LAB:1

Introduction to SQL

SQL Overview

SQL is a language of database, it includes database creation, deletion, fetching rows and modifying rows etc.SQL is an ANSI (American National Standards Institute) standard, but there are many different versions of the SQL language.

What is SQL?
SQL is Structured Query Language, which is a computer language for storing, manipulating and retrieving data stored in relational database.

SQL is the standard language for Relation Database System. All relational database management systems like MySQL, MS Access, Oracle, Sybase, Informix, postgres and SQL Server use SQL as standard database language.

Also, they are using different dialects, such as:

- MS SQL Server using T-SQL,
- Oracle using PL/SQL,
- MS Access version of SQL is called JET SQL (native format) etc.

Why SQL?

- Allows users to access data in relational database management systems.
- Allows users to describe the data.
- Allows users to define the data in database and manipulate that data.
- Allows to embed within other languages using SQL modules, libraries & pre -compilers.
- Allows users to create and drop databases and tables.
- Allows users to create view, stored procedure, functions in a database.
- Allows users to set permissions on tables, procedures and views

History:

- 1970 -- Dr. E. F. "Ted" of IBM is known as the father of relational databases. He described a relational model for databases.
- 1974 -- Structured Query Language appeared.
- 1978 -- IBM worked to develop Codd's ideas and released a product named System/R.

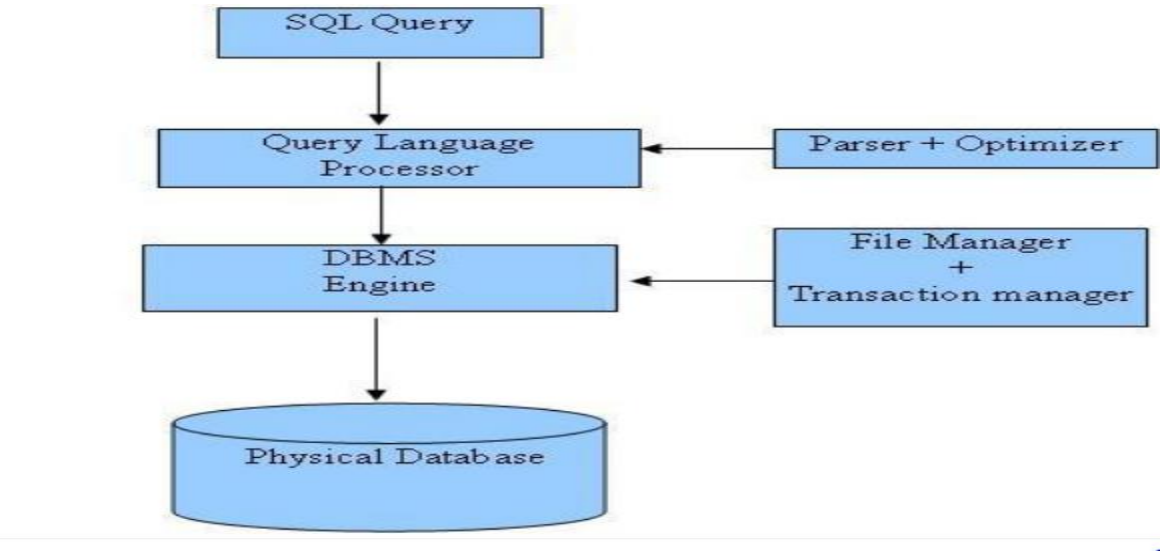
 1986 -- IBM developed the first prototype of relational database and standardized by ANSI. The first relational database was released by Relational Software and its later becoming Oracle.

SQL Process:

When you are executing an SQL command for any RDBMS, the system determines the best way to carry out your request and SQL engine figures out how to interpret the task.

There are various components included in the process. These components are Query Dispatcher, Optimization Engines, Classic Query Engine and SQL Query Engine, etc. Classic query engine handles all non-SQL queries, but SQL query engine won't handle logical files.

Following is a simple diagram showing SQL Architecture:



SQL Commands:

The standard SQL commands to interact with relational databases are CREATE, SELECT, INSERT, UPDATE, DELETE and DROP. These commands can be classified into groups based on their nature:

DDL - Data Definition Language:

Command	Description
CREATE	Creates a new table, a view of a table, or other object in database
ALTER	Modifies an existing database object, such as a table.
DROP	Deletes an entire table, a view of a table or other object in the database.

DML - Data Manipulation Language:

Command	Description
INSERT	Creates a record
UPDATE	Modifies records
DELETE	Deletes records

DCL - Data Control Language:

Command	Description
GRANT	Gives a privilege to user
REVOKE	Takes back privileges granted from user

DQL - Data Query Language:

Command	Description
SELECT	Retrieves certain records from one or more tables

What is RDBMS?

RDBMS stands for Relational Database Management System. RDBMS is the basis for SQL and for all modern database systems like MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.

A Relational database management system (RDBMS) is a database management system (DBMS) that is based on the relational model as introduced by E. F. Codd.

What is table?

The data0 in RDBMS is stored in database objects called tables. The table is a collection of related data entries and it consists of columns and rows.

Remember, a table is the most common and simplest form of data storage in a relational database. Following is the example of a CUSTOMERS table:

```
+----+----------+-----+-----------+----------+
| 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 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

What is field?

Every table is broken up into smaller entities called fields. The fields in the CUSTOMERS table consist of ID, NAME, AGE, ADDRESS and SALARY.

A field is a column in a table that is designed to maintain specific information about every record in the table.

What is record or row?

A record, also called a row of data, is each individual entry that exists in a table. For example , there are 7 records in the above CUSTOMERS table. Following is a single row of data or record in the CUSTOMERS table:

```
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
+----+----------+-----+-----------+----------+
```

A record is a horizontal entity in a table.

What is column?

A column is a vertical entity in a table that contains all information associated with a specific field in a table.For example, a column in the CUSTOMERS table is ADDRESS, which represents location description and would consist of the following:

