DATABASE DESIGN AND MANAGEMENTLABMANUAL

RECORDNOTEBOOK

NameoftheStudent	:____________________
RegisterNumber	:____________________
Department/Semester	:____________________
SubjectTitle/Code	:____________________

INDEX

Ex.No.	Date	Titleoftheexperiment	Page.No.	StaffInitial

EX:No:1	Database Development Lifecycle

Aim:
To implement an Database Development Lifecycle using Pg Admin-Post gre SQL.
Procedure:
Creating a database in pg Admin involves several steps, starting from problem definition and requirement analysis to defining the scope and constraints. Here's how you can go through these stages using pg Admin:
<u>Procedure:</u>

Create the Database using pgAdmin Step-by-Step Guide:
Install pgAdmin and PostgreSQL:
Ensure pg Admin and PostgreSQL are installed on your system. You can download them from the official website.
Open pg Admin: Launch pg Admin and connect to your Post gre SQL server. Create a New Database:
In the pg Admin interface, right- click on the 'Databases 'node in the Browser tree ,then select 'Create'->'Database'.
Define the Database:
In the 'Create Database 'dialog ,provide a name for your database ,for example, Retail DB. Click 'Save' to create the database.
Create Schemas and Tables:
Right-click on the newly created database and choose 'Create'->'Schema 'to define the schema.
Right-click on the 'Schemas' node and select 'Create'->'Table' to start defining your tables based on your requirements analysis.
For each table, define the columns, data types, constraints (e.g., primary keys, foreign keys), and other attributes.
Add Constraints and Indexes:
While creating tables, you can add constraints like NOTNULL, UNIQUE, PRIMARYKEY, and FOREIGN KEY to ensure data integrity.
Right-click on a table and choose’ Properties' ->'Constraints 'to add constraints. Add indexes to columns that will be frequently queried to improve performance**. Insert Initial Data:**
Use the 'Query Tool' in pg Admin to write SQLINSERT statements to populate your tables with initial data.
For example:
sql code INSERT INTO customers (customer_id, name, email)
VALUES (1, 'porkodi', 'porkodi@gracecoe.org');
Define Relationships: Ensure tables are properly linked through foreign keys to

maintain referential integrity.

For example, link the orders table to the customers table using a
foreign key.

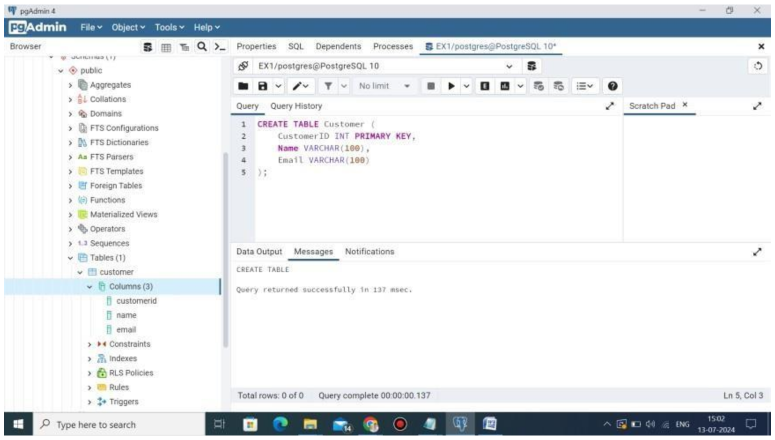
Set up User Roles and Permissions: Define roles and permissions
to control access to the database.
Right-clickonthe'Login/GroupRoles'nodeandchoose'Create'->'Login/GroupRole'to create users and assign appropriate permissions.

Example: Creating a Table with Constraints

Syntax: -- Create customers table

```
CREATE TABLE customers (
   customer_id SERIAL PRIMARY KEY,
   name VARCHAR(100) NOT NULL,
   email VARCHAR(100) UNIQUE NOT
NULL
);

-- Create orders table
CREATE TABLE orders (
   order_id SERIAL PRIMARY KEY,
   order_date DATE NOT NULL,
   customer_id INT NOT NULL,
   FOREIGN KEY (customer_id)
REFERENCES customers(customer_id)
);

-- Create employees table
CREATE TABLE employees (
   EmployeeID INT PRIMARY KEY,
--Primary Key Constraint
FirstName VARCHAR(50) NOT NULL,
-- NotNull Constraint
   LastName VARCHAR(50) NOT NULL,
-- Not Null Constraint
   Email VARCHAR(100) UNIQUE,
-- Unique Constraint
   DepartmentID INT,                    --
Foreign Key Column
   Salary DECIMAL(10, 2) DEFAULT 50000,
-- Default Constraint
   DateOfBirth DATE CHECK (DateOfBirth <
'2005-01-01'), -- Check Constraint
    CONSTRAINT FK_Department
        FOREIGN KEY (DepartmentID)
        REFERENCES
Departments(DepartmentID)       -- Foreign Key
Constraint
);
```

OUTPUT:



```
CREATE TABLE Customer (
    CustomerID INT PRIMARY KEY,
    Name VARCHAR(100),
    Email VARCHAR(100)
);
```

```
CREATE TABLE

Query returned successfully in 137 msec.
```

Total rows: 0 of 0   Query complete 00:00:00.137   Ln 5, Col 3

Procedure2:
In the database development lifecycle, the initial stages involve defining the problem, analyzing requirements, and identifying the scope and constraints. Below is a detailed procedure for each of these stages, including example SQL code snippets and expected results.

Step1: Problem Definition and Requirement Analysis Problem Definition:
Clearly define the problem your database aims to solve. For example, let's assume we are developing a database for a library management system.

Requirement Analysis:

Gather Requirements: Conduct interviews, surveys, and workshops with stakeholders (librarians, library members, administrators).
Document Requirements: List out all the required functionalities like managing books, members, loans, and returns.
Identify Entities and Attributes:
Entities: Books, Members, Loans
Attributes:
Books: Book ID, Title, Author, ISBN, Publication Year **Members:** Member ID, Name, Email, Membership Date **Loans:** Loan ID, BookID MemberID, LoanDate, ReturnDate
Step 2: Scope and Constraints
Scope:
Inclusions: The system will include functionalities to manage book details, member details, loan transactions, and report generation.
Exclusions: The system will not handle digital media, financial transactions, or integration with external systems in this phase.
Constraints:
Technical Constraints: Limited to using Post gre SQL and pg Admin.
Resource Constraints: Limited budget and time for initial development (6months).Operational Constraints: Must comply with data protection regulations (GDPR).
Creating the Database Using pg Admin **Install pgAdmin and PostgreSQL:** Ensure pgAdmin and PostgreSQL are installed on your system.
Open pg Admin: Launch pg Admin and connect to your Postgre SQL server.
Create a New Database:
In pgAdmin,right-click on the 'Databases 'node in the Browser tree and select Create'->'Database'. Name it LibraryDB.
Define Tables:

Right-click on the' Schemas 'node and select 'Create'->'Table' to start defining your tables based on the requirement analysis.

Validating Design Using Normalization Normalization Steps:

First Normal Form (1NF): Ensure all tables have atomic values and primary

keys. Example: The Books table has atomic attributes like Title, Author.
Second Normal Form (2NF): Ensure all non-key attributes are fully functional dependent on the primary key.

Example: The Loans table ensures Loan Date and Return Date are dependent on Loan ID. Third Normal Form (3NF): Ensure no transitive dependencies.

Example: The Members table ensures all attributes are dependent only on Member ID.

RESULT:

Thus the above program was implemented and executed successfully.

EXNO:2	Database design using Conceptual modeling(ER-EER)–top-down approach

Aim:
To implement an Database design using Conceptual modeling(ER-EER)–top-down approach. Mapping conceptual to relational database and validate using Normalization
Procedure:

Database Design Using Conceptual Modeling(ER-EER)–Top-Down Approach
The top-down approach in database design starts with a high-level conceptual model, typically an Entity- Relationship(ER)or Enhanced Entity-Relationship(EER)model, and maps it to a relational database schema .This process includes the following steps:

Conceptual Modeling: Create an ER or EER diagram that captures the entities, relationships, and attributes of the system.
Mapping Conceptual Model to Relational Model: Convert the ER/EER diagram into a relational schema. **Normalization:** Validate and refine the relational schema using normalization techniques to ensure it meets desired properties.
Step1:Conceptual Modeling with ER/EER Diagrams
Identify Entities: Determine the main objects (entities) in the system. For example, in a retail system, entities might include Customer, Order, and Product.
Identify Relationships: Determine how entities are related to each other .For example, a Customer places an Order. **Define Attributes:** Identify the attributes for each entity. For example, a Customer might have attributes like Customer ID, Name, and Email.
Enhanced ER (EER) Modeling: In corporate additional concepts like specialization, generalization, and aggregation if necessary.
Example ER Diagram:

Entities: Customer, Order, Product
Relationships: places, contains
Attributes: Customer: CustomerID
(PK), Name, Email Order: Order ID
(PK), Order Date
Product: Product ID (PK), Product Name, Price
Relationships: Customer places Order
Order contains Product

Step2: Mapping Conceptual Model to Relational Database Convert Entities to Tables:

Each entity in the ER diagram becomes a table.
Convert Relationships to Foreign Keys: Relationships are mapped using foreign keys.
Map Attributes to Columns: Each attribute becomes a column in the corresponding table.

Procedure
Identify Entities: Determine the main objects (entities)in the system. Identify Relationships: Determine how entities are related to each other. Define Attributes: Identify the attributes for each entity.
Enhanced ER (EER) Modeling: Incorporate additional concepts like specialization, generalization, and aggregation if necessary.

Example: Library Management System

Entities: Books, Members, Loans
Relationships: Members borrow Books
Attributes:
Books: BookID, Title, Author, ISBN, Publication
Year Members: MemberID, Name, Email,
Membership Date Loans: Loan ID, Book ID,
Member ID, LoanDate, ReturnDate ER **Diagram**
Books: BookID(PK),Title, Author,ISBN, PublicationYear
Members:MemberID(PK),Name,Email,MembershipDate
Loans:LoanID(PK),BookID(FK),MemberID(FK),LoanDate,ReturnDate
Relationships:

A Member can borrow multiple Books. A Book can
be borrowed by multiple Members.

Step2: Mapping Conceptual Model to Relational
Database Procedure Convert Entities to Tables:
Each entity becomes a table.
Convert Relationships to Foreign Keys: Relationships are mapped using foreign keys. Map Attributes to Columns: Each attribute becomes a column in the corresponding table.

SQLQUERY:
```
- Create Books Table
CREATE TABLE Books (
   BookID SERIAL PRIMARY KEY,
   Title VARCHAR(255) NOT NULL,
   Author VARCHAR(255) NOT
NULL,
   ISBN VARCHAR(13) UNIQUE
NOT NULL,
   PublicationYear INT
);

-- Create Members Table
CREATE TABLE Members (
   MemberID SERIAL PRIMARY
KEY,
   Name VARCHAR(255) NOT
NULL,
   Email VARCHAR(255) UNIQUE
NOT NULL,
   MembershipDate DATE NOT
NULL
);

-- Create Loans Table
CREATE TABLE Loans (
   LoanID SERIAL PRIMARY KEY,
   BookID INT NOT NULL,
   MemberID INT NOT NULL,
   LoanDate DATE NOT NULL,
   ReturnDate DATE,
   FOREIGN KEY (BookID)
REFERENCES Books(BookID),
   FOREIGN KEY (MemberID)
REFERENCES Members(MemberID)
);
```

Step3: Validate Using Normalization Procedure

First Normal Form (1NF): Ensure all tables have atomic values and primary keys.
Second Normal Form (2NF): Ensure all non-key attributes are fully functional dependent on the primary key. Third Normal Form (3NF): Ensure no transitive dependencies (non-key attribute dependent on another non-key attribute).

Example Normalization Process:

1NF: Ensure atomic attributes and primary keys. CREATE TABLE Members (

```
    MemberID SERIAL PRIMARY KEY,

    Name VARCHAR(255),

    Email VARCHAR(255)

);
```

2NF: Ensure non-key attributes depend on the primary key.

```
 CREATE TABLE Loans (

    LoanID SERIAL PRIMARY KEY,

    BookID INT,

    MemberID INT,

    LoanDate DATE,

    ReturnDate DATE,

    FOREIGN KEY (BookID) REFERENCES
Books (BookID),

    FOREIGN KEY (MemberID)
REFERENCES Members (MemberID)

);
```

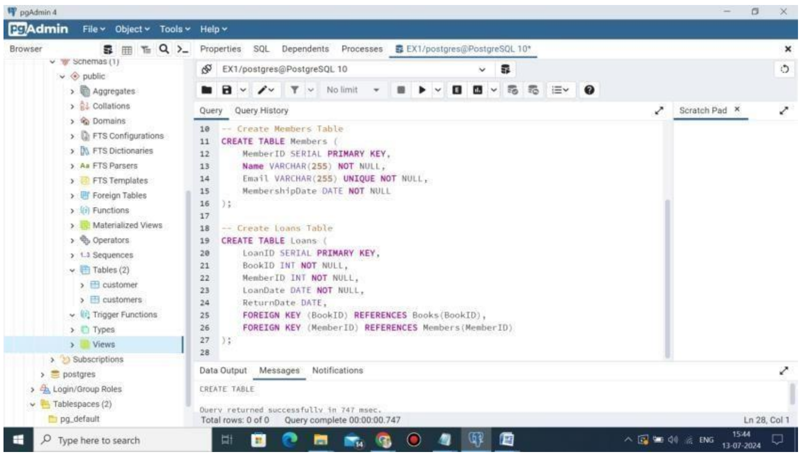
3NF: Remove transitive dependencies.

```
CREATE TABLE Books (
    BookID SERIAL PRIMARY KEY,
    Title VARCHAR(255),
    Author VARCHAR(255),
    ISBN VARCHAR(13),
    PublicationYear INT
);
```

RESULT:

Thus the above program was implemented and executed successfully.

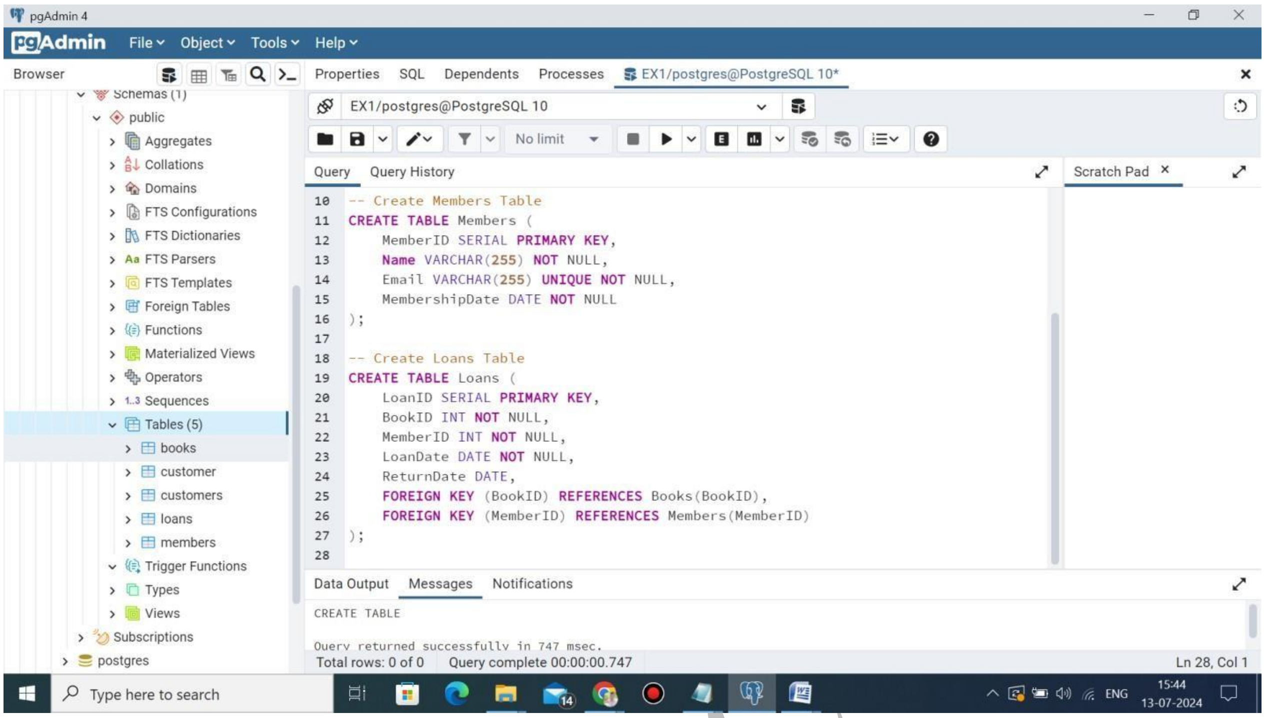
OUTPUT:



```
-- Create Members Table
CREATE TABLE Members (
    MemberID SERIAL PRIMARY KEY,
    Name VARCHAR(255) NOT NULL,
    Email VARCHAR(255) UNIQUE NOT NULL,
    MembershipDate DATE NOT NULL
);

-- Create Loans Table
CREATE TABLE Loans (
    LoanID SERIAL PRIMARY KEY,
    BookID INT NOT NULL,
    MemberID INT NOT NULL,
    LoanDate DATE NOT NULL,
    ReturnDate DATE,
    FOREIGN KEY (BookID) REFERENCES Books(BookID),
    FOREIGN KEY (MemberID) REFERENCES Members(MemberID)
);
```

```
CREATE TABLE

Query returned successfully in 747 msec.
```

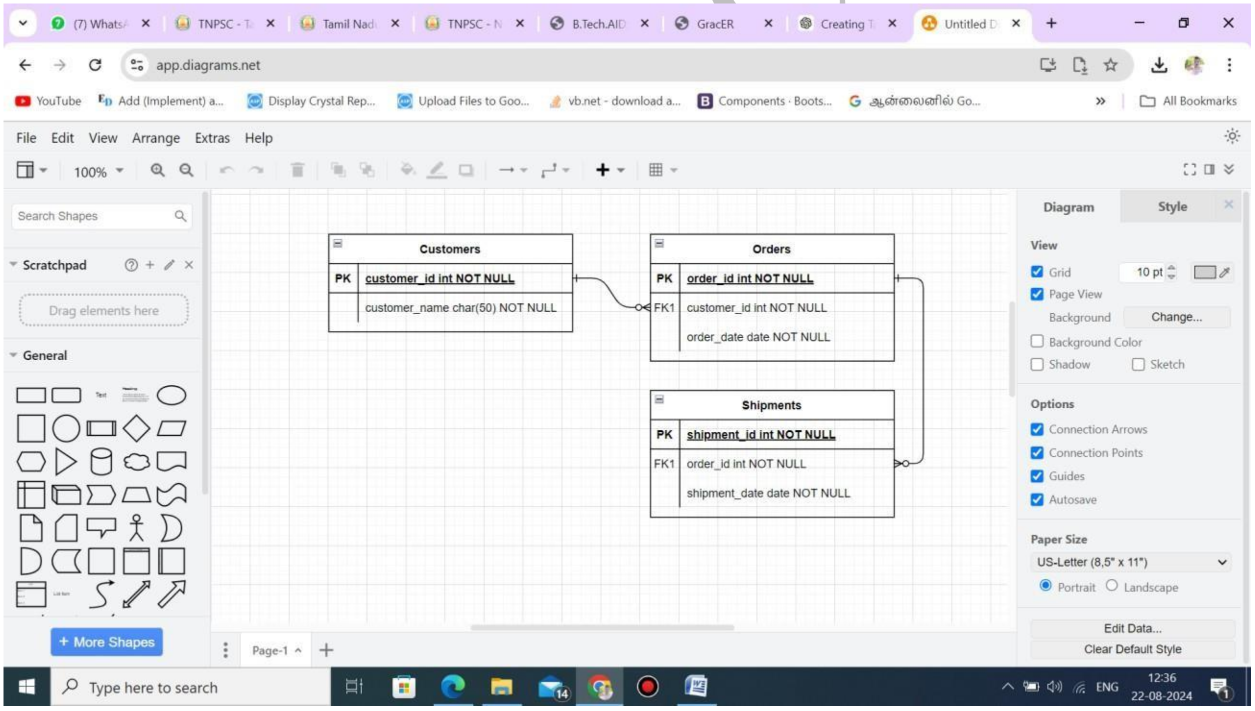
Total rows: 0 of 0   Query complete 00:00:00.747



```sql
-- Create Members Table
CREATE TABLE Members (
    MemberID SERIAL PRIMARY KEY,
    Name VARCHAR(255) NOT NULL,
    Email VARCHAR(255) UNIQUE NOT NULL,
    MembershipDate DATE NOT NULL
);

-- Create Loans Table
CREATE TABLE Loans (
    LoanID SERIAL PRIMARY KEY,
    BookID INT NOT NULL,
    MemberID INT NOT NULL,
    LoanDate DATE NOT NULL,
    ReturnDate DATE,
    FOREIGN KEY (BookID) REFERENCES Books(BookID),
    FOREIGN KEY (MemberID) REFERENCES Members(MemberID)
);
```

```
CREATE TABLE

Query returned successfully in 747 msec.
```

Total rows: 0 of 0   Query complete 00:00:00.747   Ln 28, Col 1



Customers

PK	customer_id int NOT NULL
	customer_name char(50) NOT NULL

Orders

PK	order_id int NOT NULL
FK1	customer_id int NOT NULL
	order_date date NOT NULL

Shipments

PK	shipment_id int NOT NULL
FK1	order_id int NOT NULL
	shipment_date date NOT NULL

EXNO:3	Implement the database using SQL Data definition with constraints, Views

Aim:

To Implement the database using SQL Data definition with constraints, Views

Procedure

Step1: <u>Define the Database Schema</u> Launch pgAdmin: Open pg Admin and connect to your PostgreSQL server. Create a New Database:
Right-click on the' Databases' node in the Browser tree.
Select 'Create' ->'Database'.
Name the database LibraryDB.
Step2:<u>Create Tables with Constraints</u>
Open Query Tool:

Right-click on the Library DB database.
Select ‘Query Tool'.
Create Tables: Enter the following SQL commands to create tables with

constraints.

SQL Query:

```
- Create Books Table
CREATE TABLE Books (
   BookID SERIAL PRIMARY KEY,
   Title VARCHAR(255) NOT NULL,
   Author VARCHAR(255) NOT NULL,
   ISBN VARCHAR(13) UNIQUE NOT NULL,
   PublicationYear INT CHECK (PublicationYear > 0)
);

-- Create Members Table
CREATE TABLE Members (
   MemberID SERIAL PRIMARY KEY,
   Name VARCHAR(255) NOT NULL,
   Email VARCHAR(255) UNIQUE NOT NULL,
   MembershipDate DATE NOT NULL
);

-- Create Loans Table
CREATE TABLE Loans (
   LoanID SERIAL PRIMARY KEY,
   BookID INT NOT NULL,
   MemberID INT NOT NULL,
   LoanDate DATE NOT NULL,
    ReturnDate DATE,
    FOREIGN KEY (BookID) REFERENCES
Books(BookID),
    FOREIGN KEY (MemberID) REFERENCES
Members(MemberID)
);
```

3.Execute SQL Commands: Click the' Execute/Refresh 'button to run the SQL commands.

Step3:Create Views

Open Query Tool:

Continue in the Query Tool or open it again if closed.

Create Views:

Enter the following SQL commands to create views.

