SARASWATI Education Society's

SARASWATI College of Engineering

Learn Live Achieve and Contribute

Kharghar, Navi Mumbai - 410 210.

NAAC Accredited

Department of Computer Engineering

Name: Shaikh Adnan Shaukat Ali

Rollno: 55

Subject: Database Management System (DBMS)

Class/Sem: SE/IV

INDEX

Prof. Rina Bora

Subject I/C

SARASWATI Education Society's
SARASWATI College of Engineering
Learn Live Achieve and Contribute
Kharghar, Navi Mumbai - 410 210.
NAAC Accredited

Department of Computer Engineering

INDEX

Prof. Rina Bora
Subject I/C

Experiment 1

Aim: Identify the case study and detail statement of problem.

Design an Entity-Relationship (ER) / Extended Entity-Relationship (EER) Model

Hardware and Software Requirement: P-IV and above

Theory:

The entity-relationship E-R model is very useful in mapping the meanings and interactions of real-world enterprises onto a conceptual schema. Because of this usefulness, many database-design tools draw on concepts from the E-R model. The E-R data model employs three basic concepts: entity sets, relationship sets, and attributes. The E-R model also has an associated diagrammatic representation, the ER diagram, which can express the overall logical structure of a database graphically.

An entity is an object that exists and is distinguishable from other objects.

Example: specific person, company, event, plant

An entity set is a set of entities of the same type that share the same properties.

Example: set of all persons, companies, trees, holidays

An entity is represented by a set of attributes; i.e., descriptive properties possessed by all members of an entity set.

Example:

instructor = (*ID, name, street, city, salary*)

course= (*course_id, title, credits*)

A subset of the attributes form a primary key of the entity set; i.e., uniquely identifying each member of the set.

Attribute types:

- Simple and composite attributes.
- Single-valued and multivalued attributes
    - Example: multivalued attribute: *phone_numbers*
- Derived attributes
    - an be computed from other attributes

Domain – the set of permitted values for each attribute

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

Most useful in describing binary relationship sets.

For a binary relationship set the mapping cardinality must be one of the following types:

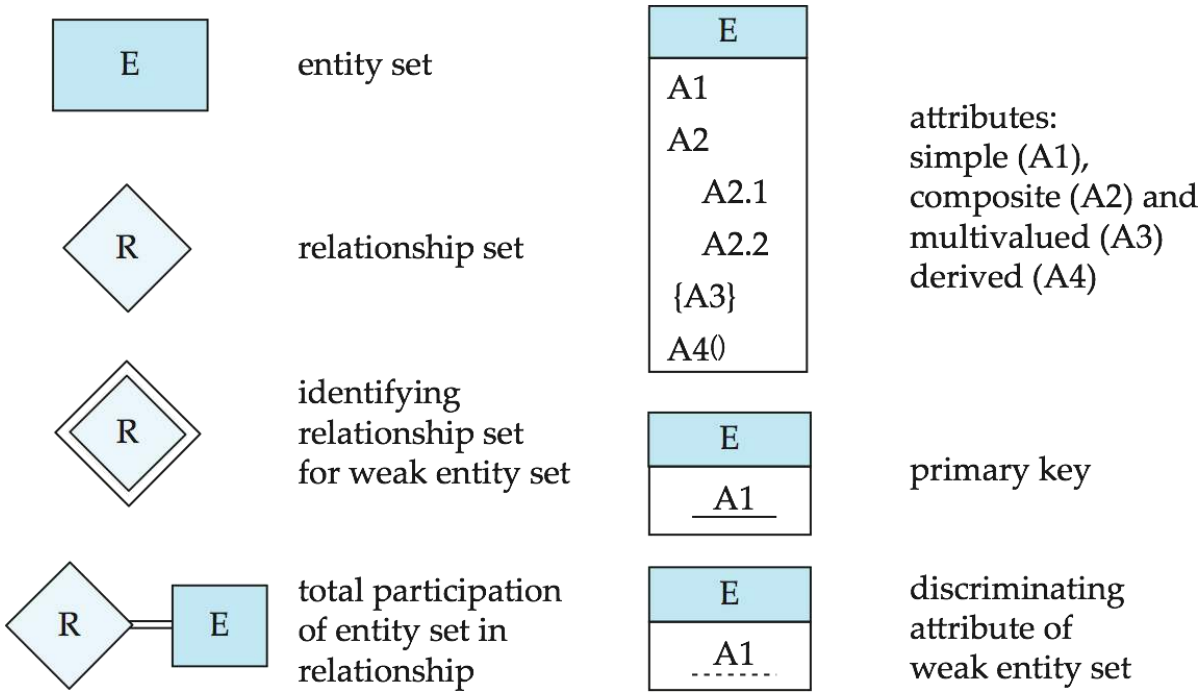
One to one

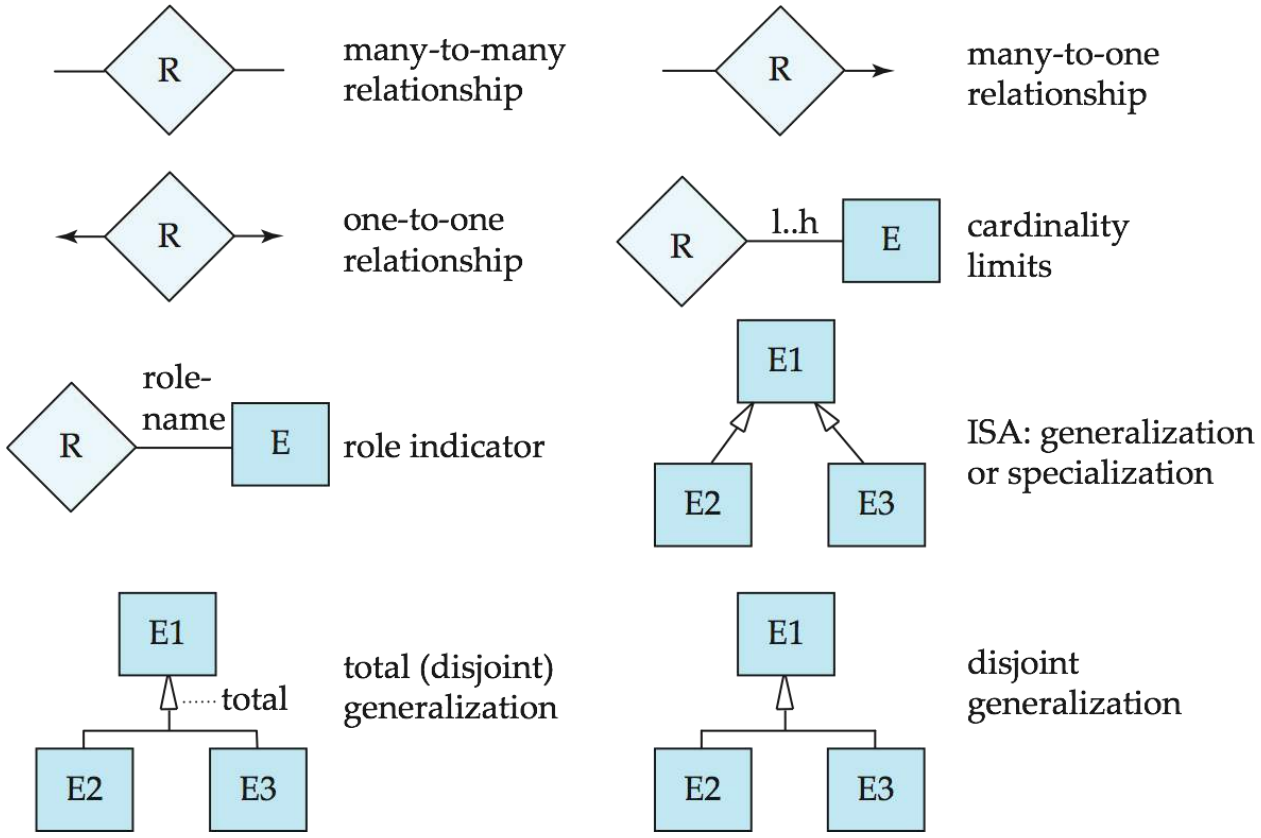
One to many

Many to one

Many to many

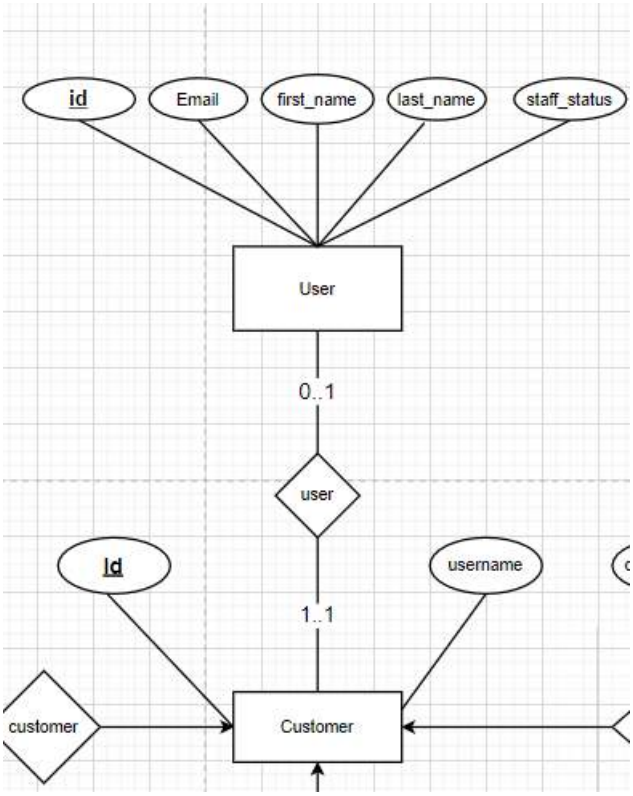
Symbols Used in E-R Notation



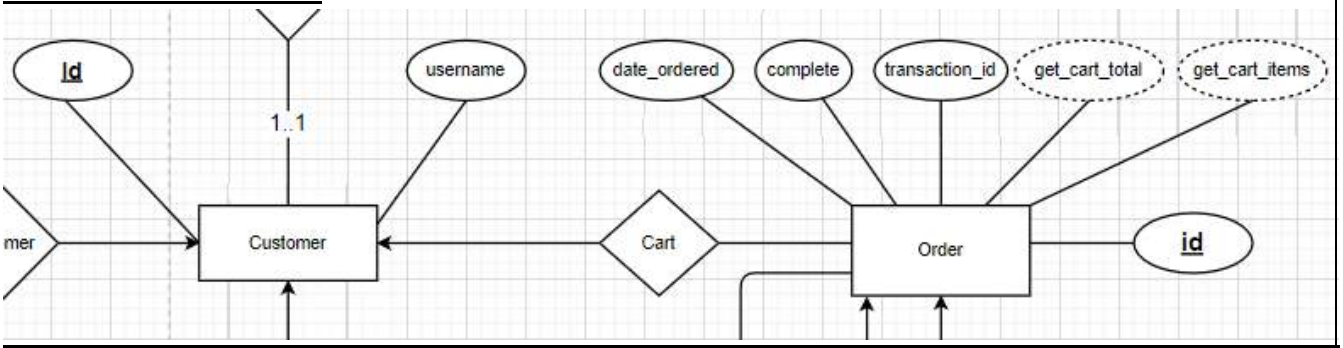


Conclusion: We have Successfully design ER model for Ecommerce website.

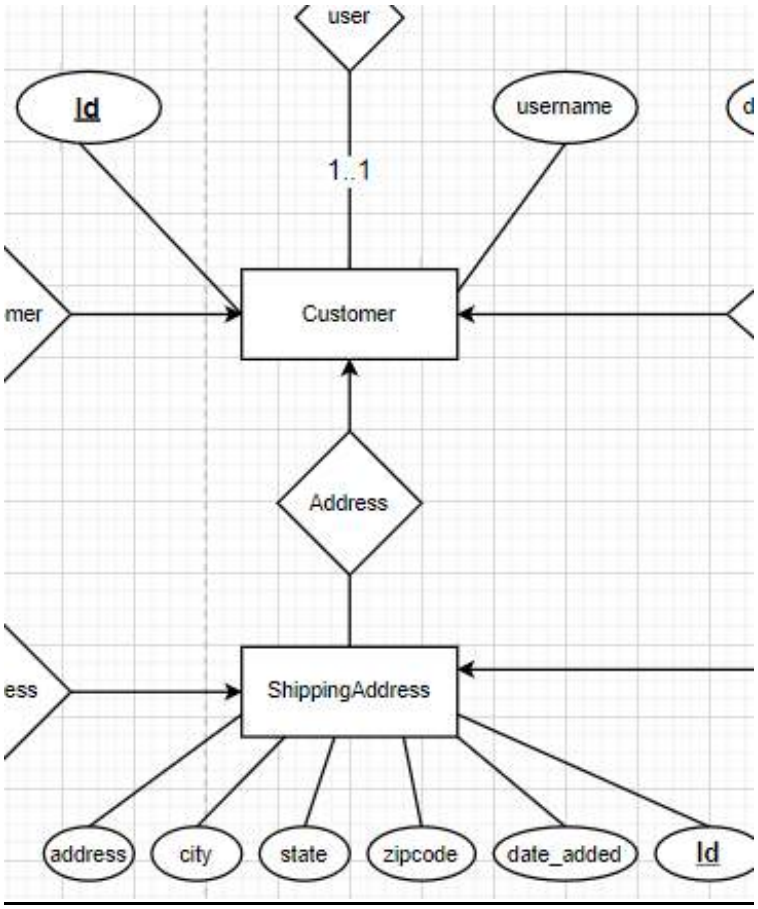
User-Customer Model:



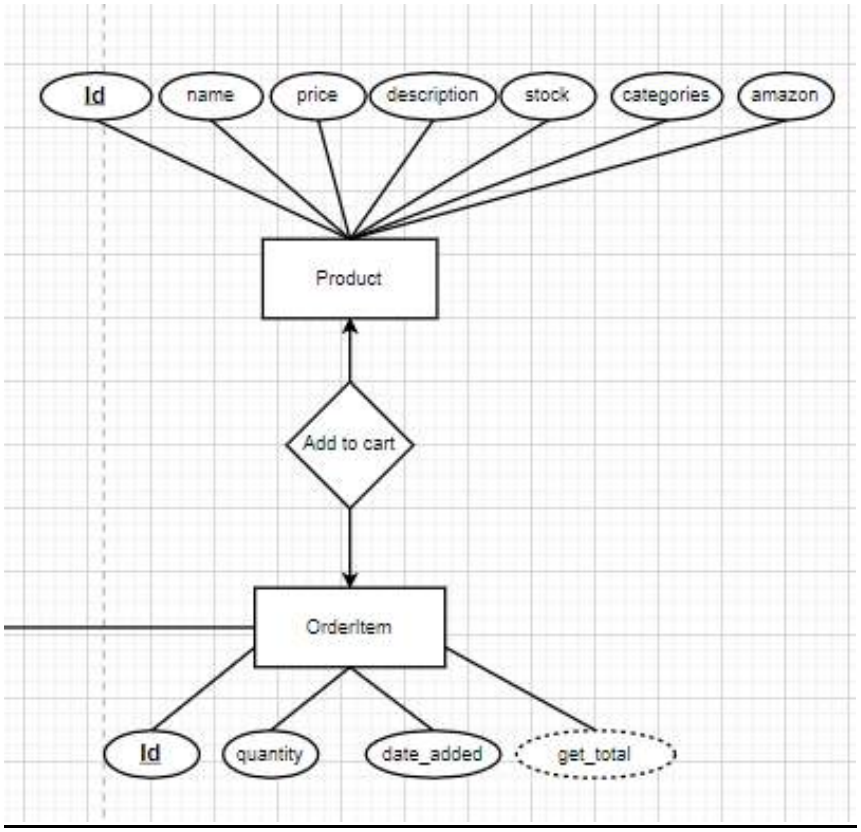
Customer-Order Model:



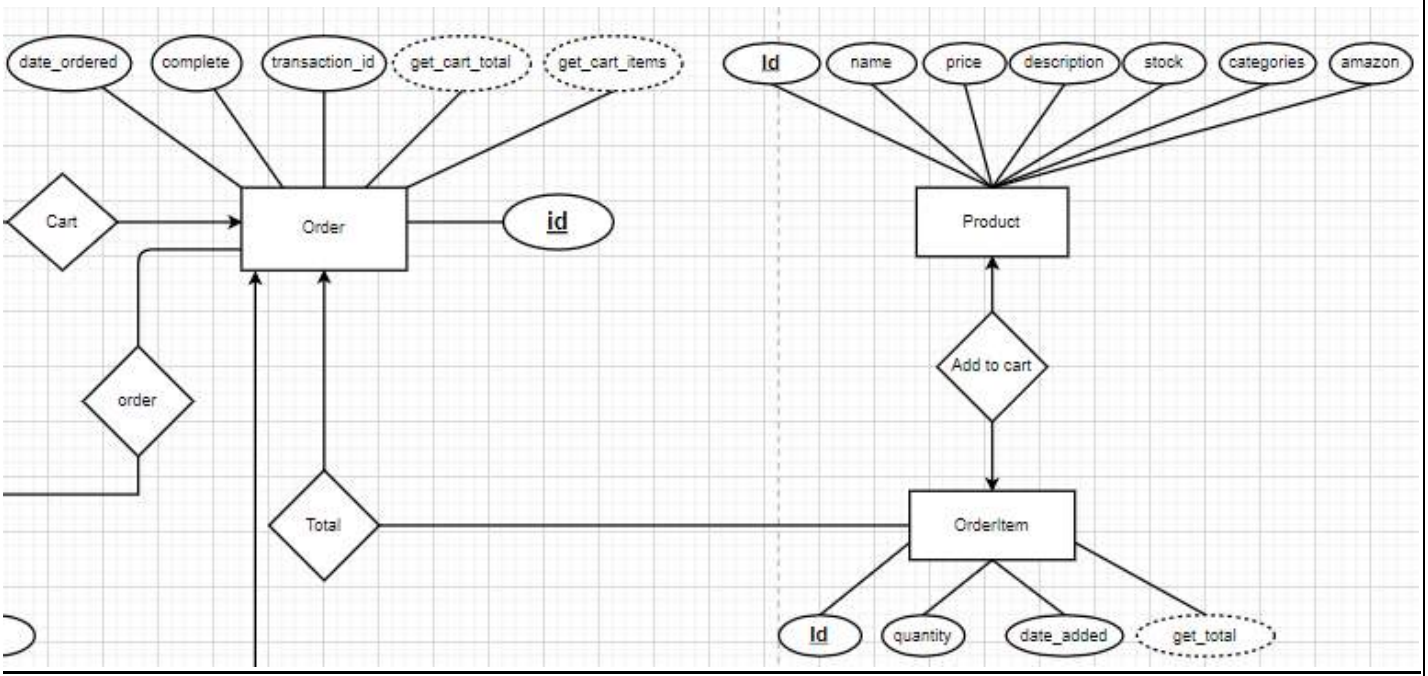
Customer-Shipping Model:



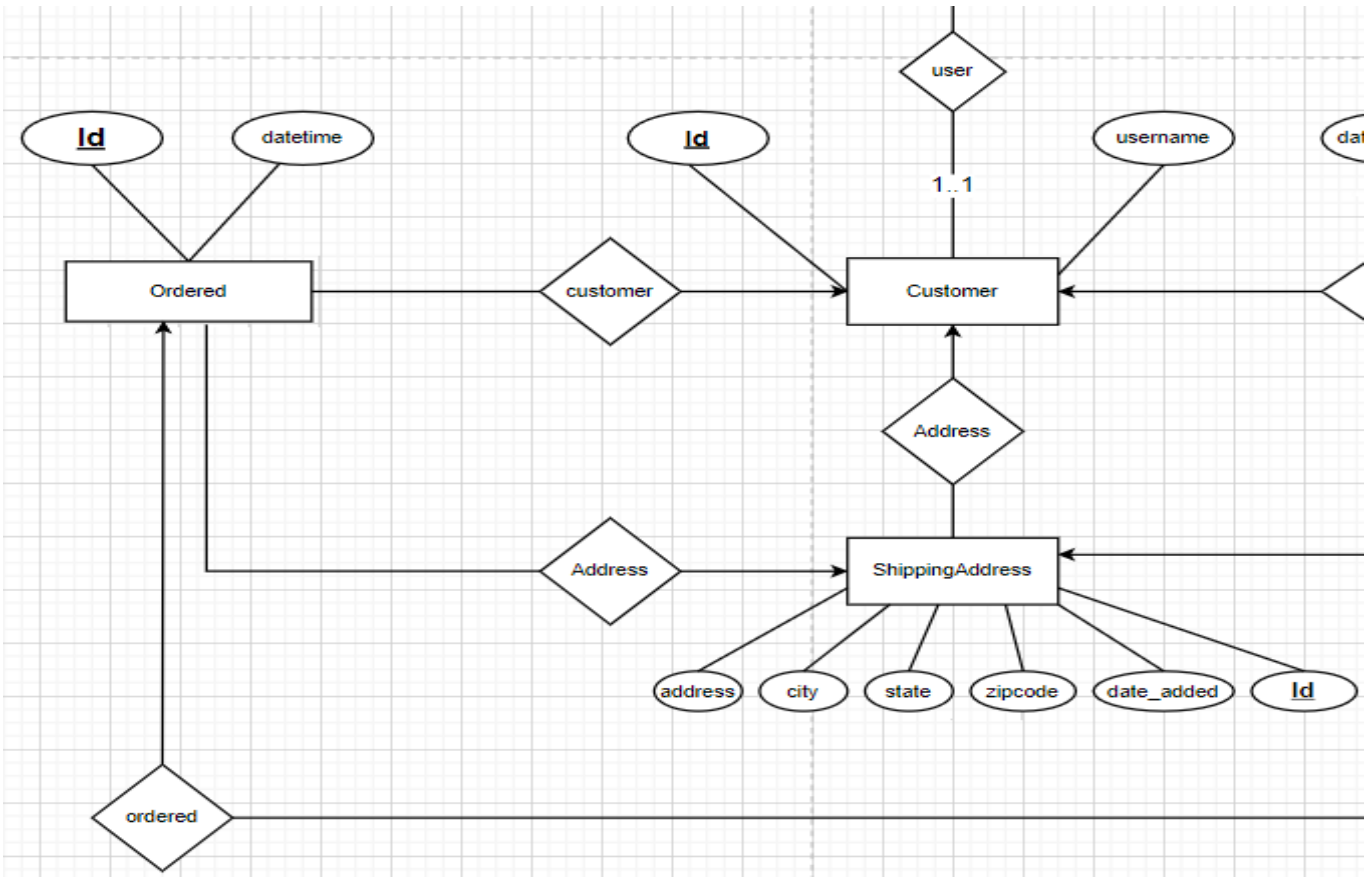
Product-OrderItem Model:



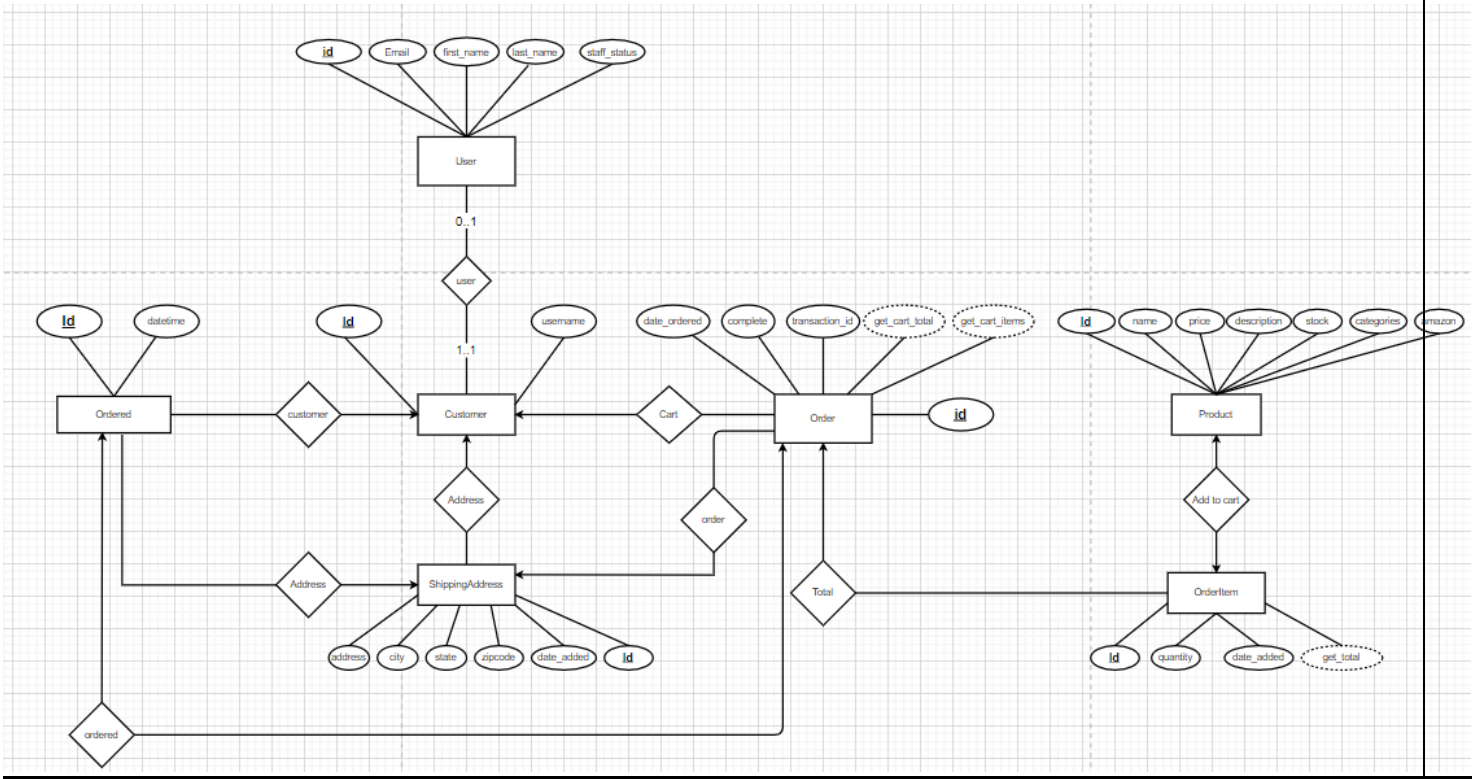
Order-OrderItem Model:



Ordered-Customer-Address-Order Model:



Final Model:


id
Email
first_name
last_name
staff_status
User
0..1
user
1..1
Id
datetime
Ordered
customer
Id
username
Customer
Address
Address
ShippingAddress
address
city
state
zipcode
date_added
Id
ordered
Cart
date_ordered
complete
transaction_id
get_cart_total
get_cart_items
Order
id
order
Total
Id
name
price
description
stock
categories
amazon
Product
Add to cart
OrderItem
Id
quantity
date_added
get_total

Experiment 2

Aim: Mapping ER/EER to Relational schema model.