```
+------------+
| ADDRESS    |
+------------+
| Ahmedabad  |
| Delhi      |
| Kota       |
| Mumbai     |
| Bhopal     |
| MP         |
| Indore     |
+----+-------+
```

SQL Data Types

SQL data type is an attribute that specifies type of data of any object. Each column, variable and expression has related data type in SQL.

You would use these data types while creating your tables. You would choose a particular data type for a table column based on your requirement.

SQL Server offers six categories of data types for your use:

Exact Numeric Data Types

DATA TYPE	FROM	TO
Bigint	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
Int	-2,147,483,648	2,147,483,647
Smallint	-32,768	32,767
Tinyint	0	255
Bit	0	1
Decimal	-10^38 +1	10^38 -1
Numeric	-10^38 +1	10^38 -1
Money	-922,337,203,685,477.5808	+922,337,203,685,477.5807
Smallmoney	-214,748.3648	+214,748.3647

Approximate Numeric Data Types:

DATA TYPE	FROM	TO
Float	-1.79E + 308	1.79E + 308
Real	-3.40E + 38	3.40E + 38

Date and Time Data Types:

DATA TYPE	FROM	TO
Datetime	Jan 1, 1753	Dec 31, 9999
Smalldatetime	Jan 1, 1900	Jun 6, 2079
Date	Stores a date like June 30, 1991	
Time	Stores a time of day like 12:30 P.M.	

Character Strings Data Types:

DATA TYPE	FROM	TO
Char	Char	Maximum length of 8,000 characters.(Fixed length non-Unicode characters)
Varchar	Varchar	Maximum of 8,000 characters.(Variable-length non-Unicode data).
varchar(max)	varchar(max)	Maximum length of 231characters, Variable-length non-Unicode data (SQL Server 2005 only).
Text	text	Variable-length non-Unicode data with a maximum length of 2,147,483,647 characters.

Unicode Character Strings Data Types:

DATA TYPE	Description
Nchar	Maximum length of 4,000 characters.(Fixed length Unicode)
Nvarchar	Maximum length of 4,000 characters.(Variable length Unicode)
nvarchar(max)	Maximum length of 231characters (SQL Server 2005 only).(Variable length Unicode)
Ntext	Maximum length of 1,073,741,823 characters. (Variable length Unicode)

Binary Data Types:

DATA TYPE	Description
Binary	Maximum length of 8,000 bytes(Fixed-length binary data)
Varbinary	Maximum length of 8,000 bytes.(Variable length binary data)
varbinary(max)	Maximum length of 231 bytes (SQL Server 2005 only). (Variable length Binary data)
Image	Maximum length of 2,147,483,647 bytes. (Variable length Binary Data)

Misc Data Types:

DATA TYPE	Description
sql_variant	Stores values of various SQL Server-supported data types, except text, ntext, and timestamp.
timestamp	Stores a database-wide unique number that gets updated every time a row gets updated
uniqueidentifier	Stores a globally unique identifier (GUID)
xml	Stores XML data. You can store xml instances in a column or a variable (SQL Server 2005 only).
cursor	Reference to a cursor object
table	Stores a result set for later processing

SQL CREATE Database

The SQL CREATE DATABASE statement is used to create new SQL database.

Syntax:

CREATE DATABASE DatabaseName;

Always database name should be unique within the RDBMS.

Example:

If you want to create new database <testDB>, then CREATE DATABASE statement would be as follows:

```
SQL> CREATE DATABASE testDB;
```

DROP or DELETE Database

The SQL DROP DATABASE statement is used to drop an existing database in SQL schema.

Syntax:

```
DROP DATABASE DatabaseName;
```

Example:

If you want to delete an existing database <testDB>, then DROP DATABASE statement would be as follows:

```
SQL> DROP DATABASE testDB;
```

NOTE: Be careful before using this operation because by deleting an existing database would result in loss of complete information stored in the database.

Make sure you have admin privilege before dropping any database.

SQL CREATE Table

Creating a basic table involves naming the table and defining its columns and each column's data type.

The SQL CREATE TABLE statement is used to create a new table.

Syntax:

```
CREATE TABLE table_name(
   column1 datatype,
   column2 datatype,
   column3 datatype,
   .....
   columnN datatype,
   PRIMARY KEY( one or more columns )
);
```

CREATE TABLE is the keyword telling the database system what you want to do. In this case, you want to create a new table. The unique name or identifier for the table follows the CREATE TABLE statement.Then in brackets comes the list defining each column in the table and what sort of data type it is. The syntax becomes clearer with an example below.

A copy of an existing table can be created using a combination of the CREATE TABLE statement and the SELECT statement. You can check complete details at Create Table Using another Table.

Create Table Using another Table (Note: it may not work in MS Sql server)

A copy of an existing table can be created using a combination of the CREATE TABLE statement and the SELECT statement.

The new table has the same column definitions. All columns or specific columns can be selected.

When you create a new table using existing table, new table would be populated using existing values in the old table.

Syntax:

```
CREATE TABLE NEW_TABLE_NAME AS
   SELECT [ column1, column2...columnN ]
   FROM EXISTING_TABLE_NAME
   [ WHERE ]
```

Example:

Following is an example, which would create a table SALARY using CUSTOMERS table and having fields customer ID and customer SALARY:

```
SQL> CREATE TABLE SALARY AS
   SELECT ID, SALARY
   FROM CUSTOMERS;
```

SQL DROP or DELETE Table

The SQL DROP TABLE statement is used to remove a table definition and all data, indexes, triggers, constraints, and permission specifications for that table.

NOTE: You have to be careful while using this command because once a table is deleted then all the information available in the table would also be lost forever. Syntax:

```
DROP TABLE table_name;
```

SQL INSERT Query

The SQL INSERT INTO Statement is used to add new rows of data to a table in the database.
Syntax:

```
INSERT INTO TABLE_NAME (column1, column2, column3,...columnN)]
VALUES (value1, value2, value3,...valueN);
```

Here, column1, column2,...columnN are the names of the columns in the table into which you want to insert data. You may not need to specify the column(s) name in the SQL query if you are adding values for all the columns of the table. But make sure the order of the values is in the same order as the columns in the table.

The SQL INSERT INTO syntax would be as follows:

```
INSERT INTO TABLE_NAME VALUES (value1,value2,value3,...valueN);
```

Example:

Following statements would create two records in CUSTOMERS table:

```
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (1, 'Ramesh', 32, 'Ahmedabad', 2000.00 );

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (2, 'Khilan', 25, 'Delhi', 1500.00 );
```

Populate one table using another table:

You can populate data into a table through select statement over another table provided another table has a set of fields, which are required to populate first table. Here is the syntax:

```
INSERT INTO first_table_name [(column1, column2, ... columnN)]
   SELECT column1, column2, ...columnN
   FROM second_table_name
   [WHERE condition];
```

SQL SELECT Query

SQL SELECT Statement is used to fetch the data from a database table which returns data in the form of result table. These result tables are called result-sets.

Syntax:

```
SELECT column1, column2, columnN FROM table_name;
```

Here, column1, column2...are the fields of a table whose values you want to fetch. If you want to fetch all the fields available in the field, then you can use the following syntax:

```
SELECT * FROM table_name;
```

Example:

Consider the CUSTOMERS table having the following records:

```
+----+----------+-----+-----------+----------+
| 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 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Following is an example, which would fetch ID, Name and Salary fields of the customers available in CUSTOMERS table:

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

```
+----+----------+----------+
| ID | NAME     | SALARY   |
+----+----------+----------+
|  1 | Ramesh   |  2000.00 |
|  2 | Khilan   |  1500.00 |
|  3 | kaushik  |  2000.00 |
|  4 | Chaitali |  6500.00 |
|  5 | Hardik   |  8500.00 |
|  6 | Komal    |  4500.00 |
|  7 | Muffy    | 10000.00 |
+----+----------+----------+
```

SELECT INTO Statement

The SQL Server (Transact-SQL) SELECT INTO statement is used to create a table from an existing table by copying the existing table's columns.

It is important to note that when creating a table in this way, the new table will be populated with the records from the existing table (based on the SELECT Statement).

Syntax

SELECT expressions
INTO new_table
```
FROM tables
WHERE conditions;
```

The ALTER TABLE Statement

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.

Syntax

To add a column in a table, use the following syntax:
ALTER TABLE table_name
ADD column_name datatype

To delete a column in a table, use the following (notice that some database systems don't allow deleting a column):

syntax
ALTER TABLE table_name
DROP COLUMN column_name

To change the data type of a column in a table, use the following :Syntax:

SQL Server / MS Access:
ALTER TABLE table_name
ALTER COLUMN column_name datatype

My SQL / Oracle (prior version 10G):
ALTER TABLE table_name
MODIFY COLUMN column_name datatype

Oracle 10G and later:
ALTER TABLE table_name
MODIFY column_name datatype

```
To rename the column table:
Syntax :
sp_rename 'table_name.old_column_name', 'new_column_name', 'COLUMN';

To rename existing table:
sp_rename 'old_table_name', 'new_table_name';
```

LAB TASK:

Create a database named ‘HCOE’ and create tables as following shcema:

Student(ID,Name,RN,Batch)

Teacher(TID,Name, Faculty)

1. Insert any five records in each table
2. Display all records.
3. Display only *ID* and *Name* from student table.
4. Display *Name* and *faculty* from Teacher table.
5. Remove ‘*RN’* attribute from student relation.
6. Add ‘*salary’* attribute to teacher relation.
7. Copy *ID* and *name* attribute to new relation ‘*info_student’*.
8. Delete all contents from *info_student* relation

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

Lab 3

Update Query, Delete query & Sub Query

CARTESIAN JOIN

The CARTESIAN JOIN or CROSS JOIN returns the cartesian product of the sets of records from the two or more joined tables. Thus, it equates to an inner join where the join-condition always evaluates to True or where the join condition is absent from the statement.

Syntax:

The basic syntax of INNER JOIN is as follows:

SELECT table1.column1, table2.column2...

FROM table1, table2 [, table3]

Example:

SELECT ID, NAME, AMOUNT, DATE

FROM CUSTOMERS, ORDERS;

UPDATE Query

The SQL UPDATE Query is used to modify the existing records in a table.You can use WHERE clause with UPDATE query to update selected rows, otherwise all the rows would be affected.
Syntax:
UPDATE table_name
SET column1 = value1, column2 = value2...., columnN = valueN
WHERE [condition];
You can combine N number of conditions using AND or OR operators.

Example:
UPDATE CUSTOMERS
SET ADDRESS = 'Pune'
WHERE ID = 6;
Output:

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
5	Hardik	27	Bhopal	8500.00
6	Komal	22	Pune	4500.00
7	Muffy	24	Indore	10000.00

Example:

```
UPDATE CUSTOMERS
SET ADDRESS = 'Pune', SALARY = 1000.00;
```

Output:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Pune	1000.00
2	Khilan	25	Pune	1000.00
3	kaushik	23	Pune	1000.00
4	Chaitali	25	Pune	1000.00
5	Hardik	27	Pune	1000.00
6	Komal	22	Pune	1000.00
7	Muffy	24	Pune	1000.00

SQL DELETE Query

The SQL DELETE Query is used to delete the existing records from a table.You can use WHERE clause with DELETE query to delete selected rows, otherwise all the records would be deleted.

Syntax:

```
DELETE FROM table_name
WHERE [condition];
```

You can combine N number of conditions using AND or OR operators.

Example:

```
DELETE FROM CUSTOMERS
WHERE ID = 6;
```

Output:

1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
7	Muffy	24	Indore	10000.00

SQL TRUNCATE TABLE

The SQL TRUNCATE TABLE command is used to delete complete data from an existing table.
You can also use DROP TABLE command to delete complete table but it would remove complete table structure form the database and you would need to re-create this table once again if you wish you store some data.
Syntax:
The basic syntax of TRUNCATE TABLE is as follows:
TRUNCATE TABLE table_name;

Example:
TRUNCATE TABLE CUSTOMERS
Output:
SELECT * FROM CUSTOMERS;
Empty set (0.00 sec)

SQL Sub Queries

A Sub query or Inner query or Nested query is a query within another SQL query and embedded within the WHERE clause.
A sub query is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.
Sub queries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN etc.
There are a few rules that sub queries must follow:
- Subqueries must be enclosed within parentheses.
- A subquery can have only one column in the SELECT clause, unless multiple columns are in the main query for the subquery to compare its selected columns.
- An ORDER BY cannot be used in a subquery, although the main query can use an ORDER BY. The GROUP BY can be used to perform the same function as the ORDER BY in a subquery.
- Subqueries that return more than one row can only be used with multiple value operators, such as the IN operator.
- The SELECT list cannot include any references to values that evaluate to a BLOB, ARRAY, CLOB, or NCLOB.
- A subquery cannot be immediately enclosed in a set function.
- The BETWEEN operator cannot be used with a subquery; however, the BETWEEN operator can be used within the subquery.

Subqueries with the SELECT Statement:
Subqueries are most frequently used with the SELECT statement. The basic syntax is as follows:

```
SELECT column_name [, column_name ]
FROM table1 [, table2 ]
WHERE column_name OPERATOR
 (SELECT column_name [, column_name ]
FROM table1 [, table2 ]
[WHERE])
```

Example:

```
SELECT *
FROM CUSTOMERS
WHERE ID IN (SELECT ID
FROM CUSTOMERS
WHERE SALARY > 4500) ;
```

Subqueries with the INSERT Statement:

Subqueries also can be used with INSERT statements. The INSERT statement uses the data returned from the subquery to insert into another table. The selected data in the subquery can be modified with any of the character, date or number functions.
The basic syntax is as follows:

```
INSERT INTO table_name [ (column1 [, column2 ]) ]
SELECT [ *|column1 [, column2 ]
FROM table1 [, table2 ]
[ WHERE VALUE OPERATOR ]
```

Example:

```
INSERT INTO CUSTOMERS_BKP
SELECT * FROM CUSTOMERS  WHERE ID IN (SELECT ID FROM CUSTOMERS) ;
```

Subqueries with the UPDATE Statement:
The subquery can be used in conjunction with the UPDATE statement. Either single or multiple columns in a table can be updated when using a subquery with the UPDATE statement.
The basic syntax is as follows:

```
UPDATE table
SET column_name = new_value
[ WHERE OPERATOR [ VALUE ]
(SELECT COLUMN_NAME
FROM TABLE_NAME)
[ WHERE) ]
```

Example:

Update SALARY by 0.25 times in CUSTOMERS table for all the customers whose AGE is greater than or equal to 27:

```
SQL> UPDATE CUSTOMERS
SET SALARY = SALARY * 0.25
WHERE AGE IN (SELECT AGE FROM CUSTOMERS_BKP
WHERE AGE >= 27 );
```

Subqueries with the DELETE Statement:

The subquery can be used in conjunction with the DELETE statement like with any other statements mentioned above.

The basic syntax is as follows:

```
DELETE FROM TABLE_NAME
[ WHERE OPERATOR [ VALUE ]
(SELECT COLUMN_NAME
FROM TABLE_NAME)
[ WHERE) ]
```

Example:

Assuming, we have CUSTOMERS_BKP table available which is backup of CUSTOMERS table.