SQL Query:

```
- Create Books View
CREATE VIEW BookDetails AS
SELECT
    BookID,
    Title,
    Author,
    ISBN,
    PublicationYear
FROM Books;

-- Create Members View
CREATE VIEW MemberDetails AS
SELECT
    MemberID,
    Name,
    Email,
    MembershipDate
FROM Members;

-- Create Loans View with JOINs
CREATE VIEW LoanDetails AS
SELECT
    Loans.LoanID,
    Books.Title AS BookTitle,
    Members.Name AS MemberName,
    Loans.LoanDate,
    Loans.ReturnDate
FROM Loans
JOIN Books ON Loans.BookID = Books.BookID
JOIN Members ON Loans.MemberID =
Members.MemberID;
```

3. Execute SQL Commands: Click the' Execute/Refresh

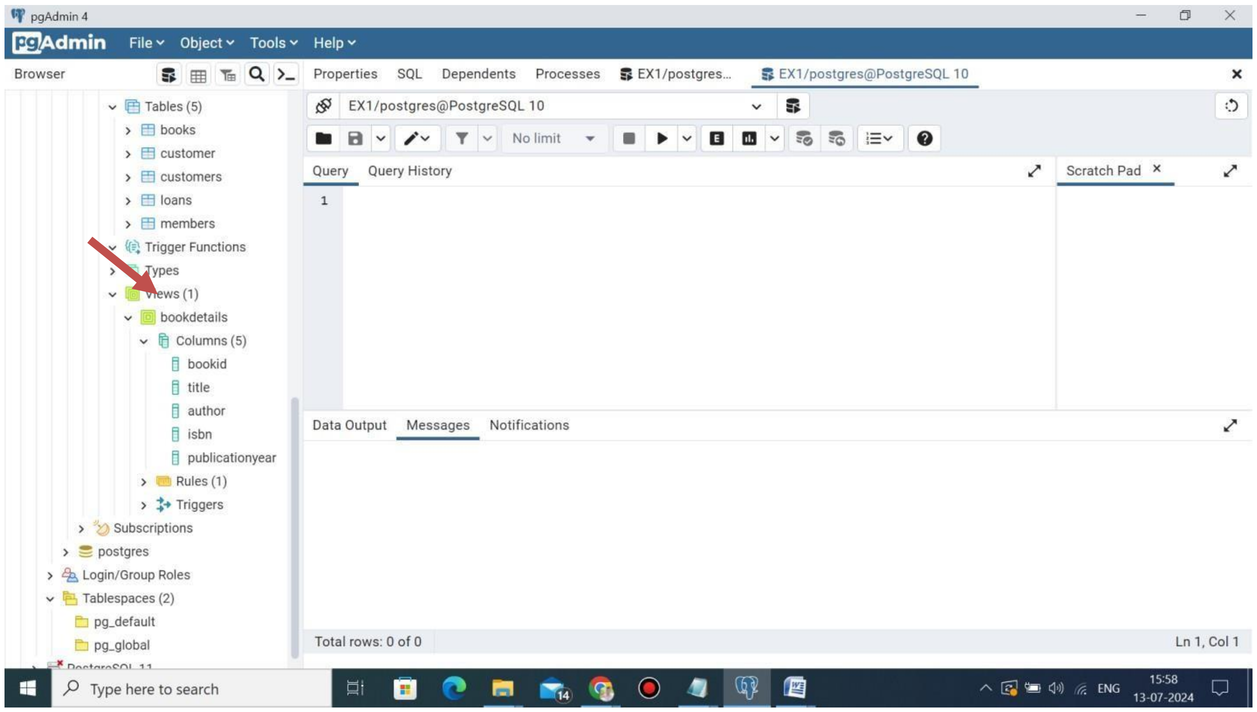
'button to run the SQL commands. **Step4:VerifyDatabase Schema**
View Tables and Views:
In the Browser tree, expand the Library DB database
.Expand the 'Schemas' node, then 'public'.
Verify that the'Tables'and'Views'nodes contain the appropriate entries.

The following constraints are commonly used in SQL:

NOTNULL-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 NOTNULL and UNIQUE. Uniquely identifies each row in a table **FOREIGNKEY**-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

OUTPUT:


pgAdmin 4
File Object Tools Help
Browser
Properties SQL Dependents Processes EX1/postgres... EX1/postgres@PostgreSQL 10
Tables (5)
books
customer
customers
loans
members
Trigger Functions
Types
Views (1)
bookdetails
Columns (5)
bookid
title
author
isbn
publicationyear
Rules (1)
Triggers
Subscriptions
postgres
Login/Group Roles
Tablespaces (2)
pg_default
pg_global
EX1/postgres@PostgreSQL 10
No limit
Query Query History
Scratch Pad
Data Output Messages Notifications
Total rows: 0 of 0
Ln 1, Col 1
Type here to search
ENG 15:58 13-07-2024

RESULT:

Thus the above program was implemented and executed successfully.

Exp:4 **PRACTICING DML COMMANDS**

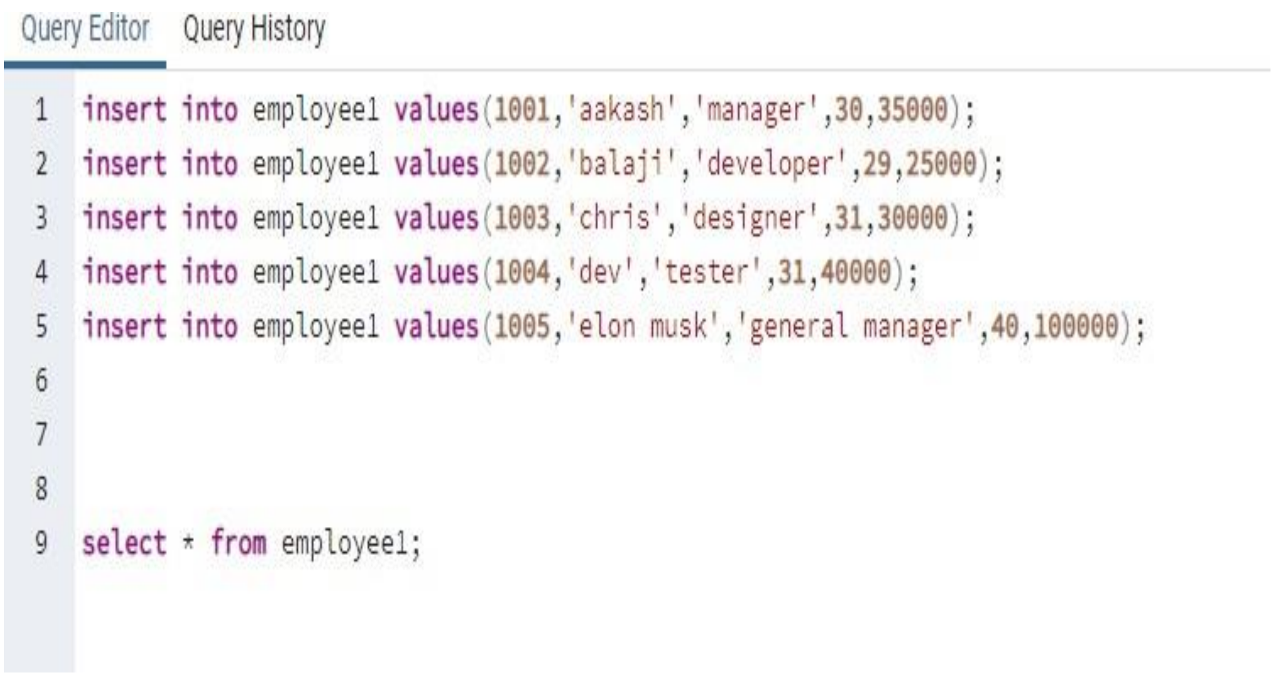
Date:

Aim:

Procedure:-

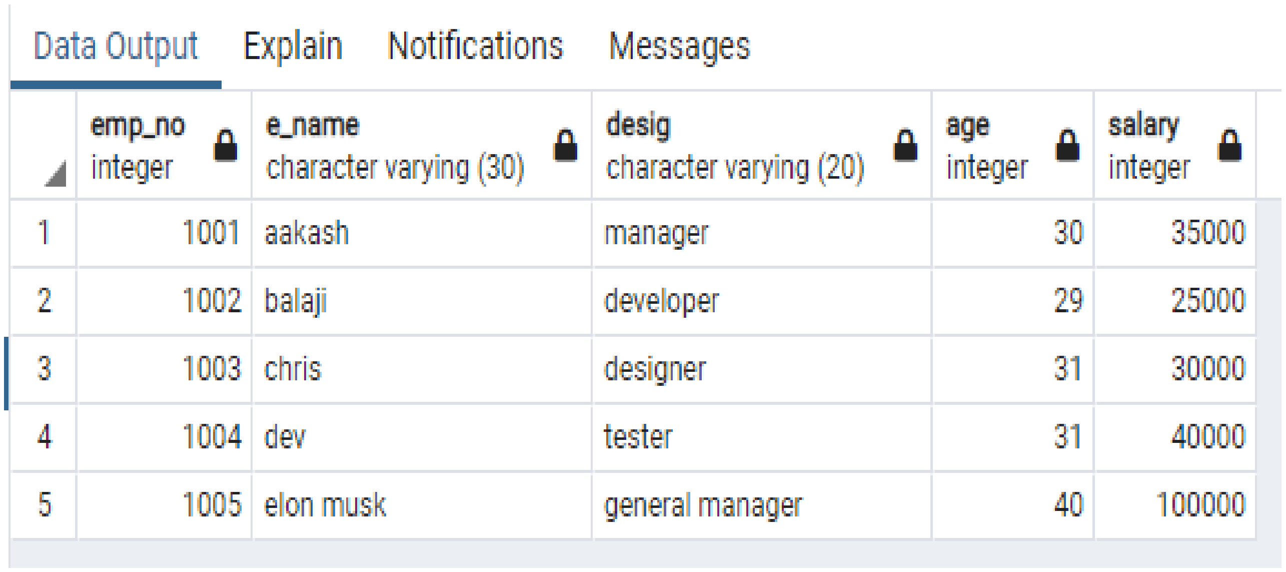
a. Insert any five records into the table.
b. Add a column to the table named place to the table using alter add command
c. Update the column details of place.
d. Delete a record from the table where the emp_no=1001.

Data Manipulation Language (DML)Commands:
Inserting Data To The Table :



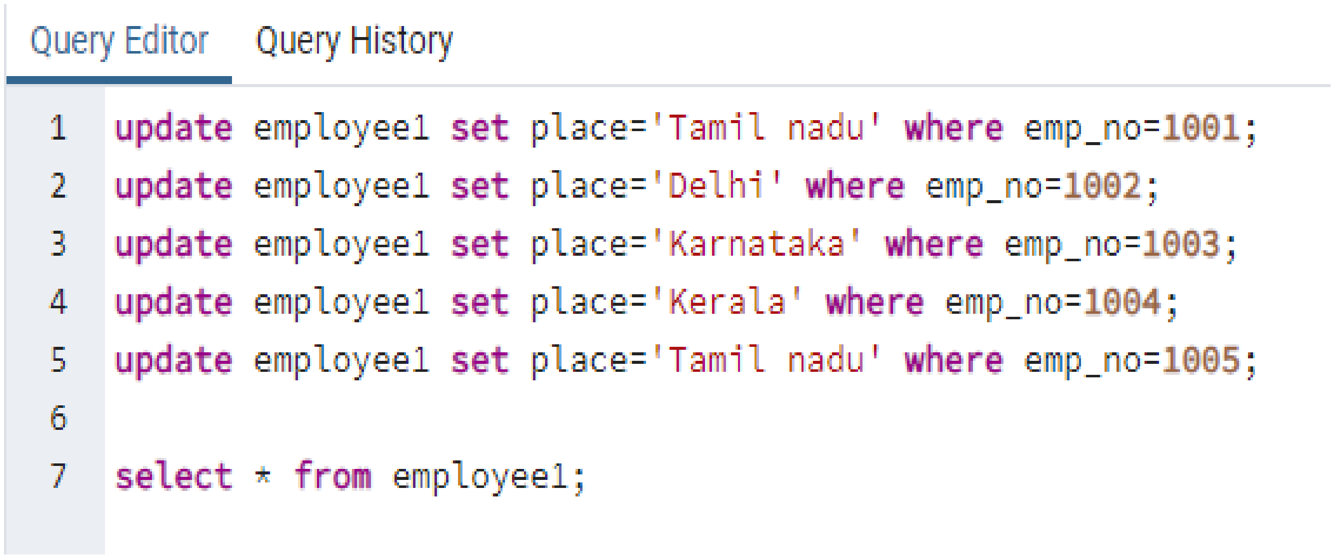
```
insert into employee1 values(1001,'aakash','manager',30,35000);
insert into employee1 values(1002,'balaji','developer',29,25000);
insert into employee1 values(1003,'chris','designer',31,30000);
insert into employee1 values(1004,'dev','tester',31,40000);
insert into employee1 values(1005,'elon musk','general manager',40,100000);



select * from employee1;
```

Output:



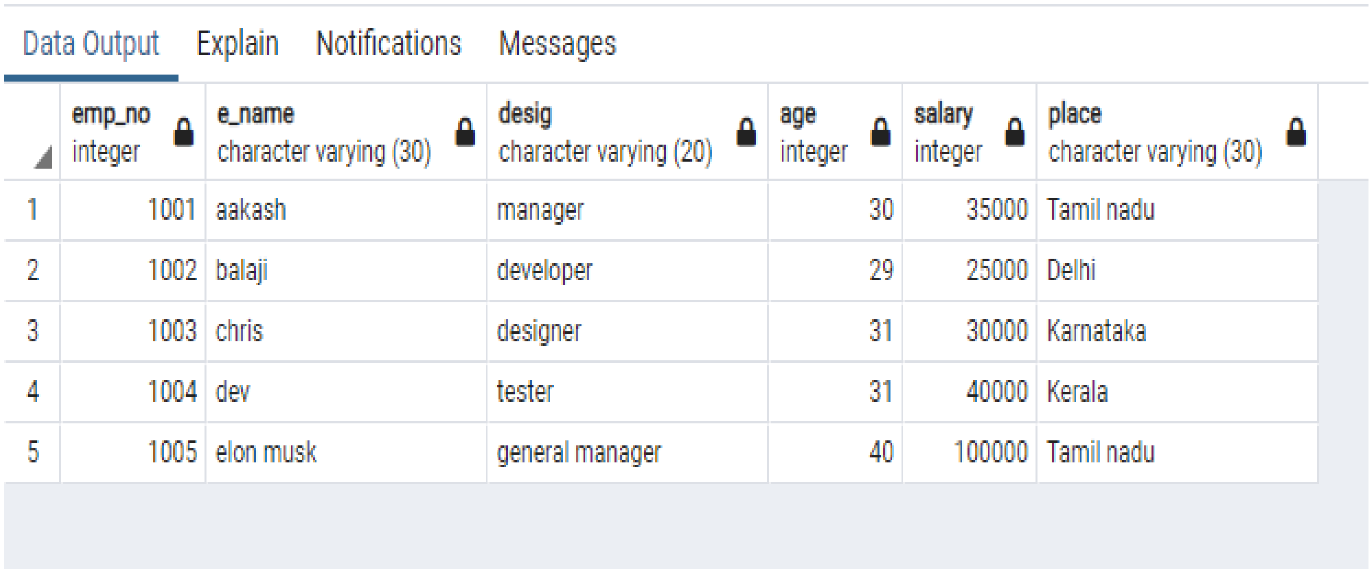
	emp_no integer	e_name character varying (30)	desig character varying (20)	age integer	salary integer
1	1001	aakash	manager	30	35000
2	1002	balaji	developer	29	25000
3	1003	chris	designer	31	30000
4	1004	dev	tester	31	40000
5	1005	elon musk	general manager	40	100000

UpdatingValuesInTheTable:



```
update employee1 set place='Tamil nadu' where emp_no=1001;
update employee1 set place='Delhi' where emp_no=1002;
update employee1 set place='Karnataka' where emp_no=1003;
update employee1 set place='Kerala' where emp_no=1004;
update employee1 set place='Tamil nadu' where emp_no=1005;

select * from employee1;
```

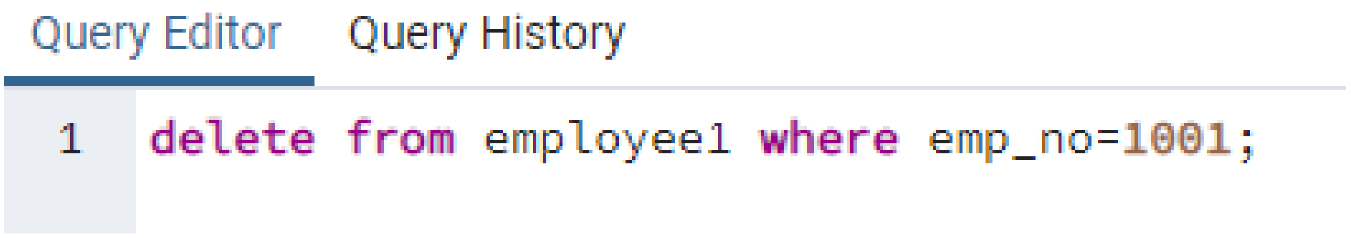
Output:



Data Output | Explain | Notifications | Messages

	emp_no integer	e_name character varying (30)	desig character varying (20)	age integer	salary integer	place character varying (30)
1	1001	aakash	manager	30	35000	Tamil nadu
2	1002	balaji	developer	29	25000	Delhi
3	1003	chris	designer	31	30000	Karnataka
4	1004	dev	tester	31	40000	Kerala
5	1005	elon musk	general manager	40	100000	Tamil nadu

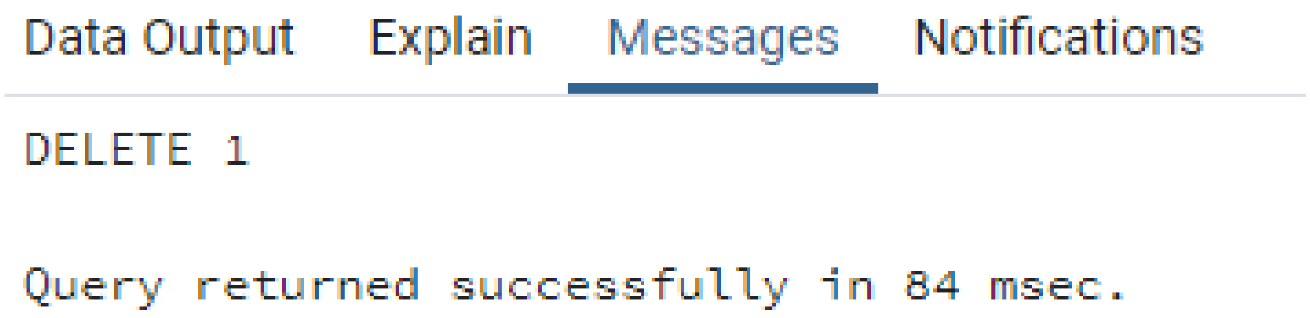
DeletingARecordFromTheTrable:



Query Editor | Query History

```
delete from employee1 where emp_no=1001;
```

OUTPUT:



Data Output | Explain | Messages | Notifications

```
DELETE 1

Query returned successfully in 84 msec.
```

Result:

Exp:5 **TRIGGERS AND STORED PROCEDURES**
Date:

Aim:

Constraints And Security Using Triggers:

Procedure:

- Create an table in the name of price-list with the following columns

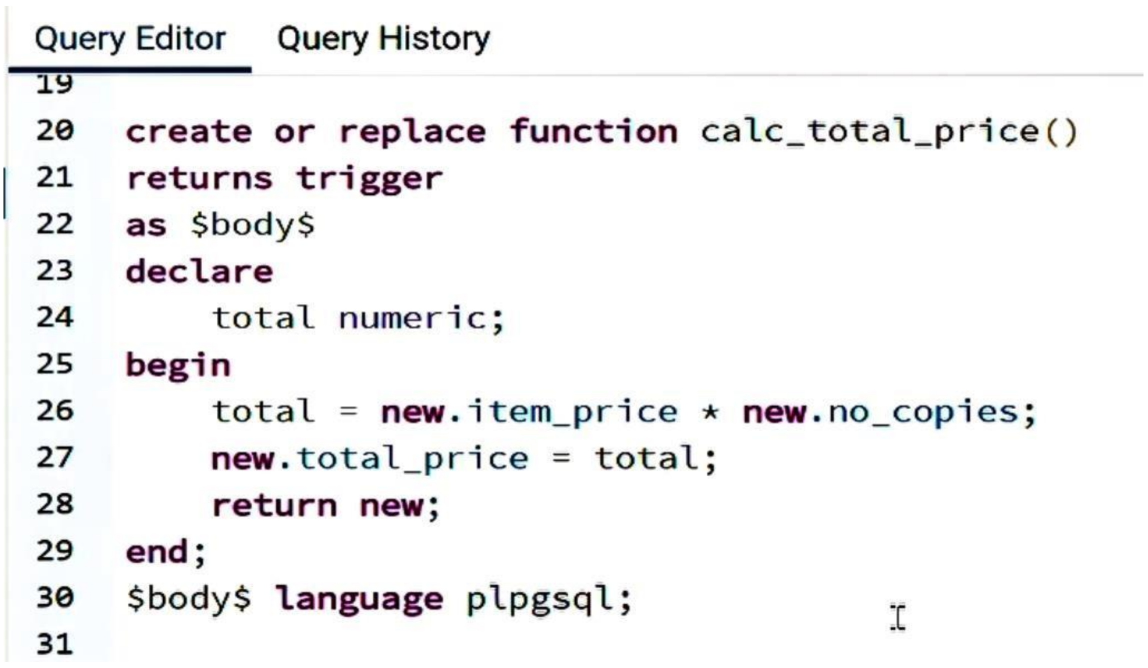
COLUMNNAME	**DATATYPE**
isbn	integer
title	varchar(50)
item_price	numeric
no_copies	integer
total_price	numeric

- Create a function **calc_total_price()** to calculate the total_price by multiplying item_price and no_copies using **create or replace function** command
- Now create a Trigger to execute the procedure **calc_total_price()**.
- Then insert the values into the table.
- View the table using **select** query

Here following two points are important and should be noted carefully:

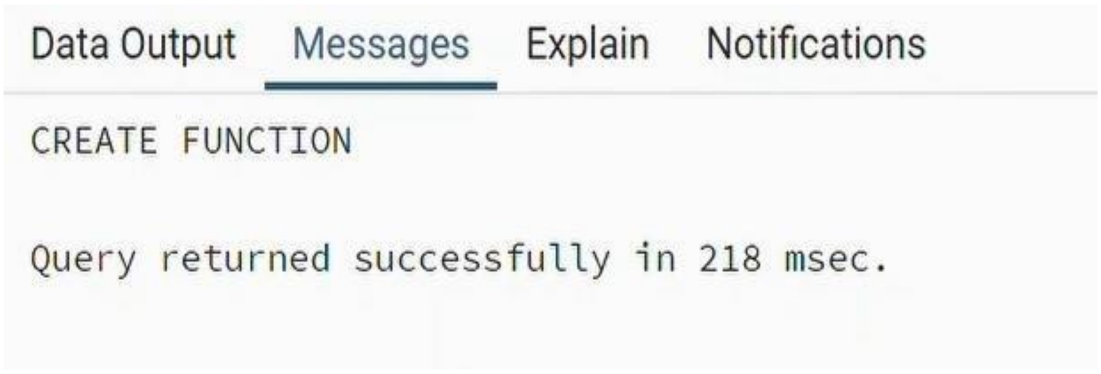
- OLD and NEW references are not available for table level triggers, rather you can use them for record level triggers.
- If you want to query the table in the same trigger, then you should use the AFTER keyword, because triggers can query the table or change it again only after the initial changes are applied and the table is back in a consistent state.
- 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 ona single or multiple operations, for example BEFORE DELETE, which will fire whenever a record will be deleted using DELETE operation on the table.

Query:



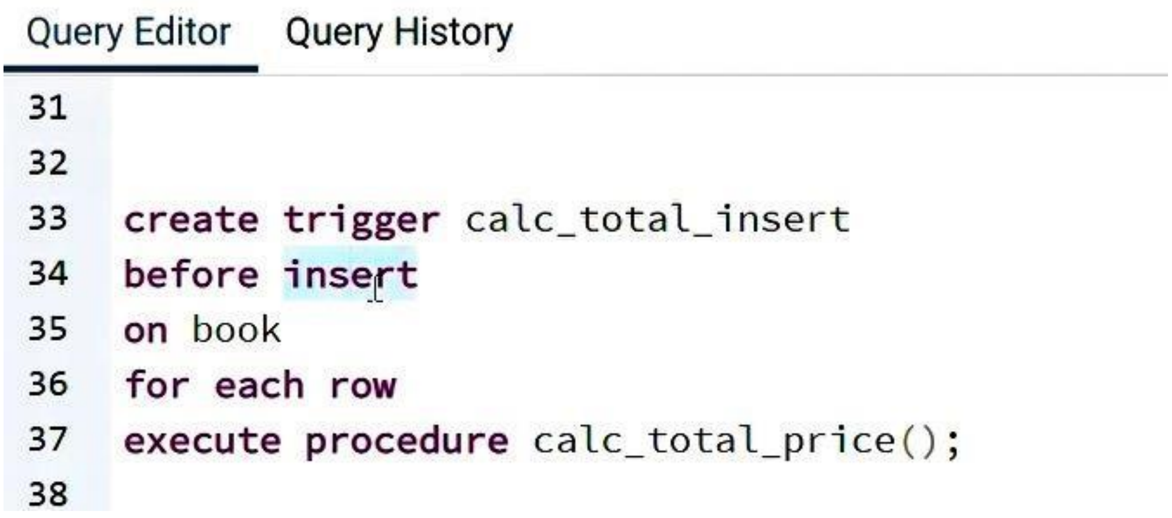
```
create or replace function calc_total_price()
returns trigger
as $body$
declare
    total numeric;
begin
    total = new.item_price * new.no_copies;
    new.total_price = total;
    return new;
end;
$body$ language plplpgsql;
```

OUTPUT :-



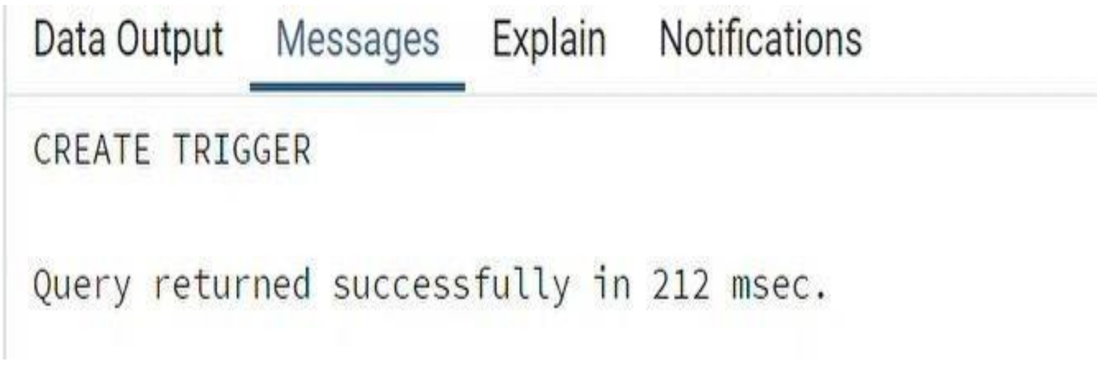
```
CREATE FUNCTION

Query returned successfully in 218 msec.
```

Query:



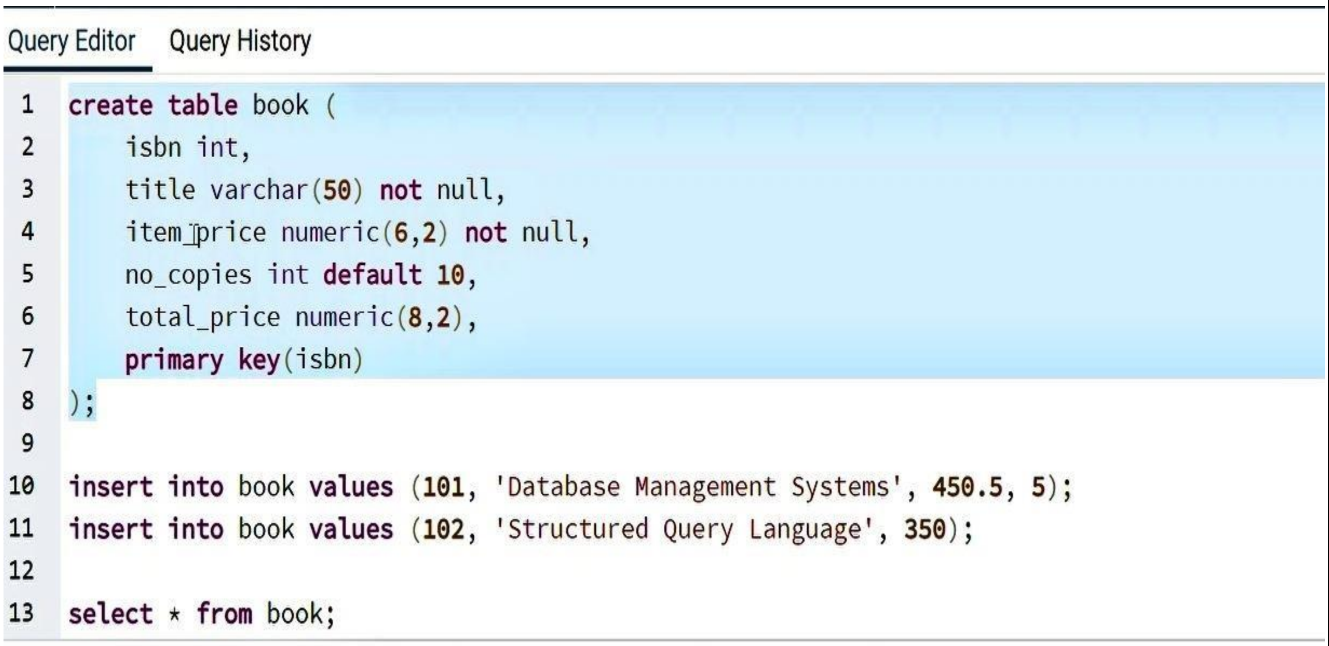
```
create trigger calc_total_insert
before insert
on book
for each row
execute procedure calc_total_price();
```

Output:



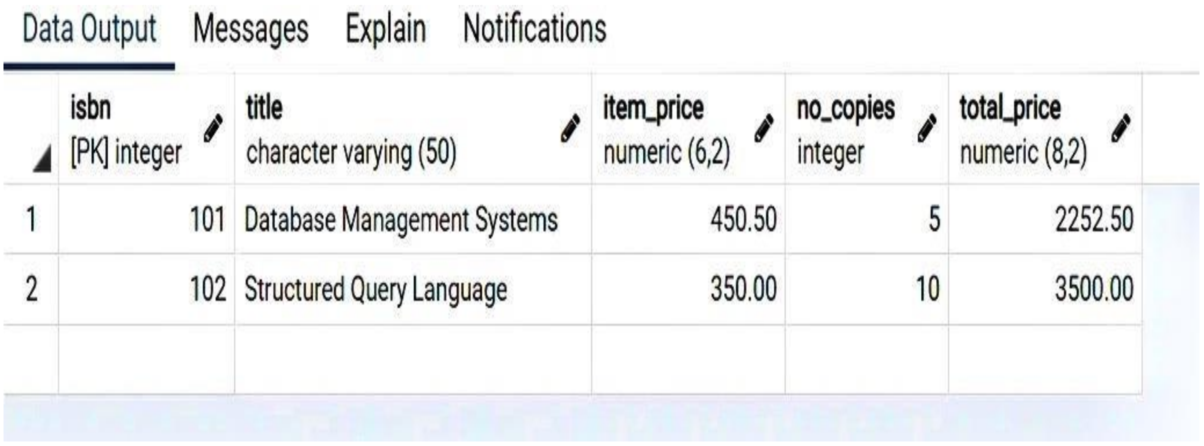
```
CREATE TRIGGER

Query returned successfully in 212 msec.
```

<u>Query</u>:



```
create table book (
    isbn int,
    title varchar(50) not null,
    item_price numeric(6,2) not null,
    no_copies int default 10,
    total_price numeric(8,2),
    primary key(isbn)
);

insert into book values (101, 'Database Management Systems', 450.5, 5);
insert into book values (102, 'Structured Query Language', 350);

select * from book;
```

<u>Output</u>:



	isbn [PK] integer	title character varying (50)	item_price numeric (6,2)	no_copies integer	total_price numeric (8,2)
1	101	Database Management Systems	450.50	5	2252.50
2	102	Structured Query Language	350.00	10	3500.00

<u>StoredProcedures/Functions</u>:

<u>Definition</u>:

AstoredprocedureisapreparedSQLcodethat youcansave,sothecodecanbe reused over and over again. So if you have an SQL query that you write over and over again, save it as a stored procedure, and then just call it to execute it.

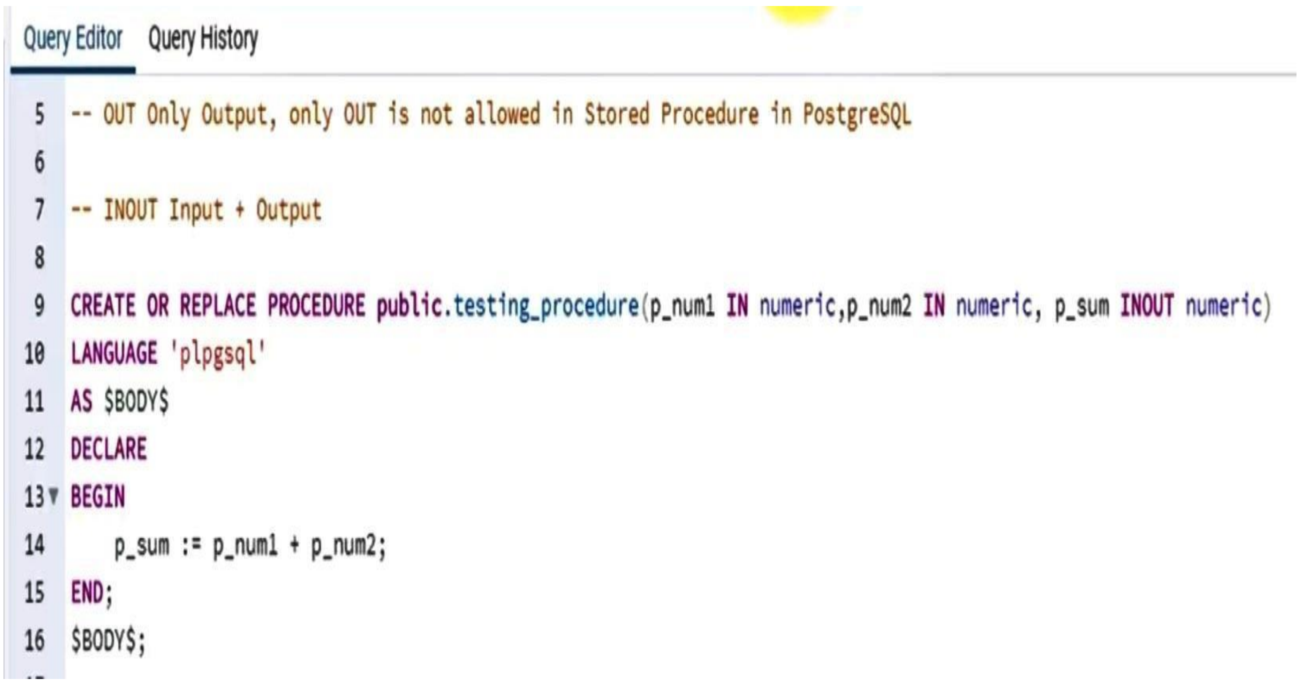
Procedure:

- Create a table in the name of sum with the following columns:

