SE-Computer-Div-A	Roll number : 9913
Experiment no. : 1	Date of Implementation :30/1/2023
Related Course outcome : At the end of the course, Students will be able to design ER model and develop relational model	

Rubrics for assessment of Experiment:

Indicator	Poor	Average	Good
Timeliness ● Maintains assignment deadline (3)	Assignment not done (0)	One or More than One week late (1-2)	Maintains deadline (3)
Completeness and neatness ● Complete all parts of ER diagram(3)	N/A	< 80% complete (1-2)	100% complete (3)
Originality ● Extent of plagiarism(2)	Copied it from someone else(0)	At least few questions have been done without copying(1)	Assignment has been solved completely without copying (2)
Knowledge ● In depth knowledge of the assignment(2)	Unable to answer 2 questions(0)	Unable to answer 1 question (1)	Able to answer 2 questions (2)

Assessment Marks :

Timeliness	
Completeness and neatness	
Originality	
Knowledge	
Total	

Total : (Out of 10)

Teacher's Sign :

Name Student	Mark Lopes	Roll No.	9913
Lab Experiment No.	1	Date	30/1/2023
Expt. Title	Write Problem Definition and Draw ER /EER diagram		

Aim : Problem Definition and draw ER /EER diagram

Objective of the Experiment:

1. To design/draw ER/EER for the selected problem

Theory:

The **entity-relationship (E-R)** data model is based on a perception of a real world that consists of a set of basic objects called **entities**, and of **relationships** among these objects.

The model is intended primarily for the database-design process. It was developed to facilitate database design by allowing the specification of an **enterprise schema**. Such a schema represents the overall logical structure of the database. This overall structure can be expressed graphically by an **E-R diagram**.

An **entity** is an object that exists in the real world and is distinguishable from other objects. We express the distinction by associating with each entity a set of attributes that describes the object.

A **relationship** is an association among several entities. The collection of all entities of the same type is an **entity set**, and the collection of all relationships of the same type is a **relationship set**.

Mapping cardinalities express the number of entities to which another entity can be associated via a relationship set.

A **superkey** of an entity set is a set of one or more attributes that, taken collectively, allows us to identify uniquely an entity in the entity set. We choose a minimal superkey for each entity set from among its superkeys; the minimal superkey is termed the entity set's **primary key**. Similarly, a relationship set is a set of one or more attributes that, taken collectively, allows us to identify uniquely a relationship in the relationship set. Likewise, we choose a minimal superkey for each relationship set from among its superkeys; this is the relationship set's primary key.

- An entity set that does not have sufficient attributes to form a primary key is termed a **weak entity set**. An entity set that has a primary key is termed a **strong entity set**.
- **Specialization** and **generalization** define a containment relationship between a higher-level entity set and one or more lower-level entity sets. Specialization is the result of taking a subset of a higher-level entity set to form a lower-level entity set. Generalization is the result of taking the union of two or more disjoint (lower-level) entity sets to produce a higher-level entity set. The attributes of higher-level entity sets are inherited by lower-level entity sets.
- **Aggregation** is an abstraction in which relationship sets (along with their associated entity sets) are treated as higher-level entity sets, and can participate in relationships.
- The various features of the E-R model offer the database designer numerous choices in how to best represent the enterprise being modeled. Concepts and objects may, in certain cases, be represented by entities, relationships, or attributes. Aspects of the overall structure of the enterprise may be best described by using weak entity sets, generalization, specialization, or aggregation. Often, the designer must weigh the merits of a simple, compact model versus those of a more precise, but more complex, one.
- A database that conforms to an E-R diagram can be represented by a collection of tables. For each entity set and for each relationship set in the database, there is a unique table that is assigned the name of the corresponding entity set or relationship set. Each table has a number of columns, each of which has a unique name. Converting database representation from an E-R diagram to a table format is the basis for deriving a relational-database design from an E-R diagram.
- The **unified modeling language (UML)** provides a graphical means of modeling various components of a software system. The class diagram component of UML is based on E-R diagrams. However, there are some differences between the two that one must beware of.

(Sample Problem statement-BANKING)

The initial specification of user requirements may be based on interviews with the database users, and on the designer's own analysis of the enterprise. The description that arises from this design phase serves as the basis for specifying the conceptual structure of the database. Here are the major characteristics of the banking enterprise.

- The bank is organized into branches. Each branch is located in a particular city and is identified by a unique name. The bank monitors the assets of each branch.

- Bank customers are identified by their *customer-id* values. The bank stores each customer's name, and the street and city where the customer lives. Customers may have accounts and can take out loans. A customer may be associated with a particular banker, who may act as a loan officer or personal banker for that customer.
- Bank employees are identified by their *employee-id* values. The bank administration stores the name and telephone number of each employee, the names of the employee's dependents, and the *employee-id* number of the employee's manager. The bank also keeps track of the employee's start date and, thus, length of employment.
- The bank offers two types of accounts—savings and checking accounts. Accounts can be held by more than one customer, and a customer can have more than one account. Each account is assigned a unique account number. The bank maintains a record of each account's balance, and the most recent date on which the account was accessed by each customer holding the account. In addition, each savings account has an interest rate, and overdrafts are recorded for each checking account.
- A loan originates at a particular branch and can be held by one or more customers. A loan is identified by a unique loan number. For each loan, the bank keeps track of the loan amount and the loan payments. Although a loan-payment number does not uniquely identify a particular payment among those for all the bank's loans, a payment number does identify a particular payment for a specific loan. The date and amount are recorded for each payment.

In a real banking enterprise, the bank would keep track of deposits and withdrawals from savings and checking accounts, just as it keeps track of payments to loan accounts. Since the modeling requirements for that tracking are similar, and we would like to keep our example application small, we do not keep track of such deposits and withdrawals in our model.

Step 1- Identify entities of problem stmt

we begin to identify entity sets and their attributes:

- The *branch* entity set, with attributes *branch-name*, *branch-city*, and *assets*.
- The *customer* entity set, with attributes *customer-id*, *customer-name*, *customer-street*; and *customer-city*. A possible additional attribute is *banker-name*.
- The *employee* entity set, with attributes *employee-id*, *employee-name*, *telephone-number*, *salary*, and *manager*. Additional descriptive features are the multivalued attribute *dependent-name*, the base attribute *start-date*, and the derived attribute *employment-length*.

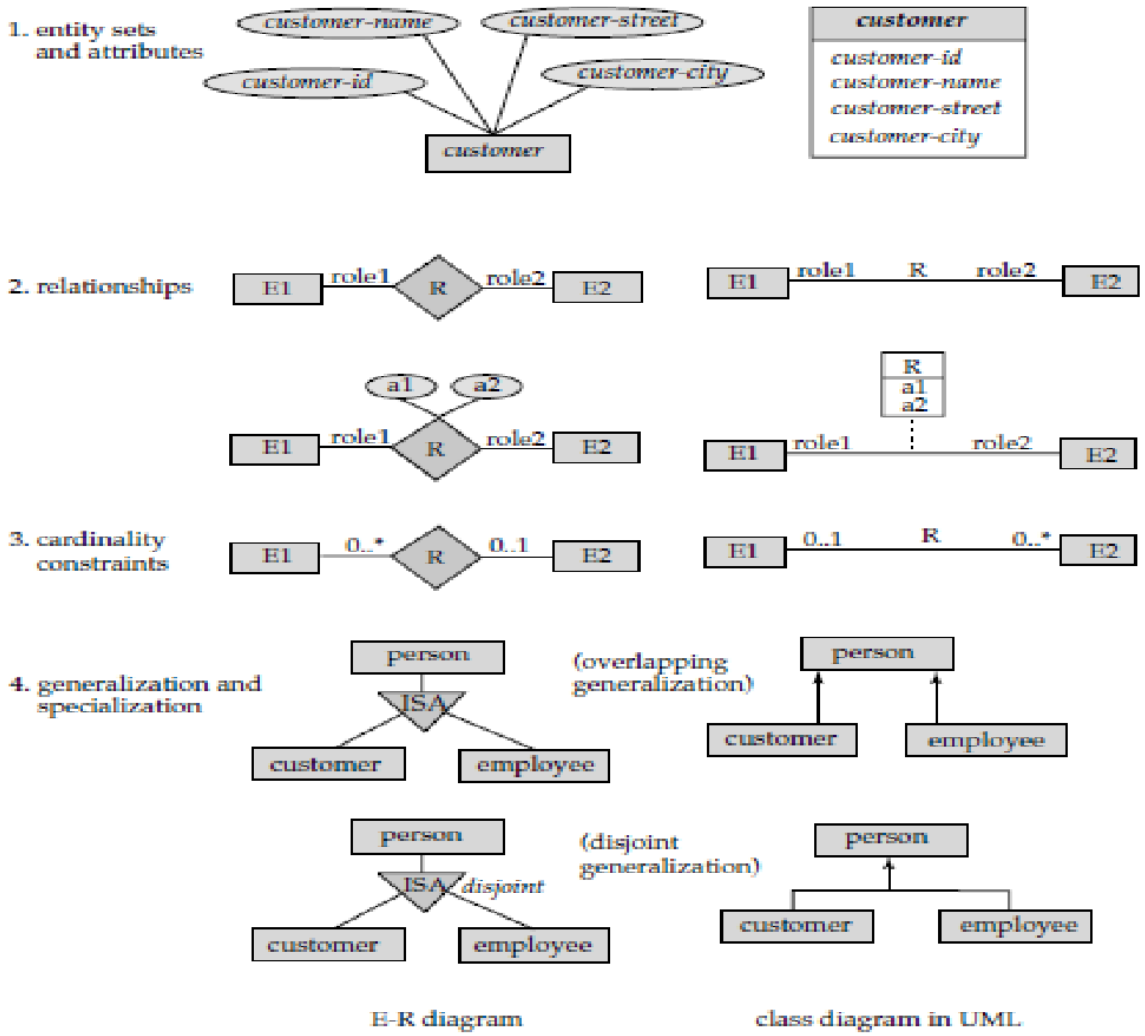
- Two account entity sets—*savings-account* and *checking-account*—with the common attributes of *account-number* and *balance*; in addition, *savings-account* has the attribute *interest-rate* and *checking-account* has the attribute *overdraft-amount*.
- The *loan* entity set, with the attributes *loan-number*, *amount*, and *originating-branch*.
- The weak entity set *loan-payment*, with attributes *payment-number*, *payment-date*, and *payment-amount*.

Step-2 identify relationship with cardinality, type, and participation

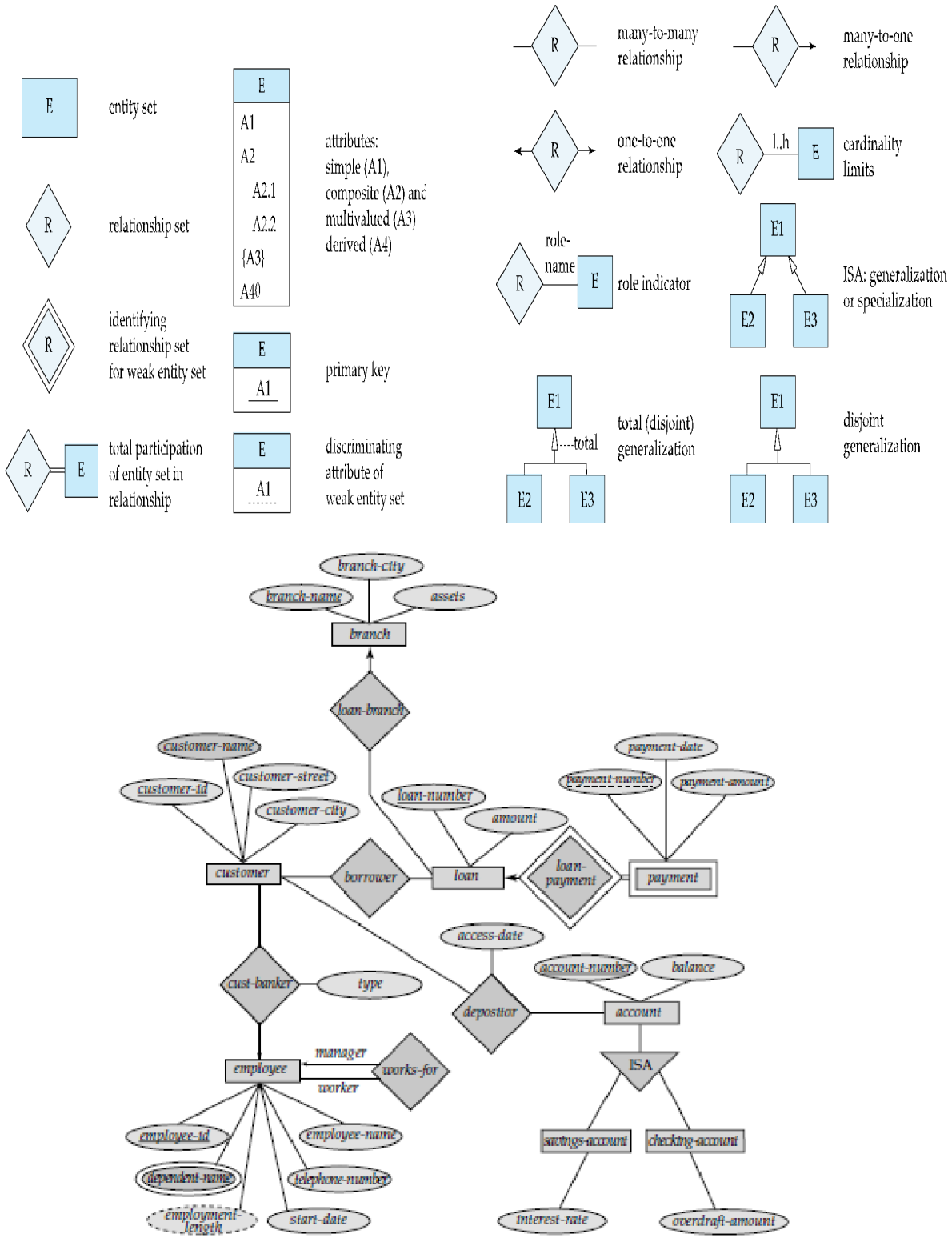
We now return to the rudimentary design scheme of Section 2.8.2.2 and specify the following relationship sets and mapping cardinalities. In the process, we also refine some of the decisions we made earlier regarding attributes of entity sets.

- *borrower*, a many-to-many relationship set between *customer* and *loan*.
- *loan-branch*, a many-to-one relationship set that indicates in which branch a loan originated. Note that this relationship set replaces the attribute *originating-branch* of the entity set *loan*.
- *loan-payment*, a one-to-many relationship from *loan* to *payment*, which documents that a payment is made on a loan.
- *depositor*, with relationship attribute *access-date*, a many-to-many relationship set between *customer* and *account*, indicating that a customer owns an account.
- *cust-banker*, with relationship attribute *type*, a many-to-one relationship set expressing that a customer can be advised by a bank employee, and that a bank employee can advise one or more customers. Note that this relationship set has replaced the attribute *banker-name* of the entity set *customer*.
- *works-for*, a relationship set between *employee* entities with role indicators *manager* and *worker*; the mapping cardinalities express that an employee works for only one manager and that a manager supervises one or more employees. Note that this relationship set has replaced the *manager* attribute of *employee*.

Step-3 use symbols to draw ER/EER model -<u>Old approach symbols-its representation in UML</u>



<u>New Approach notations-as per 6th edition of Korth</u>



ER-Diagram banking example –Old approach

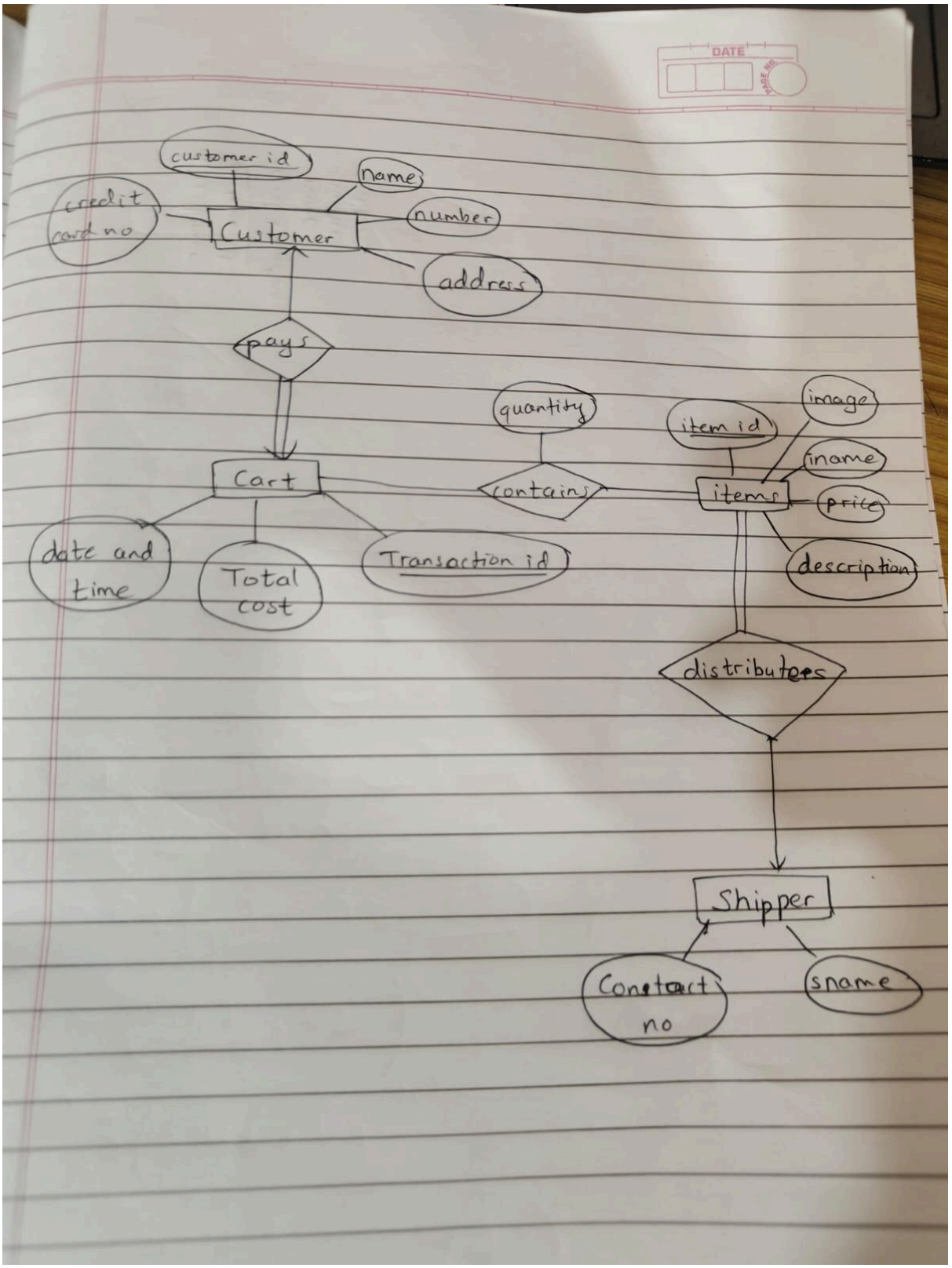
Convert this using New approach

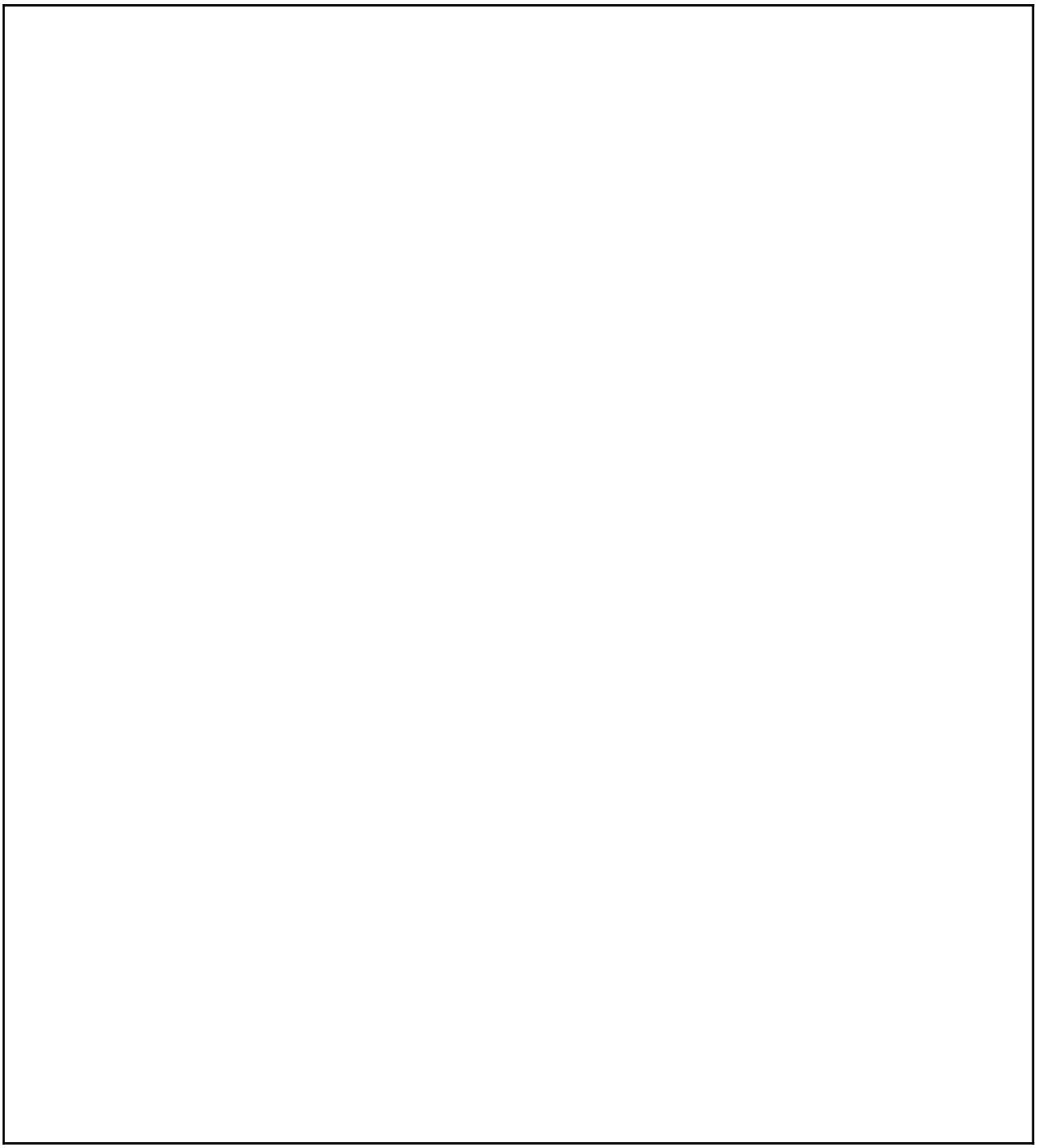
Description of Problem Statement:-NAME OF CASE STUDY- AMAZON

Amazon, a leading e-commerce platform, aims to streamline its order fulfillment process. To achieve this, the company requires a database system to manage customer information, track orders, handle payments, manage inventory, and coordinate shipping.

The system aims to optimize order processing by efficiently managing customer orders, tracking inventory availability, calculating total costs, and coordinating shipping through designated shippers.

E-R /EER diagram for the Problem to be implemented.





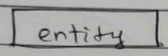
Post Lab Assignment:

1) Describe various symbols used in E-R Diagram and EER diagram

Postlab :-

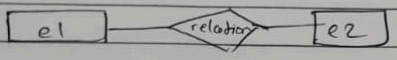
1) Entity

→ It is an object that exists and is distinguishable from other objects.



2) Relation

→ It is an association among several entities.



3) attributes

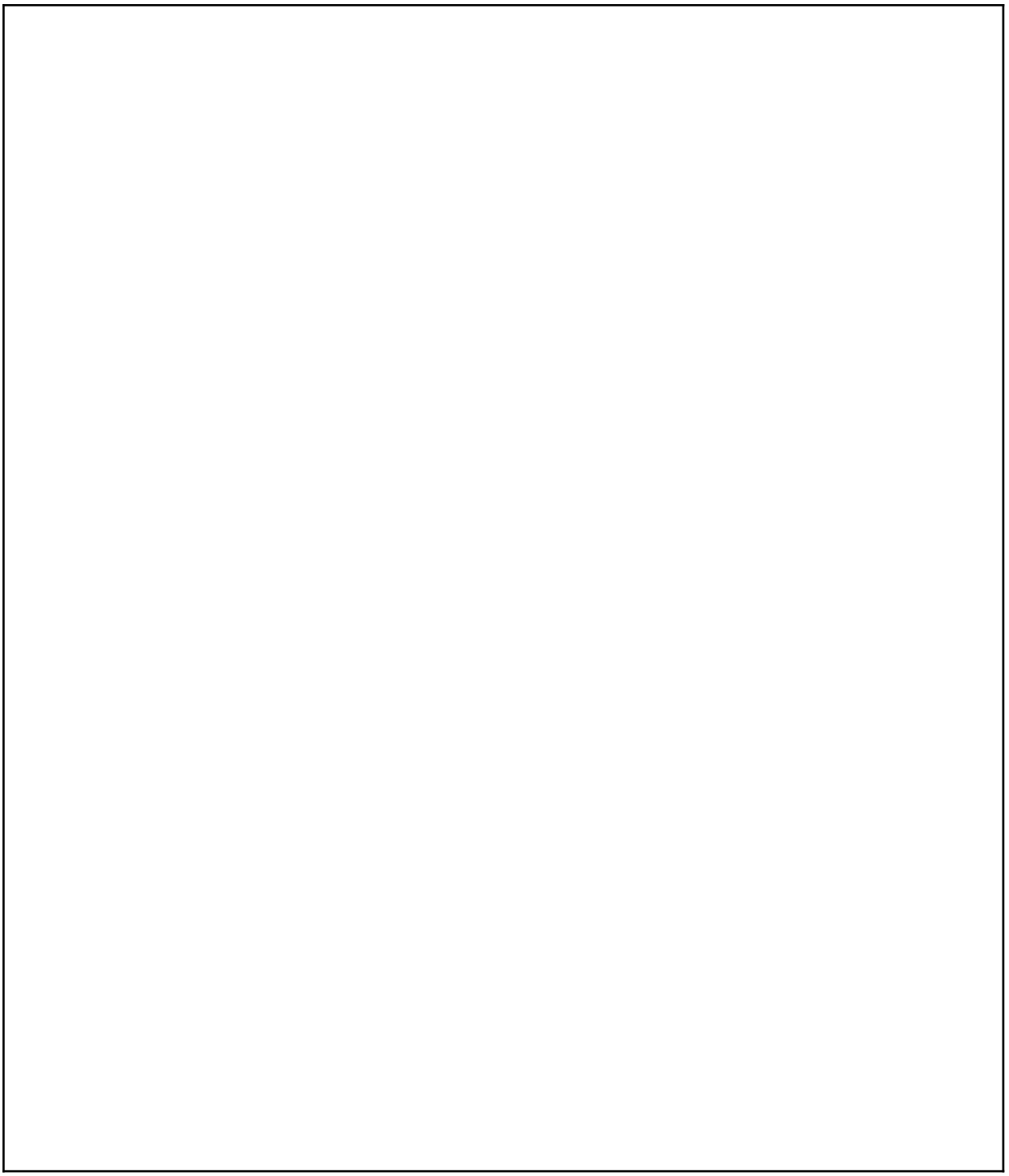
→ They are the properties or characteristic that describes an entity in a database.

4) Primary key

→ It provides a way to specify how entities and relations are distinguished.

5) Weak entity

→ A entity that does not have a primary key because of insufficient attributes is called a weak entity.



List of topics for ER diagram

1. Facebook system
2. Stock exchange syst
3. Wikipedia
4. Youtube
5. Traffic monitoring system
6. Dmart
7. Amazon
8. Twitter
9. Instagram
10. Olx.com
11. Hike
12. Whatsapp
13. Flipcart
14. Yahoo
15. Google search
16. Bio research
17. Bookmyshow
18. Election system- targeting the voters
19. Inventory management system
20. Library management system
21. College information management system
22. Banking system
23. Hospital management system
24. Airline reservation system
25. Railway reservation system
26. Ticket booking system
27. Hotel reservation system
28. Ola.
29. Other topics after the discussion and approval of subject teacher

SE-Computer-A	Roll number :9913
Experiment no. : 2	Date of Implementation : 6/2/2024

Related Course outcome : At the end of the course, Students will be able to design EER model and develop relational model

Rubrics for assessment of Experiment:

Indicator	Poor	Average	Good
Timeliness ● Maintains assignment deadline (3)	Assignment not done (0)	One or More than One week late (1-2)	Maintains deadline (3)
Completeness and neatness ● Complete all parts of ER diagram(3)	N/A	< 80% complete (1-2)	100% complete (3)
Originality ● Extent of plagiarism(2)	Copied it from someone else(0)	At least few questions have been done without copying(1)	Assignment has been solved completely without copying (2)
Knowledge ● In depth knowledge of the assignment(2)	Unable to answer 2 questions(0)	Unable to answer 1 question (1)	Able to answer 2 questions (2)

Assessment Marks :

Timeliness	
Completeness and neatness	
Originality	
Knowledge	
Total	

Total : (Out of 10)

Teacher's Sign :

Name Student	Mark lopes	**Roll No.**	9913
Lab Experiment No.	2	**Date**	6/2/2024
Expt. Title	Mapping / Convert EER diagram to Relational Model of Problem		

Aim /objective: To map ER/EER diagram to relational model.

Theory:

- ER diagram is converted into the tables in relational model.
- This is because relational models can be easily implemented by RDBMS like MySQL, Oracle etc.

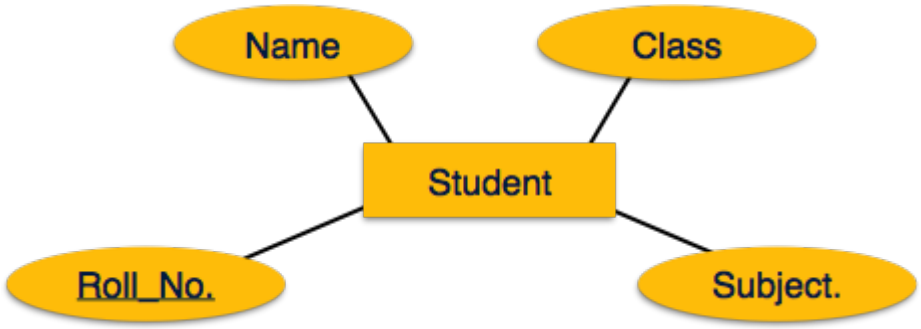
ER diagrams mainly comprise of −

- Entity and its attributes
- Relationship, which is association among entities.

1)Mapping of ER model to Relational Model

Mapping Entity

An entity is a real-world object with some attributes.

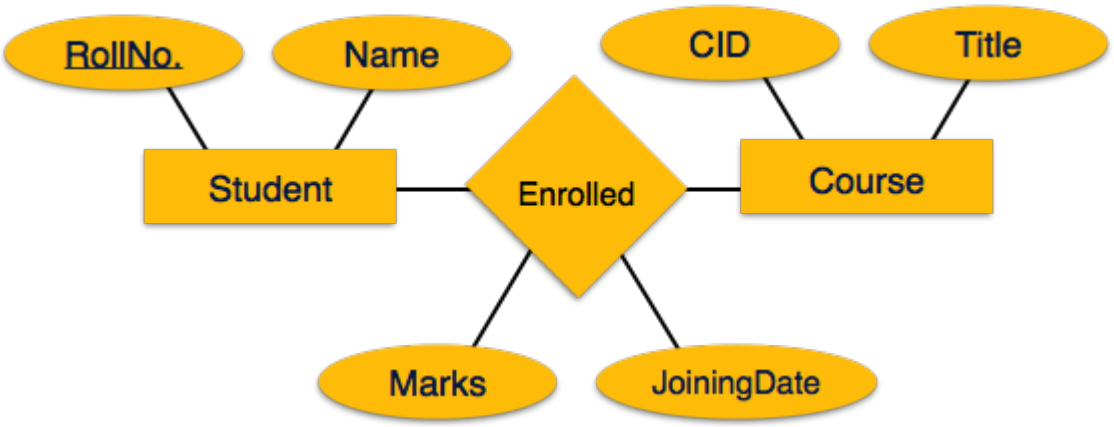


Mapping Process (Algorithm)

- Create table for each entity set.
- Entity's attributes should become fields of tables with their respective data types.
- Declare primary key.

Mapping Relationship

A relationship is an association among entities.

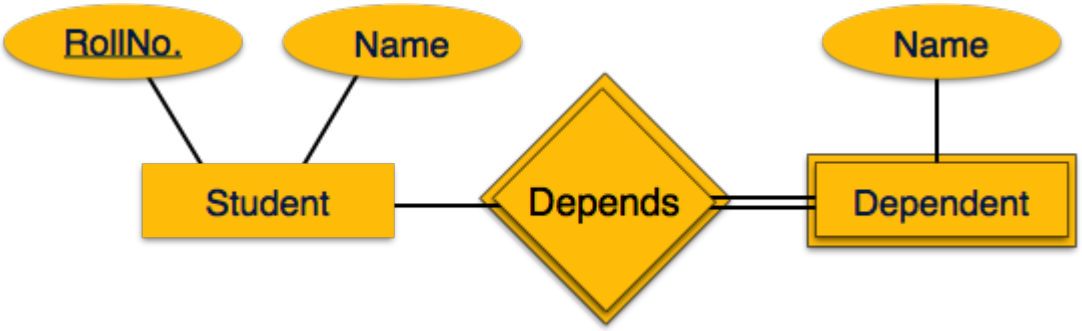


Mapping Process

- Create table for a relationship.
- Add the primary keys of all participating Entities as fields of table with their respective data types.
- If relationship has any attribute, add each attribute as field of table.
- Declare a primary key composing all the primary keys of participating entities.
- Declare all foreign key constraints.

Mapping Weak Entity Sets

A weak entity set is one which does not have any primary key associated with it.

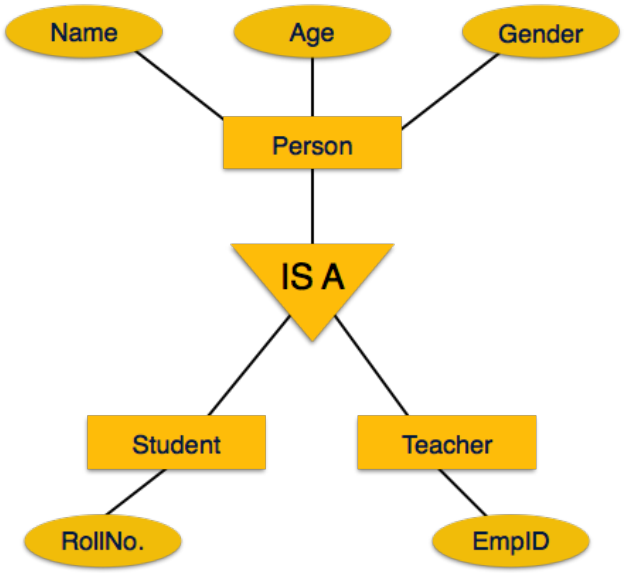


Mapping Process

- Create table for weak entity set.
- Add all its attributes to table as field.
- Add the primary key of identifying entity set.
- Declare all foreign key constraints.

Mapping Hierarchical Entities

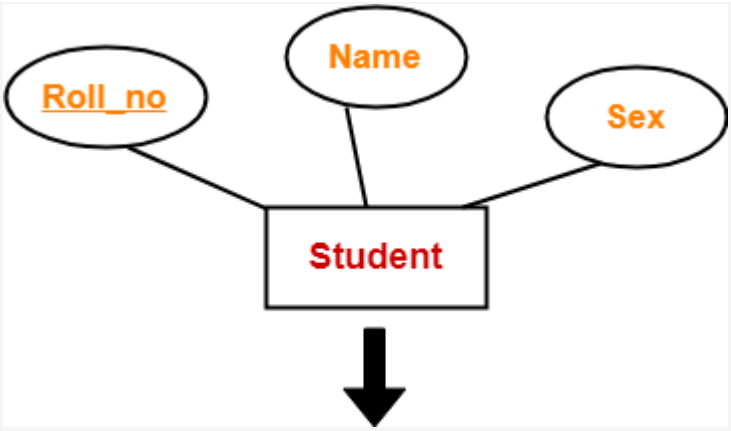
ER specialization or generalization comes in the form of hierarchical entity sets.



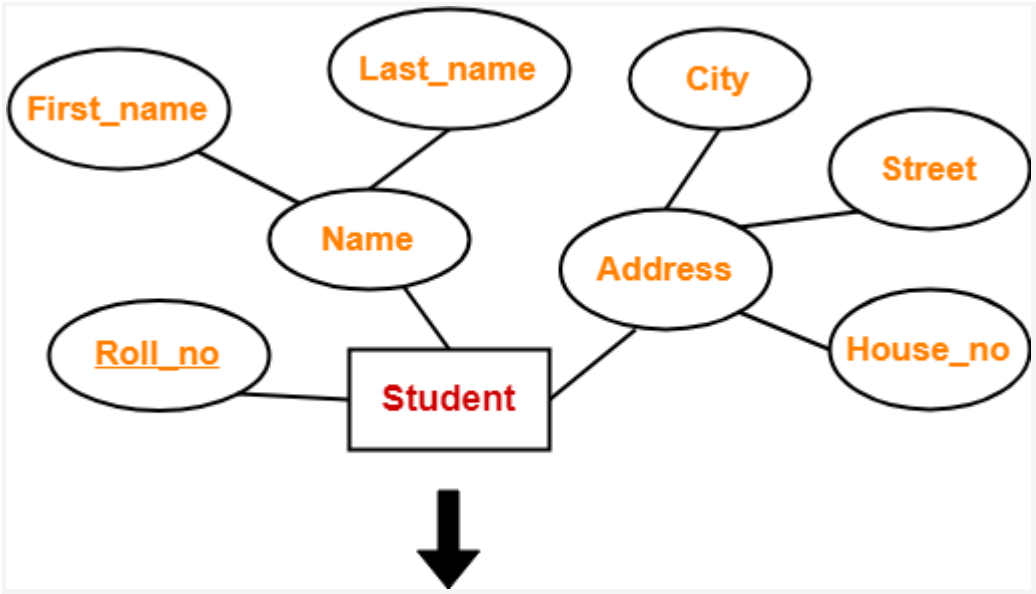
Mapping Process

- Create tables for all higher-level entities.
- Create tables for lower-level entities.
- Add primary keys of higher-level entities in the table of lower-level entities.
- In lower-level tables, add all other attributes of lower-level entities.
- Declare primary key of higher-level table and the primary key for lower-level table.
- Declare foreign key constraints.

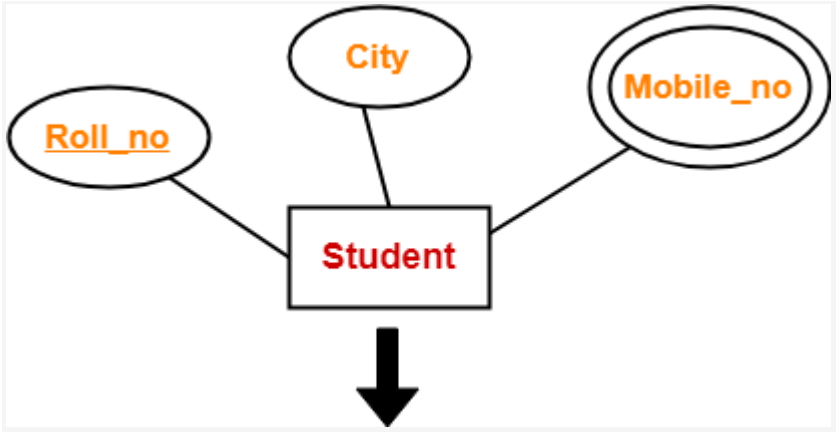
Rules:



Roll_no	Name	Sex

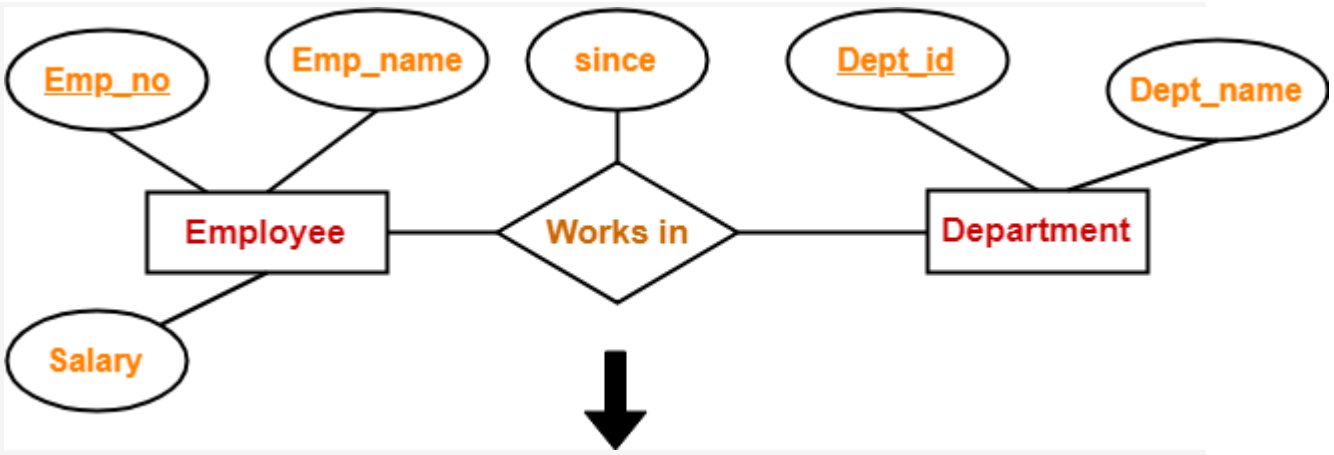


Roll_no	First_name	Last_name	House_no	Street	City

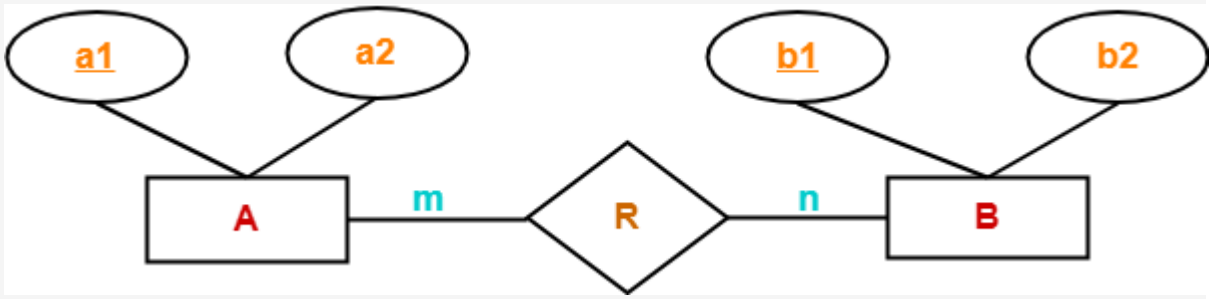


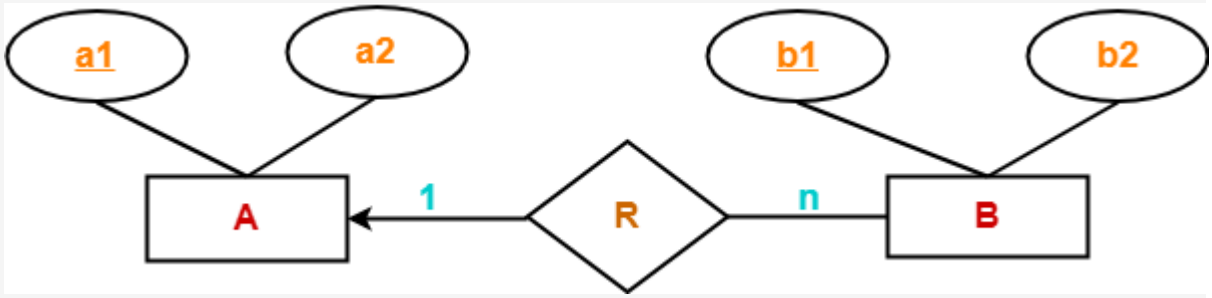
Roll_no	City

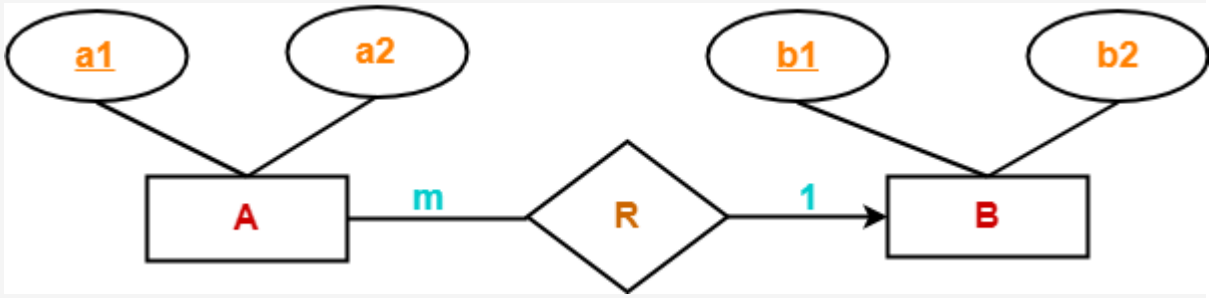
Roll_no	Mobile_no

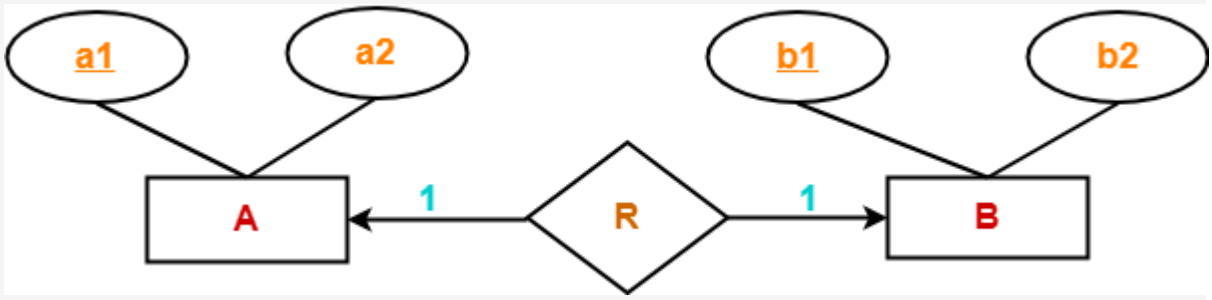


Emp_no	Dept_id	since





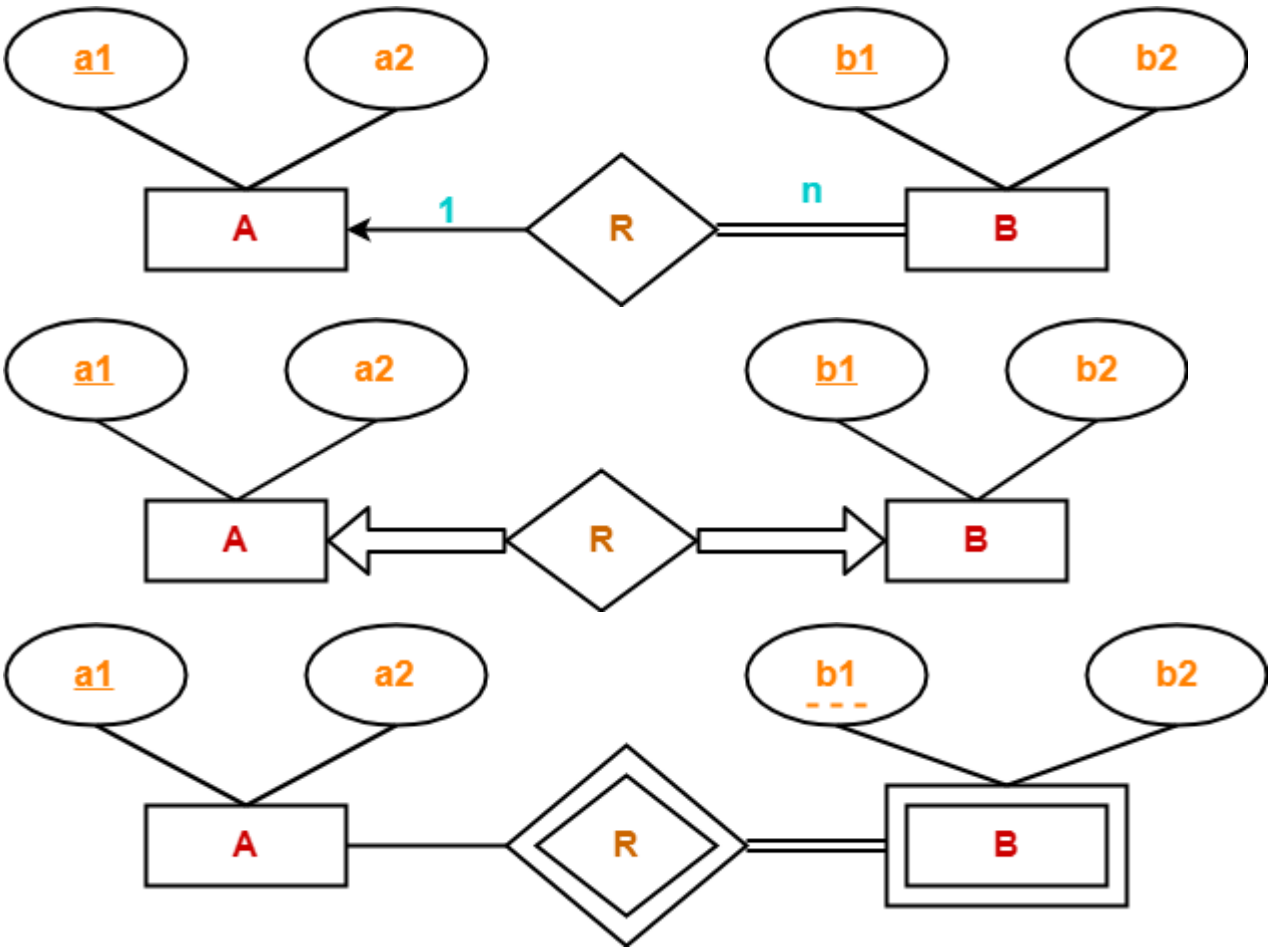




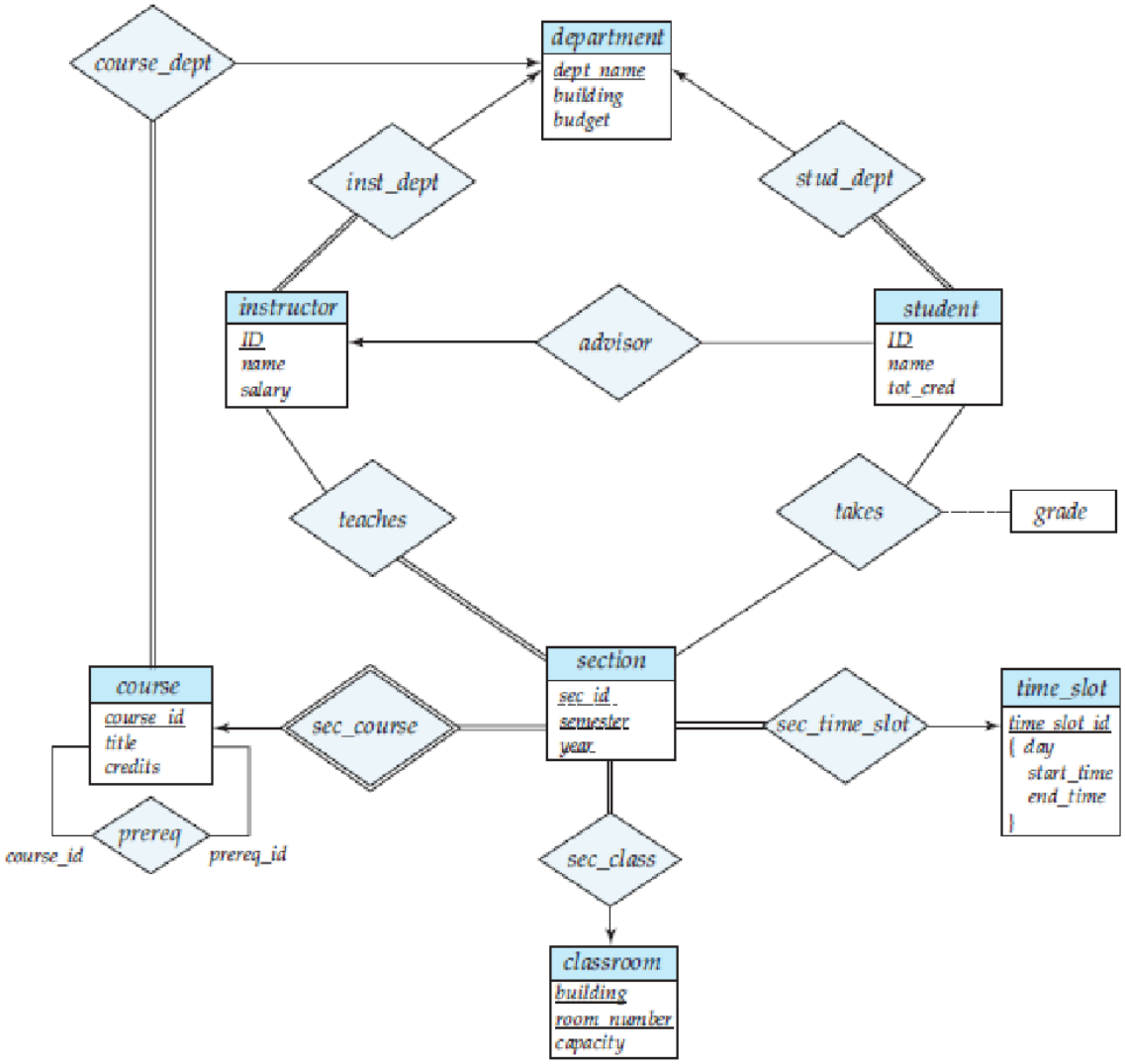
Thumb Rules to Remember

While determining the minimum number of tables required for binary relationships with given cardinality ratios, following thumb rules must be kept in mind-

- For binary relationship with cardinality ration m : n , separate and individual tables will be drawn for each entity set and relationship.
- For binary relationship with cardinality ratio either m : 1 or 1 : n , always remember “many side will consume the relationship” i.e. a combined table will be drawn for many side entity set and relationship set.
- For binary relationship with cardinality ratio 1 : 1 , two tables will be required. You can combine the relationship set with any one of the entity sets.

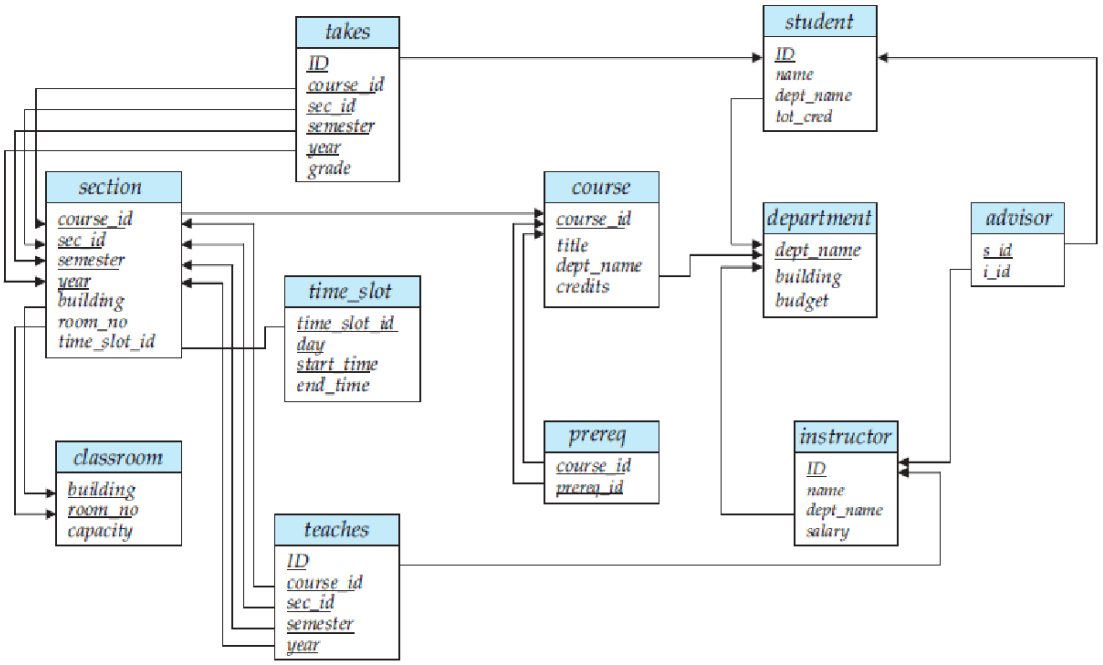

a1
a2
b1
b2
A
1
R
n
B
a1
a2
b1
b2
A
R
B
a1
a2
b1
b2
A
R
B

Sample Example



ER diagram of university database

teaches (*ID*, *course_id*, *sec_id*, *semester*, *year*)
takes (*ID*, *course_id*, *sec_id*, *semester*, *year*, *grade*)
prereq (*course_id*, *prereq_id*)
advisor (*s_ID*, *i_ID*)
sec_course (*course_id*, *sec_id*, *semester*, *year*)
sec_time_slot (*course_id*, *sec_id*, *semester*, *year*, *time_slot_id*)
sec_class (*course_id*, *sec_id*, *semester*, *year*, *building*, *room_number*)
inst_dept (*ID*, *dept_name*)
stud_dept (*ID*, *dept_name*)
course_dept (*course_id*, *dept_name*)



Schema diagram /relational model of University database

classroom(<u>*building*</u>, <u>*room_number*</u>, *capacity*)
department(<u>*dept_name*</u>, *building*, *budget*)
course(<u>*course_id*</u>, *title*, *dept_name*, *credits*)
instructor(<u>*ID*</u>, *name*, *dept_name*, *salary*)
section(<u>*course_id*</u>, <u>*sec_id*</u>, <u>*semester*</u>, <u>*year*</u>, *building*, *room_number*, *time_slot_id*)
teaches(<u>*ID*</u>, <u>*course_id*</u>, <u>*sec_id*</u>, <u>*semester*</u>, <u>*year*</u>)
student(<u>*ID*</u>, *name*, *dept_name*, *tot_cred*)
takes(<u>*ID*</u>, <u>*course_id*</u>, <u>*sec_id*</u>, <u>*semester*</u>, <u>*year*</u>, *grade*)
advisor(<u>*s_ID*</u>, *i_ID*)
time_slot(<u>*time_slot_id*</u>, <u>*day*</u>, <u>*start_time*</u>, *end_time*)
prereq(<u>*course_id*</u>, <u>*prereq_id*</u>)

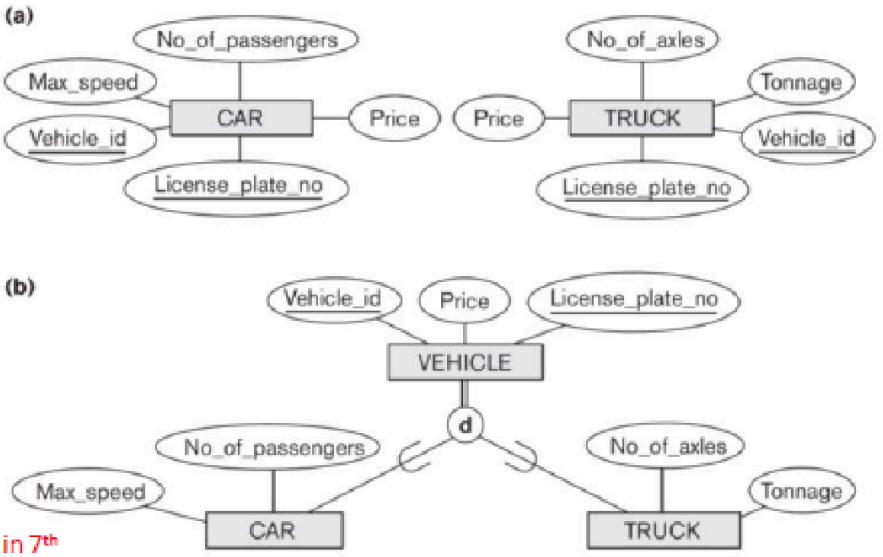
2) Mapping of Specialization or Generalization

- Step 8: Options for Mapping Specialization or Generalization.
  Can be used for shared subclasses.
  - Option 8A: Multiple relations—one for the superclass and one for each subclass
    - For any specialization
      (total or partial, disjoint or overlapping)
    - PK of subclass relation is FK to superclass relation.
    - An equi-join is needed to get all attributes for an entity that is an instance of a subclass. An entity can be represented many times.
    - Consider Figure 9.5a)

- Option 8B: Multiple relations but only for subclasses
  - Only for subclassing that is *total*
  - If specialization is overlapping there can be entities represented in more than one relation
  - Example, see figure 9.5b)

- Option 8C: Single relation representing all classes including one type attribute
  - A type (discriminating) attribute indicates subclass
  - Subclasses must be disjoint
  - Potential for generating many NULL values if many specific attributes exist in the subclasses
  - Example 9.5c)

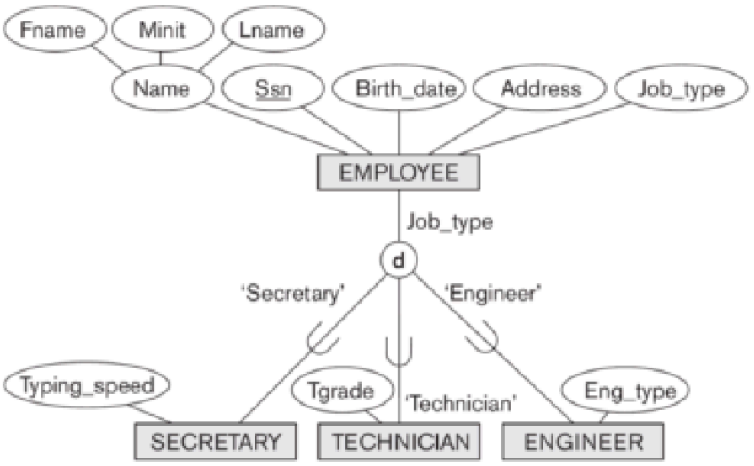
- Option 8D: Single relation representing all classes including multiple type attributes
  - Useful for overlapping subclasses
  - Potential for generating many NULL values if many specific attributes exist in the subclasses
  - Example 9.5d)



4.3 in 7th

Figure 8.3
Generalization. (a) Two entity types, CAR and TRUCK. (b) Generalizing CAR and TRUCK into the superclass VEHICLE.

Figure 8.4
EER diagram notation for an attribute-defined specialization on Job_type.



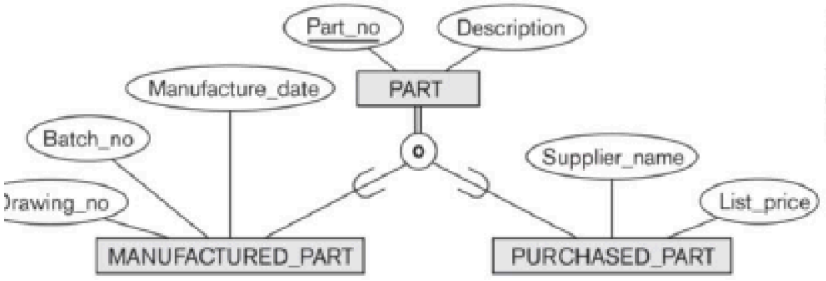


Figure 8.5
EER diagram notation for an overlapping (nondisjoint) specialization.

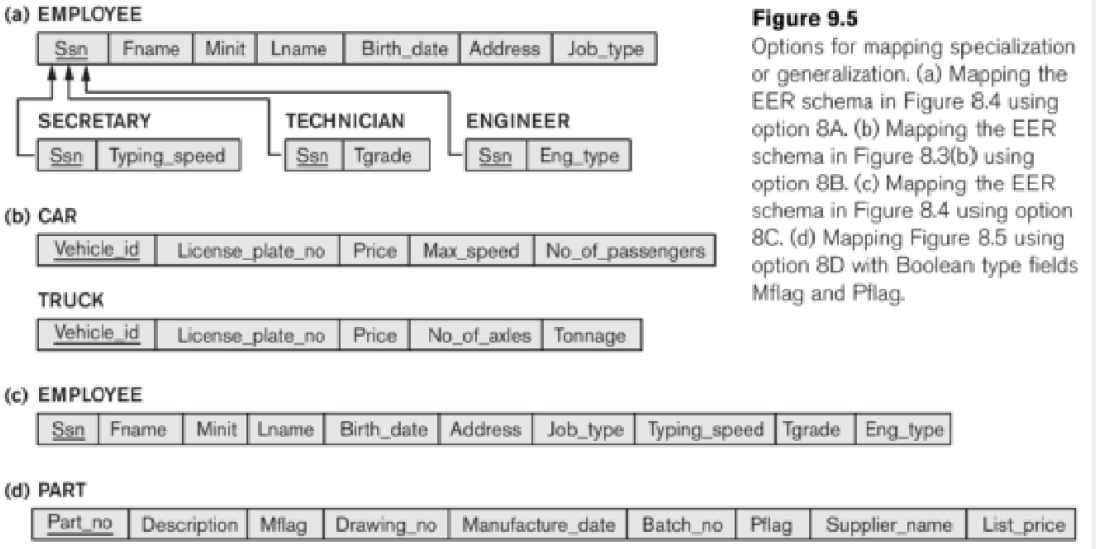


Figure 9.5
Options for mapping specialization or generalization. (a) Mapping the EER schema in Figure 8.4 using option 8A. (b) Mapping the EER schema in Figure 8.3(b) using option 8B. (c) Mapping the EER schema in Figure 8.4 using option 8C. (d) Mapping Figure 8.5 using option 8D with Boolean type fields Mflag and Pflag.

EXAMPLE

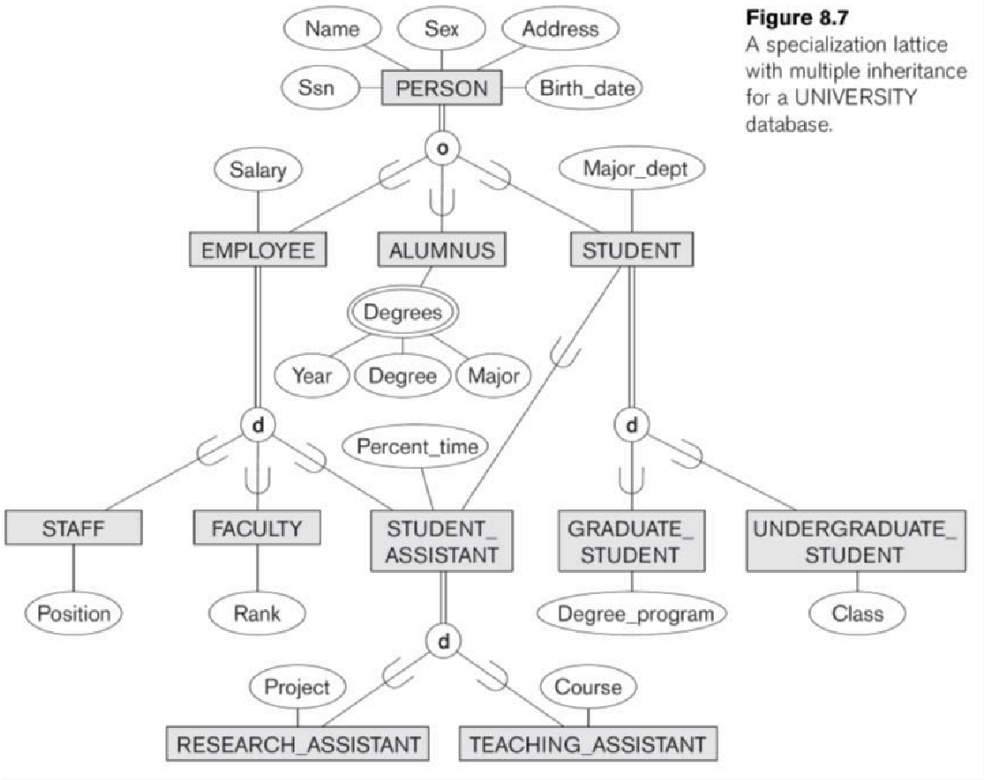


Figure 8.7
A specialization lattice with multiple inheritance for a UNIVERSITY database.

Applied 8A to Person, Employee, Alumnus, Student

Applied 8C to Employee, Staff, Faculty, Student Assistant – *Employee type*

Applied 8D to Student Assistant, Research Assistant, Teaching Assistant – *Ta flag, Ta flag*

Applied 8D to Student, Student Assistant, Graduate Student, Undergraduate Student – *Grad flag, Undergrad flag, Student assist flag*

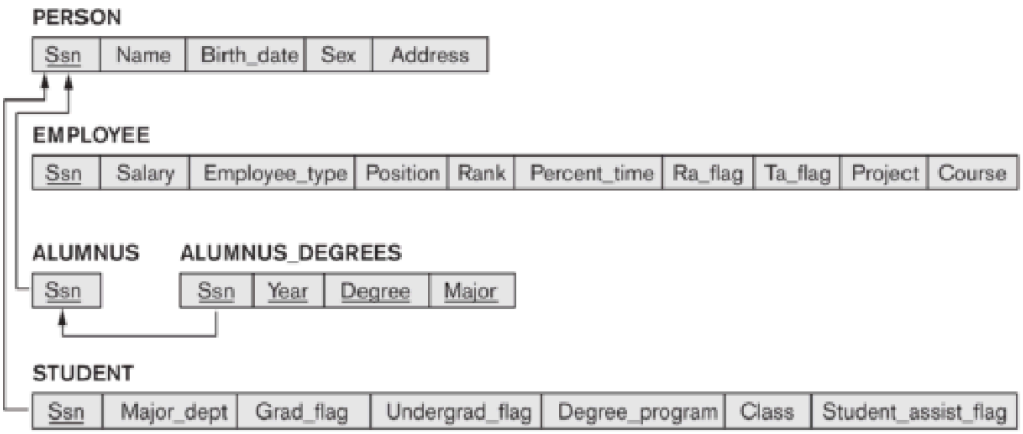
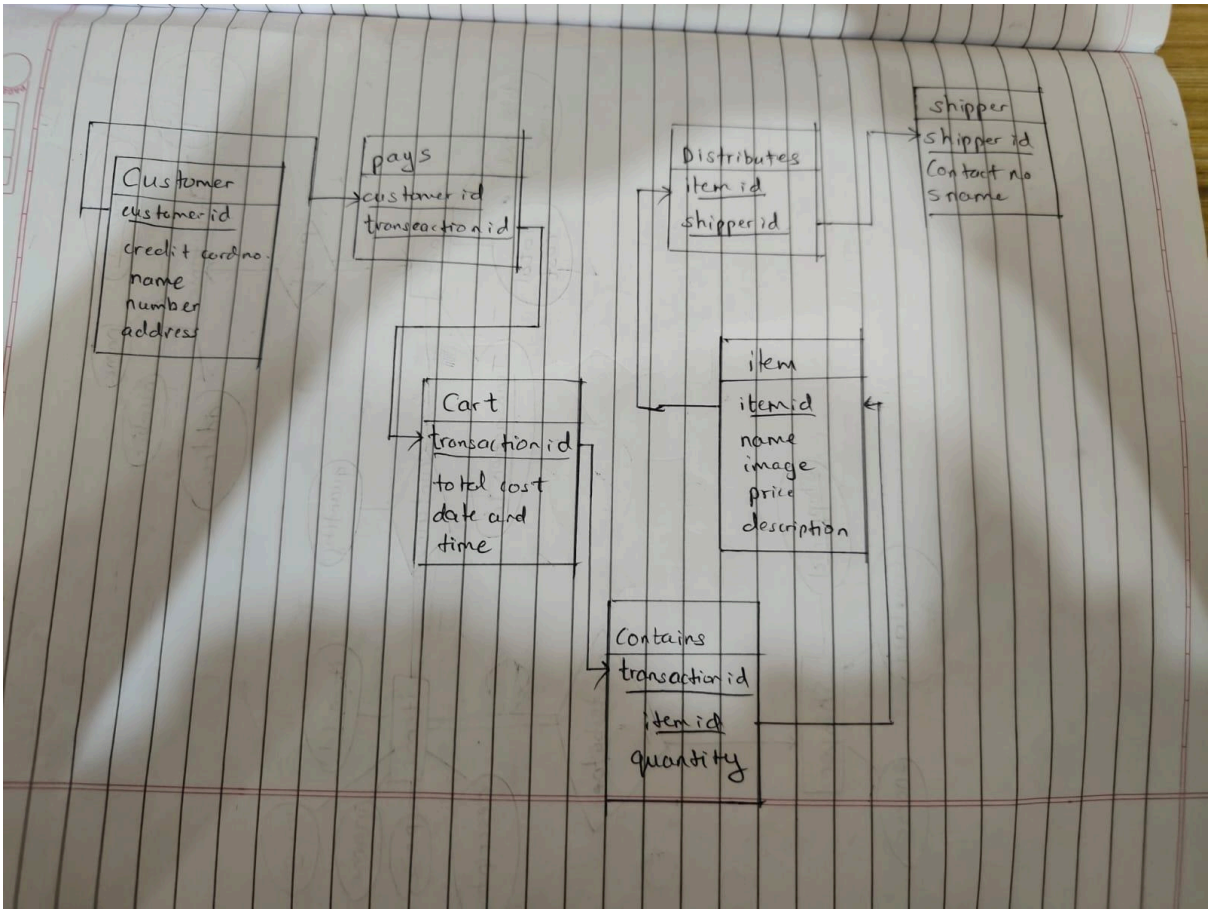


Table 9.1 Correspondence between ER and Relational Models

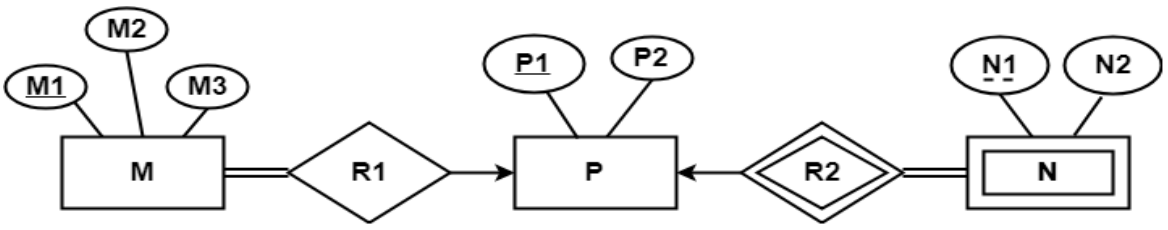
ER MODEL	RELATIONAL MODEL
Entity type	*Entity* relation
1:1 or 1:N relationship type	Foreign key (or *relationship* relation)
M:N relationship type	*Relationship* relation and *two* foreign keys
n-ary relationship type	*Relationship* relation and *n* foreign keys
Simple attribute	Attribute
Composite attribute	Set of simple component attributes
Multivalued attribute	Relation and foreign key
Key attribute	Primary (or secondary) key

Relational Model of Problem

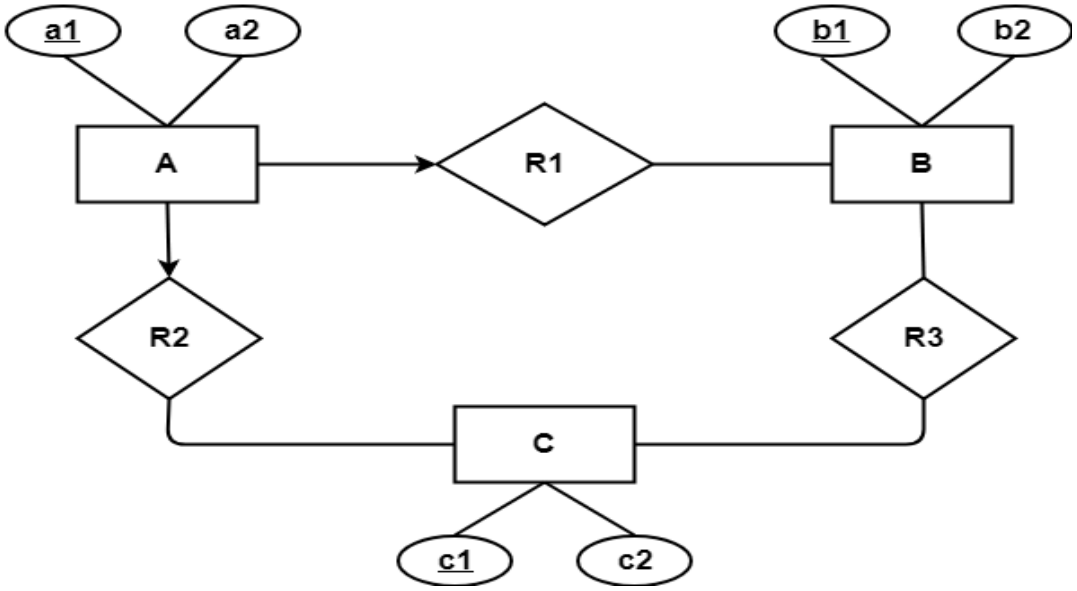


Post Lab Assignment:

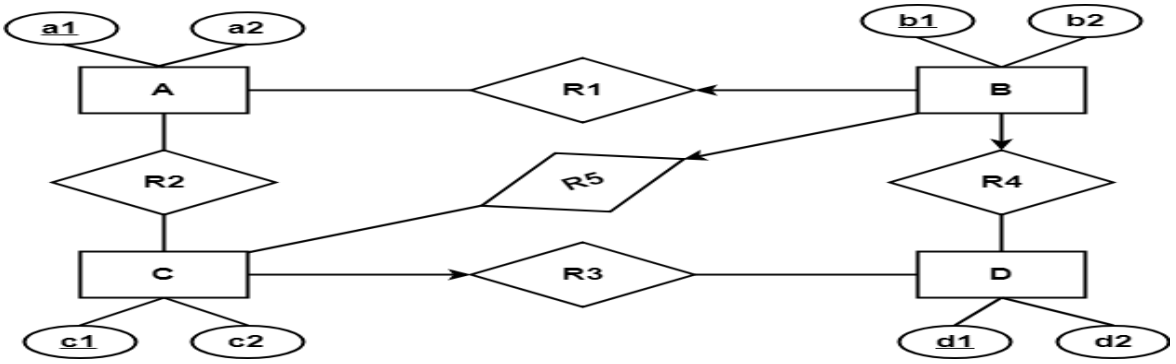
1) Find the minimum number of tables required for the following ER diagram in relational model-

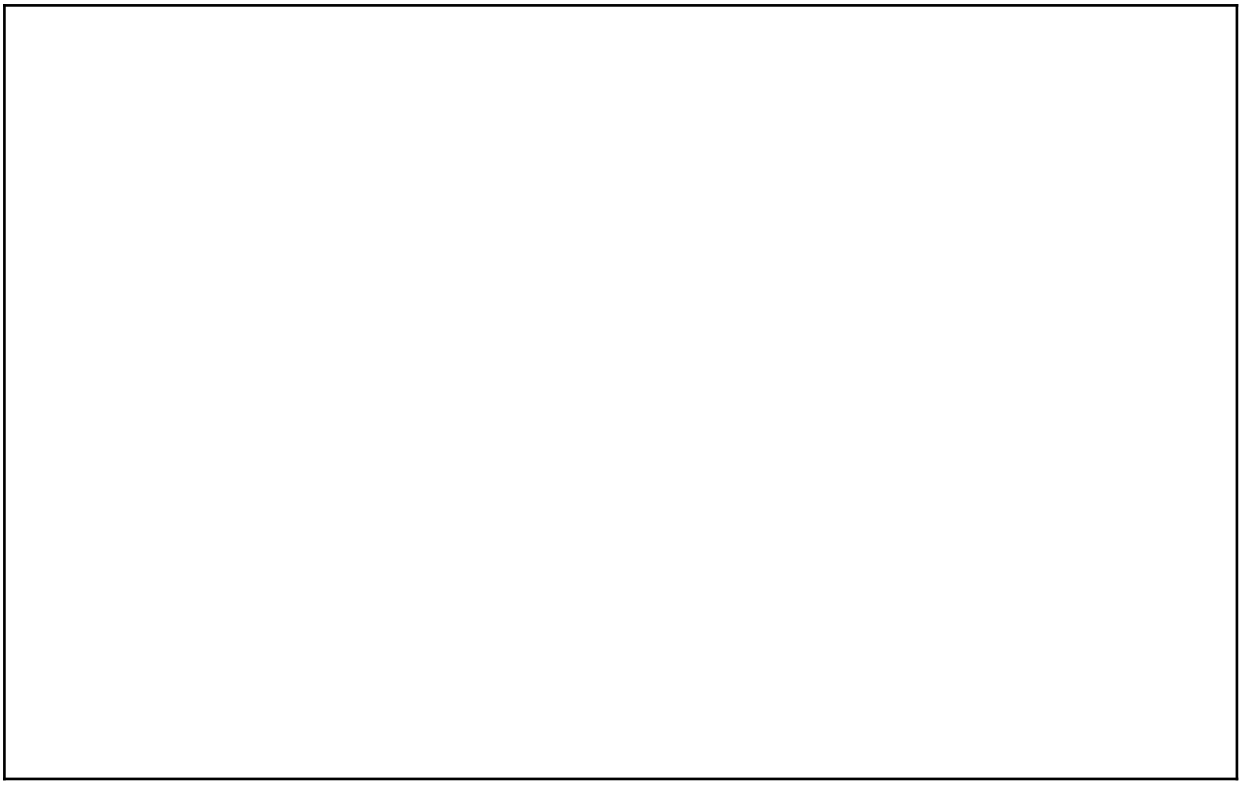


2) Find the minimum number of tables required to represent the given ER diagram in relational model-



3) Find the minimum number of tables required to represent the given ER diagram in relational model-

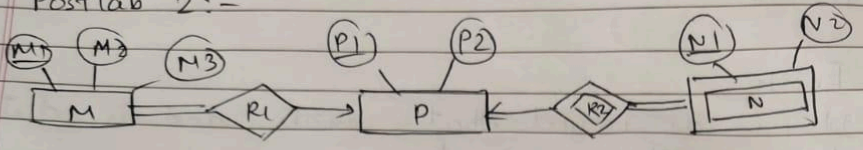




9913
Mark Lopes
SE comps A-batch-C

Postlab 2:-

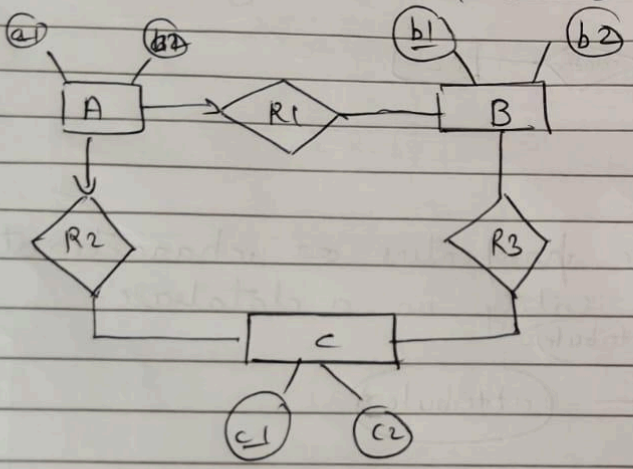
1)



MR1 (M1, M2, M3, P1) ✓
P (P1, P2) ✓
NA2 (N1, N2, P1) ✓

∴ minimum three (3) tables

2)

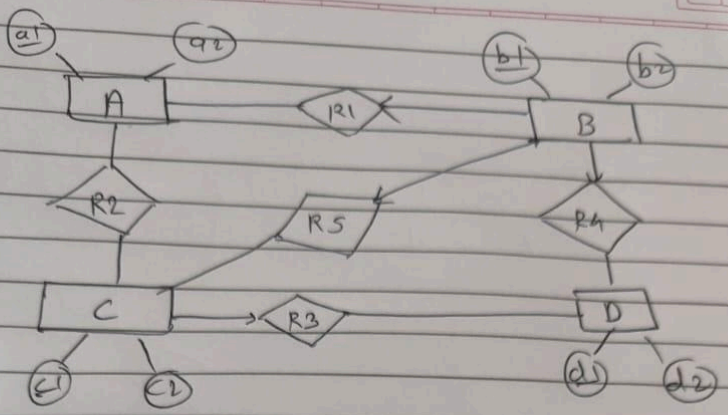


AR1 (a1, a2, b1)
B (b1, b2) ✓
AR2 (a1, a2, c1)
C (c1, c2) ✓
R3 (c1, b1) ✓

→ AR1R2 (a1, a2, b1, c1) ✓

∴ 4 tables minimum

3)



BR1 (b1, b2, a1)
BR5 (b1, b2, c1) → BR1R4R5 (b1, b2, a1, d1, c1) ✓
BR4 (b1, b2, d1)
CR3 (c1, c2, d1) ✓
A (a1, a2) ✓
R2 (a1, c1) ✓
D (d1, d2) ✓

∴ minimum 5 tables

SE-Computer A Batch C	Roll number : 9913
Experiment no. : 3(Part-1)	Date of Implementation :
Aim : To implement data definition language (DDL) commands	
Tool Used : PostgreSQL	
Related Course outcome : At the end of the course, Students will be able to Use SQL : Standard language of relational database	

Rubrics for assessment of Experiment:

Indicator	Poor	Average	Good
Timeliness ● Maintains assignment deadline (3)	Assignment not done (0)	One or More than One week late (1-2)	Maintains deadline (3)
Completeness and neatness ● Complete all parts of assignment(3)	N/A	< 80% complete (1-2)	100% complete (3)
Originality ● Extent of plagiarism(2)	Copied it from someone else(0)	At least few questions have been done without copying(1)	Assignment has been solved completely without copying (2)
Knowledge ● In depth knowledge of the assignment(2)	Unable to answer 2 questions(0)	Unable to answer 1 question (1)	Able to answer 2 questions (2)

Assessment Marks :

Timeliness	
Completeness and neatness	
Originality	
Knowledge	
Total	

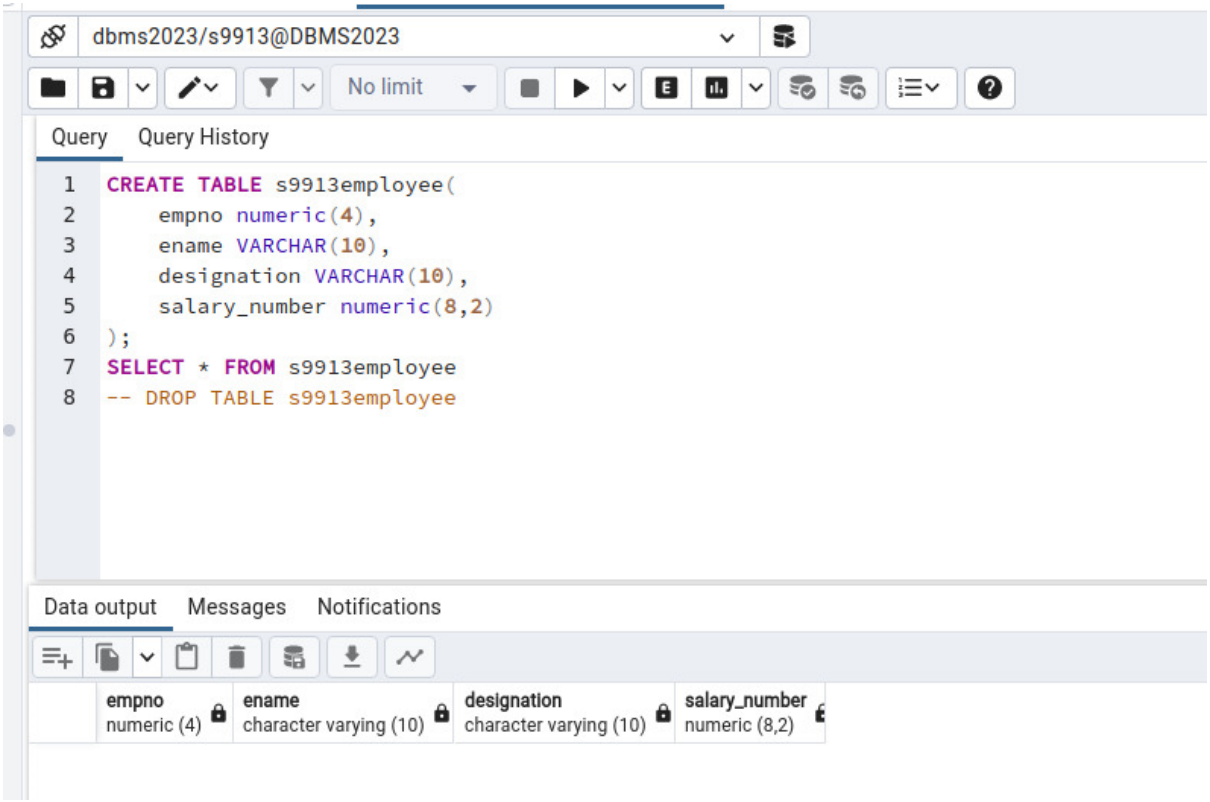
Total : (Out of 10)

Teacher's Sign :

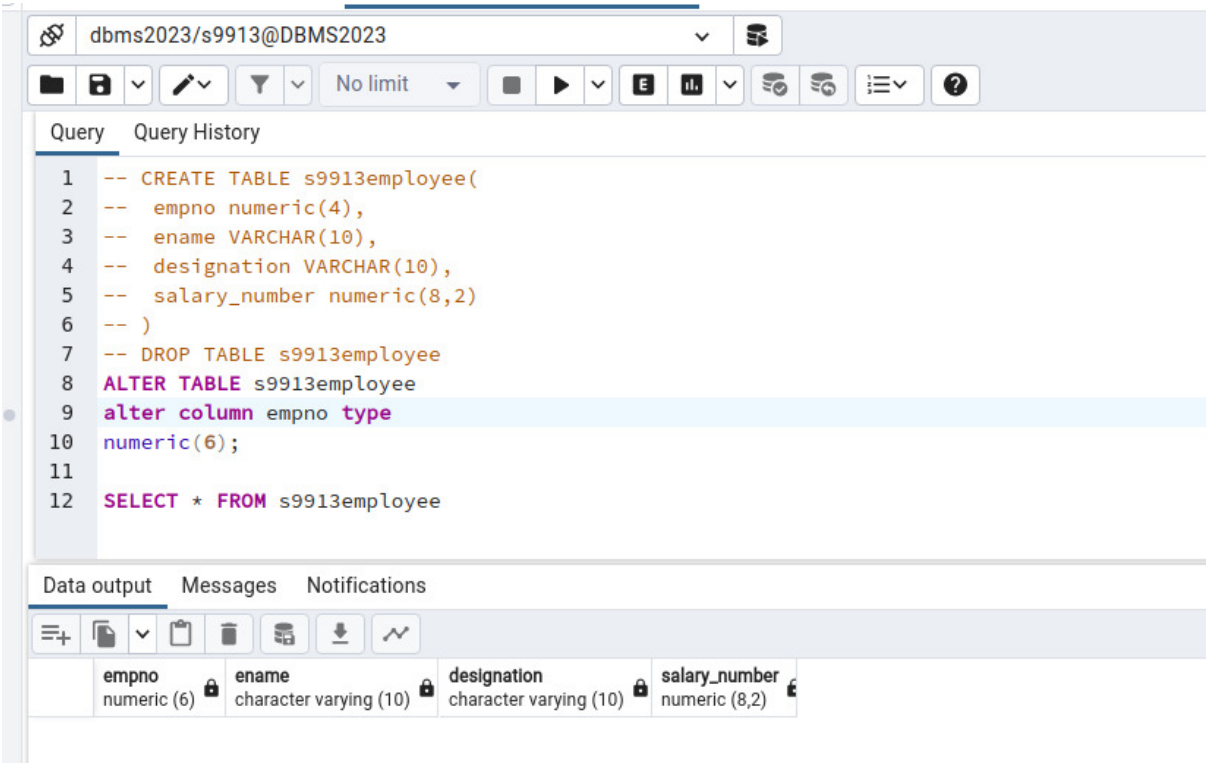
	EXPERIMENT 3	DDL Commands
	Aim	To implement DDL – Data definition language command
	Tools	PostgreSQL/MYSQL
	Theory	**SQL:** It is structured query language, basically used to pass the query to retrieve and manipulate the information from database **DDL:** The Data Definition Language (DDL) is used to create the database (i.e. tables, keys, relationships etc), maintain the structure of the database and destroy databases and database objects. Eg. Create, Drop, Alter, Describe, Truncate **1. CREATE statements: It is used to create the table.** CREATE TABLE table_name(columnName1 datatype(size), columnName2 datatype(size),……); 2. **DROP statements:** To destroy an existing database, table, index, or view. If a table is dropped all records held within it are lost and cannot be recovered. DROP TABLE table_name; 3. **ALTER statements:** To modify an existing database object. **Adding new columns:** Alter table table_name Add(New_columnName1 datatype(size), New_columnName2 datatype(size),……); **Dropping a columns from a table** : Alter table table_name DROP column columnName: **Modifying Existing columns:** Alter table table_name Modify (columnName1 Newdatatype(Newsize)); 4. **Describe statements:** To describe the structure (column and data types) of an existing database, table, index, or view. DESC table_name; 5. **Truncate statements:** To destroy the data in an existing database, table, index, or view. If a table is truncated all records held within it are lost and cannot be recovered but the table structure is maintained. TRUNCATE TABLE table_name;

	Procedure	1. Write a query to create a table employee with empno, ename, designation, and salary. Emp (empno number (4), ename varchar (10), designation varchar (10), salary number (8,2)); 2. Write a Query to Alter the column empno number (4) to empno number (6). 3. Write a Query to Alter the table employee with multiple columns (empno, ename.) 4. Write a query to add a new column in to employee as qualification varchar2(6) 5. Write a query to add multiple columns in to employee dob date , doj date 6. Write a query to drop a column 'doj' from an existing table employee 7. Write a query to drop multiple columns 'dob' and 'qualification' from employee 8. Truncate table EMP 9. Drop table EMP
	Post Lab Questions:	1. What is Data Dictionary? 2. What is Schema? 3. What are different data types in SQL?