Following example deletes records from CUSTOMERS table for all the customers whose AGE is greater than or equal to 27:

```
SQL> DELETE FROM CUSTOMERS
WHERE AGE IN (SELECT AGE FROM CUSTOMERS_BKP
WHERE AGE > 27 );
```

Lab Task:

1. Find all the bookname, publication name and author name where publication name is “Ekta”.
2. Find the teachers name and faculty who issued book on jan 1 ,2015.
3. Find the employee name whose salary is greater then 10000 and faculty is “computer”.
4. Add attribute bid on Issues relation .
5. Insert the data in bid column.
6. Find the Teacher’s name , and book name issued by the teacher whose name starts with “S”.
7. Update all salary by 10 %.
8. Update book name DBMS as DATABASE.
9. Update the salary of all employee by 20% whose salary is less than 5000.
10. Provide 5% increment to all salaries whose salary is greater than 20000 and 20% increment in rest of salaries.
    (use CASE WHEN <CONDITION>THEN <STATEMENT> ELSE<STATEMENT> END).
11. Delete the records from employee table whose eid is 111.
12. Use sub query to find all teachers name and faculty whose date of employee is jan 2., 2011
13. Use sub query to find all the book name and author name whose publication is “shaja prakashan”.

Lab 4

Built in Functions in SQL, Group by clause, Having clause

Group By clause:

The SQL GROUP BY clause is used in collaboration with the SELECT statement to arrange identical data

into groups. The GROUP BY clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.

Syntax:

The basic syntax of GROUP BY clause is given below. The GROUP BY clause must follow the conditions in the WHERE clause and must precede the ORDER BY clause if one is used.

```
SELECT column1, column2
FROM table_name
WHERE [ conditions ]
GROUP BY column1, column2
ORDER BY column1, column2
```

Example:

```
SELECT NAME, SUM(SALARY) FROM CUSTOMERS
GROUP BY NAME;
```

ORDER BY Clause

The SQL ORDER BY clause is used to sort the data in ascending or descending order, based on one or

more columns. Some database sorts query results in ascending order by default.

Syntax:

```
SELECT column-list
FROM table_name
[WHERE condition]
[ORDER BY column1, column2, .. columnN] [ASC | DESC];
```

You can use more than one column in the ORDER BY clause. Make sure whatever column you are using to sort, that column should be in column-list.

Example:

SELECT * FROM CUSTOMERS

ORDER BY NAME DESC;

Note: By default the records are ordered in ascending order.

TOP Clause

The SQL TOP clause is used to fetch a TOP N number or X percent records from a table.Note: All the databases do not support TOP clause. For example MySQL supports LIMIT clause to fetch limited number of records and Oracle uses ROWNUM to fetch limited number of records.

Syntax:

SELECT TOP number|percent column_name(s)

FROM table_name

WHERE [condition]

Example:

SELECT TOP 3 * FROM CUSTOMERS;

HAVING CLAUSE

The HAVING clause enables you to specify conditions that filter which group results appear in the final results. The WHERE clause places conditions on the selected columns, whereas the HAVING clause places conditions on groups created by the GROUP BY clause.

The HAVING clause must follow the GROUP BY clause in a query and must also precedes the ORDER BY clause if used. The following is the syntax of the SELECT statement, including the HAVING clause:

Syntax:

SELECT column1, column2

FROM table1, table2

WHERE [conditions]

GROUP BY column1, column2

HAVING [conditions]

ORDER BY column1, column2

Example: display record for which similar age count would be more than or equal to 2:

```
SELECT *
FROM CUSTOMERS
GROUP BY age
HAVING COUNT(age) >= 2;
```

Built-in functions

SQL has many built-in functions for performing processing on string or numeric data. Following is the list of all useful SQL built-in functions:

- SQL COUNT Function - The SQL COUNT aggregate function is used to count the number of rows in a database table.
- SQL MAX Function - The SQL MAX aggregate function allows us to select the highest (maximum) value for a certain column.
- SQL MIN Function - The SQL MIN aggregate function allows us to select the lowest (minimum) value for a certain column.
- SQL AVG Function - The SQL AVG aggregate function selects the average value for certain table column.
- SQL SUM Function - The SQL SUM aggregate function allows selecting the total for a numeric column.
- SQL SQRT Functions - This is used to generate a square root of a given number.
- SQL RAND Function - This is used to generate a random number using SQL command.
- SQL CONCAT Function - This is used to concatenate any string inside any SQL command.
- SQL Numeric Functions - Complete list of SQL functions required to manipulate numbers in SQL.
- SQL String Functions - Complete list of SQL functions required to manipulate strings in SQL.

SQL COUNT Function

SQL COUNT function is the simplest function and very useful in counting the number of records , which are expected to be returned by a SELECT statement.

Example:

```
SQL>SELECT COUNT(*) FROM employee_tbl ;
```

Similarly, if you want to count the number of records for Nepal, then it can be done as follows:

```
SQL>SELECT COUNT(*) FROM employee_tbl WHERE name= ‘Nepal’;
```

SQL MAX Function

SQL MAX function is used to find out the record with maximum value among a record set.
Example:

```
SQL> SELECT * FROM employee_tbl;
+------+------+------------+--------------------+
| id   | name | work_date  | daily_typing_pages |
+------+------+------------+--------------------+
|    1 | John | 2007-01-24 |                250 |
|    2 | Ram  | 2007-05-27 |                220 |
|    3 | Jack | 2007-05-06 |                170 |
|    3 | Jack | 2007-04-06 |                100 |
|    4 | Jill | 2007-04-06 |                220 |
|    5 | Zara | 2007-06-06 |                300 |
|    5 | Zara | 2007-02-06 |                350 |
```

output:

```
SQL> SELECT id, name, MAX(daily_typing_pages)
    -> FROM employee_tbl GROUP BY name;
+------+------+-------------------------+
| id   | name | MAX(daily_typing_pages) |
+------+------+-------------------------+
|    3 | Jack |                     170 |
|    4 | Jill |                     220 |
```

```
|    1 | John |                     250 |
|    2 | Ram  |                     220 |
|    5 | Zara |                     350 |
+------+------+-------------------------+
5 rows in set (0.00 sec)
```

MIN Function

SQL MIN function is used to find out the record with minimum value among a record set.

```
SQL> SELECT id, name, work_date, MIN(daily_typing_pages)
    -> FROM employee_tbl GROUP BY name;
+------+------+-------------------------+
| id   | name | MIN(daily_typing_pages) |
+------+------+-------------------------+
|    3 | Jack |                     100 |
|    4 | Jill |                     220 |
|    1 | John |                     250 |
```

AVG Function

SQL AVG function is used to find out the average of a field in various records.

```
SQL> SELECT AVG(daily_typing_pages)
    -> FROM employee_tbl;
+-------------------------+
| AVG(daily_typing_pages) |
+-------------------------+
|                230.0000 |
+-------------------------+
1 row in set (0.03 sec)
```