COLUMNNAME	DATATYPE
p_num1	numeric
p_num2	numeric
p_sum	numeric

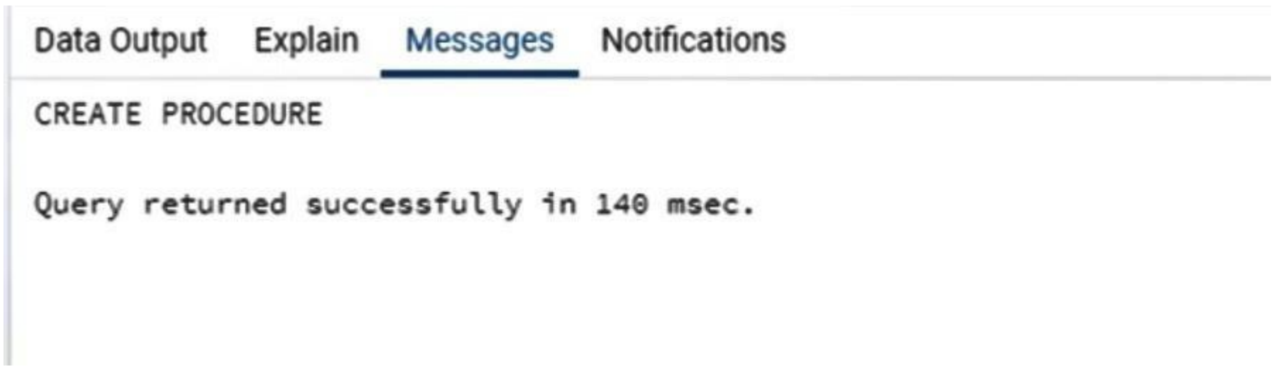
- Create a procedure**testing_procedure( )** tocalculatethep_sumbyaddingthe p_num1and p_num2 using **create or replace procedures()** statement.
- Calltheprocedureswiththevaluesusingthe**callprocedure_name()**query

QUERY:

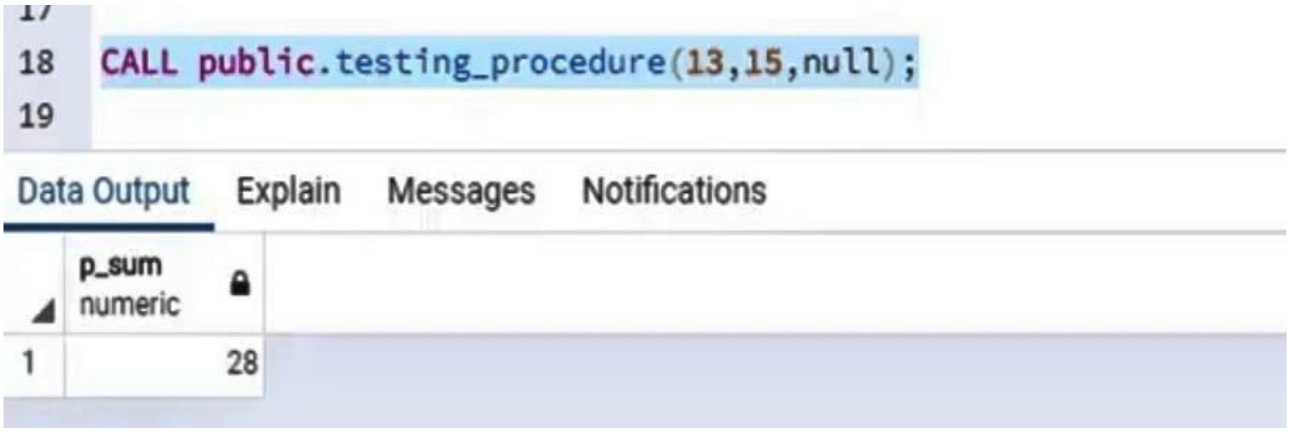


```
-- OUT Only Output, only OUT is not allowed in Stored Procedure in PostgreSQL

-- INOUT Input + Output

CREATE OR REPLACE PROCEDURE public.testing_procedure(p_num1 IN numeric,p_num2 IN numeric, p_sum INOUT numeric)
LANGUAGE 'plpgsql'
AS $BODY$
DECLARE
BEGIN
    p_sum := p_num1 + p_num2;
END;
$BODY$;
```

Output:



```
CREATE PROCEDURE

Query returned successfully in 140 msec.
```



```
CALL public.testing_procedure(13,15,null);
```

	p_sum numeric
1	28

<u>Result</u>:

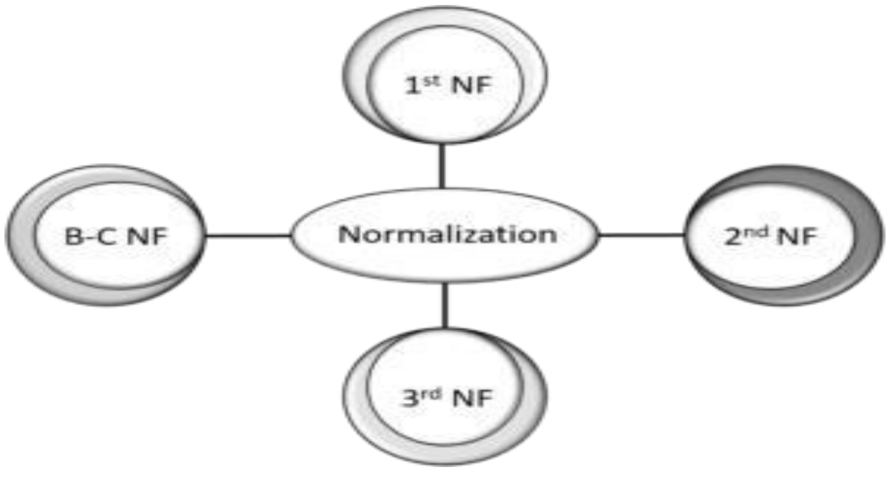
Exp: 6 Database Design Using Normalization Bottom-Up Approach

Date:

<u>Aim:</u>

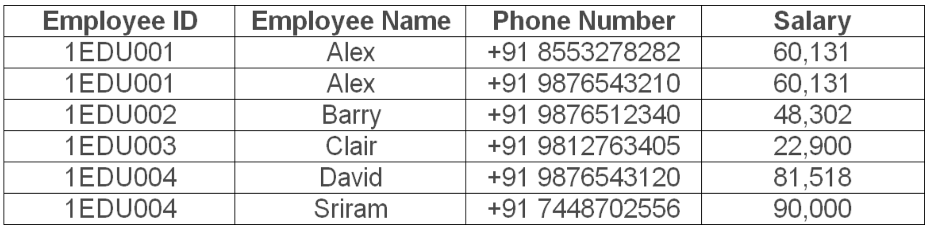
Normalization:

- It is the processes of reducing the redundancy of data in the table and also improving the data integrity. So why is this required? WithoutNormalizationin SQL, we may face many issues such as.
- ***Insertion anomaly***: It occurs when we cannot insert data to the table without the presence of another attribute.
- ***Update anomaly***:It is adata inconsistency that results from data redundancyand a partial update of data.
- ***Deletion Anomaly***: It occurs when certain attributes are lost because of the deletion of other attributes.
- Normalization entails organizing the columns and tables of a database to ensure that their dependencies are properly enforced by database integrity constraints.
- It usually divides a large table into smaller ones, so it is more efficient. In 1970 the First Normal Form was defined by *Edgar F Codd* and eventually, other Normal Forms were defined.
- Normalization in SQL will enhance the distribution of data. Now let’s understand each and every Normal Form with examples.



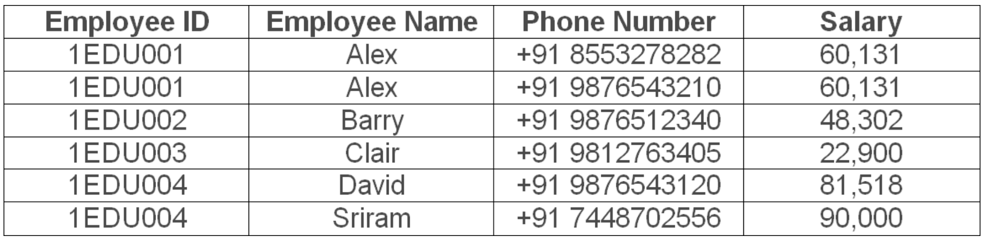
1stNormalForm(1NF)

- In this Normal Form, we tackle the problem of atomicity. Here atomicity means values in the table should not be further divided. In simple terms, a single cell cannot hold multiple values. If a table contains a composite or multi-valued attribute, it violates the First Normal Form.



Employee ID	Employee Name	Phone Number	Salary
1EDU001	Alex	+91 8553278282	60,131
1EDU001	Alex	+91 9876543210	60,131
1EDU002	Barry	+91 9876512340	48,302
1EDU003	Clair	+91 9812763405	22,900
1EDU004	David	+91 9876543120	81,518
1EDU004	Sriram	+91 7448702556	90,000

- In the above table, we can clearly see that the Phone Number column has two values. Thus it violated the 1st NF. Now if we apply the 1st NF to the above table we get the below table as the result.



Employee ID	Employee Name	Phone Number	Salary
1EDU001	Alex	+91 8553278282	60,131
1EDU001	Alex	+91 9876543210	60,131
1EDU002	Barry	+91 9876512340	48,302
1EDU003	Clair	+91 9812763405	22,900
1EDU004	David	+91 9876543120	81,518
1EDU004	Sriram	+91 7448702556	90,000

  - We have achieved atomicity and also each and every column have unique values.

2ndNormalForm (2NF)

- The first condition in the 2nd NF is that the table has to be in 1st NF. The table also should not contain partial dependency. Here partial dependency means the proper subset of candidate key determines a non-prime attribute.

EMPLOYEEID	DEPARTMENTID	OFFICE LOCATION
1EDU001	ED-T1	Pune
1EDU002	ED-S2	Bengaluru
1EDU003	ED-M1	Delhi
1EDU004	ED-T3	Mumbai

- This table has a composite primary key**Employee ID**,**Department ID**. The non-key attribute isOffice Location. In this case,Office Locationonly depends on Department ID, which is only part of the primary key.
- Therefore, this table does not satisfy the second Normal Form. To bring thistable to Second Normal Form, we need to break the table into two parts.

EMPLOYEEID	DEPARTMENTID
1EDU001	ED-T1
1EDU002	ED-S2
1EDU003	ED-M1
1EDU004	ED-T3

DEPARTMENTID	OFFICE LOCATION
ED-T1	Pune
ED-S2	Bengaluru
ED-M1	Delhi
ED-T3	Mumbai

- Inthetable,thecolumn OfficeLocationisfullydependentontheprimarykeyof that table, which is Department ID.

3rdNormalForm(3NF)

- Thetablehastobein2NFbeforeproceedingto3NF.Theotherconditionis there should be no transitive dependency for non-prime attributes.
- That means non-prime attributes (which doesn't form a candidate key) shouldnot be dependent on other non-prime attributes in a given table.
- Soatransitivedependencyisafunctionaldependencyinwhich$X \rightarrow Z$(X determines Z) indirectly, by virtue of $X \rightarrow Y$ and $Y \rightarrow Z$.

STUDENTID	STUDENT NAME	SUBJECT ID	SUBJECT	ADDRESS
1DT15ENG01	Alex	15CS11	SQL	Goa
1DT15ENG02	Barry	15CS13	JAVA	Bengaluru
1DT15ENG03	Clair	15CS12	C++	Delhi
1DT15ENG04	David	15CS13	JAVA	Kochi

- In the above table, **StudentID** determines **SubjectID**,and **Subject ID** determines **Subject**.
- Therefore,**Student ID** determines**Subject**via**Subject ID.**This implies that we have a transitive functional dependency, and this structure does not satisfy the third normal form.

STUDENT	STUDENT NAME	SUBJECT ID	ADDRESS
1DT15ENG01	Alex	15CS11	Goa
1DT15ENG02	Barry	15CS13	Bengaluru
1DT15ENG03	Clair	15CS12	Delhi
1DT15ENG04	David	15CS13	Kochi

SUBJECT ID	SUBJECT
15CS11	SQL
15CS13	JAVA
15CS12	C++
15CS13	JAVA

- The above tables all the non-key attributes are now fully functional dependent only on the primary key.
- In the first table, columns **Student Name,Subject ID**and**Address**are only dependenton **StudentID**.Inthesecondtable, **Subject**isonlydependent on **Subject ID**.

BoyceCoddNormalForm (BCNF)

- This is also known as 3.5 NF. It's the higher version 3NF and was developed by Raymond F. Boyce and Edgar F. Codd to address certain types of anomalies which were not dealt with 3NF.
- Thetablehastosatisfy3rdNormal Form.
- In BCNF if every functional dependency **A → B**, then **A**has to be the**Super Key** of that particular table.