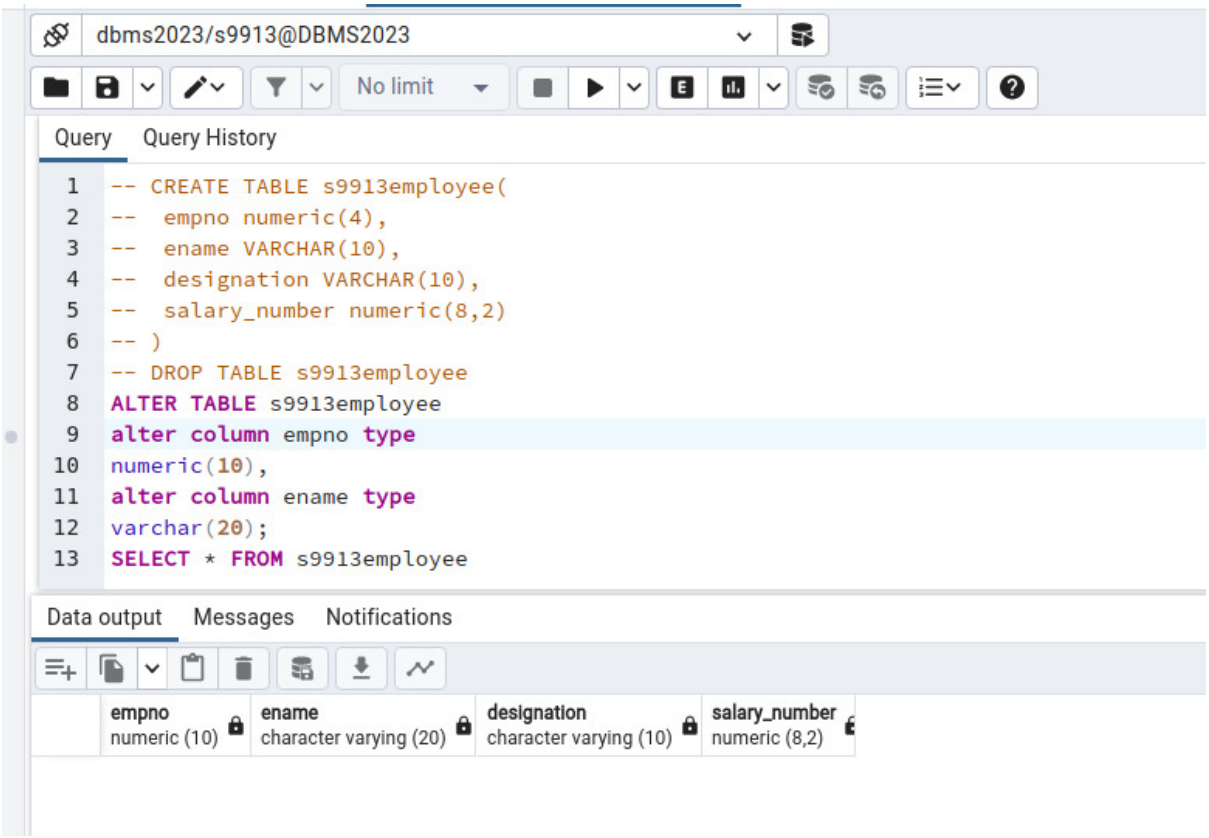
1



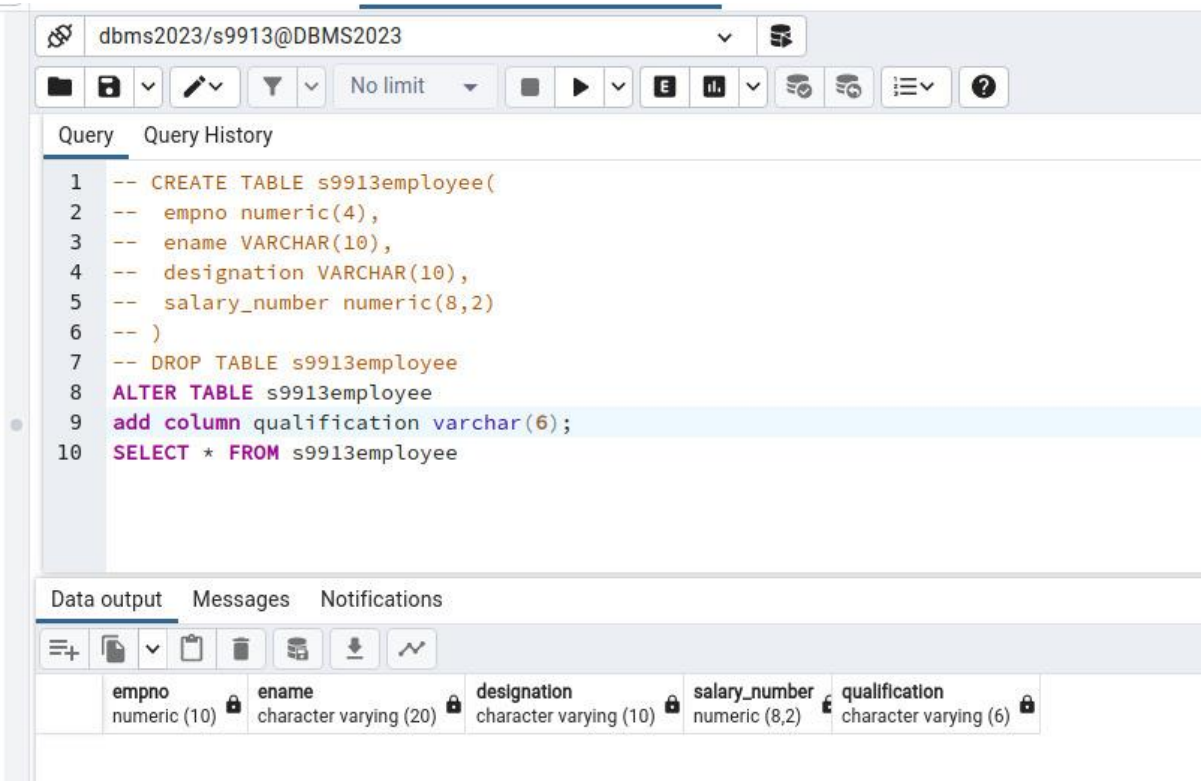
2


dbms2023/s9913@DBMS2023
No limit
Query
Query History
1 -- CREATE TABLE s9913employee(
2 -- empno numeric(4),
3 -- ename VARCHAR(10),
4 -- designation VARCHAR(10),
5 -- salary_number numeric(8,2)
6 --)
7 -- DROP TABLE s9913employee
8 ALTER TABLE s9913employee
9 alter column empno type
10 numeric(6);
11
12 SELECT * FROM s9913employee
Data output
Messages
Notifications
empno numeric (6)
ename character varying (10)
designation character varying (10)
salary_number numeric (8,2)

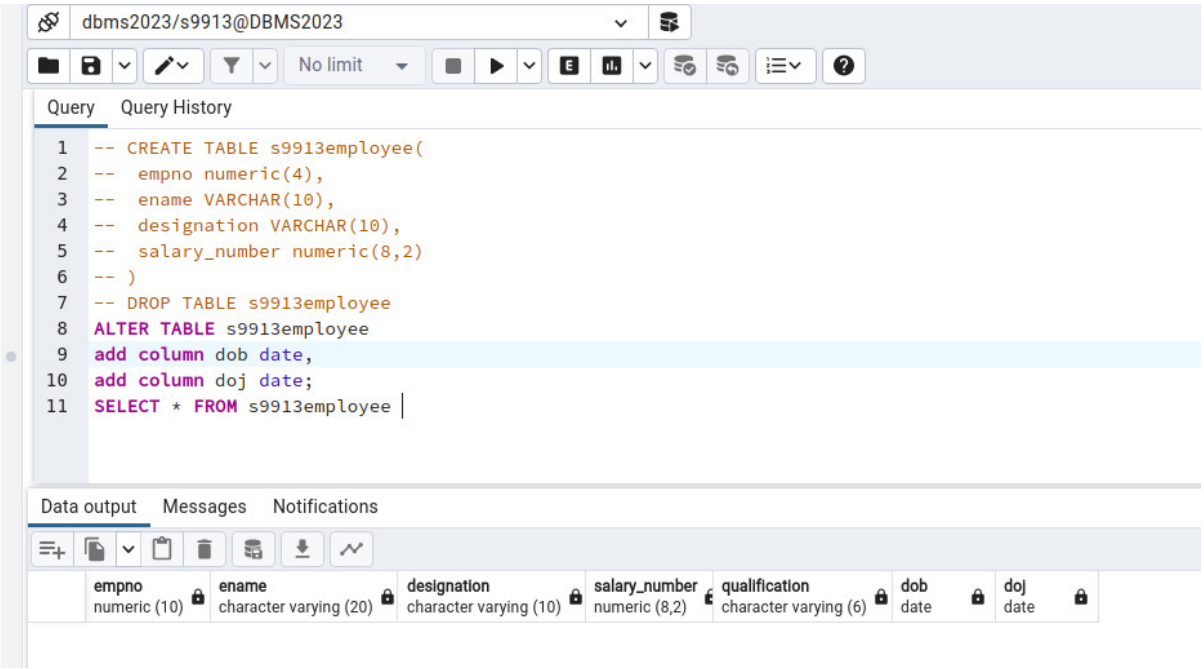
3


dbms2023/s9913@DBMS2023
No limit
Query
Query History
1 -- CREATE TABLE s9913employee(
2 -- empno numeric(4),
3 -- ename VARCHAR(10),
4 -- designation VARCHAR(10),
5 -- salary_number numeric(8,2)
6 --)
7 -- DROP TABLE s9913employee
8 ALTER TABLE s9913employee
9 alter column empno type
10 numeric(10),
11 alter column ename type
12 varchar(20);
13 SELECT * FROM s9913employee
Data output
Messages
Notifications
empno numeric (10)
ename character varying (20)
designation character varying (10)
salary_number numeric (8,2)

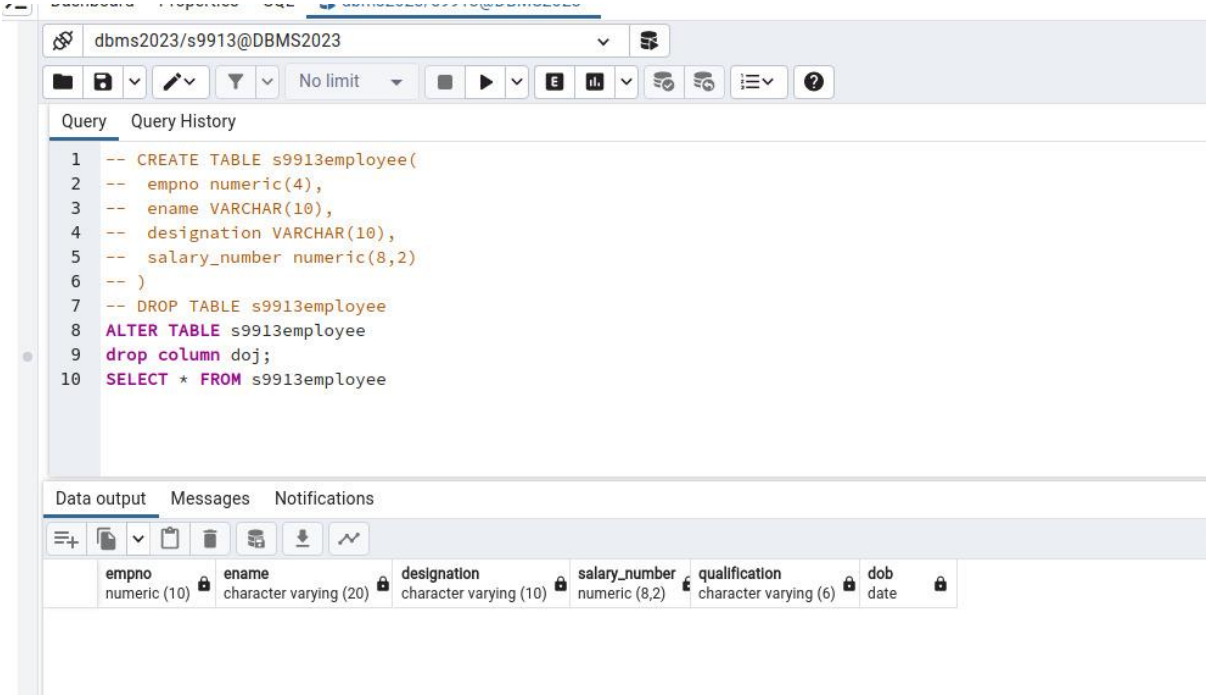
4


dbms2023/s9913@DBMS2023
No limit
Query
Query History
1 -- CREATE TABLE s9913employee(
2 -- empno numeric(4),
3 -- ename VARCHAR(10),
4 -- designation VARCHAR(10),
5 -- salary_number numeric(8,2)
6 --)
7 -- DROP TABLE s9913employee
8 ALTER TABLE s9913employee
9 add column qualification varchar(6);
10 SELECT * FROM s9913employee
Data output
Messages
Notifications
empno numeric (10)
ename character varying (20)
designation character varying (10)
salary_number numeric (8,2)
qualification character varying (6)

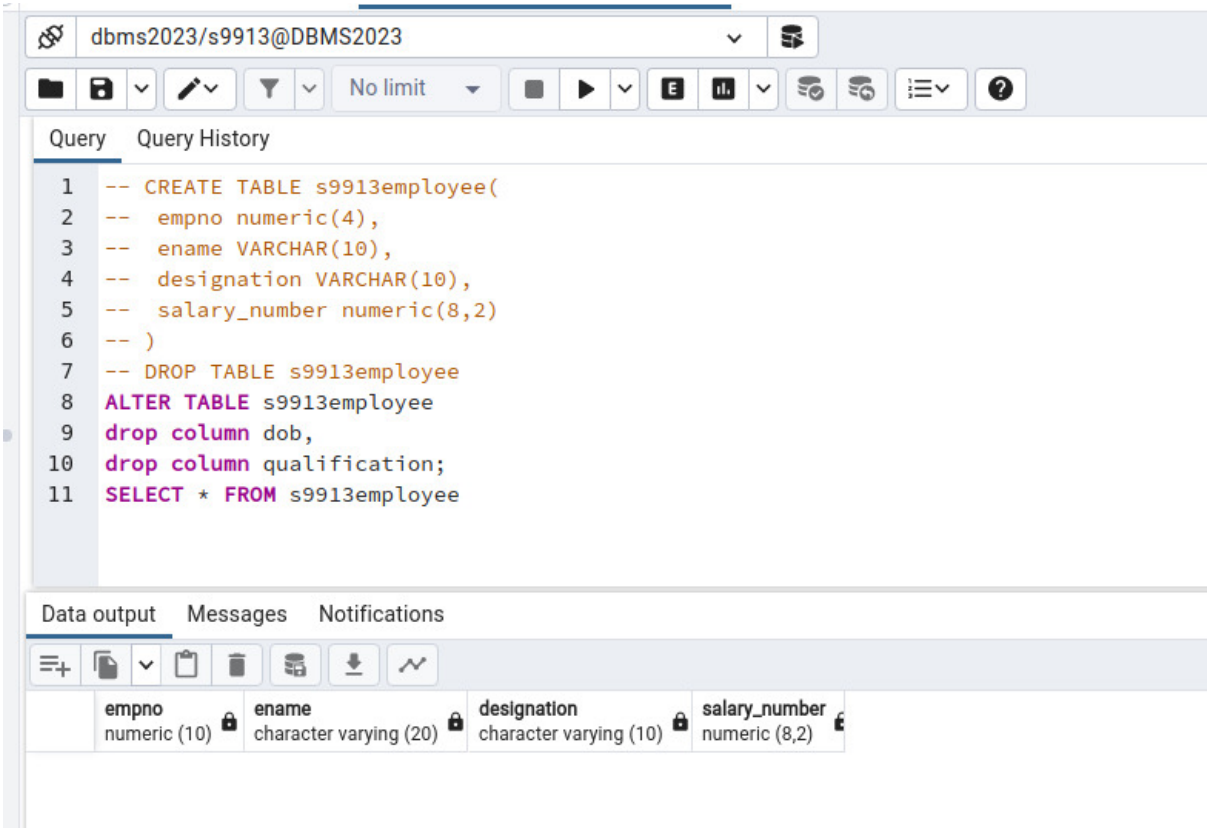
5


dbms2023/s9913@DBMS2023
No limit
Query
Query History
1 -- CREATE TABLE s9913employee(
2 -- empno numeric(4),
3 -- ename VARCHAR(10),
4 -- designation VARCHAR(10),
5 -- salary_number numeric(8,2)
6 --)
7 -- DROP TABLE s9913employee
8 ALTER TABLE s9913employee
9 add column dob date,
10 add column doj date;
11 SELECT * FROM s9913employee
Data output
Messages
Notifications
empno numeric (10)
ename character varying (20)
designation character varying (10)
salary_number numeric (8,2)
qualification character varying (6)
dob date
doj date

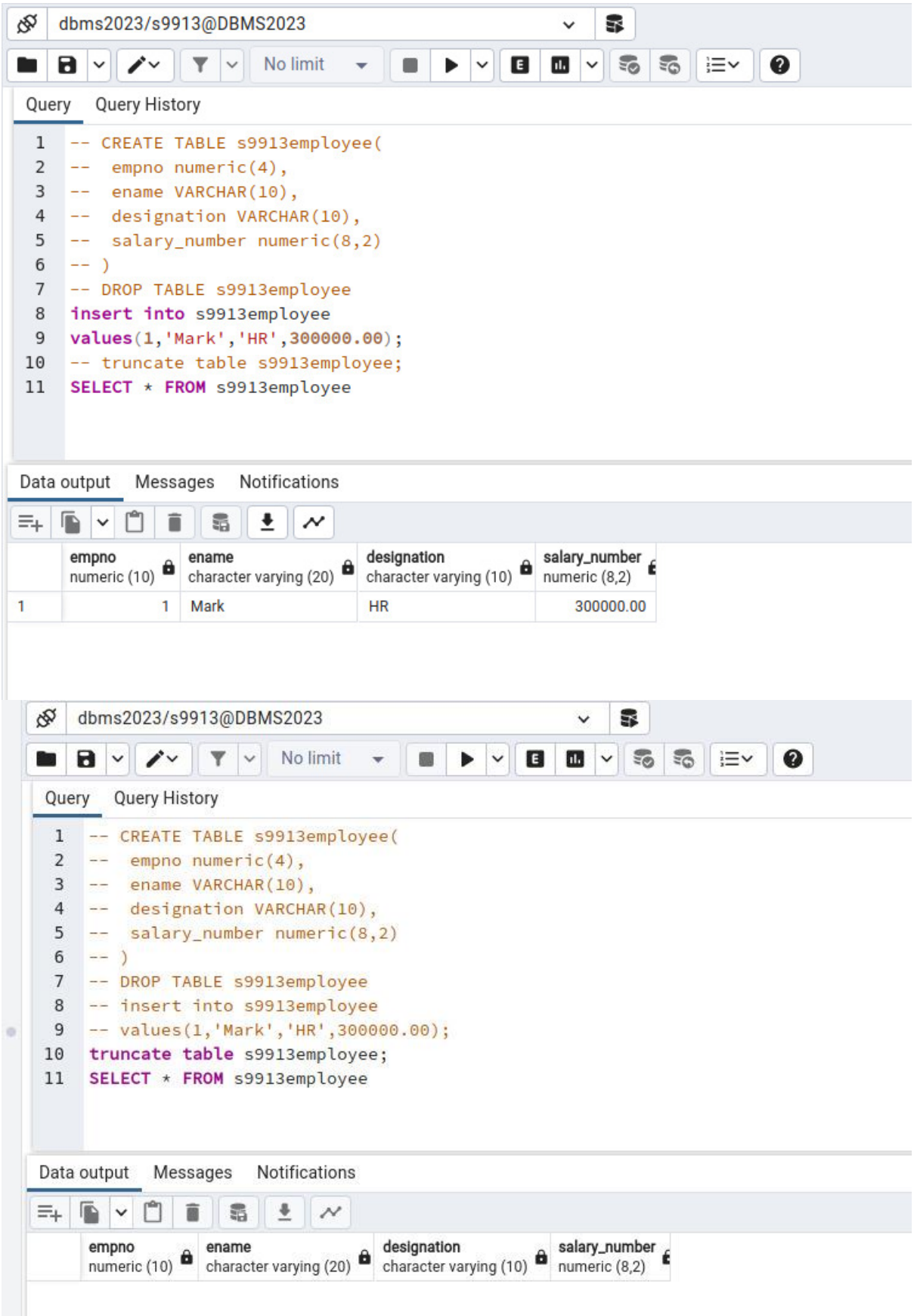
6


dbms2023/s9913@DBMS2023
No limit
Query
Query History
1 -- CREATE TABLE s9913employee(
2 -- empno numeric(4),
3 -- ename VARCHAR(10),
4 -- designation VARCHAR(10),
5 -- salary_number numeric(8,2)
6 --)
7 -- DROP TABLE s9913employee
8 ALTER TABLE s9913employee
9 drop column doj;
10 SELECT * FROM s9913employee
Data output
Messages
Notifications
empno numeric (10)
ename character varying (20)
designation character varying (10)
salary_number numeric (8,2)
qualification character varying (6)
dob date

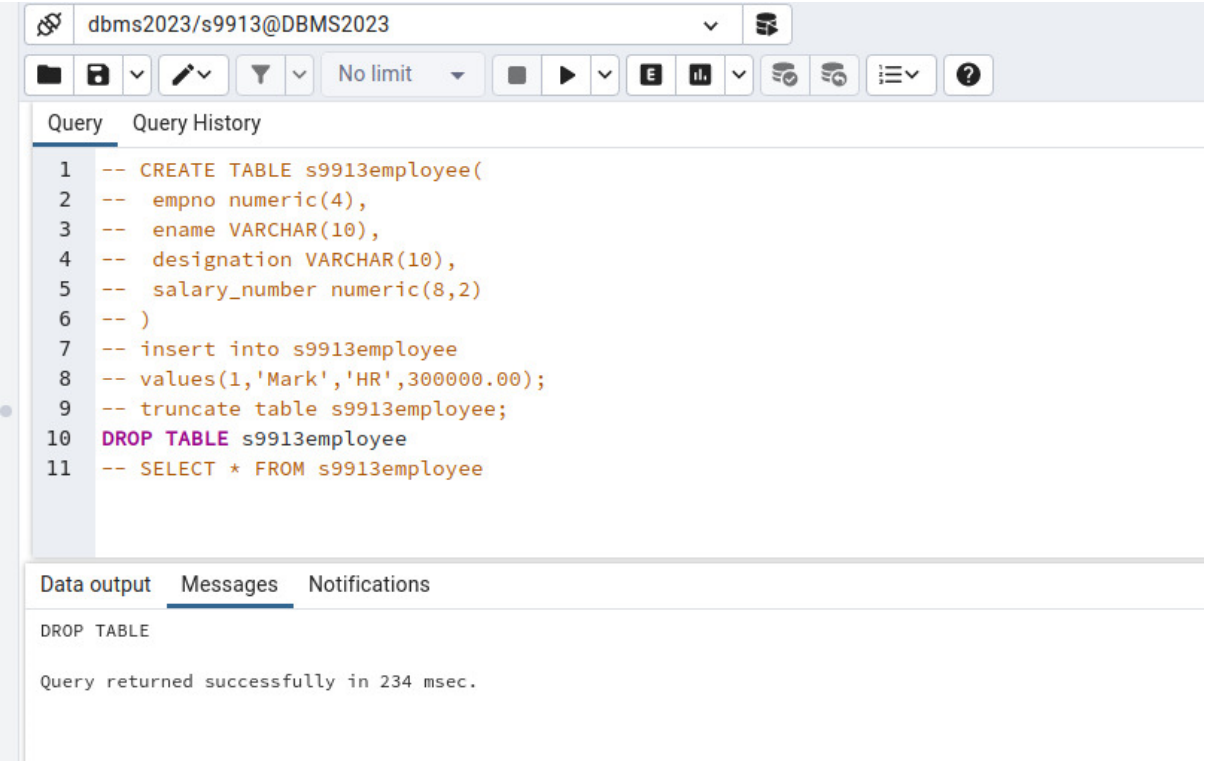
7


dbms2023/s9913@DBMS2023
No limit
Query
Query History
1 -- CREATE TABLE s9913employee(
2 -- empno numeric(4),
3 -- ename VARCHAR(10),
4 -- designation VARCHAR(10),
5 -- salary_number numeric(8,2)
6 --)
7 -- DROP TABLE s9913employee
8 ALTER TABLE s9913employee
9 drop column dob,
10 drop column qualification;
11 SELECT * FROM s9913employee
Data output
Messages
Notifications
empno numeric (10)
ename character varying (20)
designation character varying (10)
salary_number numeric (8,2)

8


dbms2023/s9913@DBMS2023
No limit
Query
Query History
1 -- CREATE TABLE s9913employee(
2 -- empno numeric(4),
3 -- ename VARCHAR(10),
4 -- designation VARCHAR(10),
5 -- salary_number numeric(8,2)
6 --)
7 -- DROP TABLE s9913employee
8 insert into s9913employee
9 values(1,'Mark','HR',300000.00);
10 -- truncate table s9913employee;
11 SELECT * FROM s9913employee
Data output
Messages
Notifications
empno numeric (10)
ename character varying (20)
designation character varying (10)
salary_number numeric (8,2)
1 1 Mark HR 300000.00
dbms2023/s9913@DBMS2023
No limit
Query
Query History
1 -- CREATE TABLE s9913employee(
2 -- empno numeric(4),
3 -- ename VARCHAR(10),
4 -- designation VARCHAR(10),
5 -- salary_number numeric(8,2)
6 --)
7 -- DROP TABLE s9913employee
8 -- insert into s9913employee
9 -- values(1,'Mark','HR',300000.00);
10 truncate table s9913employee;
11 SELECT * FROM s9913employee
Data output
Messages
Notifications
empno numeric (10)
ename character varying (20)
designation character varying (10)
salary_number numeric (8,2)

9



POSTLAB:-

Q1

A data dictionary is a centralized repository that stores metadata about a database, including definitions, data types, constraints, relationships, and other details. It serves as a reference guide for understanding and managing the structure and attributes of data within the database.

Q2

In a database, a schema is a logical container or namespace that holds a collection of database objects, including tables, views, indexes, and procedures. It provides a way to organize and manage database objects, allowing multiple users or applications to work independently within their designated schemas. Schemas help avoid naming conflicts and provide a structure for organizing and securing database elements.

Q3

Numeric Types:

INT, INTEGER: Integer.
SMALLINT: Small integer.
TINYINT: Very small integer.
BIGINT: Large integer.
DECIMAL(p, s), NUMERIC(p, s): Decimal number with a specified precision (p) and scale (s).
FLOAT: Floating-point number.
REAL: Real number.

Character/String Types:

CHAR(n): Fixed-length character string.
VARCHAR(n), VARCHAR(MAX): Variable-length character string with a maximum length of n characters or maximum allowed length.
TEXT: Variable-length character string with no specified maximum length.

Date and Time Types:

DATE: Date (year, month, day).
TIME: Time of day.
DATETIME, TIMESTAMP: Date and time.
INTERVAL: Time interval.

Boolean Type:

BOOLEAN, BOOL: Boolean

SEcomputer A batch-C	Roll number : 9913
Experiment no. : 3 part2	Date of Implementation :
Aim : To implement data manipulation language (DML) commands	
Tool Used : PostgreSQL	
Related Course outcome : Students should be able to Write queries in SQL to retrieve any type of information from a data base.	

Rubrics for assessment of Experiment:

Indicator	Poor	Average	Good
Timeliness Maintains Experiment deadline (3)	Experiment not done (0)	One or More than One week late (1-2)	Maintains deadline (3)
Completeness and neatness Complete all parts of Experiment(3)	N/A	< 80% complete (1-2)	100% complete (3)
Originality Extent of plagiarism(2)	Copied it from someone else(0)	At least try to implement but could not succeed (1)	Implemented (2)
Knowledge In depth knowledge of the Experiment(2)	Unable to answer any questions(0)	Unable to answer few questions (1)	Able to answer all questions (2)

Assessment Marks :

Timeliness	
Completeness and neatness	
Originality	
Knowledge	
Total	

Total : (Out of 10)

Teacher's Sign :	
EXPERIMENT 3	**DDL and DML Commands**
Aim	To implement DDL with integrity constraints and DML – Data manipulation language command
Tools	PostgreSQL/MySql
Theory	Data Definition Language-1) Create 2) Alter 3) Drop 4) Rename 5) Truncate • **CREATE** – is used to create the database or its objects (like table, index, function, views, store procedure and triggers). • **DROP** – is used to delete objects from the database. • **ALTER**-is used to alter the structure of the database. • **TRUNCATE**–is used to remove all records from a table, including all spaces allocated for the records are removed. • **COMMENT** –is used to add comments to the data dictionary. • **RENAME** –is used to rename an object existing in the database. 1) Create table create table tablename (column1 data type, column2 data type, column3 data type, ... columnN data type); 2) **DROP object object_name** Examples: DROP TABLE table_name; table_name: Name of the table to be deleted. DROP DATABASE database_name; database_name: Name of the database to be deleted.

3) TRUNCATE

TRUNCATE statement is a Data Definition Language (DDL) operation that is used to mark the extents of a table for deallocation (empty for reuse). The result of this operation quickly removes all data from a table, typically bypassing a number of integrity enforcing mechanisms. It was officially introduced in the standard.

The TRUNCATE TABLE mytable statement is logically (though not physically) equivalent to the DELETE FROM mytable statement (without a WHERE clause).
Syntax:
TRUNCATE TABLE table_name;
table_name: Name of the table to be truncated.
DATABASE name - student_data

- **cannot** be rolled back, so it must be used wisely.

DROP vs TRUNCATE

- Truncate is normally ultra-fast and its ideal for deleting data from a temporary table.
- Truncate preserves the structure of the table for future use, unlike drop table where the table is deleted with its full structure.

Table or Database deletion using DROP statement

- To delete the whole database
  DROP DATABASE student_data;

After running the above query whole database will be deleted.

- To truncate Student_details table from student_data database.
  TRUNCATE TABLE Student_details;

After running the above query Student_details table will be truncated, i.e, the data will be deleted but the structure will remain in the memory for further operations.

Alter

alter command is used for altering the table structure, such as,

- to add a column to existing table
- to rename any existing column
- to change data type of any column or to modify its size.
- to drop a column from the table.

