ICIT INDIA COMPUTER® INSTITUTE

AN ISO 9001 :2015 CERTIFIED COMPANY

A DIVISION OF ICIT EDUCATION PVT. LTD.

भारत सरकारमान्य संस्था

SQL REFERENCE BOOK

RELATIONAL DATABASE MANAGEMENT SYSTEM

RDBMS PROGRAMMING

SQL

ICIT INDIA COMPUTER® INSTITUTE
AN ISO 9001:2015 CERTIFIED COMPANY
A DIVISION OF ICIT EDUCATION PVT. LTD.
भारत सरकारमान्य संस्था

Message

Mr. Santosh Khadtare
(Chief Managing Director)
(ICIT Education Pvt. Ltd.)

Message from Managing Director Desk

Dear Students and Parents,
Welcome with warmest greetings and thank you for trusting ICIT Computer Institute. We are optimistic that the information provided in this institute can assist you in your academic or career plans. We are honored to have your interest in us.
I take this opportunity to thank our valued students, whose continued patronage and confidence in our courses inspires us to extend the best of services and enables us to provide value for their money.
CORE PURPOSE:
"To bring about a meaningful transformation in the Lives of Individuals through an Array of Educational Program thereby empowering them to envision a great future"
CORE VALUES:

- Providing customer value for money.
- Continuous Improvement as a way of Life
- Empowering student with new ideas
- Honesty and Integrity
- Affordable Quality & Reliability

Today's world demands for computer education. It is my vision and mission to impart International Quality IT Education at best Indian Prices as to create World Class Computer professionals on the Indian Ground.
ICIT shares endeavor's to educate students to acquire a broader global mindset of the new upcoming IT World. The concept of industrial skills and experience put our students in a commanding position when they exit as graduates to begin their careers. We hope to create best IT Professionals for the current world.
I sincerely show gratitude towards my Academic Team who shows continuous efforts in brining improvements in our study materials, Assignments, Projects, Soft Skills, Audio & Video Trainings, E-Learning. The books have been created by keeping in mind the requirements of the SME's and corporates. The books have great assignments which will give prospectus to our students in the corporate world.

Thanking You,

With Best Regards,

ICIT INDIA COMPUTER® INSTITUTE
AN ISO 9001 :2015 CERTIFIED COMPANY
A DIVISION OF ICIT EDUCATION PVT. LTD.
भारत सरकारमान्य संस्था

NOTICE OF RIGHTS

TRADEMARK NOTICE

NOTICE LIABILITY

DISCLAIMER

Table of Contents

Chapter 1

Introduction of SQL

- What is SQL?
- Why SQL?
- History of SQL
- Commands in SQL
- Difference between DBMS and RDBMS
- What are field and Row?
- SQL Language Statements
- Application for SQL
- Advantage and Disadvantage of SQL

1.1 SQL

SQL is a standard language for storing, manipulating and retrieving data in databases.

1.1.1 What is SQL?

SQL stands for Structured Query Language
SQL lets you access and manipulate databases
SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987

1.1.2 What Can SQL do?

SQL can execute queries against a database
SQL can retrieve data from a database
SQL can insert records in a database
SQL can update records in a database
SQL can delete records from a database
SQL can create new databases
SQL can create new tables in a database
SQL can create stored procedures in a database
SQL can create views in a database
SQL can set permissions on tables, procedures, and views

1.1.3 What is Database?

A **database** is *an organized collection of data*.
Database handlers create database in such a way that only one set of software program provide access of data to all the users.
The **main purpose** of database is to operate large amount of information by storing, retrieving and managing.
There are many **dynamic websites** on the world wide web now a days which are handled through databases. For example, a model to checks the availability of rooms in a hotel. It is an example of dynamic website that uses database.
There are many **databases available** like MySQL, Sybase, Oracle, Mango DB, Informix, Postgre, SQL Server etc.

SQL or Structured Query Language is used to perform operation on the data stored in a database. SQL depends on relational algebra and tuple relational calculus.
A cylindrical structure is used to display the image of a database.

1.1.4 RDBMS

RDBMS stands for Relational Database Management System.
RDBMS is the basis for SQL, and for all modern database systems such as MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.
The data in RDBMS is stored in database objects called tables. A table is a collection of related data entries and it consists of columns and rows.

How it works

Data is represented in terms of tuples (rows) in RDBMS.

Relational database is most commonly used database. It contains number of tables and each table has its own primary key.

Due to a collection of organized set of tables, data can be accessed easily in RDBMS.

What is table

The RDBMS database uses tables to store data. A table is a collection of related data entries and contains rows and columns to store data.

A table is the simplest example of data storage in RDBMS.

Let's see the example of student table.

ID	Name	AGE	COURSE
1	Ajeet	24	B.Tech
2	aryan	20	C.A
3	Mahesh	21	BCA
4	Ratan	22	MCA
5	Vimal	26	BSC

What is field?

Field is a smaller entity of the table which contains specific information about every record in the table. In the above example, the field in the student table consist of id, name, age, course.

What is row or record?

A row of a table is also called record. It contains the specific information of each individual entry in the table. It is a horizontal entity in the table. For example: The above table contains 5 records.

Let's see one record/row in the table.

1	Ajeet	24	B.Tech

What is column?

A column is a vertical entity in the table which contains all information associated with a specific field in a table. For example: "name" is a column in the above table which contains all information about student's name.

Ajeet
Aryan
Mahesh
Ratan
Vimal

NULL Values

The NULL value of the table specifies that the field has been left blank during record creation. It is totally different from the value filled with zero or a field that contains space.

Data Integrity

There are the following categories of data integrity exist with each RDBMS:

Entity integrity: It specifies that there should be no duplicate rows in a table.

Domain integrity: It enforces valid entries for a given column by restricting the type, the format, or the range of values.

Referential integrity: It specifies that rows cannot be deleted, which are used by other records.

User-defined integrity: It enforces some specific business rules that are defined by users. These rules are different from entity, domain or referential integrity.

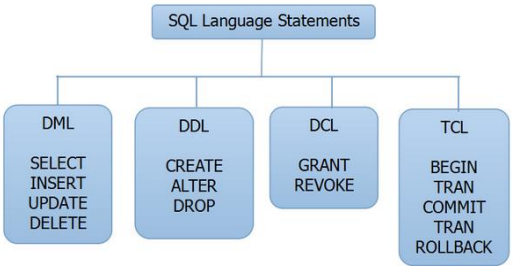
Difference between DBMS and RDBMS

No.	DBMS	RDBMS
1)	DBMS applications store **data as file**.	RDBMS applications store **data in a tabular form**.
2)	In DBMS, data is generally stored in either a hierarchical form or a navigational form.	In RDBMS, the tables have an identifier called primary key and the data values are stored in the form of tables.
3)	**Normalization is not** present in DBMS.	**Normalization is** present in RDBMS.
4)	DBMS does **not apply any security** with regards to data manipulation.	RDBMS **defines the integrity constraint** for the purpose of ACID (Atomocity, Consistency, Isolation and Durability) property.
5)	DBMS uses file system to store data, so there will be **no relation between the tables**.	in RDBMS, data values are stored in the form of tables, so a **relationship** between these data values will be stored in the form of a table as well.
6)	DBMS has to provide some uniform methods to access the stored information.	RDBMS system supports a tabular structure of the data and a relationship between them to access the stored information.
7)	DBMS **does not support distributed database**.	RDBMS **supports distributed database**.
8)	DBMS is meant to be for small organization and **deal with small data**. it supports **single user**.	RDBMS is designed to **handle large amount of data**. it supports **multiple users**.
9)	Examples of DBMS are file systems, **xml** etc.	Example of RDBMS are **mysql**, **postgre**, **sql server**, **oracle** etc.

1.2 DML, DDL, DCL and TCL Statements in SQL

The four main categories of SQL statements are as follows:

1. **DML (Data Manipulation Language)**
2. **DDL (Data Definition Language)**
3. **DCL (Data Control Language)**
4. **TCL (Transaction Control Language)**



DML (Data Manipulation Language)
DML statements affect records in a table. These are basic operations we perform on data such as selecting a few records from a table, inserting new records, deleting unnecessary records, and updating/modifying existing records.

DML statements include the following:
SELECT – select records from a table
INSERT – insert new records
UPDATE – update/Modify existing records
DELETE – delete existing records

DDL (Data Definition Language)
DDL statements are used to alter/modify a database or table structure and schema. These statements handle the design and storage of database objects.
CREATE – create a new Table, database, schema
ALTER – alter existing table, column description
DROP – delete existing objects from database

DCL (Data Control Language)
DCL statements control the level of access that users have on database objects.
GRANT – allows users to read/write on certain database objects
REVOKE – keeps users from read/write permission on database objects

TCL (Transaction Control Language)
TCL statements allow you to control and manage transactions to maintain the integrity of data within SQL statements.
BEGIN Transaction – opens a transaction
COMMIT Transaction – commits a transaction
ROLLBACK Transaction – ROLLBACK a transaction in case of any error

Applications of SQL (Structured Query Language)

Data Integration Scripts
The main application of SQL is to write data integration.

Analytical Queries
The data analysts use structured query language for setting and running analytical queries on a regular basis.

Retrieve Information
Another popular application of this language is to retrieve the subsets of information within a database for analytics applications and transaction processing. The most commonly used SQL elements are select, insert, update, add, delete, create, truncate and alter.

Other Important Applications
The SQL is used for modification of the index structures and database table. Additionally, the users can add, update and delete the rows of the data by using this language.

Advantages of SQL

There are numerous advantages of Structured Query Language and some of them are mentioned below:

No coding needed

It is very easy to manage the database systems without any need to write the substantial amount of code by using the standard SQL.

Well defined standards

Long established are used by the SQL databases that is being used by ISO and ANSI. There are no standards adhered by the non-SQL databases.

Portability

SQL can be used in the program in PCs, servers, laptops, and even some of the mobile phones.

Interactive Language

This domain language can be used for communicating with the databases and receive answers to the complex questions in seconds.

Multiple data views

With the help of SQL language, the users can make different views of database structure and databases for the different users.

Disadvantages of SQL

Along with some benefits, the Structured query language also has some certain disadvantages:

Difficult Interface

SQL has a complex interface that makes it difficult for some users to access it.

Partial Control

The programmers who use SQL doesn't have a full control over the database because of the hidden business rules.

Implementation

Some of the databases go to the proprietary extensions to standard SQL for ensuring the vendor lock-in.

Cost

The operating cost of some SQL versions makes it difficult for some programmers to access it.

Summary

- **In this chapter, you have learned about:**
- **We Studied what is SQL History of SQL**
- **What is Database why we use Database**
- **Different type of sql language commands**
- **Difference between DBMS and RDBMS**
- **Also, we studied Application, Advantage and Disadvantage for SQL**

MCQ's

1. When was the first version of Microsoft SQL Server released?
a) 1983
b) 1988
c) 1990
d) 1991

2. Which of the following companies originally worked together to create and market the first version of SQL Server?
a) Microsoft
b) Sybase
c) Ashton-Tate
d) All of the Mentioned

3. Which of the SQL Server RTM included native support for managing XML data?
a) 7.0
b) 6.5
c) 8
d) 9

4. Codename for SQL Server 2012 is _____
a) Kilimanjaro
b) Katmai
c) Denali
d) Hekaton

5. SQL Server 2005 has following features
a) Dynamic Management Views
b) FILESTREAM
c) Powerpivot
d) In-memory capability

6. _____ is a software application first launched with the Microsoft SQL Server 2005.
a) Enterprise Manager
b) Query Analyzer
c) Business Intelligence Development Studio
d) SQL Server Management Studio

7. Local DB was introduced in which of the following versions of SQL Server?
a) 2012
b) 2008
c) 2014
d) 2008 R2

8. _____ is free database software running free SQL Server technology.
a) SQL Server Express
b) SQL Server Workgroup
c) SQL Server Enterprise
d) SQL Server Web

9. _____is the first true GUI-based database server.
a) SQL Server 7.0
b) SQL Server 6.5
c) SQL Server 2005
d) SQL Server 2008

10. Which was the first version of SQL Server to introduce in-memory capability?
a) SQL Server 2012
b) SQL Server 2014
c) SQL Server 2005
d) SQL Server 2008

Fill in the blank

1. **SQL stands for----------.**
2. **SQL lets you ---------- and --------- databases.**
3. **RDBMS stands for ----------.**
4. **DDL statements are used to ---------- a database or table structure and schema.**
5. **A row of a table is also called ----------.**

Chapter 2

SQL Constraints

- SQL Server CONSTRAINTS
- NOT NULL constraint
- DEFAULT Constraint
- UNIQUE Constraint
- Creation of primary key
- Foreign key constraint
- Drop primary key constraint
- Creation of Table statements
- DROP and Check Constraint

2.1 SQL Server CONSTRAINTS

Constraints are the principles enforced on the information columns of a table.
This ensures the accuracy and dependability of the information within the info.
SQL Server Constraints may well be either on a column level or a table level.
The column level constraints are applied solely to 1 column, whereas the table level constraints are applied to the full table.

2.1.2 NOT NULL constraint

Ensures that a column cannot have NULL worth. NOT NULL constraint makes certain that a column doesn't hold a NULL value. Once we don't give worth for a specific column whereas inserting a record into a table, it takes NULL worth by default. By specifying a NULL constraint, we are able to take care that a specific column(s) cannot have NULL values.

Assignment No 1

```
CREATE TABLE STUDENT (
ROLL_NO INT NOT NULL,
STU_NAME VARCHAR (20) NOT NULL,
STU_AGE INT NOT NULL,
STU_ADDRESS VARCHAR (600),
PRIMARY KEY (ROLL_NO)
);
```

Output
Command(s) completed successfully.

DEFAULT Constraint

Provides a default worth for a column once none is specified. The DEFAULT constraint provides a default worth to a column once there's no worth provided whereas inserting a record into a table.

Assignment No 2

```
CREATE TABLE STUDENT (
ROLL_NO INT NOT NULL,
STU_NAME VARCHAR (20) NOT NULL,
STU_AGE INT NOT NULL,
EXAM_FEE INT DEFAULT 5000,
STU_ADDRESS VARCHAR (20),
PRIMARY KEY (ROLL_NO)
);
```

2.1.3 UNIQUE Constraint

Ensures that each one values in a column are totally different. UNIQUE Constraint enforces a column or set of columns to possess distinctive values. If a column contains a distinctive constraint, it means specific column cannot have duplicate values in a very table.

Assignment No 3

```
CREATE TABLE STUDENT (
ROLL_NO INT NOT NULL,
STU_NAME VARCHAR (20) NOT NULL UNIQUE,
STU_AGE INT NOT NULL,
STU_ADDRESS VARCHAR (20) UNIQUE,
PRIMARY KEY (ROLL_NO)
);
```

2.1.4 PRIMARY Key

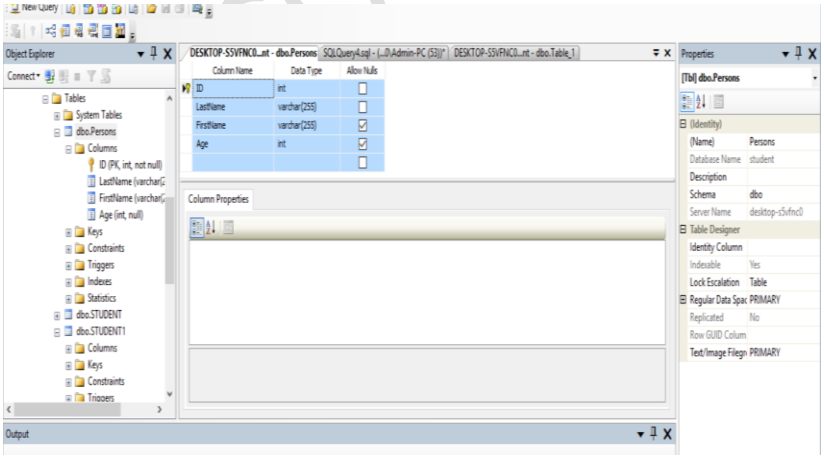
unambiguously identifies every row/record during an information table. Primary key unambiguously identifies every record in an exceeding table. It should have distinctive values and can't contain nulls. Within the below example the ROLL_NO field is marked as primary key, meaning the ROLL_NO field cannot have duplicate and null values.

Assignment No 4

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

Output

Command(s) completed successfully.



DROP a PRIMARY KEY Constraint
ALTER TABLE Persons
DROP CONSTRAINT PK_Person;

2.1.5 FOREIGN Key

A FOREIGN KEY is a key used to link two tables together.
A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table.
The table containing the foreign key is called the child table, and the table containing the candidate key is called the referenced or parent table.
The relationship between 2 tables matches the Primary Key in one of the tables with a Foreign Key in the second table.

Assignment No 5

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)
);
```

If the ORDERS table has already been created and the foreign key has not yet been set, the use the syntax for specifying a foreign key by altering a table.

```
ALTER TABLE ORDERS
  ADD FOREIGN KEY (Customer_ID) REFERENCES CUSTOMERS (ID);
```

2.2 DROP a FOREIGN KEY Constraint

To drop a FOREIGN KEY constraint, use the following SQL syntax.

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

check constraints

What is a check constraint in SQL Server?

A check constraint in SQL Server (Transact-SQL) allows you to specify a condition on each row in a table.
Note

A check constraint can NOT be defined on a SQL View.
The check constraint defined on a table must refer to only columns in that table. It cannot refer to columns in other tables.
A check constraint can NOT include a Subquery.
A check constraint can be defined in either a CREATE TABLE statement or a ALTER TABLE statement.

Using a CREATE TABLE statement
The syntax for creating a check constraint using a CREATE TABLE statement in SQL Server (Transact-SQL) is:

```
CREATE TABLE table_name
(
  column1 datatype [ NULL | NOT NULL ],
  column2 datatype [ NULL | NOT NULL ],

  ...

  CONSTRAINT constraint_name
    CHECK [ NOT FOR REPLICATION ] (column_name condition)

);
```

table_name
The name of the table that you wish to create with a check constraint.

constraint_name
The name to assign to the check constraint.

column_name
The column in the table that the check constraint applies to.

condition
The condition that must be met for the check constraint to succeed.

Assignment No 6
Example

```
CREATE TABLE employees
( employee_id INT NOT NULL,
  last_name VARCHAR(50) NOT NULL,
  first_name VARCHAR(50),
  salary MONEY,
  CONSTRAINT check_employee_id
    CHECK (employee_id BETWEEN 1 and 10000)
);
```

Let's take a look at another example.

```
CREATE TABLE employees
( employee_id INT NOT NULL,
  last_name VARCHAR(50) NOT NULL,
  first_name VARCHAR(50),
```

```
  salary MONEY,
  CONSTRAINT check_salary
    CHECK (salary > 0)
);
```

Using an ALTER TABLE statement

The syntax for creating a check constraint in an ALTER TABLE statement in SQL Server (Transact-SQL) is:

```
ALTER TABLE table_name
ADD CONSTRAINT constraint_name
  CHECK (column_name condition);
```

table_name
The name of the table that you wish to modify by adding a check constraint.
constraint_name
The name to assign to the check constraint.
column_name
The column in the table that the check constraint applies to.
condition
The condition that must be met for the check constraint to succeed.

Assignment No 7

Let's look at an example of how to use the ALTER TABLE statement to create a check constraint in SQL Server.