SUM Function

SQL SUM function is used to find out the sum of a field in various records.

```
SQL> SELECT SUM(daily_typing_pages)
    -> FROM employee_tbl;
+-------------------------+
| SUM(daily_typing_pages) |
+-------------------------+
|                    1610 |
+-------------------------+
1 row in set (0.00 sec)
```

SQRT Function

SQL SQRT function is used to find out the square root of any number. You can Use SELECT statement to find out square root of any number.

```
SQL>  select SQRT(16);
+----------+
| SQRT(16) |
+----------+
| 4.000000 |
+----------+
1 row in set (0.00 sec)
```

ABS(X)

The ABS() function returns the absolute value of X.

```
SQL> SELECT ABS(-2);
+-------------------------
| ABS(2)
+-------------------------
| 2
+-------------------------
1 row in set (0.00 sec)
```

SQRT(X)

This function returns the non-negative square root of X

```
SQL>SELECT SQRT(49);
+-------------------------
| SQRT(49)
+-------------------------
| 7
+-------------------------
1 row in set (0.00 sec)
```

Lab Task:

1. Sort the employee records in descending order.
2. Sort name and publication name in ascending order.
3. Display top three records from teachers relation.
4. Display the sum of salaries of all employees.
5. Display the minimum salary of employee.
6. Display the average price of book written by same author.
7. Display publication name and number of books published by it from book list relation publication wise.
8. Display the bid and bname of books whose price is greater than average prices of book.
9. Find the bid , bname and author in ascending order where author name is started by “s”.
10. Find the teachers name and book taken by him. The teacher’s salary who takes the book should be the maximum salary.
11. Find the authors name who have written more than one book.

LAB: 5

Introduction to Joins & Creating Views

SQL Joins

The SQL Joins clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each.

SQL Join Types:

There are different types of joins available in SQL:

- INNER JOIN: returns rows when there is a match in both tables.
- LEFT JOIN: returns all rows from the left table, even if there are no matches in the right table.
- RIGHT JOIN: returns all rows from the right table, even if there are no matches in the left table.
- FULL JOIN: returns rows when there is a match in one of the tables.
- SELF JOIN: is used to join a table to itself as if the table were two tables, temporarily renaming at least one table in the SQL statement.
- CARTESIAN JOIN: returns the Cartesian product of the sets of records from the two or more joined tables.

INNER JOIN

The most frequently used and important of the joins is the INNER JOIN. They are also referred to as an EQUIJOIN.The INNER JOIN creates a new result table by combining column values of two tables (table1 and table2) based upon the join-predicate. The query compares each row of table1 with each row of table2 to find all pairs of rows which satisfy the join-predicate. When the join-predicate is satisfied, column values for each matched pair of rows of A and B are combined into a result row.

Syntax:

SELECT *column_name(s)*
FROM *table1*
INNER JOIN *table2*
ON *table1.column_name=table2.column_name*;

OR

SELECT *column_name(s)*
FROM *table1*

JOIN *table2*
ON *table1.column_name=table2.column_name*;

LEFT JOIN

The SQL LEFT JOIN returns all rows from the left table, even if there are no matches in the right table. This means that if the ON clause matches 0 (zero) records in right table, the join will still return a row in the result, but with NULL in each column from right table.This means that a left join returns all the values from the left table, plus matched values from the right table or NULL in case of no matching join predicate.

Syntax:

SELECT *column_name(s)*
FROM *table1*
LEFT JOIN *table2*
ON *table1.column_name=table2.column_name*;

or:

SELECT *column_name(s)*
FROM *table1*
LEFT OUTER JOIN *table2*
ON *table1.column_name=table2.column_name*;

RIGHT JOIN

The SQL RIGHT JOIN returns all rows from the right table, even if there are no matches in the left table. This means that if the ON clause matches 0 (zero) records in left table, the join will still return a row in the result , but with NULL in each column from left table.This means that a right join returns all the values from the right table, plus matched values from the left table or NULL in case of no matching join predicate.

Syntax:

SELECT *column_name(s)*
FROM *table1*
RIGHT JOIN *table2*
ON *table1.column_name=table2.column_name*;

or:

SELECT *column_name(s)*
FROM *table1*
RIGHT OUTER JOIN *table2*
ON *table1.column_name=table2.column_name*;

FULL JOIN

The SQL FULL JOIN combines the results of both left and right outer joins.

The joined table will contain all records from both tables, and fill in NULLs for missing matches on either side.

Syntax:

SELECT *column_name(s)*
FROM *table1*
FULL OUTER JOIN *table2*
ON *table1.column_name=table2.column_name*;

SELF JOIN

The SQL SELF JOIN is used to join a table to itself as if the table were two tables, temporarily renaming at least one table in the SQL statement.

Syntax:

SELECT a.column_name, b.column_name...

FROM table1 a, table1 b

WHERE a.common_filed = b.common_field;

SQL UNION Operator

The UNION operator is used to combine the result-set of two or more SELECT statements.

Notice that each SELECT statement within the UNION must have the same number of columns. The columns must also have similar data types. Also, the columns in each SELECT statement must be in the same order.

Syntax

SELECT *column_name(s)* FROM *table1*
UNION
SELECT *column_name(s)* FROM *table2*;

Note: The UNION operator selects only distinct values by default. To allow duplicate values, use the ALL keyword with UNION.

UNION ALL Syntax

SELECT *column_name(s)* FROM *table1*
UNION ALL
SELECT *column_name(s)* FROM *table2*;

PS: The column names in the result-set of a UNION are usually equal to the column names in the first SELECT statement in the UNION.

There are two other clauses (i.e., operators), which are very similar to UNION clause:

 SQL INTERSECT Clause: is used to combine two SELECT statements, but returns rows only from the first

SELECT statement that are identical to a row in the second SELECT statement.

 SQL EXCEPT Clause : combines two SELECT statements and returns rows from the first SELECT statement that are not returned by the second SELECT statement.

INTERSECT Clause

The SQL INTERSECT clause/operator is used to combine two SELECT statements, but returns rows only from the first SELECT statement that are identical to a row in the second SELECT statement. This means INTERSECT returns only common rows returned by the two SELECT statements.

Just as with the UNION operator, the same rules apply when using the INTERSECT operator. MySQL does not

support INTERSECT operator

Syntax:

SELECT column1 [, column2]

FROM table1 [, table2]

[WHERE condition]

INTERSECT

SELECT column1 [, column2]

FROM table1 [, table2]

[WHERE condition]

Here given condition could be any given expression based on your requirement.

EXCEPT Clause

The SQL EXCEPT clause/operator is used to combine two SELECT statements and returns rows from the first SELECT statement that are not returned by the second SELECT statement. This means EXCEPT returns only rows, which are not available in second SELECT statement.

Just as with the UNION operator, the same rules apply when using the EXCEPT operator. MySQL does not support EXCEPT operator.

Syntax:

SELECT column1 [, column2]
FROM table1 [, table2]
[WHERE condition]
EXCEPT
SELECT column1 [, column2]
FROM table1 [, table2]s
[WHERE condition]

Here given condition could be any given expression based on your requirement.

SQL - Using Views

Aview is nothing more than a SQL statement that is stored in the database with an associated name. A

view is actually a composition of a table in the form of a predefined SQL query.

A view can contain all rows of a table or select rows from a table. A view can be created from one or many tables which depends on the written SQL query to create a view.

Views, which are kind of virtual tables, allow users to do the following:

- Structure data in a way that users or classes of users find natural or intuitive.
- Restrict access to the data such that a user can see and (sometimes) modify exactly what they need and no more.
- Summarize data from various tables which can be used to generate reports.

Creating Views:

Database views are created using the CREATE VIEW statement. Views can be created from a single table, multiple tables, or another view.

To create a view, a user must have the appropriate system privilege according to the specific implementation.

Syntax:

CREATE VIEW view_name AS

SELECT column1, column2.....

FROM table_name

WHERE [condition];

You can include multiple tables in your SELECT statement in very similar way as you use them in normal SQL

SELECT query.

The WITH CHECK OPTION:

The WITH CHECK OPTION is a CREATE VIEW statement option. The purpose of the WITH CHECK OPTION is to ensure that all UPDATE and INSERTs satisfy the condition(s) in the view definition.

If they do not satisfy the condition(s), the UPDATE or INSERT returns an error.

Example:
CREATE VIEW CUSTOMERS_VIEW AS
SELECT name, age
FROM CUSTOMERS

WHERE age IS NOT NULL
WITH CHECK OPTION;
The WITH CHECK OPTION in this case should deny the entry of any NULL values in the view's AGE column, because the view is defined by data that does not have a NULL value in the AGE column.
Updating a View:

A view can be updated under certain conditions:

- The SELECT clause may not contain the keyword DISTINCT.
- The SELECT clause may not contain summary functions.
- The SELECT clause may not contain set functions.
- The SELECT clause may not contain set operators.
- The SELECT clause may not contain an ORDER BY clause.
- The FROM clause may not contain multiple tables.
- The WHERE clause may not contain subqueries.
- The query may not contain GROUP BY or HAVING.
- Calculated columns may not be updated.
- All NOT NULL columns from the base table must be included in the view in order for the INSERT query to function.

So if a view satisfies all the above mentioned rules then you can update a view.

Following is an example to update the age of Ramesh:

SQL > UPDATE CUSTOMERS_VIEW

SET AGE = 35

WHERE name='Ramesh';

This would ultimately update the base table CUSTOMERS and same would reflect in the view itself. Now, try to query base table, and SELECT statement would produce the following result:

Inserting Rows into a View:

Rows of data can be inserted into a view. The same rules that apply to the UPDATE command also apply to the INSERT command.

Here, we can not insert rows in CUSTOMERS_VIEW because we have not included all the NOT NULL columns in this view, otherwise you can insert rows in a view in similar way as you insert them in a table.

Deleting Rows into a View:

Rows of data can be deleted from a view. The same rules that apply to the UPDATE and INSERT commands apply to the DELETE command.

Following is an example to delete a record having AGE= 22.

SQL > DELETE FROM CUSTOMERS_VIEW

WHERE age = 22;

This would ultimately delete a row from the base table CUSTOMERS and same would reflect in the view itself.

Dropping Views:

Syntax:

DROP VIEW view_name;

Following is an example to drop CUSTOMERS_VIEW from CUSTOMERS table:

DROP VIEW CUSTOMERS_VIEW;

Lab Task:

1. Perform join operation on teacher and employee table and display the Ename , Faculty and salary.
2. Perform left join on table book list and book table.
3. Perform right join on booklist and book table.
4. Perform Full join on student and issues table.
5. Display those employees name and salary whose name starts with ‘s’ and whose name consist ‘ni’ as sub string.
6. Display name of the employee who is also a teacher.
7. Display all employees name except the name who are teachers.
8. Create a view Employee-view which consist of eid, ename , salary as attributes.
9. Insert an new record in recently created view. And also display the contents of primary table.
10. Delete the information from view where salary are less than 5000.

LAB:6

Stored Procedures &Inbuilt Functions

Overview

A stored procedure is nothing more than prepared SQL code that you save so you can reuse the code over and over again. So if you think about a query that you write over and over again, instead of having to write that query each time you would save it as a stored procedure and then just call the stored procedure to execute the SQL code that you saved as part of the stored procedure.

In addition to running the same SQL code over and over again you also have the ability to pass parameters to the stored procedure, so depending on what the need is the stored procedure can act accordingly based on the parameter values that were passed.

How to create a Procedure?

Syntax:

```
CREATE PROCEDURE procedure_name
AS
Sql_expression
GO
```

Example:

```
create procedure show1
as
select * from example
go
```

How to run the procedure?

Exec procedure_name

OR

Procedure_name

When creating a stored procedure you can either use CREATE PROCEDURE or CREATE PROC. After the stored procedure name you need to use the keyword "AS" and then the rest is just the regular SQL code that you would normally execute.

One thing to note is that you cannot use the keyword "GO" in the stored procedure. Once the SQL Server compiler sees "GO" it assumes it is the end of the batch.

Also, you cannot change database context within the stored procedure such as using "USE dbName" the reason for this is because this would be a separate batch and a stored procedure is a collection of only one batch of statements.

How to create a SQL Server stored procedure with parameters:

The real power of stored procedures is the ability to pass parameters and have the stored procedure handle the differing requests that are made. In this topic we will look at passing parameter values to a stored procedure.

Explanation

Just like you have the ability to use parameters with your SQL code you can also setup your stored procedures to accept one or more parameter values.

One Parameter

In this example we will query the Person.Address table from the AdventureWorks database, but instead of getting back all records we will limit it to just a particular city. This example assumes there will be an exact match on the City value that is passed.

```
CREATE PROCEDURE uspGetAddress @City nvarchar(30)
AS
SELECT *
FROM AdventureWorks.Person.Address
WHERE City = @City
GO
```

To call this stored procedure we would execute it as follows:

```
EXEC uspGetAddress @City = 'New York'
```

We can also do the same thing, but allow the users to give us a starting point to search the data. Here we can change the "=" to a LIKE and use the "%" wildcard.

```
CREATE PROCEDURE uspGetAddress @City nvarchar(30)
AS
SELECT *
FROM AdventureWorks.Person.Address
WHERE City LIKE @City + '%'
GO
```

In both of the proceeding examples it assumes that a parameter value will always be passed. If you try to execute the procedure without passing a parameter value you will get an error message such as the following:

Msg 201, Level 16, State 4, Procedure uspGetAddress, Line 0

Procedure or function 'uspGetAddress' expects parameter '@City', which was not supplied.