Hardware and Software Requirement: P-IV and above, Oracle

Theory:

The set of allowed values for each attribute is called the domain of the attribute

Attribute values are (normally) required to be atomic; that is, indivisible

The special value *null* is a member of every domain. Indicated that the value is "unknown"

The null value causes complications in the definition of many operations

$A_1, A_2, \ldots, A_n$ are *attributes*

$R = (A_1, A_2, \ldots, A_n)$ is a *relation schema*

Example:

instructor = (*ID, name, dept_name, salary*)

Formally, given sets D_1, D_2, …. D_n a relation r is a subset of $D_1 \times D_2 \times \ldots \times D_n$

Thus, a relation is a set of n-tuples $(a_1, a_2, \ldots, a_n)$ where each $a_i \in D_i$

The current values (relation instance) of a relation are specified by a table

An element t of r is a *tuple*, represented by a *row* in a table

Primary keys allow entity sets and relationship sets to be expressed uniformly as *relation schemas* that represent the contents of the database.

A database which conforms to an E-R diagram can be represented by a collection of schemas.

For each entity set and relationship set there is a unique schema that is assigned the name of the corresponding entity set or relationship set.

Each schema has a number of columns (generally corresponding to attributes), which have unique names.

Representing Entity Sets as Schemas

A strong entity set reduces to a schema with the same attributes.

A weak entity set becomes a table that includes a column for the primary key of the identifying strong entity set

payment = (*loan_number*, *payment_number*, *payment_date*, *payment_amount*)

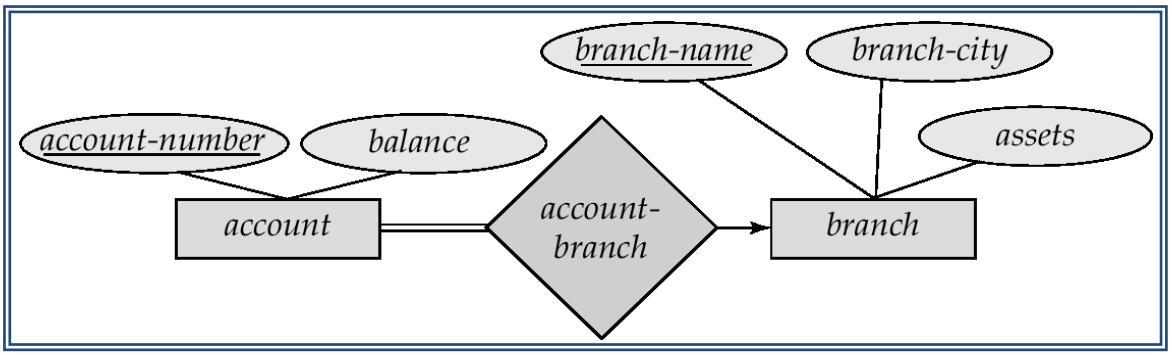
A many-to-many relationship set is represented as a schema with attributes for the primary keys of the two participating entity sets, and any descriptive attributes of the relationship set.

Example: schema for relationship set borrower

borrower = (*customer_id*, *loan_number*)

Many-to-one and one-to-many relationship sets that are total on the many-side can be represented by adding an extra attribute to the "many" side, containing the primary key of the "one" side

Example: Instead of creating a schema for relationship set *account_branch*, add an attribute *branch_name* to the schema arising from entity set *account*



For one-to-one relationship sets, either side can be chosen to act as the "many" side

That is, extra attribute can be added to either of the tables corresponding to the two entity sets

If participation is *partial* on the "many" side, replacing a schema by an extra attribute in the schema corresponding to the "many" side could result in null values

The schema corresponding to a relationship set linking a weak entity set to its identifying strong entity set is redundant.

Example: The *payment* schema already contains the attributes that would appear in the *loan_payment* schema (i.e., *loan_number* and *payment_number*).

Composite attributes are flattened out by creating a separate attribute for each component attribute

Example: given entity set *customer* with composite attribute *name* with component attributes *first_name* and *last_name* the schema corresponding to the entity set has two attributes

name.first_name and *name.last_name*

A multivalued attribute *M* of an entity *E* is represented by a separate schema *EM*

Schema *EM* has attributes corresponding to the primary key of *E* and an attribute corresponding to multivalued attribute *M*

Example: Multivalued attribute *dependent_names* of *employee* is represented by a schema:

employee_dependent_names = (*employee_id*, *dname*)

Each value of the multivalued attribute maps to a separate tuple of the relation on schema *EM*

For example, an employee entity with primary key 123-45-6789 and dependents Jack and Jane maps to two tuples:

(123-45-6789 , Jack) and (123-45-6789 , Jane)

Representing Specialization via Schemas

Form a schema for each entity set with all local and inherited attributes

schema	attributes
person	*name, street, city*
customer	*name, street, city, credit_rating*
employee	*name, street, city, salary*

If specialization is total, the schema for the generalized entity set (*person*) not required to store information

Can be defined as a "view" relation containing union of specialization relations

But explicit schema may still be needed for foreign key constraints

Drawback: *street* and *city* may be stored redundantly for people who are both customers and employees

Schemas Corresponding to Aggregation

To represent aggregation, create a schema containing

primary key of the aggregated relationship,

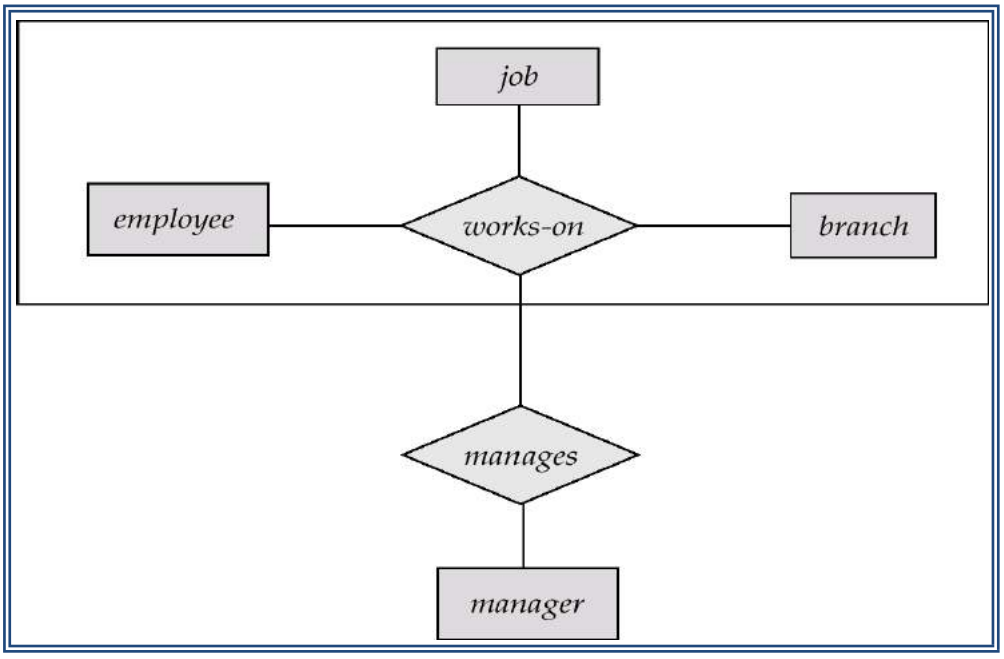
the primary key of the associated entity set

any descriptive attributes

For example, to represent aggregation manages between relationship works_on and entity set manager, create a schema

manages (*employee_id*, *branch_name*, *title*, *manager_name*)

Schema *works_on* is redundant provided we are willing to store null values for attribute *manager_name* in relation on schema *manages*



Conclusion: We have successfully mapped our ER model to Relational Schema.

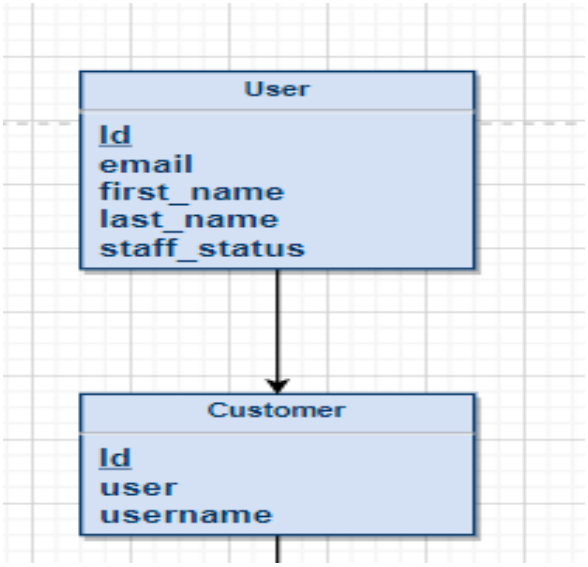
User-Customer Relation:

User = (id, email, first_name, last_name, staff_status)

id, email, first_name,last_name, staff_status all are attributes of User where id is a primary key

Customer = (id, user ,username)

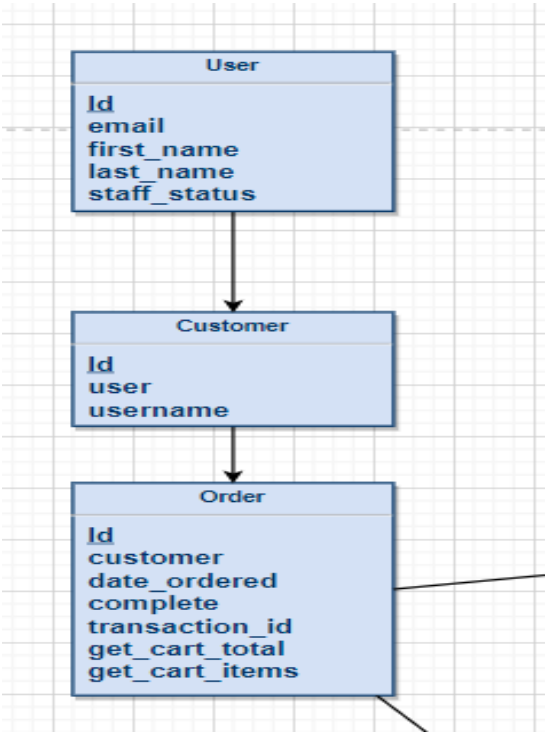
id is a primary key, user is foreign key referencing User model through User primary key(user.id) and username is a normal attribute.



Customer-Order relation:

Order = (id, Customer, date_ordered, complete, transaction_id)

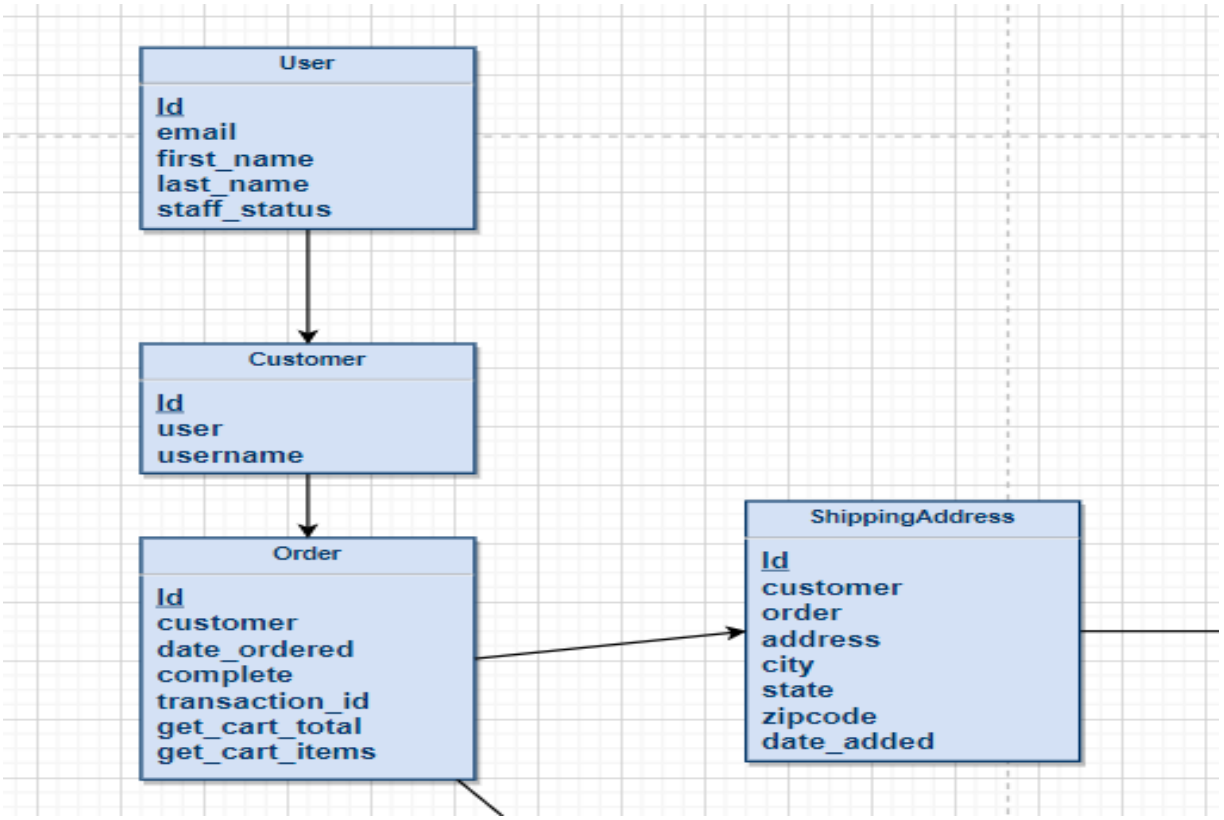
id is a primary key, Customer is a foreign key referencing Customer referencing through Customer primary key (customer.id) and other are normal attributes.



Order-Customer-Shipping Relation:

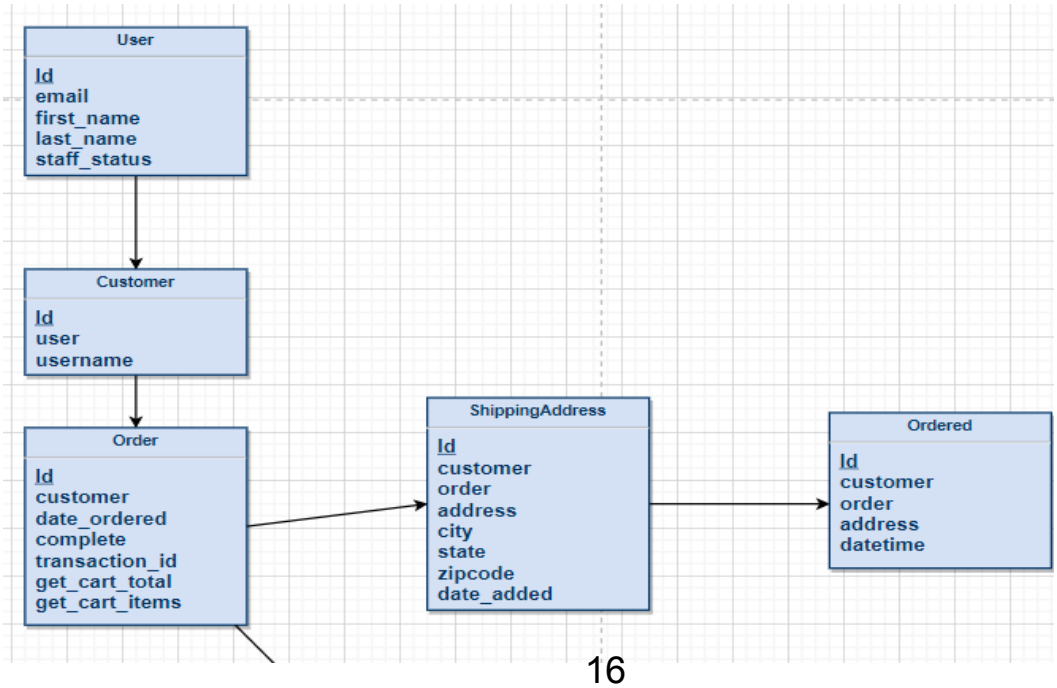
ShippingAddress = (id, customer, order, address, city, state, zip_code, date, added)

id is a primary key, customer and order are foreign keys referencing Customer and Order through their corresponding primary key(customer.id and order.id) and other are normal attribute.



Ordered-Order-Customer-Shipping Relation:

Ordered = id is a primary key, customer, order and address are foreign keys referencing Customer, Order and ShippingAddress through their corresponding primary key(customer.id, order.id and address.id) and other are normal attributes.



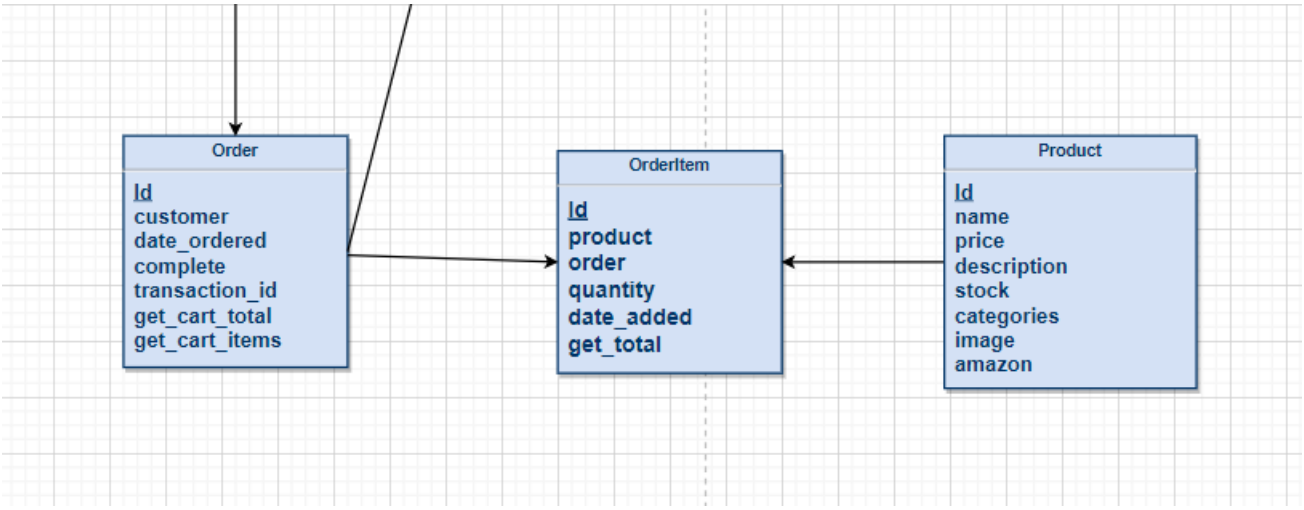
OrderItem-Order-Product Relation:

Product = (id, name, price, description, stock, categories, image, amazon)

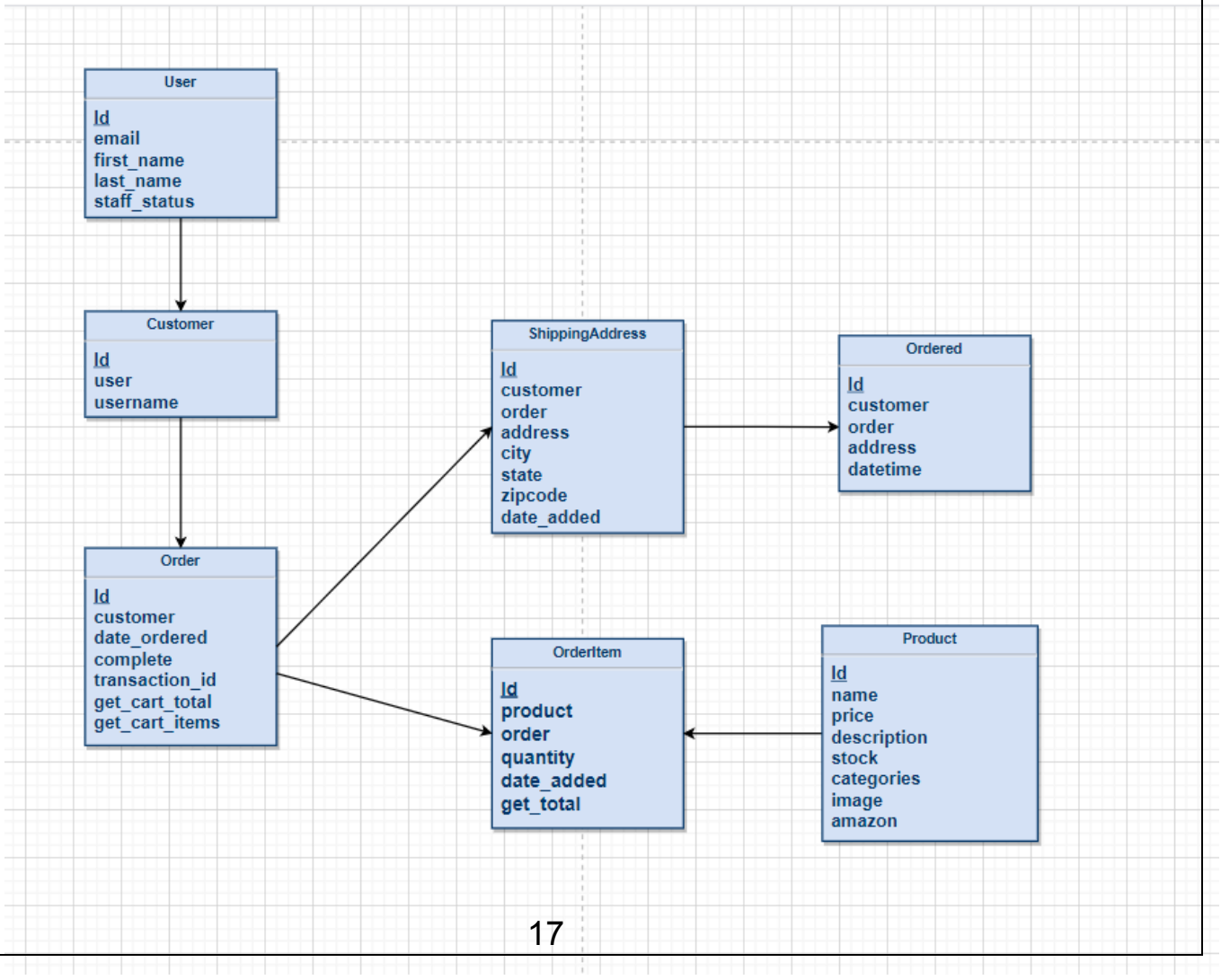
id is a primary key and other are normal attributes.

OrderedItem = (id, product, order, quantity, date_added)

id is a primary key, product and order are foreign keys referencing Product and Order through their corresponding primary key(product.id and order.id) and other are normal attributes.



Final-Relational Schema:



Experiment 3

Aim: Create a database using Data Definition Language (DDL) and apply integrity constraints for the specified System

Hardware and Software Requirement: P-IV and above, Oracle

Theory:

• **Data-definition language** (DDL). The SQL DDL provides commands for defining relation schemas, deleting relations, and modifying relation schemas.

• **Data-manipulation language** (DML). The SQL DML provides the ability to query information from the database and to insert tuples into, delete tuples from, and modify tuples in the database.

SQL Data Definition

The set of relations in a database must be specified to the system by means of a data-definition language (DDL). The SQL DDL allows specification of not only a set of relations, but also information about each relation, including:

• The schema for each relation.

• The types of values associated with each attribute.

• The integrity constraints.

• The set of indices to be maintained for each relation.

•The security and authorization information for each relation.

• The physical storage structure of each relation on disk

Basic Types

The SQL standard supports a variety of built-in types, including:

• **char**(*n*): A fixed-length character string with user-specified length *n*. The full form, **character**, can be used instead.

• **varchar**(*n*): A variable-length character string with user-specified maximum length *n*. The full form, **character varying**, is equivalent.

• **int**: An integer (a finite subset of the integers that is machine dependent). The full form, **integer**, is equivalent.

• **smallint**: A small integer (a machine-dependent subset of the integer type).

• **numeric**(*p*, *d*): A fixed-point number with user-specified precision. The number consists of *p* digits (plus a sign), and *d* of the *p* digits are to the right of the decimal point. Thus, **numeric**(3,1) allows 44.5 to be stored exactly, but neither 444.5 or 0.32 can be stored exactly in a field of this type.

• **real, double precision**: Floating-point and double-precision floating-point numbers with machine-dependent precision.

• **float**(*n*): A floating-point number, with precision of at least *n* digits.

Each type may include a special value called the **null** value. A null value indicates an absent value that may exist but be unknown or that may not exist at all.

Basic Schema Definition

Create Table Construct

An SQL relation is defined using the **create table** command:

create table r (A_1 D_1, A_2 D_2, ..., A_n D_n,
(integrity-constraint$_1$),
...,
(integrity-constraint$_k$))

r is the name of the relation

each A_i is an attribute name in the schema of relation r

D_i is the data type of values in the domain of attribute A_i

Drop and Alter Table Constructs

- The drop table command deletes all information about the dropped relation from the database.
- The alter table command is used to add attributes to an existing relation:
  - alter table r add A D

where A is the name of the attribute to be added to relation r and D is the domain of A.

- All tuples in the relation are assigned *null* as the value for the new attribute.

The alter table command can also be used to drop attributes of a relation:

- alter table r drop A

where A is the name of an attribute of relation r

TRUNCATE: Remove all records from table ,including spaces allocated for the records are removed.

Syntax:

TRUNCATE TABLE <TABLE NAME>;

Integrity Constraints

Integrity constraints guard against accidental damage to the database, by ensuring that authorized changes to the database do not result in a loss of data consistency.

A checking account must have a balance greater than $10,000.00

A salary of a bank employee must be at least $4.00 an hour

A customer must have a (non-null) phone number

Constraints on a Single Relation

- **not null**
- **primary key**
- **unique**
- **check** (*P*), where *P* is a predicate

Not Null Constraint

Declare *branch_name* for *branch* is **not null**

branch_name **char**(15) **not null**

The Unique Constraint

unique (A_1, A_2, …, A_m)

The unique specification states that the attributes

*A*1, *A*2, … *A*m
form a candidate key.

Candidate keys are permitted to be null (in contrast to primary keys).

The check clause

check (*P*), where *P* is a predicate

Example: Declare *branch_name* as the primary key for *branch* and ensure that the values of *assets* are non-negative.

```
create table branch
        (branch_name    char(15),
         branch_city        char(30),
         assets             integer,
         primary key (branch_name),
         check (assets >= 0))
```

Referential Integrity

Ensures that a value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation.

Example: If "Perryridge" is a branch name appearing in one of the tuples in the *account* relation, then there exists a tuple in the *branch* relation for branch "Perryridge".

Primary and candidate keys and foreign keys can be specified as part of the SQL **create table** statement:

The primary key clause lists attributes that comprise the primary key.

The unique key clause lists attributes that comprise a candidate key.

The foreign key clause lists the attributes that comprise the foreign key and the name of the relation referenced by the foreign key. By default, a foreign key references the primary key attributes of the referenced table.

```
create table account
(account_number char(10),
branch_name     char(15),
balance     integer,
primary key (account_number),
foreign key (branch_name) references branch )
```

Conclusion: We have Successfully executed DDL command using SQL Live

SARASWATI College of Engineering

Department of Computer Engineering

Code:

```
USE practical3;
CREATE TABLE clients (
  client_id int(11) NOT NULL,
  name varchar(50) NOT NULL,
  address varchar(50) NOT NULL,
  city varchar(50) NOT NULL,
  state char(2) NOT NULL,
  phone varchar(50) DEFAULT NULL,
  PRIMARY KEY (`client_id`)
);

DESC clients;

INSERT INTO clients VALUES (1,'Adnan','6 Shivajinagar Govandi','Mumbai','MH','315-252-7305');
INSERT INTO clients VALUES (2,'Zeeshan','Ryanpark Gutamnagar Govandi','Mumbai','MH','304-659-1170');
INSERT INTO clients VALUES (3,'Binit','096 Airoli Dombivali','Navi-Mumbai','MH','415-144-6037');

CREATE TABLE invoices (
  invoice_id int(11) NOT NULL,
  number varchar(50) NOT NULL,
  client_id int(11) NOT NULL,
  invoice_total decimal(9,2) NOT NULL,
  payment_total decimal(9,2) NOT NULL DEFAULT '0.00',
  PRIMARY KEY (invoice_id),
  KEY FK_client_id (client_id),
  CONSTRAINT FK_client_id FOREIGN KEY (client_id) REFERENCES clients (client_id) ON DELETE RESTRICT ON UPDATE CASCADE
);


ALTER TABLE invoices ADD invoice_date date NOT NULL;
DESC invoices;

INSERT INTO invoices VALUES (1,'75-587-6626',1,157.78,74.55,'2021-01-29');
INSERT INTO invoices VALUES (2,'68-093-9863',3,133.87,0.00,'2021-02-04');
INSERT INTO invoices VALUES (3,'78-145-1093',1,189.12,0.00,'2021-02-20');
INSERT INTO invoices VALUES (4,'77-593-0081',2,172.17,0.00,'2021-03-17');


SELECT * FROM clients;
SELECT * FROM invoices;

TRUNCATE invoices;
SELECT * FROM invoices;
DROP TABLE invoices;
DROP TABLE clients;
```

Output:

Action Output

#	Time	Action	Message	Duration / Fetch
1	13:46:21	USE practical3	0 row(s) affected	0.000 sec
2	13:46:21	CREATE TABLE clients (client_id int(11) NOT NULL, name varchar(50) NOT NULL, address varchar(50) ...	0 row(s) affected, 1 warning(s): 1681 Integer display width is deprecated and will be removed in a future release.	0.031 sec
3	13:46:21	DESC clients	6 row(s) returned	0.000 sec / 0.000 sec
4	13:46:21	INSERT INTO clients VALUES (1,'Adnan','6 Shivajinagar Govandi','Mumbai','MH','315-252-7305')	1 row(s) affected	0.000 sec
5	13:46:21	INSERT INTO clients VALUES (2,'Zeeshan','Ryanpark Gutamnagar Govandi','Mumbai','MH','304-659-1170')	1 row(s) affected	0.000 sec
6	13:46:21	INSERT INTO clients VALUES (3,'Binit','096 Airoli Dombivali','Navi-Mumbai','MH','415-144-6037')	1 row(s) affected	0.000 sec
7	13:46:21	CREATE TABLE invoices (invoice_id int(11) NOT NULL, number varchar(50) NOT NULL, client_id int(11) ...	0 row(s) affected, 2 warning(s): 1681 Integer display width is deprecated and will be removed in a future release...	0.031 sec
8	13:46:22	ALTER TABLE invoices ADD invoice_date date NOT NULL	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0	0.062 sec
9	13:46:22	DESC invoices	6 row(s) returned	0.000 sec / 0.000 sec
10	13:46:22	INSERT INTO invoices VALUES (1,'75-587-6626',1,157.78,74.55,'2021-01-29')	1 row(s) affected	0.000 sec
11	13:46:22	INSERT INTO invoices VALUES (2,'68-093-9863',3,133.87,0.00,'2021-02-04')	1 row(s) affected	0.015 sec
12	13:46:22	INSERT INTO invoices VALUES (3,'78-145-1093',1,189.12,0.00,'2021-02-20')	1 row(s) affected	0.000 sec
13	13:46:22	INSERT INTO invoices VALUES (4,'77-593-0081',2,172.17,0.00,'2021-03-17')	1 row(s) affected	0.000 sec
14	13:46:22	SELECT * FROM clients LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec
15	13:46:22	SELECT * FROM invoices LIMIT 0, 1000	4 row(s) returned	0.000 sec / 0.000 sec
16	13:46:22	TRUNCATE invoices	0 row(s) affected	0.062 sec
17	13:46:22	SELECT * FROM invoices LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec
18	13:51:30	DROP TABLE invoices	0 row(s) affected	0.031 sec
19	13:51:35	DROP TABLE clients	0 row(s) affected	0.031 sec

Field	Type	Null	Key	Default	Extra
client_id	int	NO	PRI	NULL	
name	varchar(50)	NO		NULL	
address	varchar(50)	NO		NULL	
city	varchar(50)	NO		NULL	
state	char(2)	NO		NULL	
phone	varchar(50)	YES		NULL	

Result Grid | Filter Rows: | Export: | Wrap Cel

Field	Type	Null	Key	Default	Extra
invoice_id	int	NO	PRI	NULL	
number	varchar(50)	NO		NULL	
client_id	int	NO	MUL	NULL	
invoice_total	decimal(9,2)	NO		NULL	
payment_total	decimal(9,2)	NO		0.00	
invoice_date	date	NO		NULL	

Result 1 × | Result 2 | clients 3 | invoices 4 | invoices 5

client_id	name	address	city	state	phone
1	Adnan	6 Shivajinagar Govandi	Mumbai	MH	315-252-7305
2	Zeeshan	Ryanpark Gutamnagar Govandi	Mumbai	MH	304-659-1170
3	Binit	096 Airoli Dombivali	Navi-Mumbai	MH	415-144-6037
NULL	NULL	NULL	NULL	NULL	NULL

Result 1 | Result 2 × | clients 3 | invoices 4 | invoices 5

invoice_id	number	client_id	invoice_total	payment_total	invoice_date
1	75-587-6626	1	157.78	74.55	2021-01-29
2	68-093-9863	3	133.87	0.00	2021-02-04
3	78-145-1093	1	189.12	0.00	2021-02-20
4	77-593-0081	2	172.17	0.00	2021-03-17
NULL	NULL	NULL	NULL	NULL	NULL

Result 1 | Result 2 | clients 3 × | invoices 4 | invoices 5

Result 1 | Result 2 | clients 3 | invoices 4 × | invoices 5

invoice_id	number	client_id	invoice_total	payment_total	invoice_date
NULL	NULL	NULL	NULL	NULL	NULL

Experiment 4

Aim: Perform DML Commands for you're the specified System.

Hardware and Software Requirement: P-IV and above, Oracle

Theory:

DML Commands :Data Manipulation Language statements

-It is the area of SQL that allows changing data within the database.

Examples:

- **INSERT:** Insert data into a table.

Syntax:

INSERT INTO tablename VALUES (value list);

- **UPDATE:** Updates existing data within a table.

Syntax:

*UPDATE t*ablename *SET* column_name =value [*WHERE* condition]

Example: update emp set sal=20000 where empno=7369;

- **DELETE:** Deletes all records from a table

Syntax:

*DELETE FROM t*ablename *WHERE* condition

DQL Commands*:* Data Query Language

-It is the components of SQL stmts that allows getting data from database.

Examples:

- **SELECT:** Retrieve data from database.

SQL is based on set and relational operations with certain modifications and enhancements

A typical SQL query has the form:

select $A_1, A_2, ..., A_n$
from $r_1, r_2, ..., r_m$
where P

A_i represents an attribute

R_i represents a relation

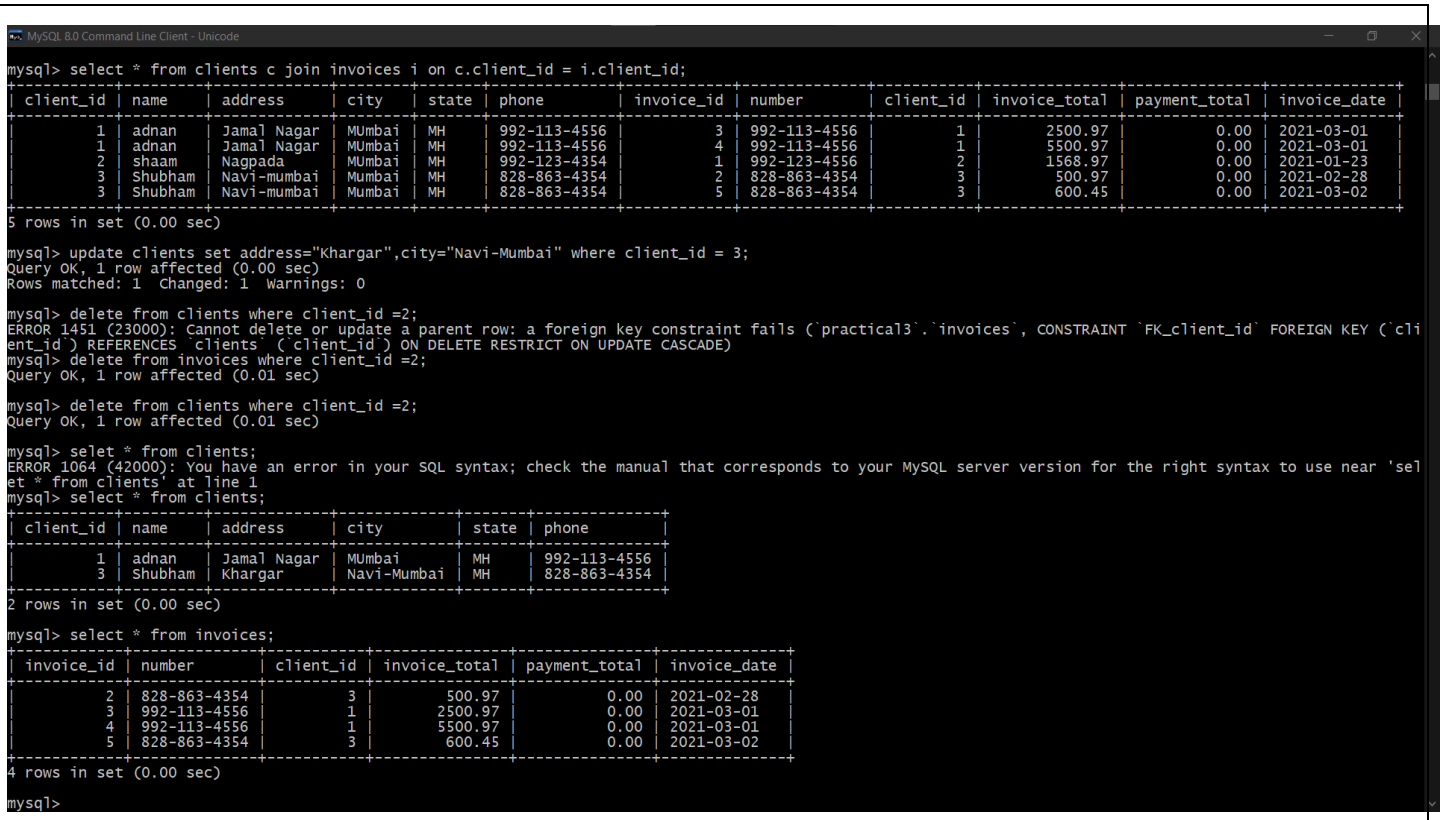
P is a predicate.

SELECT * FROM tablename

Conclusion: We have Successfully used DML commands on our specified system using MYSQL Command Line.

DML command and Output:

```
MySQL 8.0 Command Line Client - Unicode
mysql> insert into clients(address,city,client_id,name,phone,state) values('Jamal Nagar','MUmbai',01,'adnan','992-113-4556','MH');
Query OK, 1 row affected (0.00 sec)

mysql> insert into clients(address,city,client_id,name,phone,state) values('Nagpada','MUmbai',02,'shaam','992-123-4354','MH');
Query OK, 1 row affected (0.01 sec)

mysql> insert into clients(address,city,client_id,name,phone,state) values('Navi-mumbai','Mumbai',03,'Shubham','828-863-4354','MH');
Query OK, 1 row affected (0.01 sec)

mysql> select * from clients;
+-----------+---------+-------------+--------+-------+--------------+
| client_id | name    | address     | city   | state | phone        |
+-----------+---------+-------------+--------+-------+--------------+
|         1 | adnan   | Jamal Nagar | MUmbai | MH    | 992-113-4556 |
|         2 | shaam   | Nagpada     | MUmbai | MH    | 992-123-4354 |
|         3 | Shubham | Navi-mumbai | Mumbai | MH    | 828-863-4354 |
+-----------+---------+-------------+--------+-------+--------------+
3 rows in set (0.00 sec)

mysql> insert into invoices(invoice_id,client_id,number,invoice_total) values(01,02,'992-123-4556',1568.9665);
ERROR 1364 (HY000): Field 'invoice_date' doesn't have a default value
mysql> insert into invoices(invoice_id,client_id,number,invoice_total,invoice_date) values(01,02,'992-123-4556',1568.9665,'2021-01-23');
Query OK, 1 row affected, 1 warning (0.00 sec)

mysql> insert into invoices(invoice_id,client_id,number,invoice_total,invoice_date) values(02,03,'828-863-4354',500.965,'2021-02-28');
Query OK, 1 row affected, 1 warning (0.01 sec)

mysql> insert into invoices(invoice_id,client_id,number,invoice_total,invoice_date) values(03,01,'992-113-4556',2500.965,'2021-03-01');
Query OK, 1 row affected, 1 warning (0.01 sec)

mysql> insert into invoices(invoice_id,client_id,number,invoice_total,invoice_date) values(04,01,'992-113-4556',5500.965,'2021-03-01');
Query OK, 1 row affected, 1 warning (0.01 sec)

mysql> insert into invoices(invoice_id,client_id,number,invoice_total,invoice_date) values(05,03,'828-863-4354',600.4525,'2021-03-02');
Query OK, 1 row affected, 1 warning (0.01 sec)
```


MySQL 8.0 Command Line Client - Unicode
mysql> select * from clients c join invoices i on c.client_id = i.client_id;
+-----------+---------+-------------+--------+-------+--------------+------------+--------------+-----------+---------------+---------------+--------------+
| client_id | name | address | city | state | phone | invoice_id | number | client_id | invoice_total | payment_total | invoice_date |
+-----------+---------+-------------+--------+-------+--------------+------------+--------------+-----------+---------------+---------------+--------------+
1	adnan	Jamal Nagar	MUmbai	MH	992-113-4556	3	992-113-4556	1	2500.97	0.00	2021-03-01
1	adnan	Jamal Nagar	MUmbai	MH	992-113-4556	4	992-113-4556	1	5500.97	0.00	2021-03-01
2	shaam	Nagpada	MUmbai	MH	992-123-4354	1	992-123-4556	2	1568.97	0.00	2021-01-23
3	Shubham	Navi-mumbai	Mumbai	MH	828-863-4354	2	828-863-4354	3	500.97	0.00	2021-02-28
3	Shubham	Navi-mumbai	Mumbai	MH	828-863-4354	5	828-863-4354	3	600.45	0.00	2021-03-02
+-----------+---------+-------------+--------+-------+--------------+------------+--------------+-----------+---------------+---------------+--------------+
5 rows in set (0.00 sec)
mysql> update clients set address="Khargar",city="Navi-Mumbai" where client_id = 3;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0
mysql> delete from clients where client_id =2;
ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key constraint fails (`practical3`.`invoices`, CONSTRAINT `FK_client_id` FOREIGN KEY (`cli
ent_id`) REFERENCES `clients` (`client_id`) ON DELETE RESTRICT ON UPDATE CASCADE)
mysql> delete from invoices where client_id =2;
Query OK, 1 row affected (0.01 sec)
mysql> delete from clients where client_id =2;
Query OK, 1 row affected (0.01 sec)
mysql> selet * from clients;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'sel
et * from clients' at line 1
mysql> select * from clients;
+-----------+---------+-------------+-------------+-------+--------------+
| client_id | name | address | city | state | phone |
+-----------+---------+-------------+-------------+-------+--------------+
| 1 | adnan | Jamal Nagar | MUmbai | MH | 992-113-4556 |
| 3 | Shubham | Khargar | Navi-Mumbai | MH | 828-863-4354 |
+-----------+---------+-------------+-------------+-------+--------------+
2 rows in set (0.00 sec)
mysql> select * from invoices;
+------------+--------------+-----------+---------------+---------------+--------------+
| invoice_id | number | client_id | invoice_total | payment_total | invoice_date |
+------------+--------------+-----------+---------------+---------------+--------------+
2	828-863-4354	3	500.97	0.00	2021-02-28
3	992-113-4556	1	2500.97	0.00	2021-03-01
4	992-113-4556	1	5500.97	0.00	2021-03-01
5	828-863-4354	3	600.45	0.00	2021-03-02
+------------+--------------+-----------+---------------+---------------+--------------+
4 rows in set (0.00 sec)
mysql>

Experiment 5

Aim: Perform string manipulation operations and aggregate functions with group by.. Having clause.

Hardware and Software Requirement: P-IV and above, Oracle

Theory:

1. ***Like Condition:***
   A LIKE condition specifies a test involving pattern matching.

   We describe patterns by using two special characters:

   • Percent (%): The % character matches any substring.

   • Underscore ( ): The character matches any character.

Syntax:

```
SELECT columns
FROM tables
WHERE column1 LIKE '%_';
```

Example:

List the customers whose name begin with the letters''

```
SELECT FName,LName
FROM cust
WHERE FName LIKE 'Ch%';
```

The % indicates that any number of character can follow the letter Ch.

```
select lower(name) || upper(name) from try;
select lower('RINA') from dual;
select upper(name) from try;
select * from try where name like '%mal%'
```

Logical operator:

1. ***OR operator:***

The OR condition allows you to create an SQL statement where records are returned when any one of the conditions are met. It can be used in any valid SQL statement - select, insert, update, or delete.

Syntax:

```
SELECT columns
FROM tables
WHERE column1 = 'value1' or column2 = 'value2'
```

The OR condition requires that any of the conditions be must be met for the record to be included in the result set. In this case, column1 has to equal 'value1' OR column2 has to equal 'value2'.

Example:

```
SELECT *
FROM suppliers
WHERE city = 'New York'
or city = 'Newark';
```

This would return all suppliers that reside in either New York or Newark. Because the * is used in the select, all fields from the suppliers table would appear in the result set.

2. ***AND Operator:***

The AND operator displays a record if both the first condition and the second condition is true.

Syntax:

```
SELECT columns
FROM tables
WHERE column1 = 'value1' and column2 = 'value2'
```

Example:

```
SELECT *
FROM  EMP
WHERE EmpTown = 'London' AND EmpAge > 30
```

3. ***NOT Operator:***

If you want to find rows that do not satisfy a condition, you can use the logical operator, NOT. NOT results in the reverse of a condition. That is, if a condition is satisfied, then the row is not returned.

Example:

If you want to find out the names of the students who do not play football, the query would be like:

```
SELECT first_name, last_name, games
FROM student_details
WHERE NOT games = 'Football' ;
```

- ***ORDER BY clause: Sorting of data in table***

The ORDER BY keyword is used to sort the result-set by a specified column.

The ORDER BY keyword sort the records in ascending order by default.

If you want to sort the records in a descending order, you can use the DESC keyword

Syntax:

```
SELECT "column_name"
FROM "table_name"
[WHERE "condition"]
ORDER BY "column_name" [ASC, DESC];
```

Aggregate functions

Aggregate functions return a single result row based on groups of rows, rather than on single rows.

avg: average value
min: minimum value
max: maximum value
sum: sum of values

count: number of values

Find the average account balance at the Perryridge branch

select avg *(balance)*

from *account*

where *branch_name* = 'Perryridge'

Find the number of tuples in the

select count (*)
from *customer*

Find the number of depositors in the bank

select count (distinct *customer_name)*

from *depositor*

Aggregate Functions – Group By

Find the number of depositors for each branch.

select *branch_name,* **count (distinct**

customer_name)

from *depositor, account*

where *depositor.account_number = account.account_number*

group by *branch_name*

Find the names of all branches where the average account balance is more than 1,200.

select *branch_name*, **avg** (*balance*)
from *account*

group by *branch_name*

having avg (*balance*) > 1200

predicates in the **having** clause are applied after the formation of groups whereas predicates in the **where** clause are applied before forming groups

select count(ename),dno from emp group by dno;

Conclusion: We have Successfully Performed string manipulation operations and aggregate functions with group by.. Having clause using SQL Command Line Client.

Code and Output:

MySQL 8.0 Command Line Client

```
mysql> SELECT * FROM employees WHERE last_name REGEXP '^al|si|an$' ORDER BY last_name;
+-------------+------------+-----------+-----------------------------+--------+------------+------------+
| employee_id | first_name | last_name | job_title                   | salary | reports_to | office_id |
+-------------+------------+-----------+-----------------------------+--------+------------+------------+
|       56274 | Keriann    | Alloisi   | VP Marketing                | 110150 |      37270 |          1 |
|       40448 | Mindy      | Crissil   | Staff Scientist             |  94860 |      37270 |          1 |
|       98374 | Estrellita | Daleman   | Staff Accountant IV         |  70187 |      37270 |          5 |
|       72913 | Kass       | Hefferan  | Computer Systems Analyst IV |  96401 |      37270 |          3 |
|       67370 | Elladine   | Rising    | Social Worker               |  96767 |      37270 |          2 |
+-------------+------------+-----------+-----------------------------+--------+------------+------------+
5 rows in set (0.00 sec)

mysql> SELECT * FROM employees where first_name LIKE '%ri%';
+-------------+------------+-----------+-----------------+--------+------------+------------+
| employee_id | first_name | last_name | job_title       | salary | reports_to | office_id |
+-------------+------------+-----------+-----------------+--------+------------+------------+
|       56274 | Keriann    | Alloisi   | VP Marketing    | 110150 |      37270 |          1 |
|       76196 | Mirilla    | Janowski  | Cost Accountant | 119241 |      37270 |          3 |
|       80679 | Mildrid    | Sokale    | Geologist II    |  67987 |      37270 |          4 |
+-------------+------------+-----------+-----------------+--------+------------+------------+
3 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM employees where first_name LIKE '%a%' and (salary >50000 and salary< 95000);
+-------------+------------+-----------+--------------------+--------+------------+-----------+
| employee_id | first_name | last_name | job_title          | salary | reports_to | office_id |
+-------------+------------+-----------+--------------------+--------+------------+-----------+
|       33391 | D'arcy     | Nortunen  | Account Executive  |  62871 |      37270 |         1 |
|       37270 | Yovonnda   | Magrannell| Executive Secretary|  63996 |       NULL |        10 |
|       84791 | Hazel      | Tarbert   | General Manager    |  93760 |      37270 |         4 |
|       98374 | Estrellita | Daleman   | Staff Accountant IV|  70187 |      37270 |         5 |
+-------------+------------+-----------+--------------------+--------+------------+-----------+
4 rows in set (0.00 sec)

mysql> SELECT * FROM employees e JOIN offices o ON e.office_id= o.office_id and (o.state = 'NY' or o.state='MH') ;
+-------------+------------+-----------+----------------------+--------+------------+-----------+-----------+-----------------+--------------+-------+
| employee_id | first_name | last_name | job_title            | salary | reports_to | office_id | office_id | address         | city         | state |
+-------------+------------+-----------+----------------------+--------+------------+-----------+-----------+-----------------+--------------+-------+
|       63196 | Alaster    | Scutchin  | Assistant Professor  |  32179 |      37270 |         2 |         2 | 5507 Becker Terrace | New York City | NY    |
|       67009 | North      | de Clerc  | VP Product Management| 114257 |      37270 |         2 |         2 | 5507 Becker Terrace | New York City | NY    |
|       67370 | Elladine   | Rising    | Social Worker        |  96767 |      37270 |         2 |         2 | 5507 Becker Terrace | New York City | NY    |
|       68249 | Nisse      | Voysey    | Financial Advisor    |  52832 |      37270 |         2 |         2 | 5507 Becker Terrace | New York City | NY    |
+-------------+------------+-----------+----------------------+--------+------------+-----------+-----------+-----------------+--------------+-------+
4 rows in set (0.01 sec)
```

MySQL 8.0 Command Line Client

```
mysql> SELECT * FROM employees e JOIN offices o ON  e.office_id= o.office_id and NOT (o.state = 'NY' or o.state='MH') ;
+-------------+------------+-----------+----------------------------+--------+------------+-----------+-----------+-----------------+--------------+-------+
| employee_id | first_name | last_name | job_title                  | salary | reports_to | office_id | office_id | address         | city         | state |
+-------------+------------+-----------+----------------------------+--------+------------+-----------+-----------+-----------------+--------------+-------+
|       33391 | D'arcy     | Nortunen  | Account Executive          |  62871 |      37270 |         1 |         1 | 03 Reinke Trail | Cincinnati   | OH    |
|       37851 | Sayer      | Matterson | Statistician III           |  98926 |      37270 |         1 |         1 | 03 Reinke Trail | Cincinnati   | OH    |
|       40448 | Mindy      | Crissil   | Staff Scientist            |  94860 |      37270 |         1 |         1 | 03 Reinke Trail | Cincinnati   | OH    |
|       56274 | Keriann    | Alloisi   | VP Marketing               | 110150 |      37270 |         1 |         1 | 03 Reinke Trail | Cincinnati   | OH    |
|       72540 | Guthrey    | Iacopetti | Office Assistant I         | 117690 |      37270 |         3 |         3 | 54 Northland Court | Richmond  | VA    |
|       72913 | Kass       | Hefferan  | Computer Systems Analyst IV|  96401 |      37270 |         3 |         3 | 54 Northland Court | Richmond  | VA    |
|       75900 | Virge      | Goodrum   | Information Systems Manager|  54578 |      37270 |         3 |         3 | 54 Northland Court | Richmond  | VA    |
|       76196 | Mirilla    | Janowski  | Cost Accountant            | 119241 |      37270 |         3 |         3 | 54 Northland Court | Richmond  | VA    |
|       80529 | Lynde      | Aronson   | Junior Executive           |  77182 |      37270 |         4 |         4 | 08 South Crossing | Cincinnati | OH    |
|       80679 | Mildrid    | Sokale    | Geologist II               |  67987 |      37270 |         4 |         4 | 08 South Crossing | Cincinnati | OH    |
|       84791 | Hazel      | Tarbert   | General Manager            |  93760 |      37270 |         4 |         4 | 08 South Crossing | Cincinnati | OH    |
|       95213 | Cole       | Kesterton | Pharmacist                 |  86119 |      37270 |         4 |         4 | 08 South Crossing | Cincinnati | OH    |
|       96513 | Theresa    | Binney    | Food Chemist               |  47354 |      37270 |         5 |         5 | 553 Maple Drive | Minneapolis  | MN    |
|       98374 | Estrellita | Daleman   | Staff Accountant IV        |  70187 |      37270 |         5 |         5 | 553 Maple Drive | Minneapolis  | MN    |
|      115357 | Ivy        | Fearey    | Structural Engineer        |  92710 |      37270 |         5 |         5 | 553 Maple Drive | Minneapolis  | MN    |
|       37270 | Yovonnda   | Magrannell| Executive Secretary        |  63996 |       NULL |        10 |        10 | 4 Bluestem Parkway | Savannah  | GA    |
+-------------+------------+-----------+----------------------------+--------+------------+-----------+-----------+-----------------+--------------+-------+
16 rows in set (0.00 sec)

mysql> SELECT upper(first_name) AS NAME FROM employees WHERE salary > (SELECT DISTINCT(AVG(salary)) FROM employees );
+----------+
| NAME     |
+----------+
| SAYER    |
| MINDY    |
| KERIANN  |
| NORTH    |
| ELLADINE |
| GUTHREY  |
| KASS     |
| MIRILLA  |
| HAZEL    |
| COLE     |
| IVY      |
+----------+
11 rows in set (0.00 sec)
```

```
mysql> SELECT LOWER(first_name) FROM employees WHERE first_name IN ('virge','kass');
+-------------------+
| LOWER(first_name) |
+-------------------+
| kass              |
| virge             |
+-------------------+
2 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM employees e JOIN offices o ON e.office_id= o.office_id GROUP BY o.city ;
+-------------+------------+-----------+--------------------+--------+------------+-----------+-----------+--------------------+--------------+-------+
| employee_id | first_name | last_name | job_title          | salary | reports_to | office_id | office_id | address            | city         | state |
+-------------+------------+-----------+--------------------+--------+------------+-----------+-----------+--------------------+--------------+-------+
|       33391 | D'arcy     | Nortunen  | Account Executive  |  62871 |      37270 |         1 |         1 | 03 Reinke Trail    | Cincinnati   | OH    |
|       37270 | Yovonnda   | Magrannell| Executive Secretary|  63996 |       NULL |        10 |        10 | 4 Bluestem Parkway | Savannah     | GA    |
|       63196 | Alaster    | Scutchin  | Assistant Professor|  32179 |      37270 |         2 |         2 | 5507 Becker Terrace| New York City| NY    |
|       72540 | Guthrey    | Iacopetti | Office Assistant I | 117690 |      37270 |         3 |         3 | 54 Northland Court | Richmond     | VA    |
|       96513 | Theresa    | Binney    | Food Chemist       |  47354 |      37270 |         5 |         5 | 553 Maple Drive    | Minneapolis  | MN    |
+-------------+------------+-----------+--------------------+--------+------------+-----------+-----------+--------------------+--------------+-------+
5 rows in set (0.00 sec)

mysql> SELECT * FROM employees e JOIN offices o ON e.office_id= o.office_id GROUP BY o.city having salary > avg(e.salary);
+-------------+------------+-----------+--------------------+--------+------------+-----------+-----------+--------------------+----------+-------+
| employee_id | first_name | last_name | job_title          | salary | reports_to | office_id | office_id | address            | city     | state |
+-------------+------------+-----------+--------------------+--------+------------+-----------+-----------+--------------------+----------+-------+
|       72540 | Guthrey    | Iacopetti | Office Assistant I | 117690 |      37270 |         3 |         3 | 54 Northland Court | Richmond | VA    |
+-------------+------------+-----------+--------------------+--------+------------+-----------+-----------+--------------------+----------+-------+
1 row in set (0.00 sec)
```

```
mysql> SELECT SUM(salary) AS sum_of_salary,MAX(salary) AS max_salary,MIN(salary) AS min_salary,AVG(salary) AS average_salary FROM employees;
+---------------+------------+------------+----------------+
| sum_of_salary | max_salary | min_salary | average_salary |
+---------------+------------+------------+----------------+
|       1650047 |     119241 |      32179 |     82502.3500 |
+---------------+------------+------------+----------------+
1 row in set (0.00 sec)
```

Experiment 6

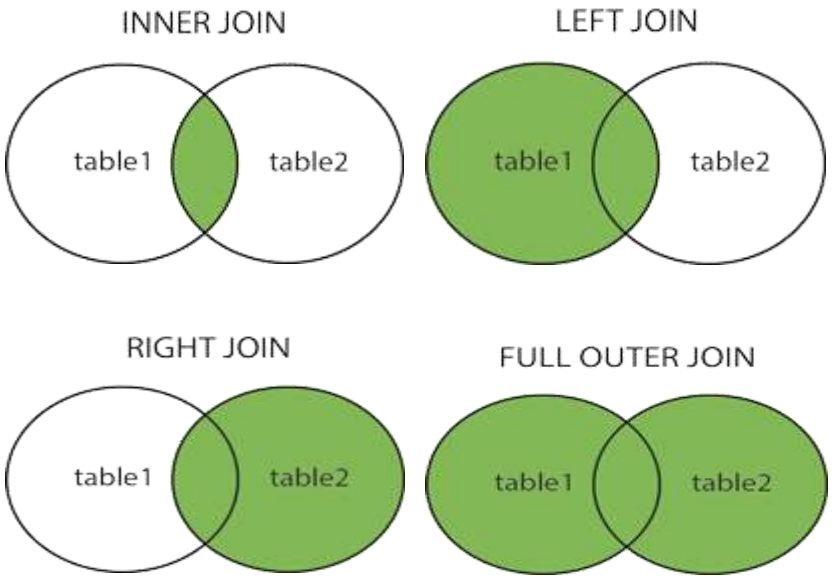
Aim: To Perform Join operations

Resources required: P-IV and above, notepad, browser, stylus studio

Theory:

Different types of the JOINs in SQL:

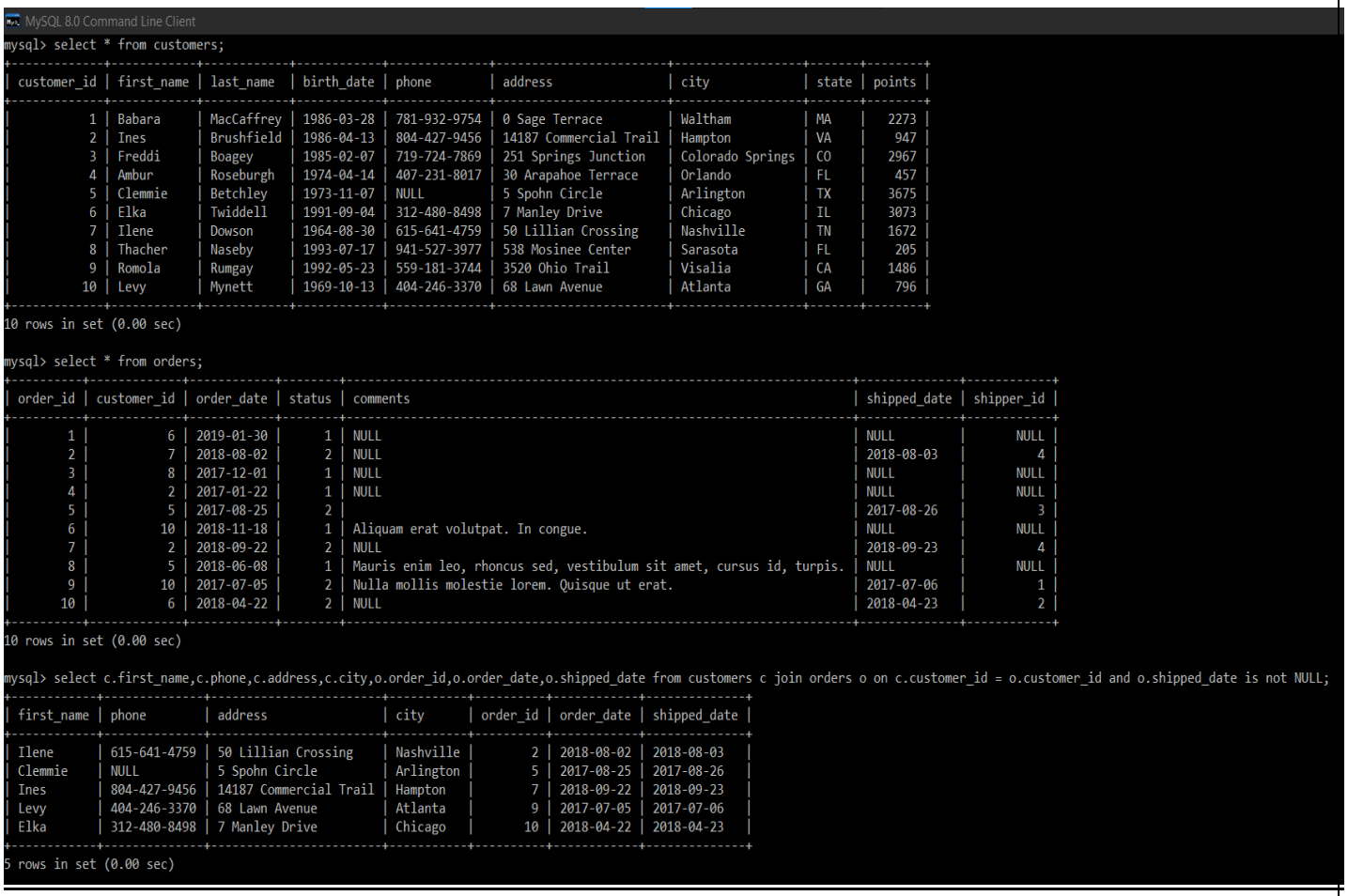
- **(INNER) JOIN**: Returns records that have matching values in both tables
- **LEFT (OUTER) JOIN**: Return all records from the left table, and the matched records from the right table
- **RIGHT (OUTER) JOIN**: Return all records from the right table, and the matched records from the left table
- **FULL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



select master.rno, demo.rno, demo.name from master right outer join demo on master.rno=demo.rno;

Conclusion: We have successfully performed various join operations using MySQL command line.

Code and Output:



```
MySQL 8.0 Command Line Client
mysql> select * from customers;
+-------------+------------+-----------+------------+--------------+-----------------------+-----------------+-------+--------+
| customer_id | first_name | last_name | birth_date | phone        | address               | city            | state | points |
+-------------+------------+-----------+------------+--------------+-----------------------+-----------------+-------+--------+
|           1 | Babara     | MacCaffrey | 1986-03-28 | 781-932-9754 | 0 Sage Terrace        | Waltham         | MA    |   2273 |
|           2 | Ines       | Brushfield | 1986-04-13 | 804-427-9456 | 14187 Commercial Trail | Hampton         | VA    |    947 |
|           3 | Freddi     | Boagey     | 1985-02-07 | 719-724-7869 | 251 Springs Junction  | Colorado Springs | CO    |   2967 |
|           4 | Ambur      | Roseburgh  | 1974-04-14 | 407-231-8017 | 30 Arapahoe Terrace   | Orlando         | FL    |    457 |
|           5 | Clemmie    | Betchley   | 1973-11-07 | NULL         | 5 Spohn Circle        | Arlington       | TX    |   3675 |
|           6 | Elka       | Twiddell   | 1991-09-04 | 312-480-8498 | 7 Manley Drive        | Chicago         | IL    |   3073 |
|           7 | Ilene      | Dowson     | 1964-08-30 | 615-641-4759 | 50 Lillian Crossing   | Nashville       | TN    |   1672 |
|           8 | Thacher    | Naseby     | 1993-07-17 | 941-527-3977 | 538 Mosinee Center    | Sarasota        | FL    |    205 |
|           9 | Romola     | Rumgay     | 1992-05-23 | 559-181-3744 | 3520 Ohio Trail       | Visalia         | CA    |   1486 |
|          10 | Levy       | Mynett     | 1969-10-13 | 404-246-3370 | 68 Lawn Avenue        | Atlanta         | GA    |    796 |
+-------------+------------+-----------+------------+--------------+-----------------------+-----------------+-------+--------+
10 rows in set (0.00 sec)

mysql> select * from orders;
+----------+-------------+------------+--------+-----------------------------------------------------------------------+--------------+------------+
| order_id | customer_id | order_date | status | comments                                                              | shipped_date | shipper_id |
+----------+-------------+------------+--------+-----------------------------------------------------------------------+--------------+------------+
|        1 |           6 | 2019-01-30 |      1 | NULL                                                                  | NULL         |       NULL |
|        2 |           7 | 2018-08-02 |      2 | NULL                                                                  | 2018-08-03   |          4 |
|        3 |           8 | 2017-12-01 |      1 | NULL                                                                  | NULL         |       NULL |
|        4 |           2 | 2017-01-22 |      1 | NULL                                                                  | NULL         |       NULL |
|        5 |           5 | 2017-08-25 |      2 |                                                                       | 2017-08-26   |          3 |
|        6 |          10 | 2018-11-18 |      1 | Aliquam erat volutpat. In congue.                                     | NULL         |       NULL |
|        7 |           2 | 2018-09-22 |      2 | NULL                                                                  | 2018-09-23   |          4 |
|        8 |           5 | 2018-06-08 |      1 | Mauris enim leo, rhoncus sed, vestibulum sit amet, cursus id, turpis. | NULL         |       NULL |
|        9 |          10 | 2017-07-05 |      2 | Nulla mollis molestie lorem. Quisque ut erat.                         | 2017-07-06   |          1 |
|       10 |           6 | 2018-04-22 |      2 | NULL                                                                  | 2018-04-23   |          2 |
+----------+-------------+------------+--------+-----------------------------------------------------------------------+--------------+------------+
10 rows in set (0.00 sec)

mysql> select c.first_name,c.phone,c.address,c.city,o.order_id,o.order_date,o.shipped_date from customers c join orders o on c.customer_id = o.customer_id and o.shipped_date is not NULL;
+------------+--------------+-----------------------+----------+----------+------------+--------------+
| first_name | phone        | address               | city     | order_id | order_date | shipped_date |
+------------+--------------+-----------------------+----------+----------+------------+--------------+
| Ilene      | 615-641-4759 | 50 Lillian Crossing   | Nashville |        2 | 2018-08-02 | 2018-08-03   |
| Clemmie    | NULL         | 5 Spohn Circle        | Arlington |        5 | 2017-08-25 | 2017-08-26   |
| Ines       | 804-427-9456 | 14187 Commercial Trail | Hampton  |        7 | 2018-09-22 | 2018-09-23   |
| Levy       | 404-246-3370 | 68 Lawn Avenue        | Atlanta  |        9 | 2017-07-05 | 2017-07-06   |
| Elka       | 312-480-8498 | 7 Manley Drive        | Chicago  |       10 | 2018-04-22 | 2018-04-23   |
+------------+--------------+-----------------------+----------+----------+------------+--------------+
5 rows in set (0.00 sec)
```

```
MySQL 8.0 Command Line Client

mysql> select c.customer_id,c.first_name as name,o.order_id from customers c left join orders o on c.customer_id = o.customer_id;
+-------------+---------+----------+
| customer_id | name    | order_id |
+-------------+---------+----------+
|           1 | Babara  |     NULL |
|           2 | Ines    |        4 |
|           2 | Ines    |        7 |
|           3 | Freddi  |     NULL |
|           4 | Ambur   |     NULL |
|           5 | Clemmie |        5 |
|           5 | Clemmie |        8 |
|           6 | Elka    |        1 |
|           6 | Elka    |       10 |
|           7 | Ilene   |        2 |
|           8 | Thacher |        3 |
|           9 | Romola  |     NULL |
|          10 | Levy    |        6 |
|          10 | Levy    |        9 |
+-------------+---------+----------+
14 rows in set (0.01 sec)
```

MySQL 8.0 Command Line Client

```
mysql> select * from employees;
+-------------+------------+------------+-----------------------------+--------+------------+-----------+
| employee_id | first_name | last_name  | job_title                   | salary | reports_to | office_id |
+-------------+------------+------------+-----------------------------+--------+------------+-----------+
|       33391 | D'arcy     | Nortunen   | Account Executive           |  62871 |      37270 |         1 |
|       37270 | Yovonnda   | Magrannell | Executive Secretary         |  63996 |       NULL |        10 |
|       37851 | Sayer      | Matterson  | Statistician III            |  98926 |      37270 |         1 |
|       40448 | Mindy      | Crissil    | Staff Scientist             |  94860 |      37270 |         1 |
|       56274 | Keriann    | Alloisi    | VP Marketing                | 110150 |      37270 |         1 |
|       63196 | Alaster    | Scutchin   | Assistant Professor         |  32179 |      37270 |         2 |
|       67009 | North      | de Clerc   | VP Product Management       | 114257 |      37270 |         2 |
|       67370 | Elladine   | Rising     | Social Worker               |  96767 |      37270 |         2 |
|       68249 | Nisse      | Voysey     | Financial Advisor           |  52832 |      37270 |         2 |
|       72540 | Guthrey    | Iacopetti  | Office Assistant I          | 117690 |      37270 |         3 |
|       72913 | Kass       | Hefferan   | Computer Systems Analyst IV |  96401 |      37270 |         3 |
|       75900 | Virge      | Goodrum    | Information Systems Manager |  54578 |      37270 |         3 |
|       76196 | Mirilla    | Janowski   | Cost Accountant             | 119241 |      37270 |         3 |
|       80529 | Lynde      | Aronson    | Junior Executive            |  77182 |      37270 |         4 |
|       80679 | Mildrid    | Sokale     | Geologist II                |  67987 |      37270 |         4 |
|       84791 | Hazel      | Tarbert    | General Manager             |  93760 |      37270 |         4 |
|       95213 | Cole       | Kesterton  | Pharmacist                  |  86119 |      37270 |         4 |
|       96513 | Theresa    | Binney     | Food Chemist                |  47354 |      37270 |         5 |
|       98374 | Estrellita | Daleman    | Staff Accountant IV         |  70187 |      37270 |         5 |
|      115357 | Ivy        | Fearey     | Structural Engineer         |  92710 |      37270 |         5 |
+-------------+------------+------------+-----------------------------+--------+------------+-----------+
20 rows in set (0.00 sec)

mysql> select e.first_name as "Employee Name", m.first_name as "Manager Name"  from employees e join employees m on e.reports_to = m.employee_id;
+---------------+--------------+
| Employee Name | Manager Name |
+---------------+--------------+
| D'arcy        | Yovonnda     |
| Sayer         | Yovonnda     |
| Mindy         | Yovonnda     |
| Keriann       | Yovonnda     |
| Alaster       | Yovonnda     |
| North         | Yovonnda     |
| Elladine      | Yovonnda     |
| Nisse         | Yovonnda     |
| Guthrey       | Yovonnda     |
| Kass          | Yovonnda     |
| Virge         | Yovonnda     |
| Mirilla       | Yovonnda     |
| Lynde         | Yovonnda     |
| Mildrid       | Yovonnda     |
| Hazel         | Yovonnda     |
| Cole          | Yovonnda     |
| Theresa       | Yovonnda     |
| Estrellita    | Yovonnda     |
| Ivy           | Yovonnda     |
```

MySQL 8.0 Command Line Client

```
mysql> select * from order_items;
+----------+------------+----------+------------+
| order_id | product_id | quantity | unit_price |
+----------+------------+----------+------------+
|        1 |          4 |        4 |       3.74 |
|        2 |          1 |        2 |       9.10 |
|        2 |          4 |        4 |       1.66 |
|        2 |          6 |        2 |       2.94 |
|        3 |          3 |       10 |       9.12 |
|        4 |          3 |        7 |       6.99 |
|        4 |         10 |        7 |       6.40 |
|        5 |          2 |        3 |       9.89 |
|        6 |          1 |        4 |       8.65 |
|        6 |          2 |        4 |       3.28 |
|        6 |          3 |        4 |       7.46 |
|        6 |          5 |        1 |       3.45 |
|        7 |          3 |        7 |       9.17 |
|        8 |          5 |        2 |       6.94 |
|        8 |          8 |        2 |       8.59 |
|        9 |          6 |        5 |       7.28 |
|       10 |          1 |       10 |       6.01 |
|       10 |          9 |        9 |       4.28 |
+----------+------------+----------+------------+
18 rows in set (0.01 sec)

mysql> select * from products;
+------------+-----------------------------+------------------+------------+
| product_id | name                        | quantity_in_stock | unit_price |
+------------+-----------------------------+------------------+------------+
|          1 | Foam Dinner Plate           |               70 |       1.21 |
|          2 | Pork - Bacon,back Peameal   |               49 |       4.65 |
|          3 | Lettuce - Romaine, Heart    |               38 |       3.35 |
|          4 | Brocolinni - Gaylan, Chinese |               90 |       4.53 |
|          5 | Sauce - Ranch Dressing      |               94 |       1.63 |
|          6 | Petit Baguette              |               14 |       2.39 |
|          7 | Sweet Pea Sprouts           |               98 |       3.29 |
|          8 | Island Oasis - Raspberry    |               26 |       0.74 |
|          9 | Longan                      |               67 |       2.26 |
|         10 | Broom - Push                |                6 |       1.09 |
+------------+-----------------------------+------------------+------------+
10 rows in set (0.00 sec)
```

MySQL 8.0 Command Line Client

```
mysql> select p.product_id,p.name,o.quantity from order_items o right join products p on p.product_id = o.product_id;
+------------+------------------------------+----------+
| product_id | name                         | quantity |
+------------+------------------------------+----------+
|          1 | Foam Dinner Plate            |        2 |
|          1 | Foam Dinner Plate            |        4 |
|          1 | Foam Dinner Plate            |       10 |
|          2 | Pork - Bacon,back Peameal    |        3 |
|          2 | Pork - Bacon,back Peameal    |        4 |
|          3 | Lettuce - Romaine, Heart     |       10 |
|          3 | Lettuce - Romaine, Heart     |        7 |
|          3 | Lettuce - Romaine, Heart     |        4 |
|          3 | Lettuce - Romaine, Heart     |        7 |
|          4 | Brocolinni - Gaylan, Chinese |        4 |
|          4 | Brocolinni - Gaylan, Chinese |        4 |
|          5 | Sauce - Ranch Dressing       |        1 |
|          5 | Sauce - Ranch Dressing       |        2 |
|          6 | Petit Baguette               |        2 |
|          6 | Petit Baguette               |        5 |
|          7 | Sweet Pea Sprouts            |     NULL |
|          8 | Island Oasis - Raspberry     |        2 |
|          9 | Longan                       |        9 |
|         10 | Broom - Push                 |        7 |
+------------+------------------------------+----------+
19 rows in set (0.00 sec)
```

MySQL 8.0 Command Line Client

```
mysql> show tables;
+--------------------+
| Tables_in_sql_store |
+--------------------+
| customers          |
| order_item_notes   |
| order_items        |
| order_statuses     |
| orders             |
| products           |
| shippers           |
+--------------------+
7 rows in set (0.00 sec)

mysql> select c.first_name as Name,o.order_id,o.order_date,sh.name as shipper,os.name from customers c join orders o on c.customer_id = o.customer_id left join shippers sh on o.shipper_id = sh.shipper_id join or
der_statuses os on o.status = os.order_status_id;
+--------+----------+------------+----------------------+-----------+
| Name   | order_id | order_date | shipper              | name      |
+--------+----------+------------+----------------------+-----------+
| Elka   |        1 | 2019-01-30 | NULL                 | Processed |
| Thacher|        3 | 2017-12-01 | NULL                 | Processed |
| Ines   |        4 | 2017-01-22 | NULL                 | Processed |
| Levy   |        6 | 2018-11-18 | NULL                 | Processed |
| Clemmie|        8 | 2018-06-08 | NULL                 | Processed |
| Ilene  |        2 | 2018-08-02 | Mraz, Renner and Nolan | Shipped |
| Clemmie|        5 | 2017-08-25 | Satterfield LLC      | Shipped   |
| Ines   |        7 | 2018-09-22 | Mraz, Renner and Nolan | Shipped |
| Levy   |        9 | 2017-07-05 | Hettinger LLC        | Shipped   |
| Elka   |       10 | 2018-04-22 | Schinner-Predovic    | Shipped   |
+--------+----------+------------+----------------------+-----------+
10 rows in set (0.00 sec)
```

MySQL 8.0 Command Line Client

```
mysql> select * from products p left join order_items o on p.product_id = o.product_id union select * from products p right join order_items o on p.product_id = o.product_id;
+------------+------------------------------+-------------------+------------+----------+------------+----------+------------+
| product_id | name                         | quantity_in_stock | unit_price | order_id | product_id | quantity | unit_price |
+------------+------------------------------+-------------------+------------+----------+------------+----------+------------+
|          1 | Foam Dinner Plate            |                70 |       1.21 |        2 |          1 |        2 |       9.10 |
|          1 | Foam Dinner Plate            |                70 |       1.21 |        6 |          1 |        4 |       8.65 |
|          1 | Foam Dinner Plate            |                70 |       1.21 |       10 |          1 |       10 |       6.01 |
|          2 | Pork - Bacon,back Peameal    |                49 |       4.65 |        5 |          2 |        3 |       9.89 |
|          2 | Pork - Bacon,back Peameal    |                49 |       4.65 |        6 |          2 |        4 |       3.28 |
|          3 | Lettuce - Romaine, Heart     |                38 |       3.35 |        3 |          3 |       10 |       9.12 |
|          3 | Lettuce - Romaine, Heart     |                38 |       3.35 |        4 |          3 |        7 |       6.99 |
|          3 | Lettuce - Romaine, Heart     |                38 |       3.35 |        6 |          3 |        4 |       7.46 |
|          3 | Lettuce - Romaine, Heart     |                38 |       3.35 |        7 |          3 |        7 |       9.17 |
|          4 | Brocolinni - Gaylan, Chinese |                90 |       4.53 |        1 |          4 |        4 |       3.74 |
|          4 | Brocolinni - Gaylan, Chinese |                90 |       4.53 |        2 |          4 |        4 |       1.66 |
|          5 | Sauce - Ranch Dressing       |                94 |       1.63 |        6 |          5 |        1 |       3.45 |
|          5 | Sauce - Ranch Dressing       |                94 |       1.63 |        8 |          5 |        2 |       6.94 |
|          6 | Petit Baguette               |                14 |       2.39 |        2 |          6 |        2 |       2.94 |
|          6 | Petit Baguette               |                14 |       2.39 |        9 |          6 |        5 |       7.28 |
|          7 | Sweet Pea Sprouts            |                98 |       3.29 |     NULL |       NULL |     NULL |       NULL |
|          8 | Island Oasis - Raspberry     |                26 |       0.74 |        8 |          8 |        2 |       8.59 |
|          9 | Longan                       |                67 |       2.26 |       10 |          9 |        9 |       4.28 |
|         10 | Broom - Push                 |                 6 |       1.09 |        4 |         10 |        7 |       6.40 |
+------------+------------------------------+-------------------+------------+----------+------------+----------+------------+
19 rows in set (0.00 sec)
```

Experiment 7

Aim: Perform TCL and DCL commands

Resources required: P-IV and above, Oracle

Theory:

Commit, Rollback and Savepoint SQL commands

Transaction Control Language(TCL) commands are used to manage transactions in the database. These are used to manage the changes made to the data in a table by DML statements. It also allows statements to be grouped together into logical transactions.

COMMIT command

COMMIT command is used to permanently save any transaction into the database.

When we use any DML command like INSERT, UPDATE or DELETE, the changes made by these commands are not permanent, until the current session is closed, the changes made by these commands can be rolled back.

To avoid that, we use the COMMIT command to mark the changes as permanent.

Following is commit command's syntax,

Commit;

ROLLBACK command

This command restores the database to last committed state. It is also used with SAVEPOINT command to jump to a savepoint in an ongoing transaction.

If we have used the UPDATE command to make some changes into the database, and realise that those changes were not required, then we can use the ROLLBACK command to rollback those changes, if they were not commited using the COMMIT command.

Following is rollback command's syntax,

Rollback;

SAVEPOINT command

SAVEPOINT command is used to temporarily save a transaction so that you can rollback to that point whenever required.

Following is savepoint command's syntax,

savepoint

savepoint a2

DCL (Data Control Language)

As always, begin by connecting to your server where Oracle is hosted, then connect to Oracle itself as the SYSTEM account.

The SYSTEM account is one of a handful of predefined administrative accounts generated automatically when Oracle is installed. SYSTEM is capable of most administrative tasks, but the task we're particularly interested in is account management.

Creating a User

Once connected as SYSTEM, simply issue the CREATE USER command to generate a new account.

create user <username> identified by <password>;

Here we're simply creating a rina account that is IDENTIFIED or authenticated by the specified password.

a. The Grant Statement

With our new account created, we can now begin adding privileges to the account using the GRANT statement. GRANT is a very powerful statement with many possible options, but the core functionality is to manage the privileges of both users and roles throughout the database.

b. Providing Roles

Typically, you'll first want to assign privileges to the user through attaching the account to various roles, starting with the CONNECT role:

GRANT CONNECT TO <username>;

In some cases to create a more powerful user, you may also consider adding the RESOURCE role (allowing the user to create named types for custom schemas) or even the DBA role, which allows the user to not only create custom named types but alter and destroy them as well.

GRANT CONNECT, RESOURCE, DBA TO <username>;

c. Assigning Privileges

Next you'll want to ensure the user has privileges to actually connect to the database and create a session using `GRANT CREATE SESSION`. We'll also combine that with all privileges using `GRANT ANY PRIVILEGES.`

GRANT CREATE SESSION TO **<username>**;

We also need to ensure our new user has disk space allocated in the system to actually create or modify tables and data, so we'll GRANT TABLESPACE like so:

GRANT UNLIMITED TABLESPACE TO **<username>**;

GRANT RESOURCE TO RINA;

GRANT SELECT, INSERT, UPDATE, DELETE ON demo TO **<username>**;

GRANT ALL ON demo TO <username>;

connect rina/abc;

select * from system.demo;

delete from system.demo where rno=1;

connect system/oracle;

revoke delete on demo from **<username>**;

Conclusion: We performed TCL and DCL command successfully in MySQL Command Line.

Code and Output:

DCL command:

```
MySQL 8.0 Command Line Client - Unicode
mysql> create user 'adnan'@'localhost' identified by 'adnan';
Query OK, 0 rows affected (0.02 sec)

mysql> select user from mysql.user;
+------------------+
| user             |
+------------------+
| admin            |
| adnan            |
| mysql.infoschema |
| mysql.session    |
| mysql.sys        |
| root             |
+------------------+
6 rows in set (0.00 sec)

mysql> grant select,insert,delete on *.* to 'adnan'@'localhost';
Query OK, 0 rows affected (0.01 sec)

mysql> system mysql -u adnan -p
Enter password: *****
```

```
MySQL 8.0 Command Line Client - Unicode
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| invoicing          |
| mysql              |
| performance_schema |
| practical3         |
| sakila             |
| sql_hr             |
| sql_inventory      |
| sql_invoicing      |
| sql_store          |
| store              |
| sys                |
| user               |
| world              |
+--------------------+
14 rows in set (0.00 sec)
```

```
MySQL 8.0 Command Line Client - Unicode
mysql> desc customers;
+-------------+------------+------+-----+---------+----------------+
| Field       | Type       | Null | Key | Default | Extra          |
+-------------+------------+------+-----+---------+----------------+
| customer_id | int        | NO   | PRI | NULL    | auto_increment |
| first_name  | varchar(50)| NO   |     | NULL    |                |
| last_name   | varchar(50)| NO   |     | NULL    |                |
| birth_date  | date       | YES  |     | NULL    |                |
| phone       | varchar(50)| YES  |     | NULL    |                |
| address     | varchar(50)| NO   |     | NULL    |                |
| city        | varchar(50)| NO   |     | NULL    |                |
| state       | char(2)    | NO   |     | NULL    |                |
| points      | int        | NO   |     | 0       |                |
+-------------+------------+------+-----+---------+----------------+
9 rows in set (0.01 sec)

mysql> insert into customers(first_name,last_name,address,city,state,points) values('Rick','Roll','Historia Palace','Mumbai','MH',0);
Query OK, 1 row affected (0.01 sec)

mysql> select * from customers;
+-------------+------------+------------+------------+--------------+-----------------------+------------------+-------+--------+
| customer_id | first_name | last_name  | birth_date | phone        | address               | city             | state | points |
+-------------+------------+------------+------------+--------------+-----------------------+------------------+-------+--------+
|           1 | Babara     | MacCaffrey | 1986-03-28 | 781-932-9754 | 0 Sage Terrace        | Waltham          | MA    |   2273 |
|           2 | Ines       | Brushfield | 1986-04-13 | 804-427-9456 | 14187 Commercial Trail| Hampton          | VA    |    947 |
|           3 | Freddi     | Boagey     | 1985-02-07 | 719-724-7869 | 251 Springs Junction  | Colorado Springs | CO    |   2967 |
|           4 | Ambur      | Roseburgh  | 1974-04-14 | 407-231-8017 | 30 Arapahoe Terrace   | Orlando          | FL    |    457 |
|           5 | Clemmie    | Betchley   | 1973-11-07 | NULL         | 5 Spohn Circle        | Arlington        | TX    |   3675 |
|           6 | Elka       | Twiddell   | 1991-09-04 | 312-480-8498 | 7 Manley Drive        | Chicago          | IL    |   3073 |
|           7 | Ilene      | Dowson     | 1964-08-30 | 615-641-4759 | 50 Lillian Crossing   | Nashville        | TN    |   1672 |
|           8 | Thacher    | Naseby     | 1993-07-17 | 941-527-3977 | 538 Mosinee Center    | Sarasota         | FL    |    205 |
|           9 | Romola     | Rumgay     | 1992-05-23 | 559-181-3744 | 3520 Ohio Trail       | Visalia          | CA    |   1486 |
|          10 | Levy       | Mynett     | 1969-10-13 | 404-246-3370 | 68 Lawn Avenue        | Atlanta          | GA    |    796 |
|          11 | Rick       | Roll       | NULL       | NULL         | Historia Palace       | Mumbai           | MH    |      0 |
+-------------+------------+------------+------------+--------------+-----------------------+------------------+-------+--------+
11 rows in set (0.00 sec)

mysql> delete from customers where customer_id = 11;
Query OK, 1 row affected (0.01 sec)
```

```
MySQL 8.0 Command Line Client - Unicode

mysql> revoke select,insert,delete on *.* from 'adnan'@'localhost';
Query OK, 0 rows affected (0.01 sec)
```

```
MySQL 8.0 Command Line Client - Unicode

mysql> use store;
ERROR 1044 (42000): Access denied for user 'adnan'@'localhost' to database 'store'
mysql>
```

TCL command:

```
MySQL 8.0 Command Line Client - Unicode
mysql> set autocommit=0;
Query OK, 0 rows affected (0.00 sec)

mysql> commit;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from customers;
+-------------+------------+-----------+------------+--------------+----------------------+-----------------+-------+--------+
| customer_id | first_name | last_name | birth_date | phone        | address              | city            | state | points |
+-------------+------------+-----------+------------+--------------+----------------------+-----------------+-------+--------+
|           1 | Babara     | MacCaffrey | 1986-03-28 | 781-932-9754 | 0 Sage Terrace       | Waltham         | MA    |   2273 |
|           2 | Ines       | Brushfield | 1986-04-13 | 804-427-9456 | 14187 Commercial Trail | Hampton       | VA    |    947 |
|           3 | Levi       | Ackermann | 1996-09-12 | NULL         | Merlin               | merliyann       | TT    |    400 |
|           4 | Ambur      | Roseburgh | 1974-04-14 | 407-231-8017 | 30 Arapahoe Terrace  | Orlando         | FL    |    457 |
|           5 | Clemmie    | Betchley  | 1973-11-07 | NULL         | 5 Spohn Circle       | Arlington       | TX    |   3675 |
|           6 | Elka       | Twiddell  | 1991-09-04 | 312-480-8498 | 7 Manley Drive       | Konosuba        | IL    |   3073 |
|           7 | Ilene      | Dowson    | 1964-08-30 | 615-641-4759 | 50 Lillian Crossing  | Nashville       | TN    |   1672 |
|           8 | Thacher    | Naseby    | 1993-07-17 | 941-527-3977 | 538 Mosinee Center   | Sarasota        | FL    |    205 |
|           9 | Romola     | Rumgay    | 1992-05-23 | 559-181-3744 | 3520 Ohio Trail      | Visalia         | CA    |   1486 |
|          10 | Levy       | Mynett    | 1969-10-13 | 404-246-3370 | 68 Lawn Avenue       | Atlanta         | GA    |    796 |
|          11 | Eren       | Jaeger    | 1995-05-30 | NULL         | Wall Maria           | Paradise Island | TT    |    650 |
+-------------+------------+-----------+------------+--------------+----------------------+-----------------+-------+--------+
11 rows in set (0.00 sec)

mysql> savepoint A;
Query OK, 0 rows affected (0.00 sec)

mysql> delete from customers where customer_id =3;
Query OK, 1 row affected (0.01 sec)

mysql> update customers set points = 400 where customer_id = 11
    -> ;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> savepoint B;
Query OK, 0 rows affected (0.00 sec)

mysql> insert into customers (customer_id,first_name,last_name,birth_date,address,city,state,points) values(3,'Adnan','Shaikh','2001-09-12','Merlin','merliyan
n','TT',400);
Query OK, 1 row affected (0.00 sec)
```

```
MySQL 8.0 Command Line Client - Unicode
mysql> delete from customers where customer_id = 9;
Query OK, 1 row affected (0.00 sec)

mysql> savepoint C;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from customers;
+-------------+------------+-----------+------------+--------------+----------------------+-----------------+-------+--------+
| customer_id | first_name | last_name | birth_date | phone        | address              | city            | state | points |
+-------------+------------+-----------+------------+--------------+----------------------+-----------------+-------+--------+
|           1 | Babara     | MacCaffrey | 1986-03-28 | 781-932-9754 | 0 Sage Terrace       | Waltham         | MA    |   2273 |
|           2 | Ines       | Brushfield | 1986-04-13 | 804-427-9456 | 14187 Commercial Trail | Hampton       | VA    |    947 |
|           3 | Adnan      | Shaikh    | 2001-09-12 | NULL         | Merlin               | merliyann       | TT    |    400 |
|           4 | Ambur      | Roseburgh | 1974-04-14 | 407-231-8017 | 30 Arapahoe Terrace  | Orlando         | FL    |    457 |
|           5 | Clemmie    | Betchley  | 1973-11-07 | NULL         | 5 Spohn Circle       | Arlington       | TX    |   3675 |
|           6 | Elka       | Twiddell  | 1991-09-04 | 312-480-8498 | 7 Manley Drive       | Konosuba        | IL    |   3073 |
|           7 | Ilene      | Dowson    | 1964-08-30 | 615-641-4759 | 50 Lillian Crossing  | Nashville       | TN    |   1672 |
|           8 | Thacher    | Naseby    | 1993-07-17 | 941-527-3977 | 538 Mosinee Center   | Sarasota        | FL    |    205 |
|          10 | Levy       | Mynett    | 1969-10-13 | 404-246-3370 | 68 Lawn Avenue       | Atlanta         | GA    |    796 |
|          11 | Eren       | Jaeger    | 1995-05-30 | NULL         | Wall Maria           | Paradise Island | TT    |    400 |
+-------------+------------+-----------+------------+--------------+----------------------+-----------------+-------+--------+
10 rows in set (0.00 sec)

mysql> rollback to B;
Query OK, 0 rows affected (0.01 sec)

mysql> select * from customers;
+-------------+------------+-----------+------------+--------------+----------------------+-----------------+-------+--------+
| customer_id | first_name | last_name | birth_date | phone        | address              | city            | state | points |
+-------------+------------+-----------+------------+--------------+----------------------+-----------------+-------+--------+
|           1 | Babara     | MacCaffrey | 1986-03-28 | 781-932-9754 | 0 Sage Terrace       | Waltham         | MA    |   2273 |
|           2 | Ines       | Brushfield | 1986-04-13 | 804-427-9456 | 14187 Commercial Trail | Hampton       | VA    |    947 |
|           4 | Ambur      | Roseburgh | 1974-04-14 | 407-231-8017 | 30 Arapahoe Terrace  | Orlando         | FL    |    457 |
|           5 | Clemmie    | Betchley  | 1973-11-07 | NULL         | 5 Spohn Circle       | Arlington       | TX    |   3675 |
|           6 | Elka       | Twiddell  | 1991-09-04 | 312-480-8498 | 7 Manley Drive       | Konosuba        | IL    |   3073 |
|           7 | Ilene      | Dowson    | 1964-08-30 | 615-641-4759 | 50 Lillian Crossing  | Nashville       | TN    |   1672 |
|           8 | Thacher    | Naseby    | 1993-07-17 | 941-527-3977 | 538 Mosinee Center   | Sarasota        | FL    |    205 |
|           9 | Romola     | Rumgay    | 1992-05-23 | 559-181-3744 | 3520 Ohio Trail      | Visalia         | CA    |   1486 |
|          10 | Levy       | Mynett    | 1969-10-13 | 404-246-3370 | 68 Lawn Avenue       | Atlanta         | GA    |    796 |
|          11 | Eren       | Jaeger    | 1995-05-30 | NULL         | Wall Maria           | Paradise Island | TT    |    400 |
+-------------+------------+-----------+------------+--------------+----------------------+-----------------+-------+--------+
10 rows in set (0.00 sec)
```

```
MySQL 8.0 Command Line Client - Unicode

mysql> rollback to A;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from customers;
+-------------+------------+-----------+------------+--------------+----------------------+-----------------+-------+--------+
| customer_id | first_name | last_name | birth_date | phone        | address              | city            | state | points |
+-------------+------------+-----------+------------+--------------+----------------------+-----------------+-------+--------+
|           1 | Babara     | MacCaffrey | 1986-03-28 | 781-932-9754 | 0 Sage Terrace       | Waltham         | MA    |   2273 |
|           2 | Ines       | Brushfield | 1986-04-13 | 804-427-9456 | 14187 Commercial Trail | Hampton       | VA    |    947 |
|           3 | Levi       | Ackermann | 1996-09-12 | NULL         | Merlin               | merliyann       | TT    |    400 |
|           4 | Ambur      | Roseburgh | 1974-04-14 | 407-231-8017 | 30 Arapahoe Terrace  | Orlando         | FL    |    457 |
|           5 | Clemmie    | Betchley  | 1973-11-07 | NULL         | 5 Spohn Circle       | Arlington       | TX    |   3675 |
|           6 | Elka       | Twiddell  | 1991-09-04 | 312-480-8498 | 7 Manley Drive       | Konosuba        | IL    |   3073 |
|           7 | Ilene      | Dowson    | 1964-08-30 | 615-641-4759 | 50 Lillian Crossing  | Nashville       | TN    |   1672 |
|           8 | Thacher    | Naseby    | 1993-07-17 | 941-527-3977 | 538 Mosinee Center   | Sarasota        | FL    |    205 |
|           9 | Romola     | Rumgay    | 1992-05-23 | 559-181-3744 | 3520 Ohio Trail      | Visalia         | CA    |   1486 |
|          10 | Levy       | Mynett    | 1969-10-13 | 404-246-3370 | 68 Lawn Avenue       | Atlanta         | GA    |    796 |
|          11 | Eren       | Jaeger    | 1995-05-30 | NULL         | Wall Maria           | Paradise Island | TT    |    650 |
+-------------+------------+-----------+------------+--------------+----------------------+-----------------+-------+--------+
11 rows in set (0.00 sec)
```

Experiment 8

Aim: Write a PL/SQL block to create Triggers for update, delete and insert

Resources required: P-IV and above, Oracle

Theory:

View

A **view** is a virtual table, which provides access to a subset of column from one or more table. A **view** can derive its data from one or more table. An output of query can be stored as a **view**. ... A **view in oracle** is nothing but a stored sql scripts.

create view v11 as (select empid, ename ,dname from emp,dept where emp.dno=dept.dno);

Views may be created for the following reasons:

1. The DBA stores the views as a definition only. Hence there is no duplication of data.
2. Simplifies Queries.
3. Can be Queried as a base table itself.
4. Provides data security.
5. Avoids data redundancy.

Creation of Views:-

Syntax:-

CREATE VIEW viewname AS
SELECT columnname,columnname
FROM tablename
WHERE columnname=expression_list;

Renaming the columns of a view:-

Syntax:-

CREATE VIEW viewname AS
SELECT newcolumnname….
FROM tablename
WHERE columnname=expression_list;

Selecting a data set from a view-

Syntax:-

SELECT columnname, columnname

FROM viewname

WHERE search condition;

Destroying a view-

Syntax:-

DROP VIEW viewname;

Triggers

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

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

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

Benefits of Triggers

Triggers can be written for the following purposes −

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

The syntax for creating a trigger is −

CREATE [OR REPLACE] TRIGGER trigger_name

{BEFORE | AFTER | INSTEAD OF }

{INSERT [OR] | UPDATE [OR] | DELETE}

[OF col_name]

```
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW]
WHEN (condition)
DECLARE
  Declaration-statements
BEGIN
  Executable-statements
EXCEPTION
  Exception-handling-statements
END;
```

Where,

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

SQL>create table customer(id number(2), name varchar2(15), age number(2), address varchar2(20), salary number(9,2));

```
SQL>create table customer_log(id number(2), name varchar2(15), age number(2), address
varchar2(20), salary number(9,2), usern varchar2(10), dt date, oper varchar2(10));
create or replace trigger cust_log
  after update or delete or insert on customer1
     for each row
        declare
        op varchar2(10);
        begin
        if updating then
        op:='update';
       insert into customer1_log(id,name,age,address,salary,dt,oper) values(:old.id, :old.name,
:old.age, :old.address, :old.salary,current_timestamp, op);
      end if;
      if deleting then
      op:='delete';
      insert into customer1_log(id,name,age,address,salary,dt,oper) values(:old.id, :old.name,
:old.age, :old.address, :old.salary,current_timestamp, op);
      end if;
      if inserting then
      op:='insert';
      insert into customer1_log(id,name,age,address,salary,dt,oper) values(:new.id, :new.name,
:new.age, :new.address, :new.salary,current_timestamp, op);
      end if;
   end;

SQL> insert into customer1(id,salary) values(9,10000);

1 row created.
```

Conclusion: We have successfully Implemented Views and Triggers in MySQL Command Line.

Code and Output:

Views:

```
MySQL 8.0 Command Line Client - Unicode

mysql> select * from invoices;
+------------+--------------+-----------+---------------+---------------+--------------+
| invoice_id | number       | client_id | invoice_total | payment_total | invoice_date |
+------------+--------------+-----------+---------------+---------------+--------------+
|          2 | 828-863-4354 |         3 |        500.97 |          0.00 | 2021-02-28   |
|          3 | 992-113-4556 |         1 |       2500.97 |          0.00 | 2021-03-01   |
|          4 | 992-113-4556 |         1 |       5500.97 |          0.00 | 2021-03-01   |
|          5 | 828-863-4354 |         3 |        600.45 |          0.00 | 2021-03-02   |
+------------+--------------+-----------+---------------+---------------+--------------+
4 rows in set (0.01 sec)

mysql>  create view invoice_v1 as (select invoice_id,number,invoice_total from invoices);
Query OK, 0 rows affected (0.03 sec)

mysql> select * from invoice_v1;
+------------+--------------+---------------+
| invoice_id | number       | invoice_total |
+------------+--------------+---------------+
|          2 | 828-863-4354 |        500.97 |
|          3 | 992-113-4556 |       2500.97 |
|          4 | 992-113-4556 |       5500.97 |
|          5 | 828-863-4354 |        600.45 |
+------------+--------------+---------------+
4 rows in set (0.01 sec)

mysql> update invoices_v1 set invoice_total = 50.97 WHERE invoice_id = 2;
ERROR 1146 (42S02): Table 'practical3.invoices_v1' doesn't exist
mysql> update invoice_v1 set invoice_total = 50.97 WHERE invoice_id = 2;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from invoices;
+------------+--------------+-----------+---------------+---------------+--------------+
| invoice_id | number       | client_id | invoice_total | payment_total | invoice_date |
+------------+--------------+-----------+---------------+---------------+--------------+
|          2 | 828-863-4354 |         3 |         50.97 |          0.00 | 2021-02-28   |
|          3 | 992-113-4556 |         1 |       2500.97 |          0.00 | 2021-03-01   |
|          4 | 992-113-4556 |         1 |       5500.97 |          0.00 | 2021-03-01   |
|          5 | 828-863-4354 |         3 |        600.45 |          0.00 | 2021-03-02   |
+------------+--------------+-----------+---------------+---------------+--------------+
4 rows in set (0.00 sec)
```

MySQL 8.0 Command Line Client - Unicode

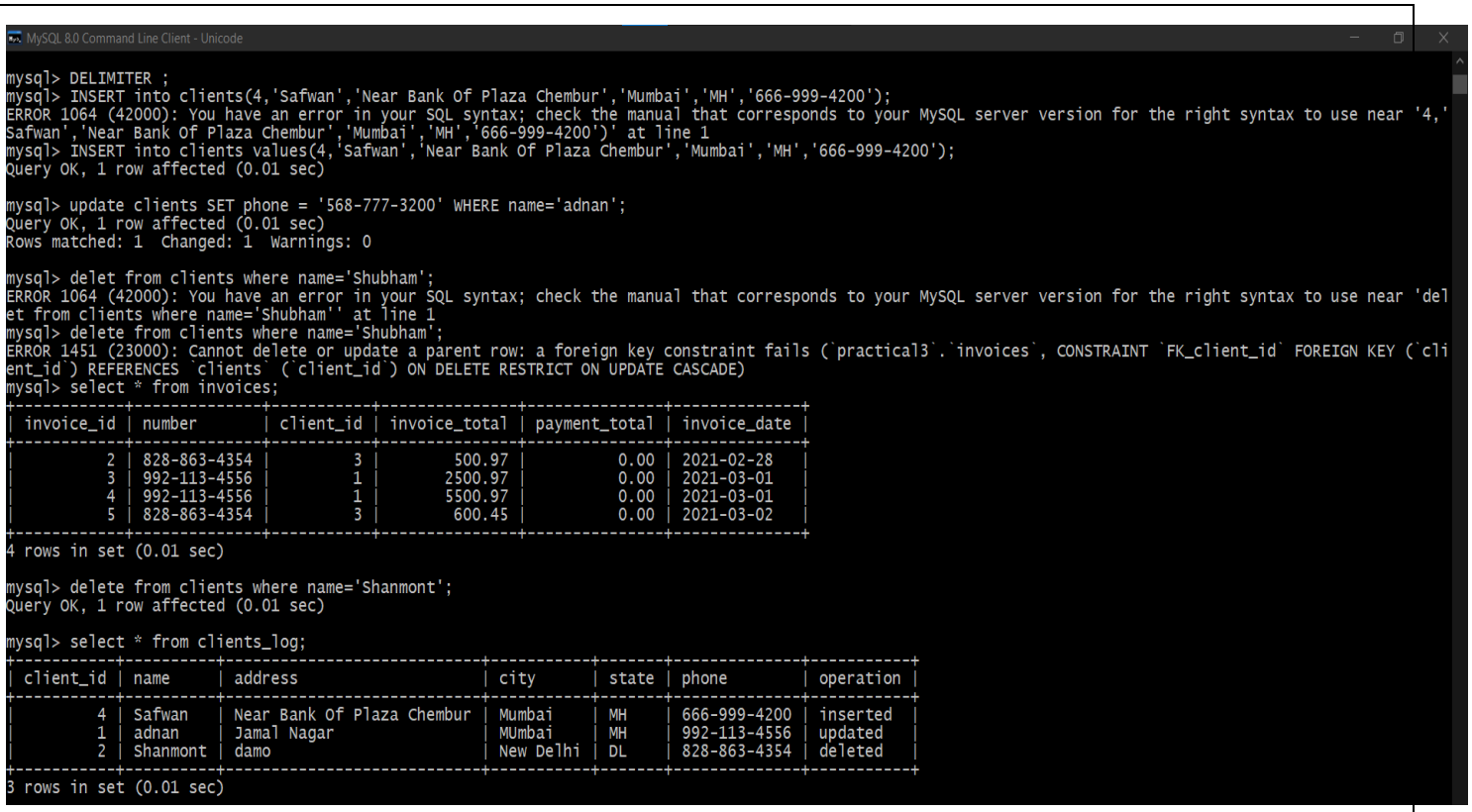
```
mysql> select * from invoice_v1;
+------------+--------------+----------------+
| invoice_id | number       | invoice_total  |
+------------+--------------+----------------+
|          2 | 828-863-4354 |          50.97 |
|          3 | 992-113-4556 |        2500.97 |
|          4 | 992-113-4556 |        5500.97 |
|          5 | 828-863-4354 |         600.45 |
+------------+--------------+----------------+
4 rows in set (0.00 sec)

mysql> DROP view invoice_v1;
Query OK, 0 rows affected (0.01 sec)

mysql> select * from invoice_v1;
ERROR 1146 (42S02): Table 'practical3.invoice_v1' doesn't exist
```

Triggers:

```
mysql> select * from clients;
+-----------+----------+-------------+-------------+-------+--------------+
| client_id | name     | address     | city        | state | phone        |
+-----------+----------+-------------+-------------+-------+--------------+
|         1 | adnan    | Jamal Nagar | MUmbai      | MH    | 992-113-4556 |
|         2 | Shanmont | damo        | New Delhi   | DL    | 828-863-4354 |
|         3 | Shubham  | Khargar     | Navi-Mumbai | MH    | 828-863-4354 |
+-----------+----------+-------------+-------------+-------+--------------+
3 rows in set (0.01 sec)

mysql> create table clients_log(client_id int(10),name varchar(50),address varchar(256),city varchar(20),state char(2),phone varchar(20),operation varchar(20)
);
Query OK, 0 rows affected, 1 warning (0.04 sec)

mysql> DELIMITER $$
```

MySQL 8.0 Command Line Client - Unicode

```
mysql> DELIMITER $$
mysql>  CREATE TRIGGER cl_log_insert
    -> AFTER INSERT ON clients
    -> FOR each row
    -> BEGIN
    -> INSERT INTO clients_log VALUES(NEW.client_id,NEW.name,NEW.address,NEW.city,NEW.state,NEW.phone,'inserted');
    -> END$$
Query OK, 0 rows affected (0.02 sec)

mysql>  CREATE TRIGGER cl_log_delete
    -> AFTER DELETE ON clients
    -> FOR each row
    -> BEGIN
    -> INSERT INTO clients_log VALUES(OLD.client_id,OLD.name,OLD.address,OLD.city,OLD.state,OLD.phone,'deleted');
    -> END $$
Query OK, 0 rows affected (0.02 sec)

mysql> CREATE TRIGGER cl_log_update
    -> AFTER UPDATE ON clients
    -> FOR each row
    -> BEGIN
    -> INSERT INTO clients_log VALUES(OLD.client_id,OLD.name,OLD.address,OLD.city,OLD.state,OLD.phone,'updated');
    -> END$$
Query OK, 0 rows affected (0.03 sec)

mysql> DELIMITER ;
```


MySQL 8.0 Command Line Client - Unicode
mysql> DELIMITER ;
mysql> INSERT into clients(4,'Safwan','Near Bank Of Plaza Chembur','Mumbai','MH','666-999-4200');
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '4,'
Safwan','Near Bank Of Plaza Chembur','Mumbai','MH','666-999-4200')' at line 1
mysql> INSERT into clients values(4,'Safwan','Near Bank Of Plaza Chembur','Mumbai','MH','666-999-4200');
Query OK, 1 row affected (0.01 sec)
mysql> update clients SET phone = '568-777-3200' WHERE name='adnan';
Query OK, 1 row affected (0.01 sec)
Rows matched: 1 Changed: 1 Warnings: 0
mysql> delet from clients where name='Shubham';
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'del
et from clients where name='Shubham'' at line 1
mysql> delete from clients where name='Shubham';
ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key constraint fails (`practical3`.`invoices`, CONSTRAINT `FK_client_id` FOREIGN KEY (`cli
ent_id`) REFERENCES `clients` (`client_id`) ON DELETE RESTRICT ON UPDATE CASCADE)
mysql> select * from invoices;
invoice_id	number	client_id	invoice_total	payment_total	invoice_date
2	828-863-4354	3	500.97	0.00	2021-02-28
3	992-113-4556	1	2500.97	0.00	2021-03-01
4	992-113-4556	1	5500.97	0.00	2021-03-01
5	828-863-4354	3	600.45	0.00	2021-03-02
4 rows in set (0.01 sec)
mysql> delete from clients where name='Shanmont';
Query OK, 1 row affected (0.01 sec)
mysql> select * from clients_log;
client_id	name	address	city	state	phone	operation
4	Safwan	Near Bank Of Plaza Chembur	Mumbai	MH	666-999-4200	inserted
1	adnan	Jamal Nagar	MUmbai	MH	992-113-4556	updated
2	Shanmont	damo	New Delhi	DL	828-863-4354	deleted
3 rows in set (0.01 sec)

SARASWATI

Department of Computer Engineering

College of Engineering

Experiment 9

Aim: Write PL/SQL code block to calculate the area of circle for a value of radius varying from 1 to 10. Store the radius and the corresponding values of calculated area in a table named areas (radius, area)

Resources required: P-IV and above, Oracle

Theory:

```
BEGIN
    DECLARE pi float(5,4) DEFAULT 3.14;
    DECLARE r int(5);
    DECLARE a float(7,2);
    SET r = 1;
    WHILE r<=10 DO
            SET a = pi*r*r;
        INSERT INTO area values(r,a);
        SET r = r+1;
        END WHILE;
END
```

Conclusion: We have Successfully calculated area of circle for radius varying from 1 to 10 using code block in MySQL Command Line and Work Bench

Code and Output:

```
MySQL 8.0 Command Line Client - Unicode
mysql> use practical3;
Database changed
mysql> SHOW CREATE PROCEDURE area_of_circle;
+----------------+---------------------------------------------+---------------------------------------------
------------------------------------------------------------------------------------------------------------
-----------------------------------------------+-----------------------+-----------------------+--------------------+
| Procedure      | sql_mode                                    | Create Procedure

                                               | character_set_client | collation_connection | Database Collation |
+----------------+---------------------------------------------+---------------------------------------------
------------------------------------------------------------------------------------------------------------
-----------------------------------------------+-----------------------+-----------------------+--------------------+
| area_of_circle | STRICT_TRANS_TABLES,NO_ENGINE_SUBSTITUTION | CREATE DEFINER=`root`@`localhost` PROCEDURE `area_of_circle`()
BEGIN
    DECLARE pi float(5,4) DEFAULT 3.14;
    DECLARE r int(5);
    DECLARE a float(7,2);
    SET r = 1;
    WHILE r<=10 DO
            SET a = pi*r*r;
        INSERT INTO area values(r,a);
        SET r = r+1;

        END WHILE;


END | utf8mb4              | utf8mb4_0900_ai_ci   | utf8mb4_0900_ai_ci |
+----------------+---------------------------------------------+---------------------------------------------
------------------------------------------------------------------------------------------------------------
-----------------------------------------------+-----------------------+-----------------------+--------------------+
1 row in set (0.00 sec)

mysql> CALL area_of_circle();
Query OK, 1 row affected (0.05 sec)

mysql> SELECT * from area;
+------+--------+
| r    | a      |
+------+--------+
|    1 |   3.14 |
|    2 |  12.56 |
|    3 |  28.26 |
|    4 |  50.24 |
|    5 |  78.50 |
|    6 | 113.04 |
|    7 | 153.86 |
|    8 | 200.96 |
|    9 | 254.34 |
|   10 | 314.00 |
+------+--------+
```

Experiment 10

Aim: Demonstrate Database connectivity

Resources required: P-IV and above, Python compiler

Theory:

Step1: Install python and set the path.

Step2: Using any IDE or in terminal write the following command to install mysql connector:

pip install mysql-connector-python

or

Python -m pip install mysql-connector-python (if upper command doesn't work)

After installing above library we're ready to use it in our python files
(Note: If you create Virtual Environment you need to install it in your Virtual Environment.)

Step3: Create a Python file of your desired name(extension: .py)

Step4: You need to write following code at the top to import mysql connector:

from mysql import connector

Step5: Now we need to connect to database using mysql.connector object, for that we have following syntax:

Object_name = connector.connect(host="hostname", username="db_username", password = "user_password", database = "db_name")

Hostname: In our case hostname is local host since we're not hosting it anywhere

db_username: It is a database username it should exist or else code will raise an error.
Password: user password should be correct or code will raise an error.
Db_name: database name which you're going to use if not exist it will raise an error.

Step6: If everything is correct in Step5 we have successfully connected to our database and we can write queries using our Object.

Conclusion: We have successfully demonstrate MySQL database connectivity in python.

Code:

```
from mysql import connector


mydb = connector.connect(
            host = "localhost",
            username = "root",
            password = "admin",
            database = "pythontemp"
        )

mycursor = mydb.cursor()

mycursor.execute("CREATE TABLE IF NOT EXISTS customers(customer_id INT AUTO_INCREMENT P
RIMARY KEY,name VARCHAR(30))")

mycursor.execute("SHOW TABLES")

print(*[x for x in mycursor])

try:
    mycursor.execute("ALTER TABLE customers ADD address VARCHAR(255)")
except:
    print("Column already exist")

mycursor.execute("DESC customers")

print(*[x for x in mycursor])

sql = "INSERT INTO customers(name,address) values(%s,%s)"

val = [
    ("Levi Ackermann","Wall Rose"),
    ("Eren Jaeger","Wall Maria"),
    ("Lalatinna","Konosuba"),
    ("Kaneki Kun","Re"),
    ("Rias Gremory","DxD"),
]

mycursor.executemany(sql,val)

mydb.commit()

print(mycursor.rowcount," was inserted")

mycursor.execute("SELECT * FROM customers ORDER BY name")

result = mycursor.fetchall()

for x in result:
    print(x)
```

Output:

```
[Running] python -u "c:\Users\adnan\OneDrive\Desktop\College\Sem 4\DBMS\Practicals\Practical-10\dbconnect.py"
('customers',)
('customer_id', b'int', 'NO', 'PRI', None, 'auto_increment') ('name', b'varchar(30)', 'YES', '', None, '') ('address', b'varchar(255)', 'YES', '', None, '')
5  was inserted
(2, 'Eren Jaeger', 'Wall Maria')
(4, 'Kaneki Kun', 'Re')
(3, 'Lalatinna', 'Konosuba')
(1, 'Levi Ackermann', 'Wall Rose')
(5, 'Rias Gremory', 'DxD')

[Done] exited with code=0 in 0.261 seconds
```

Page No. 1
Date

Assignment - 1

Q.1)) Explain overall architecture of D.B.M.S in detail.

Ans) The architecture of a database System is greatly influenced by the underlying Computer System on which databases is running.

i) Centralized.
ii) Client-Server.
iii) Parallel (multi-Processor).
iv) Distributed.

Explaination of Fig a. Database System Architecture is given below:

(1) Data base users and User Interfaces:-

(a) There are 4 different types of database-System users. (I) Naive users (II) Application-Programmers (III) Sophisticated users (IV) Specialized users.

(b) Naive users:- These are unsophisticated users who interact with the system by invoking one of the application programs. They comes View-lvl abstraction.

c) Application programmers:- These are computer professionals who write application programs. They use DDL & DML to create backend and link it with frontend of application programs for ease of Naive users. They come under logical-lvl abstraction.

d) Sophisticated users:- They write database query which is fed to query processor, which function is to breakdown DML statements. Analysts which who submit queries to explore data fall under this category. They come under logical-lvl abstraction.

e) Specialized users:- These users write specialized database applications (frame works) for the ease of application programmers.

(ii) Database Administrator (DBA):-

a) The Database administrator have control over both the data and the programs that access those data.
b) The functions of DBA include:-
c) Schema definition
d) Storage Structure and access-method definition.
e) Schema and Physical-organization modification.
f) Granting of authorization for data access.
g) Routine maintenance.

h) They come under Physical & view level abstraction.

(iii) The Query Processor includes:-

a) DDL interpreter, which interpret DDL Statements and records the definitions in the data dictionary.
b) DML compiler, which translates DML Statements in a query language.
c) Query evaluation engine, which executes low-lvl instructions generated by the DML compiler

(iv) Storage Manager:-

a) A Storage Manager is a program module that provides the interface between the low level data Stored in the database and the application programs and queries submitted to the System.

The Storage manager components include:

b) Authorization and integrity manager, which tests for the Satisfaction of integrity constraints and checks the authority of users to access data.
c) Transaction manager, ensures Atomicity.
d) File manager, which manages allocation of space on disk Storage and the data Structures used to represent information Stored on disk
e) Buffer manager fetches data from disk Storage into main memory.

(v) Disk Storage maintains all files, dictionary and hashes at physical level.

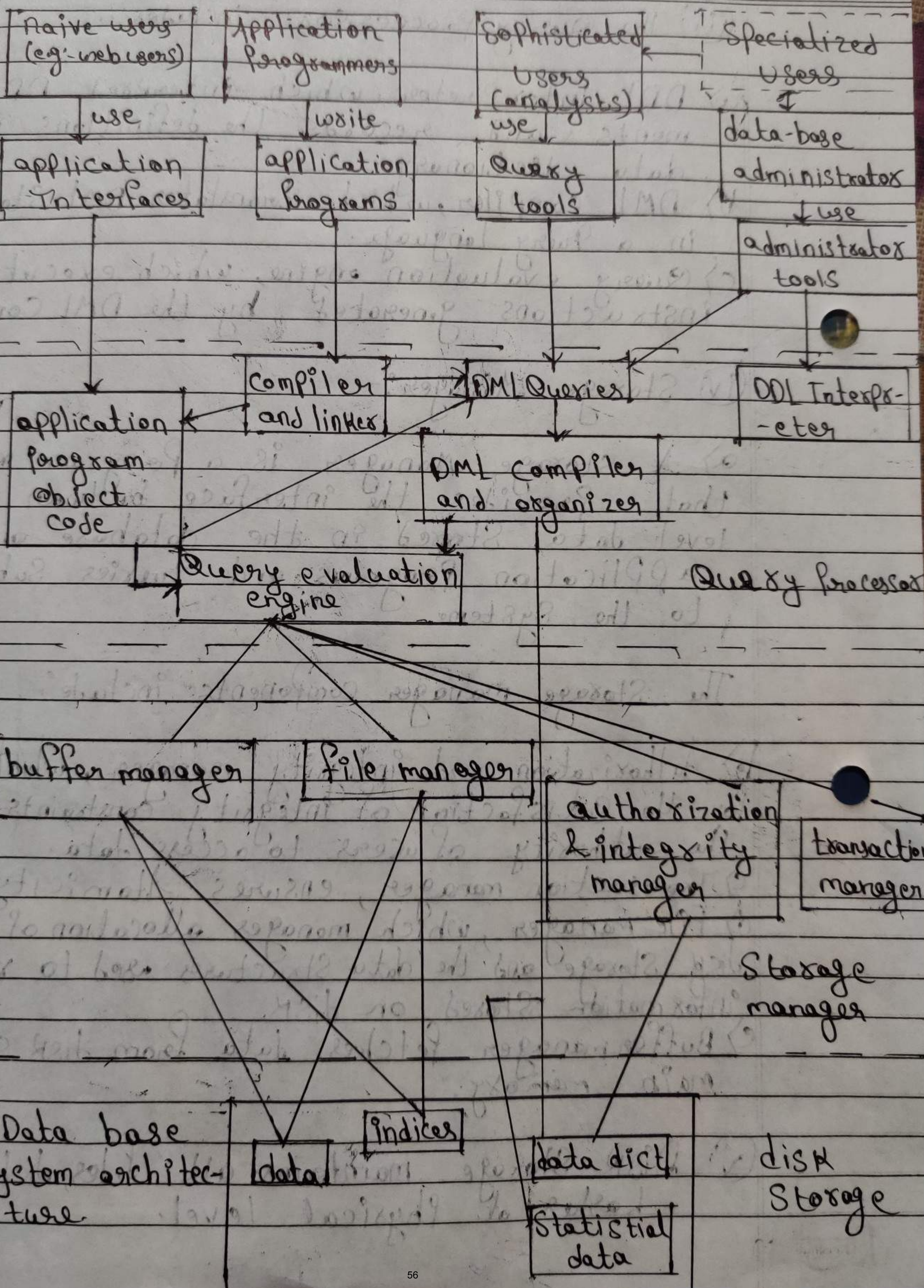
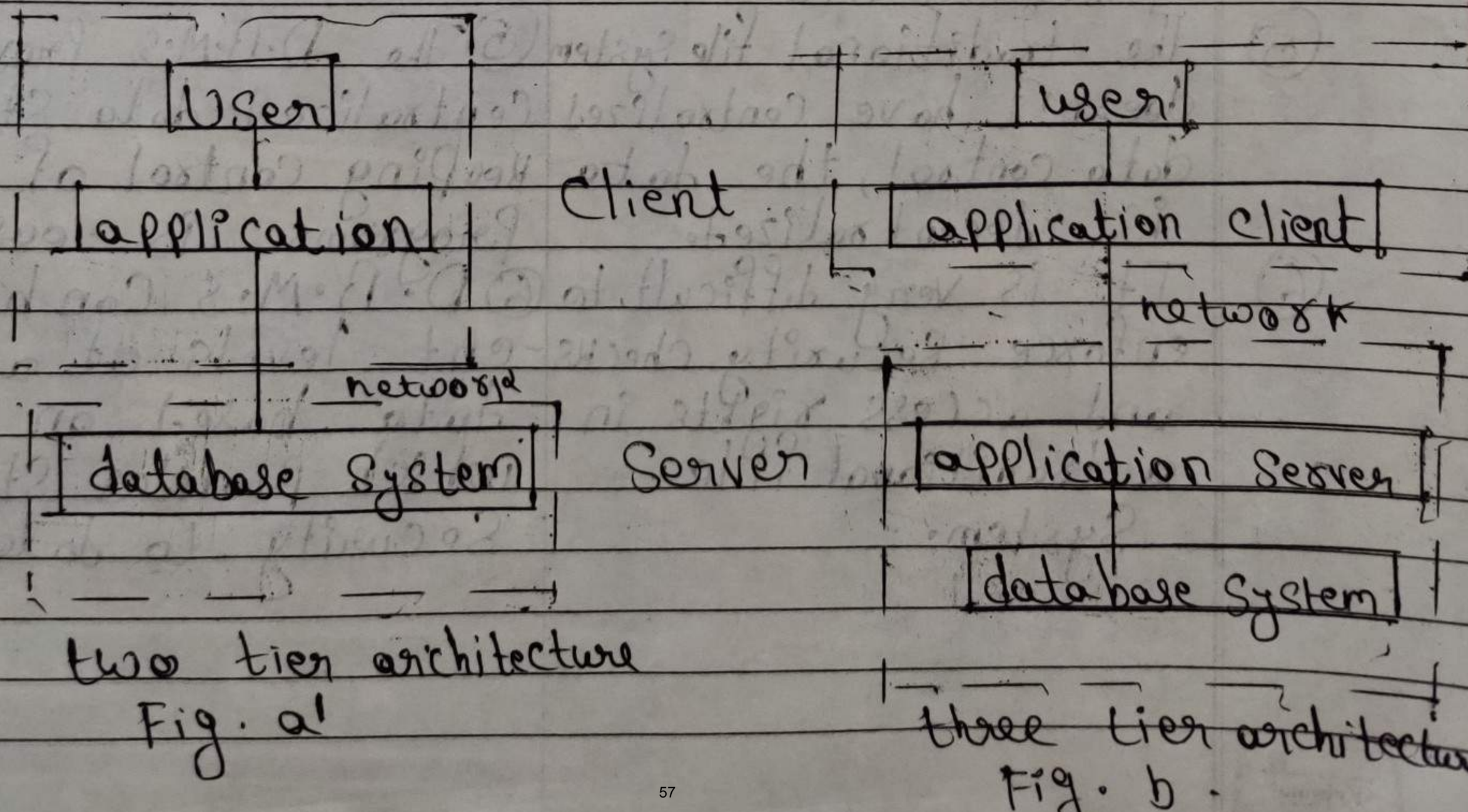


Fig a:- Data base System architecture

(v) Database applications are usually partitioned into two parts, as in Figure a, b

a) In a two tier architecture, the application is partitioned into component that resides at the client machine, which invokes database functionally at the serve machine through query language statements. Application program interface standards like ODBC & JDBC are used for interaction b/w the client and the server.

b) In a three tier architecture the client machine acts as merely a frontend and doesn't contain any direct database calls. Instead, the client end communicates with the application server, usually through form interface. The application server in turns communicates with a database system to access data.



two tier architecture
Fig. a

three tier architecture
Fig. b.

Q.2) Difference between File Processing & Database Management System.

Ans)

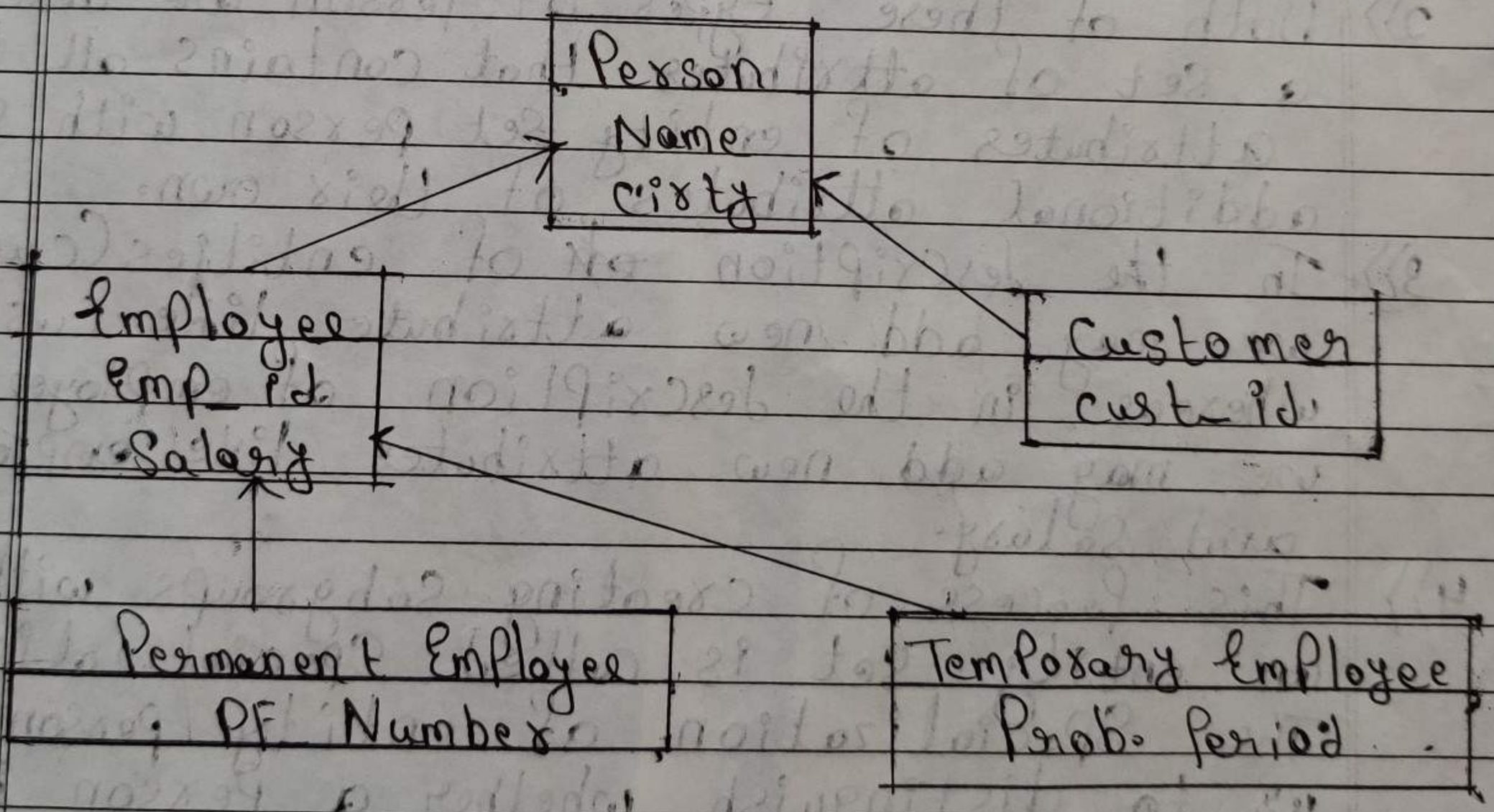
File Processing System	Database Management System
① Duplicate data may exist in multiple files which leads to data redundancy.	① The data is integrated into a single database which avoids data redundancy.
② Data inconsistency occurs when data is not updated in all files.	② The data consistency is maintained since, data is integrated in a database.
③ It is difficult to share data in traditional file system	③ In D.B.M.S data can be easily shared with multiple apps.
④ In files data is stored in specific format. If the format of any of the file is changed, then we have to make changes in the program ~~with~~ which processes file.	④ In D.B.M.S we can completely seperate the data structure of database & programs or applications which are used to access data.
⑤ The traditional file system doesn't have centralized data control, the data is decentralized.	⑤ The D.B.M.S provides the centralized data storage. Hence keeping control of data & programs is easy.
⑥ It is very difficult to enforce security checks and access rights in a traditional file system.	⑥ D.B.M.S can have different levels of access to data based on their roles which provides storage security to data.

Q.3) Explain Extended ER features.

Ans) (A) Generalization:-

① The refinement from an initial entity set into subgroups depending upon distinct features shows top-down approach.

② Generalization is termed as containment relationship that exists between higher level entity set and one or more lower level entity sets.



Generalization

③ Here, the employee is higher level entity set while the entities 'Permanent employee' and 'Temporary employee' are the lower level entity set.

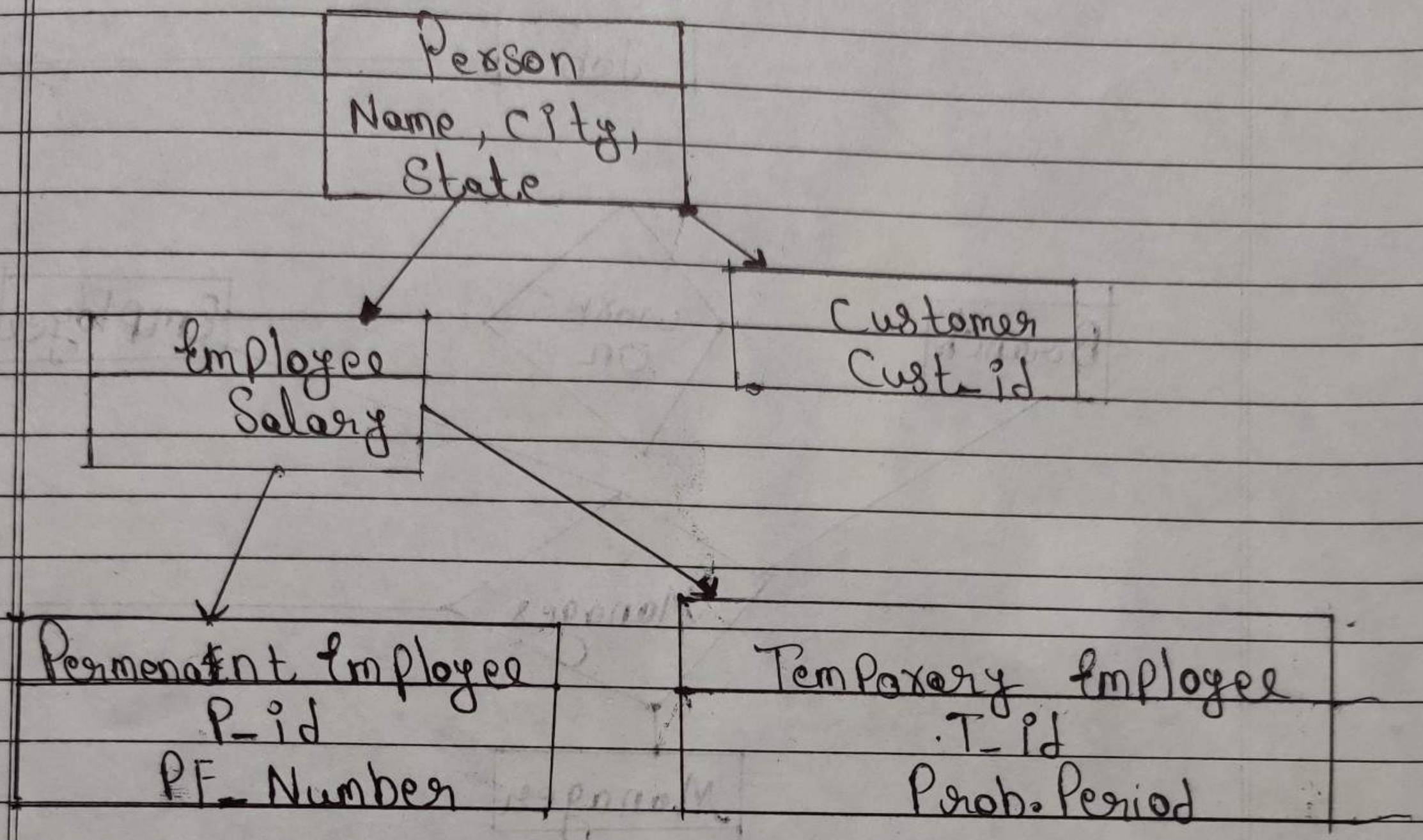
④ The higher level entity set is also known as super class while the lower level entity set also

4)) Known as Subclass.
5)) The attributes of the higher level entity sets are generally considered to be inherited from the lower level entity sets.

(B) Specialization:-

1)) Consider, an example of entity set person having attribute name, city and state. The entity person may be further classified as follows:-
- Customer
- employee

2)) Both of these types of person are described by a set of attributes that contains all the attributes of entity set person with some additional attributes of their own.
3)) In the description ~~of~~ of entities (customer), we may add new attribute like Cust_id whereas in the description of employee entities we may add new attributes like employee-id, and salary.
4)) This process of creating subgroups within an entity set is called Specialization. The Specialization of entity person helps us to distinguish whether a person is an employee or customer depending upon the attributes
5)) An entity set maybe specialized by more than one distinguishing character.



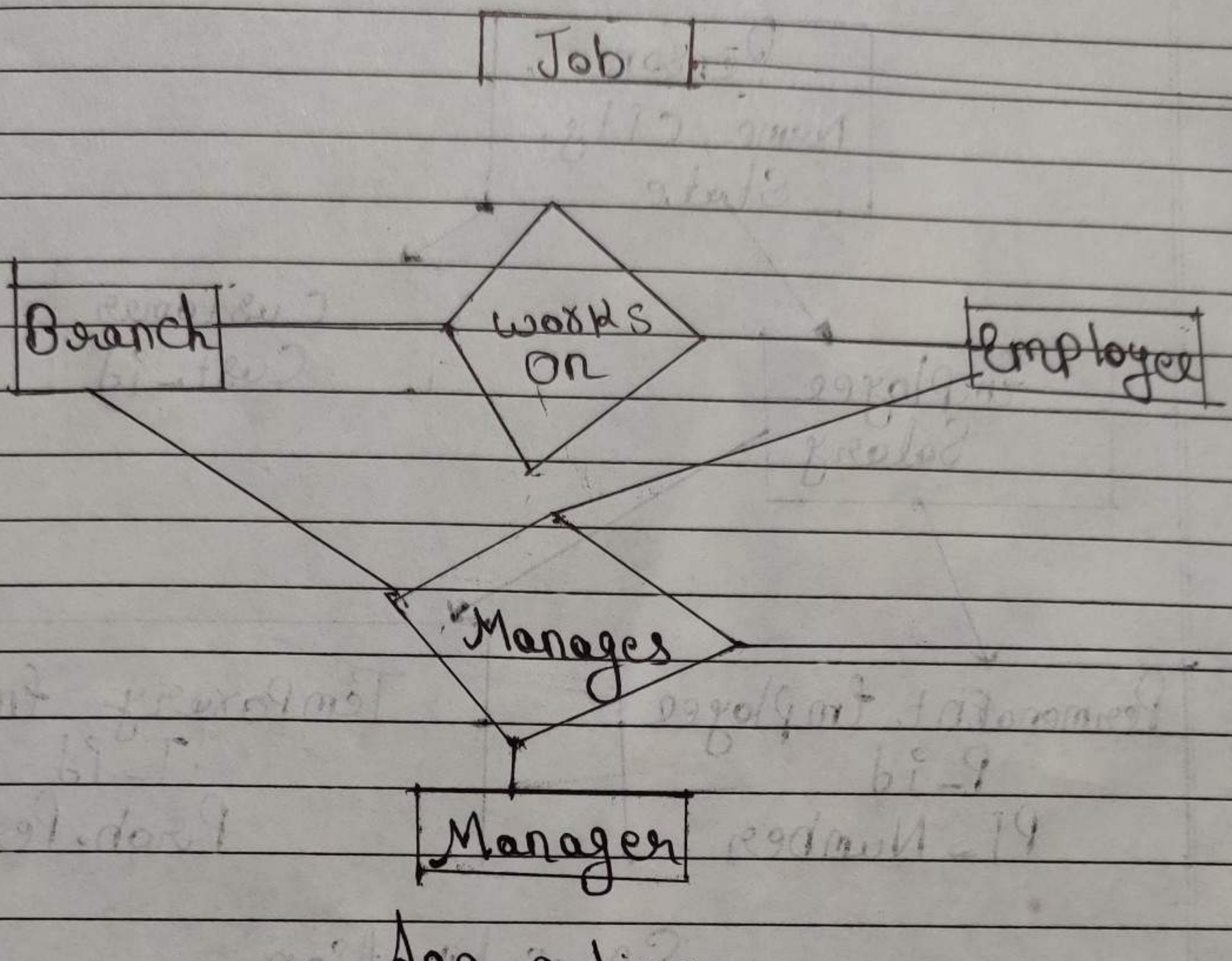
Specialization

(C) Aggregation:-

① In E-R model there is limitation i.e: it cannot express relationships among relationship. Consider the ternary relationships 'Works-on' between an employee, branch & job.

② Consider we want to record managers for the above Combination. An entity set manager is available

③ To represent this relationships, we can set a quaternary relationships manages between employee, branch, Job and manager. Using the basic E-R modelling constructs, we obtain the ~~above~~ below E-R diagram.

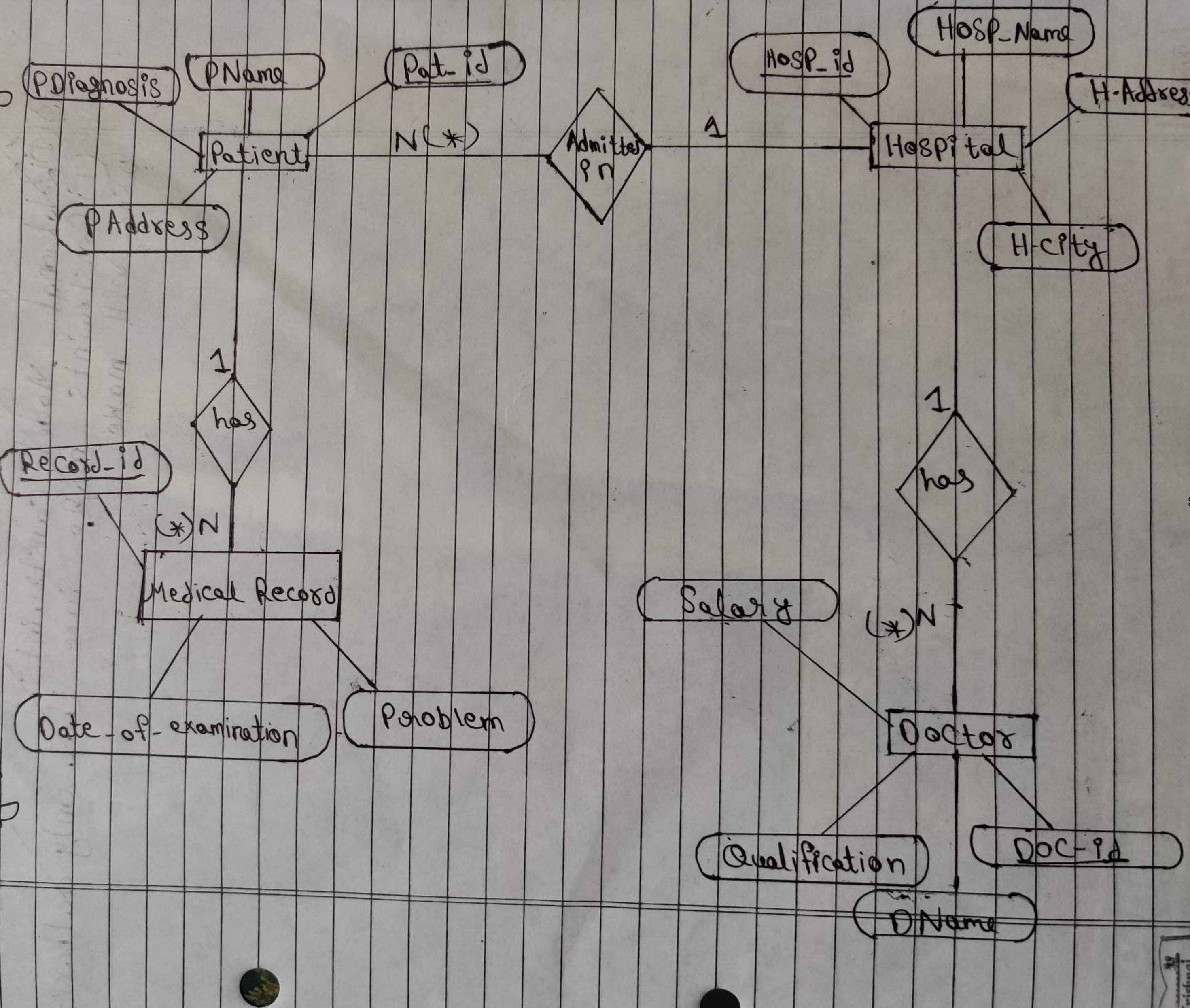


Aggregation

4)) Here, the relationship set 'works-on' and 'manages' can be combined into one single relation ship set. But Combining them is difficult, as some employee, branch, job Combination may not have a manager.

Page No. 11
Date

Q.4) Draw an E-R diagram of Hospital management System.



Q.5] Explain different types of constraints.

Ans) (I) Constraints enforce limits to the data or type data that can be inserted/deleted/updated from a table.

(II) The whole purpose of constraints is to maintain the data Integrity during an update/delete/insert queries into a table.

(III) Types of Constraints:-

(1) NOT NULL:- NOT NULL constraint makes sure that a column doesn't hold NULL value. If we don't specify this Constraint and don't provide value to particular column during insertion then by default it takes NULL.

(2) UNIQUE: UNIQUE constraint ensures that ~~to~~ two rows don't have same values for same column. Duplication of values in a column cannot be done if it is provided UNIQUE Constraint.

(3) DEFAULT: The DEFAULT constraint provides a default value to a column when there is no value provided while inserting a record into a table.

(4) CHECK:- When this constraint is being set on a column, it ensures that the specified column must have the value falling in the specified range.

(5) Key constraints:-

(i) PRIMARY KEY:- PRIMARY KEY is a Candidate Key selected by data base designer, used to differentiate between two rows of a table since, Primary key is uniquety for each entity in a entity set and it cannot be null.

(ii) FOREIGN KEY:- Foreign keys are the columns of a table that points to the unique or primary key of another table. They act as a cross-reference between tables.

(6) DOMAIN constraints:- (i) Domain constraints restrict attributes to have values only on their domains.
E.g:- Integer, float, number can only have numeric values
Varchar & varchar2 only have Strings of ≤ predefined length.

(ii) And it also restrict values on basis of constraint we have seen until now.

(7) Relationship Constraints:-

(i) Mapping cardinalities or cardinality ratio:-

(I) One-to-one:- We will consider this relation and all other cardinalities have binary relation. (Includes two entity sets)

(I) one-to-one:- Let, us consider consider two entity sets A & B connected through Relation R. In one-to-one cardinality constr-aint one entity of A is related to at most one entity of B and vice-versa.

(II) one-to-many:- In this constraint one entity of A is related to one or more entities of B. one entity of B is related to atmost one entity of A.
If we reverse the meaning of above statements ~~say~~ (inversion) we would get many-to-one constraint.

(III) many-to-many:- In this constraint one entity of A is related to one or more entities of B & vice-versa.

7.) (i) Participation constraint:- Consider, two entity sets A and B & Relation R
(I) ~~Total-Relation~~:
(I) Total-Participation:- If each entity of A is associated in a relationship R then A is said to have total-participation with R, Same goes for B.

(II) Partial-Participation:- If only some entities of A are associated in a relationship R the A is said to have partial-partcipation with R, Same goes for B.

Q.6) Consider the following database tables:-

Employee (empname, street, city, date_of_joining)
Work (empname, company_name, salary)
Company (company_name, city)
Manages (empname, manager_name)

Write SQL Queries for the following Statements:

i) Modify the database so that employee "Amruta" now leave in "KonaKan"

Ans) update Employee set city = "KonaKan"
where empname = "Amruta".;

ii) Find the number of employees in each city with date_of_joining as "01-Jul-2018"

Ans) Select count(empname) from Employee
where date_of_joining = "01-Jul-2018";

iii) List name of companies starting with letter "A

Ans) Select *
Select (company_name) from Company
where company_name like 'A%';

iv)) Display empname, manager_name, street, city only for employees having manager

Ans) Select Manages.empname, Manages.manager_name, Employee.street, Employee.city
from Manages INNER JOIN Employee
On Manages.empname = Employee.empname;

v) Give total number of employee.

Ans) Select Count(empname) from Employee;

Page :
Date : / /

Assignment 2

Q.1) Explain the following relational algebra operation:-

a) Cartesian Product:- ① The Cartesian-Product operation, denoted by a cross (x), allows us to combine information from any two relations. We write the Cartesian product of relations γ_1 and γ_2 as $\gamma_1 \times \gamma_2$.

② A relation is a subset of a Cartesian product of a set of domains.

e.g:- If $C = \{\gamma_1, \gamma_2, \gamma_3, \ldots\}$ i.e $\gamma_1 \times \gamma_2 \times \gamma_3 \times \ldots$

$\therefore \gamma_1, \gamma_2, \gamma_3, \ldots \subseteq \gamma_1 \times \gamma_2 \times \gamma_3 \times \gamma_4 \times \ldots$

③ Since the same attribute name may appear in both γ_1 and γ_2, to distinguish between these attributes we devise a naming schema.

④ Let us see by example given below:-

let, γ_1 = borrower (customer_name, loan_number)

γ_2 = loan (loan_number, branch_name, amount)

let, $\gamma = \gamma_1 \times \gamma_2$

then, γ = borrower x loan (customer_name, borrower.loan_number, loan.loan_number, branch_name, amount)

∴ loan_number in both (γ_1 & γ_2) we use naming as relation_name.attribute_name to distinguish between them.

Customer_name	loan_number
Adams	L-16
Curry	L-93

The borrower relation (r_1)

loan_number	branch_name	amount
L-11	Round Hill	900
L-14	Downtown	1500

The loan relation (r_2)

Customer_name	borrower. loan_number	loan. loan_number	branch_name	amount
Adams	L-16	L-11	Round Hill	900
Curry	L-93	L-11	Round Hill	900
Adams	L-16	L-14	Downtown	1500
Curry	L-93	L-14	Downtown	1500

borrower × loan ($r_1 \times r_2$)

b) Natural-Join:- ① The natural join is a binary operation that allows us to combine certain selections and a Cartesian product into one operation.

② The natural-join operation forms a Cartesian product of its two arguments, performs a selection forcing equality on those attributes that appear in both relation schemas, and finally removes duplicate attributes.

③ Let, us see how its word by above example:-

$r_1 \overset{\text{natural}}{\underset{\text{Join}}{\bowtie}} r_2 = \sigma_{\text{borrower.loan\_number} = \text{loan.loan\_number}} (\text{borrower} \times \text{loan})$

④ We select those tuples in borrower x loan in which borrower.loan_number = loan.loan_number.

c)» Generalize projection:- ① The generalized-projection operation extends the projection operation by allowing arithmetic functions to be used in the project list. The generalized-projection operation has the form:- $\Pi_{F1,F2,F3,\ldots,Fn}(E)$

② Where E is any relational-algebra expression, and each $F_1, F_2, F_3, \ldots, F_n$ is an arithemetic expression involving constants & attributes in the schema of E. As a special case, the arithmetic expression may be simply an attribute or constant.

③ e.g:- Let, us take a relation with credit_info having attributes (customer_name, limit, credit_balance) using generalized projection:-

$\Pi_{customer\_name, (limit - credit\_balance)}(credit\_info)$

we will get tuple of customer_name and amount of limit-credit_balance we can give it a name using 'as'.

customer_name	limit	credit-balance	
Curry	2000	1750	
Hayes	1500	1500	← credit_info
Jones	6000	700	relation.
Smith	2000	400	

Customer_name	limit - credit_balance	
Curry	250	← generalized projection.
Jones	5300	
~~Hayes~~	~~500~~	
Smith	1600	

d)) Set operation:-

(i) Union operation:- (1) If we want to show all the elements of two different relations in one table (can be more than two) having same number of attributes and corresponding attribute have same domain we use union operation.

e.g:- $\Pi_{customer\_name}(borrower) \cup \Pi_{customer\_name}(depositer)$
↑ (Union operator)

(2) It will give name of customers appear in either or both relations and will remove duplicate tuples.
(3) If we don't want to remove duplicate tuples we use Union all.

(ii) Set-Difference operation:- (1) The set-difference operation, denoted by −, allows us to find tuples that are in one relation but are not in another.
(2) The expression r-s yields relation containing those tuples in r but not in s.

e.g:- $\Pi_{customer\_name}(borrower) - \Pi_{customer\_name}(depositer)$
↑ Difference operator.

(3) It will give name of customers which appears only in borrower and remove all depositor name from borrower name.
(4) It follows same constraint as Union operation. i.e, relations should be of same arity, and that the domains of the i^{th} attribute of r and the i^{th} attribute of s be the same.

(iii) Set Intersection operation:- (1) Intersection operation will yield tuples similar in both relations.

(2) e.g. $\Pi_{customer\_name}(borrower) \cap \Pi_{customer\_name}(depositor)$
$\cap$ Intersection operator

(3) It will give customer name which appears in both borrower & depositor

(4) following equality holds:- $r \cap s = r - (r - s)$

(4) It follows same constraints as Union & Difference operations.

Q.2) Define and explain normalization with example.

(1) Normalization is a process of designing a consistent database by minimizing redundancy and ensuring data integrity through decomposition which is lossless.

(2) Normalization is step by step decomposition of complex records into simple records.

(3) Goals of database Normalization:-
(i) Ensures the integrity.
(ii) Prevents redundancy in data.
(iii) To avoid data anomaly.
<<a>> Update anomaly <<b>> Insertion anomaly
<<c>> Deletion anomaly.

(4) Normalization process will make, use of some type of keys to remove the redundancies present in data of relational tables:

5)» Types of Normal Forms:-
(i) First Normal Form
(ii) Second Normal Form
(iii) Third Normal Form
(iv) ~~BCNF~~ BCNF Boyce-Codde Normal Form.

(6) (i) First Normal form (1NF):- (1) A database is said to be in 1NF if no table have multivalued attribute i.e each attribute of table have Single domain in 1NF.
(2) If a relation is not in 1NF then it can be translated to ~~one~~ 1NF by decomposing relation in two parts :- relation 1 will contain all Single domain attributes of original relation & relation 2 will have attributes of multivalued domain of original relation & both relation Should have candidate key to uniquely identify each tuple & rejoin them in original relation.

(7) Second Normal Form (2NF):- (1) If database have Relation set $R = \{R_1, R_2, R_3, \ldots\}$ & Functional dependency on R say set $F = \{F_1, F_2, \ldots\}$ then each element of F Should show fully functional dependency for database to be in a 2NF.

(8) Third Normal Form:- A relation is in 3.N.F if all non-prime attributes are:-
(i) Fully functionally dependent on primary key.
(ii) Non-transitive dependent on every key.

Page :
Date : / /

9)) Boyce-Codd Normal Form (B.C.N.F) :- ① A relation R is said to be in a B.C.N.F if determinant of functional dependency contained candidate key i.e If $X \rightarrow Y$ then each attributes of X should contained in a candidate key.
② If a relation is in B.C.N.F then it is in 3NF also.

e.g:- Relation : R (A, B, C, D, E, F, G, H, I, J)
F.D $F = \{AB \rightarrow C, C \rightarrow EF, AD \rightarrow GH, G \rightarrow I, H \rightarrow J\}$
Convert to highest Normal form.

a) 1NF :- Assuming all attributes ~~store~~ are atomic domains ∴ R is in 1NF

b) 2NF :-

$\{ABD\}^+ = \{A, B, C, D, E, F, G, H\}$
ABD is a candidate Key of R but no ~~fun~~ attribute in relation R is fully functional dependent on above candidate key.
∴ R is not in 2NF ∴ decomposing it in 2NF.
∴ $R_1 (A, B, C, E, F)$; $AB \rightarrow C$; $C \rightarrow EF$
$R_2 (A, D, G, H, I, J)$; $AD \rightarrow GH$; $G \rightarrow I$; $H \rightarrow J$

c) 3NF:- Relation R_1 is not in 3·N·F
$\therefore AB \rightarrow C$ & $C \rightarrow EF$ $\therefore AB \rightarrow EF$
Relational R_2 is not in 3·N·F
$\therefore AO \rightarrow GH$ & $G \rightarrow I$ & $H \rightarrow J$
$\therefore AO \rightarrow I$; $AO \rightarrow J$

Decomposing R_1 & R_2 in 3·N·F

R_1 (A, B, C)	$AB \rightarrow C$
R_2 (C, E, F)	$C \rightarrow EF$
R_3 (A, D, G, H)	$AD \rightarrow GH$
R_4 (G, I)	$G \rightarrow I$
R_5 (H, J)	$H \rightarrow J$

we can combine R_4 & R_5 above.

d) B·C·N·F:-

Relation	F·D	Determinant	Key
R_1 (A, B, C)	$AB \rightarrow C$	A B	A, B
R_2 (C, E, F)	$C \rightarrow EF$	C	C
R_3 (A, D, G, H)	$AD \rightarrow GH$	A D	A, D
R_4 (G, I)	$G \rightarrow I$	G	G
R_5 (H, J)	$H \rightarrow J$	H	H

Every determinant in each relation is a primary.
$\therefore$ All Relations above is in BCNF.

Page :
Date : / /

Q.3) Explain types of functional dependency.

Ans) ① Functional dependencies are restrictions imposed between two set of attributes in a relation from a database.

② In a relation R with attributes X and Y represented as R(X, Y), where Y is functional dependent on other column X or X functionally determines Y

$X \rightarrow Y$ i.e $Y = f(X)$

X is called Determinant & Y is called determine.

Types of functional dependencies:-

1) Fully functional dependcy:- A functional dependency is a fully functional dependency if removal of any attribute from determinant invalidate the dependency.

emp_id	ename	Salary	Proj_id	Hours	allowance
10	Suresh	50000	E59	44	40000
12	Mahesh	25000	B26	31	30000
15	Adam	26000	E78	23	20000
18	Jamal	50000	A89	12	15000

employee relation

If we take following F.D:-

emp_id, Proj_id → Hours, allowance

the removal of emp_id or proj_id will invalidate our F.D i.e we won't be able to determine Hours and allowance

∴ It is fully functional dependency.

Page :
Date : / /

2) Partial Functional dependency :-

(i) F.D is said to be Partial F.D if $A \rightarrow B$ & if remove $x \in A$ where x is an attribute in A, doesn't invalidate F.D i.e F.D will hold.

(ii) In Partial F.D only part of determinant is needed for determine

If $A \rightarrow B$ where $A = \{x, y\}$

$x, y \rightarrow B$

If $y \rightarrow B$ holds then

$A \rightarrow B$ is Partial F.D.

(iii) e.g :- emp_id, proj_id → ename, salary

If we remove proj_id from above functional dependency we can still determine employee name & salary using emp_id i.e only emp_id is necessary determinant in above F.D ∴ it is a partial Functional dependency.

3) Transitive dependency :- If $X \rightarrow Y$ & $Y \rightarrow Z$ then $X \rightarrow Z$ is a transitive functional dependency

e.g :-

emp_id	name	salary	dep_id	d_name
10	Adam	55,000	ME05	Mechanical
11	James	80,000	CS 10	Computer
14	Jamal	40,000	CV09	Civil
15	Qazi	75,000	IT69	I.T

In above relation we see that emp_id → dep_id

and dep_id → d_name so we can write emp_id → d_name i.e emp_id is i.e dname is transitively dependent on emp_id.

4) Trivial functional dependency:-

If $X \to Y$ & $Y \subseteq X$ then the functional dependency is said to be Trivial functional dependency.

If $X \to Y$ & $Y \nsubseteq X$ then the F.D is said to be non-Trivial functional dependency.

e.g:- emp_id, ename → ename ← Trivial F.D
emp_id, proj_id → allowance ← Non-Trivial F.D

5) Multivalued dependency:-

Multivalues dependency defined by $X \twoheadrightarrow Y$ is said to be hold for $R(X, Y, Z)$ if for a given set of values of X there is a set of associated values of attribute Y. Y values depends only on X values and have no dependence on the set of attribute Z.

e.g:-

emp_id	ename	Car
10	Mahesh	Speedwagon
12	Suresh	Suzuki 69
15	Ganesh	Hyundai 110
10	Mahesh	BMW 420

emp_id $\twoheadrightarrow$ Car.

Page :
Date : / /

Q.4)) Explain lock based Protocol.

Ans) ① Database System equipped with lock-based Protocols use a mechanism by which any transaction cannot read or write data unit it acquires on appropiate lock on it.

② Locks are of two kinds:-

(i) Binary Locks:- A lock on a data item can be in two states; it is either ~~or~~ locked or unlocked.

(ii) Shared / Exclusive Locks:- This type of locking mechanism differentiate the locks based on their uses. If a lock is acquired on a data item to perform a write operation, it is an exclusive lock. Allowing more than one transaction to write on the same data item would lead the database into inconsistent state. Read locks are Shared lock because no data value is being exchanged.

③ There are four types of lock Protocols available:-

(i) Simplistic lock Protocol:- 1) Simplistic lock-based Protocols allow transactions to obtain a lock on every object before a 'write' operation is performed.

2) Transaction may unlock the data item after completing the 'write' operation.

(ii) Pre-claiming lock Protocol:- 1) It evaluate their operations and create a list of data items on which they need locks.

2) Before initiating an execution, the transaction requests the system for all the locks it needs before hand.

2)(ii) 3) If all the locks are granted the transaction executes and released all the locks when all its operations are over. If all the locks are not granted the transactions roll back and waits until all the locks are granted.

3)(iii) Two-phase locking 2PL :- 1) This locking protocol divides the execution phase of trasaction into three parts.
2) In the first part, when the transaction start executing, it seeks permission for the locks it requires
3) The second part, is where the transaction acquire all the locks. As soon as transaction releases its first lock.
4) In the third part, the transaction cannot demand any new locks; it only releases the acquired locks.
5) Two-phase locking has two phases, one is growing, where all the locks are being acquired by the transaction; & the second phase is shrinking, where the locks held by the transaction are being released.
6) To claim an exclusive (write) lock, a transaction must find acquire a shared [read] lock & then upgrade it to an exclusive lock.

(iv) Strict Two-Phase locking :- (1) The first phase of Strict-2PL is same as 2PL. After acquiring all the locks in the first phase, the transaction continues to execute normally.
(2) But in contrast to 2PL, Strict-2PL doesn't releases a lock after using it. It holds all the lock until commit point & releases all the lock at a time.
(3) Strict-2PL doesn't have cascading abort as 2PL does.

Page :
Date : / /

5) Explain log-based recovery.

Ans) ① The log is a sequence of records. Log of each transaction is maintained in some stable storage so that if any failure occurs then it can be recovered from there.

② If any operation is performed on the database, then it will be recorded in the log.

③ But the process of storing the logs should be done before the actual transaction is applied in the database.

④ let's assume there is a transaction to modify the city of a student. The following logs are written for this transaction :-

(i) When the transaction is initiated, then it writes 'start' log

⟨Tn, Start⟩

(ii) When the transaction modifies the city from 'Noida' to 'Bangloxe', then another log is written to the file.

⟨Tn, city, 'Noida', 'Bangloxe'⟩

(iii) When the transaction is finished, then it writes another log to indicate end of transaction

⟨Tn, Commit⟩

⑤ There are two approaches to modify the database :-

(i) Defined database modification:

- It occurs if the transaction does not modify the database until it has committed.
- In this method, all the logs are created and stored in the stable storage, and the database is updated when a transaction commits.

(ii) Immediate database modification:-
- It occurs if database modification occurs while the transaction is still active.
- In this database is modified immediately after every operation. It follows an actual database modification.

(6) Recovery using log records:-
(i) When the system is crashed, then the system consults the log to find which transactions need to be undone and which needs to be redone.
(ii) If the log contains the record $\langle T_n, Start \rangle$ & $\langle T_n, commit \rangle$ then transaction needs to be redone.
(iii) If log contains record $\langle T_n, Start \rangle$ but doesn't contain the record either $\langle T_n, commit \rangle$ or $\langle T_n, abort \rangle$, then the transaction T_n needs to be undone.