Project Report

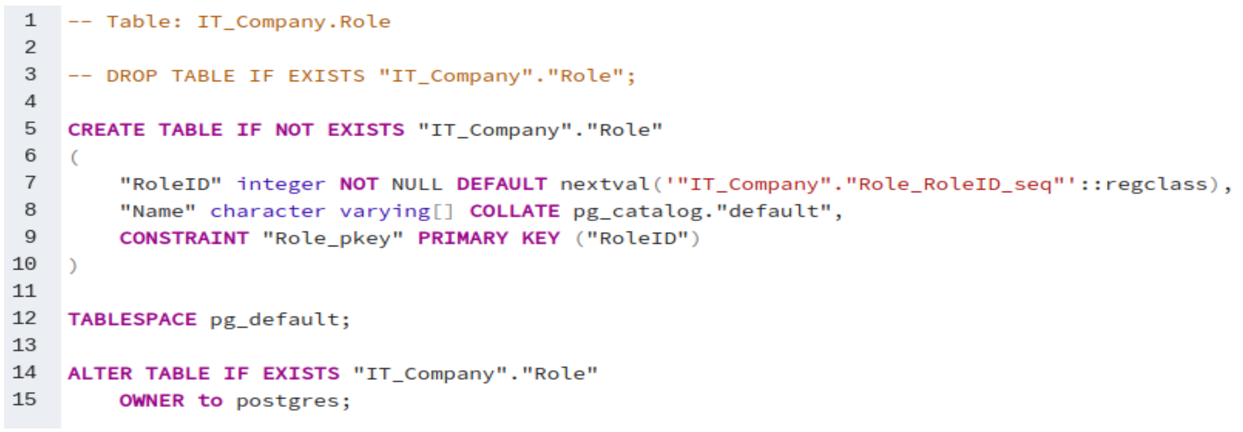
CT60A7650 Database Systems Management

Jhama kabir mim
Student ID : 000327482

Task 1

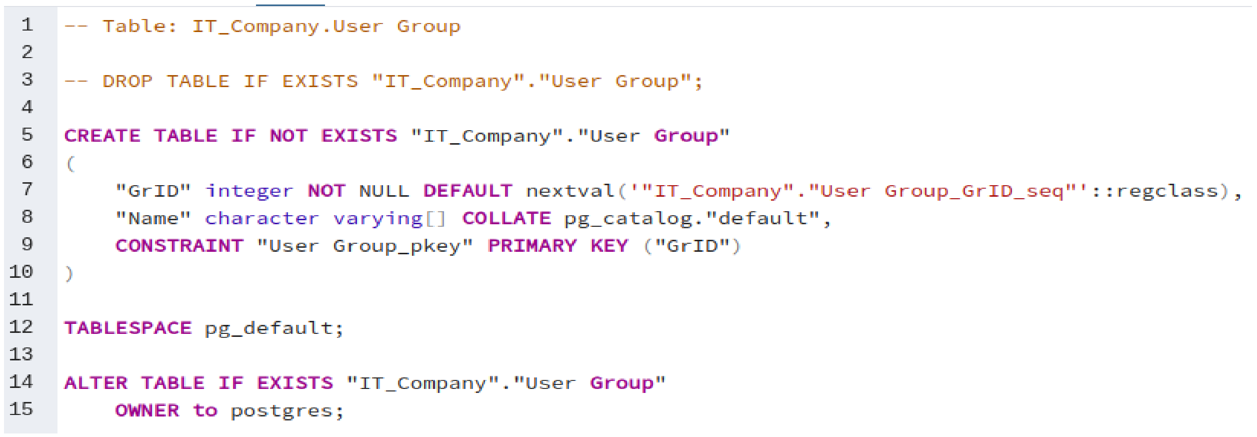
A database schema has been built based on the ER diagram provided.The implementation contains 10 tables altogether.The DDL statements are given below

