DATABASE DESIGN AND MANAGEMENT LAB MANUAL

RECORD NOTE BOOK

Name of the Student	:
Register Number	:
Department / Semester	:
Subject Title / Code	:

INDEX

Ex.No.	Date	Title of the experiment	Page.No.	Staff Initial

Exp:1 **Database Development Lifecycle of Banking Management**
Date:

Aim:

Steps:

Database Planning : To plan the effective development of banking, the efficient way to use it by the users

System Definition: The system definition is done to manage the scope and the range of boundaries. A bank collects money, cheques, bills and drafts. It accepts deposits from the public & lends loan to those who are in need of it.

Requirements Collection And Analysis :

1) The XYZ bank can have many automated teller machines(ATMs), and the new system shall provide functionally on all ATMs.
2) The bank performs 3 types of functions;
    a) Withdrawal of funds
    b) Query of account balance
    c) Transfer of funds from one bank account to another in the same bank
3) The ATM card must be authorized and issues by the
   bank.
4) The system shall allow the customer to enter the Correct PIN in no more three attempts .The failure this will lead to confiscation of the ATM card
5) The banking system also identifies that whether there is sufficient amount in the bank before transaction
6) The customer records , account records and debit card records will all be maintained at the server and shall not be the responsibility of the system
7) The system shall be linked with the bank server through communication systems, which are beyond the scope of the current system. It is assumed that this facility is always available.

Database Design: The database is designed in such a way that it contains the details of the account complaints, customer, interest, loan and transactions.

Selection of DBMS: The DBMS must be selected for the database.

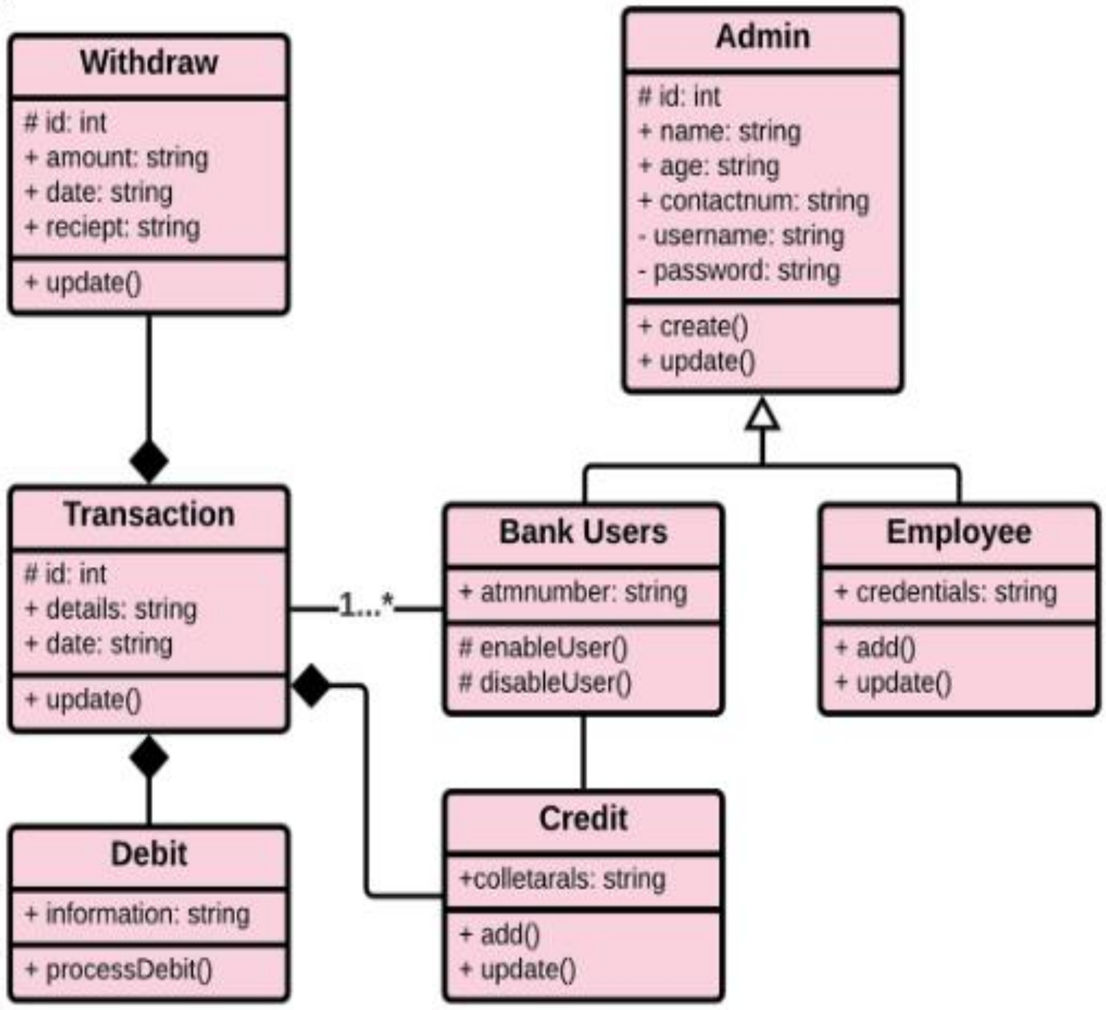
Prototyping: We must give a prototyping system of our banking management system.

Implementation: The implementation of our idea must be done.

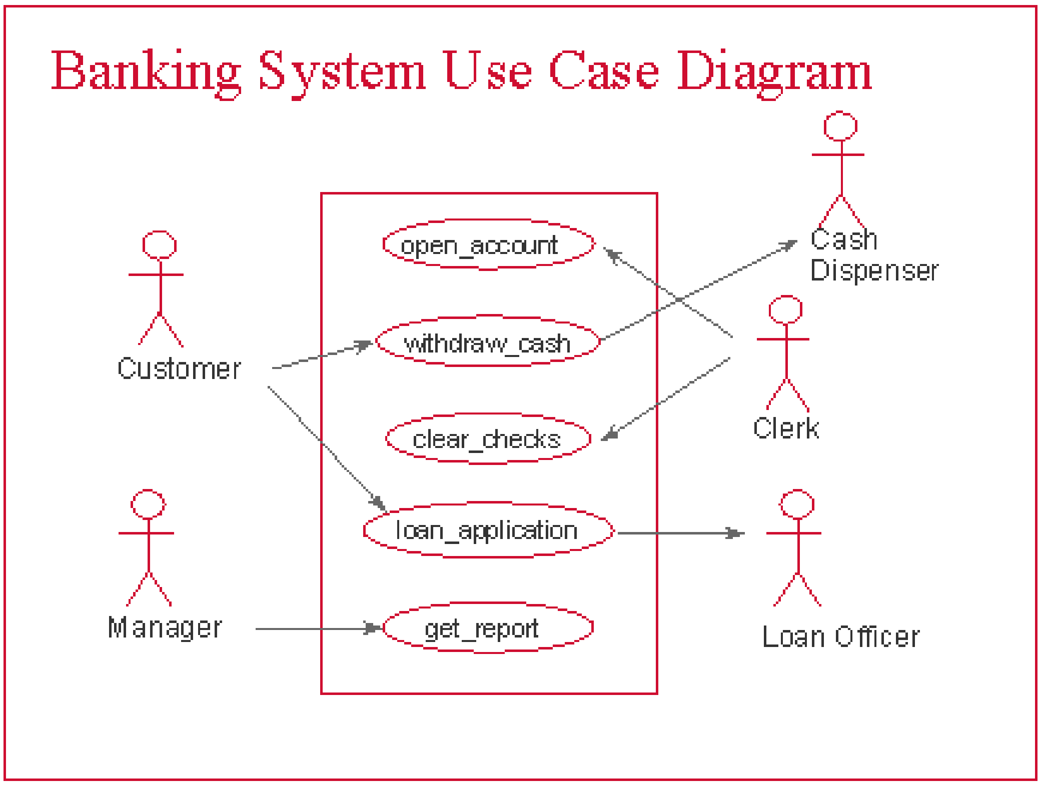
Data conversion and loading: Converting the existing application to run the new database.

Operational Maintenance: Implementing and monitoring the system.

<u>Class Diagram:</u>



Use Case Diagram:



SCOPE OF BANKING MANAGEMENT:

- It can be used by bank employees and customer depending on the bank policies. It can be used by several employees at the same time. It can be accessed using any general web browser with geographical interface

Result:

EXP: 2 **ER- and EER-to-Relational Mapping**
Date:

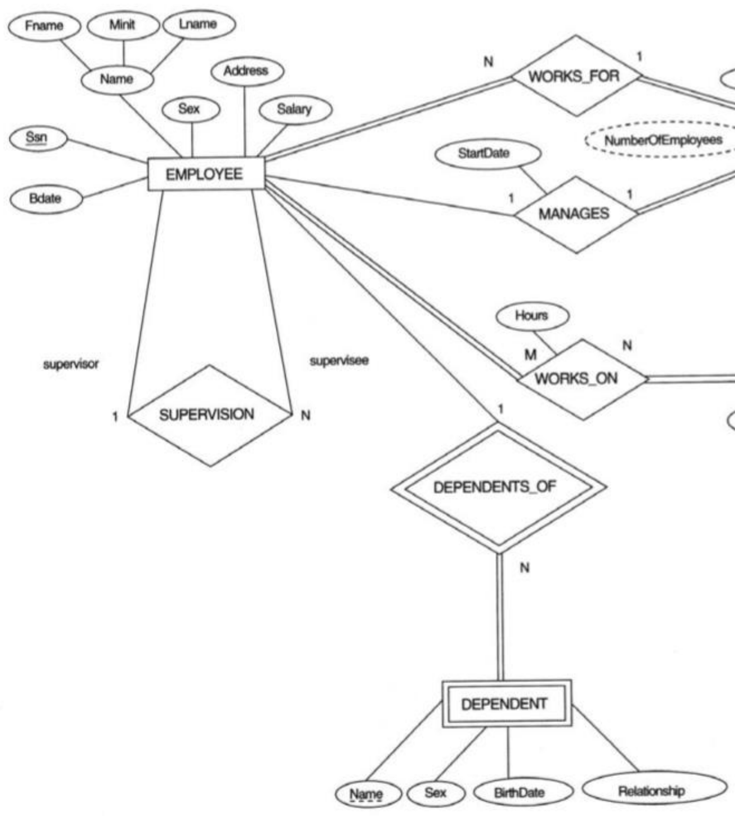
Aim:

Steps:

- ER-to-Relational mapping algorithm
  - step 1: mapping of regular entity types
  - step 2: napping of weak entity types
  - step 3: mapping of binary 1:1 relation types
  - step 4: mapping of binary 1:N relationship types
  - step 5: mapping of binary M:N relationship types
  - step 6: mapping of multi valued attributes
  - step 7: mapping of N-ary relationship types

- mapping EER model constructs to relations
  - step 8: options for mapping specialization or generalization
  - step 9: mapping of union types (categories)

ER conceptual schema:



Resulting relational database schema:

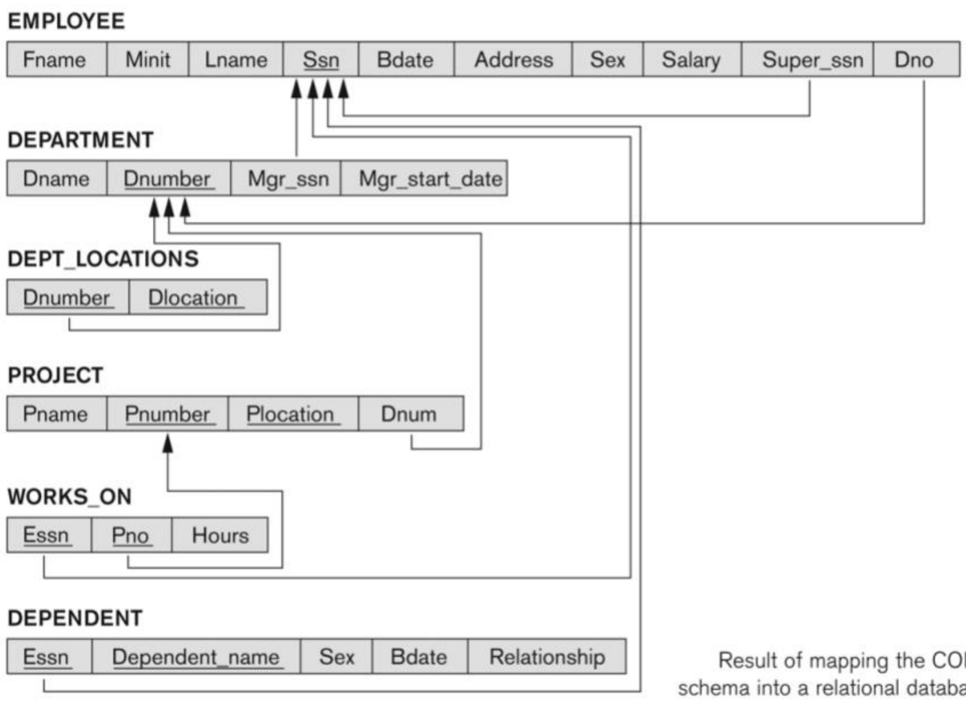
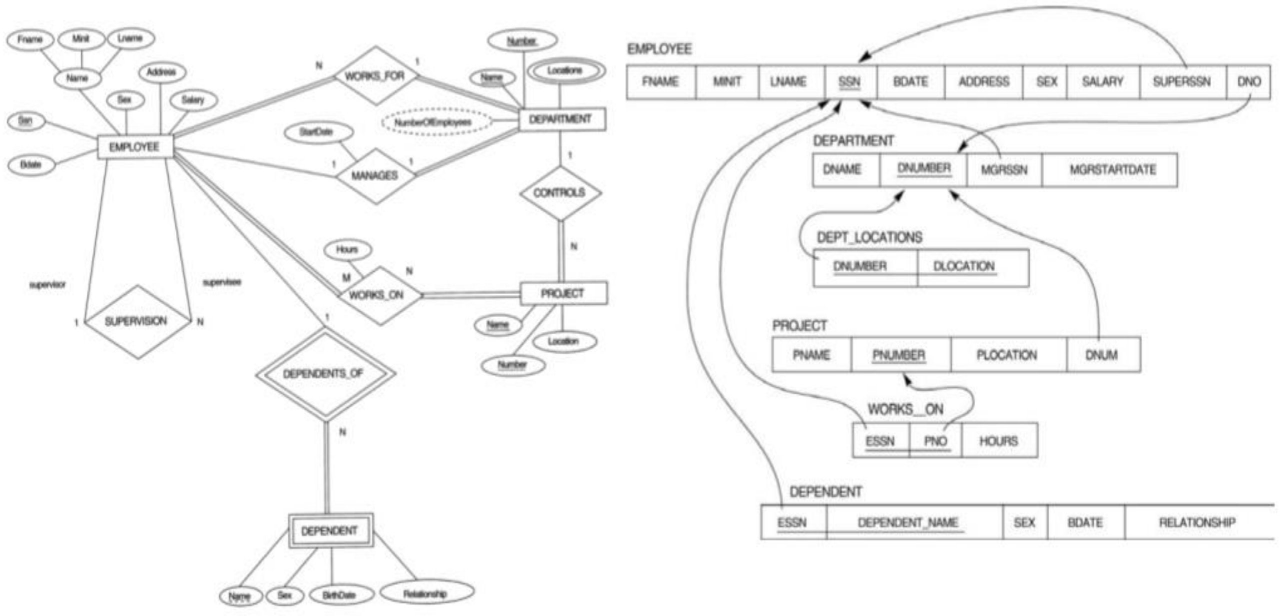


Figure 7.2
Result of mapping the COMPANY ER schema into a relational database schema.

Together of relational and conceptual schema:



Step 1:

- Mapping of regular entity types
- For each regular entity type E in the ER schema, create a relation R that includes all the simple attributes of E.
- Include only the simple component attributes of a composite attribute. Choose one of the key attributes of E as primary key for R.
- If the chosen key of E is composite, the set of simple attributes that form it will together form the primary key of R .
- e.g., EMPLOYEE, DEPARTMENT, PROJECT

Step 2:

- Mapping of weak entity types.
- For each weak entity type W in the ER schema with owner entity type E, create a relation R and include all simple attributes of W as attributes of R .

- Include as foreign key attributes of R the primary key attribute(s) of the relation(s) that correspond to the owner entity type(s). The primary key of R is the combination of the primary key(s) of the owner(s) and the partial key of the weak entity type W, if any.
- If there is a weak entity type E2 whose owner is also a weak entity type E1, then E1 should be mapped before E2 to determine its primary key first.
- e.g., DEPENDENT

Step 3:

- Mapping of binary 1:1 relationship types
- For each binary 1:1 relationship type R in the ER schema, identify the relations S and T that correspond to the entity types participating in R
- Foreign key approach
    - choose one of the relations, S, and include as a foreign key in S the primary key of T
    - include all the simple attributes of R as attributes of S
- Merged relation option
    - merge the two entity types and the relationship into a single relation
- Relationship relation option
    - set up a third relation R for the purpose of cross-referencing the primary keys of S and T
- MANAGES -> DEPARTMENT.MGRSSN,DEPARTMENT.MGRSTARTDATE

Step 4:

- Mapping of binary 1:N relationship types
- For each binary 1:N relationship type R, identify the relation S that represents the participating entity type at the N-side of the relationship type
- Include as foreign key in S the primary key of the relation T that represents the other entity type participating in R
- Include any simple attributes of the 1:N relationship type as attributes of S
- e.g., WORKS_FOR: S = EMPLOYEE, T = DEPARTMENT, DNO: the primary key of T

Step 5:

- Mapping of binary M:N relationship types
- For each binary M:N relationship type R, create a new relation S to represent R
- Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types

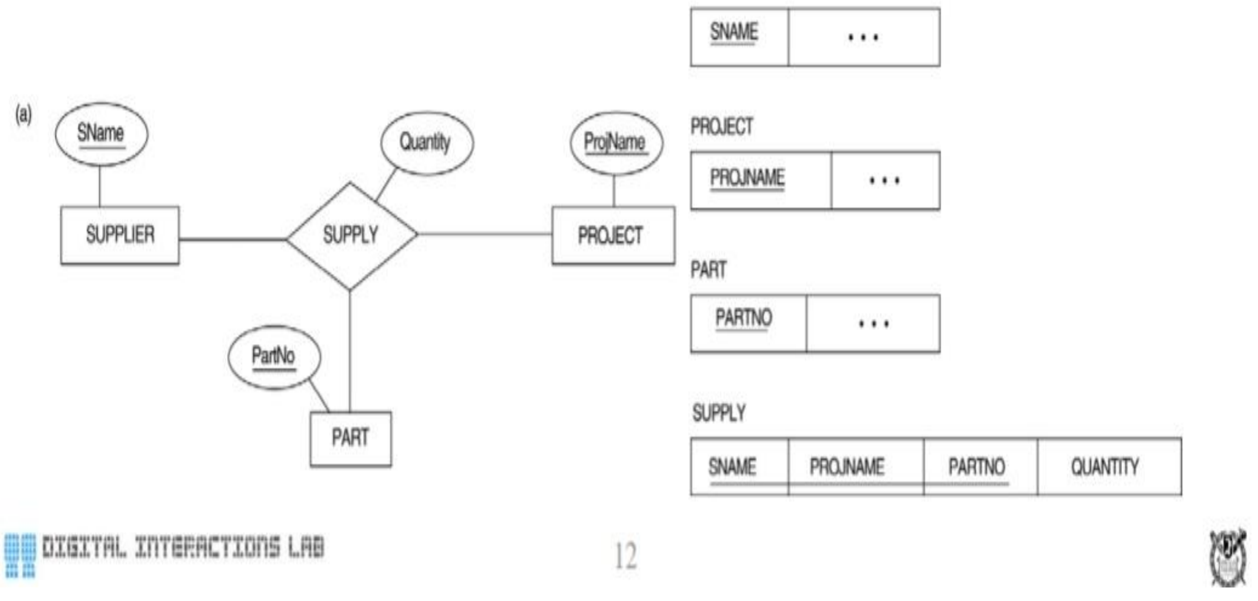
- Their combination will form the primary key of S
- Include any simple attributes of R as attributes of S
- e.g., WORKS_ON: S = WORKS_ON

Step 6:

- Mapping of multi valued attributes
- For each multi valued attribute A, create a new relation R
- R will include an attribute corresponding to A, plus the primary key attribute K - as a foreign key in R – of the relation that represents the entity type or relationship type that has A as an attribute
- The primary key of R is the combination of A and K
- If the multi valued attribute is composite, include its simple components
- e.g., Locations: A = DLOCATION, R = DEPT_LOCATIONS, K = DNUMBER.

Step 7:

- Mapping of N-ary relationship types
- For each n-ary relationship type R, where n > 2, create a new relation S to represent R
- Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types
- Include any simple attributes of R as attributes of S
- The primary key of S is usually a combination of all the foreign keys that reference the relations representing the participating entity types e.g., SUPPL

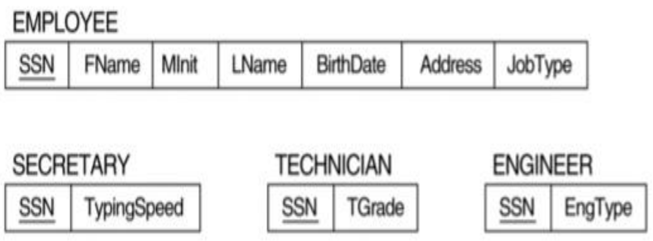


Step 8:

- options for mapping specialization or generalization
- convert each specialization with m subclasses {S1, S2, …, Sm} and superclass C, where the attributes of C are {k, a1, …, an} and k is the key, into relation schemas using one of the following options

Option 8A:

- Multiple relations-super class and subclasses
- Create a relation L for C with attributes Attrs(L) = {k, a1, …, an} and PK(L) = k
- Create a relation Li for each subclass Si, with the attributes Attrs(Li) = {k} U {attributes of Si} and PK(Li) = k
- Works for any specialization (total or partial, disjoint or overlapping)



EMPLOYEE

SSN	FName	MInit	LName	BirthDate	Address	JobType

SECRETARY

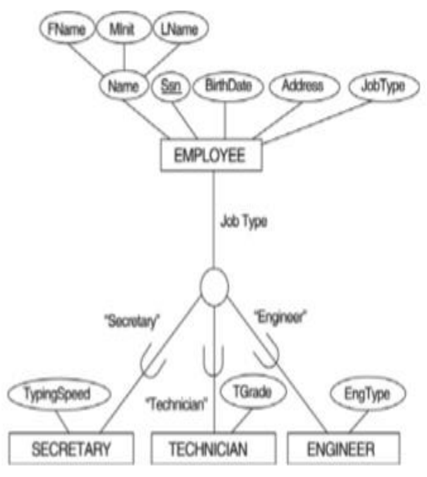
SSN	TypingSpeed

TECHNICIAN

SSN	TGrade

ENGINEER

SSN	EngType



Option 8B:

- Multiple relations-subclass relations only
- Create a relation Li for each subclass Si, with the attributes Attrs(Li) = {attributes of Si} U{k, a1, …, an} and PK(Li) = k
- Only works for a specialization whose subclasses are total

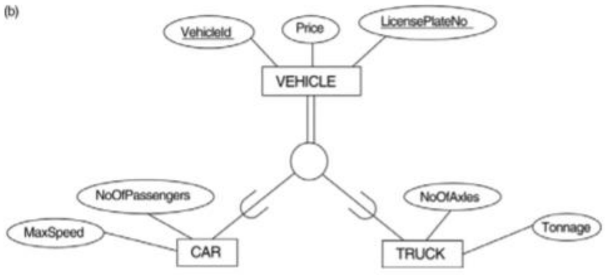
CAR

VehicleId	LicensePlateNo	Price	MaxSpeed	NoOfPassengers

TRUCK

VehicleId	LicensePlateNo	Price	NoOfAxles	

(b)

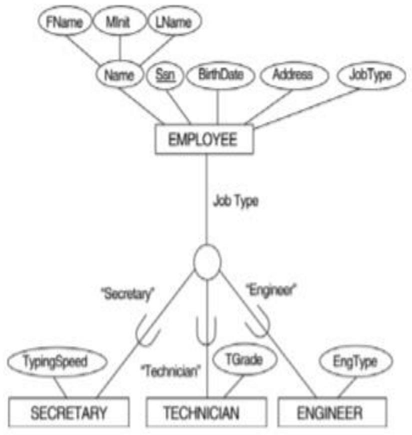


Option 8C:

- single relation with one type attribute
- create a single relation L with attributes Attrs(L) = {k, a1, …, an} U {attributes of S1} U… U{attributes of Sm} U{t} and PK(L) = k
- the attribute t is called a type attribute that indicates the subclass to which each tuple belongs
- works only for a specialization whose subclasses are disjoint

EMPLOYEE

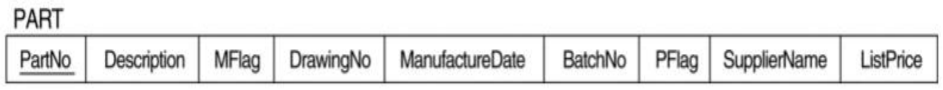
SSN	FName	MInit	LName	BirthDate	Address	JobType	TypingSpeed	TGrade	



Option 8D:

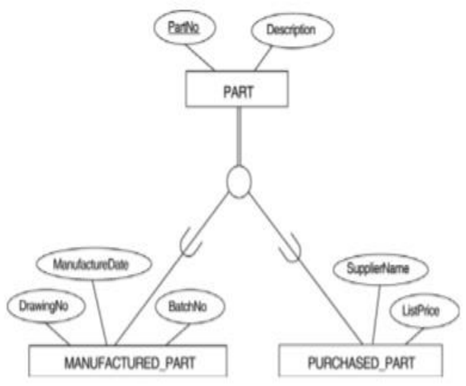
- single relation with multiple type attributes
- create a single relation schema L with attributes Attrs(L) = {k, a1, …, an} U {attributes of S1} U… U{attributes of Sm} U{t1, …, tm} and PK(L) = k

- each ti is a Boolean type attribute indicating whether a tuple belongs to subclass Si
- works for a specialization whose subclasses are overlapping



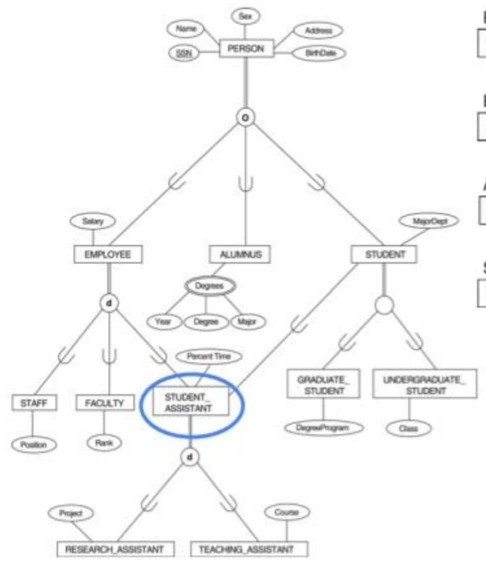
PART

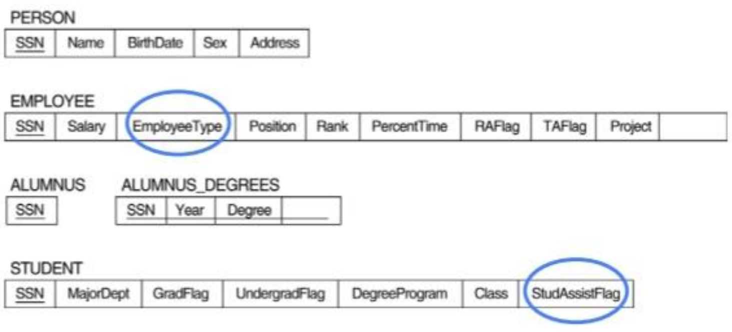
PartNo	Description	MFlag	DrawingNo	ManufactureDate	BatchNo	PFlag	SupplierName	ListPrice



Mapping of shared subclasses

- shared subclass: a subclass of several superclasses, indicating multiple inheritance
- apply any of the options in step 8 to a shared subclass





PERSON

SSN	Name	BirthDate	Sex	Address

EMPLOYEE

SSN	Salary	EmployeeType	Position	Rank	PercentTime	RAFlag	TAFlag	Project	

ALUMNUS

SSN

ALUMNUS_DEGREES

SSN	Year	Degree	

STUDENT

SSN	MajorDept	GradFlag	UndergradFlag	DegreeProgram	Class	StudAssistFlag

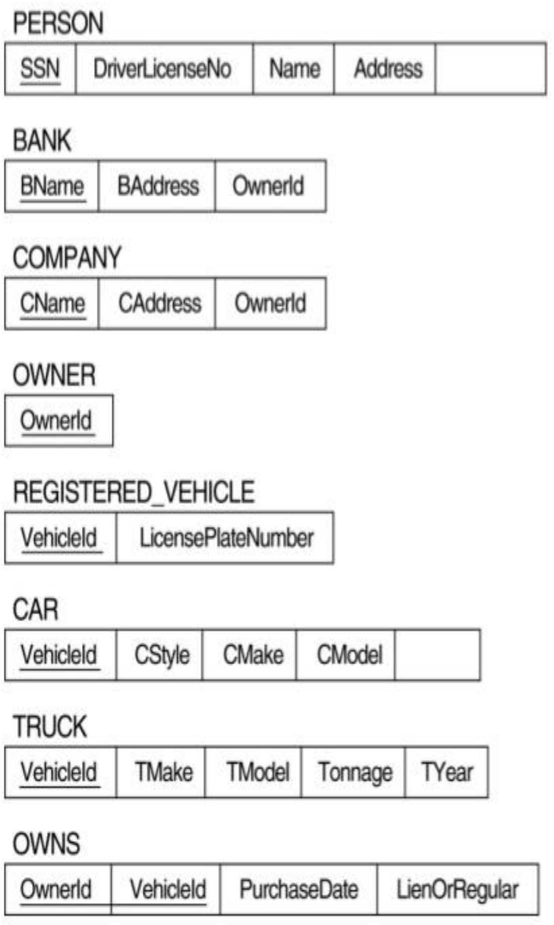
<u>Mapping of categories</u>

<u>Category:</u>

- A subclass of the union of two or more super classes that can have different keys because they can be of different entity types

<u>Step 9:</u>

- mapping of categories
- mapping a category whose defining super classes have different keys
  - ➢ specify a new key attribute, called a surrogate key
  - ➢ include the surrogate key attribute as foreign key in each relation corresponding to a super class of the category
  - ➢ e.g., OWNER category
- mapping a category whose super classes have the same key
  - ➢ no need for a surrogate key
  - ➢ e.g., REGISTERED_VEHICLE



PERSON

SSN	DriverLicenseNo	Name	Address	

BANK

BName	BAddress	OwnerId

COMPANY

CName	CAddress	OwnerId

OWNER

OwnerId

REGISTERED_VEHICLE

VehicleId	LicensePlateNumber

CAR

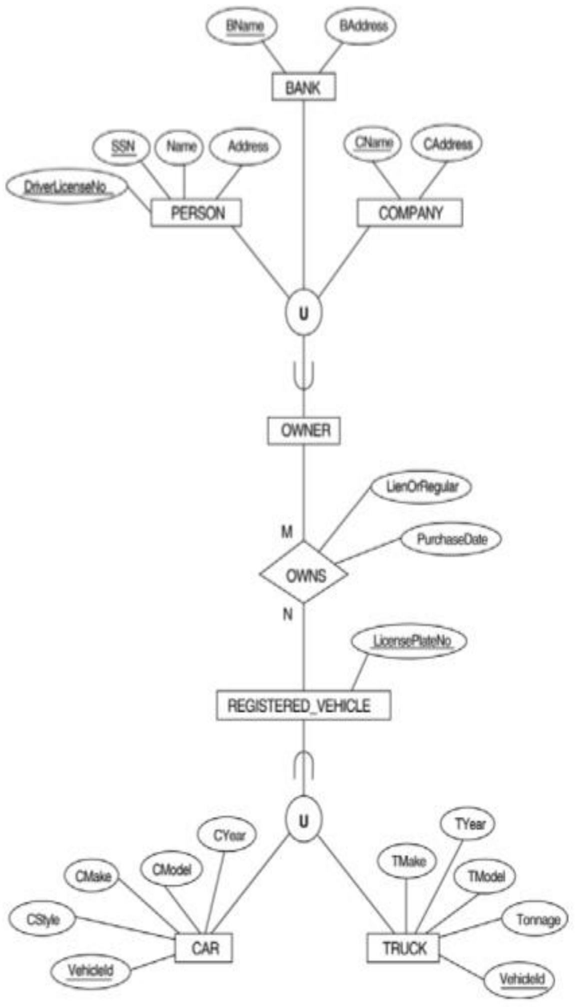
VehicleId	CStyle	CMake	CModel	

TRUCK

VehicleId	TMake	TModel	Tonnage	TYear

OWNS

OwnerId	VehicleId	PurchaseDate	LienOrRegular



Result:

Exp: 3 **PRACTICING DDL COMMANDS**

Date:

Aim :

Procedure :

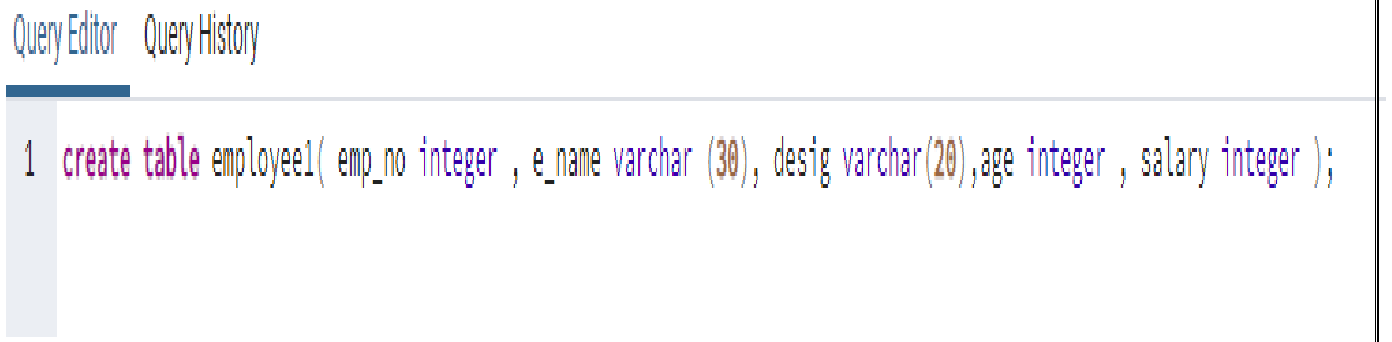
1) **Create a table called employee1 with the following structure.**

Name	**Type**
emp_no	integer
e_name	Varchar(30)
desig	Varchar(30)
age	integer
salary	integer

a. Add a column commission with domain to the Employee table.
b. Insert any five records into the table.
c. Update the column details of design.
d. Rename the column of employee1 table using alter command.
e. Truncate the table using truncate command
f. Drop the table using drop command.
g. Create a view in the name of emp_view and
   Display the contents of the table.

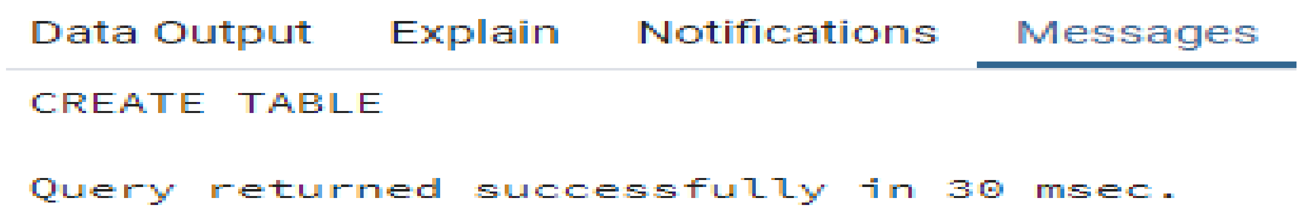
Data Definition Language (DDL) Commands :

Table Creation :



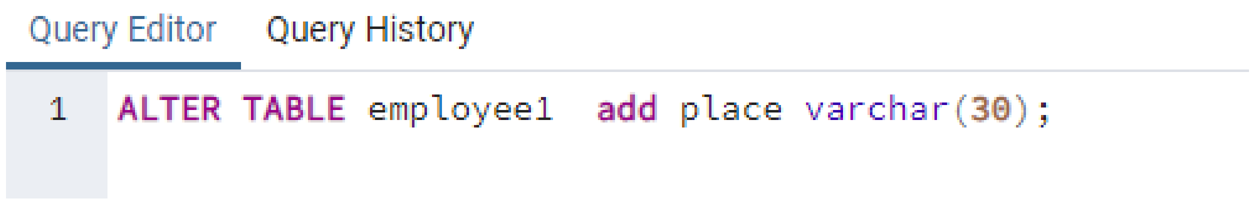
```
create table employee1( emp_no integer , e_name varchar (30), desig varchar(20),age integer , salary integer );
```

Output :



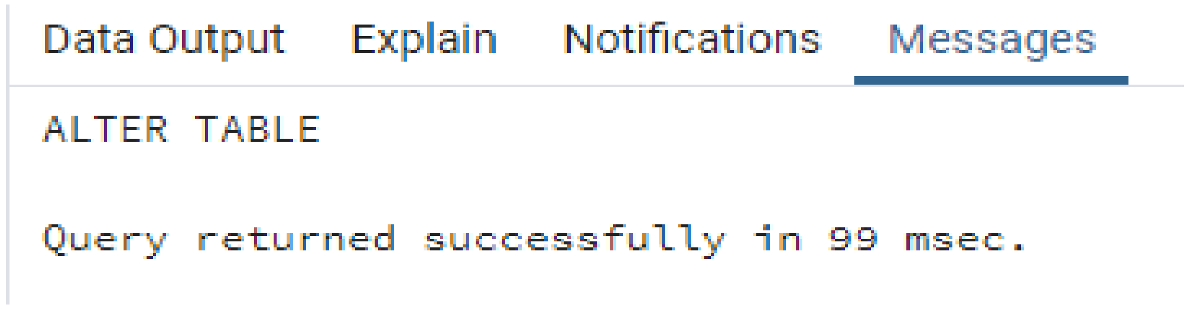
```
CREATE TABLE

Query returned successfully in 30 msec.
```

Alter Table :



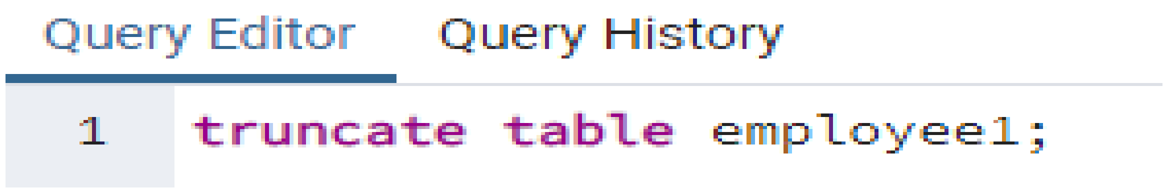
```
ALTER TABLE employee1  add place varchar(30);
```

Output :-



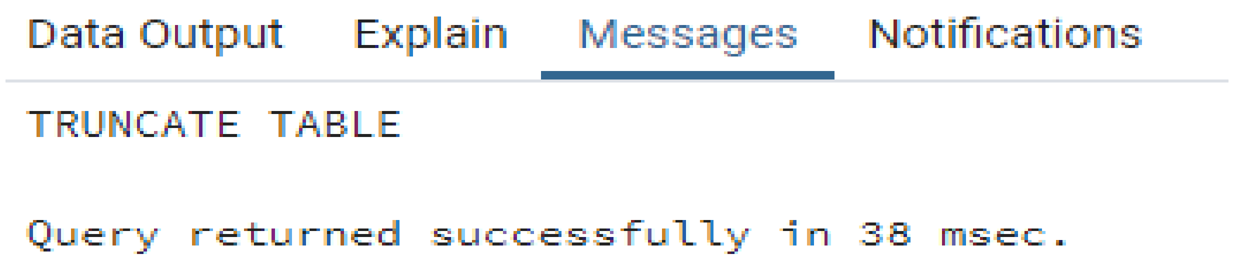
```
ALTER TABLE

Query returned successfully in 99 msec.
```

Truncate Table :



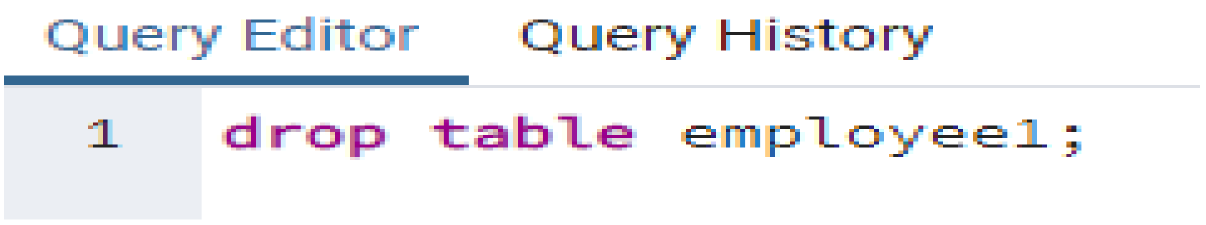
```
truncate table employee1;
```

Output :



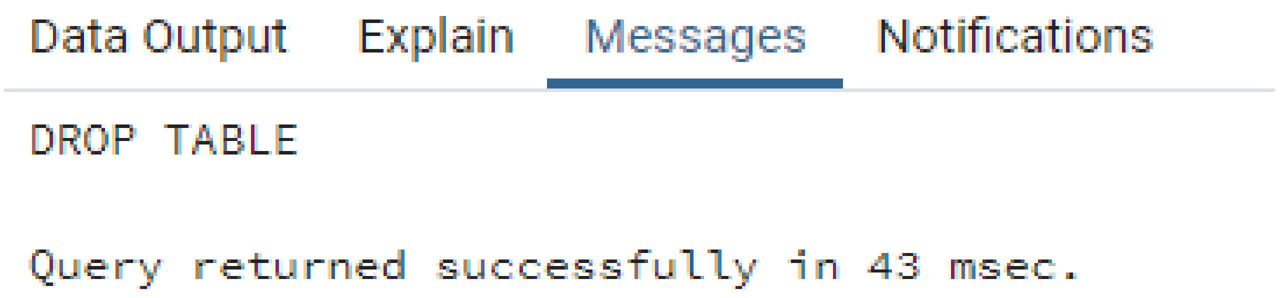
```
TRUNCATE TABLE

Query returned successfully in 38 msec.
```

Drop Table :



```
drop table employee1;
```

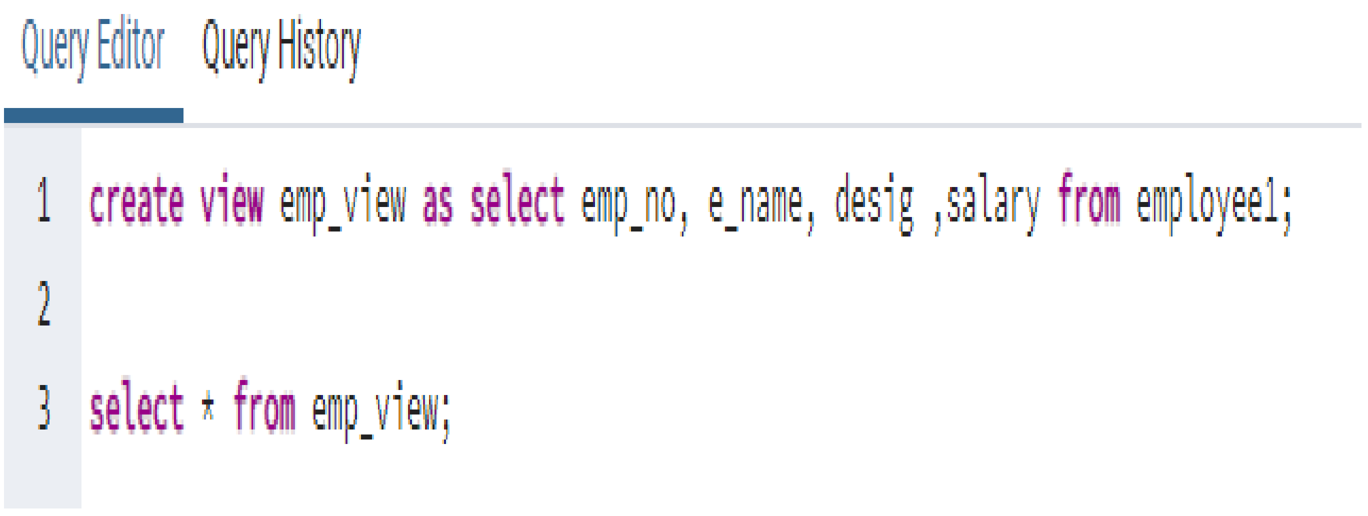
Output :



```
DROP TABLE

Query returned successfully in 43 msec.
```

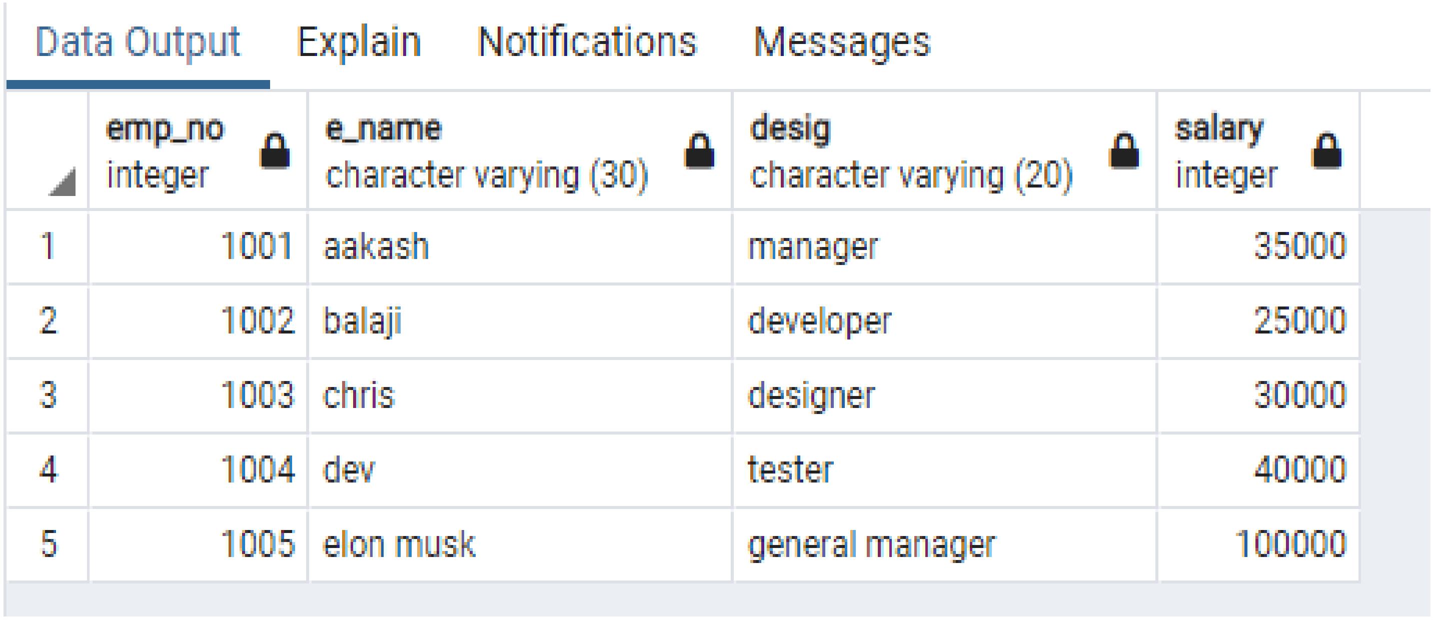
Views :

View Creation :



```
create view emp_view as select emp_no, e_name, desig ,salary from employee1;

select * from emp_view;
```

OUTPUT :-



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

Result :

Exp: 4 **PRACTICING DML COMMANDS**

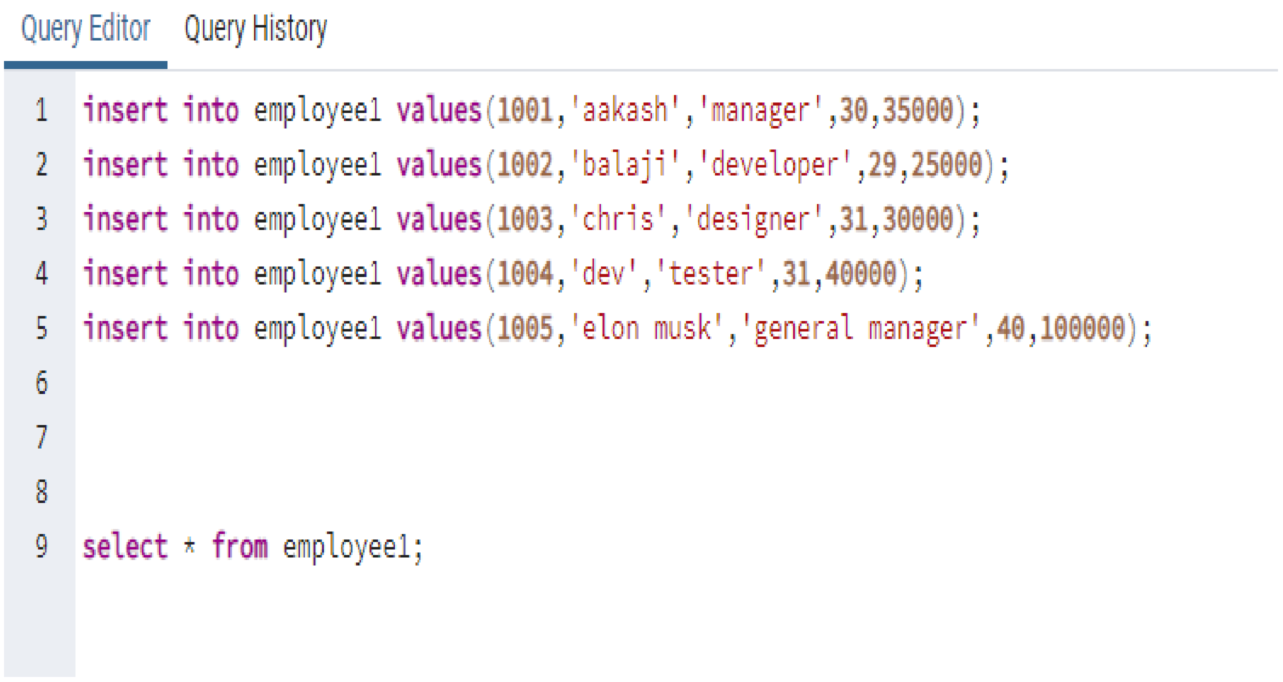
Date:

Aim :

Procedure :-

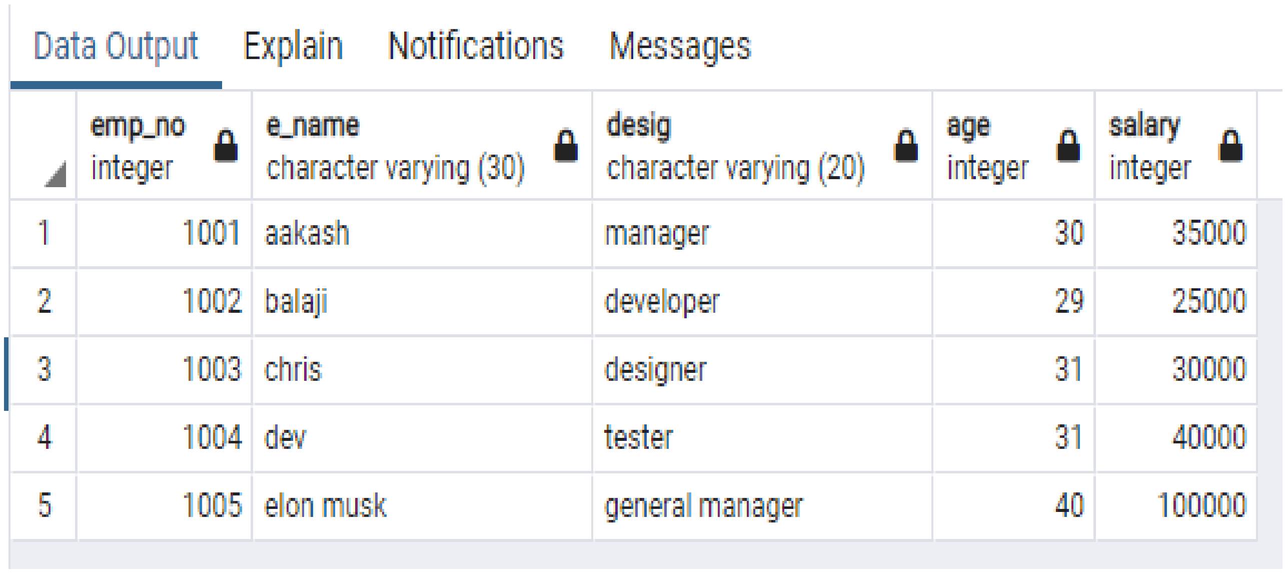
a. Insert any five records into the table.
b. Add a column to the table named place to the table using alter add command
c. Update the column details of place.
d. Delete a record from the table where the emp_no=1001.

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



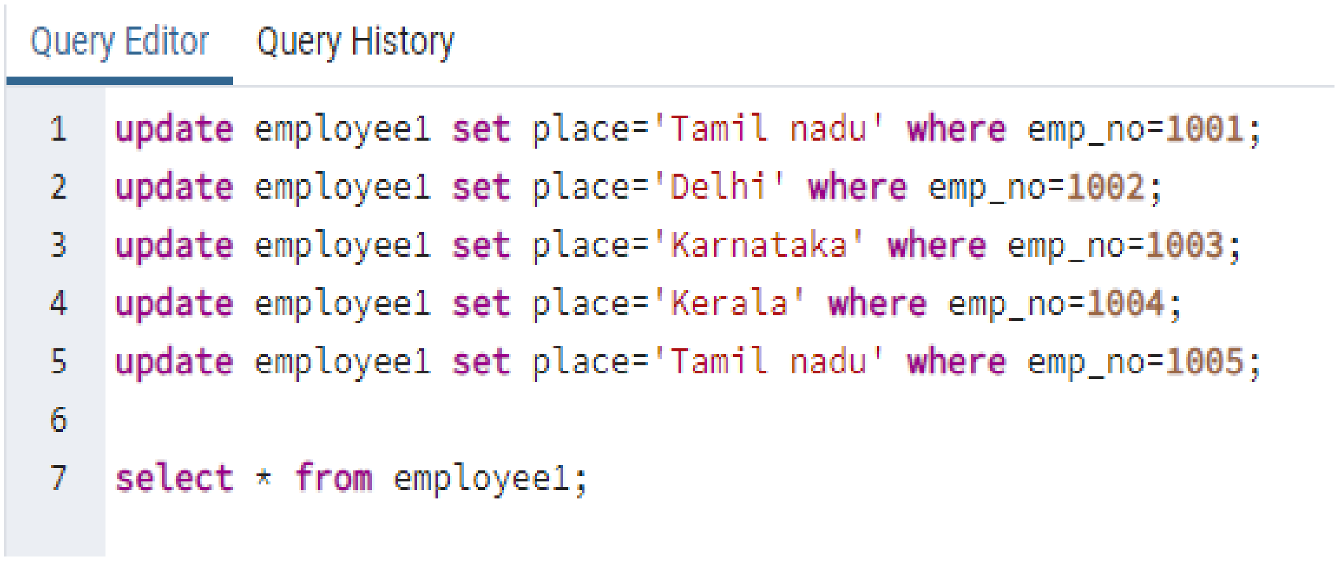
```
insert into employee1 values(1001,'aakash','manager',30,35000);
insert into employee1 values(1002,'balaji','developer',29,25000);
insert into employee1 values(1003,'chris','designer',31,30000);
insert into employee1 values(1004,'dev','tester',31,40000);
insert into employee1 values(1005,'elon musk','general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



select * from employee1;
```

Output :



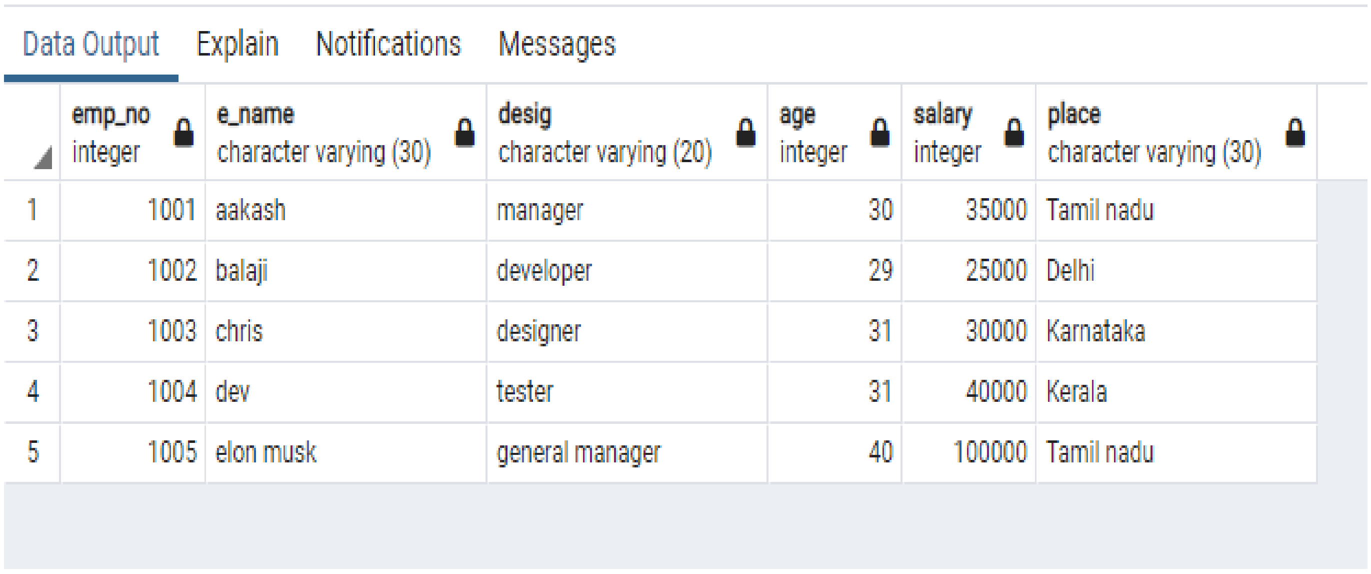
	emp_no integer	e_name character varying (30)	desig character varying (20)	age integer	salary integer
1	1001	aakash	manager	30	35000
2	1002	balaji	developer	29	25000
3	1003	chris	designer	31	30000
4	1004	dev	tester	31	40000
5	1005	elon musk	general manager	40	100000

Updating Values In The Table :



```
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;
```

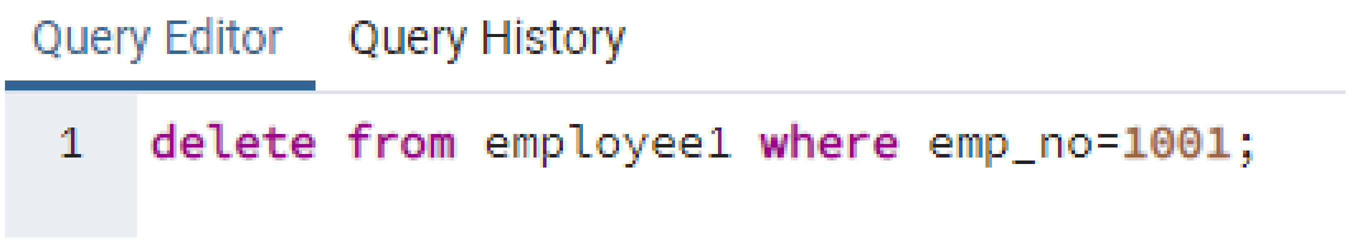
Output :



Data Output Explain Notifications Messages

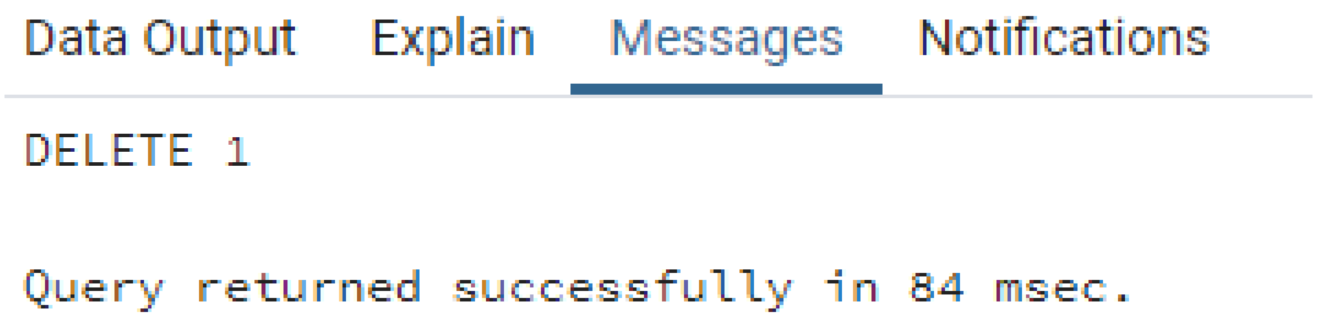
	emp_no integer	e_name character varying (30)	desig character varying (20)	age integer	salary integer	place character varying (30)
1	1001	aakash	manager	30	35000	Tamil nadu
2	1002	balaji	developer	29	25000	Delhi
3	1003	chris	designer	31	30000	Karnataka
4	1004	dev	tester	31	40000	Kerala
5	1005	elon musk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

Deleting A Record From The Trable :



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

OUTPUT :



```
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

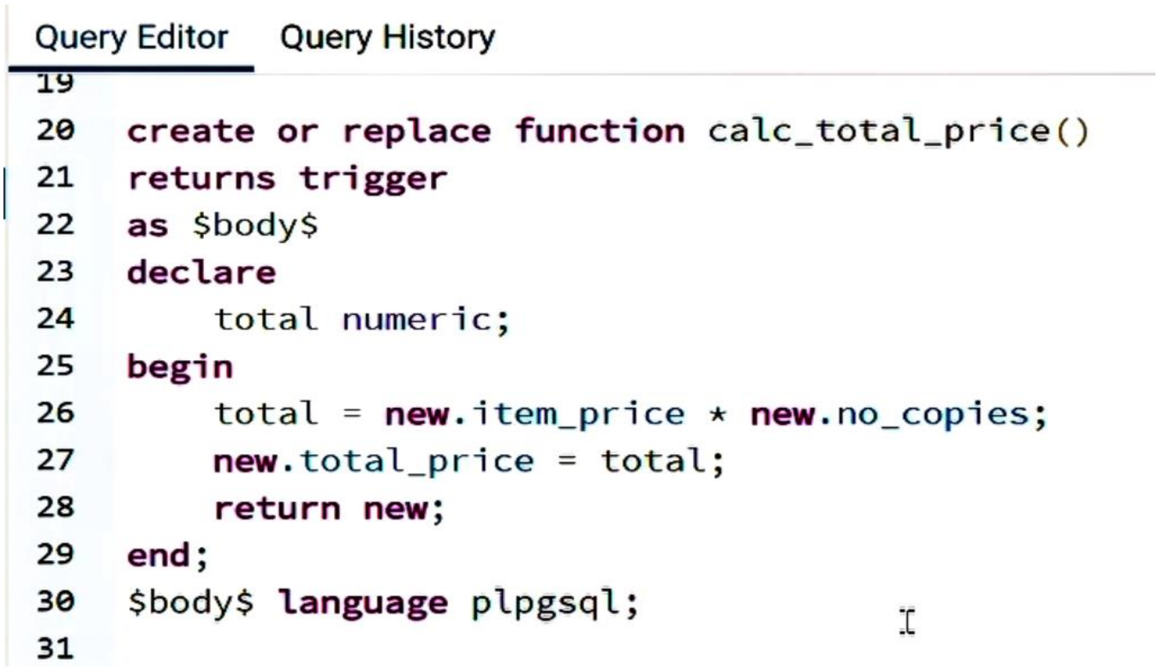
COLUMN NAME	**DATA TYPE**
isbn	integer
title	varchar(50)
item_price	numeric
no_copies	integer
total_price	numeric

- 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:

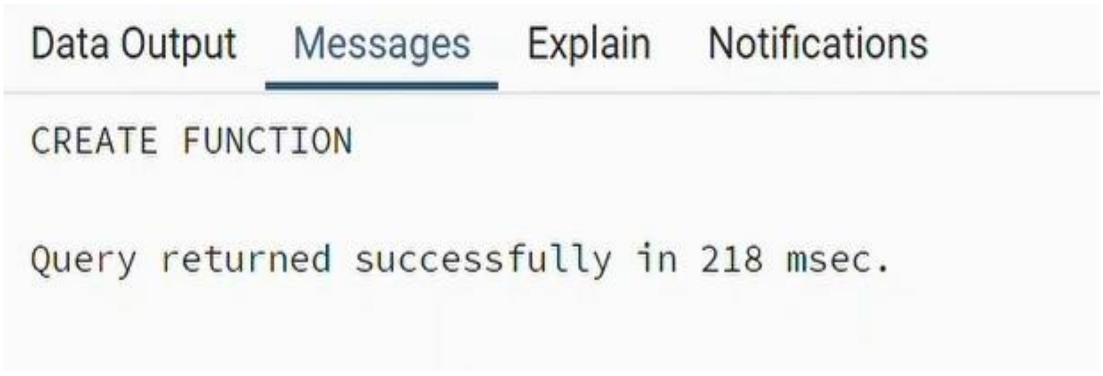
- OLD and NEW references are not available for table level triggers, rather you can use them for record level triggers.
- If you want to query the table in the same trigger, then you should use the AFTER keyword, because triggers can query the table or change it again only after the initial changes are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

Query :



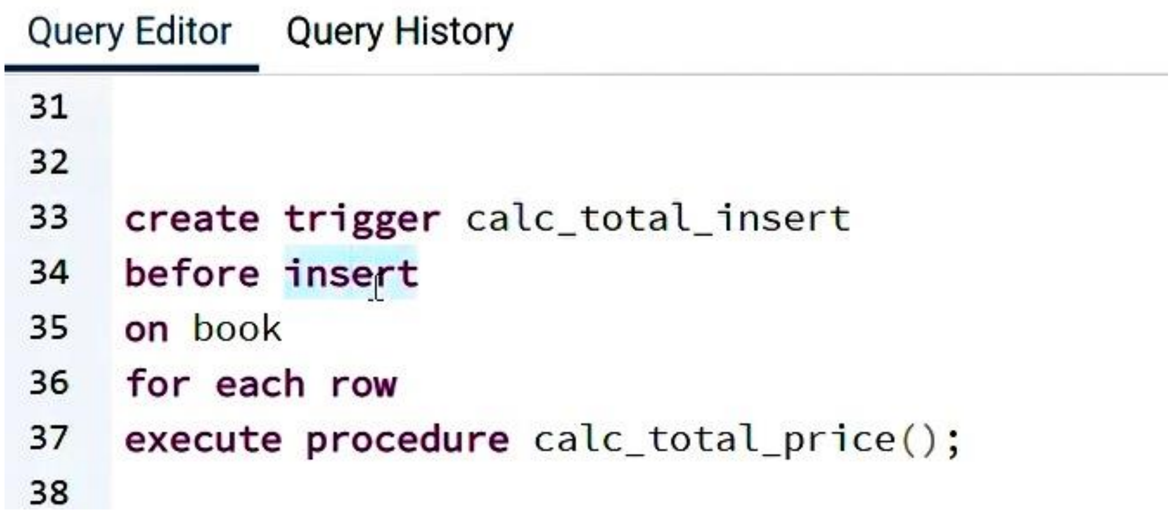
```
create or replace function calc_total_price()
returns trigger
as $body$
declare
    total numeric;
begin
    total = new.item_price * new.no_copies;
    new.total_price = total;
    return new;
end;
$body$ language plpgsql;
```

OUTPUT :-



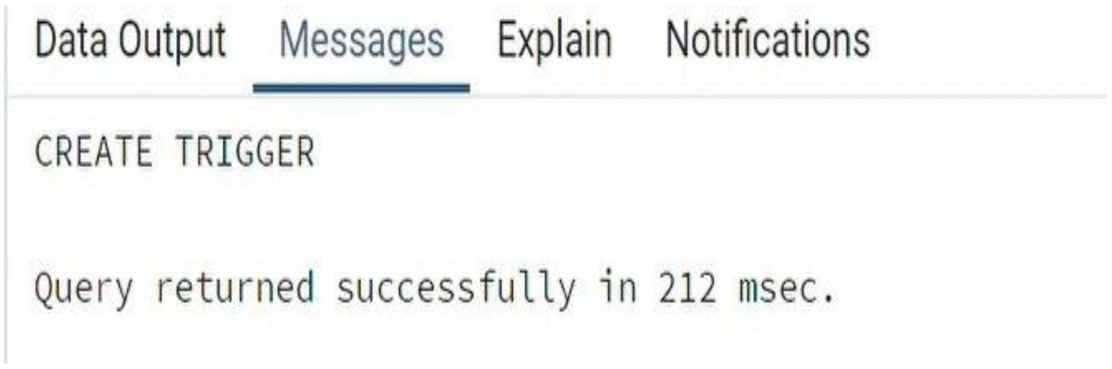
```
CREATE FUNCTION

Query returned successfully in 218 msec.
```

Query :



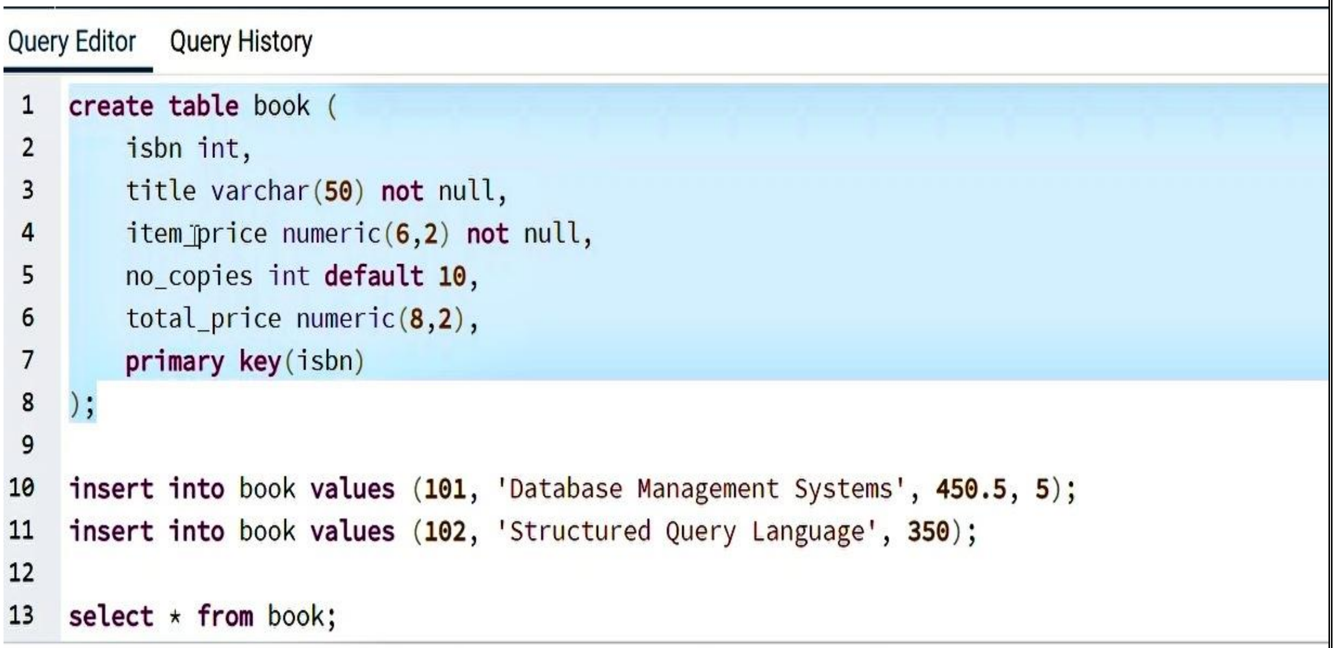
```
create trigger calc_total_insert
before insert
on book
for each row
execute procedure calc_total_price();
```

Output :



```
CREATE TRIGGER

Query returned successfully in 212 msec.
```

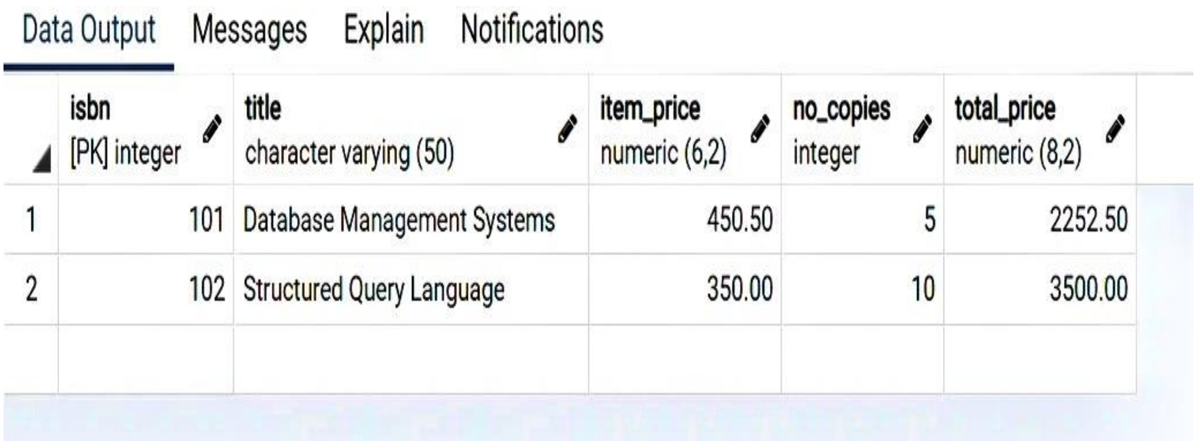
Query :



Query Editor Query History

```sql
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;
```

Output :



Data Output Messages Explain Notifications

	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

Stored Procedures / Functions :

Definition :

A stored procedure is a prepared SQL code that you can save, so the code can be 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.

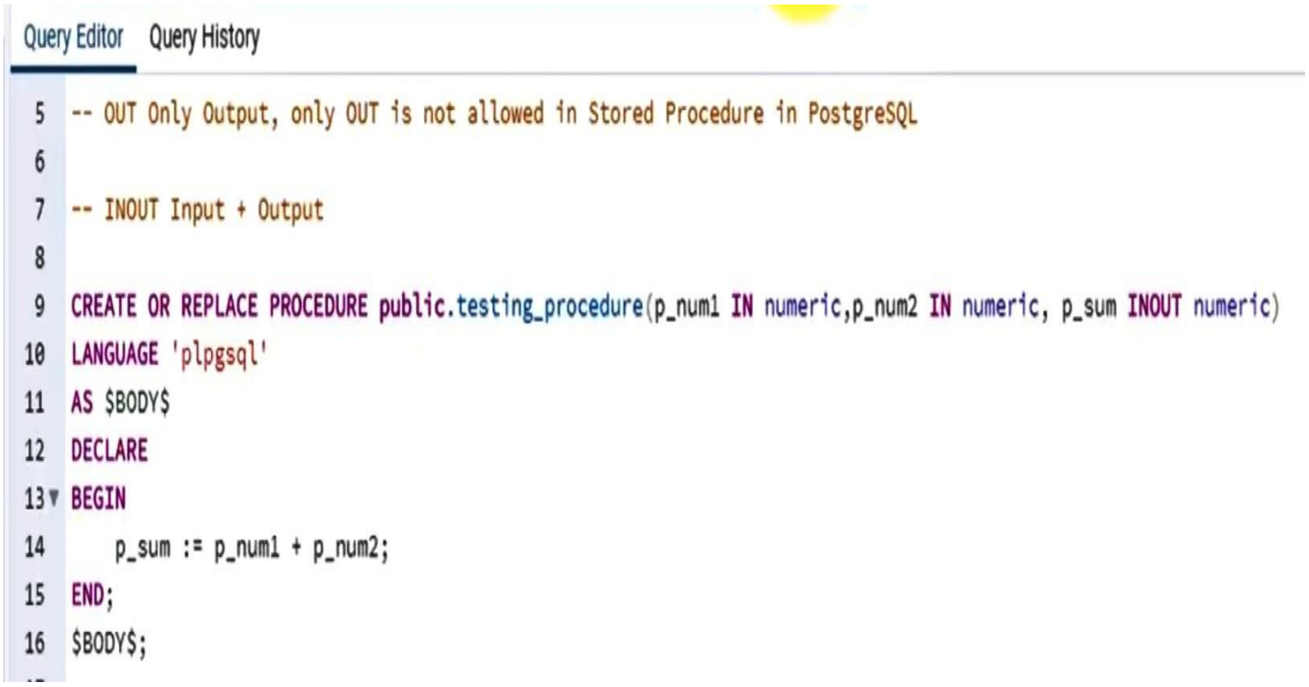
Procedure :

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

COLUMN NAME	DATA TYPE
p_num1	numeric
p_num2	numeric
p_sum	numeric

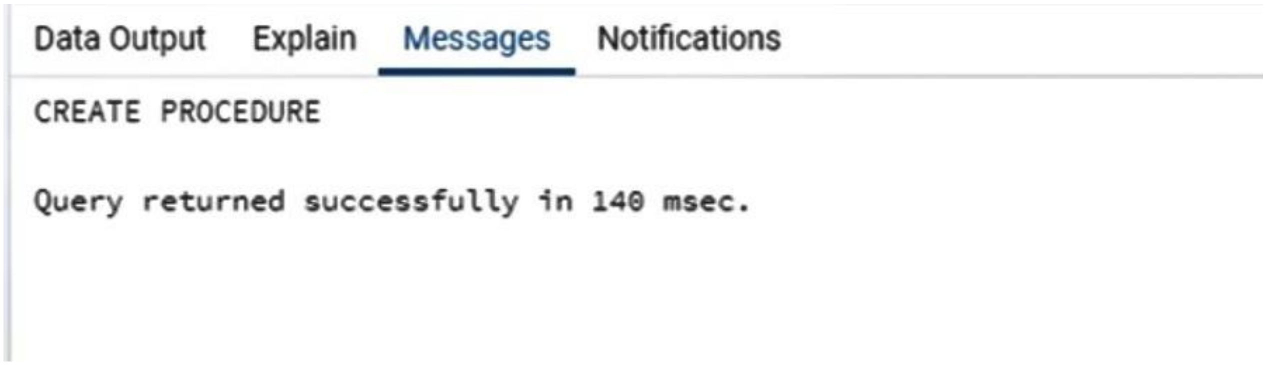
- Create a procedure **testing_procedure( )** to calculate the p_sum by adding the p_num1 and p_num2 using **create or replace procedures()** statement.
- Call the procedures with the values using the **call procedure_name()** query

QUERY :

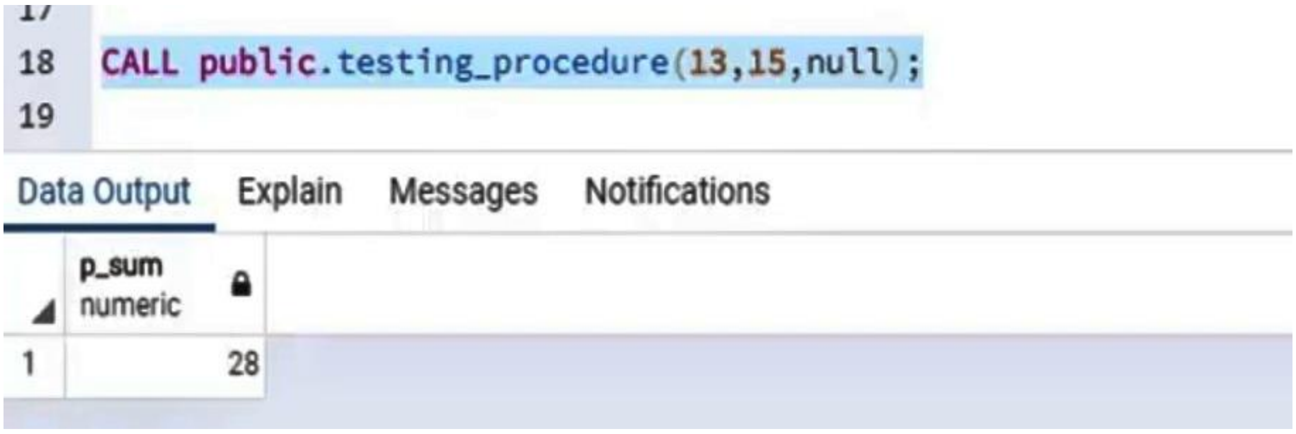


```
-- OUT Only Output, only OUT is not allowed in Stored Procedure in PostgreSQL

-- 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> :

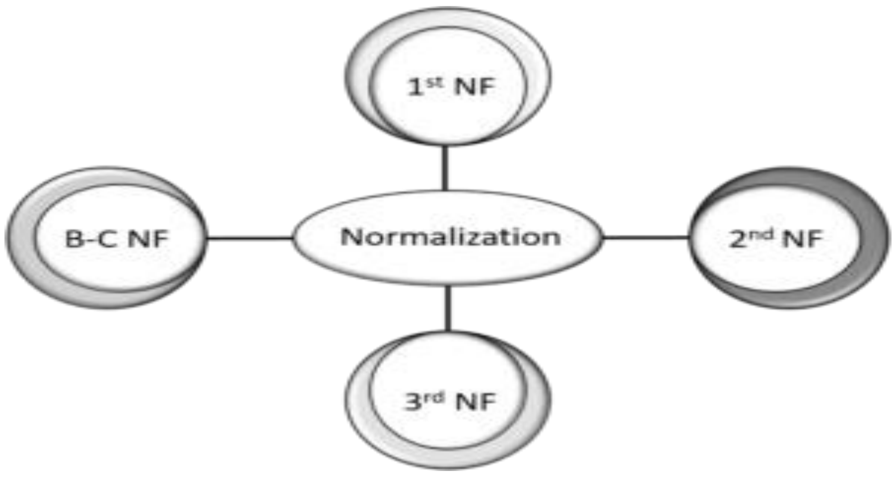
Exp: 6 **Database Design Using Normalization Bottom-Up Approach**

Date:

Aim:

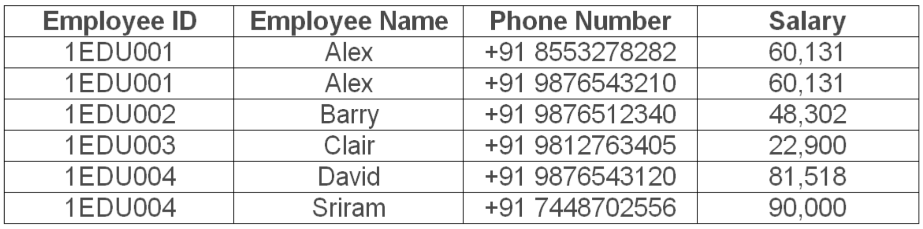
Normalization:

- It is the processes of reducing the redundancy of data in the table and also improving the data integrity. So why is this required? Without Normalization in 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 a data inconsistency that results from data redundancy and 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.



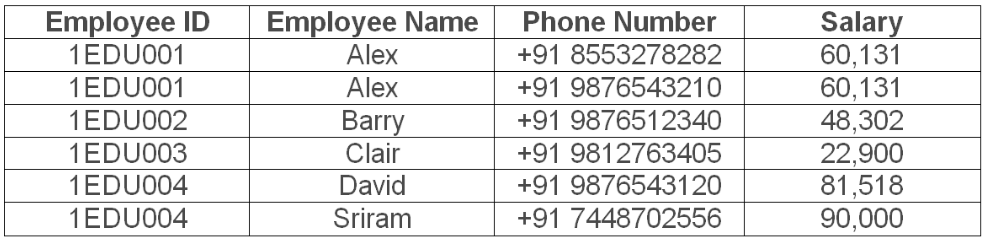
1st Normal Form (1NF)

- 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.

2nd Normal Form (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.

EMPLOYEE ID	DEPARTMENT ID	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 is Office Location. In this case, Office Location only depends on Department ID, which is only part of the primary key.
- Therefore, this table does not satisfy the second Normal Form. To bring this table to Second Normal Form, we need to break the table into two parts.

EMPLOYEE ID	DEPARTMENT ID
1EDU001	ED-T1
1EDU002	ED-S2
1EDU003	ED-M1
1EDU004	ED-T3

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

- In the table, the column Office Location is fully dependent on the primary key of that table, which is Department ID.

3rd Normal Form (3NF)

- The table has to be in 2NF before proceeding to 3NF. The other condition is there should be no transitive dependency for non-prime attributes.
- That means non-prime attributes (which doesn't form a candidate key) should not be dependent on other non-prime attributes in a given table.
- So a transitive dependency is a functional dependency in which X → Z (X determines Z) indirectly, by virtue of X → Y and Y → Z.

STUDENT ID	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, **Student ID** determines **Subject ID**, 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 dependent on **Student ID**. In the second table, **Subject** is only dependent on **Subject ID**.

Boyce Codd Normal Form (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.
- The table has to satisfy 3rd Normal Form.
- In BCNF if every functional dependency **A → B**, then **A** has to be the **Super Key** of that particular table.

STUDENT ID	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 enrol 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 **Student ID,** 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.

STUDENT ID	PROFESSOR ID
1DT15ENG01	1DTPF01
1DT15ENG02	1DTPF02
1DT15ENG02	1DTPF03

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

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

- By this we satisfiying the Boyce Codd Normal Form.

Bottom –up approach:

- Normalisation is a bottom-up approach which starts with a collection of attributes and organises them into well-structured relations which are free from redundant data.

BCNF: Boyce-Codd Normal Form

Result:

Exp:7: **Develop A Database ApplicationUsing IDE/RAD Tools**
Date:

Aim:

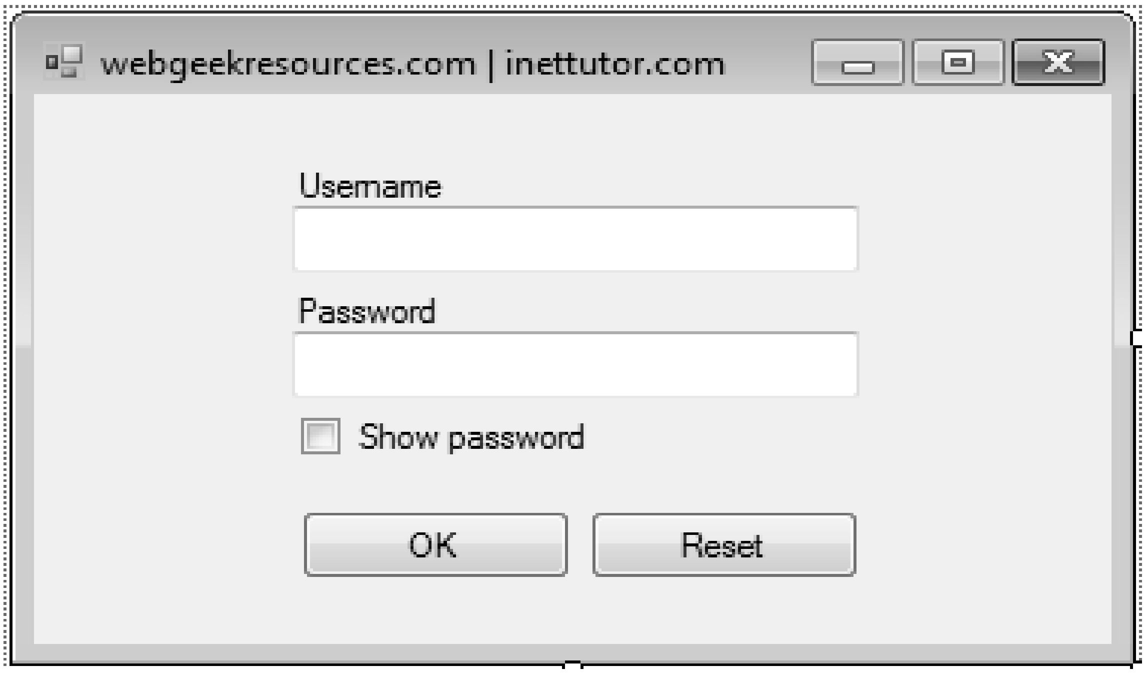
Procedure:

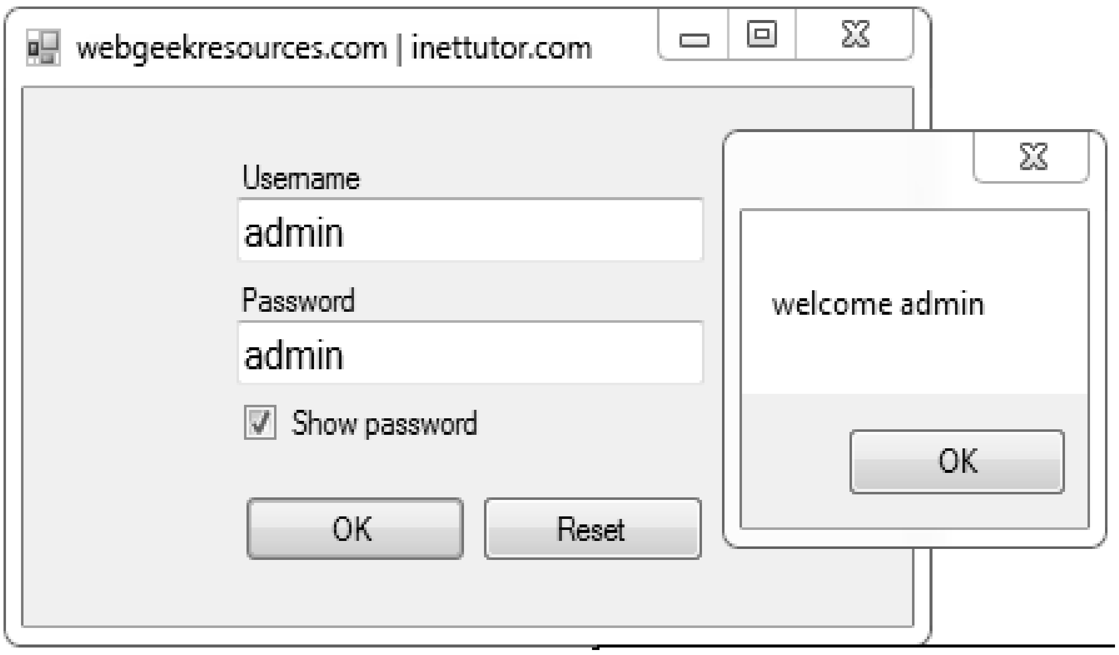
- Open you Microsoft Visual Studio 2010, 2012 or higher, or just your Microsoft Visual Basic .Net.
- Create a new project (select File and New Project).For visual studio user: (select Visual Basic then Windows FormApplication).
- Here is the sample form layout or design. Feel free to design your form.We need to add the following controls:
  - 2 labels
  - 2 textboxes
  - 2 buttons
  - 1 checkbox.
- 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 theuser 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 beadmin 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 makeits 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("Please enter username")
TextBox1.Focus()
Exit Sub
ElseIf TextBox2.Text = "" Then
MessageBox.Show("Please enter password")
TextBox2.Focus()
Exit Sub
End If
If TextBox1.Text = "admin" And TextBox2.Text = "admin" Then
MessageBox.Show("welcome admin")
Else
MessageBox.Show("incorrect username or password")
End If

If CheckBox1.Checked = True Then

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

Output:




webgeekresources.com | inettutor.com

Username
admin

Password
admin

☑ Show password

welcome admin

Result:

Exp:8 Database design using EER to- ODB mapping/ UML class diagrams

Date:

Aim:

Procedure:

Mapping an EER Schema to an ODB Schema

- It is relatively straightforward 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 ODL class 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 type and 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.

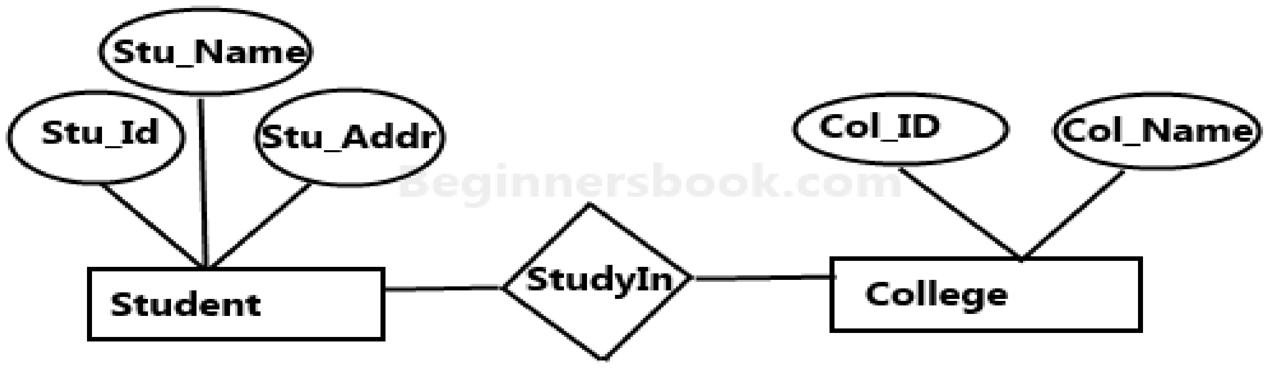
Step 6.

- 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 class to represent the category and defining 1:1 relationships between the category and each of its super classes.

Step 7.

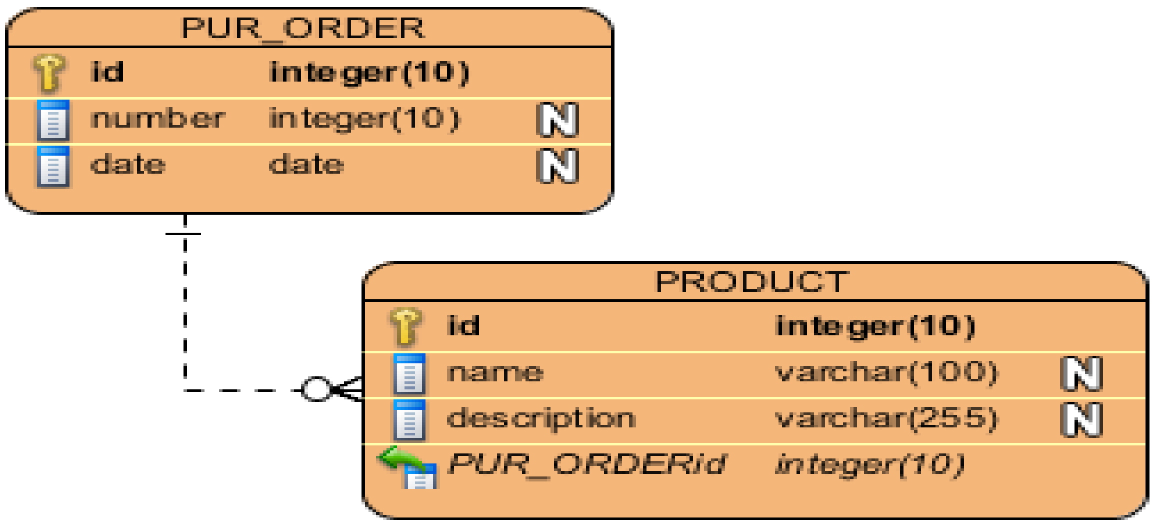
- An n-ary relationship with degree n > 2 can be mapped into a separate class, with appropriate references to each participating class.
- These references are based on mapping a 1:N relationship from each class that represents a participating entity type to the class that represents the n-ary relationship.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

EER to ODB mapping diagram:



Sample E-R Diagram

EER (UML class diagram):



Result:

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

Date:

Aim:

Objects of SQL:

- 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 data type and it uses the existing data types 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 sql server:

- 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.

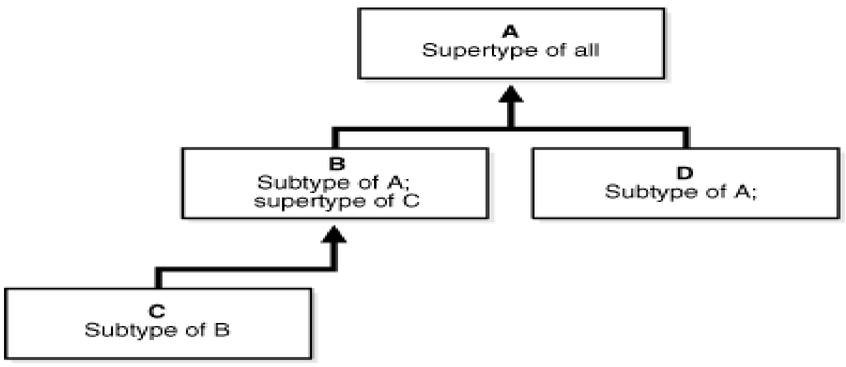
- CLR data types.
- Spatial data types

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.

Inheritance in SQL object 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 super type, 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 at most 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

Implementing Methods

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_type AS OBJECT
( numerator INTEGER,
  denominator INTEGER,
  MAP MEMBER FUNCTION rat_to_real RETURN REAL,
  MEMBER PROCEDURE normalize,
  MEMBER FUNCTION plus (x rational_type)
       RETURN rational_type);
```

Example: The following definition is shown merely because it defines the function gcd, which is used in the definition of the normalize method in the CREATE TYPE BODY statement later in this section.

```
CREATE FUNCTION gcd (x INTEGER, y INTEGER) RETURN INTEGER AS
-- 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.)
--
   ans INTEGER;
BEGIN
   IF (y <= x) AND (x MOD y = 0) THEN
      ans := y;
   ELSIF x < y THEN
      ans := gcd(y, x);  -- Recursive call
   ELSE
      ans := gcd(y, x MOD y);  -- Recursive call
   END IF;
   RETURN ans;
END;
```

Result:

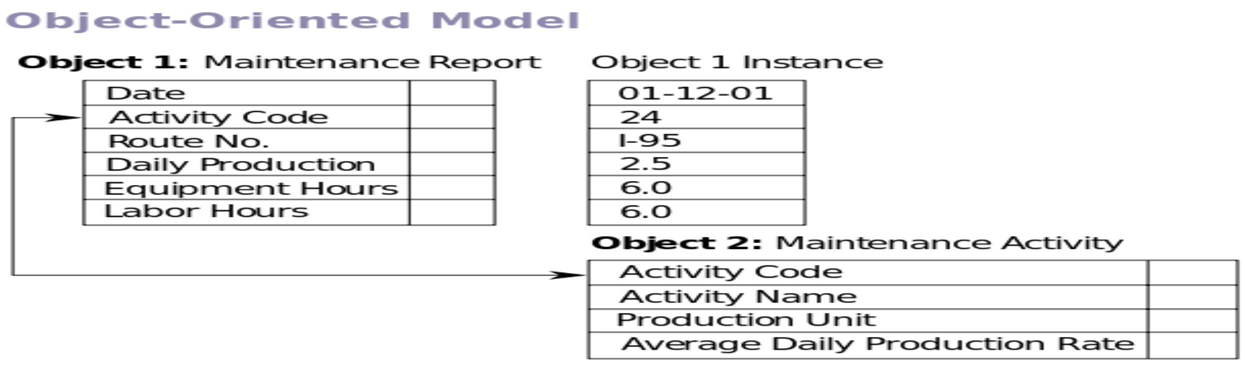
Exp: 10 Querying the Object-relational database using Object Query Language

Date:

Aim:

Object–relational database

- 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 object store for software written in an object-oriented programming language, with a

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

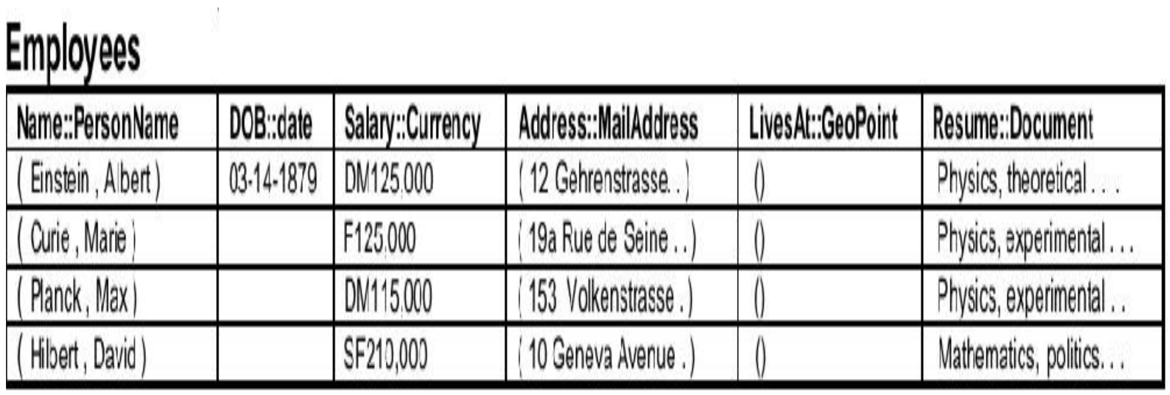
Procedure:

- CREATE.
- INSERT.
- UPDATE.
- DELETE

Program:

```
CREATE TABLE Employees (FirstName VARCHAR(32)
NOT NULL,
Surname VARCHAR(64) NOT NULL,DOB DATE NOT
NULL,
Salary DECIMAL(10,2) NOT NULLCHECK ( Salary > 0.0 ),
Address_1 VARCHAR(64) NOT NULL,Address_2 VAR-
CHAR(64) NOT NULL,
City VARCHAR(48) NOT NULL,State CHAR(2) NOT
NULL,ZipCode INTEGER NOT NULL,PRIMARY KEY (
Surname, FirstName, DOB ));

INSERT INTO Employees ( Pager_Number, Pass_Code, Mes-
sage )
SELECT E.Pager_Number,E.Pass_Code,
Print(E.Name) || ': Call 1-800-TEMPS-R-US for immediate
INFORMIX DBA job'
FROM Temporary_Employees E
WHERE Contains (GeoCircle('(-122.514, 37.221)', '60
miles')),E.LivesAt )
AND DocContains ( E.Resume, 'INFORMIX and Database
Administrator')
AND NOT IsBooked ( Period(TODAY, TODAY +
7),E.Booked );

SELECT *FROM Employees;
```

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: