 2),
        PRIMARY KEY (ID)
);
```

ORDERS table:

```
CREATE TABLE ORDERS (
        ID           INT         NOT NULL,
        DATE         DATETIME,
        CUSTOMER_ID INT references CUSTOMERS(ID),
        AMOUNT     double,
        PRIMARY KEY (ID)
);
```

DROP a FOREIGN KEY Constraint:

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

```
ALTER TABLE ORDERS
   DROP FOREIGN KEY;
```

CHECK Constraint:

The CHECK Constraint enables a condition to check the value being entered into a record. If the condition evaluates to false, the record violates the constraint and isn't entered into the table.

Example:

For example, the following SQL creates a new table called CUSTOMERS and adds five columns. Here, we add a CHECK with AGE column, so that you can not have any CUSTOMER below 18 years:

```
CREATE TABLE CUSTOMERS(
        ID   INT              NOT NULL,
        NAME VARCHAR (20)     NOT NULL,
        AGE  INT              NOT NULL CHECK (AGE >= 18),
        ADDRESS  CHAR (25) ,
        SALARY   DECIMAL (18, 2),
        PRIMARY KEY (ID)
);
```

SQL WHERE Clause

The SQL WHERE clause is used to specify a condition while fetching the data from single table or joining with multiple tables.
If the given condition is satisfied, then only it returns specific value from the table. You would use WHERE clause to filter the records and fetching only necessary records.

The WHERE clause is not only used in SELECT statement, but it is also used in UPDATE, DELETE statement,etc., which we would examine in subsequent chapters.

Syntax:
SELECT column1, column2, columnN
FROM table_name
WHERE [condition]

You can specify a condition using comparison or logical operators like >, <, =, LIKE, NOT etc. Below examples would make this concept clear. Example:

```
SQL> SELECT ID, NAME, SALARY
FROM CUSTOMERS
WHERE SALARY > 2000;
```

This would produce the following result:

```
+----+----------+----------+
| ID | NAME     | SALARY   |
+----+----------+----------+
|  4 | Chaitali |  6500.00 |
|  5 | Hardik   |  8500.00 |
|  6 | Komal    |  4500.00 |
|  7 | Muffy    | 10000.00 |
+----+----------+----------+
```

SQL LIKE Clause

The SQL LIKE clause is used to compare a value to similar values using wildcard operators. There are two wildcards used in conjunction with the LIKE operator:

- The percent sign (%)
- The underscore (_)

The percent sign represents zero, one, or multiple characters. The underscore represents a single number or character. The symbols can be used in combinations.

Syntax:

The basic syntax of % and _ is as follows:

```
SELECT FROM table_name
WHERE column LIKE 'XXXX%'

or

SELECT FROM table_name
WHERE column LIKE '%XXXX%'

or

SELECT FROM table_name
WHERE column LIKE 'XXXX_'

or

SELECT FROM table_name
WHERE column LIKE '_XXXX'

or

SELECT FROM table_name
WHERE column LIKE '_XXXX_'
```

You can combine N number of conditions using AND or OR operators. Here, XXXX could be any numeric or string value.

Example:

Here are number of examples showing WHERE part having different LIKE clause with '%' and '_' operators:

Statement	Description
WHERE SALARY LIKE '200%'	Finds any values that start with 200
WHERE SALARY LIKE '%200%'	Finds any values that have 200 in any position
WHERE SALARY LIKE '_00%'	Finds any values that have 00 in the second and third positions
WHERE SALARY LIKE '2_%_%'	Finds any values that start with 2 and are at least 3 characters in length
WHERE SALARY LIKE '%2'	Finds any values that end with 2
WHERE SALARY LIKE '_2%3'	Finds any values that have a 2 in the second position and end with a 3
WHERE SALARY LIKE '2___3'	Finds any values in a five-digit number that start with 2 and end with 3

SQL AND and OR Operators

The SQL AND and OR operators are used to combine multiple conditions to narrow data in an SQL statement. These two operators are called conjunctive operators.

These operators provide a means to make multiple comparisons with different operators in the same SQL statement.

The AND Operator:
The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause.
Syntax:
The basic syntax of AND operator with WHERE clause is as follows:

```
SELECT column1, column2, columnN
FROM table_name
WHERE [condition1] AND [condition2]...AND [conditionN];
```

You can combine N number of conditions using AND operator. For an action to be taken by the SQL statement, whether it be a transaction or query, all conditions separated by the AND must be TRUE.
Example:

```
SQL> SELECT ID, NAME, SALARY
FROM customers
WHERE SALARY > 2000 AND age < 25;
```

The OR Operator:
The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.

Syntax:

```
SELECT column1, column2, columnN
FROM table_name
WHERE [condition1] OR [condition2]...OR [conditionN]
```

You can combine N number of conditions using OR operator. For an action to be taken by the SQL statement, whether it be a transaction or query, only any ONE of the conditions separated by the OR must be TRUE.

SQL Operators
What is an 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.
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

SQL Arithmetic Operators:

Assume variable a holds 10 and variable b holds 20, then:

Operator	Description	Example
+	Addition - Adds values on either side of the operator	a + b will give 30
-	Subtraction - Subtracts right hand operand from left hand operand	a - b will give -10
*	Multiplication - Multiplies values on either side of the operator	a * b will give 200
/	Division - Divides left hand operand by right hand operand	b / a will give 2
%	Modulus - Divides left hand operand by right hand operand and returns remainder	b % a will give 0

Here are simple examples showing usage of SQL Arithmetic Operators:

```
SQL> select 10+ 20;
+---------+
| 10+ 20  |
+---------+
|      30 |
+---------+
1 row in set (0.00 sec)

SQL> select 10 * 20;
+----------+
| 10 * 20  |
+----------+
|      200 |
+----------+
1 row in set (0.00 sec)

SQL> select 10 / 5;
+---------+
| 10 / 5  |
+---------+
| 2.0000  |
+---------+
1 row in set (0.03 sec)

SQL> select 12 %  5;
+----------+
| 12 %  5  |
+----------+
|        2 |
+----------+
1 row in set (0.00 sec)
```

SQL Comparison Operators:

Assume variable a holds 10 and variable b holds 20, then:

Operator	Description	Example
=	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(a = b) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(a != b) is true.
<>	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(a <> b) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(a > b) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(a < b) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(a >= b) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(a <= b) is true.
!<	Checks if the value of left operand is not less than the value of right operand, if yes then condition becomes true.	(a !< b) is false.
!>	Checks if the value of left operand is not greater than the value of right operand, if yes then condition becomes true.	(a !> b) is true.

SQL Logical Operators:

Here is a list of all the logical operators available in SQL.

Operator	Description
ALL	The ALL operator is used to compare a value to all values in another value set.
AND	The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause.
ANY	The ANY operator is used to compare a value to any applicable value in the list according to the condition.
BETWEEN	The BETWEEN operator is used to search for values that are within a set of values, given the minimum value and the maximum value.
EXISTS	The EXISTS operator is used to search for the presence of a row in a specified table that meets certain criteria.
IN	The IN operator is used to compare a value to a list of literal values that have been specified.
LIKE	The LIKE operator is used to compare a value to similar values using wildcard operators.
NOT	The NOT operator reverses the meaning of the logical operator with which it is used. Eg: NOT EXISTS, NOT BETWEEN, NOT IN, etc. **This is a negate operator.**
OR	The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.
IS NULL	The NULL operator is used to compare a value with a NULL value.
UNIQUE	The UNIQUE operator searches every row of a specified table for uniqueness (no duplicates).

Some of the examples are:

SELECT * FROM CUSTOMERS WHERE AGE IS NOT NULL;

```
SQL>  SELECT * FROM CUSTOMERS WHERE AGE IS NOT NULL;
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
```

```
SQL> SELECT * FROM CUSTOMERS WHERE AGE IN ( 25, 27 );
+----+----------+-----+---------+---------+
| ID | NAME     | AGE | ADDRESS | SALARY  |
+----+----------+-----+---------+---------+
|  2 | Khilan   |  25 | Delhi   | 1500.00 |
|  4 | Chaitali |  25 | Mumbai  | 6500.00 |
```

```
SQL> SELECT * FROM CUSTOMERS WHERE AGE BETWEEN 25 AND 27;
+----+----------+-----+---------+---------+
| ID | NAME     | AGE | ADDRESS | SALARY  |
+----+----------+-----+---------+---------+
|  2 | Khilan   |  25 | Delhi   | 1500.00 |
|  4 | Chaitali |  25 | Mumbai  | 6500.00 |
```

```
SQL> SELECT AGE FROM CUSTOMERS
WHERE EXISTS (SELECT AGE FROM CUSTOMERS WHERE SALARY > 6500);
+-----+
| AGE |
+-----+
|  32 |
|  25 |
|  23 |
|  25 |
```

```
SQL> SELECT * FROM CUSTOMERS
WHERE AGE > ALL (SELECT AGE FROM CUSTOMERS WHERE SALARY > 6500);
+----+--------+-----+-----------+---------+
| ID | NAME   | AGE | ADDRESS   | SALARY  |
+----+--------+-----+-----------+---------+
|  1 | Ramesh |  32 | Ahmedabad | 2000.00 |
```

```
SQL> SELECT * FROM CUSTOMERS
WHERE AGE > ANY (SELECT AGE FROM CUSTOMERS WHERE SALARY > 6500);
+----+----------+-----+-----------+---------+
| ID | NAME     | AGE | ADDRESS   | SALARY  |
+----+----------+-----+-----------+---------+
|  1 | Ramesh   |  32 | Ahmedabad | 2000.00 |
|  2 | Khilan   |  25 | Delhi     | 1500.00 |
|  4 | Chaitali |  25 | Mumbai    | 6500.00 |
```

SQL Expressions

An expression is a combination of one or more values, operators, and SQL functions that evaluate to a value.

SQL EXPRESSIONs are like formulas and they ar

e written in query language. You can also use them to query the database for specific set of data.

Syntax:

SELECT column1, column2, columnN

FROM table_name

WHERE [CONDITION|EXPRESSION];

Example:

SQL> SELECT COUNT(*) AS "RECORDS" FROM CUSTOMERS;

Lab task:

Create the relations as below:

Employee (eid, ename, dateofemploy, salary)

Booklist(isbn, name, publication)

Book(bid, bname, author, price)

Issues(IID,name,dateofissue)

1. Modify relation teacher and student
   i. Set Tid as foreign key
   ii. Set SID as primary key
   iii. Delete RN attribute.
2. Set default value of 'dateofemploy' attribute as jan 1, 2010.
3. Assign Bid and iid as foreign key.
4. All the price of books must be less than 5000.
5. Ename , bname, name attribute of each relation must contain some value.
6. Insert any 4 records in each relation.
7. Display all records from all relations.
8. Display eid and ename of all employes whose salary is less than 10000.
9. Display all record of book whose price ranges from 2500 to 5000.
10. Display all the records from booklist relation whose publication name starts from 'E' eg Ekta
11. Display all records from employee table whose name ends with 'ta' eg Sita, Geeta etc.
12. Display iid and name from issues table whose name exactly consist of 5 characters.
13. Display all records from employee table where name starts with 'S' and salary greater than 10000.
14. Display all records from book table where either bookid lies in range 1001 to 2000 or price range in 1000 to 2500.
15. Display isbn number and bookname where booklist must not contain isbn no. 1003