Table Role



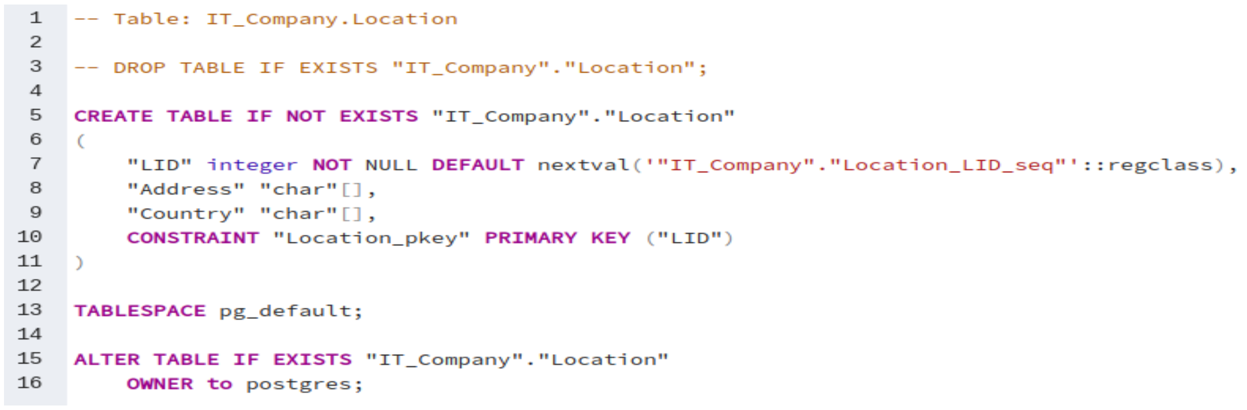
```
-- Table: IT_Company.Role

-- DROP TABLE IF EXISTS "IT_Company"."Role";

CREATE TABLE IF NOT EXISTS "IT_Company"."Role"
(
    "RoleID" integer NOT NULL DEFAULT nextval('"IT_Company"."Role_RoleID_seq"'::regclass),
    "Name" character varying[] COLLATE pg_catalog."default",
    CONSTRAINT "Role_pkey" PRIMARY KEY ("RoleID")
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS "IT_Company"."Role"
    OWNER to postgres;
```

Table User Group



```
-- Table: IT_Company.User Group

-- DROP TABLE IF EXISTS "IT_Company"."User Group";

CREATE TABLE IF NOT EXISTS "IT_Company"."User Group"
(
    "GrID" integer NOT NULL DEFAULT nextval('"IT_Company"."User Group_GrID_seq"'::regclass),
    "Name" character varying[] COLLATE pg_catalog."default",
    CONSTRAINT "User Group_pkey" PRIMARY KEY ("GrID")
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS "IT_Company"."User Group"
    OWNER to postgres;
```

Table Location



```
-- Table: IT_Company.Location

-- DROP TABLE IF EXISTS "IT_Company"."Location";

CREATE TABLE IF NOT EXISTS "IT_Company"."Location"
(
    "LID" integer NOT NULL DEFAULT nextval('"IT_Company"."Location_LID_seq"'::regclass),
    "Address" "char"[],
    "Country" "char"[],
    CONSTRAINT "Location_pkey" PRIMARY KEY ("LID")
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS "IT_Company"."Location"
    OWNER to postgres;
```

Table Customer

```
CREATE TABLE IF NOT EXISTS "IT_Company"."Customer"
(
    "CID" integer NOT NULL DEFAULT nextval('"IT_Company"."Customer_CID_seq"'::regclass),
    "Name" "char"[],
    "Email" "char"[],
    "LID" integer NOT NULL,
    CONSTRAINT "Customer_pkey" PRIMARY KEY ("CID"),
    CONSTRAINT "LID_FK" FOREIGN KEY ("LID")
        REFERENCES "IT_Company"."Location" ("LID") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE SET NULL
)
```

Table Project

```
CREATE TABLE IF NOT EXISTS "IT_Company"."Project"
(
    "PrID" integer NOT NULL DEFAULT nextval('"IT_Company"."Project_PrID_seq"'::regclass),
    "Name" "char"[],
    "Budget" "char"[],
    "StartDate" date,
    "Deadline" date,
    "CID" integer,
    CONSTRAINT "Project_pkey" PRIMARY KEY ("PrID"),
    CONSTRAINT "CID_FK" FOREIGN KEY ("CID")
        REFERENCES "IT_Company"."Customer" ("CID") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE NO ACTION
)
```

Table Department

```
-- Table: IT_Company.Department

-- DROP TABLE IF EXISTS "IT_Company"."Department";

CREATE TABLE IF NOT EXISTS "IT_Company"."Department"
(
    "DepID" integer NOT NULL DEFAULT nextval('"IT_Company"."Department_DepID_seq"'::regclass),
    "Name" "char"[],
    "LID" integer,
    CONSTRAINT "Department_pkey" PRIMARY KEY ("DepID"),
    CONSTRAINT "LID_FK" FOREIGN KEY ("LID")
        REFERENCES "IT_Company"."Location" ("LID") MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS "IT_Company"."Department"
    OWNER to postgres;
```

Table Employee

```
CREATE TABLE IF NOT EXISTS "IT_Company"."Employee"
(
    "EmpID" integer NOT NULL DEFAULT nextval('"IT_Company"."Employee_EmpID_seq"'::regclass),
    "Email" "char"[],
    "Name" "char"[],
    "DepID" integer,
    CONSTRAINT "Employee_pkey" PRIMARY KEY ("EmpID"),
    CONSTRAINT "DepID_FK" FOREIGN KEY ("DepID")
        REFERENCES "IT_Company"."Department" ("DepID") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE SET NULL
)
```

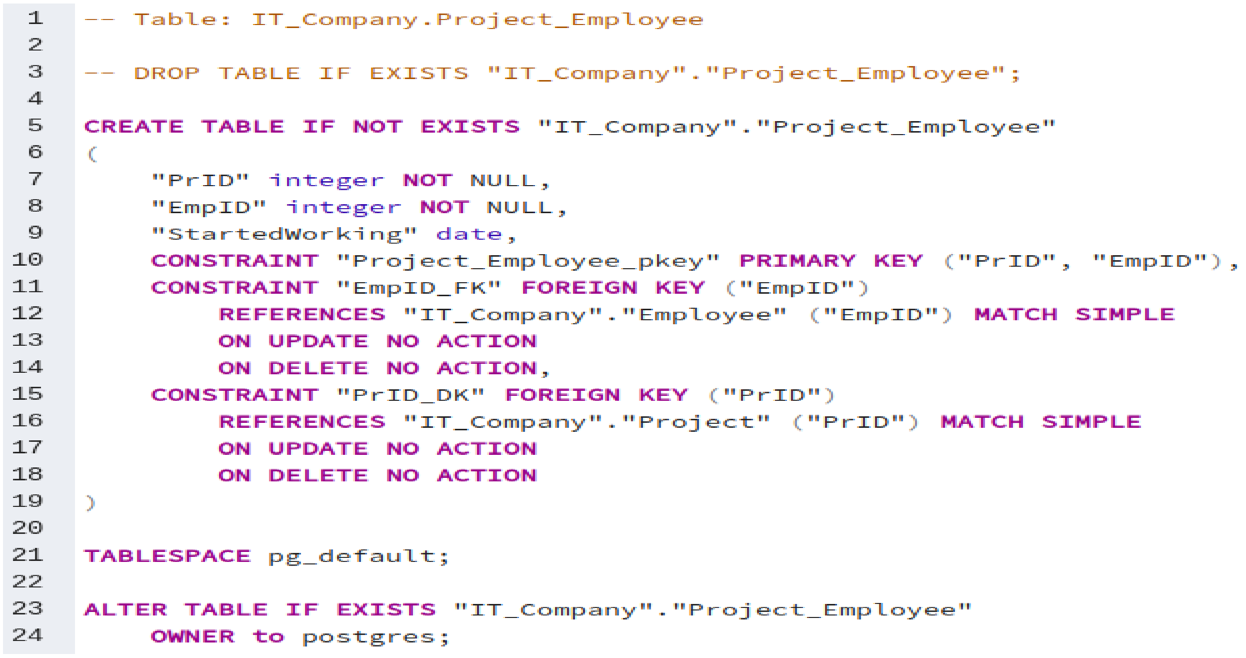
TableEmployee_Role

```
-- Table: IT_Company.Employee_Role

-- DROP TABLE IF EXISTS "IT_Company"."Employee_Role";

CREATE TABLE IF NOT EXISTS "IT_Company"."Employee_Role"
(
    "EmpID" integer NOT NULL,
    "RoleID" integer NOT NULL,
    CONSTRAINT "Employee_Role_pkey" PRIMARY KEY ("EmpID", "RoleID"),
    CONSTRAINT "EmpID_FK" FOREIGN KEY ("EmpID")
        REFERENCES "IT_Company"."Employee" ("EmpID") MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT "RoleID_FK" FOREIGN KEY ("RoleID")
        REFERENCES "IT_Company"."Role" ("RoleID") MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS "IT_Company"."Employee_Role"
    OWNER to postgres;
```

Table Employee_UserGroup

```
-- Table: IT_Company.Employee_UserGroup

-- DROP TABLE IF EXISTS "IT_Company"."Employee_UserGroup";

CREATE TABLE IF NOT EXISTS "IT_Company"."Employee_UserGroup"
(
    "EmpID" integer NOT NULL,
    "GrID" integer NOT NULL,
    CONSTRAINT "Employee_UserGroup_pkey" PRIMARY KEY ("EmpID", "GrID"),
    CONSTRAINT "EmpID_FK" FOREIGN KEY ("EmpID")
        REFERENCES "IT_Company"."Employee" ("EmpID") MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT "GrID_FK" FOREIGN KEY ("GrID")
        REFERENCES "IT_Company"."User Group" ("GrID") MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS "IT_Company"."Employee_UserGroup"
    OWNER to postgres;
```

Table Project_Employee



```
-- Table: IT_Company.Project_Employee

-- DROP TABLE IF EXISTS "IT_Company"."Project_Employee";

CREATE TABLE IF NOT EXISTS "IT_Company"."Project_Employee"
(
    "PrID" integer NOT NULL,
    "EmpID" integer NOT NULL,
    "StartedWorking" date,
    CONSTRAINT "Project_Employee_pkey" PRIMARY KEY ("PrID", "EmpID"),
    CONSTRAINT "EmpID_FK" FOREIGN KEY ("EmpID")
        REFERENCES "IT_Company"."Employee" ("EmpID") MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT "PrID_DK" FOREIGN KEY ("PrID")
        REFERENCES "IT_Company"."Project" ("PrID") MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS "IT_Company"."Project_Employee"
    OWNER to postgres;
```

Task 2

<u>Define ON UPDATE and ON DELETE rules for Project, Customer, User group, and Employee</u>

On Update and On Delete are two ways to put constraints on a schema.Both operation can take several values like Cascade , No Action, Set NULL etc.

Project table has one foreign key i.e. CID, which is the primary key of Customer table.If a customer’s ID gets updated , then it’s expected that it will also be updated in project table.Therefore,**On Update should have value Cascade** .

If On Delete is set to Cascade , then deletion of CID from customer table will ensure deletion of corresponding projects.It might be dangerous because multiple project of a customer can be deleted.Setting **On Delete to No Action** won’t be this much fatal since no project entry would be deleted.

In Customer table,there’s one foreign key, LID, which is the primary key of Location table.**On Update should be set to Cascade** so that any change in LID of Location table is passed on to Customer table too.**On Delete can be set to Set NULL** because we don’t want any customer info to be removed just because it’s Location info has been deleted.We want it to set to NULL.

User group table doesn’t have any foreign key.But there’s a join table,Employee_UserGroup, to implement the many to many relationship between Employee and Usergroup.The field GroupID is a foreign key in this table.**Both On**

Delete and On Update should have value Cascade because if GroupID of UserGroup table is updated or deleted, it should be passed on to Employee_UserGroup table too to maintain consistency of data.

In Employee table,there's a foreign key,DepID,which is the primary key of Department table.If a department entry is deleted from Department table,corresponding all employee table should have depID value set to NULL.So **On Delete SET NULL** should be used. And **On Update Cascade** should be used since we want the updates to pass along.

Task 3

<u>Define at least two different ways to partition the data. Justify your decisions.</u>
Partitioning a table comes in handy when there are lots of entries in that table for example a table with 10 Million rows.When we want to retrieve a data using a select query,it either performs a bitmap index scan if there's an index on the column we are conditioning or a sequential scan if there's no index.We can further improve the running time of a query by partitioning a table.If there are 10 Million rows in a table and we partition it into two blocks , one containing first 0.5 Million rows and other one containing remaining rows,then we've to perform sequential scan only on the partition that contains the data,not on the entire table.Hence,runtime is greatly improved.

In our schema, (1)Employee table can be horizontally partitioned since it has 100 entries for now and can increase in future.Another way of partitioning is to (2) partition the same table vertically.Vertical partitioning breaks up the table breaks up the table based on a column's value.

Task 4

<u>Define access rights using PostgreSQL syntax using the following knowledge:</u>
<u>Each user group has access to the project they work on</u>

In Postgres , we can create group of users.Although there's no concept of users in postgres rather there's a concept of role.A user is provided with some roles of what he can do and what not.Also we can create group of roles.

We can create a role with certain permissions and then grant those roles to other roles.Roles will inherit the permissions of roles granted to them, if those roles have the inherit attribute.

To create a group role ,

```
CREATE ROLE <groupname>;
```

Once the group role is created , we can add roles to the group roles using grant.

```
GRANT <groupname> TO <role>
```

Task 5

a.Define Default Values

While creating a table, we can specify default values for some fields.In our schema , it's been told that The company has offices in three different locations within one country.So we can define a default value for Country field in Location table.

```
CREATE TABLE IF NOT EXISTS "IT_Company"."Location"
(
    "LID" integer NOT NULL DEFAULT nextval('"IT_Company"."Location_LID_seq"'::regclass),
    "Address" "char"[],
    "Country" "char"[] DEFAULT "Finland",
    CONSTRAINT "Location_pkey" PRIMARY KEY ("LID")
)
```

b.Define check constraints

We can set up constraints on some field's value.In our schema , the StartDate field of Project table must be greater than current date.

```
CREATE TABLE IF NOT EXISTS "IT_Company"."Project"
(
    "PrID" integer NOT NULL DEFAULT nextval('"IT_Company"."Project_PrID_seq"'::regclass),
    "Name" "char"[],
    "Budget" "char"[],
    "StartDate" date CHECK (StartDate >= now()),
    "Deadline" date,
    "CID" integer,
    CONSTRAINT "Project_pkey" PRIMARY KEY ("PrID"),
    CONSTRAINT "CID_FK" FOREIGN KEY ("CID")
        REFERENCES "IT_Company"."Customer" ("CID") MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE NO ACTION
)
```

c.Define how to manage NULL values

Postgres provides NULLIF function to handle null values.

NULLIF(argument_1,argument_2);

```
SELECT
       NULLIF (1, 1); -- return NULL

SELECT
       NULLIF (1, 0); -- return 1
```

The NULLIF function returns a null value if argument_1 equals to argument_2, otherwise it returns argument_1.

Task 6

<u>Create at least three different (and useful) triggers using PostgreSQL syntax and trigger graph.</u>

Three different triggers from our schema

a. **EmpAuditTrigger after an employee is inserted**

After an employee is inserted , an entry is added to Employee_Audit Table.

Code for creating Employee_Audit table

```
-- Table: IT_Company.Employee_Audit

-- DROP TABLE IF EXISTS "IT_Company"."Employee_Audit";

CREATE TABLE IF NOT EXISTS "IT_Company"."Employee_Audit"
(
    "EmpID" integer NOT NULL,
    "EntryDate" date,
    CONSTRAINT "Employee_Audit_pkey" PRIMARY KEY ("EmpID")
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS "IT_Company"."Employee_Audit"
    OWNER to postgres;
```

Code of Trigger function

```
-- FUNCTION: IT_Company.empAuditFunc()

-- DROP FUNCTION IF EXISTS "IT_Company"."empAuditFunc"();

CREATE OR REPLACE FUNCTION "IT_Company"."empAuditFunc"()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE NOT LEAKPROOF
AS $BODY$
Begin
INSERT INTO Employee_Audit(EmpID , EntryDate) VALUES (new.EmpID, now());
      RETURN NEW;

End
$BODY$;

ALTER FUNCTION "IT_Company"."empAuditFunc"()
    OWNER TO postgres;
```

.