```
ALTER TABLE employees
ADD CONSTRAINT check_last_name
  CHECK (last_name IN ('Smith', 'Anderson', 'Jones'));
```

In this example, we've created a check constraint on the existing *employees* table called *check_last_name*. It ensures that the *last_name* field only contains the following values: Smith, Anderson, or Jones.

Drop a Check Constraint

The syntax for dropping a check constraint in SQL Server (Transact-SQL) is:

```
ALTER TABLE table_name
DROP CONSTRAINT constraint_name;
```

table_name
The name of the table that you wish to drop the check constraint.
constraint_name
The name of the check constraint to remove.

Assignment No 8

Example

Let's look at an example of how to drop a check constraint in SQL Server.

For example:

```
ALTER TABLE employees
DROP CONSTRAINT check_last_name;
```

In this SQL Server example, we are dropping a check constraint on the *employees* table called *check_last_name*.

Enable a Check Constraint

The syntax for enabling a check constraint in SQL Server (Transact-SQL) is:

```
ALTER TABLE table_name
WITH CHECK CHECK CONSTRAINT constraint_name;
```

table_name

The name of the table that you wish to enable the check constraint.

constraint_name

The name of the check constraint to enable.

Example

Let's look at an example of how to enable a check constraint in SQL Server.

For example:

```
ALTER TABLE employees
WITH CHECK CHECK CONSTRAINT check_salary;
```

In this example, we are enabling a check constraint on the *employees* table called *check_salary*.

Disable a Check Constraint

The syntax for disabling a check constraint in SQL Server (Transact-SQL) is:

```
ALTER TABLE table_name
NOCHECK CONSTRAINT constraint_name;
```

table_name

The name of the table that you wish to disable the check constraint.

constraint_name

The name of the check constraint to disable.

Example

Let's look at an example of how to disable a check constraint in SQL Server.

For example:

```
ALTER TABLE employees
NOCHECK CONSTRAINT check_salary;
```

In this SQL Server example, we are disabling a check constraint on the *employees* table called *check_salary*.

Summary

- **We studied what is constraint**
- **Drop and Unique constraint**
- **Different types of constraint key**
- **Also, we studied what is primary key how primary key is used and how to delete primary key**
- **Drop and check constraint**

MCQ's

1. Which of the following is not a class of constraint in SQL Server?
a) NOT NULL
b) CHECK
c) NULL
d) UNIQUE

2. Point out the correct statement:
a) CHECK constraints enforce domain integrity
b) UNIQUE constraints enforce the uniqueness of the values in a set of columns
c) In a UNIQUE constraint, no two rows in the table can have the same value for the columns
d) All of the mentioned

3. Which of the following constraint does not enforce uniqueness?
a) UNIQUE
b) Primary key
c) Foreign key
d) None of the mentioned

4. Constraints can be applied on:
a) Column
b) Table
c) Field
d) All of the mentioned

5. Point out the wrong statement :
a) Table constraints must be used when more than one column must be included in a constraint
b) A column constraint is specified as part of a column definition and applies only to that column
c) A table constraint is declared independently from a column definition and can apply to more than one column in a table
d) Primary keys allow for NULL as one of the unique values

6. Purpose of foreign key constraint in SQL Server is :
a) FOREIGN KEY constraints identify and enforce the relationships between tables
b) A foreign key in one table points to a candidate key in another table
c) You cannot insert a row with a foreign key value, except NULL, if there is no candidate key with that value
d) None of the mentioned

7. Which of the following is not a foreign key constraint?
a) NO ACTION
b) CASCADE
c) SET NULL
d) All of the mentioned

8. Which of the following foreign key constraint specifies that the deletion fails with an error?
a) NO ACTION
b) CASCADE
c) SET NULL
d) All of the mentioned

9. How many types of constraints are present in SQL Server?
a) 4
b) 5
c) 6
d) 7

10. Which of the constraint can be enforced one per table?
a) Primary key constraint
b) Not Null constraint
c) Foreign Key constraint
d) Check constraint

Fill in the blank

1.SQL constraints are used to specify ----------.
2Constraints can be divided into the following two types, ----------.
3. The CHECK Constraint enables a condition to check the ----------.
4. A primary key is a field in a table which uniquely ----------.
5. The UNIQUE Constraint prevents two records from ---------- in a column.

Chapter 3

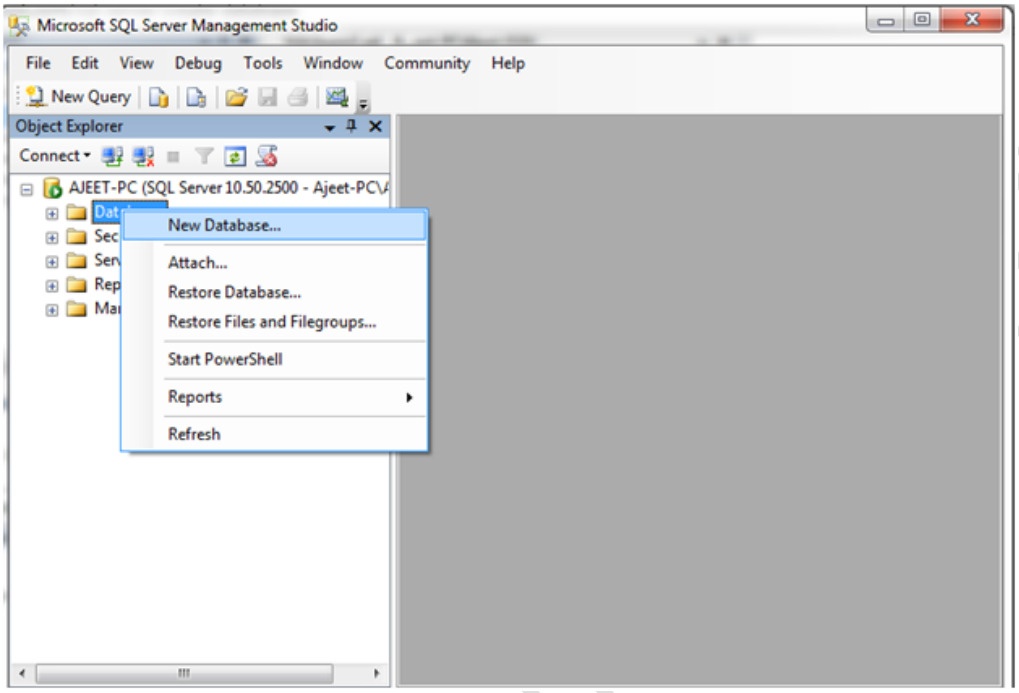
SQL Statements

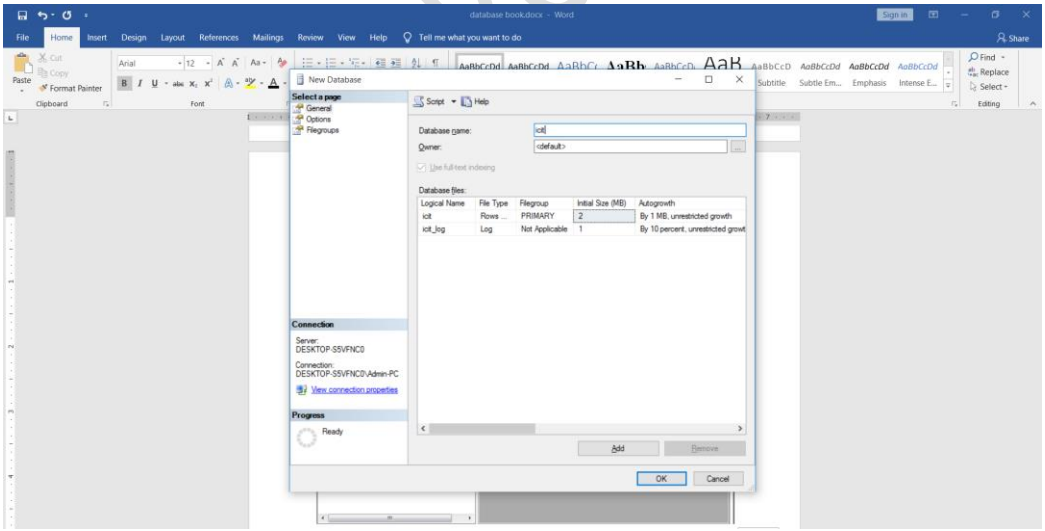
- Creation Database
- SQL Server Update Data
- SQL Insert Statement Table
- SQL Server DELETE Data
- SQL SELECT Statement
- SQL CLAUSES
- SQL SELECT IN
- BETWEEN like clause
- SQL indexes
- DROP INDEX Statement
- DESC

Assignment No 9

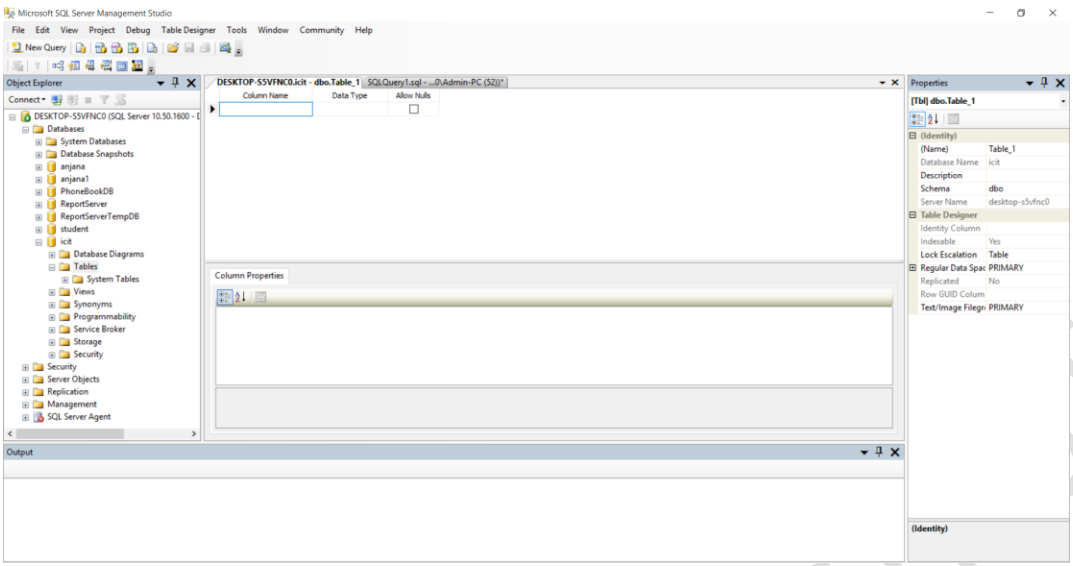
3.1 Creation for Database

Step 1: -Create new database and give name like “ICIT”

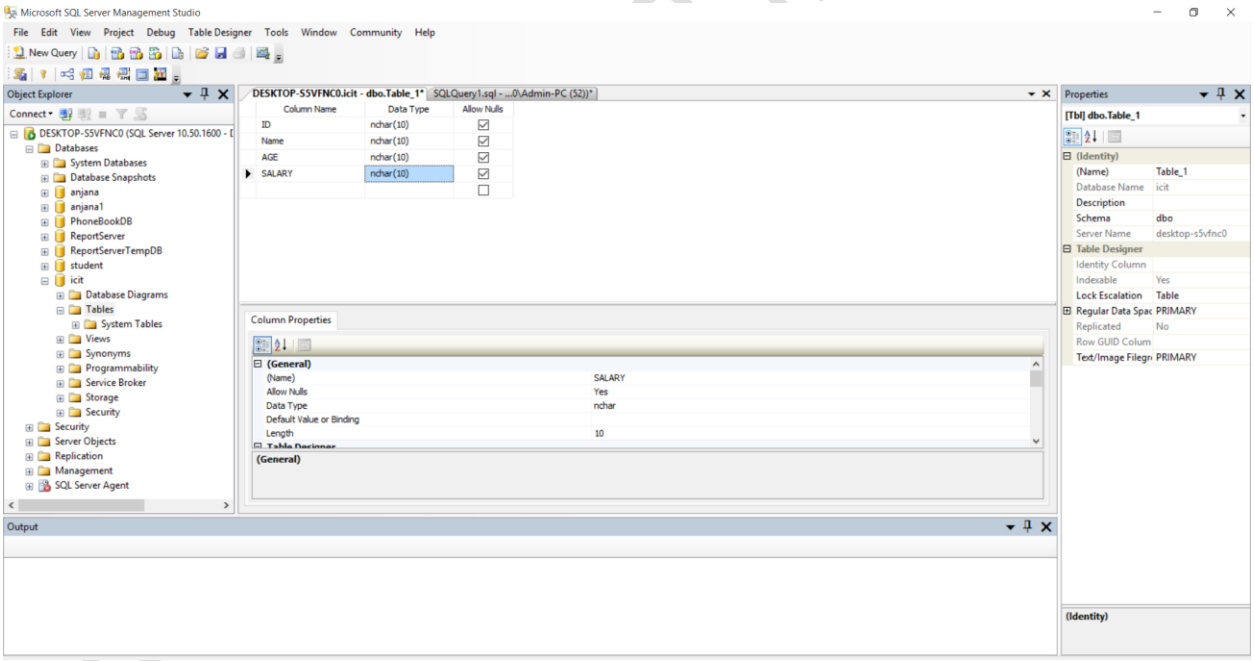




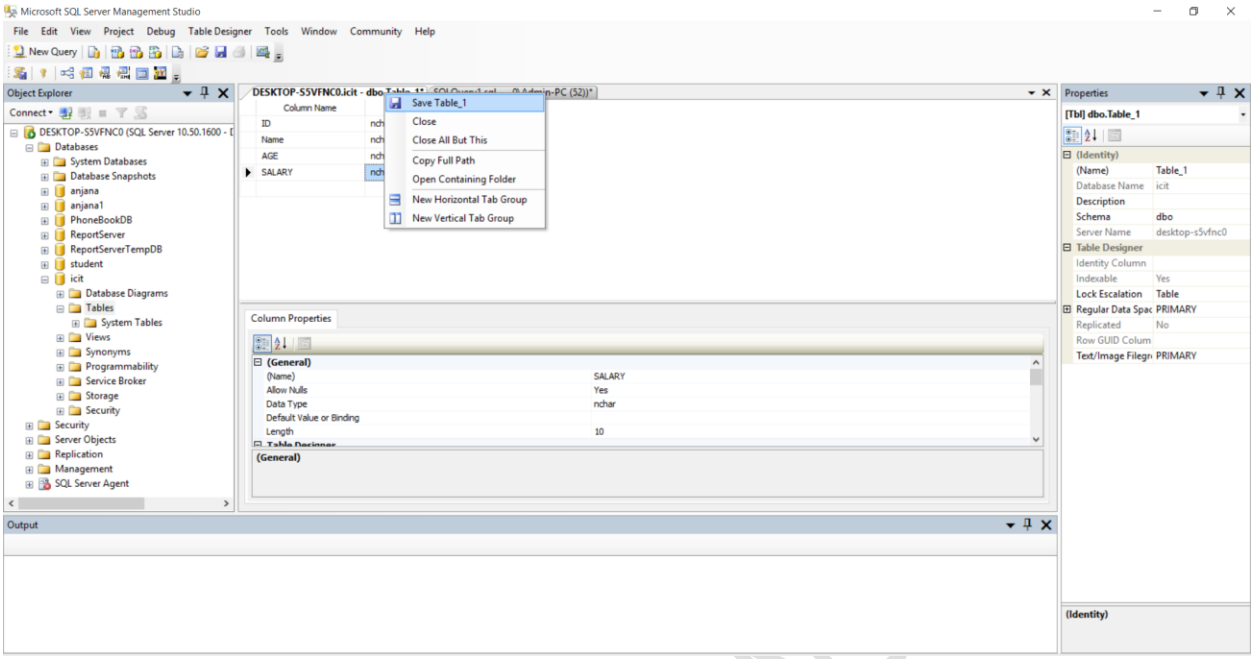
Step 2: Right click on "Tables" and you will see New Table. Click on New Table and you will see the following page:



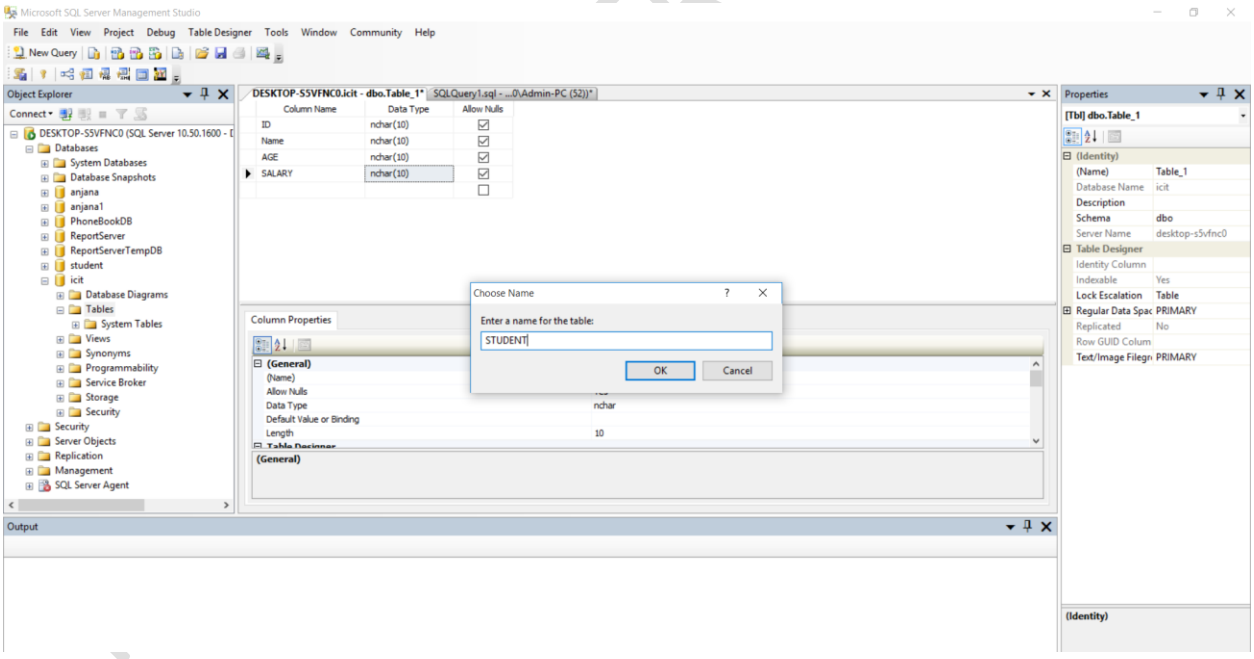
Step 3: You can add Column Name and Datatype as amay may as you want. Every time you fill one entry, another entry will be automatically created.



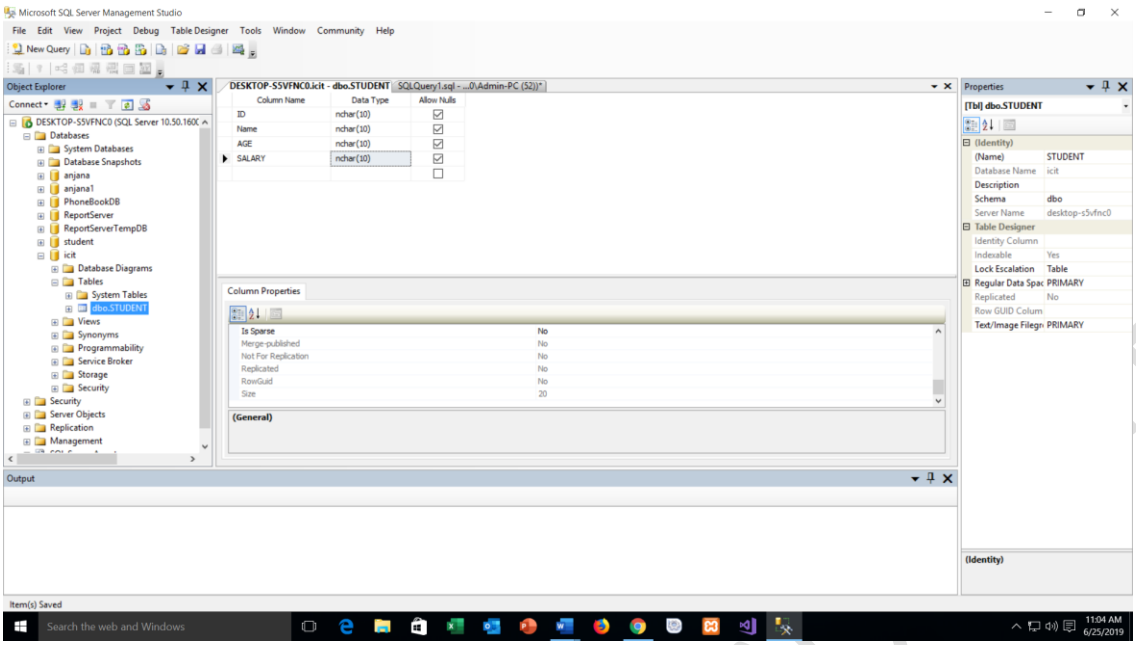
Step 4: Go to the encircled area and right click there, you will see like following image:



Step 5: Now click on the "Save Table" and save the table by any name. For example: Let's save the table by "STUDENT" name.



Step 6: Now a table "STUDENT" is created.



3.2 SQL Insert Statement Table

In SQL Server database, INSERT statement is used to insert a single record or multiple records into a table.

Syntax:

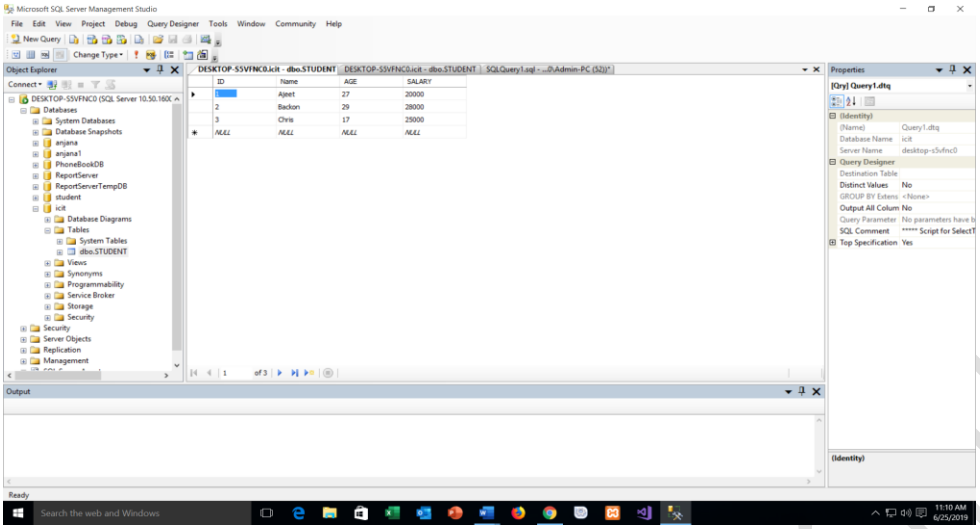
INSERT INTO [database_name].[dbo].[table_name]
(column1, column2, ...)
VALUES
(expression1, expression2, ...),
(expression1, expression2, ...),
...;

Assignment No. 10