Default Parameter Values

In most cases it is always a good practice to pass in all parameter values, but sometimes it is not possible. So in this example we use the NULL option to allow you to not pass in a parameter value. If we create and run this stored procedure as is it will not return any data, because it is looking for any City values that equal NULL.

```
CREATE PROCEDURE uspGetAddress @City nvarchar(30) = NULL
```

```
AS
SELECT *
FROM AdventureWorks.Person.Address
WHERE City = @City
GO
```

We could change this stored procedure and use the ISNULL function to get around this. So if a value is passed it will use the value to narrow the result set and if a value is not passed it will return all records. (Note: if the City column has NULL values this will not include these values. You will have to add additional logic for City IS NULL)

```
CREATE PROCEDURE uspGetAddress @City nvarchar(30) = NULL
AS
SELECT *
FROM AdventureWorks.Person.Address
WHERE City = ISNULL(@City,City)
GO
```

Multiple Parameters

Setting up multiple parameters is very easy to do. You just need to list each parameter and the data type separated by a comma as shown below.

```
CREATE PROCEDURE uspGetAddress @City nvarchar(30) = NULL, @AddressLine1 nvarchar(60) =
NULL
AS
SELECT *
FROM AdventureWorks.Person.Address
WHERE City = ISNULL(@City,City)
AND AddressLine1 LIKE '%' + ISNULL(@AddressLine1 ,AddressLine1) + '%'
GO
```

To execute this you could do any of the following:

```
EXEC uspGetAddress @City = 'Calgary'
--or
EXEC uspGetAddress @City = 'Calgary', @AddressLine1 = 'A'
--or
EXEC uspGetAddress @AddressLine1 = 'Acardia'
-- etc...
```

Example:

```
CREATE PROCEDURE show4 @nam varchar(30),@idi int
AS
SELECT *
FROM example
WHERE name like @nam +'%' or id=@idi
GO
drop procedure show4
 show4 @nam ='s',@idi=12
```

Deleting a SQL Server stored procedure

Dropping Single Stored Procedure

To drop a single stored procedure you use the DROP PROCEDURE or DROP PROC command as follows.

```
DROP PROCEDURE uspGetAddress
GO
-- or
DROP PROC uspGetAddress
GO
-- or
DROP PROC dbo.uspGetAddress -- also specify the schema
```

Dropping Multiple Stored Procedures

To drop multiple stored procedures with one command you specify each procedure separated by a comma as shown below.

```
DROP PROCEDURE uspGetAddress, uspInsertAddress, uspDeleteAddress
GO
-- or
DROP PROC uspGetAddress, uspInsertAddress, uspDeleteAddress
GO
```

Naming conventions for SQL Server stored Procedures

SQL Server uses object names and schema names to find a particular object that it needs to work with. This could be a table, stored procedure, function ,etc...

It is a good practice to come up with a standard naming convention for you objects including stored procedures.

Do not use sp_ as a prefix

One of the things you do not want to use as a standard is "sp_". This is a standard naming convention that is used in the master database. If you do not specify the database where the object is, SQL Server will first search the master database to see if the object exists there and then it will search the user database. So avoid using this as a naming convention.

Standardize on a Prefix

It is a good idea to come up with a standard prefix to use for your stored procedures. As mentioned above do not use "sp_", so here are some other options.

- usp_
- sp
- usp
- etc...

To be honest it does not really matter what you use. SQL Server will figure out that it is a stored procedure, but it is helpful to differentiate the objects, so it is easier to manage.

So a few examples could be:

- spInsertPerson
- uspInsertPerson
- usp_InsertPerson
- InsertPerson

Again this is totally up to you, but some standard is better than none.

Naming Stored Procedure Action

Based on the actions that you may take with a stored procedure, you may use:

- Insert
- Delete
- Update
- Select
- Get
- Validate
- etc...

So here are a few examples:

- uspInsertPerson
- uspGetPerson
- spValidatePerson
- SelectPerson
- etc...

Another option is to put the object name first and the action second, this way all of the stored procedures for an object will be together.

- uspPersonInsert
- uspPersonDelete
- uspPersonGet
- etc...

Again, this does not really matter what action words that you use, but this will be helpful to classify the behavior characteristics.

Naming Stored Procedure Object

Keep the names simple, but meaningful. As your database grows and you add more and more objects you will be glad that you created some standards.

So some of these may be:

- uspInsertPerson - insert a new person record
- uspGetAccountBalance - get the balance of an account
- uspGetOrderHistory - return list of orders

Using comments in a SQL Server stored procedure

SQL Server offers two types of comments in a stored procedure; line comments and block comments. The following examples show you how to add comments using both techniques. Comments are displayed in green in a SQL Server query window.

Line Comments

To create line comments you just use two dashes "--" in front of the code you want to comment. You can comment out one or multiple lines with this technique.

In this example the entire line is commented out.

```
-- this procedure gets a list of addresses based
-- on the city value that is passed
CREATE PROCEDURE uspGetAddress @City nvarchar(30)
AS
SELECT *
FROM AdventureWorks.Person.Address
WHERE City = @City
GO
```

This next example shows you how to put the comment on the same line.

```
-- this procedure gets a list of addresses based on the city value that is passed
CREATE PROCEDURE uspGetAddress @City nvarchar(30)
AS
SELECT *
FROM AdventureWorks.Person.Address
WHERE City = @City -- the @City parameter value will narrow the search criteria
GO
```

Block Comments

To create block comments the block is started with "/*" and ends with "*/". Anything within that block will be a comment section.

```
/*
-this procedure gets a list of addresses based
 on the city value that is passed
-this procedure is used by the HR system
*/
CREATE PROCEDURE uspGetAddress @City nvarchar(30)
AS
SELECT *
FROM AdventureWorks.Person.Address
WHERE City = @City
GO
```

Combining Line and Block Comments

You can also use both types of comments within a stored procedure.

```
/*
-this procedure gets a list of addresses based
 on the city value that is passed
-this procedure is used by the HR system
*/
CREATE PROCEDURE uspGetAddress @City nvarchar(30)
AS
SELECT *
FROM AdventureWorks.Person.Address
WHERE City = @City -- the @City parameter value will narrow the search criteria
GO
```

Returning stored procedure parameter values to a calling stored procedure

In a previous topic we discussed how to pass parameters into a stored procedure, but another option is to pass parameter values back out from a stored procedure. One option for this may be that you call another stored procedure that does not return any data, but returns parameter values to be used by the calling stored procedure.

Explanation

Setting up output paramters for a stored procedure is basically the same as setting up input parameters, the only difference is that you use the OUTPUT clause after the parameter name to specify that it should return a value. The output clause can be specified by either using the keyword "OUTPUT" or just "OUT".

Simple Output

```
CREATE PROCEDURE uspGetAddressCount @City nvarchar(30), @AddressCount int OUTPUT
AS
SELECT @AddressCount = count(*)
FROM AdventureWorks.Person.Address
WHERE City = @City
```

Or it can be done this way:

```
CREATE PROCEDURE uspGetAddressCount @City nvarchar(30), @AddressCount int OUT
AS
SELECT @AddressCount = count(*)
FROM AdventureWorks.Person.Address
WHERE City = @City
```

To call this stored procedure we would execute it as follows. First we are going to declare a variable, execute the stored procedure and then select the returned valued.

```
DECLARE @AddressCount int
EXEC uspGetAddressCount @City = 'Calgary', @AddressCount = @AddressCount OUTPUT
SELECT @AddressCount
```

This can also be done as follows, where the stored procedure parameter names are not passed.

```
DECLARE @AddressCount int
EXEC uspGetAddressCount 'Calgary', @AddressCount OUTPUT
SELECT @AddressCount
```

Modifying an existing SQL Server stored procedure

Modifying or ALTERing a stored procedure is pretty simple. Once a stored procedure has been created it is stored within one of the system tables in the database that is was created in. When you modify a stored procedure the entry that was originally made in the system table is replaced by this new code. Also, SQL Server will recompile the stored procedure the next time it is run, so your users are using the new logic. The command to modify an existing stored procedure is ALTER PROCEDURE or ALTER PROC.