Code of creating trigger

```
-- Trigger: EmpAuditTrigger

-- DROP TRIGGER IF EXISTS "EmpAuditTrigger" ON "IT_Company"."Employee";

CREATE TRIGGER "EmpAuditTrigger"
    AFTER INSERT
    ON "IT_Company"."Employee"
    FOR EACH ROW
    EXECUTE FUNCTION "IT_Company"."empAuditFunc"();
```

b.Trigger to be executed on deadline update of a project
Code of trigger function to insert into Ptoject_Update table

```
CREATE OR REPLACE FUNCTION "IT_Company"."projectUpdateFunc"()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE NOT LEAKPROOF
AS $BODY$
Begin
IF new.Deadline <> old.Deadline THEN
    INSERT INTO Project_Updates(PrID , NewDeadline, EntryDate)
    VALUES(new.PrID , new.Deadline , now());
END IF;
End
$BODY$;
ALTER FUNCTION "IT_Company"."projectUpdateFunc"()
    OWNER TO postgres;
```

Code of trigger creation

```
CREATE TRIGGER "projectUpdateTrigger"
    AFTER UPDATE
    ON "IT_Company"."Project"
    FOR EACH ROW
    EXECUTE FUNCTION "IT_Company"."projectUpdateFunc"();
```

c.Trigger to audit project entries
Whenever a project entry is inserted,it’s kept track of through Project_Audit table which contains entry time and project duration.

Code of trigger function

```
CREATE OR REPLACE FUNCTION "IT_Company"."projectAuditFunc"()
    RETURNS trigger
    LANGUAGE 'plpgsql'
    COST 100
    VOLATILE NOT LEAKPROOF
AS $BODY$
Begin
INSERT INTO Project_Audit(PrID , EntryDate , Duration)
VALUES (new.PrID, now() , old.Deadline - old.StartDate);
      RETURN NEW;
End
$BODY$;
ALTER FUNCTION "IT_Company"."projectAuditFunc"()
    OWNER TO postgres;
```

Code of binding this function to a trigger

```
CREATE TRIGGER "ProjectAuditTrigger"
    AFTER INSERT
    ON "IT_Company"."Project"
    FOR EACH ROW
    EXECUTE FUNCTION "IT_Company"."projectAuditFunc"();
```

Task 7

Come up with at least three different security issues and have a counter measure for each.

Three security issues in database design and their preventive measures

a.Failures during deployment

A common reason for an insecure database is the negligence during deployment phase.Although functional testing and different types of software testing are usually performed on the to-be-deployed application (including its database), such tests don't test the security of the database. If the database is doing something wrong, software testing won't be able to detect it.

The solution is to perform penetration and security testing so that there is zero loopholes in the database.

b.Flaws in database features

a feature may be exploited by cybercriminals to attack the database.The less-used features usually are the last to get fixes for their vulnerabilities.In some cases, their bugs are not even discovered until it's too late, i.e., less-used features or tools present more security risks.
The chances of possible attacks must be minimized by removing less-used or unnecessary features. Then, the database will also get simpler for testing, finding, and fixing the bugs.

c.Poor Encryption Standards

A database has a networking interface which might be tracked by cybercriminals and data coming in and out of the database may be read by them.
Database admins should encrypt all traffic of the database using an industry-standard encryption scheme, like SSL or TLS.

Task 8

a.Define the backup method and schedule. Justify your decisions.

Backup scheduling is an integral part of database maintenance. Determining how to schedule backup is both a function of how often data changes and how valuable the data is.Backup should be scheduled periodically and as often as necessary to minimize the consequence of data loss.
Sometimes.it might be necessary to do a complete backup.A complete database backup is appropriate when we need to capture the entire database and every record of it.
Sometimes,we need to backup only what has changed.This kind of backup speeds up the process of frequent backups since only modified portion is backed up.

Since the storage amount of our schema is not that high , we can consider doing a complete backup which will provide us with more reliability and security of data.

b.Come up with at least three realistic disasters and one extremely unlikely disaster.

In our schema , there might be everal realistic disasters.For example , if permissions aren't granted properly , an employee titled to work on a project might work in other projects where he isn't assigned to.There might be lack of proper backup scheduling plan and thus loss of data might be quite obvious.There might be security issues arising out of lack of data encryption.

A disaster that is unlikely is the unavailability of services since the company has roughly 50 concurrent users at a time.

c.Define the recovery method from the disasters

To get around the problem of roles and permissions , it must be carefully taken care of at the application level that an employee belonging to a group of users can only perform tasks assigned to that group of users.

To deal with availability an backup plans , proper scheduling , in this case , complete scheduling should be followed.

To improve security , data encryption and decryption must be performed while reading and writing to database.