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

Procedure	**B)Data Manipulation Language** A Data Manipulation Language enables programmers and users of the database to retrieve insert, delete and update data in a database. e.g. INSERT, UPDATE, DELETE, SELECT. **INSERT:** INSERT statement adds one or more records to any single table in a relational database. INSERT INTO tablename VALUES (expr1,expr2……..); **UPDATE:** UPDATE statement that changes the data of one or more records in a table. Either all the rows can be updated, or a subset may be chosen using a condition. UPDATE table_name SET column_name = value [, column_name = value ...] [WHERE condition] **DELETE:** DELETE statement removes one or more records from a table. A subset may be defined for deletion using a condition, otherwise all records are removed. DELETE FROM tablename WHERE condition

Task1: 1. Create following tables:

Table name : client_master

Column Name	Data type	Size	
Client_no	varchar	6	Primary key
Name	varchar	20	Not null
Address	varchar	30	
City	varchar	15	
Pincode	numeric	8	
State	varchar	15	
Bal_due	numeric	10,2	>0

Table name: Product_master

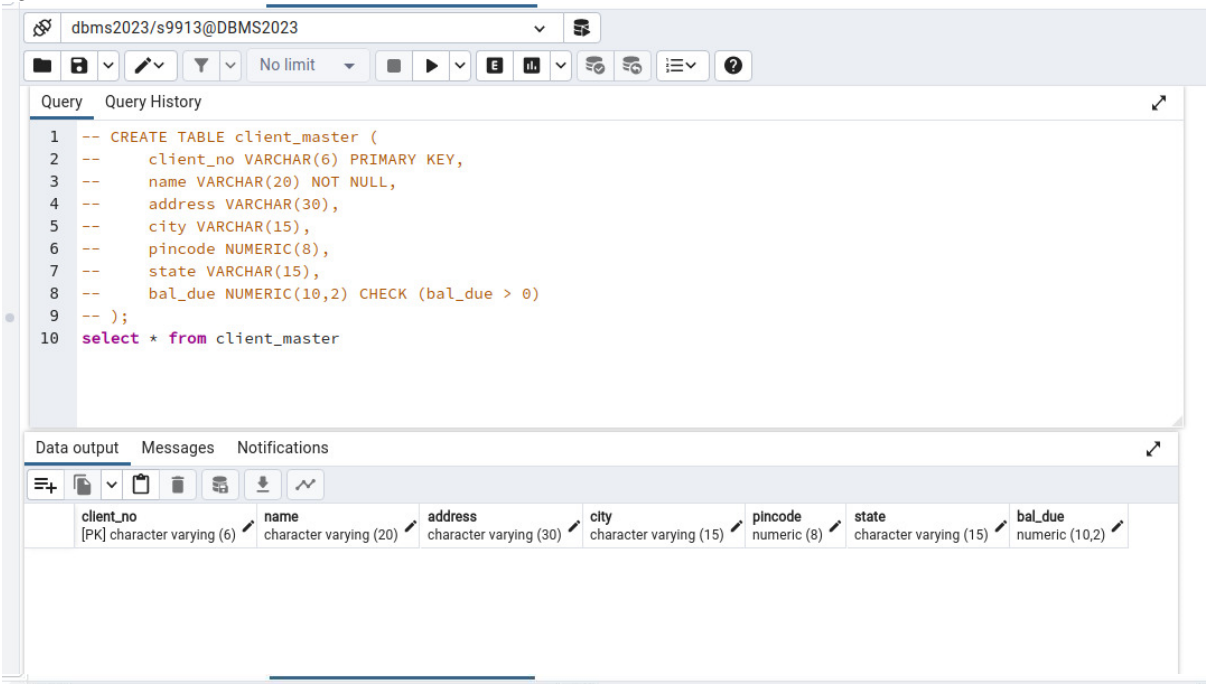
Column Name	Data type	Size	
product_no	varchar	6	Primary key
description	varchar	15	Not null
Profit_percent	numeric	4,2	
Unit_measure	varchar	10	
Qty_on_hand	numeric	8	>0
Reorder_level	numeric	8	
Sell_price	numeric	8,2	
Cost_price	numeric	8,2	

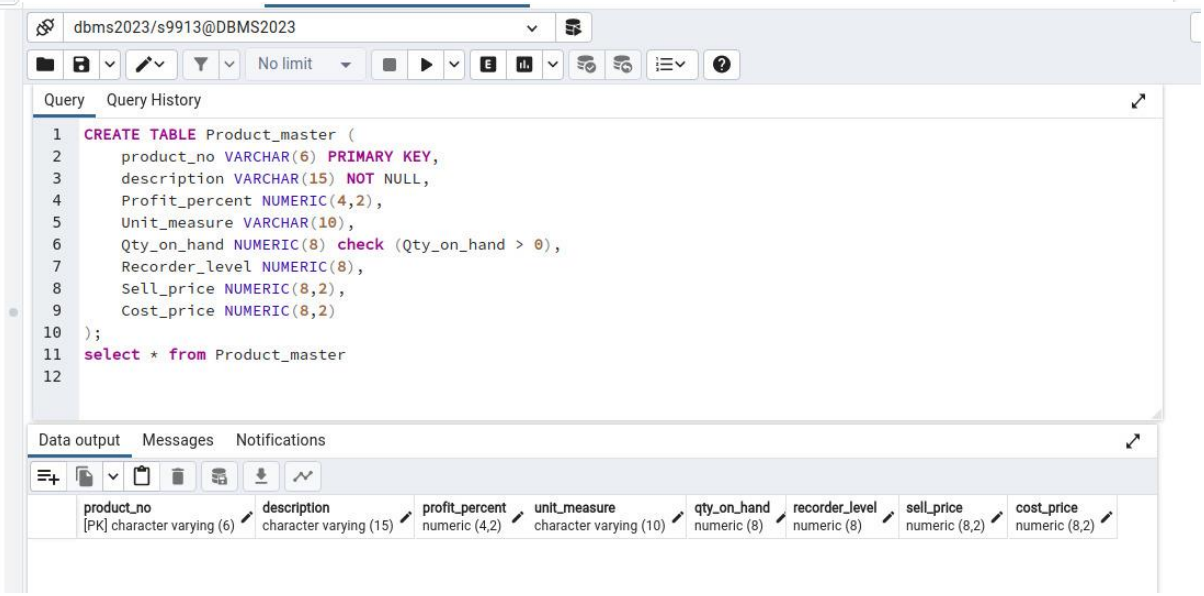
2. Insert 5-6 records in each table.
3. Find out the names of all clients
4. Retrieve the entire contents of the client_master table.
5. Retrieve the list of names and cities of all the clients
6. List the various products available from the product_master table
7. List all the clients who are located in mumbai.
8. Change the city of client_no C001 to mumbai
9. Change the bal_due of client_no C005 to Rs. 1000
10. Change the cost price of 'hard disk' to Rs. 3000
11. Delete all the products from product_master where the qty_on_hand is less than 100
12. Delete from client_master where the column state holds the value 'Tamil Nadu'

Task2: Create the tables for EER diagram of EXPT. no 2

Post Lab Questions:	1. Explain different data types of Mysql/postgresql 2. Perform delete and truncate in lab and Differentiate delete and truncate

Q1





Q2

PostgreSQL

```
1 INSERT INTO client_master
2 VALUES
3 ('C001', 'John', '123 Main', 'Mumbai', 400091, 'Maharashtra', 20000.00),
4 ('C002', 'Doe', '456 Elm', 'Kolkata', 300023, 'West Bengal', 50000.00),
5 ('C003', 'Alfred', '789 Oak', 'Pune', 720001, 'Maharashtra', 2000.00),
6 ('C004', 'James', '321 Map', 'Chennai', 498880, 'Tamil Nadu', 100000.00),
7 ('C005', 'Dalton', '567 Pine', 'Mumbai', 560000, 'Karnataka', 70000.00);
8
9 SELECT * FROM client_master
```

client_master - TABLE

client_no	name	address	city	pincode	state	bal_due
C001	John	123 Main	Mumbai	400091	Maharashtra	20000.00
C002	Doe	456 Elm	Kolkata	300023	West Bengal	50000.00
C003	Alfred	789 Oak	Pune	720001	Maharashtra	2000.00
C004	James	321 Map	Chennai	498880	Tamil Nadu	100000.00
C005	Dalton	567 Pine	Mumbai	560000	Karnataka	70000.00

```
1 INSERT INTO product_master
2 VALUES
3 ('P001', 'Laptop', 20.00, '2kg', 200, 2, 20000.00, 15000.00),
4 ('P002', 'Hard disk', 10.00, '500g', 80, 2, 1500.00, 500.00),
5 ('P003', 'Processor', 70.00, '3kg', 150, 2, 70000.00, 3500.00),
6 ('P004', 'Keypad', 10.00, '500g', 70, 2, 2000.00, 100.00),
7 ('P005', 'Printer', 30.00, '1.5kg', 300, 2, 10000.00, 1500.00);
8
9 SELECT * FROM product_master
```

product_...	description	profit_perc...	unit_measure	qty_on_hand	recorder_le...	sell_price	cost_price
P001	Laptop	20.00	2kg	200	2	20000.00	15000.00
P002	Hard disk	10.00	500g	80	2	1500.00	500.00
P003	Processor	70.00	3kg	150	2	70000.00	3500.00
P004	Keypad	10.00	500g	70	2	2000.00	100.00
P005	Printer	30.00	1.5kg	300	2	10000.00	1500.00

PostgreSQL

```
1 SELECT NAME FROM client_master
```

name
John
Doe
Alfred
James
Dalton

4

PostgreSQL

```
1 INSERT INTO client_master
2 VALUES
3 ('C001', 'John', '123 Main', 'Mumbai', 400091, 'Maharashtra', 20000.00),
4 ('C002', 'Doe', '456 Elm', 'Kolkata', 300023, 'West Bengal', 50000.00),
5 ('C003', 'Alfred', '789 Oak', 'Pune', 720001, 'Maharashtra', 2000.00),
6 ('C004', 'James', '321 Map', 'Chennai', 498880, 'Tamil Nadu', 100000.00),
7 ('C005', 'Dalton', '567 Pine', 'Mumbai', 560000, 'Karnataka', 70000.00);
8
9 SELECT * FROM client_master
```

client_master - TABLE

client_no	name	address	city	pincode	state	bal_due
C001	John	123 Main	Mumbai	400091	Maharashtra	20000.00
C002	Doe	456 Elm	Kolkata	300023	West Bengal	50000.00
C003	Alfred	789 Oak	Pune	720001	Maharashtra	2000.00
C004	James	321 Map	Chennai	498880	Tamil Nadu	100000.00
C005	Dalton	567 Pine	Mumbai	560000	Karnataka	70000.00

5

PostgreSQL

```
1 SELECT NAME, city FROM client_master
```

name	city
John	Mumbai
Doe	Kolkata
Alfred	Pune
James	Chennai
Dalton	Mumbai

6

PostgreSQL

```
1 SELECT description FROM product_master
```

description
Laptop
Hard disk
Processor
Keypad
Printer

7

PostgreSQL

```
SELECT * FROM client_master WHERE city = 'Mumbai';
```

client_no	name	address	city	pincode	state	bal_due
C001	John	123 Main	Mumbai	400091	Maharashtra	20000.00
C005	Dalton	567 Pine	Mumbai	560000	Karnataka	70000.00

8

```
UPDATE client_master SET city = 'Mumbai' WHERE client_no = 'C001';
```

History

Syntax | History

PostgreSQL

```
UPDATE client_master SET city = 'Mumbai
```

20:54:09

9

```
UPDATE client_master SET bal_due = 1000 WHERE client_no = 'C005';
SELECT * FROM client_master
```

client_no	name	address	city	pincode	state	bal_due
C002	Doe	456 Elm	Kolkata	300023	West Bengal	50000.00
C003	Alfred	789 Oak	Pune	720001	Maharashtra	2000.00
C004	James	321 Map	Chennai	498880	Tamil Nadu	100000.00
C001	John	123 Main	Mumbai	400091	Maharashtra	20000.00
C005	Dalton	567 Pine	Mumbai	560000	Karnataka	1000.00

10

```
1 UPDATE product_master SET Cost_price = 3000 WHERE description = 'Hard disk';
2 SELECT * FROM product_master
```

product_...	description	profit_perc...	unit_measure	qty_on_hand	recorder_le...	sell_price	cost_price
P001	Laptop	20.00	2kg	200	2	20000.00	15000.00
P003	Processor	70.00	3kg	150	2	70000.00	3500.00
P004	Keypad	10.00	500g	70	2	2000.00	100.00
P005	Printer	30.00	1.5kg	300	2	10000.00	1500.00
P002	Hard disk	10.00	500g	80	2	1500.00	3000.00

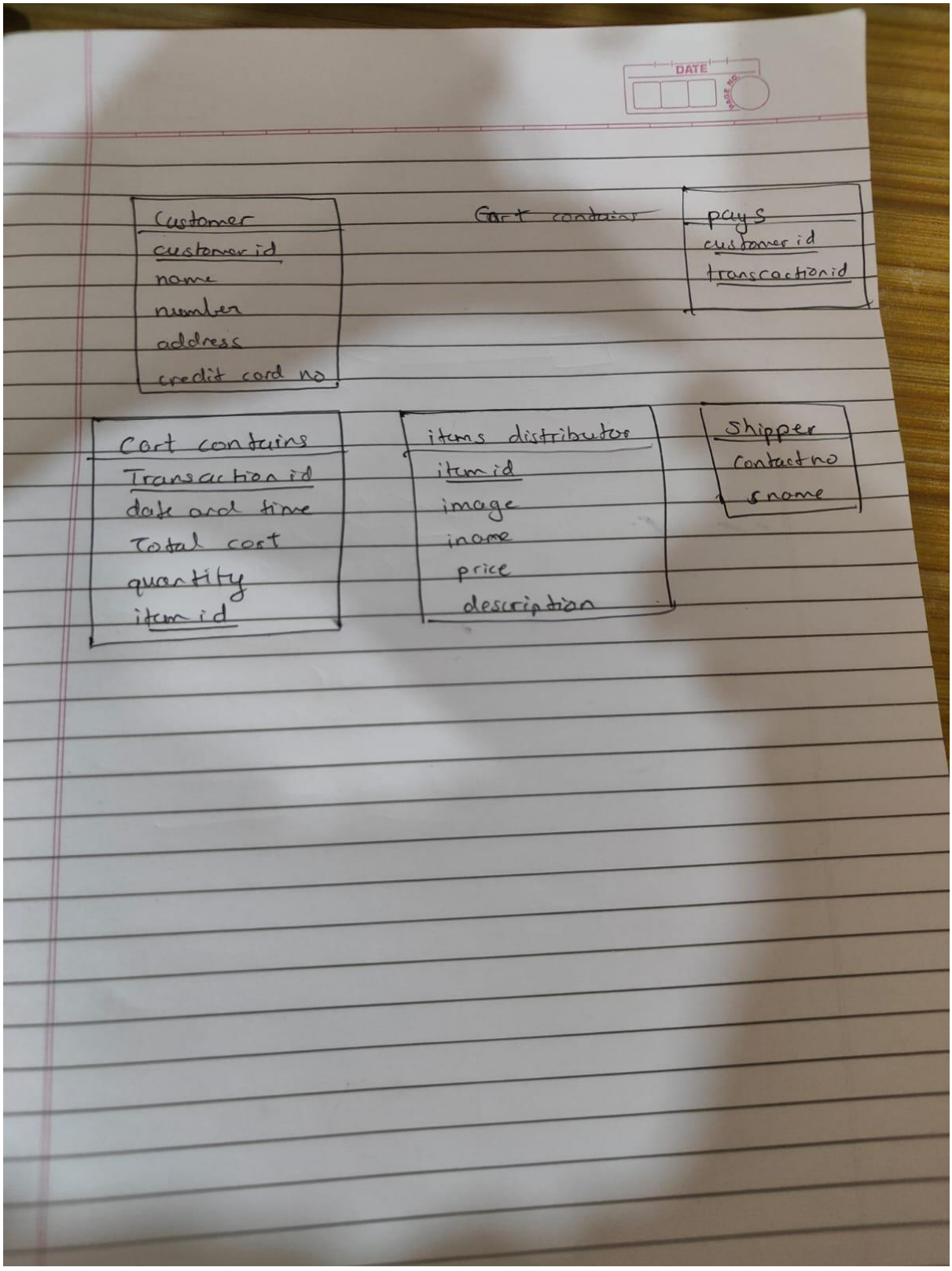
11

```
1 DELETE FROM product_master WHERE qty_on_hand < 100;
2 SELECT * FROM product_master
```

product_...	description	profit_perc...	unit_measure	qty_on_hand	recorder_le...	sell_price	cost_price
P001	Laptop	20.00	2kg	200	2	20000.00	15000.00
P003	Processor	70.00	3kg	150	2	70000.00	3500.00
P005	Printer	30.00	1.5kg	300	2	10000.00	1500.00

12

client_no	name	address	city	pincode	state	bal_due
C002	Doe	456 Elm	Kolkata	300023	West Bengal	50000.00
C003	Alfred	789 Oak	Pune	720001	Maharashtra	2000.00
C001	John	123 Main	Mumbai	400091	Maharashtra	20000.00
C005	Dalton	567 Pine	Mumbai	560000	Karnataka	1000.00


Customer
customer id
name
number
address
credit card no
Cart contains
pays
customer id
transcactionid
Cart contains
Transaction id
date and time
Total cost
quantity
item id
items distributor
item id
image
iname
price
description
Shipper
Contact no
sname

1. Integer Types:
    MySQL:
        TINYINT, SMALLINT, MEDIUMINT, INT, BIGINT
    PostgreSQL:
        SMALLINT, INTEGER, BIGINT
2. Decimal/Floating-Point Types:
    MySQL:
        DECIMAL, FLOAT, DOUBLE
    PostgreSQL:
        DECIMAL, NUMERIC, REAL, DOUBLE PRECISION
3. String/Character Types:
    MySQL:
        CHAR, VARCHAR, TEXT
    PostgreSQL:
        CHAR, VARCHAR, TEXT
4. Date and Time Types:
    MySQL:
        DATE, TIME, DATETIME, TIMESTAMP
    PostgreSQL:
        DATE, TIME, TIMESTAMP, INTERVAL
5. Boolean Type:
    MySQL:
        BOOLEAN
    PostgreSQL:
        BOOLEAN

DELETE:
    The DELETE statement is used to remove specific rows from a table based on a condition specified in the WHERE clause.
    It allows more flexibility as you can delete specific rows that meet certain criteria.
TRUNCATE:
    The TRUNCATE statement is used to remove all rows from a table.
    It removes all rows without considering any conditions. It effectively deletes all data from the table.

SE-COMP A Batch-C **Roll number : 9913**

Experiment no. : 4 (part3) **Date of Implementation : 12/2/23**

Aim : To implement Integrity constraints

Tool Used : PostgreSQL

Related Course outcome : At the end of the course, Students will be able to Use

SQL : Standard language of relational database

Rubrics for assessment of Experiment:

Indicator	Poor	Average	Good
Timeliness ● Maintains assignment deadline (3)	Assignment not done (0)	One or More than One week late (1-2)	Maintains deadline (3)
Completeness and neatness ● Complete all parts of QUERY assignment(3)	N/A	< 80% complete (1-2)	100% complete (3)
Originality ● Extent of plagiarism(2)	Copied it from someone else(0)	At least few questions have been done without copying(1)	Assignment has been solved completely without copying (2)
Knowledge ● In depth knowledge of the QUERY assignment(2)	Unable to answer 2 questions(0)	Unable to answer 1 question (1)	Able to answer 2 questions (2)

Assessment Marks :

Timeliness(3)	
Completeness and neatness(3)	
Originality (2)	
Knowledge (2)	
Total (Out of 10)	

Remark:

Teacher's Sign :

Experiment No. 4- Integrity Constraints

AIM:

- To implement database for relational model using DDL statement
- Apply Integrity Constraints for the specified system

Objective of the Experiment:

After completing this experiment you will be able to:

Create database.

Create table with constraints

Modify the schema of the table.

Theory :

Pre Lab/ Prior Concepts:

The Data Definition Language (DDL) is used to create and modify the relational schema. Also it is used to add various constraints to the table like the primary key, foreign key, check constraint, not null constraint and unique constraint.

The DDL statements are:
CREATE
DROP
ALTER

SQL supports the standard int, smallint, real, double precision, char(N), varchar(N), date, time, timestamp, and interval for creating tables.

Procedure / Algorithm:

Create Database and use it:

$ createdb mydb

$ psql mydb

Delete a database:

$ dropdb mydb

Create table:

CREATE TABLE my_first_table
(first_column text,
second_column integer
);

CREATE TABLE products (
product_no integer, name
text, price numeric);

Drop Table:

DROP TABLE my_first_table;
DROP TABLE products;

Default Value:

CREATE TABLE products (
product_no integer, name text,
price numeric **DEFAULT 9.99**);

Constraints:

1. Primary Key CREATE TABLE
products (product_no integer
PRIMARY KEY, name text,
price numeric);

Primary keys can also constrain more than one column.

CREATE TABLE example (

a integer,

b integer,

c integer,

PRIMARY KEY (a, c)

);

2. Check Constraint

CREATE TABLE products (

product_no integer, name text, price

numeric **CHECK (price > 0)**);

3. Not Null Constraint

CREATE TABLE products (

product_no integer **NOT NULL**,

name text **NOT NULL**, price

numeric);

4. Unique Constraint

CREATE TABLE products (

product_no integer **UNIQUE**,

name text, price numeric);

5. Foreign Key Constarint CREATE TABLE products (product_no integer PRIMARY KEY,

name text, price numeric);

CREATE TABLE orders (

order_id integer PRIMARY KEY,

product_no integer **REFERENCES products (product_no)**,

quantity integer

);

Here a foreign key constraint in the order table references the products table.

Modifying table:

Adding column

ALTER TABLE products ADD COLUMN description text;

Removing column

ALTER TABLE products DROP COLUMN description;

Adding Constraint

ALTER TABLE products ADD CONSTRAINT some_name UNIQUE (product_no); ALTER TABLE products ADD FOREIGN KEY (product_group_id) REFERENCES product_groups;

Removing Constraint

ALTER TABLE products DROP CONSTRAINT some_name;

Adding Not Null Constraint

ALTER TABLE products ALTER COLUMN product_no SET NOT NULL;

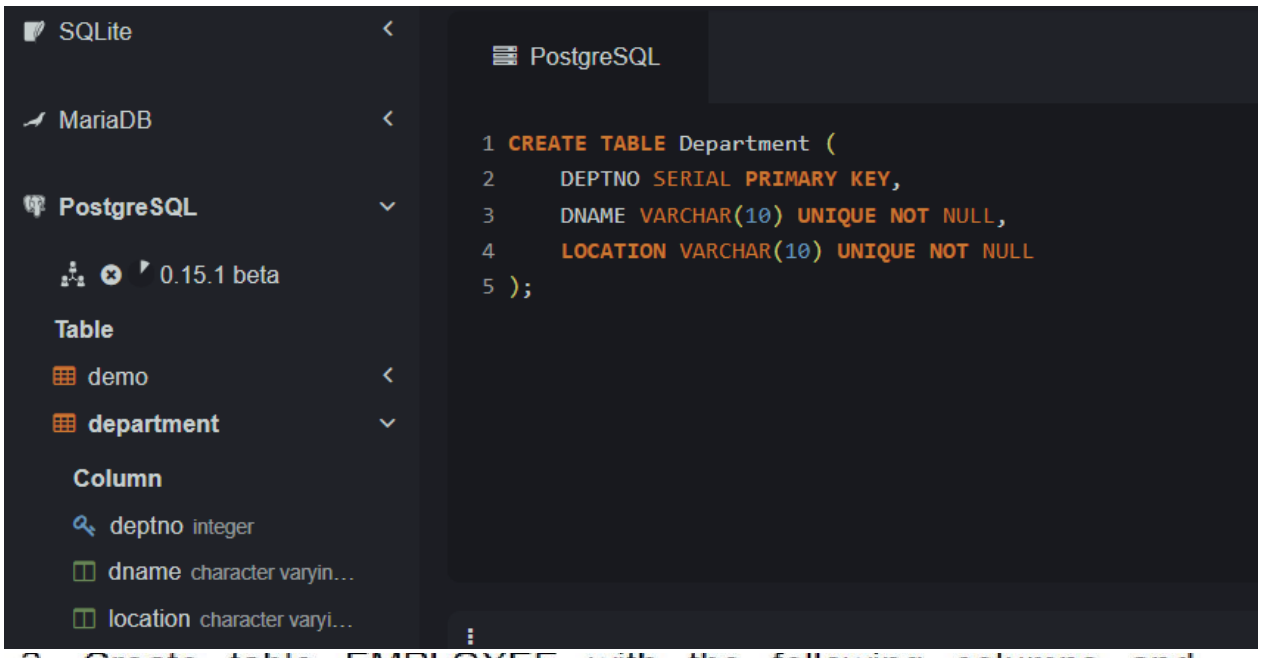
Removing Not Null Constraint

ALTER TABLE products ALTER COLUMN product_no DROP NOT NULL;

Task1: Exercise –

1. Create table DEPT with the following columns and constraints

Column name	Data type	Size	Constraint
DEPTNO	NUMBER	2	PRIMARY KEY
DNAME	VARCHAR2	10	UNIQUE + NOT NULL
LOCATION	VARCHAR2	10	UNIQUE + NOT NULL

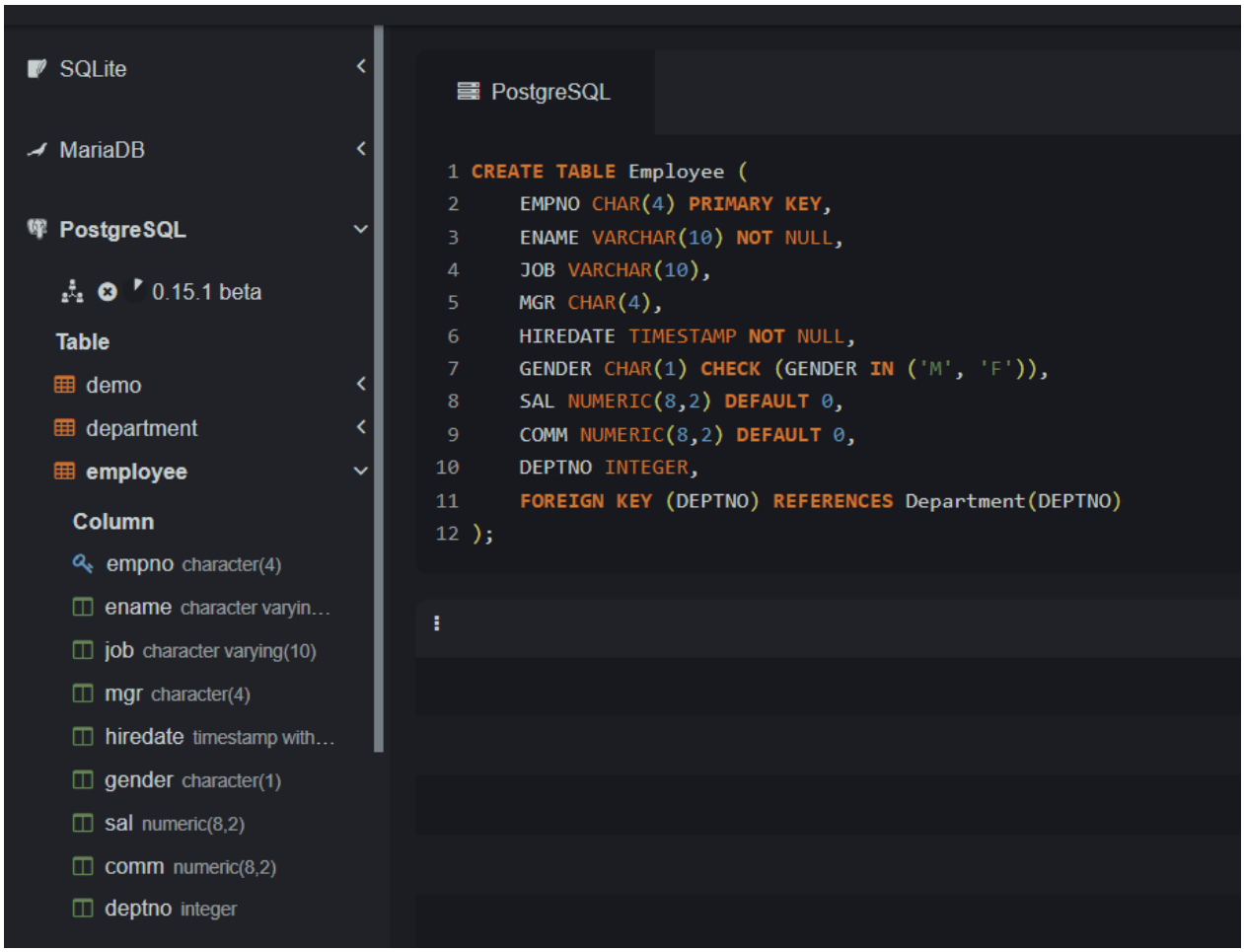


2. Create table EMPLOYEE with the following columns and constraints

Column name	Data type	Size	Constraint
EMPNO	CHAR	4	PRIMARY KEY
ENAME	VARCHAR2	10	NOT NULL
JOB	VARCHAR2	10	
MGR	CHAR	4	
HIREDATE	TIMESTAMP		NOT NULL
GENDER	CHAR	1	'M' OR 'F' ONLY
SAL	NUMBER	8,2	DEFAULT 0
COMM	NUMBER	8,2	DEFAULT 0
DEPTNO	NUMBER	2	FOREIGN KEY REFERRING TO DEPTNO of DEPT table

3. Insert 5 records in both the tables.

Q.2)



Q.3)

```sql
-- Inserting records into Department table
INSERT INTO Department (DEPTNO, DNAME, LOCATION) VALUES
(1, 'HR', 'New York'),
(2, 'IT', 'San Fran'),
(3, 'Finance', 'Chicago'),
(4, 'Marketing', 'LA'),
(5, 'Ops', 'Seattle');

-- Inserting records into Employee table
INSERT INTO Employee (EMPNO, ENAME, JOB, MGR, HIREDATE, GENDER, SAL, COMM, DEPTNO) VALUES
('E001', 'John', 'Manager', NULL, '2022-01-01', 'M', 50000.00, 2000.00, 1),
('E002', 'Jane', 'Developer', 'E001', '2022-02-15', 'F', 45000.00, 1500.00, 2),
('E003', 'Bob', 'Analyst', 'E001', '2022-03-10', 'M', 40000.00, 1000.00, 2),
('E004', 'Alice', 'Assistant', 'E002', '2022-04-20', 'F', 35000.00, 800.00, 3),
('E005', 'Mike', 'Clerk', 'E003', '2022-05-05', 'M', 30000.00, 500.00, 3);
```

\

empno	ename	job	mgr	hiredate	gender	sal	comm	deptno
E001	John	Manager	NULL	2022-01-0…	M	50000.00	2000.00	1
E002	Jane	Developer	E001	2022-02-1…	F	45000.00	1500.00	2
E003	Bob	Analyst	E001	2022-03-1…	M	40000.00	1000.00	2
E004	Alice	Assistant	E002	2022-04-2…	F	35000.00	800.00	3
E005	Mike	Clerk	E003	2022-05-0…	M	30000.00	500.00	3

deptno	dname	location
1	HR	New York
2	IT	San Fran
3	Finance	Chicago
4	Marketing	LA
5	Ops	Seattle

4. Add table level constraint such that commission cannot be greater than 30% of salary after the table has been created. Assign the constraint name COMM_30_SAL.

5. Add new constraint with the name DEPT_CHK_LOCATION to DEPT table such that LOCATION can be any one of the following cities MUMBAI, PUNE, BENGALURU, LONDON, SAN FRANSISCO only.

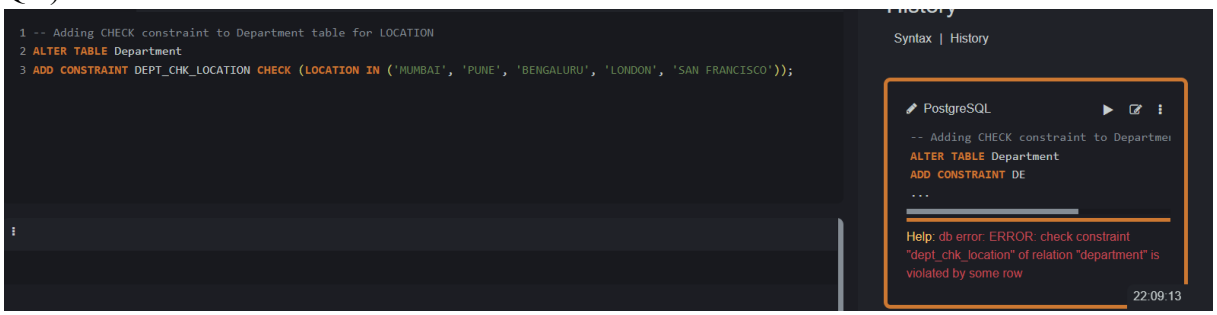
6. Remove the UNIQUE constraint from the LOCATION column.

Q.4)

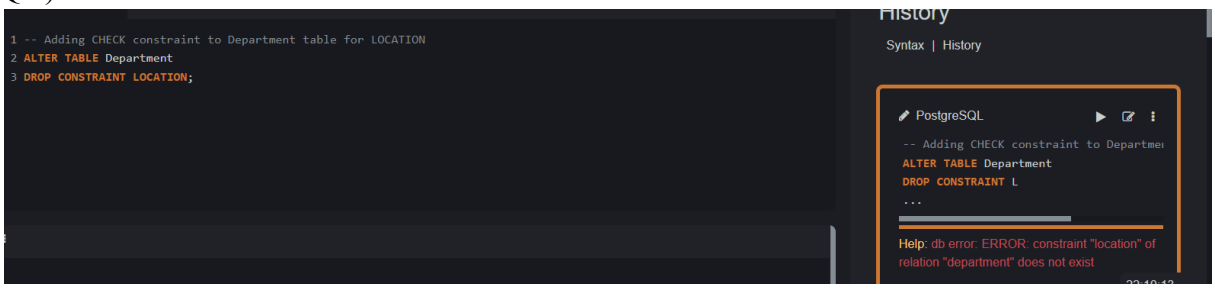
```
1 -- Adding table-level constraint to Employee table
2 ALTER TABLE Employee
3 ADD CONSTRAINT COMM_30_SAL CHECK (COMM <= SAL * 0.3);
4 SELECT*FROM employee
```

empno	ename	job	mgr	hiredate	gender	sal	comm	deptno
E001	John	Manager	NULL	2022-01-0…	M	50000.00	2000.00	1
E002	Jane	Developer	E001	2022-02-1…	F	45000.00	1500.00	2
E003	Bob	Analyst	E001	2022-03-1…	M	40000.00	1000.00	2
E004	Alice	Assistant	E002	2022-04-2…	F	35000.00	800.00	3
E005	Mike	Clerk	E003	2022-05-0…	M	30000.00	500.00	3

Q.5)



Q.6)



Conclusion:

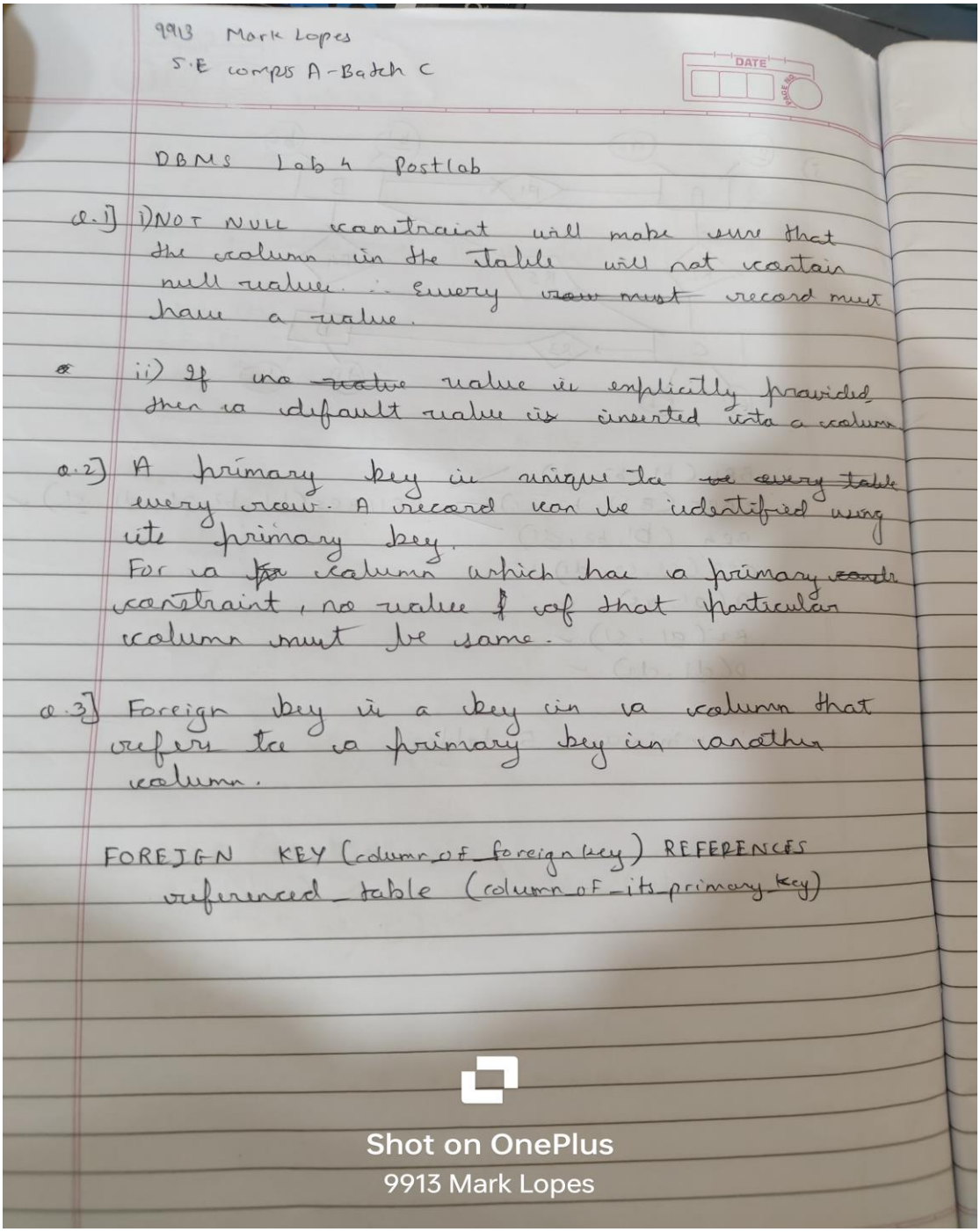
Thus using the schema diagram from the previous experiment, the tables were created using CREATE DDL statement with the primary key and foreign key constraint. Other constraints like Check, Unique and Not Null were added to the appropriate column by using ALTER DDL statement.