STUDENTID	SUBJECT	PROFESSOR
1DT15ENG01	SQL	Prof. Mishra
1DT15ENG02	JAVA	Prof. Anand
1DT15ENG02	C++	Prof. Kanthi
1DT15ENG03	JAVA	Prof. Anand
1DT15ENG04	DBMS	Prof. Lokesh

- One student can enroll for multiple subjects.
- There can be multiple professors teaching one subject.
- And,for each subject,a professor is assigned to the student.
- In the table **StudentID,** and **Subject** form the primary key, which means the **Subject** column is a **prime attribute**.But,there is one more dependency, **Professor → Subject**.
- And while **Subject** is a prime attribute, **Professor** is a **non-prime attribute**, which is not allowed by BCNF.
- Dividing the table into two parts. One table will hold Student ID which already exists and newly created column Professor ID.

STUDENTID	PROFESSORID
1DT15ENG01	1DTPF01
1DT15ENG02	1DTPF02
1DT15ENG02	1DTPF03

- And in these cond table,we will have the columns Professor ID, Professor and Subject.

PROFESSORID	PROFESSOR	SUBJECT
1DTPF01	Prof. Mishra	SQL
1DTPF01	Prof. Anand	JAVA
1DTPF01	Prof. Kanthi	C++
:	:	:

- By this we satisfiying the Boyce Codd Normal Form.

Bottom–upapproach:

- Normalisation is a bottom-up approach which starts with a collection of attributesandorganisesthemintowell-structuredrelationswhicharefreefrom redundant data.

BCNF:Boyce-CoddNormalForm

Result:

Exp:7: Develop A DatabaseApplicationUsingIDE/RADTools
Date:

Aim:

Procedure:

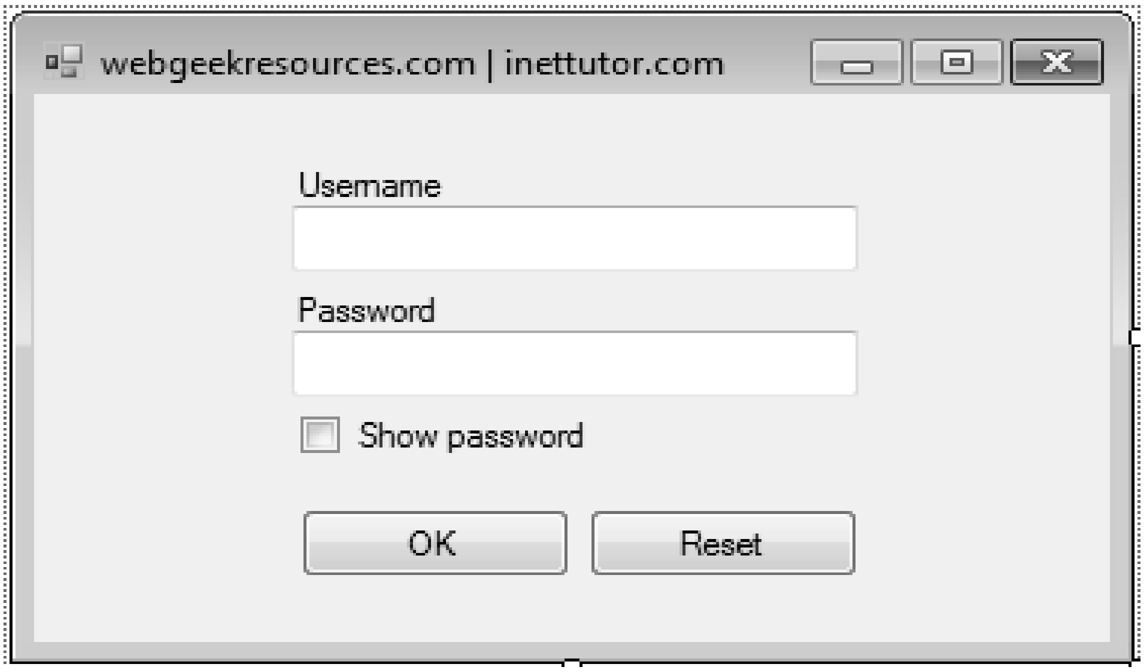
- Open you MicrosoftVisualStudio2010,2012orhigher,orjustyour Microsoft Visual Basic .Net.
- Create a newproject(selectFileandNewProject).For visual studio user: (select Visual Basic then Windows Form Application).
- Here is the sample form layout or design.Feel free to design your form.We need to add the following controls:
  - 2labels
  - 2textboxes
  - 2buttons
  - 1checkbox.
- The program will first validate the input of the user, the user must enter a username and password or else a message will appear that will notify the user that username and password field is required.
- The program will then match or compare the user input to the criteria of the program. The username must be admin and password must also be admin which means that the username and password combination must be admin or else a message will prompt you that your username and password is incorrect.
- To clear the username and password field,kindly double click the Reset button and paste the code below.

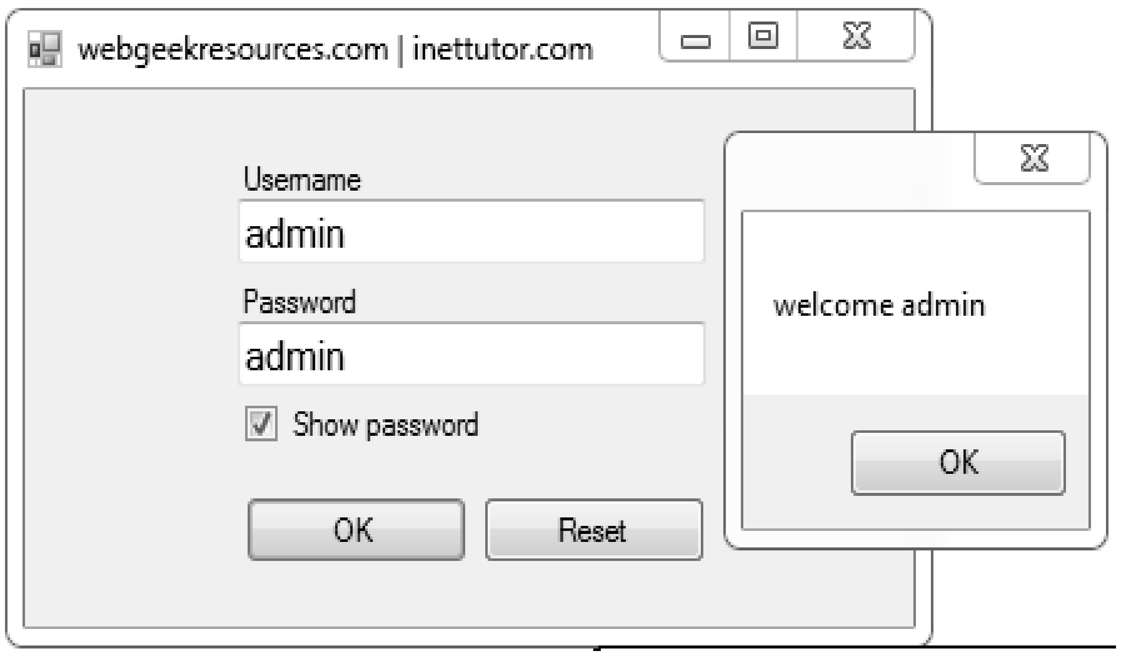
  ```
  TextBox1.Clear()
  TextBox2.Clear()
  ```
- Additional feature of this program is to allow the user to view or to make its password visible or in simplest explanation is to view what you are typing in the password field. Kindly double click the Show password checkbox and paste the line of codes below.

Program:

```
If TextBox1.Text = "" Then
MessageBox.Show("Pleaseenterusername")
TextBox1.Focus()
ExitSub
ElseIf TextBox2.Text = "" Then
MessageBox.Show("Pleaseenterpassword")
TextBox2.Focus()
ExitSub
End If
IfTextBox1.Text="admin"AndTextBox2.Text="admin"Then
MessageBox.Show("welcome admin")
Else
MessageBox.Show("incorrectusernameorpassword")
End If

IfCheckBox1.Checked=TrueThen

TextBox2.PasswordChar = ""
Else
TextBox2.PasswordChar="*" End
If
```

Output:





webgeekresources.com | inettutor.com
Username
admin
Password
admin
Show password
OK
Reset
welcome admin
OK

Result:

Exp:8 Database design using EERto-ODBmapping/UMLclassdiagrams
Date:

Aim:

Procedure:

Mapping an EERSchema to an ODB Schema

- It is relatively straight forward to design the type declarations of object classes for an ODBMS from an EER schema that contains neither categories nor n ary relationships with n > 2.
- However, the operations of classes are not specified in the EER diagram and must be added to the class declarations after the structural mapping is completed. The outline of the mapping from EER to ODL is as follows:

Step 1.

- Create an ODL class for each EER entity type or subclass. The type of the ODLclass should include all the attributes of the EER class.
- Multivalued attributes are typically declared by using the set, bag, or list constructors. If the values of the multivalued attribute for an object should be ordered, the list constructor is chosen; if duplicates are allowed, the bag constructor should be chosen; otherwise, the set constructor is chosen.
- Composite attributes are mapped into a tuple constructor (by using a struct declaration in ODL).

Step 2.

- Add relationship properties or reference attributes for each binary relationship into the ODL classes that participate in the relationship. These may be created in one or both directions.
- If a binary relationship is represented by references in both directions, declare the references to be relationship properties that are inverses of one another, if such a facility exists.

- If a binary relationship is represented by a reference in only one direction, declare the reference to be an attribute in the referencing class whose type is the referenced class name.
- Depending on the cardinality ratio of the binary relationship, the relationship properties or reference attributes may be single-valued or collection types. They will be single valued for binary relationships in the 1:1 or N:1 directions

Step 3.

- Include appropriate operations for each class. These are not available from the EER schema and must be added to the database design by referring to the original requirements.
- A constructor method should include program code that checks any constraints that must hold when a new object is created.
- A destructor method should check any constraints that may be violated when an object is deleted.

Step 4.

- An ODL class that corresponds to a subclass in the EER schema inherits the typeand methods of its super class in the ODL schema.

Step 5.

- Weak entity types can be mapped in the same way as regular entity types. An alternative mapping is possible for weak entity types that do not participate in any relationships except their identifying relationship
- these can be mapped as though they were composite multivalued attributes of the owner entity type, by using the set < struct < ... >> or list < struct < ... >> constructors. The attributes of the weak entity are included in the struct < ... > construct, which corresponds to a tuple constructor.

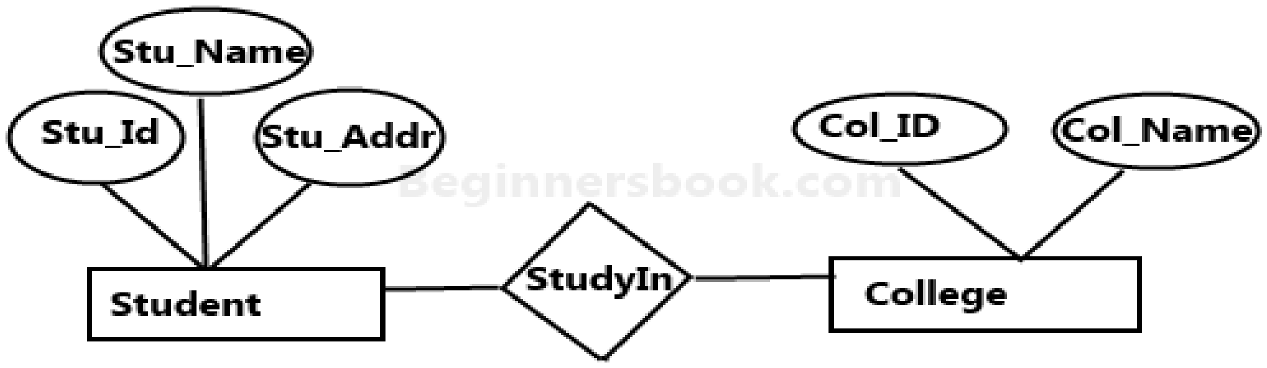
Step 6.

- Categories (union types) in an EER schema are difficult to map to ODL. It is possible to create a mapping similar to the EER-to-relational mapping
- By declaring a classto representthecategoryanddefining1:1relationships between the category and each of its super classes.

Step 7.

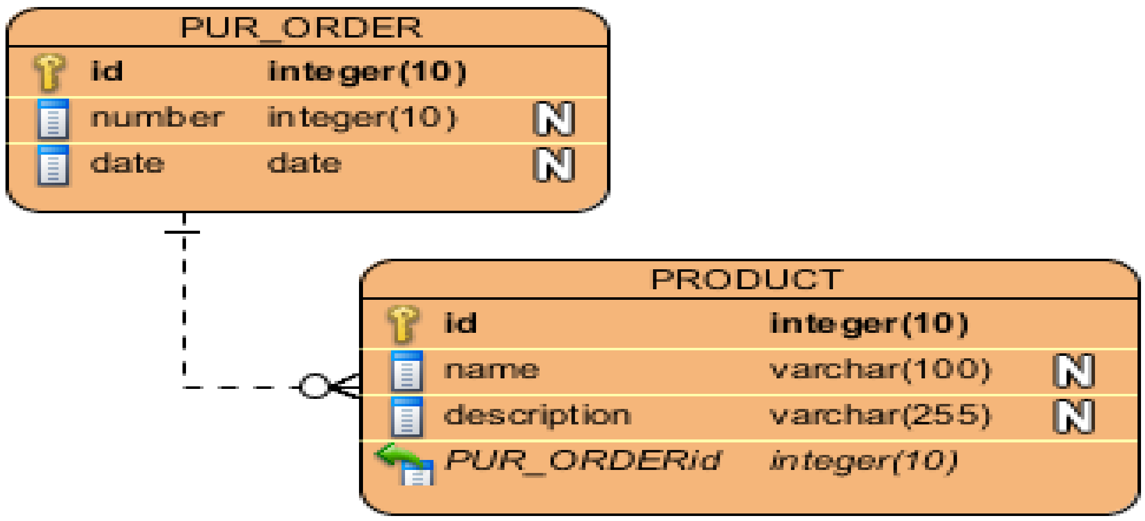
- An n-ary relationship with degree n > 2 can be mapped into a separate class, with appropriate references to each participating class.
- These references are based on mapping a 1:N relationship from each class that represents a participating entity type to the class that represents the n-ary relationship.
- AnM:Nbinary relationship,especiallyifit contains relationship attributes,may also use this mapping option, if desired.
- The mapping has been applied to a subset of the UNIVERSITY database schemain the context of the ODMG object database standard. The mapped object schemausing the ODL notation is shown.

EERtoODBmapping diagram:



Sample E-R Diagram

EER(UMLclassdiagram):



Result:

Exp:9 **OBJECT FEATURES OF SQL-UDTs**

Date:

Aim:

ObjectsofSQL:

- SQL objects are schemas, journals, catalogues, tables, aliases, views, indexes, constraints, triggers, sequences, stored procedures, user-defined functions,user-defined types, global variables, and SQL packages, SQL creates and maintains these objects.

UDT in SQL:

- The UDT is similar to an alias datatype and it uses the existing datatypes in SQL server or Azure SQL database.
- SQL server supports two kinds of user defined types
  - ➢ User-defined data type.
  - ➢ User-defined table type

Use of UDT in sqlserver:

- User defined type can be used in the definition of database objects such as variables in transact-SQL batches, in functions and stored procedures, and as arguments in functions and stored procedures.

Sub-types of UDT in SQL:

- ➢ Exact numeric.
- ➢ Approximate numeric.
- ➢ Date and Time.
- ➢ Character String.
- ➢ Unicode character strings.

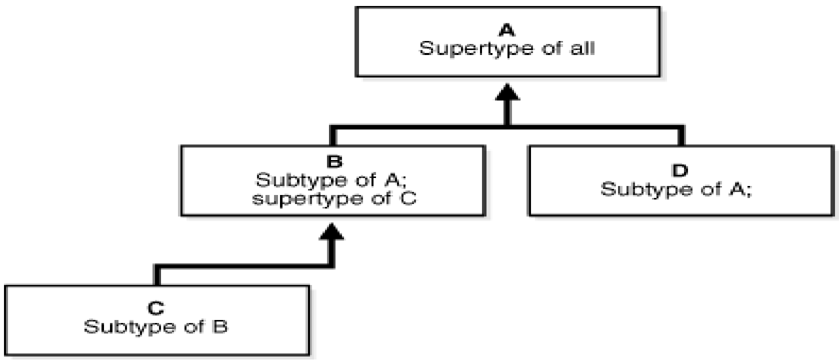
- CLR datatypes.
- Spatial datatypes

Tables using UDTs:

- There is no special syntax for creating a UDT column in a table.You can use the name of the UDT in column definition as though it were one of the intrinsic SQL server data types. The following CREATE TABLE Transact- SQL statement creates a table named points, with a column named ID, which is defined as an into identity column is named PointgValue, with a data type of Point.

InheritanceinSQLobject types:

- SQL object inheritance is based on a family tree of object types that forms a type hierarchy. The type hierarchy consists of a parent object type, called a supertype,and one or more levels of child object types, called subtypes,which are derived from the parent.
- A subtype can be derived from a super type either directly or indirectly through intervening levels of other subtypes.
- A super type can have multiple sibling subtypes, but a subtype can have atmost one direct parent super type (single inheritance).



Method Definition:

- A **method** is procedure or function that is part of the object type definition, and that can operate on the attributes of the type. Such methods are also called **member methods**, and they take the keyword `MEMBER` when you specify them as a component of the object type.
- Method specification

- Method names
- Method name overloading

<u>ImplementingMethods</u>

To implement a method,create the PL/SQL code and specify it within a CREATE TYPE BODY statement.

For example,consider the following definition of an object type named *rational type*:

```
CREATE TYPE rational_typeASOBJECT (
numerator INTEGER,
  Denominator INTEGER,
  MAP MEMBER FUNCTION rat_to_realRETURNREAL,
  MEMBER PROCEDURE normalize,
  MEMBER FUNCTION plus(xrational_type)
      RETURN rational_type);
```

<u>Example:</u>The following definition is shown merely because it defines the func- tiong cd,which is used in the definition of the normalize method in the CREATE TYPE BODY statement later in this section.

```
CREATE FUNCTION gcd(xINTEGER,yINTEGER)RETURNINTEGERAS
--Find greatest common divisor of x and y. For example ,if
--(8,12)is input, the greatest common divisor is 4.
--This will be used in normalizing(simplifying)fractions.
--(You need not try to understand how this code works ,unless
--you are a math wizard. It does.)
--
   ansINTEGER;
BEGIN
   IF(y<=x)AND (xMODy=0) THEN
      ans:=y;
   ELSIFx<yTHEN
      ans:=gcd(y,x);--Recursivecall ELSE
      ans:=gcd(y,xMODy);--Recursivecall END
   IF;
   RETURNans;
END;
```

Result:

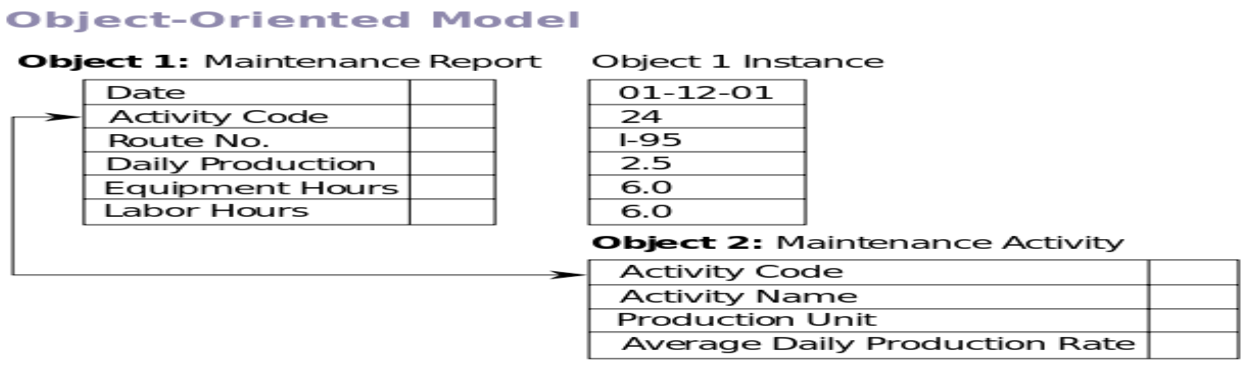
Exp: 10 Querying the Object-relational database using ObjectQueryLanguage

Date:

Aim:

Object–relationaldatabase

- An object–relational database (ORD), or object–relational database management system(ORDBMS),is a database management system (DBMS)similar to a relational database, but with an object-oriented database model: objects, classes and inheritance are directly supported in database schemas and in the query language. In addition, just as with pure relational systems, it supports extension of the data model with custom data types and methods.



- An object–relational database can be said to provide a middle ground between relational databases and object-oriented databases. In object–relational databases, the approach is essentially that of relational databases.
- The data resides in the database and is manipulated collectively with queries in a query language.
- At the other extreme are OODBMS in which the database is essentially a persistent objectstoreforsoftwarewritteninanobject-orientedprogramminglanguage,witha

programming API for storing and retrieving objects, and little or no specific support for querying.

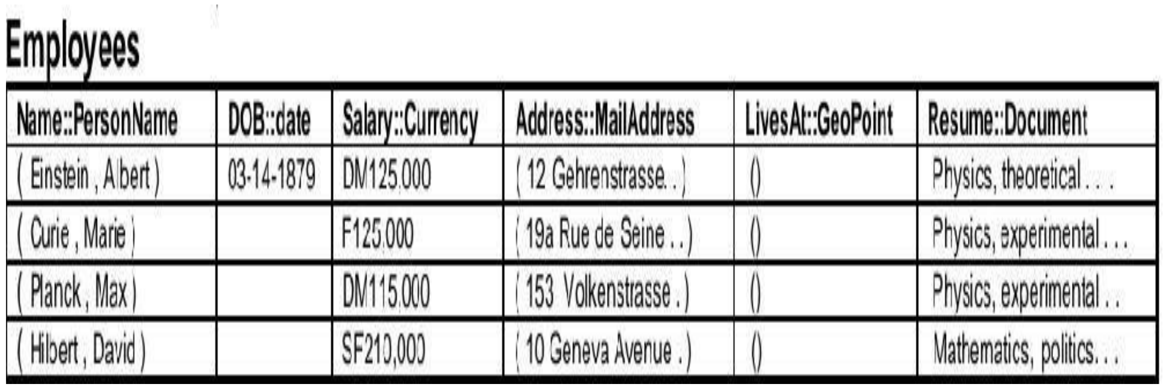
Procedure:

- CREATE.
- INSERT.
- UPDATE.
- DELETE

Program:

```
CREATE TABLE Employees(First Name VARCHAR(32)
NOT NULL,
Surname VARCHAR(64)NOTNULL,DOBDATENOT NULL,
Salary DECIMAL(10,2)NOTNULLCHECK(Salary>0.0),
Address_1VARCHAR(64)NOTNULL,Address_2VAR-
CHAR(64) NOT NULL,
City VARCHAR(48) NOT NULL, State CHAR(2) NOT
NULL, Zip Code INTEGERNOTNULL,PRIMARYKEY (
Surname, First Name, DOB));

INSERT INTO Employees(Pager_Number,Pass_Code,Mes-
sage )
SELECT E.Pager_Number,E.Pass_Code,
Print(E.Name)||':Call1-800-TEMPS-R-USforimmediate
INFORMIX DBA job'
FROMTemporary_EmployeesE
WHEREContains(GeoCircle('(-122.514,37.221)','60
miles')),E.LivesAt)
ANDDocContains(E.Resume,'INFORMIXandDatabase
Administrator')
ANDNOTIsBooked(Period(TODAY,TODAY+ 7),E.Booked );

SELECT*FROMEmployees;
```

Output:



Employees

Name::PersonName	DOB::date	Salary::Currency	Address::MailAddress	LivesAt::GeoPoint	Resume::Document
(Einstein , Albert)	03-14-1879	DM125.000	(12 Gehrenstrasse. .)	()	Physics, theoretical . . .
(Curie , Marie)		F125.000	(19a Rue de Seine . .)	()	Physics, experimental . . .
(Planck , Max)		DM115.000	(153 Volkenstrasse .)	()	Physics, experimental . .
(Hilbert , David)		SF210,000	(10 Geneva Avenue .)	()	Mathematics, politics. . .

Result: