TITLE: Experiment Write-up (EW)
Doc. No.: DBMS&SQL/SOP/EW

Subject: Database Management System & SQL Laboratory

EXPERIMENT NO.: 01

TITLE: Draw E-R diagram for a given scenario (eg. bank, college etc.)

LEARNING OBJECTIVES:

1. To study the fundamental concepts of database management.
2. To learn the basic issues of transaction processing and concurrency control.
3. To learn a powerful, flexible and scalable general-purpose distributed database.

THEORY:

Database:
If the data is stored in a well-organized manner, then it allows user to access and search data very easily. The **Database** is used to store information in a well-organized manner (i.e. Tables) useful to an organization. A collection of records or files of information grouped together is called as **Database**. **Record** is a collection of related data. (ex. A person's name, telephone number, and address).

DBMS:
Data storage is the most important thing while programming. This facility ensures access to data at any moment when user runs the program. (Eg. In C programming the data storage facility is provided by an operating system. There is no special software to store the data.) When amount of data to be stored becomes large, it becomes difficult to manipulate the data operations like storing and searching. The **D**ata**B**ase **M**anagement **S**ystem is the set of the interrelated data and the set of programs to operate on it.

The collection of interrelated data is known as **Database**.**The basic goal of Database Management System is to provide the way to store and retrieve the data in efficient and convenient manner**. When the data is in large amount is to be manipulated, we need a DBMS to manage it

Functions of DBMS :

1. Data Storage Management: It provides a mechanism for management of permanent storage of the data. The internal schema defines how the data should be stored by the storage management mechanism and the storage manager interfaces with the operating system to access the physical storage.

2. Data Manipulation Management: A DBMS facilitates users with the ability to retrieve, update and delete existing data in the database.

3. Data Definition Services: The DBMS accepts the data definitions such as external schema, the conceptual schema, the internal schema, and all the associated mappings in source form.

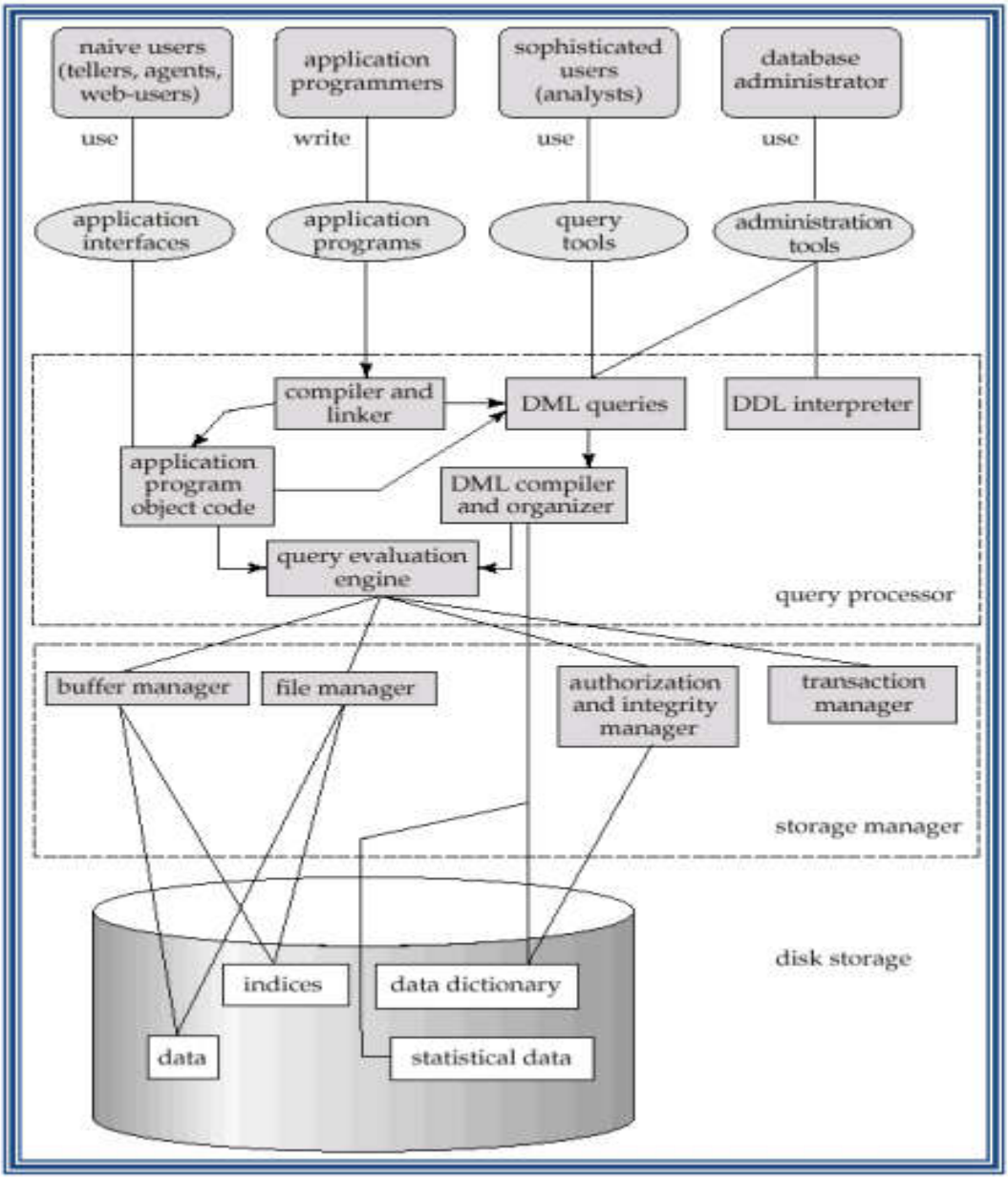
4. Data Dictionary/System Catalog Management: The DBMS provides a data dictionary or system catalog function in which descriptions of data items are stored and which is accessible to users.

5.Authorization / Security Management: The DBMS protects the database against unauthorized access, either international or accidental. It furnishes mechanism to ensure that only authorized users an access to the database.

6. Backup and Recovery Management: The DBMS provides mechanisms for backing up data periodically and recovering from different types of failures. This prevents the loss of data.

7. Concurrency Control Service: Since DBMSs support sharing of data among multiple users, they must provide a mechanism for managing concurrent access to the database. DBMSs ensure that the database kept in consistent state and that integrity of the data is preserved.

8. Transaction Management: A transaction is a series of database operations, carried out by a single user or application program, which accesses or changes the contents of the database. Therefore, a DBMS must provide a mechanism to ensure either that all the updates corresponding to a given transactions are made or that none of them is made.



Components of DBMS and Overall Structure of DBMS :

Components of DBMS are broadly classified as follows :

1) Storage Manager :

In overall structure of the DBMS, the storage manager can be referred as a program or module. The storage manager is very important as very large databases require the storage space in gigabytes or even in terabytes. Storage manager provides an interface between the

lower level data stored in the database and application programs. It also acts as an interface between database and queries submitted by the system. The storage manager takes the responsibility to interact with the file manager. The storage manager compiles and translates the commands of DMLs into low level file system commands, then they are executed. Thus the storage manager is responsible for storing, retrieving and updating data in the database. The components of storage manager are :

a) Authorization and Integrity Manager : The major functions of this manager are to check the integrity constraints and check the authority of the user to access the data. Integrity constraints are the restrictions on the data given by the users. So the satisfaction level of the integrity constraints is checked. Another function is to check the authorities of the users to access, retrieve or update the data form or to the database.

b) Transaction Manager : A transaction is collection of operations that performs a single logical function in a database application. The transaction manager ensures that the database remains in a consistent (correct) state despite system failures and that concurrent execution proceed without conflicting.

c) File Manager : File manager manages the allocation of space on disk. It allocates required space to the files which are stored on the storage disk. It also takes care of the data structure which represents the information stored on the disk.

d) Buffer Manager : Buffer manager is the temporary memory which is used to transfer data or information from one device to other. Buffer manager manages the transfer of the data from the disk to the main memory. Some complicated situations occur if the database is very large and main memory is less. Such a type of critical situation is handled by the buffer manager.

2) Data Structures :

The storage manager uses some data structures to represent as a part of physical system implementations. These data structures are :

a) Data Files : It stores the actual data itself.

b) Data Dictionary : This the data about the data i.e. metadata. This is about the structure of the database in the system.

c) Indices : Indices are used for the fast search of the data in the database. Using indexing it becomes easy to find out the expected data. If we index the file, the records gets arranged in the particular order.

d) Statistical Data : It stores statistical information about the data in the database. This information is used by query processor to select efficient ways to execute the query.

3) Query Processor :

The Query Processor includes :

i) DDL Interpreter
ii) DML Compiler
iii) Query Evaluation Engine
iv) DML Compiler and Organizer
v) Compiler and Linker
vi) Application Program Object Code

i) DDL Interpreter : The DDL Interpreter interprets or reads the DDL statements and records the definitions of the file and adds it in the data dictionary. A user of the DDL script would ask the DDL interpreter to load the script and provide the interpreter with a block of data (a file) from which user wishes to retrieve data.

ii) DML Compiler : DML compiler translates DML statements into the low level instructions. Low level instructions are understandable by query evaluation engine. DML compiler generates the evaluation plans which are chosen by query evaluation engine. A query evaluation is done using different alternatives. The compiler then does query optimization i.e. it chooses a proper plan which is appropriate.

iii) Query Evaluation Engine : The Query Evaluation Engine executes the low level instructions generated by the DML compiler.

iv) DML Compiler and Organizer : This compiles the DMLs into lower level instructions and arranges or organizes the output.

v) Compiler and Linker : Compiler compiles the instructions given by the user and the linker links them with the standard library functions or methods or statements provided by the query language. Compilation is the process of converting programs written in high level language to

low level language.

vi) Application Program Object Code : Source code is the original program written. Application is viewed after execution of the source code. But when source code gets executed, first it gets converted into object code which is understandable by computer system.

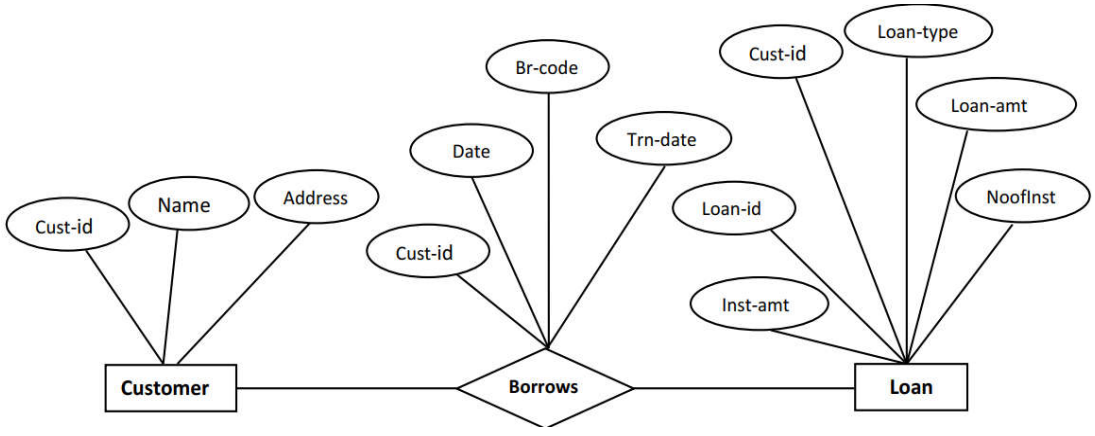
E-R Model

The **Entity-Relationship(E-R) model** was originally proposed by Peter in 1976 for database design. E-R model is a conceptual data model that views the real world as **entities** and **relationships**. It is a **Top down approach** of database designing.
Relationship is a natural association that exists between one or more entities.

Features of E-R Model :

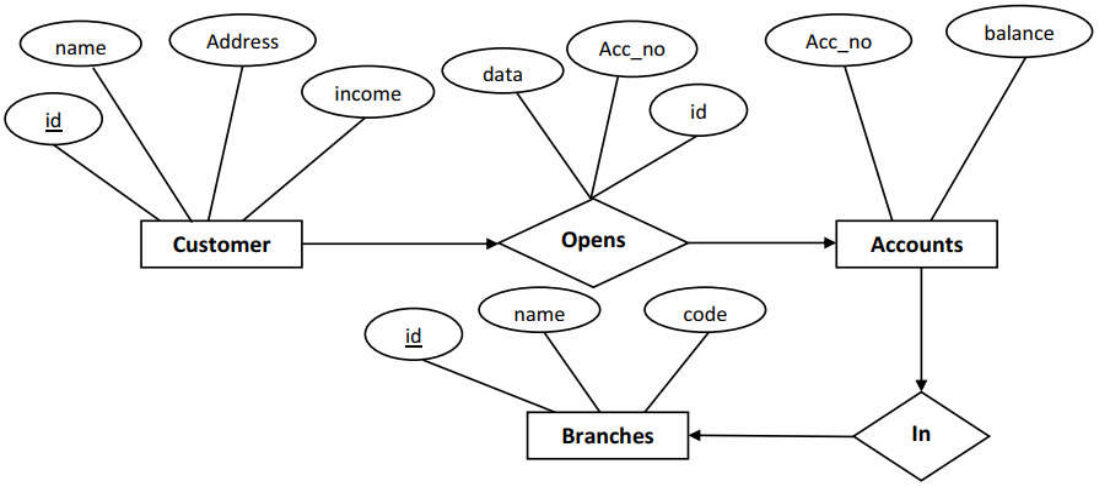
1) E-R diagram used for representing E-R model can be easily converted into Relations(tables) in Relational model.
2) The E-R model is used for the purpose of good database design by the database developer so to use that data model in various DBMS.
3) It is helpful as a problem decomposition tool as it shows the entities and relationships between those entities.
4) It is inherently an iterative process.
5) It is very simple and easy to understand by various types of users and designers because specific standards are used for their presentation.

E-R Diagram for Bank & Customer:

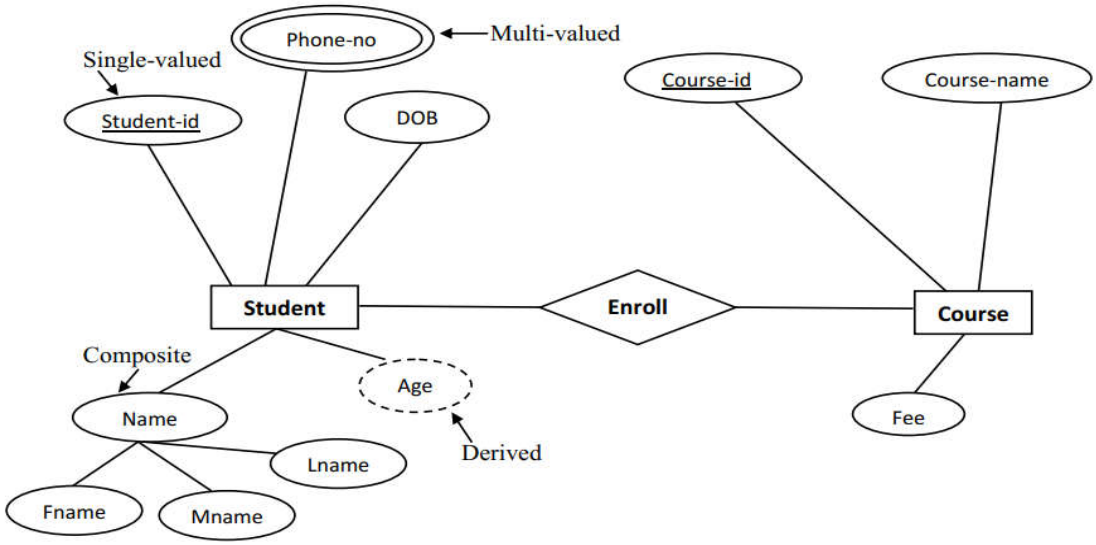


TITLE: Experiment Write-up (EW)
Doc. No.: DBMS&SQL/SOP/EW

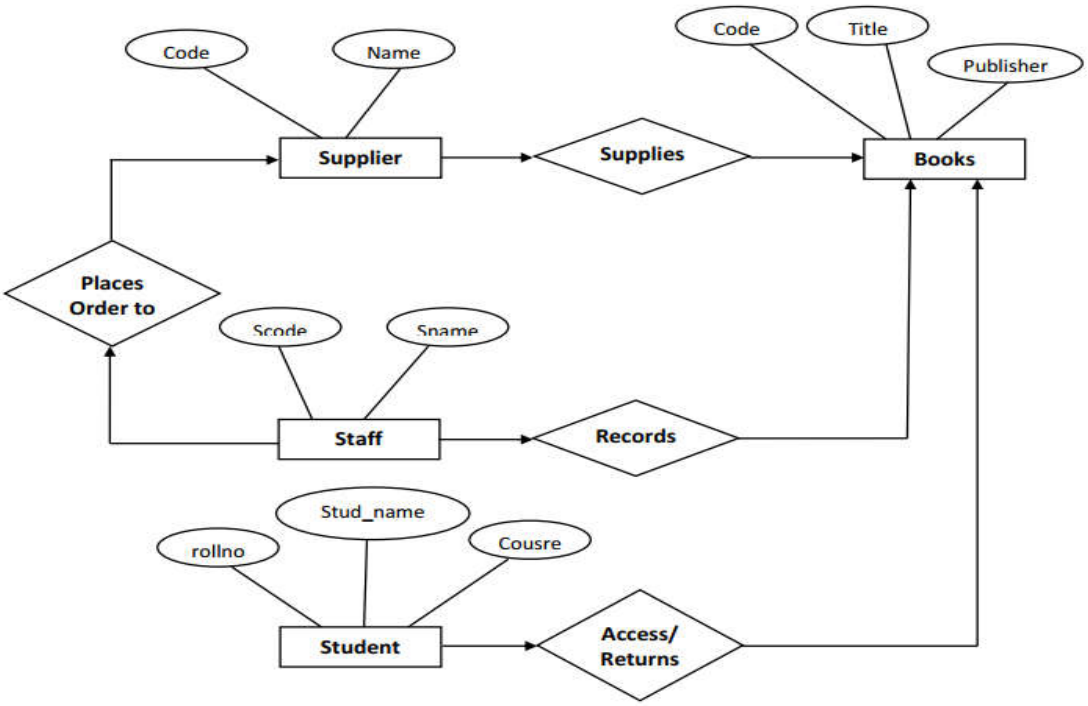
E-R Diagram for Customer, Account, Branch



E-R Diagram for Student, Course, College:



TITLE: Experiment Write-up (EW)
Doc. No.: DBMS&SQL/SOP/EW



Note: Represent any one ER diagram into relation table for any given scenario.

References for Theory:

1. Silberschatz A., Korth H., Sudarshan S., "Database System Concepts", MGH
2. Connally T, Begg C., "Database Systems", Pearson Education
3. Raghurama Krishan, "Database Management Systems", McGrawHill
4. S.K.Singh, "Database Systems : Concepts, Design and Application", Pearson

CONCLUSION:_____

_____

_____

Prepared by

Mr. N. I. Bhopale
(Subject Teacher)

Verified by

Dr. B. S. Agarkar
(HOD, Deptt. of ECE)

TITLE: Experiment Write-up (EW)
Doc. No.: DBMS&SQL/SOP/EW

Subject: Database Management System & SQL Laboratory

EXPERIMENT NO.: 02

TITLE: For a given set of relation schemes, create tables and perform the following Simple Queries, Simple Queries with Aggregate functions, Queries with Aggregate functions (group by and having clause).

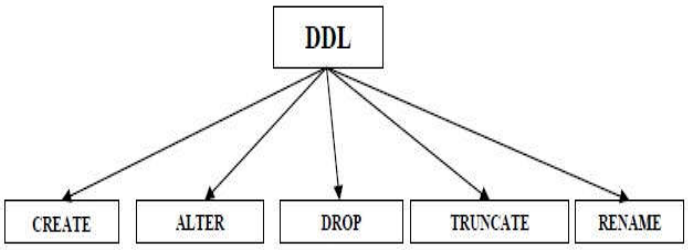
LEARNING OBJECTIVES:

1. To study the fundamental concepts of database management.
2. To learn the basic issues of transaction processing and concurrency control.
3. To learn a powerful, flexible and scalable general-purpose distributed database.

THEORY:

DDL

The DDL part of SQL permits database tables to be created or deleted. DDL is used to specify the structure of the database. We can also define indexes(Keys), specify links between tables and impose constraints between database tables.



CREATE

The CREATE TABLE command is used to define the table with its name. Also the list of fields, their data types as well as sizes can be specified using this command. ***Syntax:***
CREATE TABLE <table-name> (<column name1> <data type>(<size>), <column name2>

<data type>(<size>)…….);

ALTER

Sometimes it is necessary to change the basic structure of the table.

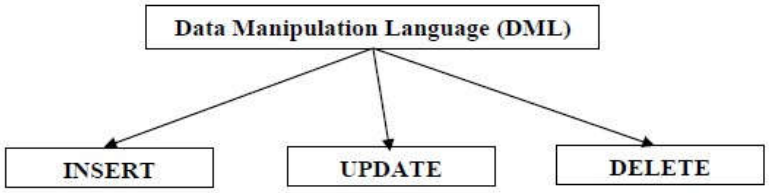
ALTER TABLE command is used to make the changes or modification in the table structure. With the *alter table* command adding and deleting of the column is possible.

***Syntax*:**

ALTER TABLE <table-name> ADD (<column-name> <data type>(<size>));

DML

The DML is used to manipulate i.e. add, modify or delete data from table. DML involves different commands to update, delete, insert data into tables. It also involves queries. Query is the statement requesting data from database.



INSERT

After creating table successfully, now records can be entered. The INSERT INTO command is used to insert new rows or records into a table.

Syntax:

INSERT INTO <table-name> VALUES (value1,value2,…..,);

UPDATE

The **UPDATE** statement modifies the values of one or more columns in selected rows of a table. Either all rows will be updated or selected rows can be updated using WHERE clause.

Syntax:

UPDATE <table-name> SET <column-name1> = <expression1> , <columnname2> = <expression2> WHERE <column-name>= value

DELETE

The **DELETE** command is used to delete one row or even all the rows from the table.

Syntax:

DELETE FROM <table-name> WHERE <condition>

SQL Aggregate Functions:-

- **AVG()** - Returns the average value
- **COUNT()** - Returns the number of rows
- **FIRST()** - Returns the first value
- **LAST()** - Returns the last value
- **MAX()** - Returns the largest value

- **MIN()** - Returns the smallest value
- **SUM()** - Returns the sum

Example:

mysql> show databases;

```
+--------------------+
| Database           |
+--------------------+
| information_schema |
| buynsale           |
| car                |
| college            |
| college1           |
| db                 |
| dummy              |
| dummy123           |
| employee           |
| learn              |
| localdb            |
| localdb1           |
| localdb2           |
| localdb3           |
| mysql              |
| mytransaction      |
| new_dummy          |
| newdummy           |
| performance_schema |
| quiz               |
| sakila             |
| sample             |
| sanjivani          |
| sanjivani1         |
| sanjivani123       |
| sanjivani2         |
| sanjivanidb        |
| scoe               |
| student            |
| sys                |
| test               |
| test1              |
| trade              |
| transaction        |
| userdb             |
| world              |
| xyz                |
+--------------------+
```

```
mysql> create database school; Query OK, 1
row affected (0.01 sec)
mysql> use school;
Database changed
mysql> create table student (rn int primary key not null,
    -> name varchar(22) not null,
    -> marks varchar(22) not null);
Query OK, 0 rows affected (0.06 sec)

mysql> insert into student (rn,name,marks) values
    -> (1,"sarthak",55),
    -> (2,"nilesh",77),
    -> (3,"shruti",66);
Query OK, 3 rows affected (0.00 sec)

 mysql> select * from student;
+----+---------+-------+
| rn | name    | marks |
+----+---------+-------+
|  1 | sarthak | 55    |
|  2 | nilesh  | 77    |
|  3 | shruti  | 66    |
+----+---------+-------+

mysql> select * from student order by rn desc;
+----+---------+-------+
| rn | name    | marks |
+----+---------+-------+
|  3 | shruti  | 66    |
|  2 | nilesh  | 77    |
|  1 | sarthak | 55    |
+----+---------+-------+

mysql> select * from student order by rn asc;
+----+---------+-------+
| rn | name    | marks |
+----+---------+-------+
|  1 | sarthak | 55    |
|  2 | nilesh  | 77    |
|  3 | shruti  | 66    |
+----+---------+-------+

mysql> select distinct(name) from student;
```

```
+---------+
| name    |
+---------+
| sarthak |
| nilesh  |
| shruti  |
+---------+
```

mysql> select * from student;

```
+----+---------+-------+
| rn | name    | marks |
+----+---------+-------+
|  1 | sarthak | 55    |
|  2 | nilesh  | 77    |
|  3 | shruti  | 66    |
|  4 | nilesh  | 88    |
+----+---------+-------+
```

mysql> select min(marks) from student;

```
+------------+
| min(marks) |
+------------+
| 55         |
+------------+
```

mysql> select max(marks),name,rn from student;

```
+------------+---------+----+
| max(marks) | name    | rn |
+------------+---------+----+
| 88         | sarthak |  1 |
+------------+---------+----+
```

mysql> select avg(marks) from student;

```
+------------+
| avg(marks) |
+------------+
|       71.5 |
+------------+
```

mysql> insert into student (rn,name,marks) values(4,"abc",66);
Query OK, 1 row affected (0.02 sec)

```
mysql> select * from student;
+----+---------+-------+
| rn | name    | marks |
+----+---------+-------+
|  1 | sarthak | 55    |
|  2 | nilesh  | 77    |
|  3 | shruti  | 66    |
|  4 | abc     | 66    |
+----+---------+-------+
```

mysql> alter table student rename stude;
Query OK, 0 rows affected (0.03 sec)

mysql> select * from stude;

```
+----+---------+-------+
| rn | name    | marks |
+----+---------+-------+
|  1 | sarthak | 55    |
|  2 | nilesh  | 77    |
|  3 | shruti  | 66    |
+----+---------+-------+
```

mysql> update stude set marks=89 where rn=1; Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql> select * from stude;

```
+----+---------+-------+
| rn | name    | marks |
+----+---------+-------+
|  1 | sarthak | 89    |
|  2 | nilesh  | 77    |
|  3 | shruti  | 66    |
+----+---------+-------+
```

References for Theory:

1. Silberschatz A., Korth H., Sudarshan S., "Database System Concepts", MGH
2. Connally T, Begg C., "Database Systems", Pearson Education
3. Raghurama Krishan, "Database Management Systems", McGrawHill
4. S.K.Singh, "Database Systems : Concepts, Design and Application", Pearson

CONCLUSION:_____

TITLE: Experiment Write-up (EW)
Doc. No.: DBMS&SQL/SOP/EW

Subject: Database Management System & SQL Laboratory

EXPERIMENT NO.:3

TITLE: Implement joins, aggregate functions and Views for DB.

LEARNING OBJECTIVES:

1. To study the fundamental concepts of database management.
2. To learn the basic issues of transaction processing and concurrency control.

THEORY: Views: Views in SQL are kind of virtual tables. A view also has rows and columns as they are in a real table in the database. We can create a view by selecting fields from one or more tables present in the database. A view can either have all the rows of a table or specific rows based on criterion condition. A view is nothing more than a SQL statement that is stored in the database with an associated name. A view is actually a composition of a table in the form of a predefined SQL query.

Operations on Views:

Sample Table: Student Details

S_ID	NAME	ADDRESS
1	Harsh	Kolkata
2	Ashish	Durgapur
3	Pratik	Delhi
4	Dhanraj	Bihar
5	Ram	Rajasthan

Student Marks

ID	NAME	MARKS	AGE
1	Harsh	90	19
2	Suresh	50	20
3	Pratik	80	19
4	Dhanraj	95	21
5	Ram	85	18

Creating Views:

We can create View using CREATE VIEW statement. A View can be created from a single table or multiple tables.

Syntax:

CREATE VIEW view_name AS

SELECT column1, column2.....

FROM table_name

WHERE condition;

view_name: Name for the View

table_name: Name of the table

condition: Condition to select rows

Examples:

Example 1.

Creating View from a single table:

In this example we will create a View named DetailsView from the table StudentDetails.

Query:

CREATE VIEW DetailsView AS

SELECT NAME, ADDRESS

FROM StudentDetails

WHERE S_ID < 5;

To see the data in the View, we can query the view in the same manner as we query a table.

SELECT * FROM DetailsView;

Output:

NAME	ADDRESS
Harsh	Kolkata
Ashish	Durgapur
Pratik	Delhi
Dhanraj	Bihar

Example 2.

In this example, we will create a view named StudentNames from the table StudentDetails.

Query:

CREATE VIEW StudentNames AS

SELECT S_ID, NAME

FROM StudentDetails

ORDER BY NAME;

If we now query the view as,

SELECT * FROM StudentNames;

Output:

S_ID	NAMES
2	Ashish
4	Dhanraj
1	Harsh
3	Pratik
5	Ram

References for Theory:

- Silberschatz A., Korth H., Sudarshan S., "Database System Concepts", MGH
- Connally T, Begg C., "Database Systems", Pearson Education
- Raghurama Krishan, "Database Management Systems", McGrawHill
- S.K.Singh, "Database Systems : Concepts, Design and Application", Pearson

CONCLUSION:

TITLE: Experiment Write-up (EW)
Doc. No.: DBMS&SQL/SOP/EW

Subject: Database Management System & SQL Laboratory

EXPERIMENT NO.: 04

TITLE: Write a code for the implementation of Indexes and Stored Procedures**.**

LEARNING OBJECTIVES:

1. To study the fundamental concepts of database management.
2. To learn the basic issues of transaction processing and concurrency control.
3. To learn a powerful, flexible and scalable general-purpose distributed database.

THEORY:

Procedure is a subprogram used to perform a specific action. A subprogram is a named block of PL/SQL. There are two types of subprograms in PL/SQL namely Procedures and Functions. Every subprogram will have a declarative part, an executable part or body, and an exception handling part, which is optional. Declarative part contains variable declarations. Body of a subprogram contains executable statements of SQL and PL/SQL. Statements to handle exceptions are written in exception part. Procedure specification begins with CREATE and ends with procedure name or parameters list.
Procedures that do not take parameters are written without a parenthesis. The body of the procedure starts after the keyword IS or AS and ends with keyword END.

Syntax for Procedure:

CREATE [OR REPLACE] PROCEDURE procedure_name [(parameter_name [IN|OUT|IN OUT] type [, ...])]

{ISAS}
BEGIN
procedure
body
EXCEPTION
Exception handling
END procedure_name;

Procedure is created using CREATE PROCEDURE statement.
OR REPLACE specifies the procedure is to replace an existing procedure if present. A procedure may be passed multiple parameters. IN OUT | IN OUT specifies the mode of the parameter. Type specifies the data type of the parameter.

IN -The parameter can be referenced by the procedure or function. The value of the parameter cannot be overwritten by the procedure or function.

OUT- The parameter cannot be referenced by the procedure or function, but the value of the parameter can be overwritten by the procedure or function.

IN OUT- The parameter can be referenced by the procedure or function and the value of the parameter can be overwritten by the procedure or function.

A function is a named PL/SQL Block which is similar to a procedure. The major difference between a procedure and a function is, a function must always return a value, but a procedure may or may not return a value.

Syntax for Function:

CREATE [OR REPLACE] FUNCTION function_name [parameters]
RETURN return_datatype; IS Declaration_section BEGIN Execution_section
Return return_variable; EXCEPTION exception section Return return_variable;
END;

Example :

- Write a PL/SQL Procedure using IN OUT type paramter

```
CREATE OR REPLACE PROCEDURE min (a IN number , b IN number , c OUTnumber)
IS
BEGIN
  IF a < b THEN
c:=a;
END IF;
END min;
```

Procedure call:

```
DECLARE
  x number:=40;
y number:=50;    z
number;
BEGIN
  min(x,y,z);
   dbms_output.put_line('Minimum of (40,50): ' || z);
END;
```

TITLE: Experiment Write-up (EW)
Doc. No.: DBMS&SQL/SOP/EW

- Write Function to Calculate Factorial of given number

Ans: CREATE OR REPLACE FUNCTION fact (a IN number)
i number; fct number :=1; BEGIN
for I IN 1..a LOOP
fct:=fct*1;
END LOOP
Return fct;
END fct;

Function call:

```
DECLARE
a number ;
        b number;
BEGIN
a:=:a;
        b:=:fct(a);
    dbms_output.put_line('Factorial of '|| a || 'is '|| b);
END;
```

References for Theory:

- Silberschatz A., Korth H., Sudarshan S., "Database System Concepts", MGH
- Connally T, Begg C., "Database Systems", Pearson Education
- Raghurama Krishan, "Database Management Systems", McGrawHill
- S.K.Singh, "Database Systems : Concepts, Design and Application", Pearson

CONCLUSION:

Prepared by | Verified by

Mr. N. I. Bhopale
(Subject Teacher)

Dr. B. S. Agarkar
(HOD, Deptt. of ECE)

TITLE: Experiment Write-up (EW)
Doc. No.: DBMS&SQL/SOP/EW

Subject: Database Management System & SQL Laboratory

EXPERIMENT NO.: 05

TITLE: Write a code to implement User defined Functions on DB.

LEARNING OBJECTIVES:

1. To study the fundamental concepts of UDF.
2. To learn the basic issues of functions.

THEORY:

1. Introduction to User-Defined Functions (UDFs)

- Briefly explain what UDFs are: Functions created by the user in a database to perform specific tasks or calculations.
- Mention that UDFs allow for reusable logic within SQL queries.

2. Types of UDFs

- **Scalar Functions**: Return a single value, used for operations like calculations, string manipulations, etc.
- **Table-Valued Functions (TVF)**: Return a table of results, useful for queries involving sets of data.
- **Inline vs. Multi-statement Table-Valued Functions**: Explain that inline TVFs return the result set from a single SELECT statement, while multi-statement TVFs can have multiple SQL statements.

3. Advantages of UDFs

- **Code Reusability**: Encapsulate frequently used logic or calculations.
- **Modularity**: Break down complex queries into simpler, reusable functions.
- **Maintainability**: Centralize logic that can be updated easily, affecting all queries that use the function.
- **Improved Readability**: Using functions can make SQL queries more understandable by abstracting complex logic.

4. Syntax of UDFs

```
CREATE FUNCTION FunctionName (@param1 DataType)
RETURNS ReturnType
AS
```

```
BEGIN
  -- Function logic here
  RETURN @result
END
```

5.Examples of UDF Usage

```
CREATE FUNCTION CalculateDiscount(@price DECIMAL(10, 2), @discount
DECIMAL(5, 2))
RETURNS DECIMAL(10, 2)
AS
BEGIN
  RETURN @price - (@price * @discount / 100)
END
```

Table-Valued Function Example: Returning rows of employees above a certain salary.

```
CREATE FUNCTION EmployeesAboveSalary(@salary INT)
RETURNS TABLE
AS
RETURN (SELECT * FROM Employees WHERE Salary > @salary)
```

6. Parameterization in UDFs

- Explain how UDFs can accept parameters to work dynamically with different inputs.
- Discuss optional parameters and default values.

7. Best Practices

- **Avoid Side-Effects**: UDFs should not modify data in the database (i.e., no INSERT, UPDATE, or DELETE operations).
- **Optimize for Performance**: Explain the performance considerations, such as execution time and indexing, particularly for table-valued functions.
- **Error Handling**: Ensure proper error checking and handling within the UDF.

8. Performance Considerations

- Scalar UDFs may sometimes have performance drawbacks in certain database systems, such as SQL Server, since they may not be fully optimized for parallelism.
- Inline TVFs tend to be faster than multi-statement TVFs because they allow the database optimizer to work more effectively.

9. Security in UDFs

- Discuss how UDFs respect the permissions and privileges set on the objects they access.
- Suggest restricting permissions for UDF creation to authorized users only.

10. Real-World Use Cases

- **Data Transformation**: Use UDFs for transforming data before loading it into reports or dashboards.
- **Custom Business Logic**: Encapsulate custom calculations (e.g., tax computations, financial formulas) inside a UDF.
- **Data Validation**: Create UDFs to validate input data before processing it.

11. Limitations of UDFs

- **No DML Support**: UDFs cannot perform INSERT, UPDATE, DELETE operations.
- **Performance Overhead**: In certain cases, using UDFs can slow down queries, especially when used excessively or inefficiently.

References for Theory:

- Silberschatz A., Korth H., Sudarshan S., "Database System Concepts", MGH
- Connally T, Begg C., "Database Systems", Pearson Education
- Raghurama Krishan, "Database Management Systems", McGrawHill
- S.K.Singh, "Database Systems : Concepts, Design and Application", Pearson

CONCLUSION:

TITLE: Experiment Write-up (EW)
Doc. No.: DBMS&SQL/SOP/EW

Subject: Database Management System & SQL Laboratory

EXPERIMENT NO.: 06

TITLE: Write a code to implement Triggers on DB.

LEARNING OBJECTIVES:

1. To **Understand the Concept of SQL Triggers**:.
2. To learn Learn Trigger Syntax.

THEORY:

1. A **trigger** is a special type of stored procedure that automatically executes or "fires" in response to certain events on a specified table or view in a database.

2 . Types of Triggers:

- **BEFORE Trigger**: Executes before the event operation (e.g., before an INSERT, UPDATE, or DELETE occurs).
- **AFTER Trigger**: Executes after the event operation has occurred.
- **INSTEAD OF Trigger**: Used on views to substitute the trigger action in place of the triggering operation.

3. Row-Level vs Statement-Level Triggers:

- **Row-level Trigger**: Fires once for each row affected by the triggering event.
- **Statement-level Trigger**: Fires once for the entire operation, regardless of how many rows are affected.

4. Trigger Components:

- **Trigger Event**: The event that causes the trigger to activate (e.g., INSERT, UPDATE, DELETE).
- **Trigger Timing**: Defines when the trigger fires, either BEFORE or AFTER the event.
- **Trigger Action**: The SQL statements that are executed when the trigger fires.

5. NEW and OLD Variables:

- In INSERT triggers, the NEW keyword refers to the new row being inserted.
- In UPDATE triggers, OLD refers to the previous values, and NEW refers to the updated values.
- In DELETE triggers, OLD refers to the row values before deletion.

6. Purpose of Triggers:

- **Data Integrity**: Automatically enforce business rules and data validation.
- **Audit Trails**: Track changes made to a table by recording old and new values.
- **Cascading Changes**: Automatically update or modify other related tables in response to changes.

7.Advantages of Triggers:

- **Automation**: Triggers reduce manual interventions and can automate routine tasks.
- **Consistency**: Triggers help ensure consistency across the database by enforcing rules automatically.
- **Audit Capabilities**: They can log detailed information about data changes without requiring explicit logging code.

8. Disadvantages of Triggers:

- **Performance Overhead**: Excessive or complex triggers can lead to performance degradation as they introduce additional processing for every affected row.
- **Hidden Logic**: Triggers can obscure the logic of an application, making it harder to track where certain changes occur.
- **Recursive Triggers**: Triggers can cause recursion (trigger firing itself repeatedly) if not designed carefully, leading to unintended consequences.

9. Recursive and Nested Triggers:

- Some database systems support **recursive triggers**, where one trigger causes another to fire, and **nested triggers**, where a trigger calls another trigger.

10. Trigger Restrictions:

- Certain operations might not be allowed within triggers (e.g., creating or dropping tables).
- A trigger cannot directly modify the table on which it is defined (in some database systems) to avoid recursive firing.

11. Trigger Syntax:

```
CREATE TRIGGER trigger_name
{BEFORE | AFTER } {INSERT | UPDATE | DELETE}
ON table_name
FOR EACH ROW
BEGIN
  -- SQL statements (trigger action)
END;
```

12. Trigger Use Cases:

- **Audit Logging**: Automatically record changes to important tables.
- **Enforcing Business Rules**: E.g., preventing negative values in a column.
- **Maintaining Derived Data**: Automatically update aggregated data (e.g., totals or averages) when underlying data changes.
- **Cascading Operations**: Automatically update related tables (e.g., cascading deletes).

References for Theory:

- Silberschatz A., Korth H., Sudarshan S., "Database System Concepts", MGH
- Connally T, Begg C., "Database Systems", Pearson Education
- Raghurama Krishan, "Database Management Systems", McGrawHill
- S.K.Singh, "Database Systems : Concepts, Design and Application", Pearson

CONCLUSION:

TITLE: Experiment Write-up (EW)
Doc. No.: DBMS&SQL/SOP/EW

Subject: Database Management System & SQL Laboratory

EXPERIMENT NO.: 07

TITLE: For the given table write a cursor (Implicit, Explicit, Cursor using Looping statements, Parameterized Cursor)

LEARNING OBJECTIVES:

1. To study the fundamental concepts of database management.
2. To learn the basic issues of transaction processing and concurrency control.
3. To learn a powerful, flexible and scalable general-purpose distributed database.

THEORY: Cursor attributes (PL/SQL): Each cursor has a set of attributes that enables an application program to test the state of the cursor.

These attributes are %ISOPEN, %FOUND, %NOTFOUND, and %ROWCOUNT.
%ISOPEN: This attribute is used to determine whether a cursor is in the open state. When a cursor is passed as a parameter to a function or procedure, it is useful to know (before attempting to open the cursor) whether the cursor is already open.

%FOUND: This attribute is used to determine whether a cursor contains rows after the execution of a FETCH statement. If FETCH statement execution was successful, the %FOUND attribute has a value of true. If FETCH statement execution was not successful, the %FOUND attribute has a value of false. The result is unknown when:

- The value of *cursor-variable-name* is null
- The underlying cursor of *cursor-variable-name* is not open
- The %FOUND attribute is evaluated before the first FETCH statement was executed against the underlying cursor
- FETCH statement execution returns an error

The %FOUND attribute provides an efficient alternative to using a condition handler that checks for the error that is returned when no more rows remain to be fetched.

%NOTFOUND : This attribute is the logical opposite of the %FOUND attribute.
%ROWCOUNT:This attribute is used to determine the number of rows that have been fetched since a cursor was opened.

Cursor attribute	%ISOPEN	%FOUND	%NOTFOUND	%ROWCOUNT
Before OPEN	False	Undefined	Undefined	"Cursor not open" exception
After OPEN and before 1st FETCH	True	Undefined	Undefined	0
After 1st successful FETCH	True	True	False	1
After *n*th successful FETCH (last row)	True	True	False	*n*
After *n*+1st FETCH (after last row)	True	False	True	*n*
After CLOSE	False	Undefined	Undefined	"Cursor not open" exception

Static cursors (PL/SQL)

A *static cursor* is a cursor whose associated query is fixed at compile time. Declaring a cursor is a prerequisite to using it. Declarations of static cursors using PL/SQL syntax within PL/SQL contexts are supported by the data server.

Description
cursor-name: Specifies an identifier for the cursor that can be used to reference the cursor and its result set.

Query: Specifies a SELECT statement that determines a result set for the cursor.

- **Parameterized cursors (PL/SQL)**
  Parameterized cursors are static cursors that can accept passed-in parameter values when they are opened.
- **Opening a cursor (PL/SQL)**
  The result set that is associated with a cursor cannot be referenced until the cursor has been opened.
- **Fetching rows from a cursor (PL/SQL)**
  The FETCH statement that is required to fetch rows from a PL/SQL cursor is supported by the data server in PL/SQL contexts.
- **Closing a cursor (PL/SQL)**
  After all rows have been retrieved from the result set that is associated with a cursor, the cursor must be closed. The result set cannot be referenced after the cursor has been closed.
- **Using %ROWTYPE with cursors (PL/SQL)**
  The %ROWTYPE attribute is used to define a record with fields corresponding to all

of the columns that are fetched from a cursor or cursor variable. Each field assumes the data type of its corresponding column.

- **Cursor attributes (PL/SQL)**
  Each cursor has a set of attributes that enables an application program to test the state of the cursor.

References for Theory:

- Silberschatz A., Korth H., Sudarshan S., "Database System Concepts", MGH
- Connally T, Begg C., "Database Systems", Pearson Education
- Raghurama Krishan, "Database Management Systems", McGrawHill
- S.K.Singh, "Database Systems : Concepts, Design and Application", Pearson

CONCLUSION:_____

TITLE: Experiment Write-up (EW)
Doc. No.: DBMS&SQL/SOP/EW

Subject: Database Management System & SQL Laboratory

EXPERIMENT NO.: 08

TITLE: For the given table write a PL SQL code to insert ten values into the table

LEARNING OBJECTIVES:

1. To study the fundamental concepts of database management.
2. To learn the basic issues of transaction processing and concurrency control.
3. To learn a powerful, flexible and scalable general-purpose distributed database.

THEORY:

In most situations that require an explicit cursor, can simplify coding by using a cursor `FOR` loop instead of the `OPEN`, `FETCH`, and `CLOSE` statements. A cursor `FOR` loop implicitly declares its loop index as a `%ROWTYPE` record, opens a cursor, repeatedly fetches rows of values from the result set into fields in the record, and closes the cursor when all rows have been processed.

Consider the PL/SQL block below, which computes results from an experiment, then stores the results in a temporary table. The `FOR` loop index `c1_rec` is implicitly declared as a record. Its fields store all the column values fetched from the cursor `c1`. Dot notation is used to reference individual fields.

```
DECLARE
   result temp.col1%TYPE;
   CURSOR c1 IS
      SELECT n1, n2, n3 FROM data_table WHERE exper_num = 1;
BEGIN
   FOR c1_rec IN c1 LOOP
      /* calculate and store the results */
      result := c1_rec.n2 / (c1_rec.n1 + c1_rec.n3);
      INSERT INTO temp VALUES (result, NULL, NULL);
   END LOOP;
   COMMIT;
END;
```

When the cursor `FOR` loop is entered, the cursor name cannot belong to a cursor already opened by an `OPEN` statement or enclosing cursor `FOR` loop. Before each iteration of the `FOR` loop, PL/SQL fetches into the implicitly declared record. The record is defined only inside the loop.

Similary ten rows insertion can be done like

```
CREATE TABLE employee (
   employee_id NUMBER(10),
   employee_name VARCHAR2(50)
);

DECLARE
   -- Variable to store the employee id
   v_emp_id NUMBER(10);
   -- Variable to store the employee name
   v_emp_name VARCHAR2(50);
BEGIN
   -- Initialize the employee id and employee name
   v_emp_id := 1;

   -- Loop to insert 10 rows
   FOR i IN 1..10 LOOP
      -- Generate employee name dynamically using loop index
      v_emp_name := 'Employee_' || i;

      -- Insert statement
      INSERT INTO employee (employee_id, employee_name)
      VALUES (v_emp_id, v_emp_name);

      -- Increment the employee id for the next row
      v_emp_id := v_emp_id + 1;
   END LOOP;

   -- Commit the transaction to save the changes in the table
   COMMIT;

   -- Display message for successful insertion
   DBMS_OUTPUT.PUT_LINE('10 rows have been inserted successfully.');
END;

/
```

References for Theory:

- Silberschatz A., Korth H., Sudarshan S., "Database System Concepts", MGH
- Connally T, Begg C., "Database Systems", Pearson Education
- Raghurama Krishan, "Database Management Systems", McGrawHill
- S.K.Singh, "Database Systems : Concepts, Design and Application", Pearson

TITLE: Experiment Write-up (EW)
Doc. No.: DBMS&SQL/SOP/EW

OUTPUT:

CONCLUSION:

TITLE: Experiment Write-up (EW)
Doc. No.: DBMS&SQL/SOP/EW

Subject: Database Management System & SQL Laboratory

EXPERIMENT NO.: 09

TITLE: For the given table write a PL SQL code to exception handling

LEARNING OBJECTIVES:

1. To study the fundamental concepts of database management.
2. To learn the basic issues of transaction processing and concurrency control.
3. To learn a powerful, flexible and scalable general-purpose distributed database.

An error occurs during the program execution is called Exception in PL/SQL.

PL/SQL facilitates programmers to catch such conditions using exception block in the program and an appropriate action is taken against the error condition.

There are two type of exceptions:

- System-defined Exceptions
- User-defined Exceptions

Syntax for exception handling:

```
DECLARE
  <declarations section>
BEGIN
  <executable command(s)>
EXCEPTION
  <exception handling goes here >
  WHEN exception1 THEN
     exception1-handling-statements
  WHEN exception2  THEN
    exception2-handling-statements
  WHEN exception3 THEN
    exception3-handling-statements
  ........
  WHEN others THEN
    exception3-handling-statements
END;
```

Example of exception handling

```
SELECT* FROM COUSTOMERS;
```

TITLE: Experiment Write-up (EW)
Doc. No.: DBMS&SQL/SOP/EW

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	23	Allahabad	20000
2	Suresh	22	Kanpur	22000
3	Mahesh	24	Ghaziabad	24000
4	Chandan	25	Noida	26000
5	Alex	21	Paris	28000
6	Sunita	20	Delhi	30000

```
DECLARE
  c_id customers.id%type := 8;
  c_name  customers.name%type;
  c_addr customers.address%type;
BEGIN
  SELECT  name, address INTO  c_name, c_addr
  FROM customers
  WHERE id = c_id;
DBMS_OUTPUT.PUT_LINE ('Name: '||  c_name);
 DBMS_OUTPUT.PUT_LINE ('Address: ' || c_addr);
EXCEPTION
  WHEN no_data_found THEN
    dbms_output.put_line('No such customer!');
  WHEN others THEN
    dbms_output.put_line('Error!');
END;
/
```

RESULT:
```
No such customer!
PL/SQL procedure successfully completed.
```

The above program should show the name and address of a customer as result whose ID is given. But there is no customer with ID value 8 in our database, so the program raises the run-time exception NO_DATA_FOUND, which is captured in EXCEPTION block.

Syntax for raising an exception:

```
DECLARE
  exception_name EXCEPTION;
BEGIN
  IF condition THEN
    RAISE exception_name;
  END IF;
EXCEPTION
  WHEN exception_name THEN
```

statement;
END;

Syntax for user define exceptions

DECLARE
my-exception EXCEPTION;

Exception	Oracle Error	SQL Code	Description
ACCESS_INTO_NULL	06530	-6530	It is raised when a NULL object is automatically assigned a value.
CASE_NOT_FOUND	06592	-6592	It is raised when none of the choices in the "WHEN" clauses of a CASE statement is selected, and there is no else clause.
COLLECTION_IS_NULL	06531	-6531	It is raised when a program attempts to apply collection methods other than exists to an uninitialized nested table or varray, or the program attempts to assign values to the elements of an uninitialized nested table or varray.
DUP_VAL_ON_INDEX	00001	-1	It is raised when duplicate values are attempted to be stored in a column with unique index.
INVALID_CURSOR	01001	-1001	It is raised when attempts are made to make a cursor operation that is not allowed, such as closing an unopened cursor.
INVALID_NUMBER	01722	-1722	It is raised when the conversion of a character string into a number fails because the string does not represent a valid number.
LOGIN_DENIED	01017	-1017	It is raised when s program attempts to log on to the database with an invalid username or password.
NO_DATA_FOUND	01403	+100	It is raised when a select into statement returns no rows.
NOT_LOGGED_ON	01012	-1012	It is raised when a database call is issued without being connected to the database.
PROGRAM_ERROR	06501	-6501	It is raised when PL/SQL has an internal problem.

ROWTYPE_MISMATCH	06504	-6504	It is raised when a cursor fetches value in a variable having incompatible data type.
SELF_IS_NULL	30625	-30625	It is raised when a member method is invoked, but the instance of the object type was not initialized.
STORAGE_ERROR	06500	-6500	It is raised when PL/SQL ran out of memory or memory was corrupted.
TOO_MANY_ROWS	01422	-1422	It is raised when a SELECT INTO statement returns more than one row.
VALUE_ERROR	06502	-6502	It is raised when an arithmetic, conversion, truncation, or size-constraint error occurs.
ZERO_DIVIDE	01476	1476	It is raised when an attempt is made to divide a number by zero.

```
DECLARE
        a int;
        b int;
        c int;
BEGIN
        a := 11;
        b := 0;
        c := a/b;
        dbms_output.put_line('RESULT=' || c);
EXCEPTION
        when ZERO_DIVIDE then
                dbms_output.put_line('Division by 0 is not possible');
END;
```

Output:

Statement processed.

Division by 0 is not possible

Numbered Exception Handling

In oracle, some of the pre-defined exceptions are numbered in the form of **four integers preceded by a hyphen symbol**. To handle such exceptions we should assign a name to them before using them.

This can be done by using the **Pragma exception technique** in which a numbered exception handler is bound to a name. For this purpose, we use a keyword in PL/SQL program and write a statement that binds a name to a numbered exception using the following syntax and this statement is written in the **DECLARE** section of program:

```
pragma exception_init(exception_name, exception _number);
```

where, `pragma exception_init`(case doesn't matter) is a keyword indicating Pragma exception technique with two arguments:

- **exception_name**, which is a user-defined name given to a predefined numbered exception if it occurs.
- **exception_number**, is the number allotted to the exception by oracle.

For example:

ROLLNO	SNAME	AGE	COURSE
11	Anu	20	BSC
12	Asha	21	BCOM
13	Arpit	18	BCA
14	Chetan	20	BCA
15	Nihal	19	BBA

Code:

```
DECLARE
        sno student.rollno%type;
        snm student.sname%type;
        s_age student.age%type;
        cr student.course%type;
        -- Exception name declared below
        already_exist EXCEPTION;
        -- pragma statement to provide name to numbered exception
        pragma exception_init(already_exist, -1);
BEGIN
        sno:=&rollno;
        snm:='&sname';
        s_age:=&age;
        cr:='&course';
        INSERT into student values(sno, snm, s_age, cr);
        dbms_output.put_line('Record inserted');
```

```
        EXCEPTION
                WHEN already_exist THEN
                        dbms_output.put_line('Record already exist');
END;
```

Output:

```
Enter the value for sno:11
Enter the value for snm:heena
Enter the value for s_age:20
Enter the value for cr:bsc
Record already exist
PL/SQL procedure successfully completed.
```

User-defined Exception

ROLLNO	SNAME	Total_Courses
11	Anu	2
12	Asha	1
13	Arpit	3
14	Chetan	1

Code:

```
DECLARE
        sno student.rollno%type;
        snm student.sname%type;
        crno student.total_course%type;
        invalid_total EXCEPTION;
BEGIN
        sno := &rollno;
        snm := '&sname';
        crno:=total_courses;
        IF (crno > 3) THEN
                RAISE invalid_total;
        END IF;
        INSERT into student values(sno, snm, crno);
        EXCEPTION
                WHEN invalid_total THEN
```

```
                dbms_output.put_line('Total number of courses cannot be more than
3');
END;
```

Output:

```
Enter the value for sno:15
Enter the value for snm:Akash
Enter the value for crno:5
Total number of courses cannot be more than 3
PL/SQL procedure successfully completed.
```

References for Theory:

- Silberschatz A., Korth H., Sudarshan S., "Database System Concepts", MGH
- Connally T, Begg C., "Database Systems", Pearson Education
- Raghurama Krishan, "Database Management Systems", McGrawHill
- S.K.Singh, "Database Systems : Concepts, Design and Application", Pearson

OUTPUT:_____

CONCLUSION:_____