Modifying an Existing Stored Procedure

Let's say we have the following existing stored procedure: This allows us to do an exact match on the City.

```
CREATE PROCEDURE uspGetAddress @City nvarchar(30)
AS
SELECT *
FROM AdventureWorks.Person.Address
WHERE City = @City
GO
```

Let's say we want to change this to do a LIKE instead of an equals.

To change the stored procedure and save the updated code you would use the ALTER PROCEDURE command as follows.

```
ALTER PROCEDURE uspGetAddress @City nvarchar(30)
AS
SELECT *
FROM AdventureWorks.Person.Address
WHERE City LIKE @City + '%'
GO
```

Now the next time that the stored procedure is called by an end user it will use this new logic.

Sql date time functions:

- DateTime Function in SQL Server

  - GETDATE()
  - DATEPART()
  - DATEDIFF()
  - DATENAME()
  - DAY()
  - MONTH()
  - YEAR()

GETDATE() is very common used method which returns exact date time from the system. It does not accept any parameter. Just call it like simple function.

Example:

```
SELECT getdate()
```

DATEADD()

`DATEADD()` is used to add or subtract datetime. Its return a new datetime based on the added or subtracted interval.

```
General Syntax

DATEADD(datepart, number, date)
```

`datepart` is the parameter that specifies on which part of the date to return a new value. `Number` parameter is used to increment datepart.

Example:

```
SELECT getdate();

SELECT DATEADD(day, 5,getdate() ) AS NewTime
```

Output:

2015-08-12 09:18:18.080

2015-08-17 09:18:18.080

DATEPART()

`DATEPART()` is used when we need a part of date or time from a datetime variable. We can use `DATEPART()` method only with select command.

Syntax

```
DATEPART(datepart, date)

Example:
SELECT DATEPART(year, GETDATE()) AS 'Year'
SELECT DATEPART(hour, GETDATE()) AS 'Hour'
```

DATEDIFF()

`DATEDIFF()` is very common function to find out the difference between two `DateTime` elements.

Syntax

```
DATEDIFF(datepart, startdate, enddate)

Example:
SELECT DATEDIFF(day, @Date1, @Date2) AS DifferenceOfDay
```

DATENAME()

`DATENAME()` is very common and most useful function to find out the date name from the datetime value.

```
-- Get Today
SELECT DATENAME(dw, getdate()) AS 'Today Is'
-- Get Mont name
SELECT DATENAME(month, getdate()) AS 'Month'
```

DAY()

`DAY()` is used to get the day from any date time object.

Example:

```
SELECT DAY(getdate()) AS 'DAY'
```

Output :

```
DAY
-----------
12
```

MONTH()

```
SELECT MONTH(getdate()) AS 'Month'
```

YEAR()

```
SELECT YEAR(getdate()) AS 'Year'
```

String Functions

- ASCII()
- CHAR()
- LEFT()
- RIGHT()
- LTRIM()
- RTRIM()
- REPLACE()
- QUOTNAME()
- REVERSE()
- CHARINDEX()
- PATINDEX()
- LEN()
- STUFF()
- SUBSTRING()
- LOWER/UPPER()

ASCII()

Returns the ASCII code value of the leftmost character of a character expression.

Syntax

```
ASCII ( character_expression )
```

CHAR()

Converts an **int** ASCII code to a character.

Syntax

```
CHAR ( integer_expression )
```

Arguments: integer_expression: Is an integer from 0 through 255. NULL is returned if the integer expression is not in this range.

LEFT()

Returns the left most characters of a string.

Syntax

```
LEFT(string, length)
```

string
Specifies the string from which to obtain the left-most characters.

length
Specifies the number of characters to obtain.

```
Example:
SELECT LEFT('kathmandu',5)
 Output:
Kathm
```

RIGHT()

Returns the right most characters of a string.

Syntax

```
RIGHT(string, length)
```

string
Specifies the string from which to obtain the left-most characters.

length
Specifies the number of characters to obtain.

LTRIM()

Returns a character expression after it removes leading blanks.

Example :

```
SELECT LTRIM('   Md. Ashok')
```

Output :

Md. Ashok

RTRIM()

Returns a character string after truncating all trailing blanks.

Example :

```
SELECT RTRIM('Md. barsha    ')
```
Output:

Md. Barsha

REVERSE()

Returns a character expression in reverse order.

Example :

```
SELECT REVERSE('nepal')
```

CHARINDEX

`CharIndex` returns the first occurance of a string or characters within another string. The Format of CharIndex is given Below:

`CHARINDEX ( expression1 , expression2 [ , start_location ] )`
Here *expression1* is the string of characters to be found within *expression2*. So if you want to search *ij* within the word *Abhijit*, we will use *ij* as *expression1* and *Abhijit* as *expression2*. start_location is an optional integer argument which identifies the position from where the string will be searched. Now let us look into some examples :

Hide Copy Code

```
SELECT CHARINDEX('SQL', 'Microsoft SQL Server')
```

OUTPUT:

```
11
```

So it will start from 1 and go on searching until it finds the total string element searched, and returns its first position. The Result will be 0 if the searched string is not found.

We can also mention the Start_Location of the string to be searched.

EXAMPLE:

```
SELECT CHARINDEX('SQL', 'Microsoft SQL server has a great SQL Engine',12)
```
So in the above example we can have the Output as 34 as we specified the StartLocation as 12, which is greater than initial SQL position(11).

PATINDEX

As a contrast `PatIndex` is used to search a pattern within an expression. The Difference between `CharIndex` and `PatIndex` is the later allows WildCard Characters.

```
PATINDEX ('%pattern%' , expression)
```

Here the first argument takes a pattern with wildcard characters like `'%' (meaning any string) or '_' (meaning any character).`

For Example
PATINDEX('%BC%','ABCD')

Output:

```
2
```

LEN

Len is a function which returns the length of a string. This is the most common and simplest function that everyone use. Len Function excludes trailing blank spaces.

```
SELECT LEN('ABHISHEK IS WRITING THIS')
```

This will output 24, it is same when we write `LEN('ABHISHEK IS WRITING THIS ')` as LEN doesnt take trailing spaces in count.

SUBSTRING

`Substring` returns the part of the string from a given characterexpression. The general syntax of Substring is as follows :

```
SUBSTRING(expression, start, length)
```

Here the function gets the string from start to length. Let us take an example below:

```
SELECT OUT = SUBSTRING('abcdefgh', 2, 3)
```

The output will be "bcd".
Note : substring also works on ntext, VARCHAR, CHAR etc.

LOWER / UPPER

Another simple but handy function is Lower / UPPER. The will just change case of a string expression. For Example,

```
SELECT UPPER('this is Lower TEXT')
```