```
INSERT INTO [icit].[dbo].[STUDENT]
        ([ID]
        ,[NAME]
        ,[AGE]
        ,[SALARY])
    VALUES
        (1,'Ajeet',27,20000),
        (2,'Backon',29,28000),
        (3,'Chris',17,25000);
```

Output

(3 row(s) affected)



3.3 SQL Server Update Data

In SQL Server database, UPDATE statement is used to update or modify the already inserted records into a table.

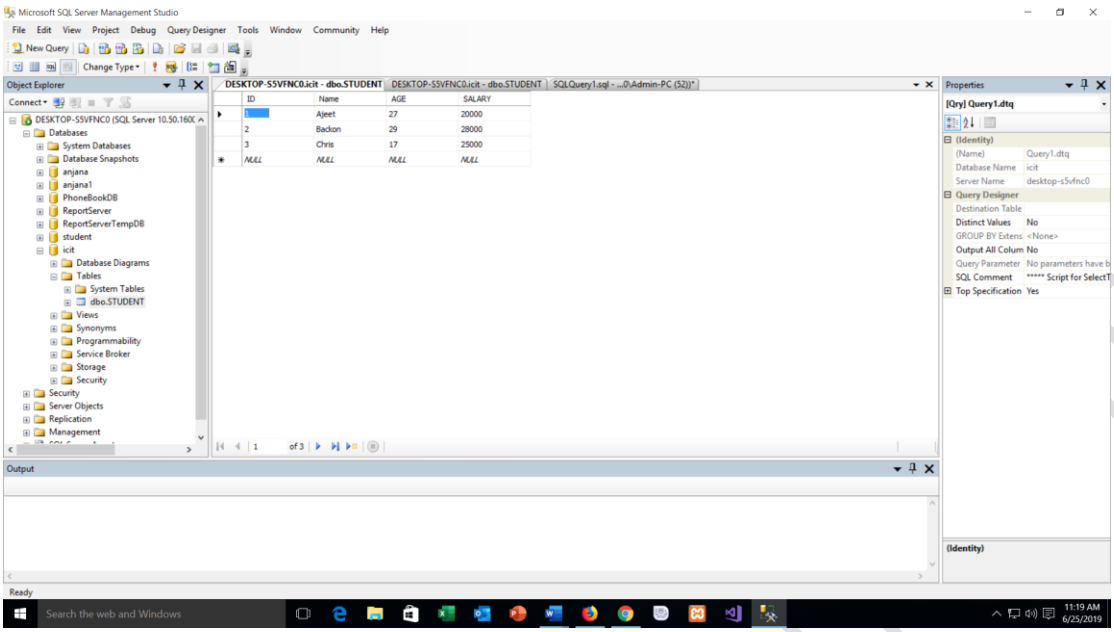
Syntax:

```
UPDATE [icit].[dbo].[STUDENT]
  SET [ID] = <ID, nchar(10),>
    ,[NAME] = <NAME, nchar(10),>
    ,[AGE] = <AGE, nchar(10),>
    ,[SALARY] = <SALARY, nchar(10),>
 WHERE <Search Conditions,,>
GO
```

Assignment No. 11

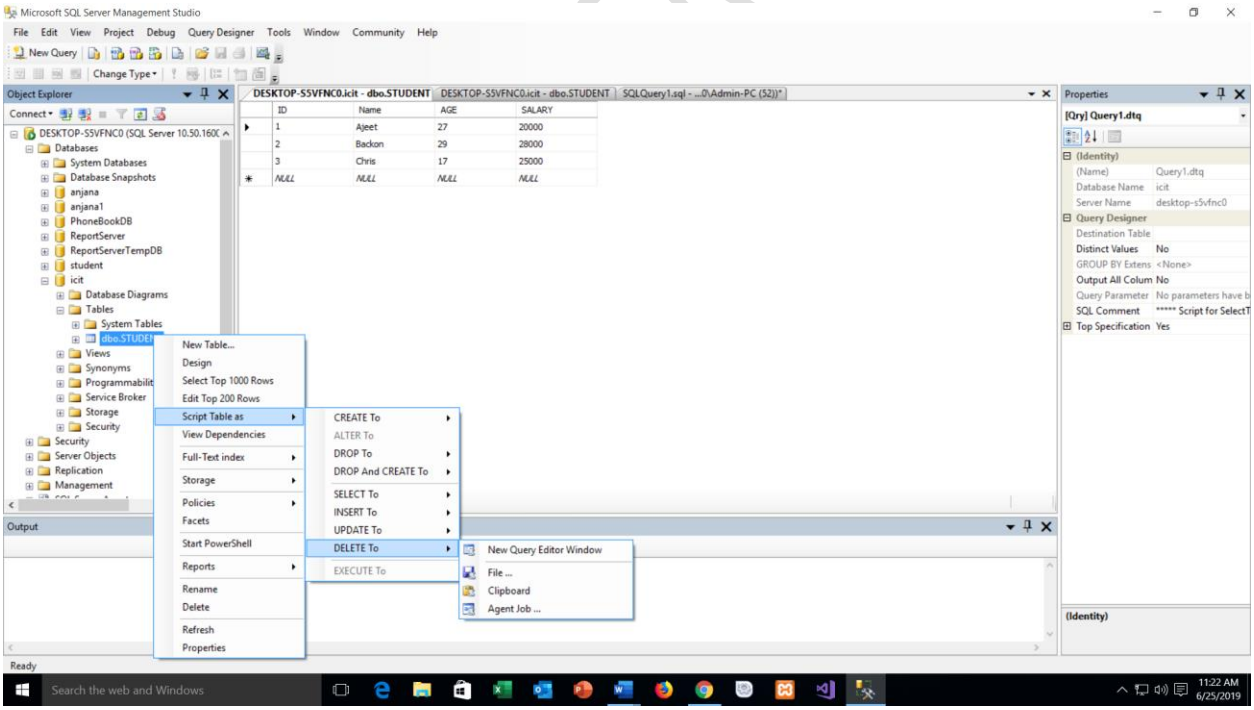
```
UPDATE [icit].[dbo].[STUDENT]
   SET [ID] = 4,[NAME] = 'Malvika',[AGE] = 18,[SALARY] = 22000
 WHERE ID =2;
GO
```

Output



3.4 SQL Server DELETE Data

In SQL Server database, DELETE statement is used to delete records from the table



Syntax:

```
SELECT TOP 1000 [ID]
      ,[NAME]
```

```
      ,[AGE]
      ,[SALARY]
  FROM [icit].[dbo].[STUDENT]
```

Assignment 12

```
DELETE FROM [icit].[dbo].[STUDENT]
    WHERE ID =3;
```

Output

(1 row(s) affected)

3.5 SQL SELECT Statement

The SELECT statement is used to select data from a database.

The data returned is stored in a result table, called the result-set.

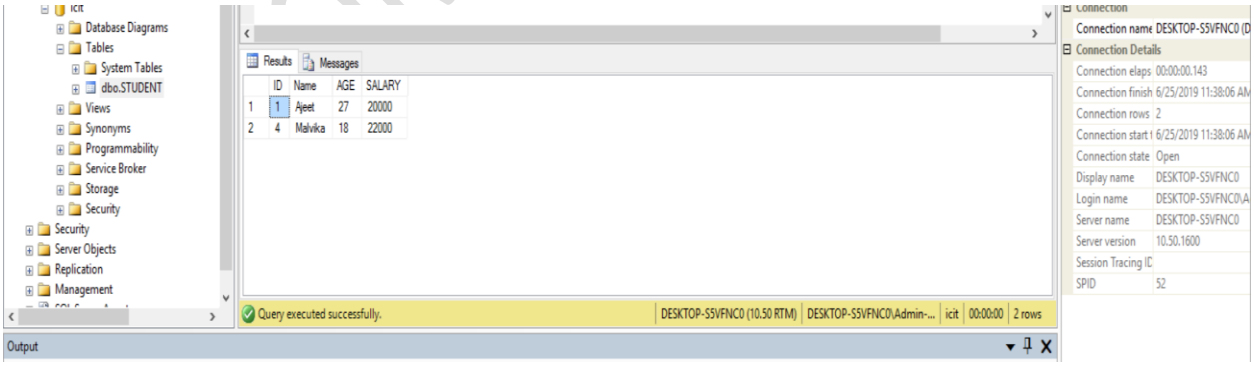
SELECT Syntax

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

Assignment No12

```
SELECT * FROM student;
```

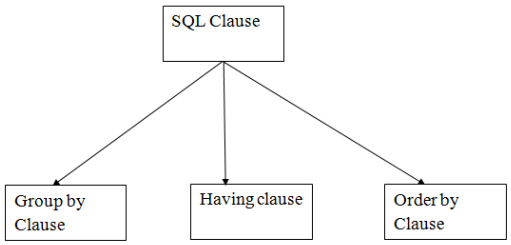
Output



3.6 SQL CLAUSES

SQL clauses site was designed to help programmers and IT professionals, yet unfamiliar with SQL (Structured Query Language) to learn the language and use it in their everyday work.

If SQL clauses and commands like SELECT, INSERT, UPDATE, DELETE, WHERE, JOIN, DISTINCT, ORDER BY, GROUP BY, HAVING, and UNION sound like ancient Greek to you, then you have come to the right place.



1. GROUP BY

- SQL GROUP BY statement is used to arrange identical data into groups. The GROUP BY statement is used with the SQL SELECT statement.
- The GROUP BY statement follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.
- The GROUP BY statement is used with aggregation function.

Syntax

SELECT column
FROM table_name
WHERE conditions
GROUP BY column
ORDER BY column

Assignment No 13

Demo Database

Below is a selection from the "Customers" table in the Northwind sample database:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country;
Output
Number of Records: 21

Result:

Number of Records: 21

COUNT(CustomerID)	Country
3	Argentina
2	Austria
2	Belgium
9	Brazil
3	Canada
2	Denmark
2	Finland
11	France
11	Germany
1	Ireland
3	Italy
5	Mexico
1	Norway
1	Poland
2	Portugal
5	Spain
2	Sweden
7	UK
13	USA
4	Venezuela

Assignment No 14
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
ORDER BY COUNT(CustomerID) DESC;
Output

Result:

Number of Records: 21

COUNT(CustomerID)	Country
13	USA
11	France
11	Germany
9	Brazil
7	UK
5	Mexico
5	Spain
4	Venezuela
3	Argentina
3	Canada
3	Italy
2	Austria
2	Belgium
2	Denmark
2	Finland
2	Portugal

2. HAVING

- HAVING clause is used to specify a search condition for a group or an aggregate.
- Having is used in a GROUP BY clause. If you are not using GROUP BY clause then you can use HAVING function like a WHERE clause.

Syntax:

SELECT *column_name(s)*
FROM *table_name*
WHERE *condition*
GROUP BY *column_name(s)*
HAVING *condition*
ORDER BY *column_name(s);*

Assignment No 15

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5
ORDER BY COUNT(CustomerID) DESC;
```

Output

Number of Records: 5

COUNT(CustomerID)	Country
13	USA
11	France
11	Germany
9	Brazil
7	UK

Demo Database

Demo Database

Below is a selection from the "Orders" table in the Northwind sample database:

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10248	90	5	1996-07-04	3
10249	81	6	1996-07-05	1
10250	34	4	1996-07-08	2

And a selection from the "Employees" table:

EmployeeID	LastName	FirstName	BirthDate	Photo	Notes
1	Davolio	Nancy	1968-12-08	EmpID1.pic	Education includes a BA....
2	Fuller	Andrew	1952-02-19	EmpID2.pic	Andrew received his BTS....
3	Leverling	Janet	1963-08-30	EmpID3.pic	Janet has a BS degree....

Assignment No 16

```
SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders
FROM (Orders
INNER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID)
GROUP BY LastName
HAVING COUNT (Orders.OrderID) > 10;
```

Output

Number of Records: 8

LastName	NumberOfOrders
Buchanan	11
Callahan	27
Davolio	29
Fuller	20
King	14
Leverling	31
Peacock	40
Suyama	18

3.7 SQL SELECT IN

The IN operator allows you to specify multiple values in a WHERE clause.
The IN operator is a shorthand for multiple OR conditions
IN Syntax
SELECT *column_name(s)*
FROM *table_name*
WHERE *column_name* IN (*value1*, *value2*, ...);

Assignment No17

SELECT * FROM Customers
WHERE Country IN ('Germany', 'France', 'UK');

Number of Records: 29

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
6	Blauer See Delikatessen	Hanna Moos	Forsterstr. 57	Mannheim	68306	Germany
7	Blondel père et fils	Frédérique Citeaux	24, place Kléber	Strasbourg	67000	France
9	Bon app'	Laurence Lebihans	12, rue des Bouchers	Marseille	13008	France
11	B's Beverages	Victoria Ashworth	Fauntleroy Circus	London	EC2 5NT	UK
16	Consolidated Holdings	Elizabeth Brown	Berkeley Gardens 12 Brewery	London	WX1 6LT	UK
17	Drachenblut Delikatessend	Sven Ottlieb	Walserweg 21	Aachen	52066	Germany
18	Du monde entier	Janine Labrune	67, rue des Cinquante Otages	Nantes	44000	France
19	Eastern Connection	Ann Devon	35 King George	London	WX3 6FW	UK
23	Folies gourmandes	Martine Rancé	184, chaussée de Tournai	Lille	59000	France
25	Frankenversand	Peter Franken	Berliner Platz 43	München	80805	Germany
26	France restauration	Carine Schmitt	54, rue Royale	Nantes	44000	France
38	Island Trading	Helen Bennett	Garden House Crowther Way	Cowes	PO31 7PJ	UK

3.8 BETWEEN

The SQL BETWEEN condition allows you to easily test if an expression is within a range of values (inclusive). The values can be text, date, or numbers. It can be used in a SELECT, INSERT, UPDATE, or DELETE statement.

The SQL BETWEEN Condition will return the records where expression is within the range of value1 and value2.

Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

Assignment No18

Consider the following Employee Table,

Fname	Lname	SSN	Salary	DOB
John	Smith	123456789	30000	1988-05-02
Franklin	Wong	333445555	40000	1986-01-02
Joyce	English	453453453	80000	1977-12-08
Ramesh	Narayan	666884444	38000	1987-03-05
James	Borg	888665555	55000	1982-10-10
Jennifer	Wallace	987654321	43000	1985-08-07
Ahmad	Jabbar	987987987	25000	1990-06-28
Alicia	Zeala	999887777	25000	1980-09-14

Queries

- **Using BETWEEN with Numeric Values:**

  List all the Employee Fname, Lname who is having salary between 30000 and 45000.

SELECT Fname, Lname
FROM Employee
WHERE Salary
BETWEEN 30000 AND 45000;
Output:

Fname	Lname
John	Smith
Franklin	Wong
Ramesh	Narayan
Jennifer	Wallace

3.9 LIKE CLAUSE

Description

The SQL LIKE condition allows you to use wildcards to perform pattern matching in a query. The LIKE condition is used in the WHERE clause of a SELECT, INSERT, UPDATE, or DELETE statement.

Syntax

The syntax for the LIKE condition in SQL is:

expression LIKE pattern [ESCAPE 'escape_character']

Parameters or Arguments

expression

A character expression such as a column or field.

pattern

A character expression that contains pattern matching. The wildcards that you can choose from are:

Wildcard	Explanation
%	Allows you to match any string of any length (including zero length)
_	Allows you to match on a single character

ESCAPE 'escape_character'

Optional. It allows you to pattern match on literal instances of a wildcard character such as % or _.

TIP: If you are pattern matching with char datatypes, remember that chars are padded with spaces at the end to fill the length of the field. This may give you unexpected results when you use the LIKE condition to pattern match at the end of a string.

Example - Using % Wildcard in the LIKE Condition

Assignment 19

Let's explain how the % wildcard works in the SQL LIKE condition. Remember that the % wildcard matches any string of any length (including zero length).

In this first example, we want to find all of the records in the *customers* table where the customer's *last_name* begins with 'J'.

In this example, we have a table called *customers* with the following data:

customer_id	last_name	first_name	favorite_website
4000	Jackson	Joe	techonthenet.com
5000	Smith	Jane	digminecraft.com

6000	Ferguson	Samantha	bigactivities.com
7000	Reynolds	Allen	checkyourmath.com
8000	Anderson	Paige	NULL
9000	Johnson	Derek	techonthenet.com

Query

```
SELECT *
FROM customers
WHERE last_name LIKE 'J%'
ORDER BY last_name;
```

Result

customer_id	last_name	first_name	favorite_website
4000	Jackson	Joe	techonthenet.com
9000	Johnson	Derek	techonthenet.com

This example returns the records in the *customers* table where the *last_name* starts with 'J'. As you can see, the records for the last names Jackson and Johnson have been returned.

Because the LIKE condition is not case-sensitive, the following SQL statement would return the same results:

```
SELECT *
FROM customers
WHERE last_name LIKE 'j%'
ORDER BY last_name;
```

Using Multiple **%** Wildcards in the LIKE Condition

You can also using the % wildcard multiple times with the LIKE condition.

Using the same *customers* table with the following data:

customer_id	last_name	first_name	favorite_website
4000	Jackson	Joe	techonthenet.com
5000	Smith	Jane	digminecraft.com
6000	Ferguson	Samantha	bigactivities.com
7000	Reynolds	Allen	checkyourmath.com
8000	Anderson	Paige	NULL
9000	Johnson	Derek	techonthenet.com

Let's try to find all *last_name* values from the *customers* table where the *last_name* contains the letter 'e'. Enter the following SQL statement:

```
SELECT last_name
FROM customers
WHERE last_name LIKE '%e%'
ORDER BY last_name;
```

There will be 3 records selected. These are the results that you should see:

last_name
Anderson
Ferguson
Reynolds

In this example, the last names Anderson, Ferguson and Reynolds contain the letter 'e'.

Example - Using _ Wildcard in the LIKE Condition
Next, let's explain how the _ wildcard (underscore wildcard) works in the LIKE condition. Remember that _ wildcard is looking for exactly one character, unlike the % wildcard.
Using the *categories* table with the following data:

category_id	category_name
25	Deli
50	Produce
75	Bakery
100	General Merchandise
125	Technology

Let's try to find all records from the *categories* table where the *category_id* is 2-digits long and ends with '5'. Enter the following SQL statement:

```
SELECT *
FROM categories
WHERE category_id LIKE '_5';
```

There will be 2 records selected. These are the results that you should see:

category_id	category_name
25	Deli
75	Bakery

In this example, there are 2 records that will pattern match - the *category_id values* 25 and 75. Notice that the *category_id* of 125 was not selected because, the _ wilcard matches only on a single character.

Using Multiple _ Wildcards in the LIKE Condition

If you wanted to match on a 3-digit value that ended with '5', you would need to use the _ wildcard two times. You could modify your query as follows:

```
SELECT *
FROM categories
WHERE category_id LIKE '__5';
```

Now you will return the *category_id* value of 125:

category_id	category_name
125	Technology

Example - Using the NOT Operator with the LIKE Condition

Next, let's look at an example of how to use the NOT Operator with the LIKE condition.

In this example, we have a table called *suppliers* with the following data:

supplier_id	supplier_name	city	state
100	Microsoft	Redmond	Washington
200	Google	Mountain View	California
300	Oracle	Redwood City	California
400	Kimberly-Clark	Irving	Texas
500	Tyson Foods	Springdale	Arkansas
600	SC Johnson	Racine	Wisconsin

700	Dole Food Company	Westlake Village	California
800	Flowers Foods	Thomasville	Georgia
900	Electronic Arts	Redwood City	California

Let's look for all records in the *suppliers* table where the *supplier_name* does **not** contain the letter 'o'. Enter the following SQL statement:

```
SELECT *
FROM suppliers
WHERE supplier_name NOT LIKE '%o%';
```

There will be 1 record selected. These are the results that you should see:

supplier_id	supplier_name	city	state
400	Kimberly-Clark	Irving	Texas

In this example, there is only one record in the *suppliers* table where the *supplier_name* does not contain the letter 'o'.

Example - Using Escape Characters with the LIKE Condition

It is important to understand how to "Escape Characters" when pattern matching. You can escape % or _ and search for the literal versions instead.

Let's say you wanted to search for % as a literal in the LIKE condition. You can do this using an Escape character. In our example, we will use ! as the escape character in the LIKE condition.

NOTE: You can only define an escape character as a single character. It is best to choose a character that will not appear in your data very often such as ! or #.

In this example, we a table called *test* with the following data:

test_id	test_value
1	10%
2	25%
3	100
4	99

We could return all records from the *test* table where the *test_value* contains the % literal. Enter the following SQL statement:

```
SELECT *
FROM test
WHERE test_value LIKE '%!%%' escape '!';
```

These are the results that you should see:

test_id	test_value
1	10%
2	25%

This example identifies the ! character as an escape character. The first and last % values in the LIKE condition are treated as regular wildcards. The !% is an escaped % so it is treated as a literal % value.

You could further modify the above example and only return *test_values* that start with 1 and contain the % literal. Enter the following SQL statement:

SELECT *
FROM test
WHERE test_value LIKE '1%!%%' escape '!';
These are the results that you should see:

test_id	test_value
1	10%

This example will only return one record this time. Because there is only one *test_value* that starts with 1 and contains the % literal.

3.9 SQL indexes

An index is a schema object. It is used by the server to speed up the retrieval of rows by using a pointer.

It can reduce disk I/O(input/output) by using a rapid path access method to locate data quickly. An index helps to speed up select queries and where clauses, but it slows down data input, with the update and the insert statements.

Indexes are used to retrieve data from the database very fast. The users cannot see the indexes, they are just used to speed up searches/queries.

Note: Updating a table with indexes takes more time than updating a table without (because the indexes also need an update). So, only create indexes on columns that will be frequently searched against.

CREATE INDEX Syntax

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

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

CREATE UNIQUE INDEX Syntax

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

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

Note: The syntax for creating indexes varies among different databases. Therefore: Check the syntax for creating indexes in your database.

Example

Assignment No 20

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

CREATE INDEX Example

The SQL statement below creates an index named "idx_lastname" on the "LastName" column in the "Persons" table:

```
CREATE INDEX idx_lastname
ON Persons (LastName);
```

DROP INDEX Statement

The DROP INDEX statement is used to delete an index in a table.

DROP INDEX *table_name*. *index_name*;

DROP TABLE

The DROP TABLE command deletes a table in the database.

The following SQL deletes the table "Shippers":

DROP TABLE Shippers;

Note: Be careful before deleting a table. Deleting a table results in loss of all information stored in the table!

3.10 DESC

The DESC command is used to sort the data returned in descending order.

The following SQL statement selects all the columns from the "Customers" table, sorted descending by the "CustomerName" column:

Assignment No 21

SELECT * FROM Customers
ORDER BY CustomerName DESC;

Output

Result:

Number of Records: 91

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
91	Wolski	Zbyszek	ul. Filtrowa 68	Walla	01-012	Poland
90	Wilman Kala	Matti Karttunen	Keskuskatu 45	Helsinki	21240	Finland
89	White Clover Markets	Karl Jablonski	305 - 14th Ave. S. Suite 3B	Seattle	98128	USA
88	Wellington Importadora	Paula Parente	Rua do Mercado, 12	Resende	08737-363	Brazil
87	Wartian Herkku	Pirkko Koskitalo	Torikatu 38	Oulu	90110	Finland
85	Vins et alcools Chevalier	Paul Henriot	59 rue de l'Abbaye	Reims	51100	France
84	Victuailles en stock	Mary Saveley	2, rue du Commerce	Lyon	69004	France
83	Vaffeljernet	Palle Ibsen	Smagsløget 45	Århus	8200	Denmark
82	Trail's Head Gourmet Provisioners	Helvetius Nagy	722 DaVinci Blvd.	Kirkland	98034	USA
81	Tradição Hipermercados	Anabela Domingues	Av. Inês de Castro, 414	São Paulo	05634-030	Brazil
80	Tortuga Restaurante	Miguel Angel Paolino	Avda. Azteca 123	México D.F.	05033	Mexico

Summary

- **In this chapter, you have learned about:**
- **We studied, how to create table in SQL**
- **Different types of statements**
- **Also, we studied syntax for all statements and Examples**
- **We studied clauses in SQL**
- **Also, we studied like and between clauses**
- **Also, we studied Index statement DESC statement**

MCQ's

1. What is the meaning of "SELECT" clause in MySQL?

a) Show me all Columns and rows
b) Show me all columns
c) Show me all rows
d) None of the mentioned

2. Which of the following clause is evaluated in the last by database server?

a) SELECT
b) WHERE
c) FROM
d) None of the mentioned

3. What will be the output of a query given below?

```
SELECT *
 FROM person;
```

a) Show all rows and columns of table “person”
b) Show all rows of table “person”
c) Show all columns of table “person”
d) None of the mentioned

4. What will be the output of a query given below?

```
SELECT person_id, Fname, lname
FROM person;
```

a) Show only columns (person_id, Fname, lname) and rows related to these columns
b) Show only columns (person_id, Fname, lname)
c) Show all rows
d) Show all columns except (person_id, Fname, lname)

5. Can “SELECT” clause be used without the clause “FROM”?
a) YES
b) NO
c) DEPENDS
d) None of the mentioned

6. Find the error?

```
SELECT *;
```

a) No Error
b) No table mentioned
c) Depends
d) None of the mentioned

7. What will be the output of a query given below?

```
SELECT  * FROM person
WHERE person_id=1;
```

a) Show all columns but only those rows which belongs to person_id=1
b) Show all columns and rows
c) Shows only columns person_id
d) None of the mentioned

8. What will be the output of a query given below?

```
SELECT person_id, fname, lname
FROM person
WHERE person_id=1;
```

a) Show only columns (person_id, fname, lname) but only those rows which belongs to person_id=1
b) Show all columns and rows
c) Shows only columns person_id
d) None of the mentioned

9. Which clause is mandatory with clause “SELECT” in Mysql?
a) FROM
b) WHERE
c) Both FROM and WHERE
d) None of the mentioned

10.What is the need of “column Aliases” in “SELECT” clause?
a) To assign a new label to the column in result set
b) To overwrite the existing column name in result set
c) To modify the column name while using literals, Expression, built_in functions with “SELECT clause
d) All of the mentioned

Fill in the blank

1.The HAVING clause is an additional filter that is applied to the ------

2. The ---------- command deletes a table in the database

3. An index is a ----------

4. The CASE statement goes through conditions and returns a value when the ---------- (like an IF-THEN-ELSE statement).

5. In particular it is used in the SELECT column list, ----------clauses.

Chapter 4

SQL Table

- What is SQL Table?
- SQL Table Variable
- SQL Create Table
- Delete Truncate Statements
- Difference between DROP and TRUNCATE Statements
- SQL Alter Table
- Insert into SELECT Syntax
- SQL Update Statement
- SQL Delete Statement

4.1 SQL Table

Table is a collection of data, organized in terms of rows and columns. In DBMS term, table is known as relation and row as tuple.

Note: A table has a specified number of columns, but can have any number of rows.

Table is the simple form of data storage. A table is also considered as a convenient representation of relations.

Let's see an example of an employee table:

EMP_NAME	ADDRESS	SALARY
Ankit	Lucknow	15000
Raman	Allahabad	18000
Mike	New York	20000

In the above table, "Employee" is the table name, "EMP_NAME", "ADDRESS" and "SALARY" are the column names. The combination of data of multiple columns forms a row e.g. "Ankit", "Lucknow" and 15000 are the data of one row.

4.1.1 SQL TABLE Variable

The SQL Table variable is used to create, modify, rename, copy and delete tables. Table variable was introduced by Microsoft.
It was introduced with SQL server 2000 to be an alternative of temporary tables.
It is a variable where we temporary store records and results. This is same like temp table but in the case of temp table we need to explicitly drop it.
Table variables are used to store a set of records. So declaration syntax generally looks like
CREATE TABLE syntax.

```
create table "tablename"
("column1" "data type",
"column2" "data type",
...
"columnN" "data type");
```

When a transaction rolled back the data associated with table variable is not rolled back.
A table variable generally uses lesser resources than a temporary variable.
Table variable cannot be used as an input or an output parameter.

4.1.2SQL CREATE TABLE

SQL CREATE TABLE statement is used to create table in a database.

If you want to create a table, you should name the table and define its column and each column's data type.

Let's see the simple syntax to create the table.

```
create table "tablename"
("column1" "data type",
"column2" "data type",
"column3" "data type",
...
"columnN" "data type");
```

The data type of the columns may vary from one database to another. For example, NUMBER is supported in Oracle database for integer value whereas INT is supported in MySQL.

Let us take an example to create a STUDENTS table with ID as primary key and NOT NULL are the constraint showing that these fields cannot be NULL while creating records in the table.

Assignment No 22

```
CREATE TABLE STUDENTS (
ID INT                  NOT NULL,
NAME VARCHAR (20) NOT NULL,
AGE INT                  NOT NULL,
ADDRESS CHAR (25),
PRIMARY KEY (ID)
);
```

SQL CREATE TABLE Example in Microsoft SQLServer

Let's see the command to create a table in SQLServer database. It is same as MySQL and Oracle.

```
CREATE TABLE Employee
(
EmployeeID int,
FirstName varchar(255),
LastName varchar(255),
Email varchar (255),
AddressLine varchar (255),
City varchar(255)
);
```

SQL Primary Key with CREATE TABLE Statement

```
CREATE TABLE Employee(
EmployeeID NOT NULL PRIMARY KEY,
FirstName varchar(255) NOT NULL,
LastName varchar(255),
City varchar(255)
);
```

4.1.3 DROP TABLE

A SQL DROP TABLE statement is used to delete a table definition and all data from a table.
This is very important to know that once a table is deleted all the information available in the table is lost forever, so we have to be very careful when using this command.

Let's see the syntax to drop the table from the database.
DROP TABLE "table_name";

4.1.4 SQL DELETE TABLE

The DELETE statement is used to delete rows from a table. If you want to remove a specific row from a table you should use WHERE condition.

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

But if you do not specify the WHERE condition it will remove all the rows from the table.

DELETE FROM table_name;

There are some more terms similar to DELETE statement like as DROP statement and TRUNCATE statement but they are not exactly same there are some differences between them.

Difference between DELETE and TRUNCATE statements

There is a slight difference b/w delete and truncate statement. The DELETE statement only deletes the rows from the table based on the condition defined by WHERE clause or delete all the rows from the table when condition is not specified.

But it does not free the space containing by the table.

The TRUNCATE statement: it is used to delete all the rows from the table and free the containing space.

Assignment No 23
Let's see an "employee" table.

Emp_id	Name	Address	Salary
1	Aryan	Allahabad	22000
2	Shurabhi	Varanasi	13000
3	Pappu	Delhi	24000

Execute the following query to truncate the table:

4.1.5 TRUNCATE TABLE employee;

Difference b/w DROP and TRUNCATE statements

When you use the drop statement it deletes the table's row together with the table's definition so all the relationships of that table with other tables will no longer be valid.
When you drop a table:
Table structure will be dropped
Relationship will be dropped
Integrity constraints will be dropped
Access privileges will also be dropped
On the other hand when we TRUNCATE a table, the table structure remains the same, so you will not face any of the above problems.

4.1.6 SQL RENAME TABLE

SQL RENAME TABLE syntax is used to change the name of a table. Sometimes, we choose non-meaningful name for the table. So it is required to be changed.

Let's see the syntax to rename a table from the database.

ALTER TABLE table_name
RENAME TO new_table_name;
Optionally, you can write following command to rename the table.
RENAME old_table _name To new_table_name;
Let us take an example of a table named "STUDENTS", now due to some reason we want to change it into table name "ARTISTS".

Table1: students

Name	Age	City
Amrita gill	25	Amritsar
Amrender sirohi	22	Ghaziabad
Divya khosla	20	Delhi

You should use any one of the following syntax to RENAME the table name:

ALTER TABLE STUDENTS
RENAME TO ARTISTS;
Or
RENAME STUDENTS TO ARTISTS;
After that the table "students" will be changed into table name "artists"

4.1.7 SQL TRUNCATE TABLE

A truncate SQL statement is used to remove all rows (complete data) from a table. It is similar to the DELETE statement with no WHERE clause.

TRUNCATE TABLE Vs DELETE TABLE

Truncate table is faster and uses lesser resources than DELETE TABLE command.

4.1.8 TRUNCATE TABLE Vs DROP TABLE

Drop table command can also be used to delete complete table but it deletes table structure too. TRUNCATE TABLE doesn't delete the structure of the table.

Let's see the syntax to truncate the table from the database.

TRUNCATE TABLE table_name;

For example, you can write following command to truncate the data of employee table

TRUNCATE TABLE Employee;

Note: The rollback process is not possible after truncate table statement. Once you truncate a table you cannot use a flashback table statement to retrieve the content of the table.

4.1.9 SQL ALTER TABLE

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

You can also use SQL ALTER TABLE command to add and drop various constraints on an existing table.

SQL ALTER TABLE Add Column

If you want to add columns in SQL table, the SQL alter table syntax is given below:

```
ALTER TABLE table_name ADD column_name column-definition;
```

If you want to add multiple columns in table, the SQL table will be

```
ALTER TABLE table_name
ADD (column_1 column-definition,
      column_2 column-definition,
      .....
      column_n column-definition);
```

SQL ALTER TABLE Modify Column

If you want to modify an existing column in SQL table, syntax is given below:

ALTER TABLE table_name **MODIFY** column_name column_type;

If you want to modify multiple columns in table, the SQL table will be

```
ALTER TABLE table_name
MODIFY (column_1 column_type,
          column_2 column_type,
          .....
```

column_n column_type);

SQL ALTER TABLE DROP Column
The syntax of alter table drop column is given below:
ALTER TABLE table_name **DROP COLUMN** column_name;

SQL ALTER TABLE RENAME Column
The syntax of alter table rename column is given below:
ALTER TABLE table_name
RENAME **COLUMN** old_name **to** new_name;

SQL INSERT INTO SELECT Statement

The INSERT INTO SELECT statement copies data from one table and inserts it into another table.
INSERT INTO SELECT requires that data types in source and target tables match
The existing records in the target table are unaffected
INSERT INTO SELECT Syntax
Copy all columns from one table to another table:
INSERT INTO *table2*
SELECT * FROM *table1*
WHERE *condition*;
Copy only some columns from one table into another table:
INSERT INTO *table2* (*column1*, *column2*, *column3*, ...)
SELECT *column1*, *column2*, *column3*, ...
FROM *table1*
WHERE *condition*;

Assignment No 24

Demo Database
In this tutorial we will use the well-known Northwind sample database.
Below is a selection from the "Customers" table:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

And a selection from the "Suppliers" table:

SupplierID	SupplierName	ContactName	Address	City	Postal Code	Country
1	Exotic Liquid	Charlotte Cooper	49 Gilbert St.	Londona	EC1 4SD	UK
2	New Orleans Cajun Delights	Shelley Burke	P.O. Box 78934	New Orleans	70117	USA
3	Grandma Kelly's Homestead	Regina Murphy	707 Oxford Rd.	Ann Arbor	48104	USA

SQL Statement:

INSERT INTO Customers (CustomerName, City, Country)
SELECT SupplierName, City, Country FROM Suppliers;
SQL Statement:
INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)
SELECT SupplierName, ContactName, Address, City, PostalCode, Country FROM Suppliers;
The following SQL statement copies "Suppliers" into "Customers" (fill all columns):
INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)
SELECT SupplierName, ContactName, Address, City, PostalCode, Country FROM Suppliers;

SQL Statement:

INSERT INTO Customers (CustomerName, City, Country)
SELECT SupplierName, City, Country FROM Suppliers
WHERE Country='Germany';
The SQL UPDATE Statement
The UPDATE statement is used to modify the existing records in a table.
UPDATE Syntax
UPDATE *table_name*
SET *column1 = value1, column2 = value2, ...*
WHERE *condition*;
Note: Be careful when updating records in a table! Notice the WHERE clause in the UPDATE statement. The WHERE clause specifies which record(s) that should be updated. If you omit the WHERE clause, all records in the table will be updated!

Assignment No 25

Demo Database

Below is a selection from the "Customers" table in the Northwind sample database:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

UPDATE Customers
SET ContactName='Alfred Schmidt', City='Frankfurt'
WHERE CustomerID=1;

UPDATE Multiple Records
It is the WHERE clause that determines how many records that will be updated.
The following SQL statement will update the contactname to "Juan" for all records where country is "Mexico":
UPDATE Customers
SET ContactName='Juan'
WHERE Country='Mexico';
The SQL DELETE Statement
The DELETE statement is used to delete existing records in a table.
DELETE Syntax
DELETE FROM *table_name* WHERE *condition*;

Note: Be careful when deleting records in a table! Notice the WHERE clause in the DELETE statement. The WHERE clause specifies which record(s) should be deleted. If you omit the WHERE clause, all records in the table will be deleted!

Assignment No 24

Demo Database

Below is a selection from the "Customers" table in the Northwind sample database:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

SQL DELETE Example

The following SQL statement deletes the customer "Alfreds Futterkiste" from the "Customers" table:

DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';

Delete All Records

It is possible to delete all rows in a table without deleting the table. This means that the table structure, attributes, and indexes will be intact:

DELETE FROM *table_name*;

The following SQL statement deletes all rows in the "Customers" table, without deleting the table:

Example

DELETE FROM Customers;

Summary

- **In this chapter, you have learned about:**
- **In this chapter We studied Sql Table, Sql Table variable.**
- **Also, we studied all the syntax for Table creation, Updation and Deletion, as well as Example for all Table creation, Updation and Deletion.**

MCQ's

1. Which SQL function is used to count the number of rows in a SQL query?

a) COUNT()

b) NUMBER()

c) SUM()

d) COUNT (*)

2. Which SQL keyword is used to retrieve a maximum value?

a) MOST

b) TOP

c) MAX
d) UPPER

3. Which of the following SQL clauses is used to DELETE tuples from a database table?
a) DELETE
b) REMOVE
c) DROP
d) CLEAR

4. _____removes all rows from a table without logging the individual row deletions.
a) DELETE
b) REMOVE
c) DROP
d) TRUNCATE

5. Which of the following is not a DDL command?
a) UPDATE
b) TRUNCATE
c) ALTER
d) None of the Mentioned

6. Which of the following are TCL commands?
a) UPDATE and TRUNCATE
b) SELECT and INSERT
c) GRANT and REVOKE
d) ROLLBACK and SAVEPOINT

7. _____ is not a category of SQL command.
a) TCL
b) SCL
c) DCL
d) DDL

8. If you don't specify ASC or DESC after a SQL ORDER BY clause, the following is used by default
a) ASC
b) DESC
c) There is no default value
d) None of the mentioned

9. Which of the following statement is true?
a) DELETE does not free the space containing the table and TRUNCATE free the space containing the table
b) Both DELETE and TRUNCATE free the space containing the table
c) Both DELETE and TRUNCATE does not free the space containing the table
d) DELETE free the space containing the table and TRUNCATE does not free the space containing the table

10. What is the purpose of the SQL AS clause?
a) The AS SQL clause is used change the name of a column in the result set or to assign a name to a derived column
b) The AS clause is used with the JOIN clause only
c) The AS clause defines a search condition
d) All of the mentioned

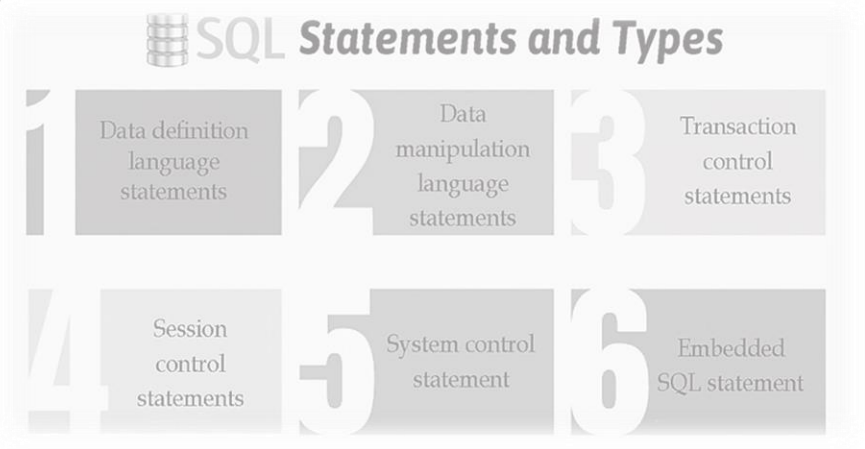
Fill in the blank

1. Table is a collection of data, organized in terms of ----------.
2. SQL CREATE TABLE statement is used to create table ----------.
3. A SQL DROP TABLE statement is used to ----------.
4. The TRUNCATE statement: it is used to delete all the ----------.
5. SQL RENAME TABLE syntax is ----------.

Chapter 5

SQL Statements and Data Types

- SQL Use statement
- SQL COMMIT and ROLLBACK Statement
- SQL Commit Example
- SQL Rollback
- Updated command with Rollback
- Delete with Rollback



5.1 SQL USE Statement

The USE Statement is used to select a database and perform SQL operations into that database. The database remains default until end of session or execution of another USE statement with some other database.

5.1.1 SQL USE DATABASE Statement:

The Syntax for the USE Statement is:
USE database_name;

database_name - is the name of the database to be selected

USE DATABASE Example:
If you want to use database MyDatabase, the statement would be like

```
USE MyDatabase ;
```

5.1.2 SQL COMMIT and Rollback Statement

COMMIT and ROLLBACK are performed on transactions.
A transaction is the smallest unit of work that is performed against a database.
It's a sequence of instructions in a logical order.
A transaction can be performed manually by a programmer or it can be triggered using an automated program.

5.1.3 SQL Commit

COMMIT is the SQL command that is used for storing changes performed by a transaction.
When a COMMIT command is issued it saves all the changes since last COMMIT or ROLLBACK.

Syntax for SQL Commit
COMMIT
SQL Commit Example

Assignment No 27

Let us consider the following table for understanding Commit in a better way.;
Customer: -

CUSTOMER ID	CUSTOMER NAME	STATE	COUNTRY
1	Akash	Delhi	India

2	**Amit**	**Hyderabad**	**India**
3	**Jason**	**California**	**USA**
4	**John**	**Texas**	**USA**

Now let us delete one row from the above table where State is “Texas”.

```
1  DELETE from public."Customer" where "State"='Texas'
```

Data Output | Explain | Messages | Query History

	Customer_Id bigint	Customer_Name character varying (50)	State character varying (20)	Country character varying (20)
1	1	Akash	Delhi	India
2	2	Amit	Hyderabad	India
3	3	Jason	California	USA
4	4	John	Texas	USA

5.1.4 SQL Delete without Commit

Post the DELETE command if we will not publish COMMIT, and if the session is closed then the change that is made due to the DELETE command will be lost.
Updated Command with COMMIT

```
1  DELETE from public."Customer" where "State"='Texas';
2  COMMIT;
```

Data Output | Explain | Messages | Query History

```
COMMIT

Query returned successfully in 92 msec.
```

5.1.5 SQL Commit Execution

Using the above-mentioned command sequence will ensure that the change post DELETE command will be saved successfully.

Output After Commit

CUSTOMER ID	CUSTOMER NAME	STATE	COUNTRY
1	Akash	Delhi	India
2	Amit	Hyderabad	India
3	Jason	California	USA

```
SELECT "Customer_Id", "Customer_Name", "State", "Country"
    FROM public."Customer";
```

Data Output | Explain | Messages | Query History

	Customer_Id bigint	Customer_Name character varying (50)	State character varying (20)	Country character varying (20)
1	1	Akash	Delhi	India
2	2	Amit	Hyderabad	India
3	3	Jason	California	USA

5.1.6 SQL Rollback

ROLLBACK is the SQL command that is used for reverting changes performed by a transaction. When a ROLLBACK command is issued it reverts all the changes since last COMMIT or ROLLBACK.
Syntax for SQL Rollback
ROLLBACK;
The syntax for rollback includes just one keyword ROLLBACK.

Assignment No 28

Customer: -

CUSTOMER ID	CUSTOMER NAME	STATE	COUNTRY
1	Akash	Delhi	India
2	Amit	Hyderabad	India
3	Jason	California	USA
4	John	Texas	USA

Now let us delete one row from the above table where State is “Texas”.
DELETE from Customer where State=’Texas’;

```
1  DELETE from public."Customer" where "State"='Texas'
```

Data Output | Explain | Messages | Query History

	Customer_Id bigint	Customer_Name character varying (50)	State character varying (20)	Country character varying (20)
1	1	Akash	Delhi	India
2	2	Amit	Hyderabad	India
3	3	Jason	California	USA
4	4	John	Texas	USA

Post the DELETE command if we publish ROLLBACK it will revert the change that is performed due to the delete command.

5.1.7 Updated Command with ROLLBACK

```
1  DELETE from public."Customer" where "State" = 'Texas';
2  ROLLBACK
```

Data Output | Explain | Messages | Query History

```
ROLLBACK

Query returned successfully in 99 msec.
```

5.1.8 SQL Delete with Rollback

Using the above-mentioned command sequence will ensure that the change post DELETE command will be reverted successfully.

Output After Rollback

CUSTOMER ID	CUSTOMER NAME	STATE	COUNTRY
1	Akash	Delhi	India
2	Amit	Hyderabad	India
3	Jason	California	USA
4	John	Texas	USA

```
SELECT "Customer_Id", "Customer_Name", "State", "Country"
    FROM public."Customer";
```

Data Output | Explain | Messages | Query History

	Customer_Id bigint	Customer_Name character varying (50)	State character varying (20)	Country character varying (20)
1	1	Akash	Delhi	India
2	2	Amit	Hyderabad	India
3	3	Jason	California	USA
4	4	John	Texas	USA

5.1.9 SQL Data Types

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

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

String data types:

Data type	Description	Max size	Storage
char(n)	Fixed width character string	8,000 characters	Defined width
varchar(n)	Variable width character string	8,000 characters	2 bytes + number of chars
varchar(max)	Variable width character string	1,073,741,824 characters	2 bytes + number of chars
text	Variable width character string	2GB of text data	4 bytes + number of chars
nchar	Fixed width Unicode string	4,000 characters	Defined width x 2
nvarchar	Variable width Unicode string	4,000 characters	
nvarchar(max)	Variable width Unicode string	536,870,912 characters	
ntext	Variable width Unicode string	2GB of text data	
binary(n)	Fixed width binary string	8,000 bytes	
varbinary	Variable width binary string	8,000 bytes	
varbinary(max)	Variable width binary string	2GB	
image	Variable width binary string	2GB	

Numeric data types:

Data type	Description	Storage
bit	Integer that can be 0, 1, or NULL	
tinyint	Allows whole numbers from 0 to 255	1 byte
smallint	Allows whole numbers between -32,768 and 32,767	2 bytes
int	Allows whole numbers between -2,147,483,648 and 2,147,483,647	4 bytes
bigint	Allows whole numbers between -9,223,372,036,854,775,808 and 9,223,372,036,854,775,807	8 bytes
decimal(p,s)	Fixed precision and scale numbers. Allows numbers from -10^38 +1 to 10^38 –1. The p parameter indicates the maximum total number of digits that can be stored (both to the left and to the right of the decimal point). p must be a value from 1 to 38. Default is 18. The s parameter indicates the maximum number of digits stored to the right of the decimal point. s must be a value from 0 to p. Default value is 0	5-17 bytes
numeric(p,s)	Fixed precision and scale numbers. Allows numbers from -10^38 +1 to 10^38 –1. The p parameter indicates the maximum total number of digits that can be stored (both to the left and to the right of the decimal point). p must be a value from 1 to 38. Default is 18. The s parameter indicates the maximum number of digits stored to the right of the decimal point. s must be a value from 0 to p. Default value is 0	5-17 bytes

Date and Time data types:

Data type	Description	Storage
datetime	From January 1, 1753 to December 31, 9999 with an accuracy of 3.33 milliseconds	8 bytes
datetime2	From January 1, 0001 to December 31, 9999 with an accuracy of 100 nanoseconds	6-8 bytes
smalldatetime	From January 1, 1900 to June 6, 2079 with an accuracy of 1 minute	4 bytes
date	Store a date only. From January 1, 0001 to December 31, 9999	3 bytes
time	Store a time only to an accuracy of 100 nanoseconds	3-5 bytes
datetimeoffset	The same as datetime2 with the addition of a time zone offset	8-10 bytes
timestamp	Stores a unique number that gets updated every time a row gets created or modified. The timestamp value is based upon an internal clock and does not correspond to real time. Each table may have only one timestamp variable	

Other data types:

Data type	Description
sql_variant	Stores up to 8,000 bytes of data of various data types, except text, ntext, and timestamp
uniqueidentifier	Stores a globally unique identifier (GUID)
xml	Stores XML formatted data. Maximum 2GB
cursor	Stores a reference to a cursor used for database operations
table	Stores a result-set for later processing

Summary

- **In this chapter, you have learned about:**
- **We studied How to use "use statement" in SQL**
- **Also, we studied commit and Rollback statement with update and delete**
- **Also, we studied Different Data types in SQL**

Fill in the blank

1.The ----------statement backs out, or cancels, the database changes that are made by the current transaction and restores changed data to the state before the transaction began.

2. A ---------- can be performed manually by a programmer or it can be triggered using an automated program

3. It's a ---------- of instructions in a logical order.

4. The ---------- is used to select a database and perform SQL operations into that database.

5. --------- is the SQL command that is used for storing changes performed by a transaction

Chapter 6

Operators in SQL

- What is operator in SQL?
- SQL Arithmetic Operators
- SQL Comparison Operators
- SQL Logical Operators
- Boolean Expressions
- Numeric Expressions
- Date Expressions
- Built-in Functions in SQL

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. These Operators are used to specify conditions in an SQL statement and to serve as conjunctions for multiple conditions in a statement.
Arithmetic operators
Comparison operators
Logical operators
Operators used to negate conditions

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

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.
Show Examples

Sr.No.	Operator & Description
1	ALL The ALL operator is used to compare a value to all values in another value set.
2	AND The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause.

3	ANY The ANY operator is used to compare a value to any applicable value in the list as per the condition.
4	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.
5	EXISTS The EXISTS operator is used to search for the presence of a row in a specified table that meets a certain criterion.
6	IN The IN operator is used to compare a value to a list of literal values that have been specified.
7	LIKE The LIKE operator is used to compare a value to similar values using wildcard operators.
8	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.
9	OR The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.
10	IS NULL The NULL operator is used to compare a value with a NULL value.
11	UNIQUE The UNIQUE operator searches every row of a specified table for uniqueness (no duplicates).

Assignment No 29

Example 1
SQL> select 10+ 20;
Output

```
+--------+
| 10+ 20 |
+--------+
```

```
|    30 |
+--------+
1 row in set (0.00 sec)
```

Example 2

```
SQL> select 10 * 20;
```

Output

```
+---------+
| 10 * 20 |
+---------+
|    200 |
+---------+
1 row in set (0.00 sec)
```

Example 3

```
SQL> select 10 / 5;
```

Output

```
+--------+
| 10 / 5 |
+--------+
| 2.0000 |
+--------+
1 row in set (0.03 sec)
```

Example 4

```
SQL> select 12 %  5;
```

Output

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

An expression is a combination of one or more values, operators and SQL functions that evaluate to a value. These SQL EXPRESSIONs are like formulae and they are written in query language. You can also use them to query the database for a specific set of data.

Syntax

Consider the basic syntax of the SELECT statement as follows −

```
SELECT column1, column2, columnN
FROM table_name
WHERE [CONDITION|EXPRESSION];
```

There are different types of SQL expressions, which are mentioned below −

Boolean

Numeric

Date

Boolean Expressions

SQL Boolean Expressions fetch the data based on matching a single value. Following is the syntax −
SELECT column1, column2, columnN
FROM table_name
WHERE SINGLE VALUE MATCHING EXPRESSION;
Consider the CUSTOMERS table having the following records

Assignment No 30

```
SQL> SELECT * FROM CUSTOMERS;
+----+----------+-----+-----------+----------+
| 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 |
+----+----------+-----+-----------+----------+
7 rows in set (0.00 sec)
```

SQL> SELECT * FROM CUSTOMERS WHERE SALARY = 10000;

Output

```
SQL> SELECT * FROM CUSTOMERS WHERE SALARY = 10000;
+----+-------+-----+---------+----------+
| ID | NAME  | AGE | ADDRESS | SALARY   |
+----+-------+-----+---------+----------+
|  7 | Muffy |  24 | Indore  | 10000.00 |
+----+-------+-----+---------+----------+
1 row in set (0.00 sec)
```

Numeric Expression
These expressions are used to perform any mathematical operation in any query. Following is the syntax −
SELECT numerical_expression as OPERATION_NAME
[FROM table_name
WHERE CONDITION] ;
Here, the numerical_expression is used for a mathematical expression or any formula. Following is a simple example showing the usage of SQL Numeric Expressions −

Assignment No 31

SQL> SELECT (15 + 6) AS ADDITION
Output
ADDITION |
+----------+

Aggregate Function	Description
Min ()	This will return Minimum value
Max()	This will return Maximum value
Avg ()	This will return Average value
Count ()	This will return number of counts
Sum ()	This will return total sum of numeric value

```
|      21 |
+----------+
1 row in set (0.00 sec)
```

Date Expressions

Date Expressions return current system date and time values –

Assignment No 32

```
SQL> SELECT CURRENT_TIMESTAMP;
Output
Current_Timestamp   |
+--------------------+
| 2009-11-12 06:40:23 |
+--------------------+
1 row in set (0.00 sec)
```

Built-In functions in Sql

SQL COUNT Function

The SQL COUNT aggregate function is used to count the number of rows in a database table. 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.

To understand COUNT function, consider an employee_tbl table, which is having the following records –

Aggregate Functions:

Aggregates the values and return a single value, below is the list of some aggregate values in sql server.

Assignment No 33

Example 1

```
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 |
+------+------+------------+--------------------+
7 rows in set (0.00 sec)
```

Now suppose based on the above table you want to count total number of rows in this table, then you can do it as follows –
SQL>SELECT COUNT (*) FROM employee_tbl;

Output
7
1 row in set (0.01 sec)

Example 2
SQL>SELECT COUNT(*) FROM employee_tbl
-> WHERE name="Zara";

Output
2
1 row in set (0.04 sec)
The SQL MIN () and MAX () Functions
The MIN () function returns the smallest value of the selected column.
The MAX () function returns the largest value of the selected column.
MIN () Syntax
SELECT MIN (*column_name*)
FROM *table_name*
WHERE *condition*;
MAX () Syntax
SELECT MAX(*column_name*)
FROM *table_name*
WHERE *condition*;

Demo Database

Below is a selection from the "Products" table in the Northwind sample database:

ProductID	ProductName	SupplierID	CategoryID	Unit	Price
1	Chais	1	1	10 boxes x 20 bags	18
2	Chang	1	1	24 - 12 oz bottles	19
3	Aniseed Syrup	1	2	12 - 550 ml bottles	10
4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars	22
5	Chef Anton's Gumbo Mix	2	2	36 boxes	21.35

Assignment No 34

MIN () Example

```
SELECT MIN(Price) AS SmallestPrice
FROM Products;
```

Result:

Number of Records: 1

SmallestPrice
2.5

MAX() Example
The following SQL statement finds the price of the most expensive product:

```
SELECT MAX(Price) AS LargestPrice
FROM Products;
```

Result:

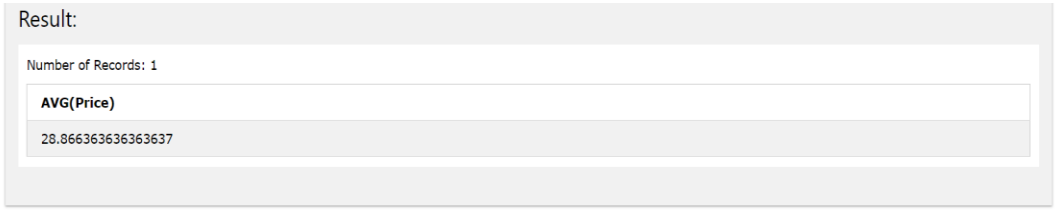
Number of Records: 1

LargestPrice
263.5

Assignment No35

The AVG () function returns the average value of a numeric column.
AVG () Syntax

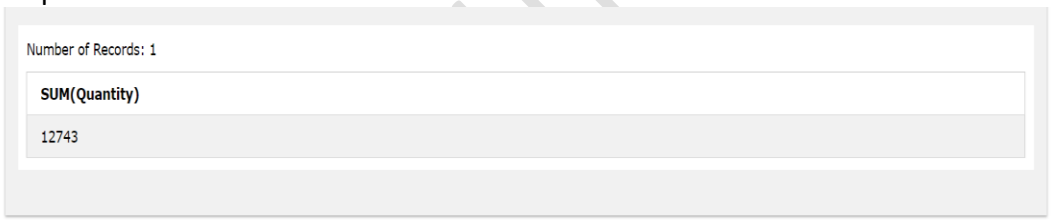
SELECT AVG(*column_name*)
FROM *table_name*
WHERE *condition*;
Example
SELECT AVG(Price)
FROM Products;


Result:

Number of Records: 1

AVG(Price)
28.866363636363637

SUM () function
The SUM () function returns the total sum of a numeric column.
SUM () Syntax
SELECT SUM (*column_name*)
FROM *table_name*
WHERE *condition*;
Example
SELECT SUM(Quantity)
FROM OrderDetails;
Output


Number of Records: 1

SUM(Quantity)
12743

Summary

- **In this chapter, you have learned about:**
- **We studied What is operator in SQL?**
- **Also, Studied SQL Arithmetic Operators**
- **We Also Studied Different SQL Comparison Operators**
- **We Also Studied Different SQL Logical Operators**
- **We Also Studied Different Boolean Expressions**
- **We Also Studied Different Numeric Expressions**
- **We Also Studied Different Date Expressions**
- **We Also Studied Different Built-in Functions in SQL**

MCQ's

1. Which operator is used to return value from JSON columns after evaluating the path and unquoting the result?
a) ->
b) ->>
c) <<
d) >>

2. The right shift operator is _____
a) >>
b) <<
c) <
d) >

3. Which operator compares sounds?
a) MATCH SOUNDS
b) CHECK SOUNDS
c) SOUNDS LIKE
d) SOUNDS SIMILAR

4. MySQL automatically converts a date or time value to a number if the value is used in a numeric context.
a) True
b) False

5. @@NESTLEVEL function falls under which of the following category?
a) Configuration functions
b) Cursor functions
c) Mathematical functions
d) Date and Time Data Functions

6. DecryptByKeyAutoCert is _____ type function.
a) Symmetric Encryption and decryption
b) Encryption Hashing
c) Asymmetric Encryption and decryption
d) Symmetric decryption with Automatic key handling

7. Which of the the following is not conversion function?
a) CAST and CONVERT
b) PARSE
c) TRY_CAST
d) TRY_CASE

8. Built in Functions in SQL Server is categorized in to how many category?
a) 4
b) 5
c) 6
d) 7

9. What is the purpose of MIN function in SQL Server?
a) It returns the minimum value in the expression
b) It is use for decrementing the integer value
c) MIN is not a SQL server function
d) None of the above

10. Which function in SQL server is used in query to find the count of the items in the concerned object (i.e. table, view, etc.)?
a) COUNTER
b) COUNT
c) DETECT
d) None of the above

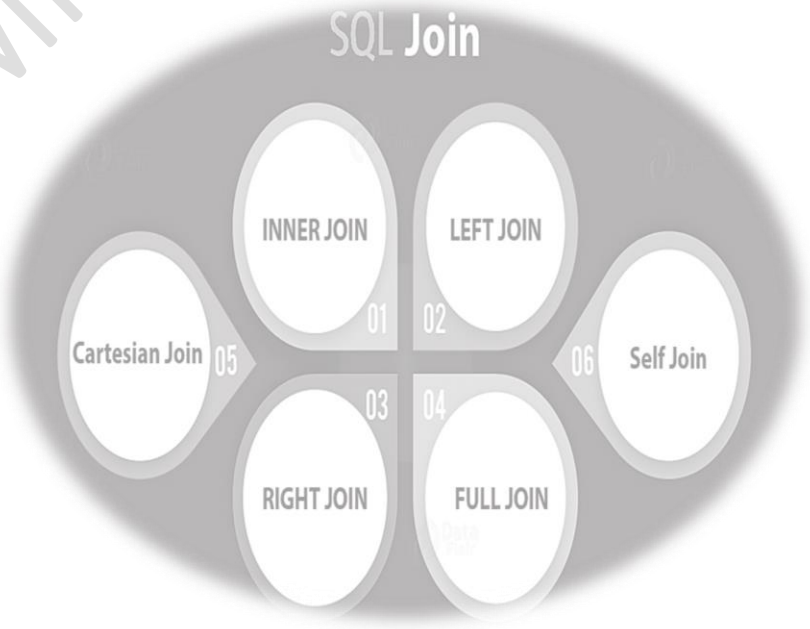
Fill in the blank

1.The ---------- function returns the smallest value of the selected column.
2.The ---------- is used to compare a value to similar values using wildcard operators
3.The SUM () function returns the ---------- column.
4.The ---------- is used to compare a value with a NULL value.
5. The ---------- operator is used to search for values that are within a set of values, given the minimum value and the maximum value

Chapter 7

Joins in SQL

- SQL Join
- Different Types of joins
- SQL Self join
- SQL Index
- Cluster-NonCluster indexes

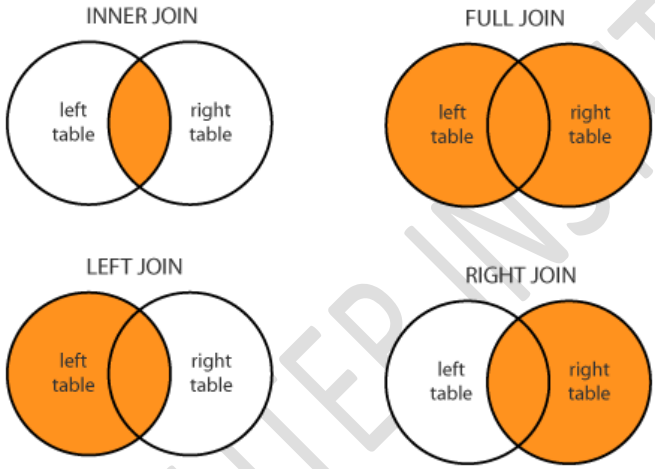


7.1 SQL JOIN

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

Different types of JOINs

(INNER) JOIN: Select records that have matching values in both tables.
LEFT (OUTER) JOIN: Select records from the first (left-most) table with matching right table records.
RIGHT (OUTER) JOIN: Select records from the second (right-most) table with matching left table records.
FULL (OUTER) JOIN: Selects all records that match either left or right table records.



7.1.1 SQL INNER JOIN Keyword

The INNER JOIN keyword selects records that have matching values in both tables.
INNER JOIN Syntax
SELECT *column_name(s)*
FROM *table1*
INNER JOIN *table2*
ON *table1.column_name* = *table2.column_name*;

Assignment No 36

Example on Inner Join

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10308	2	7	1996-09-18	3
10309	37	3	1996-09-19	1
10310	77	8	1996-09-20	2

And a selection from the "Customers" table:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

SELECT Orders.OrderID, Customers.CustomerName
FROM Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;

Output

Number of Records: 196

OrderID	CustomerName
10248	Wilman Kala
10249	Tradição Hipermercados
10250	Hanari Carnes
10251	Victuailles en stock
10252	Suprêmes délices
10253	Hanari Carnes
10254	Chop-suey Chinese
10255	Richter Supermarkt
10256	Wellington Importadora
10257	HILARIÓN-Abastos
10258	Ernst Handel
10259	Centro comercial Moctezuma
10260	Old World Delicatessen
10261	Que Delícia
10262	Rattlesnake Canyon Grocery
10263	Ernst Handel

7.1.2 SQL LEFT JOIN Keyword

The LEFT JOIN keyword returns all records from the left table (table1), and the matched records from the right table (table2). The result is NULL from the right side, if there is no match.

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

Note: In some databases LEFT JOIN is called LEFT OUTER JOIN.

Assignment No 37

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

And a selection from the "Orders" table:

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10308	2	7	1996-09-18	3
10309	37	3	1996-09-19	1
10310	77	8	1996-09-20	2

SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
LEFT JOIN Orders
ON Customers.CustomerID=Orders.CustomerID
ORDER BY Customers.CustomerName;

Output

Number of Records: 273

CustomerName	OrderID
Ana Trujillo Emparedados y helados	10308
Antonio Moreno Taquería	10365
Around the Horn	10355
Around the Horn	10383
Aux joyeux ecclésiastiques	*null*
Aux joyeux ecclésiastiques	*null*
B's Beverages	10289
Berglunds snabbköp	10278
Berglunds snabbköp	10280
Berglunds snabbköp	10384
Bigfoot Breweries	*null*
Bigfoot Breweries	*null*
Blauer See Delikatessen	*null*
Blondel père et fils	10265
Blondel père et fils	10297
Blondel père et fils	10360

7.1.3 SQL RIGHT JOIN Keyword

The RIGHT JOIN keyword returns all records from the right table (table2), and the matched records from the left table (table1). The result is NULL from the left side, when there is no match.

RIGHT JOIN Syntax

SELECT *column_name(s)*
FROM *table1*
RIGHT JOIN *table2*
ON *table1.column_name* = *table2.column_name*;
Note: In some databases RIGHT JOIN is called RIGHT OUTER JOIN.

Assignment No 38

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10308	2	7	1996-09-18	3
10309	37	3	1996-09-19	1
10310	77	8	1996-09-20	2

And a selection from the "Employees" table:

EmployeeID	LastName	FirstName	BirthDate	Photo
1	Davolio	Nancy	12/8/1968	EmpID1.pic
2	Fuller	Andrew	2/19/1952	EmpID2.pic
3	Leverling	Janet	8/30/1963	EmpID3.pic

SELECT Orders.OrderID, Employees.LastName, Employees.FirstName
FROM Orders
RIGHT JOIN Employees
ON Orders.EmployeeID = Employees.EmployeeID
ORDER BY Orders.OrderID;

Output

Number of Records: 197

OrderID	LastName	FirstName
	West	Adam
10248	Buchanan	Steven
10249	Suyama	Michael
10250	Peacock	Margaret
10251	Leverling	Janet
10252	Peacock	Margaret
10253	Leverling	Janet
10254	Buchanan	Steven

7.1.4 SQL FULL OUTER JOIN Keyword

The FULL OUTER JOIN keyword returns all records when there is a match in either left (table1) or right (table2) table records.

Note: FULL OUTER JOIN can potentially return very large result-sets!
Tip: FULL OUTER JOIN and FULL JOIN are the same.

FULL OUTER JOIN Syntax

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

Assignment No 39

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

And a selection from the "Orders" table:

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10308	2	7	1996-09-18	3
10309	37	3	1996-09-19	1
10310	77	8	1996-09-20	2

SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
FULL OUTER JOIN Orders ON Customers.CustomerID=Orders.CustomerID
ORDER BY Customers.CustomerName;

Output

CustomerName	OrderID
Alfreds Futterkiste	
Ana Trujillo Emparedados y helados	10308
Antonio Moreno Taquería	10365
	10382
	10351

Note: The FULL OUTER JOIN keyword returns all matching records from both tables whether the other table matches or not. So, if there are rows in "Customers" that do not have matches in "Orders", or if there are rows in "Orders" that do not have matches in "Customers", those rows will be listed as well.

7.1.5 SQL Self JOIN

A self-JOIN is a regular join, but the table is joined with itself.

Self-JOIN Syntax

SELECT *column_name(s)*
FROM *table1 T1, table1 T2*
WHERE *condition*;
T1 and *T2* are different table aliases for the same table.

Assignment No 40

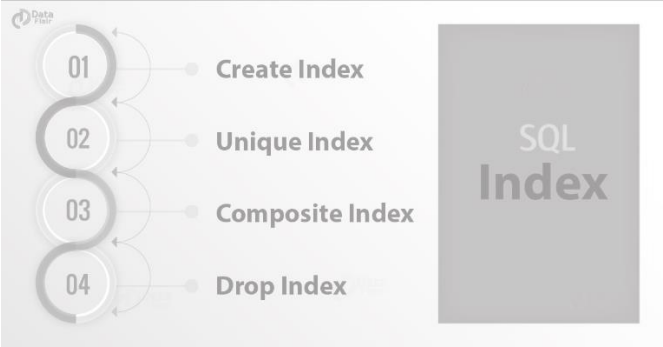
CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

SELECT A. CustomerName AS CustomerName1, B. CustomerName AS CustomerName2, A. City
FROM Customers A, Customers B
WHERE A. CustomerID <> B. CustomerID
AND A. City = B. City
ORDER BY A. City;

Output

CustomerName1	CustomerName2	City
Grandma Kelly's Homestead	Grandma Kelly's Homestead	Ann Arbor
Grandma Kelly's Homestead	Grandma Kelly's Homestead	Ann Arbor
Gai pâturage	Gai pâturage	Annecy
Gai pâturage	Gai pâturage	Annecy
Bigfoot Breweries	Bigfoot Breweries	Bend
Bigfoot Breweries	Bigfoot Breweries	Bend
Heli Süßwaren GmbH & Co. KG	Heli Süßwaren GmbH & Co. KG	Berlin
Heli Süßwaren GmbH & Co. KG	Heli Süßwaren GmbH & Co. KG	Berlin
Heli Süßwaren GmbH & Co. KG	Heli Süßwaren GmbH & Co. KG	Berlin
Heli Süßwaren GmbH & Co. KG	Heli Süßwaren GmbH & Co. KG	Berlin
Heli Süßwaren GmbH & Co. KG	Heli Süßwaren GmbH & Co. KG	Berlin
Heli Süßwaren GmbH & Co. KG	Heli Süßwaren GmbH & Co. KG	Berlin
New England Seafood Cannery	New England Seafood Cannery	Boston
New England Seafood Cannery	New England Seafood Cannery	Boston
Cactus Comidas para llevar	Océano Atlántico Ltda.	Buenos Aires
Cactus Comidas para llevar	Rancho grande	Buenos Aires

7.1.6 SQL Index



7.1.7 SQL CREATE INDEX Statement

The CREATE INDEX statement is used to create indexes in tables.
Indexes are used to retrieve data from the database very fast. The users cannot see the indexes, they are just used to speed up searches/queries.

Note: Updating a table with indexes takes more time than updating a table without (because the indexes also need an update). So, only create indexes on columns that will be frequently searched against.

CREATE INDEX Syntax
Creates an index on a table. Duplicate values are allowed:
CREATE INDEX *index_name*
ON *table_name* (*column1*, *column2*, ...);

Assignment No 41

```
CREATE TABLE Persons (
   PersonID int,
   LastName varchar (255),
   FirstName varchar (255),
   Address varchar (255),
   City varchar (255)
);
```

CREATE INDEX Example
The SQL statement below creates an index named "idx_lastname" on the "LastName" column in the "Persons" table:

```
CREATE INDEX idx_lastname
ON Persons (LastName);
```

If you want to create an index on a combination of columns, you can list the column names within the parentheses, separated by commas:

CREATE INDEX idx_pname
ON Persons (LastName, FirstName);

DROP INDEX Statement
The DROP INDEX statement is used to delete an index in a table.

Assignment No 42

DROP INDEX *table_name.index_name*;
CREATE UNIQUE INDEX
The CREATE UNIQUE INDEX command creates a unique index on a table (no duplicate values allowed)
Indexes are used to retrieve data from the database very fast. The users cannot see the indexes, they are just used to speed up searches/queries.

The following SQL creates an index named "uidx_pid" on the "PersonID" column in the "Persons" table:

CREATE UNIQUE INDEX uidx_pid
ON Persons (PersonID);

Note: The syntax for creating indexes varies among different databases. Therefore: Check the syntax for creating indexes in your database.

Note**:** Updating a table with indexes takes more time than updating a table without (because the indexes also need an update). So, only create indexes on columns that will be frequently searched against.

7.1.8 Composite SQL Server Index

A composite Index is an Index on two or more columns of a table.
Its basic syntax is as follows.

CREATE INDEX index_name
on **table_name** (column1, column2);

7.1.9 Clustered and Non-Clustered Indexes

1.Clustered
Clustered index is the type of indexing that established a physical shorting order of rows.
Suppose you have a table *Student_info* which contains *ROLL_NO* as a primary key than Clustered index which is self-created on that primary key will short the *Student_info* table as per *ROLL_NO*.
Clustered index is like Dictionary, in the dictionary shorting order is alphabetical there is no separate index page.

Assignment No 43

CREATE TABLE Student_info
(

```
ROLL_NO int(10) primary key,
NAME varchar(20),
DEPARTMENT varchar(20),
);
insert into Student_info values(1410110405, 'H Agarwal', 'CSE')
insert into Student_info values (1410110404, 'S Samadder', 'CSE')
insert into Student_info values (1410110403, 'MD Irfan', 'CSE')

SELECT * FROM Student_info
```

Output:

ROLL_NO	NAME	DEPARTMENT
1410110403	MD Irfan	CSE
1410110404	S Samadder	CSE
1410110405	H Agarwal	CSE

2. Non-clustered:

The Non-Clustered index is an index structure separate from the data stored in a table that reorders one or more selected columns.

The non-clustered index is created to improve the performance of frequently used queries not covered by clustered index.

It's like a textbook, the index page is created separately at the beginning of that book.
Syntax:

```
//Create Non-Clustered index
create NonClustered index IX_table_name_column_name
    on table_name (column_name ASC)
Assignment No 44
CREATE TABLE Student_info
(
ROLL_NO int(10),
NAME varchar(20),
DEPARTMENT varchar(20),
);
insert into Student_info values(1410110405, 'H Agarwal', 'CSE')
insert into Student_info values(1410110404, 'S Samadder', 'CSE')
insert into Student_info values(1410110403, 'MD Irfan', 'CSE')

SELECT * FROM Student_info
```

Output:

ROLL_NO	NAME	DEPARTMENT
1410110405	H Agarwal	CSE
1410110404	S Samadder	CSE
1410110403	MD Irfan	CSE

Note: We can create one or more Non_Clustered index in a table.

Clustered vs Non-Clustered index:

In a table there can be only one clustered index or one or more tha one non_clustered index.
In Clustered index there is no separate index storage but in Non_Clustered index there is separate index storage for the index.
Clustered index is slower than Non_Clustered index.

SQL Aliases

SQL aliases are used to give a table, or a column in a table, a temporary name.
Aliases are often used to make column names more readable.
An alias only exists for the duration of the query.

Alias Column Syntax
SELECT *column_name* AS *alias_name*
FROM *table_name;*

Alias Table Syntax
SELECT *column_name(s)*
FROM *table_name* AS *alias_name;*

Assignment No 45

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK

And a selection from the "Orders" table:

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10354	58	8	1996-11-14	3
10355	4	6	1996-11-15	1
10356	86	6	1996-11-18	2

SELECT CustomerName AS Customer, ContactName AS [Contact Person]

FROM Customers;

Output

Number of Records: 151

Customer	Contact Person
Ana Trujillo Emparedados y helados	Ana Trujillo
Antonio Moreno Taquería	Antonio Moreno
Around the Horn	Thomas Hardy
Berglunds snabbköp	Christina Berglund
Blauer See Delikatessen	Hanna Moos
Blondel père et fils	Frédérique Citeaux
Bólido Comidas preparadas	Martín Sommer
Bon app'	Laurence Lebihans
Bottom-Dollar Marketse	Elizabeth Lincoln
B's Beverages	Victoria Ashworth
Cactus Comidas para llevar	Patricio Simpson
Centro comercial Moctezuma	Francisco Chang
Chop-suey Chinese	Yang Wang
Comércio Mineiro	Pedro Afonso
Consolidated Holdings	Elizabeth Brown
Drachenblut Delikatessend	Sven Ottlieb

iting for cdn.bannerflow.com...

Assignment No 46

The following SQL statement selects all the orders from the customer with CustomerID=4 (Around the Horn). We use the "Customers" and "Orders" tables, and give them the table aliases of "c" and "o" respectively (Here we use aliases to make the SQL shorter):

```
SELECT o.OrderID, o.OrderDate, c.CustomerName
FROM Customers AS c, Orders AS o
WHERE c. CustomerName="Around the Horn" AND c. CustomerID=o.CustomerID;
```

Output

Number of Records: 2

OrderID	OrderDate	CustomerName
10355	1996-11-15	Around the Horn
10383	1996-12-16	Around the Horn

Summary

- **In this chapter, you have learned about:**
- **We have studied, SQL Joins.**
- **We have studied all join types syntax and example**
- **Also studied SQL index**
- **Cluster and NonCluster index**

MCQ's

1. What type of join is needed when you wish to include rows that do not have matching values?
a) Equi-join
b) Natural join
c) Outer join
d) All of the Mentioned

2. What type of join is needed when you wish to return rows that do have matching values?
a) Equi-join
b) Natural join
c) Outer join
d) All of the Mentioned

3. Which of the following is one of the basic approaches for joining tables?
a) Subqueries
b) Union Join
c) Natural join
d) All of the Mentioned

4. The following SQL is which type of join:
SELECT CUSTOMER_T. CUSTOMER_ID, ORDER_T. CUSTOMER_ID, NAME, ORDER_ID FROM CUSTOMER_T,ORDER_T WHERE CUSTOMER_T. CUSTOMER_ID = ORDER_T. CUSTOMER_ID?
a) Equi-join
b) Natural join
c) Outer join
d) Cartesian join

5. A UNION query is which of the following?
a) Combines the output from no more than two queries and must include the same number of columns
b) Combines the output from no more than two queries and does not include the same number of columns
c) Combines the output from multiple queries and must include the same number of columns
d) Combines the output from multiple queries and does not include the same number of columns

6. Which of the following statements is true concerning subqueries?
a) Involves the use of an inner and outer query
b) Cannot return the same result as a query that is not a subquery
c) Does not start with the word SELECT
d) All of the mentioned

7. Which of the following is a correlated subquery?
a) Uses the result of an inner query to determine the processing of an outer query
b) Uses the result of an outer query to determine the processing of an inner query
c) Uses the result of an inner query to determine the processing of an inner query
d) Uses the result of an outer query to determine the processing of an outer query

8. How many tables may be included with a join?
a) One
b) Two
c) Three
d) All of the Mentioned

9. The following SQL is which type of join:
SELECT CUSTOMER_T. CUSTOMER_ID, ORDER_T. CUSTOMER_ID, NAME, ORDER_ID FROM CUSTOMER_T,ORDER_T?
a) Equi-join
b) Natural join
c) Outer join
d) Cartesian join

10. Which is not a type of join in T-SQL?
a) Equi-join
b) Natural join
c) Outer join
d) Cartesian join

Fill in the blank

1. A NATURAL JOIN is ---------- where the RDBMS automatically selects the join columns based on common columns names
2. The ---------- statement is used to delete rows in a table.
3. An ---------- is a specific type of comparator-based join, that uses only equality comparisons in the join-predicate.
4. A Cartesian coordinate system is a coordinate system that specifies each point uniquely in a plane by a pair of ----------.
5. A JOIN is a means for ---------- by using values common to each.

Chapter 8

Date and Time Functions in SQL

- SQL Date Functions
- ADDDATE (expr, days)
- ADDDATE (date, INTERVAL expr unit)
- Convert _TZ (dt, from_tz, to_tz)
- CURDATE ()
- CUURENT_DATE ()
- CURRENT_TIMESTAMP ()
- DAYOFMONTH(date)

8.1 Date and Time Functions in SQL

In SQL, dates are complicated for newbies, since while working with database, the format of the date in table must be matched with the input date in order to insert. In various scenarios instead of date, datetime (time is also involved with date) is used.

1)CURRENT_TIMESTAMP

The CURRENT_TIMESTAMP function returns the current timestamp of the operating system of the server on which the SQL Server Database runs. The returned timestamp is a DATETIME value without the time zone offset.
The CURRENT_TIMESTAMP is the ANSI SQL equivalent to GETDATE ().
You can use the CURRENT_TIMESTAMP function anywhere a DATETIME expression is accepted.
SQL Server CURRENT_TIMESTAMP function example.

Let's take some example of using the CURRENT_TIMESTAMP function.

Assignment No 47

A) SELECT
CURRENT_TIMESTAMP AS current_date_time;
Output
current_date_time

2019-02-23 20:02:21.550

(1 row affected)
B) Using **CURRENT_TIMESTAMP** function as a default value for table columns example
First, create a new table named current_timestamp_demos whose created_at column accepts a default value as the timestamp at which a row is inserted:

```
CREATE TABLE current_timestamp_demos
(
    id         INT IDENTITY,
    msg        VARCHAR(255) NOT NULL,
    created_at DATETIME NOT NULL
               DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY(id)
);
```

Second, insert two rows into the table
INSERT INTO current_timestamp_demos(msg)
VALUES('This is the first message.');

INSERT INTO current_timestamp_demos(msg)
VALUES('current_timestamp demo');
Third, query data from the current_timestamp_demos table:

```
SELECT
    id,
    msg,
    created_at
FROM
    current_timestamp_demos;
```

Output

id	msg	created_at
1	This is the first message.	2019-05-03 21:43:23.523
2	current_timestamp demo	2019-05-03 21:43:23.523

2)ADDDATE (date, INTERVAL expr unit), ADDDATE (expr, days)

When invoked with the INTERVAL form of the second argument, ADDDATE () is a synonym for DATE_ADD (). The related function SUBDATE () is a synonym for DATE_SUB (). For information on the INTERVAL unit argument, see the discussion for DATE_ADD ().

Assignment No 48

```
mysql> SELECT DATE_ADD('1998-01-02', INTERVAL 31 DAY);
+------------------------------------------------+
| DATE_ADD('1998-01-02', INTERVAL 31 DAY)        |
+------------------------------------------------+
| 1998-02-02                                     |
+------------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT ADDDATE('1998-01-02', INTERVAL 31 DAY);
+------------------------------------------------+
| ADDDATE('1998-01-02', INTERVAL 31 DAY)         |
+------------------------------------------------+
| 1998-02-02                                     |
+------------------------------------------------+
1 row in set (0.00 sec)
```

When invoked with the days form of the second argument, MySQL treats it as an integer number of days to be added to expr.

```
mysql> SELECT ADDDATE('1998-01-02', 31);
+------------------------------------------------+
| DATE_ADD('1998-01-02', INTERVAL 31 DAY)        |
+------------------------------------------------+
| 1998-02-02                                     |
```

```
+------------------------------------------------------+
1 row in set (0.00 sec)
```

3) ADDTIME (expr1, expr2)

ADDTIME () adds expr2 to expr1 and returns the result. The expr1 is a time or datetime expression, while the expr2 is a time expression.

Assignment No 49

```
mysql> SELECT ADDTIME('1997-12-31 23:59:59.999999','1 1:1:1.000002');
+------------------------------------------------------+
| DATE_ADD('1997-12-31 23:59:59.999999','1 1:1:1.000002') |
+------------------------------------------------------+
| 1998-01-02 01:01:01.000001                           |
+------------------------------------------------------+
1 row in set (0.00 sec)
```

CONVERT_TZ(dt,from_tz,to_tz)

This converts a datetime value dt from the time zone given by from_tz to the time zone given by to_tz and returns the resulting value. This function returns NULL if the arguments are invalid.

```
mysql> SELECT CONVERT_TZ('2004-01-01 12:00:00','GMT','MET');
+------------------------------------------------------+
| CONVERT_TZ('2004-01-01 12:00:00','GMT','MET')        |
+------------------------------------------------------+
| 2004-01-01 13:00:00                                  |
+------------------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT CONVERT_TZ('2004-01-01 12:00:00','+00:00','+10:00');
+------------------------------------------------------+
| CONVERT_TZ('2004-01-01 12:00:00','+00:00','+10:00')  |
+------------------------------------------------------+
| 2004-01-01 22:00:00                                  |
+------------------------------------------------------+
1 row in set (0.00 sec)
```

4) CURDATE()

Returns the current date as a value in 'YYYY-MM-DD' or YYYYMMDD format, depending on whether the function is used in a string or in a numeric context.

Assignment No 50

```
mysql> SELECT CURDATE();
+------------------------------------------------------+
| CURDATE()                                            |
+------------------------------------------------------+
| 1997-12-15                                           |
```

```
+-----------------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT CURDATE() + 0;
+-----------------------------------------------------+
| CURDATE() + 0                                       |
+-----------------------------------------------------+
| 19971215                                            |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

5) CURRENT_DATE and CURRENT_DATE ()

CURRENT_DATE and CURRENT_DATE () are synonyms for CURDATE ()
CURTIME ()
Returns the current time as a value in 'HH:MM:SS' or HHMMSS format, depending on whether the function is used in a string or in a numeric context. The value is expressed in the current time zone.

Assignment No 51

```
mysql> SELECT CURTIME();
+-----------------------------------------------------+
| CURTIME()                                           |
+-----------------------------------------------------+
| 23:50:26                                            |
+-----------------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT CURTIME() + 0;
+-----------------------------------------------------+
| CURTIME() + 0                                       |
+-----------------------------------------------------+
| 235026                                              |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

CURRENT_TIME and CURRENT_TIME ()

CURRENT_TIME and CURRENT_TIME () are synonyms for CURTIME ().
CURRENT_TIMESTAMP and CURRENT_TIMESTAMP ()
CURRENT_TIMESTAMP and CURRENT_TIMESTAMP () are synonyms for NOW ().

1.DATE (expr)

Extracts the date part of the date or datetime expression expr.

```
mysql> SELECT DATE('2003-12-31 01:02:03');
+-----------------------------------------------------+
```

```
| DATE('2003-12-31 01:02:03')                    |
+------------------------------------------------+
|  2003-12-31                                    |
+------------------------------------------------+
1 row in set (0.00 sec)
```

2. DATEDIFF(expr1,expr2)

DATEDIFF() returns expr1 . expr2 expressed as a value in days from one date to the other. Both expr1 and expr2 are date or date-and-time expressions. Only the date parts of the values are used in the calculation.

```
mysql> SELECT DATEDIFF('1997-12-31 23:59:59','1997-12-30');
+------------------------------------------------+
| DATEDIFF('1997-12-31 23:59:59','1997-12-30')   |
+------------------------------------------------+
| 1                                              |
+------------------------------------------------+
1 row in set (0.00 sec)
```

3.DATE_ADD (date, INTERVAL expr unit), DATE_SUB (date, INTERVAL expr unit)

These functions perform date arithmetic. The date is a DATETIME or DATE value specifying the starting date. The expr is an expression specifying the interval value to be added or subtracted from the starting date. The expr is a string; it may start with a '-' for negative intervals.

A unit is a keyword indicating the units in which the expression should be interpreted.
The INTERVAL keyword and the unit specifier are not case sensitive.
The following table shows the expected form of the expr argument for each unit value.

unit Value	Expected exprFormat
MICROSECOND	MICROSECONDS
SECOND	SECONDS
MINUTE	MINUTES
HOUR	HOURS
DAY	DAYS
WEEK	WEEKS
MONTH	MONTHS
QUARTER	QUARTERS
YEAR	YEARS
SECOND_MICROSECOND	'SECONDS.MICROSECONDS'
MINUTE_MICROSECOND	'MINUTES.MICROSECONDS'
MINUTE_SECOND	'MINUTES:SECONDS'
HOUR_MICROSECOND	'HOURS.MICROSECONDS'
HOUR_SECOND	'HOURS:MINUTES:SECONDS'
HOUR_MINUTE	'HOURS:MINUTES'

DAY_MICROSECOND	**'DAYS.MICROSECONDS'**
DAY_SECOND	**'DAYS HOURS:MINUTES:SECONDS'**
DAY_MINUTE	**'DAYS HOURS:MINUTES'**
DAY_HOUR	**'DAYS HOURS'**
YEAR_MONTH	**'YEARS-MONTHS'**

Assignment No 52

The values QUARTER and WEEK are available from the MySQL 5.0.0. version.

```
mysql> SELECT DATE_ADD('1997-12-31 23:59:59',
  -> INTERVAL '1:1' MINUTE_SECOND);
+-----------------------------------------------------+
| DATE_ADD('1997-12-31 23:59:59', INTERVAL...         |
+-----------------------------------------------------+
| 1998-01-01 00:01:00                                 |
+-----------------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT DATE_ADD('1999-01-01', INTERVAL 1 HOUR);
+-----------------------------------------------------+
| DATE_ADD('1999-01-01', INTERVAL 1 HOUR)             |
+-----------------------------------------------------+
| 1999-01-01 01:00:00                                 |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

1. DATE_SUB(date,INTERVAL expr unit)

This is similar to the DATE_ADD() function.

2. DAY(date)

The DAY() is a synonym for the DAYOFMONTH() function.

3. DAYNAME(date)

Returns the name of the weekday for date.

```
mysql> SELECT DAYNAME('1998-02-05');
+-----------------------------------------------------+
| DAYNAME('1998-02-05')                               |
+-----------------------------------------------------+
| Thursday                                            |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

4. DAYOFMONTH(date)

Returns the day of the month for date, in the range 0 to 31.

```
mysql> SELECT DAYOFMONTH('1998-02-03');
+-----------------------------------------------------+
| DAYOFMONTH('1998-02-03')                            |
+-----------------------------------------------------+
| 3                                                   |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

5. DAYOFWEEK(date)

Returns the weekday index for date (1 = Sunday, 2 = Monday, ., 7 = Saturday). These index values correspond to the ODBC standard.

```
mysql> SELECT DAYOFWEEK('1998-02-03');
+-----------------------------------------------------+
|DAYOFWEEK('1998-02-03')                              |
+-----------------------------------------------------+
| 3                                                   |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

6. DAYOFYEAR(date)

Returns the day of the year for date, in the range 1 to 366.

```
mysql> SELECT DAYOFYEAR('1998-02-03');
+-----------------------------------------------------+
| DAYOFYEAR('1998-02-03')                             |
+-----------------------------------------------------+
| 34                                                  |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

7. EXTRACT(unit FROM date)

The EXTRACT() function uses the same kinds of unit specifiers as DATE_ADD() or DATE_SUB(), but extracts parts from the date rather than performing date arithmetic.

```
mysql> SELECT EXTRACT(YEAR FROM '1999-07-02');
+-----------------------------------------------------+
| EXTRACT(YEAR FROM '1999-07-02')                     |
+-----------------------------------------------------+
| 1999                                                |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT EXTRACT(YEAR_MONTH FROM '1999-07-02 01:02:03');
+-----------------------------------------------------+
| EXTRACT(YEAR_MONTH FROM '1999-07-02 01:02:03')      |
+-----------------------------------------------------+
| 199907                                              |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

8. FROM_DAYS(N)

Given a day number N, returns a DATE value.

```
mysql> SELECT FROM_DAYS(729669);
+-----------------------------------------------------+
| FROM_DAYS(729669)                                   |
+-----------------------------------------------------+
| 1997-10-07                                          |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

Note − Use FROM_DAYS() with caution on old dates. It is not intended for use with values that precede the advent of the Gregorian calendar (1582)

9. (A) FROM_UNIXTIME(unix_timestamp)

9. (B) FROM_UNIXTIME(unix_timestamp,format)

Returns a representation of the unix_timestamp argument as a value in 'YYYY-MM-DD HH:MM:SS or YYYYMMDDHHMMSS format, depending on whether the function is used in a string or in a numeric context. The value is expressed in the current time zone. The unix_timestamp argument is an internal timestamp values, which are produced by the UNIX_TIMESTAMP()function.
If the format is given, the result is formatted according to the format string, which is used in the same way as is listed in the entry for the DATE_FORMAT() function.

```
mysql> SELECT FROM_UNIXTIME(875996580);
+-----------------------------------------------------+
| FROM_UNIXTIME(875996580)                            |
+-----------------------------------------------------+
| 1997-10-04 22:23:00                                 |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

10. HOUR(time)

Returns the hour for time. The range of the return value is 0 to 23 for time-of-day values. However, the range of TIME values actually is much larger, so HOUR can return values greater than 23.

```
mysql> SELECT HOUR('10:05:03');
+-----------------------------------------------------+
```

```
| HOUR('10:05:03')                                    |
+-----------------------------------------------------+
| 10                                                  |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

11. LAST_DAY(date)

Takes a date or datetime value and returns the corresponding value for the last day of the month. Returns NULL if the argument is invalid.

```
mysql> SELECT LAST_DAY('2003-02-05');
+-----------------------------------------------------+
| LAST_DAY('2003-02-05')                              |
+-----------------------------------------------------+
| 2003-02-28                                          |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

LOCALTIME and LOCALTIME()

LOCALTIME and LOCALTIME() are synonyms for NOW().

LOCALTIMESTAMP and LOCALTIMESTAMP()

LOCALTIMESTAMP and LOCALTIMESTAMP() are synonyms for NOW().

12. MAKEDATE(year,dayofyear)

Returns a date, given year and day-of-year values. The dayofyear value must be greater than 0 or the result will be NULL.

```
mysql> SELECT MAKEDATE(2001,31), MAKEDATE(2001,32);
+-----------------------------------------------------+
| MAKEDATE(2001,31), MAKEDATE(2001,32)                |
+-----------------------------------------------------+
| '2001-01-31', '2001-02-01'                          |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

13. MAKETIME(hour,minute,second)

Returns a time value calculated from the hour, minute and second arguments.

```
mysql> SELECT MAKETIME(12,15,30);
+-----------------------------------------------------+
| MAKETIME(12,15,30)                                  |
+-----------------------------------------------------+
| '12:15:30'                                          |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

14. MICROSECOND(expr)

Returns the microseconds from the time or datetime expression (expr) as a number in the range from 0 to 999999.

```
mysql> SELECT MICROSECOND('12:00:00.123456');
+-----------------------------------------------------+
| MICROSECOND('12:00:00.123456')                      |
+-----------------------------------------------------+
| 123456                                              |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

15. MINUTE(time)

Returns the minute for time, in the range 0 to 59.

```
mysql> SELECT MINUTE('98-02-03 10:05:03');
+-----------------------------------------------------+
| MINUTE('98-02-03 10:05:03')                         |
+-----------------------------------------------------+
| 5                                                   |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

16. MONTH(date)

Returns the month for date, in the range 0 to 12.

```
mysql> SELECT MONTH('1998-02-03')
+-----------------------------------------------------+
| MONTH('1998-02-03')                                 |
+-----------------------------------------------------+
| 2                                                   |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

17. MONTHNAME(date)

Returns the full name of the month for a date.

```
mysql> SELECT MONTHNAME('1998-02-05');
+-----------------------------------------------------+
| MONTHNAME('1998-02-05')                             |
+-----------------------------------------------------+
| February                                            |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

18. NOW()

Returns the current date and time as a value in 'YYYY-MM-DD HH:MM:SS' or YYYYMMDDHHMMSS format, depending on whether the function is used in a string or numeric context. This value is expressed in the current time zone.

```
mysql> SELECT NOW();
+---------------------------------------------------+
| NOW()                                             |
+---------------------------------------------------+
| 1997-12-15 23:50:26                               |
+---------------------------------------------------+
1 row in set (0.00 sec)
```

19. PERIOD_ADD(P,N)

Adds N months to a period P (in the format YYMM or YYYYMM). Returns a value in the format YYYYMM. Note that the period argument P is not a date value.

```
mysql> SELECT PERIOD_ADD(9801,2);
+---------------------------------------------------+
| PERIOD_ADD(9801,2)                                |
+---------------------------------------------------+
| 199803                                            |
+---------------------------------------------------+
1 row in set (0.00 sec)
```

20. PERIOD_DIFF(P1,P2)

Returns the number of months between periods P1 and P2. These periods P1 and P2 should be in the format YYMM or YYYYMM. Note that the period arguments P1 and P2 are not date values.

```
mysql> SELECT PERIOD_DIFF(9802,199703);
+---------------------------------------------------+
| PERIOD_DIFF(9802,199703)                          |
+---------------------------------------------------+
| 11                                                |
+---------------------------------------------------+
1 row in set (0.00 sec)
```

21. QUARTER(date)

Returns the quarter of the year for date, in the range 1 to 4.

```
mysql> SELECT QUARTER('98-04-01');
+---------------------------------------------------+
| QUARTER('98-04-01')                               |
+---------------------------------------------------+
| 2                                                 |
+---------------------------------------------------+
```

1 row in set (0.00 sec)

22. SECOND(time)

Returns the second for time, in the range 0 to 59.

```
mysql> SELECT SECOND('10:05:03');
+-----------------------------------------------------+
| SECOND('10:05:03')                                  |
+-----------------------------------------------------+
| 3                                                   |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

23. SEC_TO_TIME(seconds)

Returns the seconds argument, converted to hours, minutes and seconds, as a value in 'HH:MM:SS' or HHMMSS format, depending on whether the function is used in a string or numeric context.

```
mysql> SELECT SEC_TO_TIME(2378);
+-----------------------------------------------------+
| SEC_TO_TIME(2378)                                   |
+-----------------------------------------------------+
| 00:39:38                                            |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

24. STR_TO_DATE(str,format)

This is the inverse of the DATE_FORMAT() function. It takes a string str and a format string format. The STR_TO_DATE() function returns a DATETIME value if the format string contains both date and time parts. Else, it returns a DATE or TIME value if the string contains only date or time parts.

```
mysql> SELECT STR_TO_DATE('04/31/2004', '%m/%d/%Y');
+-----------------------------------------------------+
| STR_TO_DATE('04/31/2004', '%m/%d/%Y')               |
+-----------------------------------------------------+
| 2004-04-31                                          |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

25. SUBDATE(date,INTERVAL expr unit) and SUBDATE(expr,days)

When invoked with the INTERVAL form of the second argument, SUBDATE() is a synonym for DATE_SUB(). For information on the INTERVAL unit argument, see the discussion for DATE_ADD().

```
mysql> SELECT DATE_SUB('1998-01-02', INTERVAL 31 DAY);
+-----------------------------------------------------+
| DATE_SUB('1998-01-02', INTERVAL 31 DAY)             |
+-----------------------------------------------------+
```

```
| 1997-12-02                                  |
+---------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT SUBDATE('1998-01-02', INTERVAL 31 DAY);
+---------------------------------------------+
| SUBDATE('1998-01-02', INTERVAL 31 DAY)      |
+---------------------------------------------+
| 1997-12-02                                  |
+---------------------------------------------+
1 row in set (0.00 sec)
```

26. SUBTIME(expr1,expr2)

The SUBTIME() function returns expr1 . expr2 expressed as a value in the same format as expr1. The expr1 value is a time or a datetime expression, while the expr2 value is a time expression.

```
mysql> SELECT SUBTIME('1997-12-31 23:59:59.999999',
  -> '1 1:1:1.000002');
+---------------------------------------------+
| SUBTIME('1997-12-31 23:59:59.999999'...     |
+---------------------------------------------+
| 1997-12-30 22:58:58.999997                  |
+---------------------------------------------+
1 row in set (0.00 sec)
```

27. SYSDATE()

Returns the current date and time as a value in 'YYYY-MM-DD HH:MM:SS' or YYYYMMDDHHMMSS format, depending on whether the function is used in a string or in a numeric context.

```
mysql> SELECT SYSDATE();
+---------------------------------------------+
| SYSDATE()                                   |
+---------------------------------------------+
| 2006-04-12 13:47:44                         |
+---------------------------------------------+
1 row in set (0.00 sec)
```

28. TIME(expr)

Extracts the time part of the time or datetime expression expr and returns it as a string.

```
mysql> SELECT TIME('2003-12-31 01:02:03');
+---------------------------------------------+
| TIME('2003-12-31 01:02:03')                 |
+---------------------------------------------+
| 01:02:03                                    |
+---------------------------------------------+
1 row in set (0.00 sec)
```

29. TIMEDIFF(expr1,expr2)

The TIMEDIFF() function returns expr1 . expr2 expressed as a time value. These expr1 and expr2 values are time or date-and-time expressions, but both must be of the same type.

```
mysql> SELECT TIMEDIFF('1997-12-31 23:59:59.000001',
  -> '1997-12-30 01:01:01.000002');
+----------------------------------------------------+
| TIMEDIFF('1997-12-31 23:59:59.000001'.....         |
+----------------------------------------------------+
|  46:58:57.999999                                   |
+----------------------------------------------------+
1 row in set (0.00 sec)
```

30. TIMESTAMP(expr), TIMESTAMP(expr1,expr2)

With a single argument, this function returns the date or datetime expression expr as a datetime value. With two arguments, it adds the time expression expr2 to the date or datetime expression expr1 and returns the result as a datetime value.

```
mysql> SELECT TIMESTAMP('2003-12-31');
+----------------------------------------------------+
| TIMESTAMP('2003-12-31')                            |
+----------------------------------------------------+
| 2003-12-31 00:00:00                                |
+----------------------------------------------------+
1 row in set (0.00 sec)
```

31. TIMESTAMPADD(unit,interval,datetime_expr)

This function adds the integer expression interval to the date or datetime expression datetime_expr. The unit for interval is given by the unit argument, which should be one of the following values −

FRAC_SECOND
SECOND, MINUTE
HOUR, DAY
WEEK
MONTH
QUARTER or
YEAR

The unit value may be specified using one of the keywords as shown or with a prefix of SQL_TSI_. For example, DAY and SQL_TSI_DAY both are legal.

```
mysql> SELECT TIMESTAMPADD(MINUTE,1,'2003-01-02');
+----------------------------------------------------+
| TIMESTAMPADD(MINUTE,1,'2003-01-02')                |
+----------------------------------------------------+
| 2003-01-02 00:01:00                                |
+----------------------------------------------------+
1 row in set (0.00 sec)
```

32. TIMESTAMPDIFF(unit,datetime_expr1,datetime_expr2)

Returns the integer difference between the date or datetime expressions datetime_expr1 and datetime_expr2. The unit for the result is given by the unit argument. The legal values for the unit are the same as those listed in the description of the TIMESTAMPADD() function.

```
mysql> SELECT TIMESTAMPDIFF(MONTH,'2003-02-01','2003-05-01');
+-----------------------------------------------------+
| TIMESTAMPDIFF(MONTH,'2003-02-01','2003-05-01')      |
+-----------------------------------------------------+
| 3                                                   |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

33. TIME_FORMAT(time,format)

This function is used like the DATE_FORMAT() function, but the format string may contain format specifiers only for hours, minutes and seconds.

If the time value contains an hour part that is greater than 23, the %H and %k hour format specifiers produce a value larger than the usual range of 0 to 23. The other hour format specifiers produce the hour value modulo 12.

```
mysql> SELECT TIME_FORMAT('100:00:00', '%H %k %h %I %l');
+-----------------------------------------------------+
| TIME_FORMAT('100:00:00', '%H %k %h %I %l')          |
+-----------------------------------------------------+
| 100 100 04 04 4                                     |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

34. TIME_TO_SEC(time)

Returns the time argument converted to seconds.

```
mysql> SELECT TIME_TO_SEC('22:23:00');
+-----------------------------------------------------+
| TIME_TO_SEC('22:23:00')                             |
+-----------------------------------------------------+
| 80580                                               |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

35. TO_DAYS(date)

Given a date, returns a day number (the number of days since year 0).

```
mysql> SELECT TO_DAYS(950501);
+-----------------------------------------------------+
| TO_DAYS(950501)                                     |
+-----------------------------------------------------+
```

```
| 728779                                  |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

36. UNIX_TIMESTAMP(), UNIX_TIMESTAMP(date)

If called with no argument, this function returns a Unix timestamp (seconds since '1970-01-01 00:00:00' UTC) as an unsigned integer. If UNIX_TIMESTAMP() is called with a date argument, it returns the value of the argument as seconds since '1970-01-01 00:00:00' UTC. date may be a DATE string, a DATETIME string, a TIMESTAMP, or a number in the format YYMMDD or YYYYMMDD.

```
mysql> SELECT UNIX_TIMESTAMP();
+-----------------------------------------------------+
| UNIX_TIMESTAMP()                                    |
+-----------------------------------------------------+
| 882226357                                           |
+-----------------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT UNIX_TIMESTAMP('1997-10-04 22:23:00');
+-----------------------------------------------------+
| UNIX_TIMESTAMP('1997-10-04 22:23:00')               |
+-----------------------------------------------------+
| 875996580                                           |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

37. UTC_DATE, UTC_DATE()

Returns the current UTC date as a value in 'YYYY-MM-DD' or YYYYMMDD format, depending on whether the function is used in a string or numeric context.

```
mysql> SELECT UTC_DATE(), UTC_DATE() + 0;
+-----------------------------------------------------+
| UTC_DATE(), UTC_DATE() + 0                          |
+-----------------------------------------------------+
| 2003-08-14, 20030814                                |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

38. UTC_TIME, UTC_TIME()

Returns the current UTC time as a value in 'HH:MM:SS' or HHMMSS format, depending on whether the function is used in a string or numeric context.

```
mysql> SELECT UTC_TIME(), UTC_TIME() + 0;
+-----------------------------------------------------+
| UTC_TIME(), UTC_TIME() + 0                          |
+-----------------------------------------------------+
| 18:07:53, 180753                                    |
```

```
+---------------------------------------------------+
1 row in set (0.00 sec)
```

UTC_TIMESTAMP, UTC_TIMESTAMP()

Returns the current UTC date and time as a value in 'YYYY-MM-DD HH:MM:SS' or in a YYYYMMDDHHMMSS format, depending on whether the function is used in a string or in a numeric context.

```
mysql> SELECT UTC_TIMESTAMP(), UTC_TIMESTAMP() + 0;
+---------------------------------------------------+
| UTC_TIMESTAMP(), UTC_TIMESTAMP() + 0              |
+---------------------------------------------------+
| 2003-08-14 18:08:04, 20030814180804               |
+---------------------------------------------------+
1 row in set (0.00 sec)
```

39. WEEK(date[,mode])

This function returns the week number for date. The two-argument form of WEEK() allows you to specify whether the week starts on a Sunday or a Monday and whether the return value should be in the range from 0 to 53 or from 1 to 53. If the mode argument is omitted, the value of the default_week_format system variable is used

Mode	First Day of week	Range	Week 1 is the first week.
0	Sunday	0-53	with a Sunday in this year
1	Monday	0-53	with more than 3 days this year
2	Sunday	1-53	with a Sunday in this year
3	Monday	1-53	with more than 3 days this year
4	Sunday	0-53	with more than 3 days this year
5	Monday	0-53	with a Monday in this year
6	Sunday	1-53	with more than 3 days this year
7	Monday	1-53	with a Monday in this year

```
mysql> SELECT WEEK('1998-02-20');
+---------------------------------------------------+
| WEEK('1998-02-20')                                |
+---------------------------------------------------+
| 7                                                 |
+---------------------------------------------------+
1 row in set (0.00 sec)
```

40. WEEKDAY (date)

Returns the weekday index for date (0 = Monday, 1 = Tuesday, . 6 = Sunday).

```
mysql> SELECT WEEKDAY('1998-02-03 22:23:00');
+-----------------------------------------------------+
| WEEKDAY('1998-02-03 22:23:00')                      |
+-----------------------------------------------------+
| 1                                                   |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

41. WEEKOFYEAR(date)

Returns the calendar week of the date as a number in the range from 1 to 53. WEEKOFYEAR() is a compatibility function that is equivalent to WEEK(date,3).

```
mysql> SELECT WEEKOFYEAR('1998-02-20');
+-----------------------------------------------------+
| WEEKOFYEAR('1998-02-20')                            |
+-----------------------------------------------------+
| 8                                                   |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

42. YEAR (date)

Returns the year for date, in the range 1000 to 9999, or 0 for the .zero. date.

```
mysql> SELECT YEAR('98-02-03');
+-----------------------------------------------------+
| YEAR('98-02-03')                                    |
+-----------------------------------------------------+
| 1998                                                |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

43. YEARWEEK(date), YEARWEEK(date,mode)

Returns the year and the week for a date. The mode argument works exactly like the mode argument to the WEEK() function. The year in the result may be different from the year in the date argument for the first and the last week of the year.

```
mysql> SELECT YEARWEEK('1987-01-01');
+-----------------------------------------------------+
| YEAR('98-02-03')YEARWEEK('1987-01-01')              |
+-----------------------------------------------------+
| 198653                                              |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

Note − The week number is different from what the WEEK() function would return (0) for optional arguments 0 or 1, as WEEK() then returns the week in the context of the given year.

Summary

In this chapter, you have learned about:
We studied, Different types of SQL Date functions
Also, we studied, Timestamp functions
Also studied all the examples for Date and Time Functions

MCQ's

1. _____ function returns current date and time.
a) SET DATEFIRST
b) SYSDATETIME
c) Cert_ID
d) GETDATE

2. Which of the following function checks whether the expression is a valid date or not ?
a) ISDATE
b) ISDAY
c) ISVALID
d) ISYEAR

3. Which of the following is not a mathematical function?
a) ATN2
b) POWER
c) PI
d) CEIL

4. @@NESTLEVEL function falls under which of the following category?
a) Configuration functions
b) Cursor functions
c) Mathematical functions
d) Date and Time Data Functions

5. _____ are used for supporting encryption, decryption, digital signing and their validation.
a) Cryptographic functions
b) Cursor functions
c) Configuration functions
d) None of the mentioned

6. DecryptByKeyAutoCert is _____ type function.
a) Symmetric Encryption and decryption
b) Encryption Hashing
c) Asymmetric Encryption and decryption
d) Symmetric decryption with Automatic key handling

7. Text and Image Functions are _____
a) nondeterministic
b) deterministic
c) table valued
d) All of the mentioned

8. Which of the the following is not conversion function?
a) CAST and CONVERT
b) PARSE
c) TRY_CAST
d) TRY_CASE

9. _____ returns the rank of rows within the partition of a result set, without any gaps in the ranking.

a) RANK
b) NTILE
c) DENSE_RANK
d) ROW_NUMBER

10. Built in Functions in SQL Server is categorized in to how many categories?

a) 4
b) 5
c) 6
d) 7

Fill in the blank

1. The week number is different from what the ----------function would return (0) for optional arguments 0 or 1.
2. ----------Subtracts a specified time interval from a date
3. %a---------- (Sun-Sat)
4. ---------- is use to find the difference between dates
5. ----------returns the name of the day and month of a given date expression.

Chapter 9

Subqueries in SQL

- What is subquery
- Subqueries with the select statement
- Subqueries with the insert statement
- Subqueries with the Update statement
- Subqueries with the Delete Statement
- Using Auto_increment column

- A Subquery or Inner query or a Nested query is a query within another SQL query and embedded within the WHERE clause.

- A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.

- Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.

9.1 Rules for subqueries

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 command cannot be used in a subquery, although the main query can use an ORDER BY. The GROUP BY command 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.

Assignment No 53

Subqueries with the SELECT Statement

Syntax

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

Example

Consider the CUSTOMERS table having the following records –

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  35 | 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 |
+----+----------+-----+-----------+----------+
```

Now, let us check the following subquery with a SELECT statement.

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

This would produce the following result.

```
+----+----------+-----+---------+----------+
| ID | NAME     | AGE | ADDRESS | SALARY   |
+----+----------+-----+---------+----------+
|  4 | Chaitali |  25 | Mumbai  |  6500.00 |
|  5 | Hardik   |  27 | Bhopal  |  8500.00 |
|  7 | Muffy    |  24 | Indore  | 10000.00 |
+----+----------+-----+---------+----------+
```

9.1.1 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 ]
```

Assignment No 54

Example

Consider a table CUSTOMERS_BKP with similar structure as CUSTOMERS table. Now to copy the complete CUSTOMERS table into the CUSTOMERS_BKP table, you can use the following syntax.

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

9.1.2 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) ]
```

Assignment No 55

Assuming, we have CUSTOMERS_BKP table available which is backup of CUSTOMERS table. The following example updates SALARY by 0.25 times in the 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 );
```

This would impact two rows and finally CUSTOMERS table would have the following records.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  35 | Ahmedabad |   125.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  2125.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

9.1.3 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) ]
```

Assignment No 56

Example

Assuming, we have a CUSTOMERS_BKP table available which is a backup of the CUSTOMERS table. The following example deletes the records from the 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 );
```

This would impact two rows and finally the CUSTOMERS table would have the following records.

```
+----+----------+-----+---------+----------+
| ID | NAME     | AGE | ADDRESS | SALARY   |
+----+----------+-----+---------+----------+
|  2 | Khilan   |  25 | Delhi   |  1500.00 |
|  3 | kaushik  |  23 | Kota    |  2000.00 |
```

```
|  4 | Chaitali |  25 | Mumbai  |  6500.00 |
|  6 | Komal    |  22 | MP      |  4500.00 |
|  7 | Muffy    |  24 | Indore  | 10000.00 |
+----+----------+-----+---------+----------+
```

9.1.4 Using AUTO_INCREMENT column

The simplest way in MySQL to use sequences is to define a column as AUTO_INCREMENT and leave the rest to MySQL to take care.

Example

Try out the following example. This will create a table and after that it will insert a few rows in this table where it is not required to give a record ID because its auto-incremented by MySQL.

Assignment No 57

```
mysql> CREATE TABLE INSECT
   -> (
   -> id INT UNSIGNED NOT NULL AUTO_INCREMENT,
   -> PRIMARY KEY (id),
   -> name VARCHAR(30) NOT NULL, # type of insect
   -> date DATE NOT NULL, # date collected
   -> origin VARCHAR(30) NOT NULL # where collected
);
Query OK, 0 rows affected (0.02 sec)
mysql> INSERT INTO INSECT (id,name,date,origin) VALUES
   -> (NULL,'housefly','2001-09-10','kitchen'),
   -> (NULL,'millipede','2001-09-10','driveway'),
   -> (NULL,'grasshopper','2001-09-10','front yard');
Query OK, 3 rows affected (0.02 sec)
Records: 3  Duplicates: 0  Warnings: 0
mysql> SELECT * FROM INSECT ORDER BY id;
+----+-------------+------------+------------+
| id | name        | date       | origin     |
+----+-------------+------------+------------+
|  1 | housefly    | 2001-09-10 | kitchen    |
|  2 | millipede   | 2001-09-10 | driveway   |
|  3 | grasshopper | 2001-09-10 | front yard |
+----+-------------+------------+------------+
3 rows in set (0.00 sec)
```

Summary

In this chapter, you have learned about:
We studied what is Subquery
Also, we studied Subquery with select, insert, update and Delete statement
Also, we studied how values increment Automatically.

MCQ's

1. Select _____ from instructor where dept name='Comp. Sci.';
Which of the following should be used to find the mean of the salary ?
a) Mean(salary)
b) Avg(salary)
c) Sum(salary)
d) Count(salary)

2. The _____ connective tests for set membership, where the set is a collection of values produced by a select clause. The _____ connective tests for the absence of set membership.
a) Or, in
b) Not in, in
c) In, not in
d) In, or

3. Select ID, GPA from student grades order by GPA _____
Inorder to give only 10 rank on the whole we should use :
a) Limit 10
b) Upto 10
c) Only 10
d) Max 10

4. Suppose we are given a view tot credits (year, num credits) giving the total number of credits taken by students in each year. The query that computes averages over the 3 preceding tuples in the preceding tuples in the specified sort order is:
a) SELECT YEAR, avg (num credits)
OVER (ORDER BY YEAR ROWS 3 preceding) AS avg total credits
FROM tot credits;
b) SELECT YEAR, avg(num credits)
OVER (ORDER BY YEAR ROWS 3 unbounded preceding) AS avg total credits
FROM tot credits;
c) SELECT YEAR, MIN(num credits)
OVER (ORDER BY YEAR ROWS 3 unbounded preceding) AS avg total credits
FROM tot credits;
d) SELECT YEAR, SUM(num credits)
OVER (ORDER BY YEAR ROWS 3 unbounded preceding) AS avg total credits
FROM tot credits;

5. Which of the following is not the function of client?
a) Compile queries
b) Query optimization
c) Receive queries
d) Result formatting and presentation

6. Which server can join the indexes when only multiple indexes combined can cover the query?
a) SQL
b) DBMS
c) RDBMS
d) All of the mentioned

7. Select _____ dept_name from instructor;
Here which of the following displays the unique values of the column?
a) All
b) From
c) Distinct
d) Name

8. Select ID, name, dept name, salary * 1.1 where instructor;
The query given below will not give an error. Which one of the following has to be replaced to get the desired output?
a) Salary*1.1
b) ID

c) Where
d) Instructor

9. What is a subquery?

a) A subquery is a select-from-where expression that is nested within another query
b) A subquery is any query that is nested within another query
c) A subquery is a relation that is externally specified which can be used to handle data in queries
d) A subquery is a condition that excludes all the invalid tuples from the database

10. If a set is a collection of values given by the select clause, the _____ connective tests for set membership

a) within
b) include
c) under
d) in

Fill in the blank

1.A ---------- is a select-from-where expression that is nested within another query.
2. If a set is a collection of values given by the select clause, The _____ connective tests for set membership.
3. A sub-query that uses the correlation name of an --------.
4. The _____ construct returns true if the argument in the sub-query is void of duplicates.
5. It returns the -------- of the results of the results of any two different queries.