Post Lab Assignment:

1)What is NOT NULL constraint? What is DEFAULT constraint?

2) What is primary key? What is PRIMARY KEY constraint?

3) What is foreign key? How do you define a foreign key in your table?



9913 Mark Lopes
S.E comps A-Batch C

DATE

DBMS Lab 4 Postlab

Q.1] i) NOT NULL constraint will make sure that the column in the table will not contain null values. ∴ Every record must have a value.

ii) If no value is explicitly provided then a default value is inserted into a column

Q.2] A primary key is unique to every row. A record can be identified using its primary key.
For a column which has a primary constraint, no value of that particular column must be same.

Q.3] Foreign key is a key in a column that refers to a primary key in another column.

FOREIGN KEY (column_of_foreign_key) REFERENCES referenced_table (column_of_its_primary_key)

Shot on OnePlus
9913 Mark Lopes

Signature of Faculty	Date of Completion:

SE-COMP A BATCH-C	Roll number : 9913
Experiment no. : 5	Date of Implementation :27/2/2024
Aim : To implement simple SQL commands, string manipulation operations and aggregate functions.	
Tool Used : PostgreSQL/Mysql	
Related Course outcome : At the end of the course, Students will be able to Use SQL : Standard language of relational database	

Rubrics for assessment of Experiment:

Indicator	Poor	Average	Good
Timeliness ● Maintains assignment deadline (3)	Assignment not done (0)	One or More than One week late (1-2)	Maintains deadline (3)
Completeness and neatness ● Complete all parts of QUERY assignment(3)	N/A	< 80% complete (1-2)	100% complete (3)
Originality ● Extent of plagiarism(2)	Copied it from someone else(0)	At least few questions have been done without copying(1)	Assignment has been solved completely without copying (2)
Knowledge ● In depth knowledge of the QUERY assignment(2)	Unable to answer 2 questions(0)	Unable to answer 1 question (1)	Able to answer 2 questions (2)

Assessment Marks :

Timeliness	
Completeness and neatness	
Originality	
Knowledge	
Total	

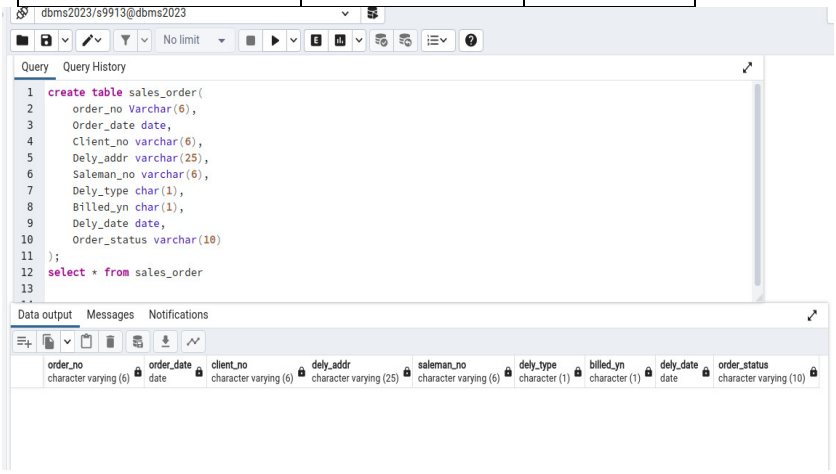
Total : (Out of 10)

Teacher's Sign :	
EXPERIMENT 5	Basic SQL Commands
Aim	To implement simple SQL commands, string manipulation operations and aggregate functions.
Tools	PostgreSQL
Theory	**SELECT:** SELECT statement returns a result set of records from one or more tables. The select statement has optional clauses: ● WHERE specifies which rows to retrieve ● GROUP BY groups rows sharing a property so that an aggregate function can be applied to each group having group. ● HAVING selects among the groups defined by the GROUP BY clause. ● ORDER BY specifies an order in which to return the rows. Syntax: SELECT<attribute list> FROM<table list> WHERE<condition> Where ● Attribute list is a list of attribute name whose values to be retrieved by the query. ● Table list is a list of table name required to process query. ● Condition is a Boolean expression that identifies the tuples to be retrieved by query. **SQL Aggregate Functions** SQL aggregate functions return a single value, calculated from values in a column. Useful aggregate functions: ● AVG() - Returns the average value ● COUNT() - Returns the number of rows ● FIRST() - Returns the first value ● LAST() - Returns the last value ● MAX() - Returns the largest value ● MIN() - Returns the smallest value ● SUM() - Returns the sum **The SQL ORDER BY Keyword** The ORDER BY keyword is used to sort the result-set by one or more columns. The ORDER BY keyword sorts the records in ascending order by default. To sort the records in a descending order, you can use the DESC keyword. **SQL ORDER BY Syntax** SELECT column_name1, column_name2 FROM table_name ORDER BY column_name1 ASC|DESC, column_name2 ASC|DESC;

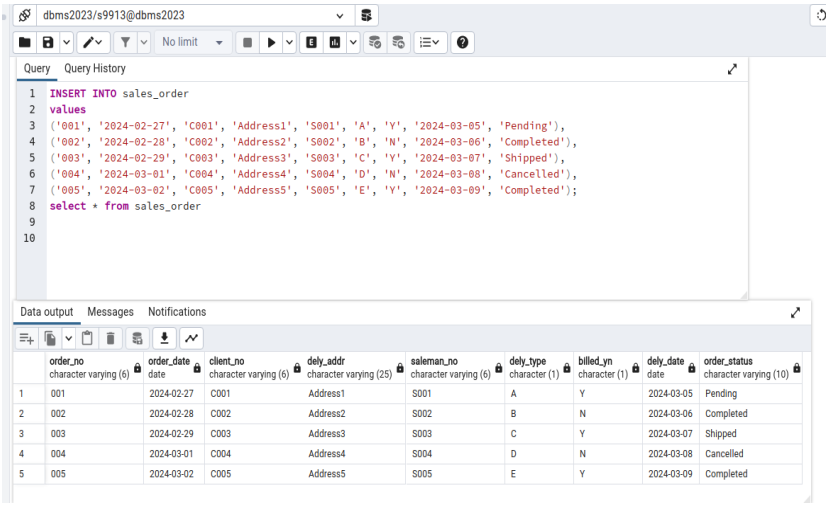
	Procedure	TASK 1:1. Create following table:

Table name : sales_order

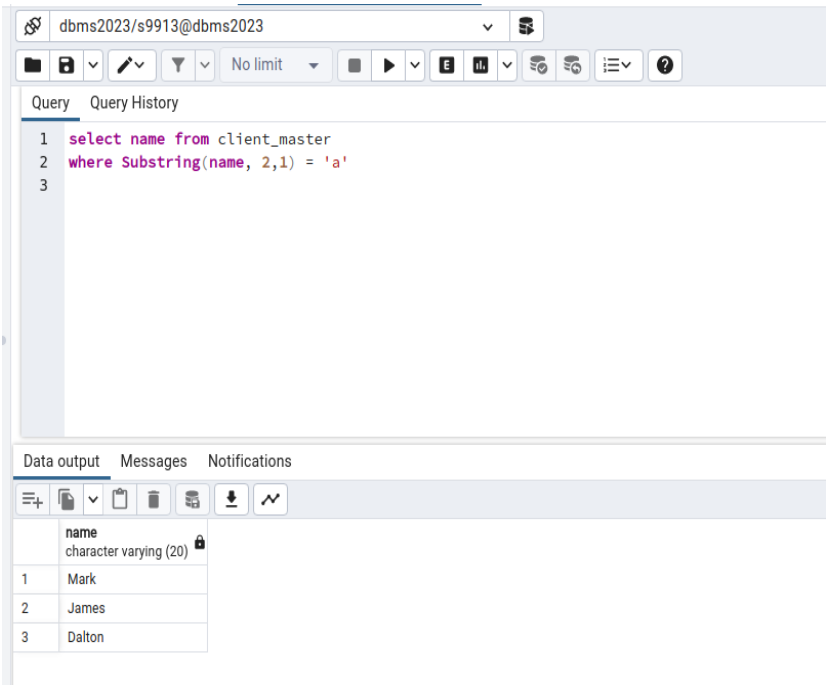
Column Name	Data type	Size
order_no	varchar	6
Order_date	date	
Client_no	varchar	6
Dely_addr	varchar	25
Salesman_no	varchar	6
Dely_type	char	1
Billed_yn	char	1
Dely_date	Date	
Order_status	varchar	10



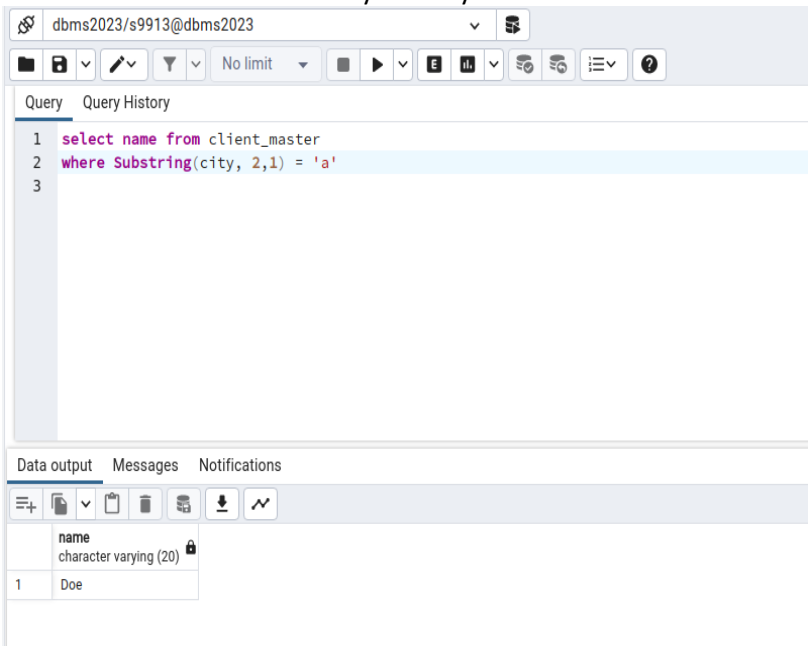
2. Insert 5-6 records in table.



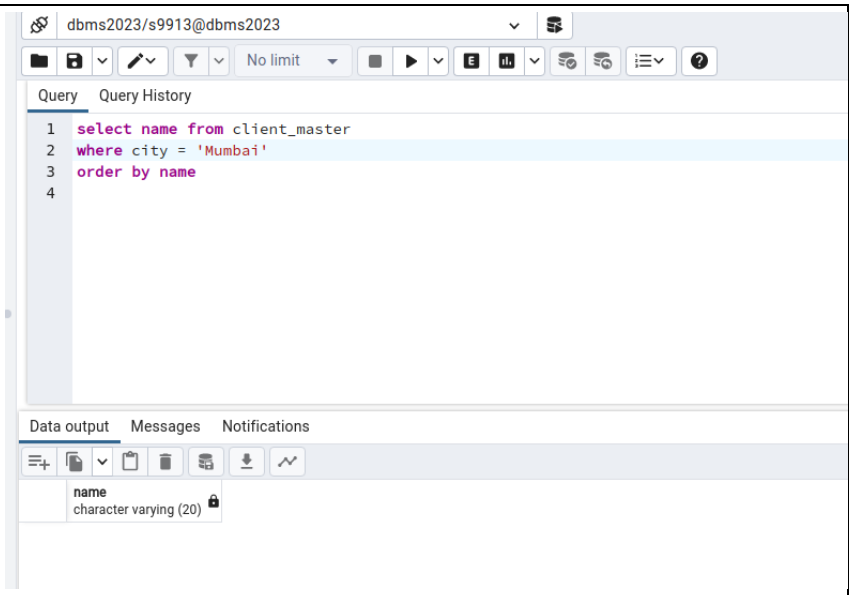
3. Find the names of all clients having ‘a’ as the second letter in their names.



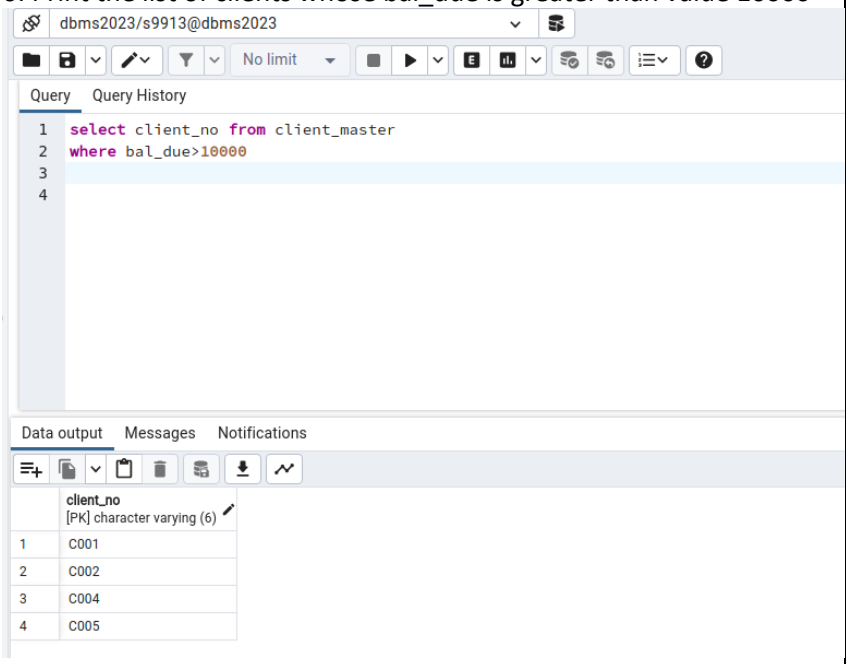
4. Find out the clients who stay in a city whose second letter is ‘a’



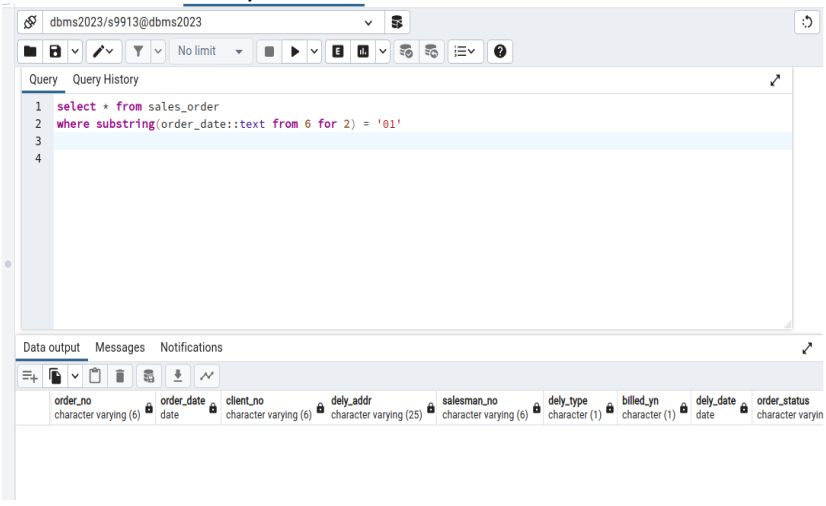
5. Find the list of all clients who stay in ‘mumbai’ ordered by their names



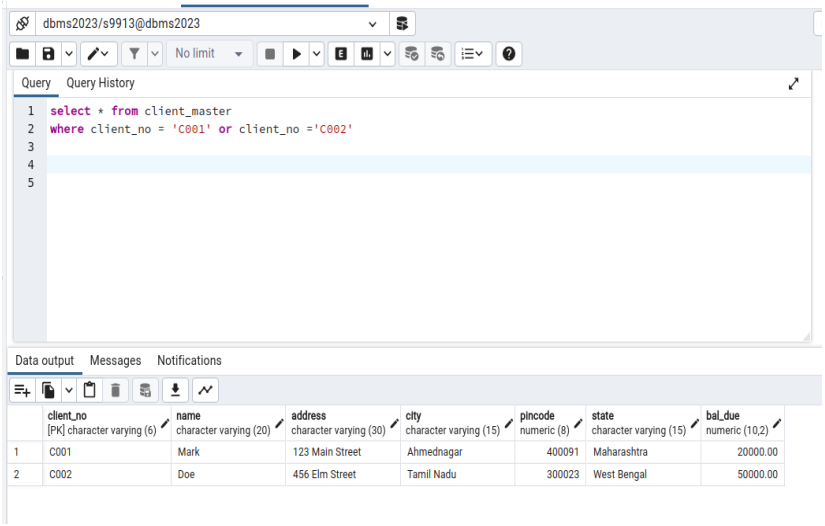
6. Print the list of clients whose bal_due is greater than value 10000



7. Print the information from sales_order table for orders placed in the month of January

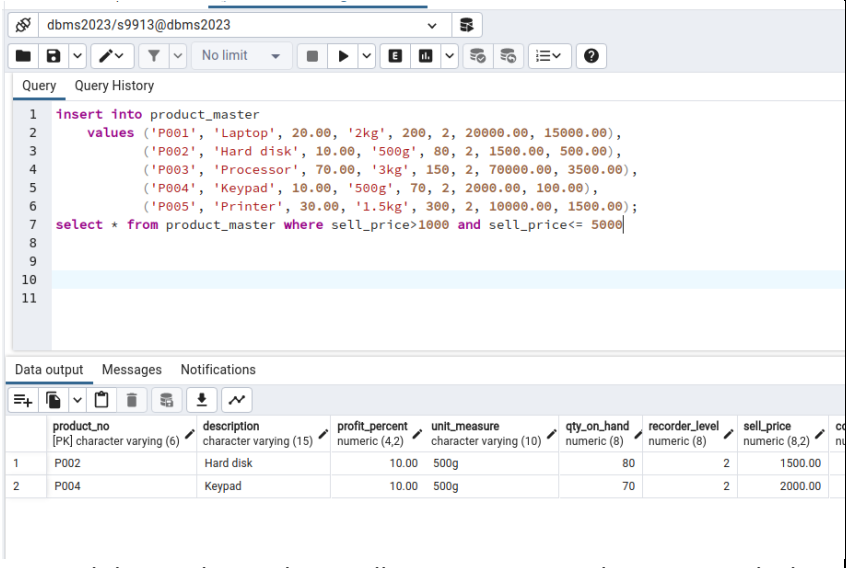


8. Display the order information for client_no C001 and C002



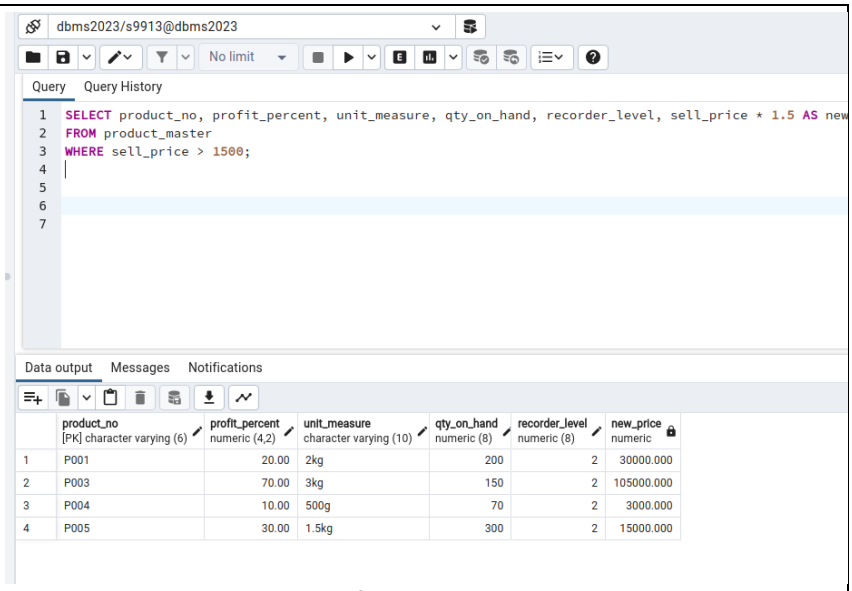
	client_no [PK] character varying (6)	name character varying (20)	address character varying (30)	city character varying (15)	pincode numeric (8)	state character varying (15)	bal_due numeric (10,2)
1	C001	Mark	123 Main Street	Ahmednagar	400091	Maharashtra	20000.00
2	C002	Doe	456 Elm Street	Tamil Nadu	300023	West Bengal	50000.00

9. Find the products whose selling price is greater than 2000 and less than or equal to 5000

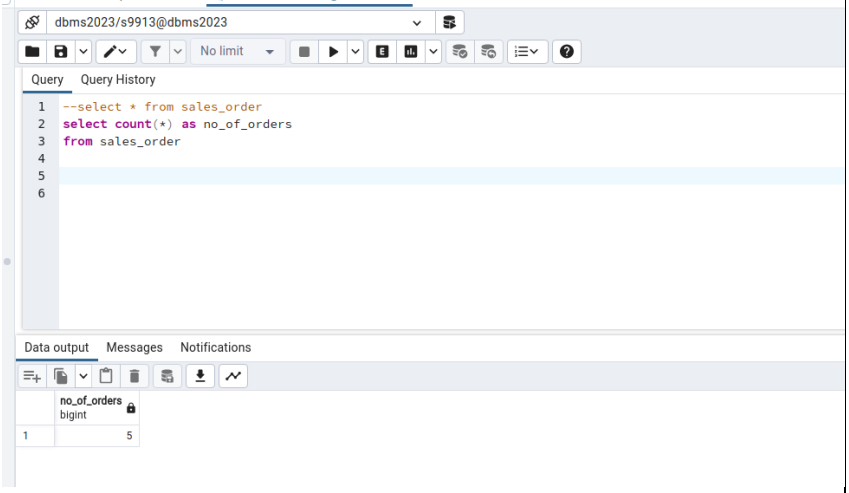


	product_no [PK] character varying (6)	description character varying (15)	profit_percent numeric (4,2)	unit_measure character varying (10)	qty_on_hand numeric (8)	recorder_level numeric (8)	sell_price numeric (8,2)
1	P002	Hard disk	10.00	500g	80	2	1500.00
2	P004	Keypad	10.00	500g	70	2	2000.00

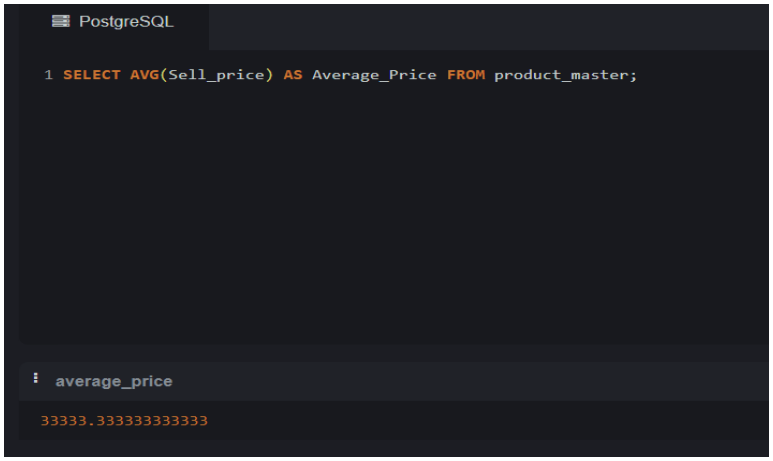
10. Find the products whose selling price is more than 1500. Calculate new selling price as original selling price * 1.5. Rename the new column in the above query as new_price



11. Count the total number of orders



12. Calculate the average price of all the product



13. Determine minimum and maximum product prices

```
SELECT MIN(Sell_price) AS Minimum_Price, MAX(Sell_price) AS Maximum_Price
FROM product_master;
```

minimum_price	maximum_price
10000.00	70000.00

14. count the number of products having price greater than or equal to 1500

```
SELECT COUNT(*) AS ProductCount
FROM product_master
WHERE Sell_price >= 1500;
```

productcount
3

15. Display the order number and day on which clients placed their order

PostgreSQL

```
SELECT order_no, EXTRACT(DOW FROM Order_date) AS Order_Day
FROM sales_order;
```

order_no	order_day
001	2
002	3
003	4
004	5
005	6

16. Display the order_date in the format ‘dd-month-yy’

```
SELECT order_no, TO_CHAR(Order_date, 'DD-Month-YY') AS Formatted_Order_Date
FROM sales_order;
```

order_no	formatted_order_date
001	27-February -24
002	28-February -24
003	29-February -24
004	01-March -24
005	02-March -24

17. Display the month (in alphabets) and date when the order must be delivered

```
SELECT order_no, TO_CHAR(Dely_date, 'Mon DD') AS Formatted_Delivery_Date
FROM sales_order;
```

order_no	formatted_delivery_date
001	Mar 05
002	Mar 06
003	Mar 07
004	Mar 08
005	Mar 09

18. Find the date, 15 days after today’s date

PostgreSQL

```
SELECT CURRENT_DATE + INTERVAL '15 days' AS Date_15_Days_After_Today;
```

date_15_days_after_today
2024-03-18 00:00:00

19. Find the no. of days elapsed between today's date and the delivery date of orders placed by the clients.

```
1 SELECT order_no, Dely_date, CURRENT_DATE - Dely_date AS Days_Elapsed
2 FROM sales_order;
```

order_no	dely_date	days_elapsed
001	2024-03-05	-2
002	2024-03-06	-3
003	2024-03-07	-4
004	2024-03-08	-5
005	2024-03-09	-6

Task2: Use select with where statement with SQL aggregate functions for the tables created in Expt. no. 3/mini project

1.product-master

To find the avg profit %

```
1 SELECT AVG(Profit_percent) AS Average_Profit
2 FROM product_master
3 WHERE Sell_price > 10000;
```

average_profit
45.0000000000000000

2.cliient master

to find the total balance due for clients in the state of Maharashtra.

```sql
SELECT SUM(bal_due) AS Total_Balance_Due
FROM client_master
WHERE STATE = 'Maharashtra';
```

total_balance_due
22000.00

Post Lab Questions:

1. Write a short note on DBA

   A Database Administrator (DBA) is a professional responsible for designing, implementing, and managing database systems. They handle tasks like database installation, performance optimization, security management, backup and recovery, and ensure overall efficiency and reliability of the database.

2. Explain system structure of DBMS

   The system structure of a Database Management System (DBMS) includes users, applications, the DBMS itself, and key components such as the database engine, query processor, transaction manager, storage manager, buffer manager, data dictionary, and database files. These components work together to manage data storage, retrieval, and ensure data integrity and security.

3. Write different date functions

```
SELECT CURRENT_DATE;
SELECT CURRENT_TIME;
SELECT CURRENT_TIMESTAMP;
SELECT DATE_FORMAT(NOW(), '%Y-%m-%d') AS FormattedDate;
SELECT EXTRACT(MONTH FROM hire_date) AS HireMonth FROM employees;
```

4. Differentiate between group by and having with example

   GROUP BY: Used to group rows based on specified columns and apply aggregate functions to each group.

```
SELECT department_id, COUNT(*) AS EmployeeCount
FROM employees
GROUP BY department_id;
```

   HAVING: Used to filter the results of a `GROUP BY` query based on conditions involving aggregate functions.

```
SELECT department_id, COUNT(*) AS EmployeeCount
FROM employees
GROUP BY department_id
HAVING COUNT(*) > 5;
```

5. Give different string functions

   CONCAT:

```
SELECT CONCAT(first_name, ' ', last_name) AS full_name
FROM employees;
```

   UPPER and LOWER:

```
SELECT UPPER(last_name) AS UpperCaseLastName
FROM employees;
```

   LENGTH:

```
SELECT LENGTH(email) AS EmailLength
```

		FROM employees; **SUBSTRING:** SELECT SUBSTRING(last_name, 1, 3) AS Initials FROM employees; **CONVERT:** SELECT CAST('123' AS INT) AS ConvertedNumber;

SE-COMPUTER	Roll number : 9913
Experiment no. : 6	Date of Implementation : 12/ 3/ 2024
Aim : To implement Join and complex SQL commands	
Tool Used : PostgreSQL	
Related Course outcome : At the end of the course, Students will be able to Use SQL : Standard language of relational database	

Rubrics for assessment of Experiment:

Indicator	Poor	Average	Good
Timeliness ● Maintains assignment deadline (3)	Assignment not done (0)	One or More than One week late (1-2)	Maintains deadline (3)
Completeness and neatness ● Complete all parts of assignment(3)	N/A	< 80% complete (1-2)	100% complete (3)
Originality ● Extent of plagiarism(2)	Copied it from someone else(0)	At least few questions have been done without copying(1)	Assignment has been solved completely without copying (2)
Knowledge ● In depth knowledge of the assignment(2)	Unable to answer 2 questions(0)	Unable to answer 1 question (1)	Able to answer 2 questions (2)

Assessment Marks :

Timeliness	
Completeness and neatness	
Originality	
Knowledge	
Total	

Total : (Out of 10)

Teacher's Sign :

EXPERIMENT 5	Complex SQL commands
Aim	To implement complex SQL queries
Tools	PostgreSQL
Theory	

Joining Tables

The FROM clause allows more than 1 table in its list, however simply listing more than one table will *very* rarely produce the expected results. The rows from one table must be correlated with the rows of the others. This correlation is known as *joining*. In the subsequent text, the following 3 example tables are used:

p Table (parts) **s Table (suppliers)** **sp Table (suppliers & parts)**

pno	descr	color
P1	Widget	Blue
P2	Widget	Red
P3	Dongle	Green

sno	name	city
S1	Pierre	Paris
S2	John	London
S3	Mario	Rome

sno	pno	qty
S1	P1	NULL
S2	P1	200
S3	P1	1000
S3	P2	200

An example can best illustrate the rationale behind joins. The following query:

SELECT * FROM sp, p

Produces:

sno	pno	qty	pno	descr	color
S1	P1	NULL	P1	Widget	Blue
S1	P1	NULL	P2	Widget	Red
S1	P1	NULL	P3	Dongle	Green
S2	P1	200	P1	Widget	Blue
S2	P1	200	P2	Widget	Red
S2	P1	200	P3	Dongle	Green
S3	P1	1000	P1	Widget	Blue
S3	P1	1000	P2	Widget	Red
S3	P1	1000	P3	Dongle	Green
S3	P2	200	P1	Widget	Blue
S3	P2	200	P2	Widget	Red
S3	P2	200	P3	Dongle	Green

Each row in *sp* is arbitrarily combined with each row in *p*, giving 12 result rows (4 rows in *sp* X 3 rows in *p*.) This is known as a *cartesian product*.

A more usable query would correlate the rows from *sp* with rows from *p*, for instance matching on the common column -- *pno*:

SELECT *
FROM sp, p
WHERE sp.pno = p.pno

This produces:

sno	pno	qty	pno	descr	color
S1	P1	NULL	P1	Widget	Blue
S2	P1	200	P1	Widget	Blue
S3	P1	1000	P1	Widget	Blue
S3	P2	200	P2	Widget	Red

More information refer this
https://www.tutorialspoint.com/sql/sql-using-joins.htm

Procedure

1. Create following table:

Table name : sales_order_details

Column Name	Data type	Size
order_no	varchar	6
Product_no	varchar	6
Qty_ordered	numeric	8
Qty_disp	numeric	8
Product_rate	numeric	10,2

Create table- customer(cid, cname, address, pno)

Create table- cust_order(cid foreign key, order_no foreign key)

```
1  CREATE TABLE sales_order_details (
2      order_no VARCHAR(6) REFERENCES salesorder(order_no),
3      Product_no VARCHAR(6),
4      Qty_ordered NUMERIC(8),
5      Qty_disp NUMERIC(8),
6      Product_rate NUMERIC(10,2)
7  );
8  SELECT * FROM sales_order_details;
9
```

Scratch Pad × Messages Data output Notifications

order_no character varying (6)	product_no character varying (6)	qty_ordered numeric (8)	qty_disp numeric (8)	product_rate numeric (10,2)

```
9  CREATE TABLE customer (
10     cid SERIAL UNIQUE PRIMARY KEY,
11     cname VARCHAR(255),
12     address VARCHAR(255),
13     pno VARCHAR(15)
14 );
15 SELECT * FROM customer;
```

Scratch Pad × Messages Data output Notifications

cid [PK] integer	cname character varying (255)	address character varying (255)	pno character varying (15)

```sql
CREATE TABLE cust_order (
    cid INT REFERENCES customer(cid) UNIQUE,
    order_no VARCHAR(6) REFERENCES sales_order_details(order_no)
);
SELECT * FROM cust_order;
```

Scratch Pad × | Messages | Data output | Notifications

cid integer	order_no character varying (6)

2. Insert 5-6 records in table in each tables.

```sql
-- Insert into sales_order_details
/*INSERT INTO sales_order_details (or
VALUES
    ('SO001', 'P001', 10, 8, 25.50),
    ('SO002', 'P002', 15, 12, 30.75),
    ('SO003', 'P003', 20, 18, 15.20),
    ('SO004', 'P004', 12, 10, 40.00),
    ('SO005', 'P005', 8, 6, 18.75),
    ('SO006', 'P006', 25, 20, 22.50);
```

Data Output | Messages | Notifications

	order_no [PK] character varying (6)	product_no character varying (6)	qty_ordered numeric (8)	qty_disp numeric (8)	product_rate numeric (10,2)
1	SO001	P001	10	8	25.50
2	SO002	P002	15	12	30.75
3	SO003	P003	20	18	15.20
4	SO004	P004	12	10	40.00
5	SO005	P005	8	6	18.75
6	SO006	P006	25	20	22.50

```sql
-- Insert into customer
/*INSERT INTO customer (cid, cname, address, pno)
VALUES
    (1, 'John Doe', '123 Main St, Cityville', '555-1234'),
    (2, 'Jane Smith', '456 Oak St, Townsville', '555-5678'),
    (3, 'Bob Johnson', '789 Maple St, Villagetown', '555-9101'),
    (4, 'Alice Williams', '101 Pine St, Hamletville', '555-1122'),
    (5, 'Charlie Brown', '202 Cedar St, Boroughville', '555-3344'),
    (6, 'Eva Davis', '303 Elm St, Villageville', '555-5566');*/

-- Insert into cust_order
/*INSERT INTO cust_order (cid, order_no)
```

cid [PK] integer	cname character varying (255)	address character varying (255)	pno character varying (15)
1	John Doe	123 Main St, Cityville	555-1234
2	Jane Smith	456 Oak St, Townsville	555-5678
3	Bob Johnson	789 Maple St, Villagetown	555-9101
4	Alice Williams	101 Pine St, Hamletville	555-1122
5	Charlie Brown	202 Cedar St, Boroughville	555-3344
6	Eva Davis	303 Elm St, Villageville	555-5566

```
-- Insert into cust_order
/*INSERT INTO cust_order (cid, order_
VALUES
    (1, 'SO001'),
    (2, 'SO002'),
    (3, 'SO003'),
    (4, 'SO004'),
    (5, 'SO005'),
    (6, 'SO006');*/

--SELECT * FROM sales_order_details;
--SELECT * FROM customer;
SELECT * FROM cust_order;
```

ta Output Messages Notifications

order_no [PK] character varying (6)	cid integer
SO001	1
SO002	2
SO003	3
SO004	4
SO005	5
SO006	6

3. Print the description and total qty sold for each product

```sql
SELECT
    p.product_no,
    p.description,
    COALESCE(SUM(sod.qty_disp), 0) AS total_qty_sold
FROM
    products p
LEFT JOIN
    sales_order_details sod ON p.product_no = sod.produc
GROUP BY
    p.product_no, p.description;


--SELECT * FROM sales_order_details;
--SELECT * FROM customer;
--SELECT * FROM cust_order;
--SELECT * FROM products;
```

a Output | Messages | Notifications

product_no [PK] character varying (6)	description character varying (255)	total_qty_sold numeric
P005	Product 5 Description	6
P001	Product 1 Description	8
P006	Product 6 Description	20
P002	Product 2 Description	12
P004	Product 4 Description	10
P003	Product 3 Description	18

4. Find the value of each product sold

```sql
-- Query to find the value of each product sold
SELECT
    p.product_no,
    p.description,
    COALESCE(SUM(sod.qty_disp * sod.product_rate), 0) AS total_value_s
FROM
    products p
LEFT JOIN
    sales_order_details sod ON p.product_no = sod.product_no
GROUP BY
    p.product_no, p.description;


--SELECT * FROM sales_order_details;
--SELECT * FROM customer;
--SELECT * FROM cust_order;
--SELECT * FROM products;
```

ta Output | Messages | Notifications

product_no [PK] character varying (6)	description character varying (255)	total_value_sold numeric
P005	Product 5 Description	112.50
P001	Product 1 Description	204.00
P006	Product 6 Description	450.00
P002	Product 2 Description	369.00
P004	Product 4 Description	400.00
P003	Product 3 Description	273.60

5. Calculate the average quantity sold for each client that has a maximum order value of 15000

```
-- Query to calculate the average quantity sold for each cli
WITH MaxOrderClients AS (
    SELECT
        co.cid,
        MAX(sod.qty_disp * sod.product_rate) AS max_order_val
    FROM
        cust_order co
    JOIN
        sales_order_details sod ON co.order_no = sod.order_n
    GROUP BY
        co.cid
    HAVING
        MAX(sod.qty_disp * sod.product_rate) <= 15000
)
```

```
SELECT
    co.cid,
    c.cname,
    AVG(sod.qty_disp) AS avg_qty_sold
FROM
    cust_order co
JOIN
    sales_order_details sod ON co.order_no = sod.or
JOIN
    customer c ON co.cid = c.cid
JOIN
    MaxOrderClients moc ON co.cid = moc.cid
GROUP BY
    co.cid, c.cname;
```

ta Output | Messages | Notifications

cid integer	cname character varying (255)	avg_qty_sold numeric
5	Charlie Brown	6.0000000000000000
3	Bob Johnson	18.0000000000000000
2	Jane Smith	12.0000000000000000
6	Eva Davis	20.0000000000000000
1	John Doe	8.0000000000000000
4	Alice Williams	10.0000000000000000

6. Find out the sum total of all the billed orders for the month of January

```
-- Assuming the correct column name is "order_date"
SELECT
    SUM(qty_disp * product_rate) AS total_billed_amount
FROM
    sales_order_details
WHERE
    EXTRACT(MONTH FROM order_date) = 1;  -- January
```

ata Output | Messages | Notifications

order_no [PK] character varying (6)	product_no character varying (6)	qty_ordered numeric (8)	qty_disp numeric (8)
SO001	P001	10	8
SO002	P002	15	12
SO003	P003	20	18
SO004	P004	12	10
SO005	P005	8	6
SO006	P006	25	20

7. Find out the name of customers who have given the order of more than 10 qty.

```
SELECT
    c.cname
FROM
    customer c
JOIN
    cust_order co ON c.cid = co.cid
JOIN
    sales_order_details sod ON co.order_no = sod.order_
WHERE
    sod.qty_ordered > 10;


--SELECT * FROM sales_order_details;
--SELECT * FROM customer;
```

a Output | Messages | Notifications

cname character varying (255)
Jane Smith
Bob Johnson
Alice Williams
Eva Davis

8. Find out the customer names with product no with maximum qty ordered.

```
WITH MaxQtyPerProduct AS (
  SELECT
    co.cid,
    sod.product_no,
    MAX(sod.qty_ordered) AS max_qty_ordered
  FROM
    cust_order co
  JOIN
    sales_order_details sod ON co.order_no = sod.order_no
  GROUP BY
    co.cid, sod.product_no
)
SELECT
  c.cname,
  m.product_no
FROM
  customer c
JOIN
  MaxQtyPerProduct m ON c.cid = m.cid
WHERE
```

```
WHERE
  (c.cid, m.max_qty_ordered) IN (
    SELECT
      cid,
      MAX(max_qty_ordered) AS max_qty_ordered
    FROM
      MaxQtyPerProduct
    GROUP BY
      cid
  )
```

cname character varying (255)	product_no character va
Eva Davis	P006
Charlie Brown	P005
Bob Johnson	P003
Alice Williams	P004
Jane Smith	P002
John Doe	P001

9. Find out most frequent orders

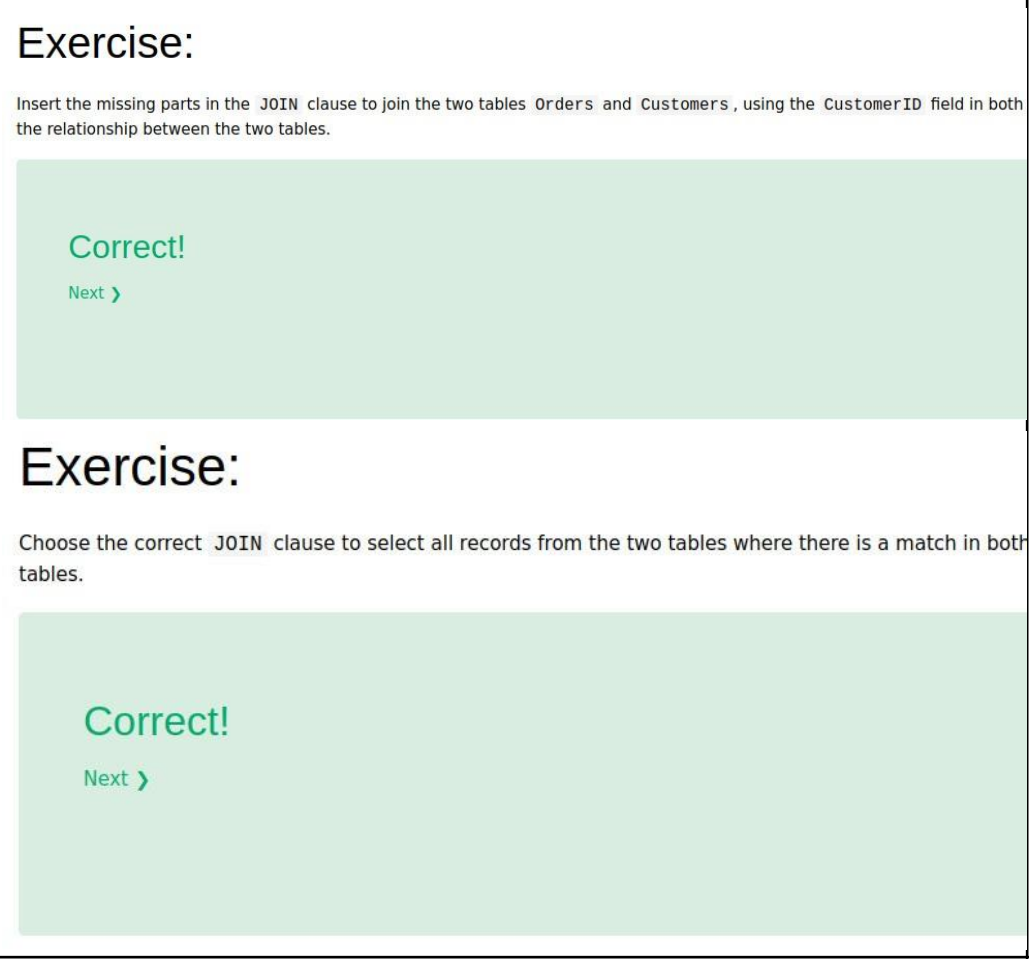
```
SELECT
    order_no,
    COUNT(*) AS order_frequency
FROM
    cust_order
GROUP BY
    order_no
ORDER BY
    order_frequency DESC
LIMIT 1;
```

a Output | Messages | Notifications

order_no [PK] character varying (6)	order_frequency bigint
SO005	1

Post Lab Questions:

1. What is the difference between inner Join and outer Join.
   ⇒ The main difference between inner join and outer join in SQL is that an inner join returns only the rows with matching values in both tables, while an outer join returns all the rows from the database tables, including those that do not have a match in the other table. There are three types of outer joins: left outer join, right outer join, and full outer join. An inner join is a simple join that provides the result directly, while an outer join is a complex join that requires additional syntax to specify the type of join. Outer joins are generally faster than inner joins because they are less restrictive and do not require precise matches.
2. Give one example for equi_join and non equi_join.
   ⇒
   - In an Equi Join, the join operation is based on an equality condition using the equals sign (=).For instance, consider two tables: "state" and "city." The "state" table contains State_ID and State_Name columns, while the "city" table contains City_ID and City_Name columns. An Equi Join can be used to map cities with the states they belong to based on a common column.
   - In a Non Equi Join, the join condition involves comparison operators other than the equals sign, such as >, <, >=, <=. For example, consider two tables: "orders" and "customer." To retrieve order numbers and order amounts from the "orders" table and customer names and working areas from the "customer" table where the order amount matches any opening amount in the customer table, a Non Equi Join can be used.
3. complete online exercise and add screen shots https://www.w3schools.com/sql/exercise.asp?filename=exercise_join1



Exercise:

Choose the correct `JOIN` clause to select all the records from the `Customers` table plus all the matches
the `Orders` table.

Correct!

Next ❯

SE COMP - A	Roll number : 9913
Experiment no. : 7	Date of Implementation: 26/ 03/ 2024
Aim : To implement Nested Sub-queries in SQL	
Tool Used : PostgreSQL/ Mysql	
Related Course outcome : At the end of the course, Students will be able to Use SQL : Standard language of relational database	

Rubrics for assessment of Experiment:

Indicator	Poor	Average	Good
Timeliness ● Maintains assignment deadline (3)	Assignment not done (0)	One or More than One week late (1-2)	Maintains deadline (3)
Completeness and neatness ● Complete all parts of assignment(3)	N/A	< 80% complete (1-2)	100% complete (3)
Originality ● Extent of plagiarism(2)	Copied it from someone else(0)	At least few questions have been done without copying(1)	Assignment has been solved completely without copying (2)
Knowledge ● In depth knowledge of the assignment(2)	Unable to answer 2 questions(0)	Unable to answer 1 question (1)	Able to answer 2 questions (2)

Assessment Marks :

Timeliness	
Completeness and neatness	
Originality	
Knowledge	
Total	

Total : (Out of 10)

Teacher's Sign :

EXPERIMENT 7	Nested subqueries in SQL
Aim	To implementnested sub-queries in SQL
Tools	PostgreSQL/Mysql

Procedure	Use the tables created in the previous experiments and Perform the following queries using nested sub-queries. **Client_master** (client_no, name, address, city, pincode, state, bal_due) **Product_master** (product_no, description, profit_percentage, unit_measure,qty_on_hand, reorder_level, sell_price, cost_price) **Sales_order**(order_no, order_date, client_no, dely_Addr, salesman_no, dely_type, billed_yn, dely_date, order_status) **Sales_order_details**(order_no, product_no, qty_ordered, qty_disp, product_rate)

1. Find the product no. and description of non-moving products i.e. products not being sold.

```
SELECT product_no, description
FROM product_master
WHERE product_no NOT IN (SELECT DISTINCT product_no FROM sales_order_details);
```

	product_no [PK] character varying (6)	description character varying (15)
1	P004	Keypad
2	P005	Printer

2. Find the customer name, address for the client who has placed order no 'O191'

```
SELECT name, address FROM client_master
WHERE client_no = (SELECT client_no FROM salesorder WHERE order_no = 'ORD191')
```

Data output | Messages | Notifications

	name character varying (20)	address character varying (30)

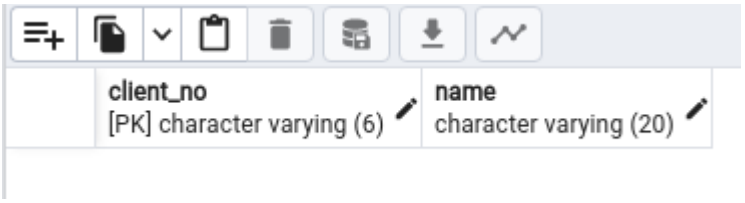
3. Find the clients names who have placed orders before the month of May'96

```
SELECT name FROM client_master
WHERE client_no IN
    (SELECT cLIENT_no
     FROM salesorder
     WHERE EXTRACT(MONTH FROM order_date) < 5
     AND
     EXTRACT
     (YEAR FROM order_date) = 1996);
```

	name character varying (20)

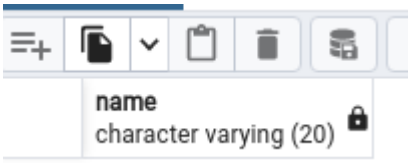
4. Find out if the product '1.44 Drive' has been ordered by any client and print the client_no, name to whom it was sold

```
SELECT client_no, name FROM client_master
WHERE client_no
IN
(SELECT client_no FROM sales_order_details
WHERE product_no = (SELECT product_no
                    FROM product_master
                    WHERE description = '1.44 Drive'));
```



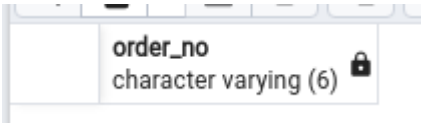
5. Find the names of clients who have placed orders worth Rs. 10000 or more

```
SELECT name FROM client_master
WHERE client_no
IN
(SELECT client_no FROM sales_order_details
 GROUP BY client_no
 HAVING
 SUM(qty_ordered * product_rate) >= 10000);
```



6. Retrieve all the orders placed by a client named 'Rahul Desai' from the sales_order table.

```
SELECT order_no FROM salesorder
WHERE
client_no = (SELECT client_no
             FROM client_master
             WHERE name = 'Rahul Desai');
```



7. Retrieve name, address, city of all the clients who have placed an order through salesman no 's001'.

```
SELECT name, address, city FROM client_master
WHERE client_no IN (SELECT client_no FROM salesorder WHERE salesman_no = 'S001')
```

	name character varying (20)	address character varying (30)	city character varying (15)
1	John	Something1	Ahmednagar

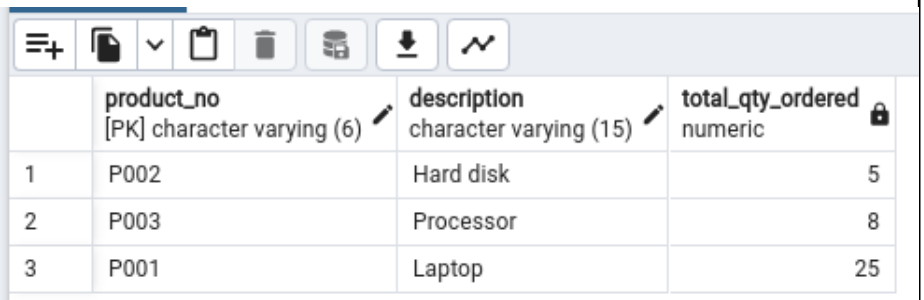
8. Find out all the products that are not being sold from the product_master table, based on the products actually sold as shown in the sales_order_details table.

```
SELECT product_no FROM product_master
WHERE product_no
NOT IN
(SELECT product_no
 FROM sales_order_details);
```

	product_no [PK] character varying (6)
1	P004
2	P005

9. Retrieve the product numbers, their description and the total quantity ordered for each product.

```
SELECT product_master.product_no, description,
SUM(qty_ordered)
AS total_qty_ordered
FROM sales_order_details
JOIN product_master
ON sales_order_details.product_no = product_master.product_n
GROUP BY product_master.product_no, description;
```



	product_no [PK] character varying (6)	description character varying (15)	total_qty_ordered numeric
1	P002	Hard disk	5
2	P003	Processor	8
3	P001	Laptop	25

	Post Lab Questions:	1. What is incremental Update? In database management systems (DBMS), an incremental update refers to updating data by only modifying the specific changes made since the last update, rather than reprocessing the entire dataset. This method is efficient for large datasets where only a portion of the data has been altered. 2. Explain is use of **on delete cascade** and **on update cascade** with suitable example? In DBMS, the "ON DELETE CASCADE" and "ON UPDATE CASCADE" are referential actions that can be specified when defining foreign key constraints in relational databases. Here is how they are used: - ON DELETE CASCADE: When a record in the parent table is deleted, all related records in the child table will be automatically deleted. This maintains referential integrity by removing dependent records when the referenced record is removed. It ensures that there are no orphaned rows in the child table(s). - ON UPDATE CASCADE: When a record in the parent table is updated, the matching records in the child table will be automatically updated. This action ensures that changes made to the parent table are reflected in the child table, maintaining consistency between the two tables.

SE Comp A	Roll number : 9913
Experiment no. : 8	Date of Implementation : 26/3/2024
Aim : To implement PL/pgSQL	
Tool Used : PostgreSQL	
Related Course outcome : At the end of the course, Students will be able to Use SQL : Standard language of relational database	

Rubrics for assessment of Experiment:

Indicator	Poor	Average	Good
Timeliness ● Maintains assignment deadline (3)	Assignment not done (0)	One or More than One week late (1-2)	Maintains deadline (3)
Completeness and neatness ● Complete all parts of assignment(3)	N/A	< 80% complete (1-2)	100% complete (3)
Originality ● Extent of plagiarism(2)	Copied it from someone else(0)	At least few questions have been done without copying(1)	Assignment has been solved completely without copying (2)
Knowledge ● In depth knowledge of the assignment(2)	Unable to answer 2 questions(0)	Unable to answer 1 question (1)	Able to answer 2 questions (2)

Assessment Marks :

Timeliness	
Completeness and neatness	
Originality	
Knowledge	
Total	

Total : (Out of 10)

Teacher's Sign :

EXPERIMENT 8	PL/pgSQL

	Aim	To implement PL/pgSQL
	Tools	PostgreSQL http://www.postgresqltutorial.com/postgresql-stored-procedures/ mysql https://dev.mysql.com/doc/refman/8.0/en/cursors.html https://www.mysqltutorial.org/mysql-error-handling-in-stored-procedures/ https://dev.mysql.com/doc/refman/8.0/en/error-message-elements.html https://www.mysqltutorial.org/mysql-stored-procedure-tutorial.aspx
	Procedure	PL/pgSQL is a loadable procedural language for the Postgres database system.

This package was originally written by Jan Wieck. The design goals of PL/pgSQL were to create a loadable procedural language that can be used to create functions and trigger procedures, adds control structures to the SQL language.

Structure of PL/pgSQL

PL/pgSQL is a block-structured language. The complete text of a function definition must be a block. A block is defined as:

[<<label>>]
[DECLARE Declarations]
BEGIN
statements
END [label];

Each declaration and each statement within a block is terminated by a semicolon. A block that appears within another block must have a semicolon after END , as shown above; however the final END that concludes a function body does not require a semicolon

IF boolean-expression THEN
statements
END IF;
	IF boolean-expression
	THEN
	statements
	ELSE statements
	END IF;

WHILE boolean-expression LOOP statements END LOOP [label];	FOR name IN [REVERSE] expression..expression [BY expression] LOOP statements END LOOP [label]; FOR i IN 1..10 LOOP -- i will take on the values 1,2,3,4,5,6,7,8,9,10 within the loop END LOOP; FOR i IN REVERSE 10..1 LOOP -- i will take on the values 10,9,8,7,6,5,4,3,2,1 within the loop END LOOP; FOR i IN REVERSE 10..1 BY 2 LOOP -- i will take on the values 10,8,6,4,2 within the loop END LOOP;

Procedure

1. Write a PROCEDURE to display sum of digits of a three digit number

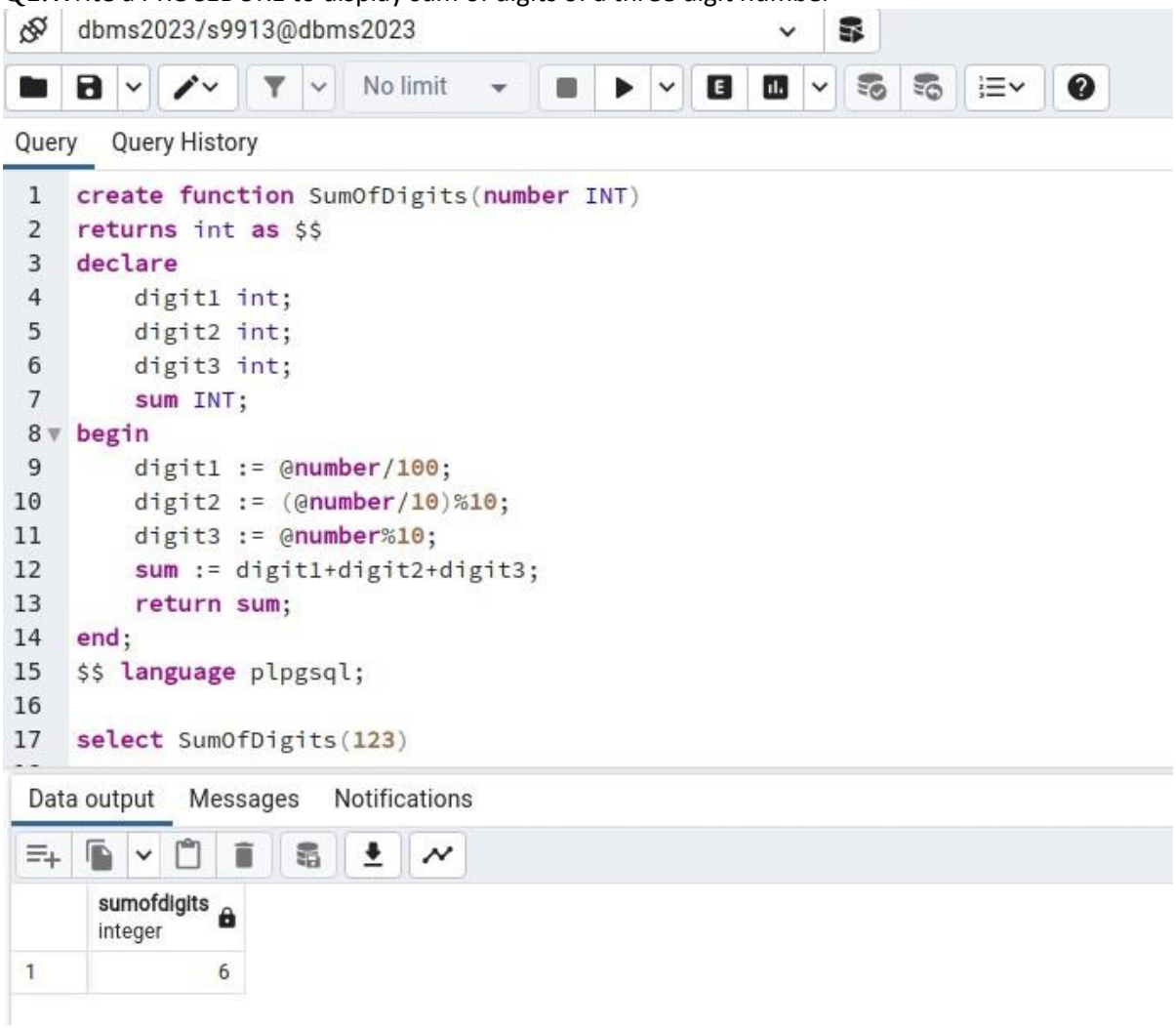
2. Write a procedure/ block to display prime numbers

```
Input : N = 20
Output : 2, 3, 5, 7, 11, 13, 17, 19
```

3. Write a procedure/block to display Fibonacci series upto 8th term (start with 0,1)
4. Create or use EMP(eid, Name, location, mid). Write a procedure using cursor to display list of managers(mid) with name;

	Post Lab Questions:	1. Give advantages of PLSQL vs SQL 2. Explain data types of PgSQL/plsql of mysql

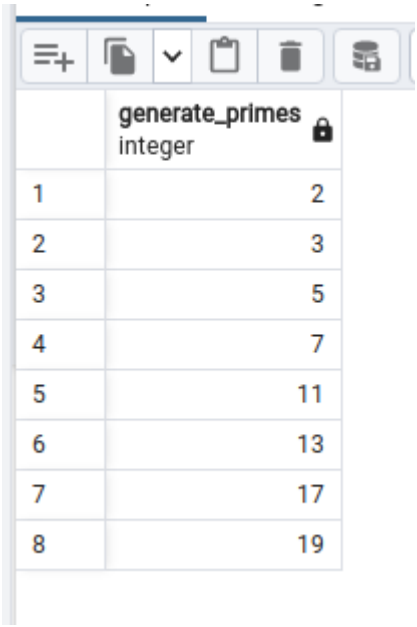
Q1.Write a PROCEDURE to display sum of digits of a three digit number



Q2 Write a procedure/ block to display prime numbers

```
CREATE OR REPLACE FUNCTION generate_primes(limit_num INT)
RETURNS SETOF INT AS $$
DECLARE
        num INT;
        divisor INT;
        is_prime BOOLEAN;
BEGIN
        num := 2; -- Starting from 2, as it's the smallest prime number

        WHILE num <= limit_num LOOP
        is_prime := TRUE;

        -- Check if num is divisible by any number other than 1 and itself
        FOR divisor IN 2..ROUND(SQRT(num)) LOOP
        IF num % divisor = 0 THEN
        is_prime := FALSE;
        EXIT;
```

```
        END IF;
        END LOOP;

        IF is_prime THEN
        RETURN NEXT num;
        END IF;

        num := num + 1;
        END LOOP;

        RETURN;
END;
$$ LANGUAGE PLPGSQL;

select generate_primes(20)
```

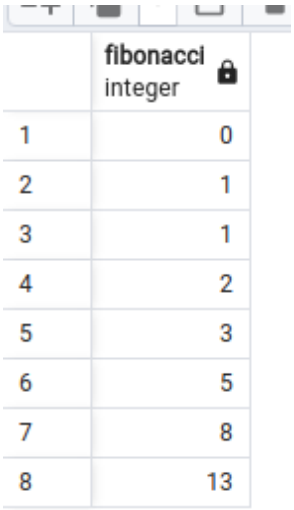


	generate_primes integer
1	2
2	3
3	5
4	7
5	11
6	13
7	17
8	19

Q3 Write a procedure/block to display Fibonacci series upto 8th term (start with 0,1)

```
CREATE OR REPLACE FUNCTION fibonacci(limit_num INT)
RETURNS SETOF INT AS $$
DECLARE
   num1 int := 0;
   num2 int := 1;
        num3 INT;
   n int := 2;
BEGIN
   return next 0;
   return next 1;
   while n<limit_num LOOP
        num3 := num1 + num2;
        num1 := num2;
        num2 := num3;
        n := n+1;

        RETURN next num3;
```

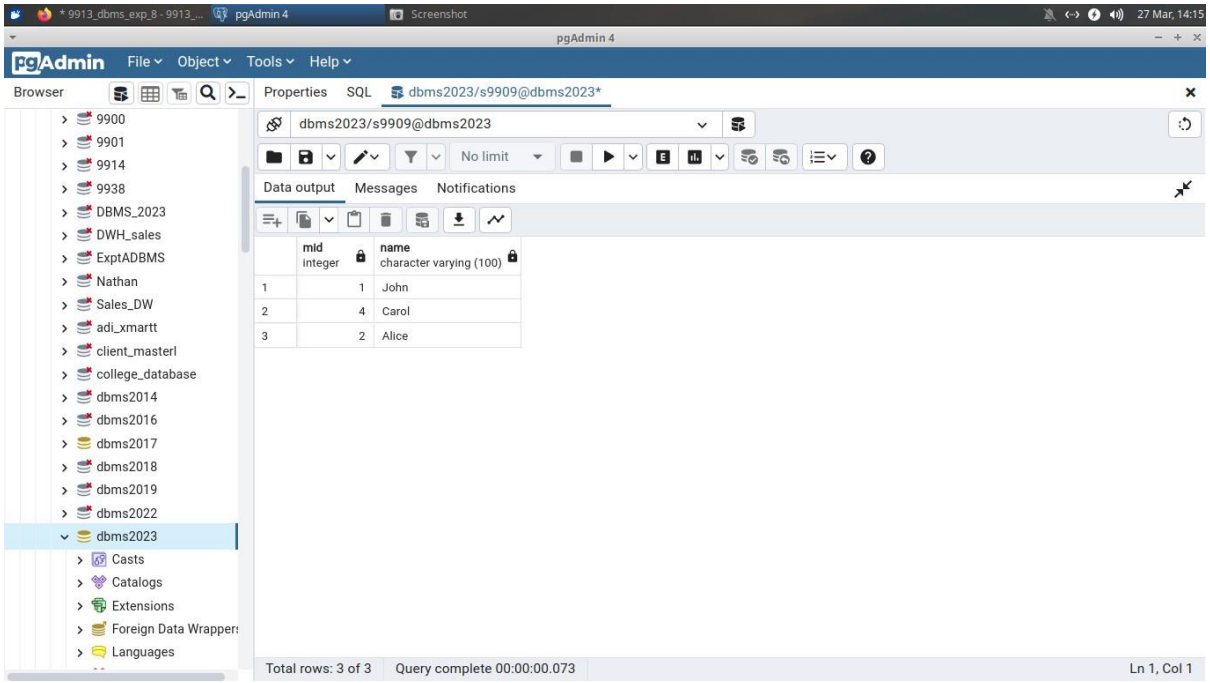
```
    end loop;
    return;
END;
$$ LANGUAGE PLPGSQL;

select fibonacci(8)
```



	fibonacci integer
1	0
2	1
3	1
4	2
5	3
6	5
7	8
8	13

Q4 Create or use EMP(eid, Name, location, mid). Write a procedure using cursor to display list of managers(mid) with name;

```
CREATE TABLE EMP (
eid SERIAL PRIMARY KEY,Name
VARCHAR(100),
location VARCHAR(100),mid INT,
FOREIGN KEY (mid) REFERENCES EMP(eid) -- Self-reference
);
-- Insert sample data
INSERT INTO EMP (Name, location, mid) VALUES ('John', 'New York', 1); INSERT INTO EMP
(Name, location, mid) VALUES ('Alice', 'Los Angeles', 2); INSERT INTO EMP (Name, location,
mid) VALUES ('Bob', 'Chicago', 1); INSERT INTO EMP (Name, location, mid) VALUES ('Carol',
'Houston', NULL);INSERT INTO EMP (Name, location, mid) VALUES ('David', 'Boston', 4);
-- Join EMP with itself based on eid=midSELECT
distinct e1.mid , e2.Name FROM EMP e1
join emp e2
on e1.mid = e2.eid
```



Postlab

Q1

PL/SQL offers several advantages over SQL:

- Procedural Capabilities: PL/SQL provides procedural constructs such as loops, conditional statements, exception handling, and subprograms like functions and procedures. This allows for more complex logic to be implemented directly within the database, reducing the need for round-trips between the application and the database server.
- Encapsulation and Modularity: PL/SQL allows for the encapsulation of SQL statements within blocks of code. This promotes modularity and code reusability, making it easier to maintain and update database logic.
- Performance Optimization: PL/SQL can improve performance by reducing the number of interactions between the application and the database. By executing multiple SQL statements within a single PL/SQL block, you can minimize network traffic and reduce overhead.
- Enhanced Error Handling: PL/SQL provides robust error handling mechanisms, including exception handling blocks, which allow for graceful handling of errors within the database. This improves the reliability and maintainability of database applications.

Q2 Explain data types of PgSQL/plsql of mysql

1. Numeric Data Types:

- PgSQL: Includes integer types like int, smallint, bigint, and floating-point types like real, double precision.
- PL/SQL: Offers similar numeric types such as INTEGER, SMALLINT, NUMBER, and FLOAT.
- MySQL: Provides numeric types like INT, SMALLINT, BIGINT, FLOAT, DOUBLE, etc.

2. Character Data Types:

- PgSQL: Offers character varying(n) (VARCHAR), character(n) (CHAR), text, etc.
- PL/SQL: Provides CHAR, VARCHAR2, CLOB for character data.
- MySQL: Supports CHAR, VARCHAR, TEXT, etc.

3. Date and Time Data Types:

- PgSQL: Includes timestamp, date, time, interval, etc.
- PL/SQL: Offers DATE, TIMESTAMP, INTERVAL for date and time handling.
- MySQL: Provides DATE, TIME, DATETIME, TIMESTAMP, YEAR, etc.

SE Comp A	Roll number : 9913
Experiment no. : 9	Date of Implementation : 26/ 03/ 2024
Aim : To implement Functions and Triggers	
Tool Used : PostgreSQL	
Related Course outcome : At the end of the course, Students will be able to Use SQL : Standard language of relational database	

Rubrics for assessment of Experiment:

Indicator	Poor	Average	Good
Timeliness • Maintains assignment deadline (3)	Assignment not done (0)	One or More than One week late (1-2)	Maintains deadline (3)
Completeness and neatness • Complete all parts of assignment(3)	N/A	< 80% complete (1-2)	100% complete (3)
Originality • Extent of plagiarism(2)	Copied it from someone else(0)	At least few questions have been done without copying(1)	Assignment has been solved completely without copying (2)
Knowledge • In depth knowledge of the assignment(2)	Unable to answer 2 questions(0)	Unable to answer 1 question (1)	Able to answer 2 questions (2)

Assessment Marks :

Timeliness	
Completeness and neatness	
Originality	
Knowledge	
Total	

Total : (Out of 10)

Teacher's Sign :

EXPERIMENT 09	***Functions and Triggers***
Aim	To implement PL/pgSQL function and trigger
Tools	PostgreSQL http://www.postgresqltutorial.com/postgresql-create-function/ http://www.postgresqltutorial.com/plpgsql-function-overloading/ http://www.postgresqltutorial.com/plpgsql-function-returns-a-table/ http://www.postgresqltutorial.com/creating-first-trigger-postgresql/ PostgreSQL: Documentation: 15: 43.10. Trigger Functions

	Theory	CREATE FUNCTION defines a new function. CREATE OR REPLACE FUNCTION will either create a new function, or replace an existing definition. To be able to define a function, the user must have the USAGE privilege on the language.If a schema name is included, then the function is created in the specified schema. Otherwise it is created in the current schema. The name of the new function must not match any existing function with the same input argument types in the same schema. However, functions of different argument types can share a name (this is called *overloading*). **Syntax for Function** CREATE [OR REPLACE] FUNCTION name ([[argmode] [argname] argtype [{ DEFAULT | = } default_expr] [, ...]]) [RETURNS rettype | RETURNS TABLE (column_name column_type [, ...])] { LANGUAGE lang_name | WINDOW | IMMUTABLE | STABLE | VOLATILE | CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT | [EXTERNAL] SECURITY INVOKER | [EXTERNAL] SECURITY DEFINER | COST execution_cost | ROWS result_rows | SET configuration_parameter { TO value | = value | FROM CURRENT } | AS 'definition' | AS 'obj_file', 'link_symbol' } ... [WITH (attribute [, ...])] If you drop and then recreate a function, the new function is not the same entity as the old; you will have to drop existing rules, views, triggers, etc. that refer to the old function. Use CREATE OR REPLACE FUNCTION to change a function definition without breaking objects that refer to the function. The trigger can be specified to fire before the operation is attempted on a row (before constraints are checked and the INSERT, UPDATE, or DELETE is attempted); or after the operation has completed (after constraints are checked and the INSERT, UPDATE, or DELETE has completed); or instead of the operation (in the case of inserts, updates or deletes on a view). If the trigger fires before or instead of the event, the trigger can skip the operation for the current row, or change the row being inserted (for INSERT and UPDATE operations only). If the trigger fires after the event, all changes, including the effects of other triggers, are "visible" to the trigger.

		Syntax of Trigger CREATE [CONSTRAINT] TRIGGER name { BEFORE | AFTER | INSTEAD OF } { event [OR ...] } ON table [FROM referenced_table_name] [NOT DEFERRABLE | [DEFERRABLE] { INITIALLY IMMEDIATE | INITIALLY DEFERRED }] [FOR [EACH] { ROW | STATEMENT }] [WHEN (condition)] EXECUTE PROCEDURE function_name (arguments) where event can be one of: INSERT UPDATE [OF column_name [, ...]] DELETE TRUNCATE To create a trigger on a table, the user must have the TRIGGER privilege on the table. The user must also have EXECUTE privilege on the trigger function. Use DROP TRIGGER to remove a trigger.

AIM	1. Write a function to find factorial of a number use it and write the observation for it. 2. Create table emp(id,name,salary) and insert 3 records in it.

```
CREATE or REPLACE Function factorial(n numeric)
returns numeric
language plpgsql
as
$$
begin
  IF n = 0 THEN
  RETURN 1;
  ELSE
   RETURN n * factorial(n-1);
   end if;
end;
$$;

SELECT factorial(5);
```

Data output | Messages | Notifications

	factorial integer
1	120

```sql
CREATE TABLE emp(
   emp_id numeric(4),
    emp_name varchar (10),
     salary numeric (8,2)
);

INSERT INTO emp
values (1, 'Shreya', 20000),
(2, 'Fiza', 50000),
(3, 'Khushi', 60000),
(4, 'Kush', 80000),
(5, 'Krishna', 80000);

SELECT * FROM emp
```

Data output | Messages | Notifications

	emp_id numeric (4)	emp_name character varying (10)	salary numeric (8,2)
1	1	Shreya	20000.00
2	2	Fiza	50000.00
3	3	Khushi	60000.00
4	4	Kush	80000.00
5	5	Krishna	80000.00

3. Write a function find average salary from emp table

```
CREATE function get_avg_salary()
returns numeric
language plpgsql
as
$$
begin
   return (SELECT AVG(salary)::numeric(8,2) FROM
end;
$$;

SELECT get_avg_salary();
```

Data output | Messages | Notifications

	get_avg_salary numeric
1	58000.00

4. Write a row level trigger that would fire before insert/ update/delete operations performed on emp table, not allowing these operations and display the appropriate message.

```
CREATE or REPLACE Function prevent_operation()
returns TRIGGER
language plpgsql
as
$$
begin
    RAISE EXCEPTION 'Insertion , deletion or any updation is not allowed on this
    RETURN NULL;
end;
$$;

CREATE TRIGGER prevent_operation_trigger
BEFORE INSERT OR DELETE OR UPDATE ON emp
FOR EACH ROW
```

Data output | Messages | Notifications

```
NOTICE:  drop cascades to trigger prevent_operation_trigger on table emp
DROP FUNCTION

Query returned successfully in 29 msec.
```

5. Write a row level trigger that would fire after insert/update/delete operations performed on emp table displaying date on which data manipulation performed.

ry Query History

```
CREATE or REPLACE Function emp_modifications()
returns TRIGGER
language plpgsql
as
$$
begin
    RAISE INFO 'Updated on : % ',NOW();
    RETURN NULL;
end;
$$;

CREATE TRIGGER emp_modifications_trigger
AFTER INSERT OR DELETE OR UPDATE ON emp
FOR EACH ROW
EXECUTE PROCEDURE emp_modifications();

SELECT * FROM emp
```

Data output Messages Notifications

	emp_id numeric (4)	emp_name character varying (10)	sal nur
1	1	Shreya	
2	2	Fiza	
3	3	Khushi	
4	4	Kush	
5	5	Krishna	

Post Lab Questions:

1. Explain syntax of function in Mysql /PostgreSQL with example

⇒ The general structure of mysql function is:

```
DELIMITER //
CREATE FUNCTION function_name(parameter INT) RETURNS
INT
BEGIN
   DECLARE variable_name INT;
   -- Function logic
   RETURN variable_name;
END;
//
DELIMITER ;
```

Here is an example of creating a function in MySQL:

```
DELIMITER //
CREATE FUNCTION CalcIncome (starting_value INT) RETURNS
INT
BEGIN
   DECLARE income INT;
   SET income = 0;
   label1: WHILE income <= 3000 DO
      SET income = income + starting_value;
   END WHILE label1;
   RETURN income;
END;
//
DELIMITER ;
```

The general structure of a function in postgresql is:

```
CREATE FUNCTION somefunc(integer, text) RETURNS integer AS
$$
DECLARE
   -- Local variables declaration
BEGIN
   -- Function logic
END;
$$
 LANGUAGE plpgsql;
```

Here is an example of creating a function in PostgreSQL:

```
CREATE FUNCTION totalRecords() RETURNS integer AS $total$
DECLARE
   total integer;
BEGIN
   SELECT count(*) INTO total FROM COMPANY;
   RETURN total;
```

END;
$total$ LANGUAGE plpgsql;

2. Explain trigger example with syntax in Mysql/postgreSQL.

⇒ **MySQL Trigger Example:**

In MySQL, to create a trigger, you use the CREATE TRIGGER statement. Below is an example of a trigger that updates a timestamp column whenever a row is inserted into a table:

sql

```
CREATE TRIGGER update_timestamp
BEFORE INSERT ON table_name
FOR EACH ROW
SET NEW.timestamp_column = NOW();
```

- Explanation:
  - CREATE TRIGGER: Initiates the trigger creation.
  - update_timestamp: Name of the trigger.
  - BEFORE INSERT ON table_name: Specifies the trigger to execute before an insert operation on a specific table.
  - FOR EACH ROW: Indicates that the trigger should be executed for each row affected by the operation.

SET NEW.timestamp_column = NOW(): Sets the timestamp_column to the current timestamp when a new row is inserted.

PostgreSQL Trigger Example:

In PostgreSQL, triggers are created using the CREATE TRIGGER statement. Here is an example of a trigger that logs changes made to a specific column in a table:

```
CREATE TRIGGER log_changes
AFTER UPDATE OF column_name ON table_name
FOR EACH ROW
EXECUTE FUNCTION log_update();
```

Explanation:

- CREATE TRIGGER: Starts the trigger creation.
- log_changes: Name of the trigger.
- AFTER UPDATE OF column_name ON table_name: Specifies the trigger to execute after an update operation on a specific column in a table.
- FOR EACH ROW: Indicates that the trigger should be executed for each row affected by the operation.
- EXECUTE FUNCTION log_update(): Calls the log_update function to log the changes made.

SE Comp A	Roll number : 9913
Experiment no. : 10	Date of Implementation :
Aim: Simple Transaction implementation	
Tool Used : PostgreSQL/Mysql	
Related Course outcome : At the end of the course, Students will be able to Use and Apply the concept of transaction, concurrency and recovery	

Rubrics for assessment of Experiment:

Indicator	Poor	Average	Good
Timeliness • **Maintains assignment deadline (3)**	**Assignment not done (0)**	**One or More than One week late (1-2)**	**Maintains deadline (3)**
Implementation of concepts (3)	**N/A**	**< 80% complete (1-2)**	**100% complete (3)**
Originality • **Extent of plagiarism(2)**	**Copied it from someone else(0)**	**At least few parts of it have been done without copying(1)**	**Experiment has been solved completely without copying (2)**
Knowledge • **In depth knowledge of the assignment(2)**	**Unable to answer 2 questions(0)**	**Unable to answer 1 question (1)**	**Able to answer 2 questions (2)**

Assessment Marks :

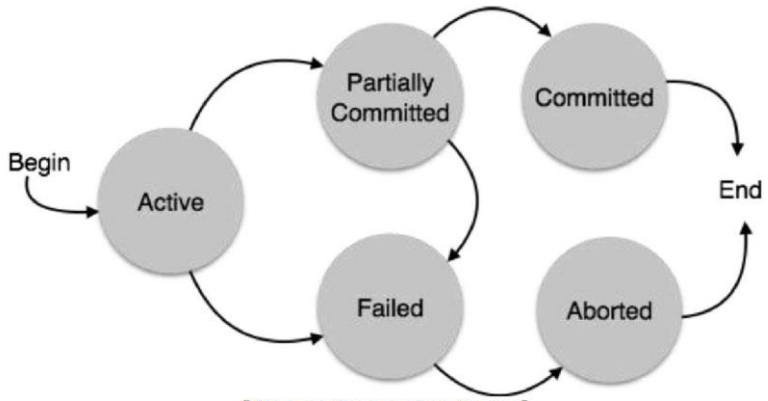
Timeliness	
Completeness and neatness	
Originality	
Knowledge	
Total	

Total : (Out of 10)

Teacher's Sign :

EXPERIMENT 10	***Transaction concept***
Aim	To implement Simple Transaction concept
Tools	PostgreSQL/Mysql

Theory	A transaction can be defined as a group of tasks. A single task is the minimum processing unit which cannot be divided further.
	Transactions are a fundamental concept of all database systems. The essential point of a transaction is that it bundles multiple steps into a single, all-or-nothing operation. The intermediate states between the steps are not visible to other concurrent transactions, and if some failure occurs that prevents the transaction from completing, then none of the steps affect the database at all.
	Properties of Transactions
	Transactions have the following four standard properties, usually referred to by the acronym ACID −
	● **Atomicity** − Ensures that all operations within the work unit are completed successfully; otherwise, the transaction is aborted at the point of failure and previous operations are rolled back to their former state.
	● **Consistency** − Ensures that the database properly changes states upon a successfully committed transaction.
	● **Isolation** − Enables transactions to operate independently of and transparent to each other.
	● **Durability** − Ensures that the result or effect of a committed transaction persists in case of a system failure.
	
	In PostgreSQL, a transaction is set up by surrounding the SQL commands of the transaction with BEGIN and COMMIT commands. So our banking transaction would actually look like:
	BEGIN;
	UPDATE accounts SET balance = balance - 100.00
	WHERE name = 'Alice';
	-- etc etc
	COMMIT;
	End;

Theory	**State Diagram :** A transaction in a database can be in one of the following states:  *[Image: Transaction States]* For example, consider a bank database that contains balances for various customer accounts, as well as total deposit balances for branches. Suppose that we want to record a payment of $100.00 from Alice's account to Bob's account. BEGIN; --sql SAVEPOINT my_savepoint; UPDATE accounts SET balance = balance - 100.00 WHERE name = 'Alice'; UPDATE accounts SET balance = balance + 100.00 WHERE name = 'Bob'; ROLLBACK TO my_savepoint; or commit; --UPDATE accounts SET balance = balance + 100.00 WHERE name = 'Wally'; COMMIT;

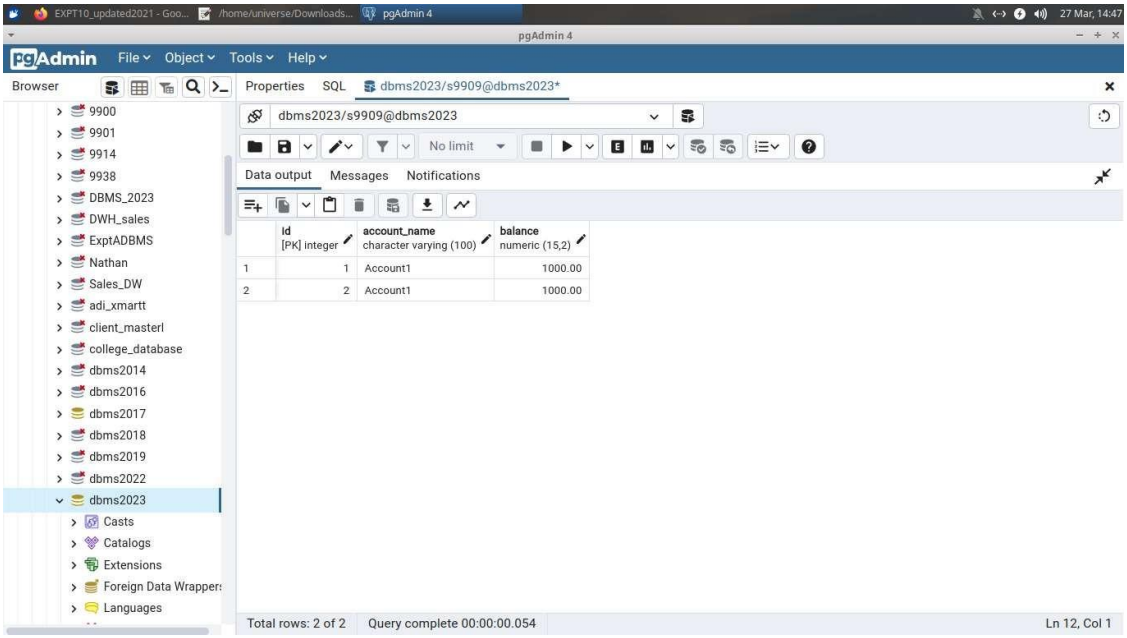
Theory	# DOCXTransaction Control The following commands are used to control transactions − ● **BEGIN TRANSACTION** − To start a transaction. ● **COMMIT** − To save the changes, alternatively you can use **END TRANSACTION** command. ● **ROLLBACK** − To rollback the changes. Transactional control commands are only used with the DML commands INSERT, UPDATE and DELETE only. They cannot be used while creating tables or dropping them because these operations are automatically committed in the database. ## The BEGIN TRANSACTION Command Transactions can be started using BEGIN TRANSACTION or simply BEGIN command. Such transactions usually persist until the next COMMIT or ROLLBACK command is encountered. But a transaction will also ROLLBACK if the database is closed or if an error occurs. **The following is the simple syntax to start a transaction −** **BEGIN;** **or** **BEGIN TRANSACTION;** **The COMMIT Command** **The COMMIT command is the transactional command used to save changes invoked by a transaction to the database.** **The COMMIT command saves all transactions to the database since the last COMMIT or ROLLBACK command.** **The syntax for COMMIT command is as follows −** **COMMIT;** **or** **END TRANSACTION;**

Theory	**The ROLLBACK Command** The ROLLBACK command is the transactional command used to undo transactions that have not already been saved to the database. The ROLLBACK command can only be used to undo transactions since the last COMMIT or ROLLBACK command was issued. The syntax for ROLLBACK command is as follows − ROLLBACK; PostgreSQL Transaction (postgresqltutorial.com) PostgreSQL: Documentation: 15: 13.2. Transaction Isolation https://www.postgresql.org/docs/current/transaction-iso.html
Task	1. Create any sample table for transaction testing 2. Observe the effect of auto commit on /off on insert stmt(SET AUTOCOMMIT { = | TO } { ON | OFF }) 3. How to find out the current transaction mode(POSTGRESQL-MENU-QUERY-AUTOCOMMMIT) 4. Start transaction, insert records in table and use roll back 5. Start new transaction and commit the current transaction 6. Use save point and use rollback to and commit 7. What happens to the current transaction if a start transaction is executed? 8. What happens to the current transaction if the session is ended? 9. How to view and change the current transaction isolation level? 10. Demonstrate datalocks in transaction and how long a transaction will wait for a data lock? (OPTIONAL) 11. Wa java/php program using jdbc/odbc to implement transaction java-mysql or php-mysql (jdbc/odbc)[https://www.php.net/manual/en/mysqli.begin-transaction.php] (OPTIONAL) 12. Transfer 5k from A's account to B's account using transaction concept 13. demonstrate different locks(MySQL :: MySQL 8.0 Reference Manual :: 15.7.2.4 Locking Reads) (OPTIONAL)

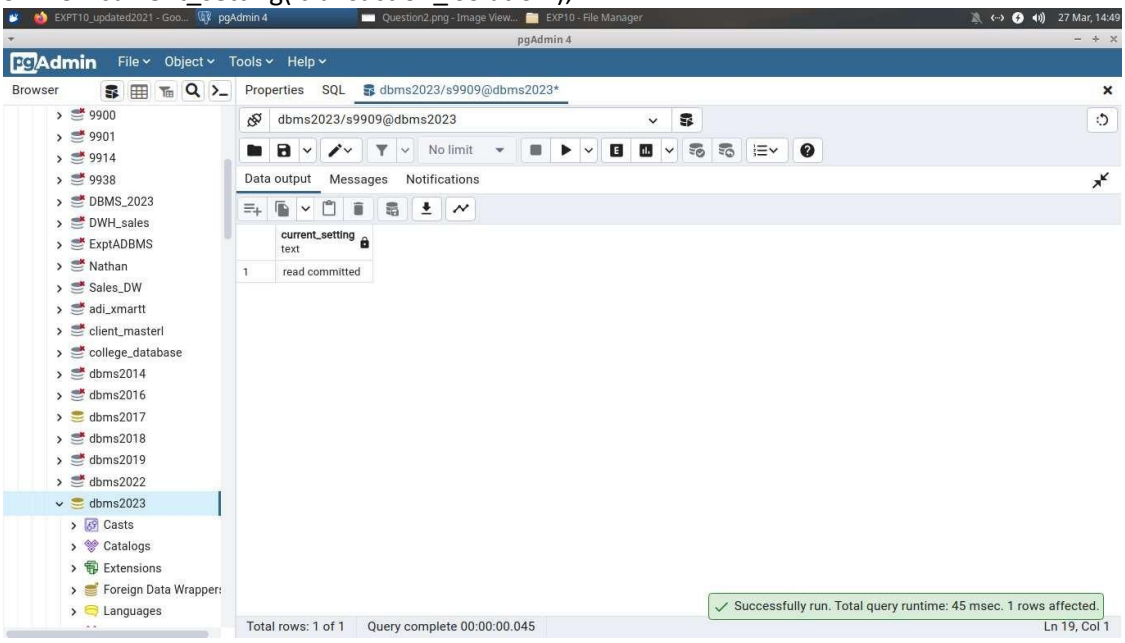
Links	**https://www.postgresql.org/docs/9.1/sql-start-transaction.html** **https://www.postgresql.org/docs/8.3/tutorial-transactions.html** **https://pgdash.io/blog/postgres-transactions.html** **https://tapoueh.org/blog/2018/07/postgresql-concurrency-isolation-and-locking/** **http://dba.fyicenter.com/faq/mysql/mysql_sql_transaction_management.html** **locks** MySQL :: MySQL 8.0 Reference Manual :: 15.7.2.4 Locking Reads Processlist in postgresql - postgres > select * from pg_stat_activity; Terminate (kill) specific session in PostgreSQL database - PostgreSQL Data Dictionary Queries (dataedo.com) PostgreSQL: Documentation: 14: SET TRANSACTION
Post Lab Questions:	**1. How Does MySQL/postgresql Handle Read Consistency?** **2. What Are Transaction Isolation Levels IN MYSQL/ Postgresql** **3. What Are Impacts on Applications from Locks, Timeouts, and DeadLocks?**

1.
```
DROP TABLE IF EXISTS accounts;
BEGIN;
CREATE TABLE accounts (
        id SERIAL PRIMARY KEY,
        account_name VARCHAR(100),
        balance DECIMAL(15, 2)
);
COMMIT;
ROLLBACK;
```

2.
```
BEGIN;
INSERT INTO accounts (account_name, balance) VALUES ('Account1', 1000);
COMMIT;
SELECT * FROM accounts;
```



3. SELECT current_setting('transaction_isolation');

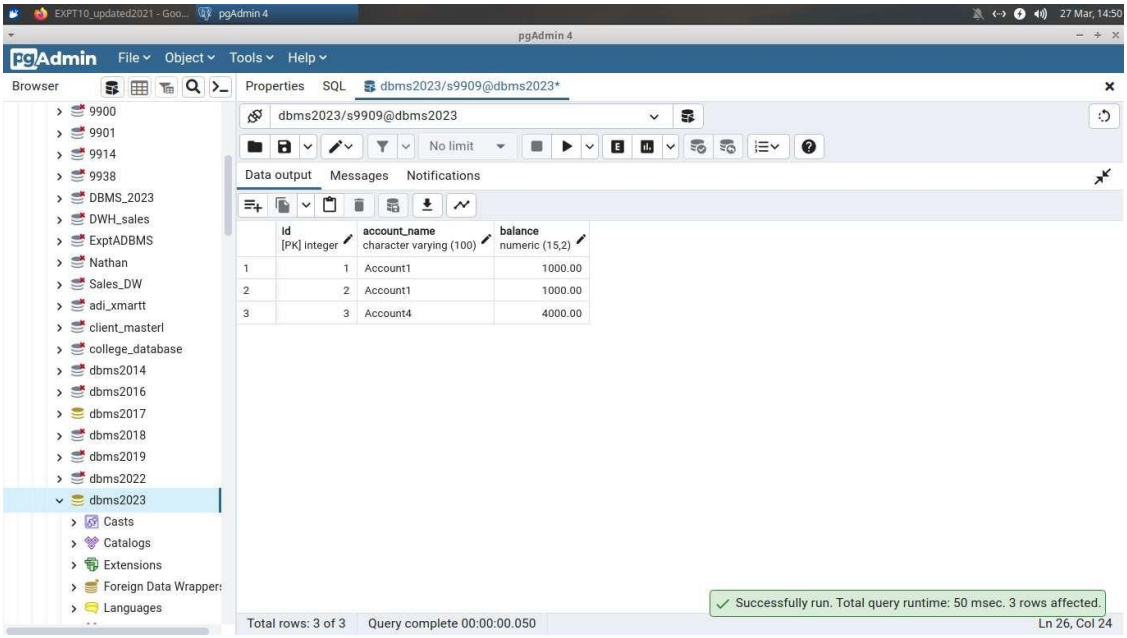


4. BEGIN;

   INSERT INTO accounts (account_name, balance) VALUES ('Account4', 4000);

5. COMMIT;

   SELECT * FROM accounts;



6. BEGIN;

   SAVEPOINT savepoint1;

   INSERT INTO accounts (account_name, balance) VALUES ('Account5', 5000);

   INSERT INTO accounts (account_name, balance) VALUES ('Account6', 6000);

   ROLLBACK TO SAVEPOINT savepoint1;

   SELECT * FROM accounts;

   COMMIT;

   SELECT * FROM accounts;

7. If a "start transaction" command is executed during a current transaction, it typically initiates a new transaction, effectively nesting the new transaction within the current one. This means that any changes made after the "start transaction" command will be part of the new transaction, and they can be committed or rolled back independently of the outer transaction.

8. If the session is ended during a current transaction without committing or rolling back the transaction explicitly, the transaction will typically be automatically rolled back by the database management system. This ensures data integrity and prevents any unintended changes from being permanently committed to the database.

9. SHOW TRANSACTION ISOLATION LEVEL;
   SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
   SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
   BEGIN;
   UPDATE accounts SET balance = balance - 5000 WHERE account_name = 'A';
   UPDATE accounts SET balance = balance + 5000 WHERE account_name = 'B';
   COMMIT;

POSTLAB :

Q1 **How Does MySQL/postgresql Handle Read Consistency?**

MySQL and PostgreSQL both use Multi-Version Concurrency Control (MVCC) to ensure read consistency. In MySQL, different transaction isolation levels, like REPEATABLE READ, control consistency, while PostgreSQL defaults to READ COMMITTED isolation. Both systems provide mechanisms to ensure that transactions see a consistent view of the database despite concurrent modifications.

Q2 **What Are Transaction Isolation Levels IN MYSQL/ Postgresql?**

- MySQL:
  - READ UNCOMMITTED: Allows dirty reads.
  - READ COMMITTED: Prevents dirty reads, allows non-repeatable reads and phantom reads.
  - REPEATABLE READ: Prevents non-repeatable reads, allows phantom reads.
  - SERIALIZABLE: Prevents all anomalies: dirty reads, non-repeatable reads, and phantom reads.

- PostgreSQL:
  - READ UNCOMMITTED: Not directly supported.
  - READ COMMITTED: Prevents dirty reads.
  - REPEATABLE READ: Prevents non-repeatable reads.
  - SERIALIZABLE: Prevents all anomalies.

Q3 **What Are Impacts on Applications from Locks, Timeouts, and DeadLocks?**

  - Locks: Can cause contention and blocking, leading to performance issues.
  - Timeouts: Can result in incomplete transactions or failed operations.
  - Deadlocks: Cause transactions to wait indefinitely for each other, potentially leading to application hangs or crashes.