Apex Institute of Technology

Program Name: BE CSE (AIML)

LABMANUAL

Semester	:	3rd
CourseName	:	Database Management System Lab
CourseCode	:	CSH – 243
CourseCoordinator	:	Mr. Saurabh Singhal

21CSH-232	**Database Management System**	**L**	**T**	**P**	**S**	**C**	**CH**	**Course type**
Version 1.00		3	0	2	0	4	5	Core
					21CSH-232			
Pre-requisites/ Exposure	Knowledge of basic file management systems							
Co-requisites	20CSP217							
Anti- requisites								

Internal Evaluation Component:

Sr. No.	**Type of Assessment Task**	**Weightage of actual conduct**	**Frequency of Task**	**Final Weightage in Internal Assessment (Prorated Marks)**	**Remarks**
1.	Assignment*	10 marks of each assignment	One Per Unit	10 marks	As applicable to course types depicted above.
2.	Time Bound Surprise Test	12 marks for each test	One per Unit	4 marks	As applicable to course types depicted above.
3.	Quiz	4 marks of each quiz	2 per Unit	4 marks	As applicable to course types depicted above.
4.	Mid-Semester Test**	20 marks for one MST.	2 per semester	20 marks	As applicable to course types depicted above.
5.	Presentation***			Non Graded: Engagement Task	Only for Self Study MNG Courses.
6.	Homework	NA	One per lecture topic (of 2 questions)	Non-Graded: Engagement Task	As applicable to course types depicted above.
7.	Discussion Forum	NA	One per Chapter	Non Graded: Engagement Task	As applicable to course types depicted above.
8.	Attendance and Engagement Score on BB	NA	NA	2 marks	

COURSE OBJECTIVES

The course aims to –

- Understand database system concepts and design databases for different applications and to acquire the knowledge on DBMS and RDBMS
- Implement and Understand different types of DDL, DML and DCL Statements
- Understand transaction concepts related to databases and recovery / back up techniques required for proper storage of data

COURSE OUTCOMES

CO – 1 – Understand the database concept, system architecture and role of Database Administrator

CO – 2 – Design database for an organization using relational model

CO – 3 – Apply Relational Algebra and Relational Calculas Query to the database of an organization

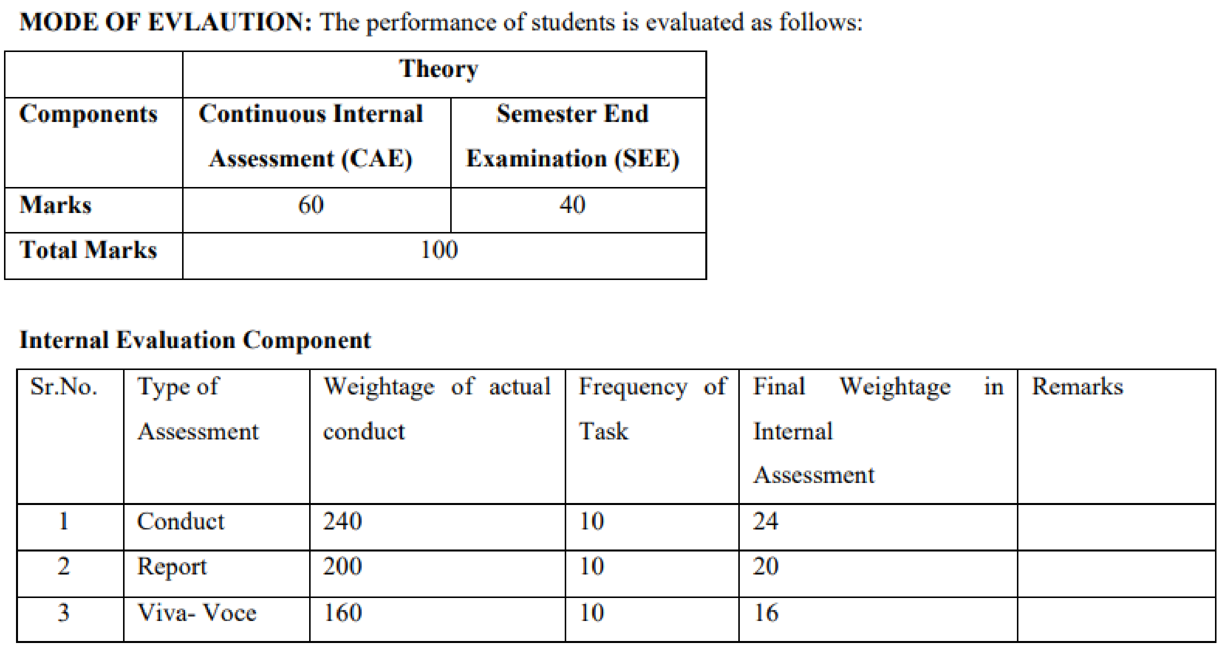
CO – 4 – Implement the packages, procedures and triggers

CO – 5 – Understand the concept of Transaction Processing and Concurrency Control

Lab Experiments with CO Mapping

S.NO.	**Experiment**	Mapped CO
	Unit-I:- Introduction to DBMS, RDBMS and SQL Commands	
1	Experiment 1.1 Introduction to DBMS, RDBMS, Oracle and Basic SQL Commands	**CO1**
2	Experiment 1.2 Create Tables and Specify Queries in SQL	**CO1**
3	Experiment 1.3 To manipulate operations on the table	**CO2**
4	Experiment 1.4 To manipulate restrictions on the table	**CO2**
5	Experiment 1.5 To implement the structure of the table	**CO2**
	Unit 2:- Database Joins, Indexes, Views, Cursors and Triggers	
6	Experiment 2.1 To implement the concept of Joins.	**CO3**
7	Experiment 2.2 To implement the concept of Grouping of Data	**CO3**
8	Experiment 2.3 To implement the concept of Sub Query	**CO3**
9	Experiment 2.4 To implement the concept of indexes, cursors, triggers and views	**CO4**
	Unit 3:- Database System Designing	
10	Experiment 3.1 Design a case study for Company database / Hospital Management System / Railway Reservation System	**CO5**

MODE OF EVALUATION: The performance of students is evaluated as follows:



MODE OF EVLAUTION: The performance of students is evaluated as follows:

	Theory	
Components	**Continuous Internal Assessment (CAE)**	**Semester End Examination (SEE)**
Marks	60	40
Total Marks	100	

Internal Evaluation Component

Sr.No.	Type of Assessment	Weightage of actual conduct	Frequency of Task	Final Weightage in Internal Assessment	Remarks
1	Conduct	240	10	24	
2	Report	200	10	20	
3	Viva- Voce	160	10	16	

CO – PO – PSO Mapping

DBMS (P)

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3	PSO4
CO1	2	1	1	1	1	1	1	1	2	2	1	3	2	2	1	1
CO2	1	2	2	2	1	2	1	1	1	1	1	2	2	2	1	1
CO3	2	3	3	2	1	2	1	1	2	1	1	2	2	2	1	1

EXPERIMENT 1.1

Mapped Course Outcomes-CO1

CO1: Understand the database concept, system architecture and role of Database Administrator

AIM: Introduction to DBMS, RDBMS, Oracle and Basic SQL Commands

SQL Commands

- SQL commands are instructions. It is used to communicate with the database. It is also used to perform specific tasks, functions, and queries of data.
- SQL can perform various tasks like create a table, add data to tables, drop the table, modify the table, set permission for users.

Data Definition Language (DDL)

- DDL changes the structure of the table like creating a table, deleting a table, altering a table, etc.
- All the command of DDL is auto-committed that means it permanently saves all the changes in the database.
- Here are some commands that come under DDL:
  - CREATE
  - ALTER
  - DROP
  - TRUNCATE

a. CREATE It is used to create a new table in the database.

Syntax:

```
CREATE TABLE TABLE_NAME (COLUMN_NAME DATATYPES[,....]);
```

Example:

```
CREATE TABLE EMPLOYEE(Name VARCHAR2(20), Email VARCHAR2(100), DOB DATE);
```

b. DROP: It is used to delete both the structure and record stored in the table.

Syntax

```
DROP TABLE table_name;
```

Example

```
DROP TABLE EMPLOYEE;
```

c. ALTER: It is used to alter the structure of the database. This change could be either to modify the characteristics of an existing attribute or probably to add a new attribute.

Syntax:

To add a new column in the table

ALTER TABLE table_name ADD column_name COLUMN-definition;

To modify existing column in the table:

ALTER TABLE table_name MODIFY(column_definitions....);

EXAMPLE

ALTER TABLE STU_DETAILS ADD(ADDRESS VARCHAR2(20));

ALTER TABLE STU_DETAILS MODIFY (NAME VARCHAR2(20));

d. TRUNCATE: It is used to delete all the rows from the table and free the space containing the table.

Syntax:

TRUNCATE TABLE table_name;

Example:

TRUNCATE TABLE EMPLOYEE;

2. Data Manipulation Language

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

Here are some commands that come under DML:

- INSERT
- UPDATE
- DELETE

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

Syntax:

INSERT INTO TABLE_NAME

(col1, col2, col3,.... col N)

VALUES (value1, value2, value3, ... valueN);

Or

INSERT INTO TABLE_NAME

VALUES (value1, value2, value3, ... valueN);

For example:

INSERT INTO javatpoint (Author, Subject) VALUES ("Sonoo", "DBMS");

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

Syntax:

UPDATE table_name SET [column_name1= value1,...column_nameN = valueN] [WHERE CONDITION]

For example:

UPDATE students SET User_Name = 'Sonoo' WHERE Student_Id = '3'

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

Syntax:

DELETE FROM table_name [WHERE condition];

For example:

DELETE FROM javatpoint WHERE Author="Sonoo";

3. Data Control Language

DCL commands are used to grant and take back authority from any database user.

Here are some commands that come under DCL:

- Grant
- Revoke

a. Grant: It is used to give user access privileges to a database.

Example

GRANT SELECT, UPDATE ON MY_TABLE TO SOME_USER, ANOTHER_USER;

b. Revoke: It is used to take back permissions from the user.

Example

REVOKE SELECT, UPDATE ON MY_TABLE FROM USER1, USER2;

4. Transaction Control Language

TCL commands can only use with DML commands like INSERT, DELETE and UPDATE only.

These operations are automatically committed in the database that's why they cannot be used while creating tables or dropping them.

Here are some commands that come under TCL:

- COMMIT
- ROLLBACK
- SAVEPOINT

a. Commit: Commit command is used to save all the transactions to the database.

Syntax:

COMMIT;

Example:

DELETE FROM CUSTOMERS WHERE AGE = 25; COMMIT;

b. Rollback: Rollback command is used to undo transactions that have not already been saved to the database.

Syntax: ROLLBACK;

Example:

DELETE FROM CUSTOMERS WHERE AGE = 25; ROLLBACK;

c. SAVEPOINT: It is used to roll the transaction back to a certain point without rolling back the entire transaction.

Syntax:

SAVEPOINT SAVEPOINT_NAME;

5. Data Query Language

DQL is used to fetch the data from the database.

It uses only one command:

- SELECT

a. SELECT: This is the same as the projection operation of relational algebra. It is used to select the attribute based on the condition described by WHERE clause.

Syntax:

SELECT expressions   FROM TABLES   WHERE conditions;

For example:

```
SELECT emp_name FROM employee  WHERE age > 20;
```

Further Reading

- Ramez Elmasriand Shamkant B. Navathe, “Fundamentals of Database System”, The Benjamin /Cummings Publishing Co
- C.J.Date, “AnIntroduction to Database Systems”, Addison Wesley

Video Links -

https://www.youtube.com/watch?v=n6

GcLrSt4yI

EXPERIMENT 1.2

Mapped Course Outcomes-CO1

CO1: Understand the database concept, system architecture and role of Database Administrator

Aim: Create Tables and Specify Queries in SQL

SQL CREATE TABLE
Syntax

```
CREATE TABLE table_name (

   column1 datatype,

   column2 datatype,

   column3 datatype,

  ....

);
```

```
CREATE TABLE Persons (

   PersonID int,

   LastName varchar(255),

   FirstName varchar(255),

   Address varchar(255),

   City varchar(255)

);
```

SQL INSERT INTO STATEMENT

(1). Specify both the column names and the values to be inserted:

```
INSERT INTO table_name (column1, column2, column3, ...)

VALUES (value1, value2, value3, ...);
```

(2). If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. Here, the INSERT INTO syntax would be as follows:

```
INSERT INTO table_name
```

```
VALUES (value1, value2, value3, ...);
```

Demo Database

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
89	White Clover Markets	Karl Jablonski	305 - 14th Ave. S. Suite 3B	Seattle	98128	USA
90	Wilman Kala	Matti Karttunen	Keskuskatu 45	Helsinki	21240	Finland
91	Wolski	Zbyszek	ul. Filtrowa 68	Walla	01-012	Poland

INSERT INTO Example

```
INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)

VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway');
```

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
89	White Clover Markets	Karl Jablonski	305 - 14th Ave. S. Suite 3B	Seattle	98128	USA
90	Wilman Kala	Matti Karttunen	Keskuskatu 45	Helsinki	21240	Finland
91	Wolski	Zbyszek	ul. Filtrowa 68	Walla	01-012	Poland
92	Cardinal	Tom B. Erichsen	Skagen 21	Stavanger	4006	Norway

Insert Data Only in Specified Columns

```
INSERT INTO Customers (CustomerName, City, Country)

VALUES ('Cardinal', 'Stavanger', 'Norway');
```

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
89	White Clover Markets	Karl Jablonski	305 - 14th Ave. S. Suite 3B	Seattle	98128	USA
90	Wilman Kala	Matti Karttunen	Keskuskatu 45	Helsinki	21240	Finland
91	Wolski	Zbyszek	ul. Filtrowa 68	Walla	01-012	Poland
92	Cardinal	null	null	Stavanger	null	Norway

The 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;
```

Here, column1, column2, ... are the field names of the table you want to select data from. If you want to select all the fields available in the table, use the following syntax:

```
SELECT * FROM table_name;
```

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 Column Example

The following SQL statement selects the "CustomerName" and "City" columns from the "Customers" table:

Example

```
SELECT CustomerName, City FROM Customers;
```

SQL SELECT DISTINCT Statement
The SQL SELECT DISTINCT Statement

The SELECT DISTINCT statement is used to return only distinct (different) values.

Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values.

SELECT DISTINCT Syntax

```
SELECT DISTINCT column1, column2, ...

FROM table_name;
```

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 Example Without DISTINCT

The following SQL statement selects all (including the duplicates) values from the "Country" column in the "Customers" table:

Example

```
SELECT Country FROM Customers;
```

SELECT DISTINCT Examples

```
SELECT DISTINCT Country FROM Customers;

SELECT COUNT(DISTINCT Country) FROM Customers;

SELECT Count(*) AS DistinctCountries

FROM (SELECT DISTINCT Country FROM Customers);
```

SQL WHERE Clause
WHERE Syntax

```
SELECT column1, column2, ...

FROM table_name

WHERE condition;
```

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 * FROM Customers

WHERE Country='Mexico';

SELECT * FROM Customers

WHERE CustomerID=1;
```

Operators in The WHERE Clause

Operator	Description	Example
=	Equal	
>	Greater than	
<	Less than	
>=	Greater than or equal	
<=	Less than or equal	
<>	Not equal. **Note:** In some versions of SQL this operator may be written as !=	
BETWEEN	Between a certain range	
LIKE	Search for a pattern	
IN	To specify multiple possible values for a column	

SQL AND, OR and NOT Operators

The WHERE clause can be combined with AND, OR, and NOT operators.

The AND and OR operators are used to filter records based on more than one condition:

- The AND operator displays a record if all the conditions separated by AND are TRUE.
- The OR operator displays a record if any of the conditions separated by OR is TRUE.

The NOT operator displays a record if the condition(s) is NOT TRUE.

AND Syntax

```
SELECT column1, column2, ...

FROM table_name

WHERE condition1 AND condition2 AND condition3 ...;
```

OR Syntax

```
SELECT column1, column2, ...

FROM table_name

WHERE condition1 OR condition2 OR condition3 ...;
```

NOT Syntax

```
SELECT column1, column2, ...

FROM table_name

WHERE NOT condition;
```

Demo Database

The table below shows the complete "Customers" table from the Northwind sample database:

AND Example

The following SQL statement selects all fields from "Customers" where country is "Germany" AND city is "Berlin":

Example

```
SELECT * FROM Customers

WHERE Country='Germany' AND City='Berlin';
```

OR Example

The following SQL statement selects all fields from "Customers" where city is "Berlin" OR "München":

Example

```
SELECT * FROM Customers

WHERE City='Berlin' OR City='München';
```

Example

```
SELECT * FROM Customers

WHERE Country='Germany' OR Country='Spain';
```

NOT Example

The following SQL statement selects all fields from "Customers" where country is NOT "Germany":

Example

```
SELECT * FROM Customers

WHERE NOT Country='Germany';
```

Combining AND, OR and NOT

You can also combine the AND, OR and NOT operators.

The following SQL statement selects all fields from "Customers" where country is "Germany" AND city must be "Berlin" OR "München" (use parenthesis to form complex expressions):

Example

```
SELECT * FROM Customers

WHERE Country='Germany' AND (City='Berlin' OR City='München');
```

The following SQL statement selects all fields from "Customers" where country is NOT "Germany" and NOT "USA":

Example

```
SELECT * FROM Customers

WHERE NOT Country='Germany' AND NOT Country='USA';
```

SQL UPDATE Statement
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!

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 Table

The following SQL statement updates the first customer (CustomerID = 1) with a new contact person *and* a new city.

Example

```
UPDATE Customers

SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'

WHERE CustomerID = 1;
```

The selection from the "Customers" table will now look like this:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Alfred Schmidt	Obere Str. 57	Frankfurt	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 Multiple Records

It is the WHERE clause that determines how many records will be updated.

The following SQL statement will update the ContactName to "Juan" for all records where country is "Mexico":

Example

```
UPDATE Customers

SET ContactName='Juan'

WHERE Country='Mexico';
```

The selection from the "Customers" table will now look like this:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Alfred Schmidt	Obere Str. 57	Frankfurt	12209	Germany
2	Ana Trujillo Emparedados y helados	Juan	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Juan	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 Warning!

Be careful when updating records. If you omit the WHERE clause, ALL records will be updated!

Example

```
UPDATE Customers

SET ContactName='Juan';
```

The selection from the "Customers" table will now look like this:

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

SQL DELETE Statement
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!

Demo 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:

Example

```
DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';
```

The "Customers" table will now look like this:

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
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

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;
```

Further Reading

- Ramez Elmasriand Shamkant B. Navathe, “Fundamentals of Database System”, The Benjamin /Cummings Publishing Co
- C.J.Date, “AnIntroduction to Database Systems”, Addison Wesley

Video Links - https://www.youtube.com/watch?v=oReH2vO8Izc

EXPERIMENT 1.3

Mapped Course Outcomes-CO2

CO2: Design database for an organization using relational model

Aim- To manipulate operations on the table

SQL CREATE TABLE

Syntax

```
CREATE TABLE table_name (
   column1 datatype,
   column2 datatype,
   column3 datatype,
  ....
);
```

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

SQL INSERT INTO STATEMENT

(1). Specify both the column names and the values to be inserted:

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

(2). If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. Here, the INSERT INTO syntax would be as follows:

INSERT INTO *table_name*

VALUES (*value1*, *value2*, *value3*, ...);

Demo Database

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
89	White Clover Markets	Karl Jablonski	305 - 14th Ave. S. Suite 3B	Seattle	98128	USA
90	Wilman Kala	Matti Karttunen	Keskuskatu 45	Helsinki	21240	Finland
91	Wolski	Zbyszek	ul. Filtrowa 68	Walla	01-012	Poland

INSERT INTO Example

INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)

VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway');

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
89	White Clover Markets	Karl Jablonski	305 - 14th Ave. S. Suite 3B	Seattle	98128	USA
90	Wilman Kala	Matti Karttunen	Keskuskatu 45	Helsinki	21240	Finland
91	Wolski	Zbyszek	ul. Filtrowa 68	Walla	01-012	Poland
92	Cardinal	Tom B. Erichsen	Skagen 21	Stavanger	4006	Norway

Insert Data Only in Specified Columns

INSERT INTO Customers (CustomerName, City, Country)

VALUES ('Cardinal', 'Stavanger', 'Norway');

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
89	White Clover Markets	Karl Jablonski	305 - 14th Ave. S. Suite 3B	Seattle	98128	USA
90	Wilman Kala	Matti Karttunen	Keskuskatu 45	Helsinki	21240	Finland
91	Wolski	Zbyszek	ul. Filtrowa 68	Walla	01-012	Poland
92	Cardinal	null	null	Stavanger	null	Norway

The 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;
```

Here, column1, column2, ... are the field names of the table you want to select data from. If you want to select all the fields available in the table, use the following syntax:

```
SELECT * FROM table_name;
```

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 Column Example

The following SQL statement selects the "CustomerName" and "City" columns from the "Customers" table:

Example

```
SELECT CustomerName, City FROM Customers;
```

SQL SELECT DISTINCT Statement

The SQL SELECT DISTINCT Statement

The SELECT DISTINCT statement is used to return only distinct (different) values.

Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values.

SELECT DISTINCT Syntax

```
SELECT DISTINCT column1, column2, ...

FROM table_name;
```

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 Example Without DISTINCT

The following SQL statement selects all (including the duplicates) values from the "Country" column in the "Customers" table:

Example

```
SELECT Country FROM Customers;
```

SELECT DISTINCT Examples

```
SELECT DISTINCT Country FROM Customers;

SELECT COUNT(DISTINCT Country) FROM Customers;

SELECT Count(*) AS DistinctCountries

FROM (SELECT DISTINCT Country FROM Customers);
```

SQL WHERE Clause

WHERE Syntax

```
SELECT column1, column2, ...

FROM table_name

WHERE condition;
```

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 * FROM Customers

WHERE Country='Mexico';

SELECT * FROM Customers

WHERE CustomerID=1;
```

Operators in The WHERE Clause

Operator	Description	Example
=	Equal	
>	Greater than	
<	Less than	
>=	Greater than or equal	
<=	Less than or equal	
<>	Not equal. **Note:** In some versions of SQL this operator may be written as !=	
BETWEEN	Between a certain range	
LIKE	Search for a pattern	
IN	To specify multiple possible values for a column	

SQL AND, OR and NOT Operators

The WHERE clause can be combined with AND, OR, and NOT operators.

The AND and OR operators are used to filter records based on more than one condition:

- The AND operator displays a record if all the conditions separated by AND are TRUE.
- The OR operator displays a record if any of the conditions separated by OR is TRUE.

The NOT operator displays a record if the condition(s) is NOT TRUE.

AND Syntax

```
SELECT column1, column2, ...

FROM table_name

WHERE condition1 AND condition2 AND condition3 ...;
```

OR Syntax

```
SELECT column1, column2, ...

FROM table_name

WHERE condition1 OR condition2 OR condition3 ...;
```

NOT Syntax

```
SELECT column1, column2, ...

FROM table_name

WHERE NOT condition;
```

Demo Database

The table below shows the complete "Customers" table from the Northwind sample database:

AND Example

The following SQL statement selects all fields from "Customers" where country is "Germany" AND city is "Berlin":

Example

```
SELECT * FROM Customers

WHERE Country='Germany' AND City='Berlin';
```

OR Example

The following SQL statement selects all fields from "Customers" where city is "Berlin" OR "München":

Example

```
SELECT * FROM Customers

WHERE City='Berlin' OR City='München';
```

Example

```
SELECT * FROM Customers

WHERE Country='Germany' OR Country='Spain';
```

NOT Example

The following SQL statement selects all fields from "Customers" where country is NOT "Germany":

Example

```
SELECT * FROM Customers

WHERE NOT Country='Germany';
```

Combining AND, OR and NOT

You can also combine the AND, OR and NOT operators.

The following SQL statement selects all fields from "Customers" where country is "Germany" AND city must be "Berlin" OR "München" (use parenthesis to form complex expressions):

Example

```
SELECT * FROM Customers

WHERE Country='Germany' AND (City='Berlin' OR City='München');
```

The following SQL statement selects all fields from "Customers" where country is NOT "Germany" and NOT "USA":

Example

```
SELECT * FROM Customers

WHERE NOT Country='Germany' AND NOT Country='USA';
```

SQL UPDATE Statement

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!

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 Table

The following SQL statement updates the first customer (CustomerID = 1) with a new contact person *and* a new city.

Example

```
UPDATE Customers

SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'

WHERE CustomerID = 1;
```

The selection from the "Customers" table will now look like this:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Alfred Schmidt	Obere Str. 57	Frankfurt	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 Multiple Records

It is the WHERE clause that determines how many records will be updated.

The following SQL statement will update the ContactName to "Juan" for all records where country is "Mexico":

Example

```
UPDATE Customers

SET ContactName='Juan'

WHERE Country='Mexico';
```

The selection from the "Customers" table will now look like this:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Alfred Schmidt	Obere Str. 57	Frankfurt	12209	Germany
2	Ana Trujillo Emparedados y helados	Juan	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Juan	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 Warning!

Be careful when updating records. If you omit the WHERE clause, ALL records will be updated!

Example

```
UPDATE Customers

SET ContactName='Juan';
```

The selection from the "Customers" table will now look like this:

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

SQL DELETE Statement

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!

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:

Example

```
DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';
```

The "Customers" table will now look like this:

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
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

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;
```

Further Readings:

- Ramez Elmasriand Shamkant B. Navathe, “Fundamentals of Database System”, The Benjamin /Cummings Publishing Co

- C.J.Date, “AnIntroduction to Database Systems”, Addison Wesley

Video Links – https://www.youtube.com/watch?v=BNbOBvrksZQ

EXPERIMENT 1.4

Mapped Course Outcomes-CO2

CO2: Design database for an organization using relational model

Aim:- To manipulate restrictions on the table.

SQL Constraints

SQL constraints are used to specify rules for the data in a table.

Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.

Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

The following constraints are commonly used in SQL:

- NOT NULL - Ensures that a column cannot have a NULL value
- UNIQUE - Ensures that all values in a column are different
- PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
- FOREIGN KEY - Prevents actions that would destroy links between tables
- CHECK - Ensures that the values in a column satisfies a specific condition
- DEFAULT - Sets a default value for a column if no value is specified
- CREATE INDEX - Used to create and retrieve data from the database very quickly
- *SQL NOT NULL Constraint*
- By default, a column can hold NULL values.
- The NOT NULL constraint enforces a column to NOT accept NULL values.
- This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.
- *SQL NOT NULL on CREATE TABLE*
- The following SQL ensures that the "ID", "LastName", and "FirstName" columns will NOT accept NULL values when the "Persons" table is created:
- Example

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

```
    Age int
);
```

- SQL NOT NULL on ALTER TABLE
- To create a NOT NULL constraint on the "Age" column when the "Persons" table is already created, use the following SQL:

-
  ```
  ALTER TABLE Persons
  MODIFY Age int NOT NULL;
  ```
- SQL UNIQUE Constraint
- The UNIQUE constraint ensures that all values in a column are different.
- Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns.
- A PRIMARY KEY constraint automatically has a UNIQUE constraint.
- However, you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.
-
- SQL UNIQUE Constraint on CREATE TABLE
- The following SQL creates a UNIQUE constraint on the "ID" column when the "Persons" table is created:
- **SQL Server / Oracle / MS Access:**

-
  ```
  CREATE TABLE Persons (
      ID int NOT NULL UNIQUE,
      LastName varchar(255) NOT NULL,
      FirstName varchar(255),
      Age int
  );
  ```
- SQL PRIMARY KEY Constraint
- The PRIMARY KEY constraint uniquely identifies each record in a table.
- Primary keys must contain UNIQUE values, and cannot contain NULL values.
- A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).
-
- SQL PRIMARY KEY on CREATE TABLE
- The following SQL creates a PRIMARY KEY on the "ID" column when the "Persons" table is created:
- **MySQL:**

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

SQL FOREIGN KEY Constraint

The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.

A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the PRIMARY KEY in another table.

The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

Look at the following two tables:

Persons Table

PersonID	LastName	FirstName	Age
1	Hansen	Ola	30
2	Svendson	Tove	23
3	Pettersen	Kari	20

Orders Table

OrderID	OrderNumber	PersonID
1	77895	3
2	44678	3
3	22456	2
4	24562	1

Notice that the "PersonID" column in the "Orders" table points to the "PersonID" column in the "Persons" table.

The "PersonID" column in the "Persons" table is the PRIMARY KEY in the "Persons" table.

The "PersonID" column in the "Orders" table is a FOREIGN KEY in the "Orders" table.

The FOREIGN KEY constraint prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the parent table.

SQL FOREIGN KEY on CREATE TABLE

The following SQL creates a FOREIGN KEY on the "PersonID" column when the "Orders" table is created:

MySQL:

```
CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
);
```

SQL CHECK Constraint

The CHECK constraint is used to limit the value range that can be placed in a column.

If you define a CHECK constraint on a column it will allow only certain values for this column.

If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

SQL CHECK on CREATE TABLE

The following SQL creates a CHECK constraint on the "Age" column when the "Persons" table is created. The CHECK constraint ensures that the age of a person must be 18, or older:

MySQL:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CHECK (Age>=18)
);
```

Further Reading:-

- Ramez Elmasriand Shamkant B. Navathe, “Fundamentals of Database System”, The Benjamin /Cummings Publishing Co
- C.J.Date, “AnIntroduction to Database Systems”, Addison Wesley

Video Links – https://www.youtube.com/watch?v=PWGducbJ5Wo

Experiment 1.5

Mapped Course Outcomes-CO2

CO2: Design database for an organization using relational model

Aim:- To implement structure of the table

Create Table Using Another Table

A copy of an existing table can also be created using CREATE TABLE.

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

If you create a new table using an existing table, the new table will be filled with the existing values from the old table.

Syntax

```
CREATE TABLE new_table_name AS
    SELECT column1, column2,...
    FROM existing_table_name
    WHERE ....;
```

The following SQL creates a new table called "TestTables" (which is a copy of the "Customers" table):

Example

```
CREATE TABLE TestTable AS
SELECT customername, contactname
FROM customers;
```

The SQL DROP TABLE Statement

The DROP TABLE statement is used to drop an existing table in a database.

Syntax

```
DROP TABLE table_name;
```

SQL DROP TABLE Example

The following SQL statement drops the existing table "Shippers":

Example

```
DROP TABLE Shippers;
```

SQL TRUNCATE TABLE

The TRUNCATE TABLE statement is used to delete the data inside a table, but not the table itself.

Syntax

```
TRUNCATE TABLE table_name;
```

SQL ALTER TABLE Statement

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

The ALTER TABLE statement is also used to add and drop various constraints on an existing table.

ALTER TABLE - ADD Column

To add a column in a table, use the following syntax:

```
ALTER TABLE table_name
ADD column_name datatype;
```

The following SQL adds an "Email" column to the "Customers" table:

Example

```
ALTER TABLE Customers
ADD Email varchar(255);
```

ALTER TABLE - DROP COLUMN

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

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

The following SQL deletes the "Email" column from the "Customers" table:

Example

```
ALTER TABLE Customers
DROP COLUMN Email;
```

ALTER TABLE - ALTER/MODIFY COLUMN

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

SQL Server / MS Access:

```
ALTER TABLE table_name
ALTER COLUMN column_name datatype;
```

Further Reading:-

- Ramez Elmasriand Shamkant B. Navathe, “Fundamentals of Database System”, The Benjamin /Cummings Publishing Co

- C.J.Date, “AnIntroduction to Database Systems”, Addison Wesley

Video Links -

https://www.youtube.com/watch?v=ovbXLb36-II

Experiment 2.1

Mapped Course Outcomes-CO3

CO3: Apply Relational Algebra and Relational Calculas Query to the database of an organization

Aim – To implement the concept of Joins.

SQL JOIN

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

Let's look at a selection from the "Orders" table:

OrderID	CustomerID	OrderDate
10308	2	1996-09-18
10309	37	1996-09-19
10310	77	1996-09-20

Then, look at a selection from the "Customers" table:

CustomerID	CustomerName	ContactName	Country
1	Alfreds Futterkiste	Maria Anders	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mexico

Notice that the "CustomerID" column in the "Orders" table refers to the "CustomerID" in the "Customers" table. The relationship between the two tables above is the "CustomerID" column.

Then, we can create the following SQL statement (that contains an INNER JOIN), that selects records that have matching values in both tables:

Example

```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate

FROM Orders

INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;
```

and it will produce something like this:

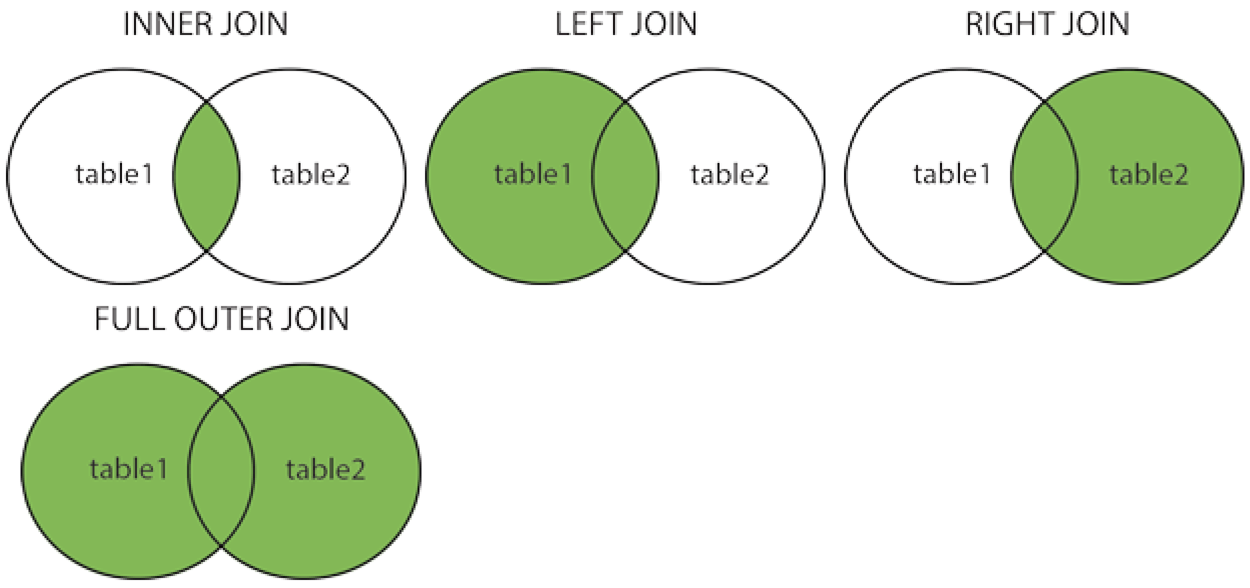
OrderID	CustomerName	OrderDate
10308	Ana Trujillo Emparedados y helados	9/18/1996

10365	Antonio Moreno Taquería	11/27/1996
10383	Around the Horn	12/16/1996
10355	Around the Horn	11/15/1996
10278	Berglunds snabbköp	8/12/1996

Different Types of SQL JOINs

Here are the different types of the JOINs in SQL:

- **(INNER) JOIN:** Returns records that have matching values in both tables
- **LEFT (OUTER) JOIN:** Returns all records from the left table, and the matched records from the right table
- **RIGHT (OUTER) JOIN:** Returns all records from the right table, and the matched records from the left table
- **FULL (OUTER) JOIN:** Returns all records when there is a match in either left or right table



SQL INNER JOIN Keyword

The INNER JOIN keyword selects records that have matching values in both tables.

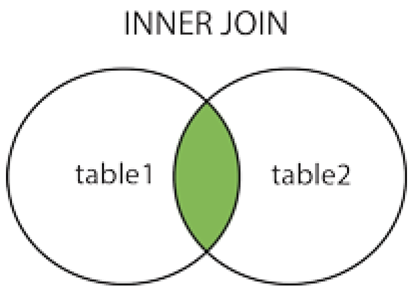
INNER JOIN Syntax

```
SELECT column_name(s)

FROM table1

INNER JOIN table2

ON table1.column_name = table2.column_name;
```



Demo Database

In this tutorial we will use the well-known Northwind sample database.

Below is 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

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

SQL INNER JOIN Example

The following SQL statement selects all orders with customer information:

Example

```
SELECT Orders.OrderID, Customers.CustomerName

FROM Orders

INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
```

Note: The INNER JOIN keyword selects all rows from both tables as long as there is a match between the columns. If there are records in the "Orders" table that do not have matches in "Customers", these orders will not be shown!

JOIN Three Tables

The following SQL statement selects all orders with customer and shipper information:

Example

```
SELECT Orders.OrderID, Customers.CustomerName, Shippers.ShipperName

FROM ((Orders

INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID)

INNER JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID);
```

SQL LEFT JOIN Keyword

The LEFT JOIN keyword returns all records from the left table (table1), and the matching records from the right table (table2). The result is 0 records from the right side, if there is no match.

LEFT JOIN Syntax

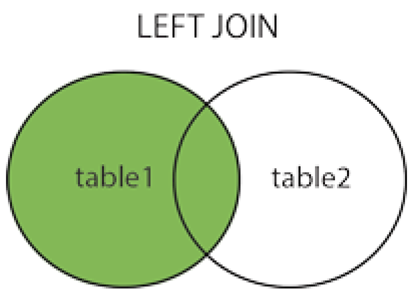
```
SELECT column_name(s)

FROM table1

LEFT JOIN table2

ON table1.column_name = table2.column_name;
```

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



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 "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

SQL LEFT JOIN Example

The following SQL statement will select all customers, and any orders they might have:

Example

```
SELECT Customers.CustomerName, Orders.OrderID

FROM Customers

LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID

ORDER BY Customers.CustomerName;
```

SQL RIGHT JOIN Keyword

The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records from the left table (table1). The result is 0 records from the left side, if there is no match.

RIGHT JOIN Syntax

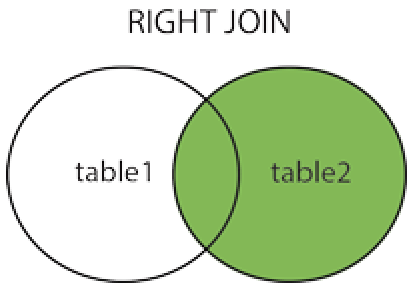
```
SELECT column_name(s)

FROM table1

RIGHT JOIN table2

ON table1.column_name = table2.column_name;
```

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



Demo Database

In this tutorial we will use the well-known Northwind sample database.

Below is 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

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

SQL RIGHT JOIN Example

The following SQL statement will return all employees, and any orders they might have placed:

Example

SELECT Orders.OrderID, Employees.LastName, Employees.FirstName

FROM Orders

RIGHT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID

ORDER BY Orders.OrderID;

SQL FULL OUTER JOIN Keyword

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

Tip: FULL OUTER JOIN and FULL JOIN are the same.

FULL OUTER JOIN Syntax

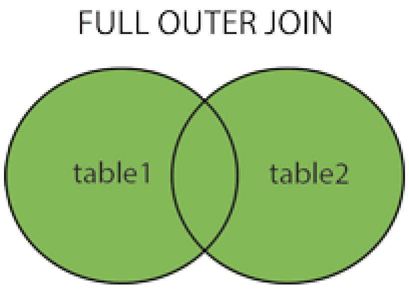
SELECT *column_name(s)*

FROM *table1*

FULL OUTER JOIN *table2*

ON *table1.column_name* = *table2.column_name*

WHERE *condition*;



Note: FULL OUTER JOIN can potentially return very large result-sets!

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 "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

SQL FULL OUTER JOIN Example

The following SQL statement selects all customers, and all orders:

```
SELECT Customers.CustomerName, Orders.OrderID

FROM Customers

FULL OUTER JOIN Orders ON Customers.CustomerID=Orders.CustomerID

ORDER BY Customers.CustomerName;
```

A selection from the result set may look like this:

CustomerName	OrderID
Null	10309
Null	10310
Alfreds Futterkiste	*Null*
Ana Trujillo Emparedados y helados	10308
Antonio Moreno Taquería	*Null*

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.

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.

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

SQL Self Join Example

The following SQL statement matches customers that are from the same city:

Example

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;

Further Readings:

- Ramez Elmasriand Shamkant B. Navathe, "Fundamentals of Database System", The Benjamin /Cummings Publishing Co
- C.J.Date, "AnIntroduction to Database Systems", Addison Wesley

Video Links - https://www.youtube.com/watch?v=9GfoEMUmyzI

Experiment 2.2

Mapped Course Outcomes-CO3

CO3: Apply Relational Algebra and Relational Calculas Query to the database of an organization

AIM:- To implement the concept of Grouping of Data

The SQL GROUP BY Statement

The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

The GROUP BY statement is often used with aggregate functions
(COUNT(), MAX(), MIN(), SUM(), AVG()) to group the result-set by one or more columns.

GROUP BY Syntax

```
SELECT column_name(s)

FROM table_name

WHERE condition

GROUP BY column_name(s)

ORDER BY column_name(s);
```

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 GROUP BY Examples

The following SQL statement lists the number of customers in each country:

Example

```
SELECT COUNT(CustomerID), Country

FROM Customers

GROUP BY Country;
```

The following SQL statement lists the number of customers in each country, sorted high to low:

Example

```
SELECT COUNT(CustomerID), Country

FROM Customers

GROUP BY Country

ORDER BY COUNT(CustomerID) DESC;
```

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 "Shippers" table:

ShipperID	ShipperName
1	Speedy Express
2	United Package
3	Federal Shipping

GROUP BY With JOIN Example

The following SQL statement lists the number of orders sent by each shipper:

Example

```
SELECT Shippers.ShipperName, COUNT(Orders.OrderID) AS NumberOfOrders FROM Orders

LEFT JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID

GROUP BY ShipperName;
```

The SQL HAVING Clause

The HAVING clause was added to SQL because the WHERE keyword cannot be used with aggregate functions.

HAVING Syntax

```
SELECT column_name(s)

FROM table_name

WHERE condition

GROUP BY column_name(s)

HAVING condition

ORDER BY column_name(s);
```

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 HAVING Examples

The following SQL statement lists the number of customers in each country. Only include countries with more than 5 customers:

Example

```
SELECT COUNT(CustomerID), Country

FROM Customers

GROUP BY Country

HAVING COUNT(CustomerID) > 5;
```

The following SQL statement lists the number of customers in each country, sorted high to low (Only include countries with more than 5 customers):

Example

```
SELECT COUNT(CustomerID), Country

FROM Customers

GROUP BY Country

HAVING COUNT(CustomerID) > 5

ORDER BY COUNT(CustomerID) DESC;
```

.

Further Reading:-

- Ramez Elmasriand Shamkant B. Navathe, “Fundamentals of Database System”, The Benjamin /Cummings Publishing Co
- C.J.Date, “AnIntroduction to Database Systems”, Addison Wesley

Video Link - https://www.youtube.com/watch?v=XOfJf7J1Vp8

Experiment 2.3

Mapped Course Outcomes-CO3

CO3: Apply Relational Algebra and Relational Calculas Query to the database of an organization

AIM:- To implement the concept of Sub Query

What is subquery in SQL?

A subquery is a SQL query nested inside a larger query.

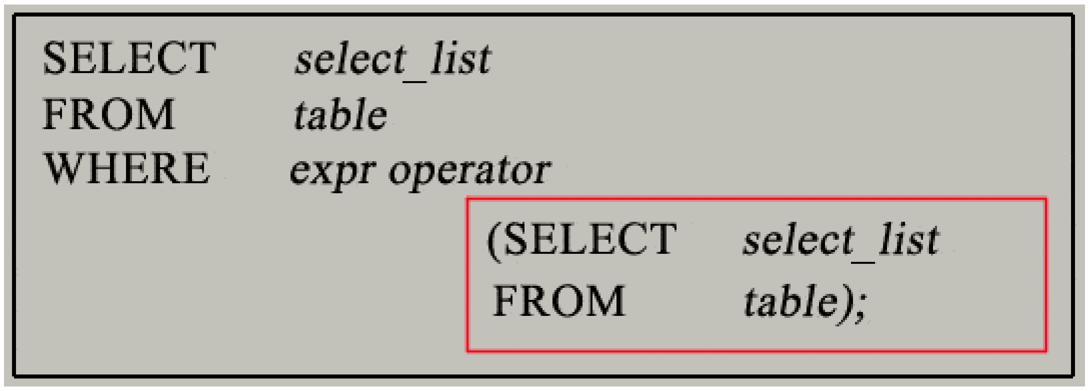
A subquery may occur in :

- A SELECT clause
- A FROM clause
- A WHERE clause

The subquery can be nested inside a SELECT, INSERT, UPDATE, or DELETE statement or inside another subquery.

- A subquery is usually added within the WHERE Clause of another SQL SELECT statement.
- You can use the comparison operators, such as >, <, or =. The comparison operator can also be a multiple-row operator, such as IN, ANY, or ALL.
- A subquery is also called an inner query or inner select, while the statement containing a subquery is also called an outer query or outer select.
- The inner query executes first before its parent query so that the results of an inner query can be passed to the outer query.
- You can use a subquery in a SELECT, INSERT, DELETE, or UPDATE statement to perform the following tasks:
    - Compare an expression to the result of the query.
    - Determine if an expression is included in the results of the query.
    - Check whether the query selects any rows.

Syntax :

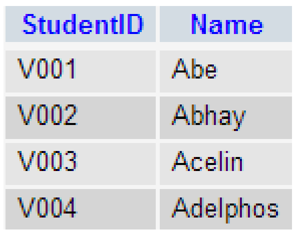


```
SELECT    select_list
FROM      table
WHERE     expr operator
                        (SELECT    select_list
                         FROM      table);
```

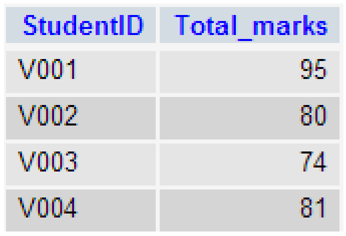
- The subquery (inner query) executes once before the main query (outer query) executes.
- The main query (outer query) use the subquery result.

SQL Subqueries Example :

In this section, you will learn the requirements of using subqueries. We have the following two tables 'student' and 'marks' with common field 'StudentID'.



StudentID	Name
V001	Abe
V002	Abhay
V003	Acelin
V004	Adelphos



StudentID	Total_marks
V001	95
V002	80
V003	74
V004	81

student marks Now we want to write a query to identify all students who get better marks than that of the student who's StudentID is 'V002', but we do not know the marks of 'V002'.

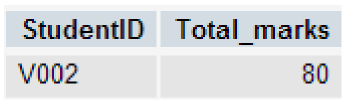
- To solve the problem, we require two queries. One query returns the marks (stored in Total_marks field) of 'V002' and a second query identifies the students who get better marks than the result of the first query.

First query:

```
SELECT *
FROM `marks`
WHERE studentid = 'V002';
```

Copy

Query result:



StudentID	Total_marks
V002	80

The result of the query is 80.

- Using the result of this query, here we have written another query to identify the students who get better marks than 80. Here is the query :

Second query:

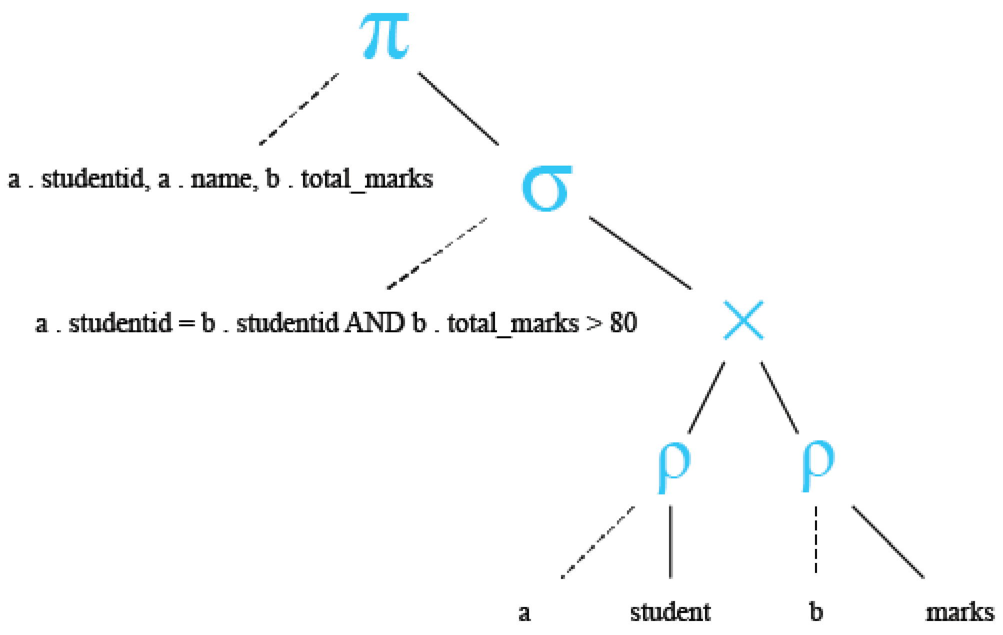
```
SELECT a.studentid, a.name, b.total_marks
FROM student a, marks b
WHERE a.studentid = b.studentid
AND b.total_marks >80;
```

Copy

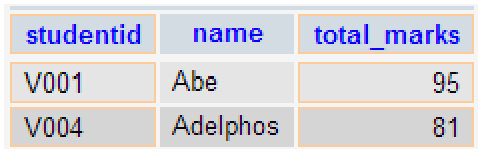
Relational Algebra Expression:

$$\pi_{a . studentid , a . name , b . total\_marks}$$
$$\sigma_{a . studentid = b . studentid \text{ AND } b . total\_marks > 80}$$
$$(\rho_a \, student \times$$
$$\rho_b \, marks)$$

Relational Algebra Tree:


π
a . studentid, a . name, b . total_marks
σ
a . studentid = b . studentid AND b . total_marks > 80
×
ρ
ρ
a
student
b
marks

Query result:



studentid	name	total_marks
V001	Abe	95
V004	Adelphos	81

Above two queries identified students who get the better number than the student who's StudentID is 'V002' (Abhay).

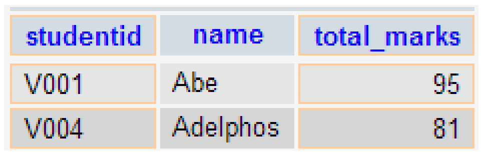
You can combine the above two queries by placing one query inside the other. The subquery (also called the 'inner query') is the query inside the parentheses. See the following code and query result :

SQL Code:

```
SELECT a.studentid, a.name, b.total_marks
FROM student a, marks b
WHERE a.studentid = b.studentid AND b.total_marks >
(SELECT total_marks
FROM marks
WHERE studentid =  'V002');
```

Copy

Query result:



studentid	name	total_marks
V001	Abe	95
V004	Adelphos	81

Pictorial Presentation of SQL Subquery:

Subqueries: General Rules

A subquery SELECT statement is almost similar to the SELECT statement and it is used to begin a regular or outer query. Here is the syntax of a subquery:

Syntax:

```
(SELECT [DISTINCT] subquery_select_argument
FROM {table_name | view_name}
{table_name | view_name} ...
[WHERE search_conditions]
[GROUP BY aggregate_expression [, aggregate_expression] ...]
[HAVING search_conditions])
```

Subqueries: Guidelines

There are some guidelines to consider when using subqueries :

- A subquery must be enclosed in parentheses.
- A subquery must be placed on the right side of the comparison operator.
- Subqueries cannot manipulate their results internally, therefore ORDER BY clause cannot be added into a subquery. You can use an ORDER BY clause in the main SELECT statement (outer query) which will be the last clause.
- Use single-row operators with single-row subqueries.
- If a subquery (inner query) returns a null value to the outer query, the outer query will not return any rows when using certain comparison operators in a WHERE clause.

Type of Subqueries

- Single row subquery : Returns zero or one row.
- Multiple row subquery : Returns one or more rows.
- Multiple column subqueries : Returns one or more columns.
- Correlated subqueries : Reference one or more columns in the outer SQL statement. The subquery is known as a correlated subquery because the subquery is related to the outer SQL statement.
- Nested subqueries : Subqueries are placed within another subquery.

In the next session, we have thoroughly discussed the above topics. Apart from the above type of subqueries, you can use a subquery inside INSERT, UPDATE and DELETE statement. Here is a brief discussion :

Subqueries with INSERT statement

INSERT statement can be used with subqueries. Here are the syntax and an example of subqueries using INSERT statement.

Syntax:

```
INSERT INTO table_name [ (column1 [, column2 ]) ]
```

```
SELECT [ *|column1 [, column2 ]
FROM table1 [, table2 ]
[ WHERE VALUE OPERATOR ];
```

If we want to insert those orders from 'orders' table which have the advance_amount 2000 or 5000 into 'neworder' table the following SQL can be used:

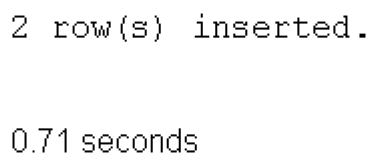
Sample table: orders

SQL Code:

```
INSERT INTO neworder
SELECT * FROM  orders
WHERE advance_amount in(2000,5000);
```

Copy

Output:



```
2 row(s) inserted.

0.71 seconds
```

Subqueries with UPDATE statement

In a UPDATE statement, you can set new column value equal to the result returned by a single row subquery. Here are the syntax and an example of subqueries using UPDATE statement.

Syntax:

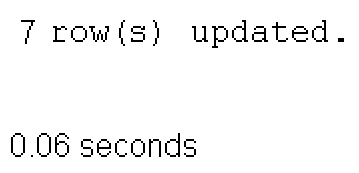
```
UPDATE table  SET column_name = new_value
[ WHERE OPERATOR [ VALUE ]
(SELECT COLUMN_NAME
FROM TABLE_NAME)
[ WHERE) ]
```

SQL Code:

```
UPDATE neworder
SET ord_date='15-JAN-10'
WHERE ord_amount-advance_amount<
(SELECT MIN(ord_amount) FROM orders);
```

Copy

Output:



```
7 row(s) updated.

0.06 seconds
```

To see more details of subqueries using UPDATE statement click here.

Subqueries with DELETE statement

DELETE statement can be used with subqueries. Here are the syntax and an example of subqueries using DELETE statement.

Syntax:

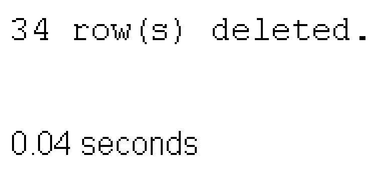
```
DELETE FROM TABLE_NAME
[ WHERE OPERATOR [ VALUE ]
(SELECT COLUMN_NAME
FROM TABLE_NAME)
[ WHERE) ]
```

SQL Code:

```
DELETE FROM neworder
WHERE advance_amount<
(SELECT MAX(advance_amount) FROM orders);
```

Copy

Output:



```
34 row(s) deleted.

0.04 seconds
```

Further Reading:-

- Ramez Elmasriand Shamkant B. Navathe, “Fundamentals of Database System”, The Benjamin

/Cummings Publishing Co

- C.J.Date, “AnIntroduction to Database Systems”, Addison Wesley

Video Link - https://www.youtube.com/watch?v=JksrTuEVEPk

Experiment 2.4

CO MAPPED- CO4

CO4: Implement the packages, procedures and triggers

AIM-To implement the concept of indexes, cursors, triggers and views

SQL CREATE INDEX Statement

The CREATE INDEX statement is used to create indexes in tables.

Indexes are used to retrieve data from the database more quickly than otherwise. The users cannot see the indexes, they are just used to speed up searches/queries.

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.

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.

MS Access:

DROP INDEX *index_name* ON *table_name*;

SQL Server:

DROP INDEX *table_name*.*index_name*;

DB2/Oracle:

DROP INDEX *index_name*;

MySQL:

ALTER TABLE *table_name*

DROP INDEX *index_name*;

A **cursor** is a pointer to this context area. PL/SQL controls the context area through a cursor. A cursor holds the rows (one or more) returned by a SQL statement. The set of rows the cursor holds is referred to as the **active set**.

You can name a cursor so that it could be referred to in a program to fetch and process the rows returned by the SQL statement, one at a time. There are two types of cursors −

- Implicit cursors
- Explicit cursors

Implicit Cursors

Implicit cursors are automatically created by Oracle whenever an SQL statement is executed, when there is no explicit cursor for the statement. Programmers cannot control the implicit cursors and the information in it.

Whenever a DML statement (INSERT, UPDATE and DELETE) is issued, an implicit cursor is associated with this statement. For INSERT operations, the cursor holds the data that needs to be inserted. For UPDATE and DELETE operations, the cursor identifies the rows that would be affected.

In PL/SQL, you can refer to the most recent implicit cursor as the **SQL cursor**, which always has attributes such as **%FOUND, %ISOPEN, %NOTFOUND**, and **%ROWCOUNT**. The SQL cursor has additional attributes, **%BULK_ROWCOUNT** and **%BULK_EXCEPTIONS**, designed for use with the **FORALL** statement. The following table provides the description of the most used attributes −

S.No	Attribute & Description
1	**%FOUND** Returns TRUE if an INSERT, UPDATE, or DELETE statement affected one or more rows or a SELECT INTO statement returned one or more rows. Otherwise, it returns FALSE.
2	**%NOTFOUND** The logical opposite of %FOUND. It returns TRUE if an INSERT, UPDATE, or DELETE statement affected no rows, or a SELECT INTO statement returned no rows. Otherwise, it returns FALSE.
3	**%ISOPEN** Always returns FALSE for implicit cursors, because Oracle closes the SQL cursor automatically after executing its associated SQL statement.
4	**%ROWCOUNT** Returns the number of rows affected by an INSERT, UPDATE, or DELETE statement, or returned by a SELECT INTO statement.

Any SQL cursor attribute will be accessed as **sql%attribute_name** as shown below in the example.

Example

We will be using the CUSTOMERS table we had created and used in the previous chapters.

```
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 |
+----+----------+-----+-----------+----------+
```

The following program will update the table and increase the salary of each customer by 500 and use the **SQL%ROWCOUNT** attribute to determine the number of rows affected −

```
DECLARE
  total_rows number(2);
BEGIN
  UPDATE customers
  SET salary = salary + 500;
  IF sql%notfound THEN
    dbms_output.put_line('no customers selected');
  ELSIF sql%found THEN
```

```
      total_rows := sql%rowcount;
      dbms_output.put_line( total_rows || ' customers selected ');
   END IF;
END;
/
```

When the above code is executed at the SQL prompt, it produces the following result −

```
6 customers selected

PL/SQL procedure successfully completed.
```

If you check the records in customers table, you will find that the rows have been updated −

```
Select * from customers;

+----+----------+-----+-----------+----------+
| ID | NAME | AGE | ADDRESS | SALARY |
+----+----------+-----+-----------+----------+
| 1 | Ramesh | 32 | Ahmedabad | 2500.00 |
| 2 | Khilan | 25 | Delhi | 2000.00 |
| 3 | kaushik | 23 | Kota | 2500.00 |
| 4 | Chaitali | 25 | Mumbai | 7000.00 |
| 5 | Hardik | 27 | Bhopal | 9000.00 |
| 6 | Komal | 22 | MP | 5000.00 |
+----+----------+-----+-----------+----------+
```

Explicit Cursors

Explicit cursors are programmer-defined cursors for gaining more control over the **context area**. An explicit cursor should be defined in the declaration section of the PL/SQL Block. It is created on a SELECT Statement which returns more than one row.

The syntax for creating an explicit cursor is −

```
CURSOR cursor_name IS select_statement;
```

Working with an explicit cursor includes the following steps −

- Declaring the cursor for initializing the memory
- Opening the cursor for allocating the memory
- Fetching the cursor for retrieving the data
- Closing the cursor to release the allocated memory

Declaring the Cursor

Declaring the cursor defines the cursor with a name and the associated SELECT statement. For example −

```
CURSOR c_customers IS
   SELECT id, name, address FROM customers;
```

Opening the Cursor

Opening the cursor allocates the memory for the cursor and makes it ready for fetching the rows returned by the SQL statement into it. For example, we will open the above defined cursor as follows −

```
OPEN c_customers;
```

Fetching the Cursor

Fetching the cursor involves accessing one row at a time. For example, we will fetch rows from the above-opened cursor as follows −

```
FETCH c_customers INTO c_id, c_name, c_addr;
```

Closing the Cursor

Closing the cursor means releasing the allocated memory. For example, we will close the above-opened cursor as follows −

```
CLOSE c_customers;
```

Example

Following is a complete example to illustrate the concepts of explicit cursors &minua;

```
DECLARE
   c_id customers.id%type;
   c_name customers.name%type;
   c_addr customers.address%type;
   CURSOR c_customers is
      SELECT id, name, address FROM customers;
BEGIN
   OPEN c_customers;
   LOOP
   FETCH c_customers into c_id, c_name, c_addr;
      EXIT WHEN c_customers%notfound;
      dbms_output.put_line(c_id || ' ' || c_name || ' ' || c_addr);
   END LOOP;
   CLOSE c_customers;
END;
/
```

When the above code is executed at the SQL prompt, it produces the following result −

```
1 Ramesh Ahmedabad
2 Khilan Delhi
3 kaushik Kota
4 Chaitali Mumbai
5 Hardik Bhopal
6 Komal MP

PL/SQL procedure successfully completed.
```

Triggers are stored programs, which are automatically executed or fired when some events occur. Triggers are, in fact, written to be executed in response to any of the following events −

- A **database manipulation (DML)** statement (DELETE, INSERT, or UPDATE)
- A **database definition (DDL)** statement (CREATE, ALTER, or DROP).
- A **database operation** (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

Triggers can be defined on the table, view, schema, or database with which the event is associated.

Benefits of Triggers

Triggers can be written for the following purposes −

- Generating some derived column values automatically
- Enforcing referential integrity
- Event logging and storing information on table access
- Auditing
- Synchronous replication of tables
- Imposing security authorizations
- Preventing invalid transactions

Creating Triggers

The syntax for creating a trigger is −

```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
DECLARE
   Declaration-statements
BEGIN
   Executable-statements
EXCEPTION
   Exception-handling-statements
END;
```

Where,

- CREATE [OR REPLACE] TRIGGER trigger_name − Creates or replaces an existing trigger with the *trigger_name*.
- {BEFORE | AFTER | INSTEAD OF} − This specifies when the trigger will be executed. The INSTEAD OF clause is used for creating trigger on a view.
- {INSERT [OR] | UPDATE [OR] | DELETE} − This specifies the DML operation.
- [OF col_name] − This specifies the column name that will be updated.
- [ON table_name] − This specifies the name of the table associated with the trigger.
- [REFERENCING OLD AS o NEW AS n] − This allows you to refer new and old values for various DML statements, such as INSERT, UPDATE, and DELETE.
- [FOR EACH ROW] − This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.
- WHEN (condition) − This provides a condition for rows for which the trigger would fire. This clause is valid only for row-level triggers.

Example

To start with, we will be using the CUSTOMERS table we had created and used in the previous chapters

−

```
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 |
+----+----------+-----+-----------+----------+
```

The following program creates a **row-level** trigger for the customers table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old values and new values −

```
CREATE OR REPLACE TRIGGER display_salary_changes
BEFORE DELETE OR INSERT OR UPDATE ON customers
FOR EACH ROW
WHEN (NEW.ID > 0)
DECLARE
   sal_diff number;
BEGIN
   sal_diff := :NEW.salary  - :OLD.salary;
   dbms_output.put_line('Old salary: ' || :OLD.salary);
   dbms_output.put_line('New salary: ' || :NEW.salary);
   dbms_output.put_line('Salary difference: ' || sal_diff);
END;
/
```

When the above code is executed at the SQL prompt, it produces the following result −

```
Trigger created.
```

The following points need to be considered here −

- OLD and NEW references are not available for table-level triggers, rather you can use them for record-level triggers.
- If you want to query the table in the same trigger, then you should use the AFTER keyword, because triggers can query the table or change it again only after the initial changes are applied and the table is back in a consistent state.
- The above trigger has been written in such a way that it will fire before any DELETE or INSERT or UPDATE operation on the table, but you can write your trigger on a single or multiple operations, for example BEFORE DELETE, which will fire whenever a record will be deleted using the DELETE operation on the table.

Triggering a Trigger

Let us perform some DML operations on the CUSTOMERS table. Here is one INSERT statement, which will create a new record in the table −

```
INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY)
VALUES (7, 'Kriti', 22, 'HP', 7500.00 );
```

When a record is created in the CUSTOMERS table, the above create trigger, **display_salary_changes** will be fired and it will display the following result −

```
Old salary:
New salary: 7500
Salary difference:
```

Because this is a new record, old salary is not available and the above result comes as null. Let us now perform one more DML operation on the CUSTOMERS table. The UPDATE statement will update an existing record in the table −

```
UPDATE customers
SET salary = salary + 500
WHERE id = 2;
```

When a record is updated in the CUSTOMERS table, the above create trigger, **display_salary_changes** will be fired and it will display the following result −

```
Old salary: 1500
New salary: 2000
Salary difference: 500
```

SQL CREATE VIEW Statement

In SQL, a view is a virtual table based on the result-set of an SQL statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

You can add SQL statements and functions to a view and present the data as if the data were coming from one single table.

A view is created with the CREATE VIEW statement.

CREATE VIEW Syntax

CREATE VIEW *view_name* AS

SELECT *column1*, *column2*, ...

FROM *table_name*

WHERE *condition*;

Note: A view always shows up-to-date data! The database engine recreates the view, every time a user queries it.

SQL CREATE VIEW Examples

The following SQL creates a view that shows all customers from Brazil:

Example

```
CREATE VIEW [Brazil Customers] AS

SELECT CustomerName, ContactName

FROM Customers

WHERE Country = 'Brazil';
```

We can query the view above as follows:

Example

```
SELECT * FROM [Brazil Customers];
```

The following SQL creates a view that selects every product in the "Products" table with a price higher than the average price:

Example

```
CREATE VIEW [Products Above Average Price] AS

SELECT ProductName, Price

FROM Products

WHERE Price > (SELECT AVG(Price) FROM Products);
```

We can query the view above as follows:

Example

```
SELECT * FROM [Products Above Average Price];
```

FurtherReading-

- Ramez Elmasriand Shamkant B. Navathe, “Fundamentals of Database System”, The Benjamin /Cummings Publishing Co

- C.J.Date, “AnIntroduction to Database Systems”, Addison Wesley

Video Links -

https://www.youtube.com/watch?v=INw_KGjyfDw

https://www.youtube.com/watch?v=7XBOPm5WbtQ

Experiment 3.1

CO MAPPED- CO5
CO5: Understand the concept of Transaction Processing and Concurrency Control

Aim- Design a case study for Company database/ Hospital Management System/ Railway Reservation System