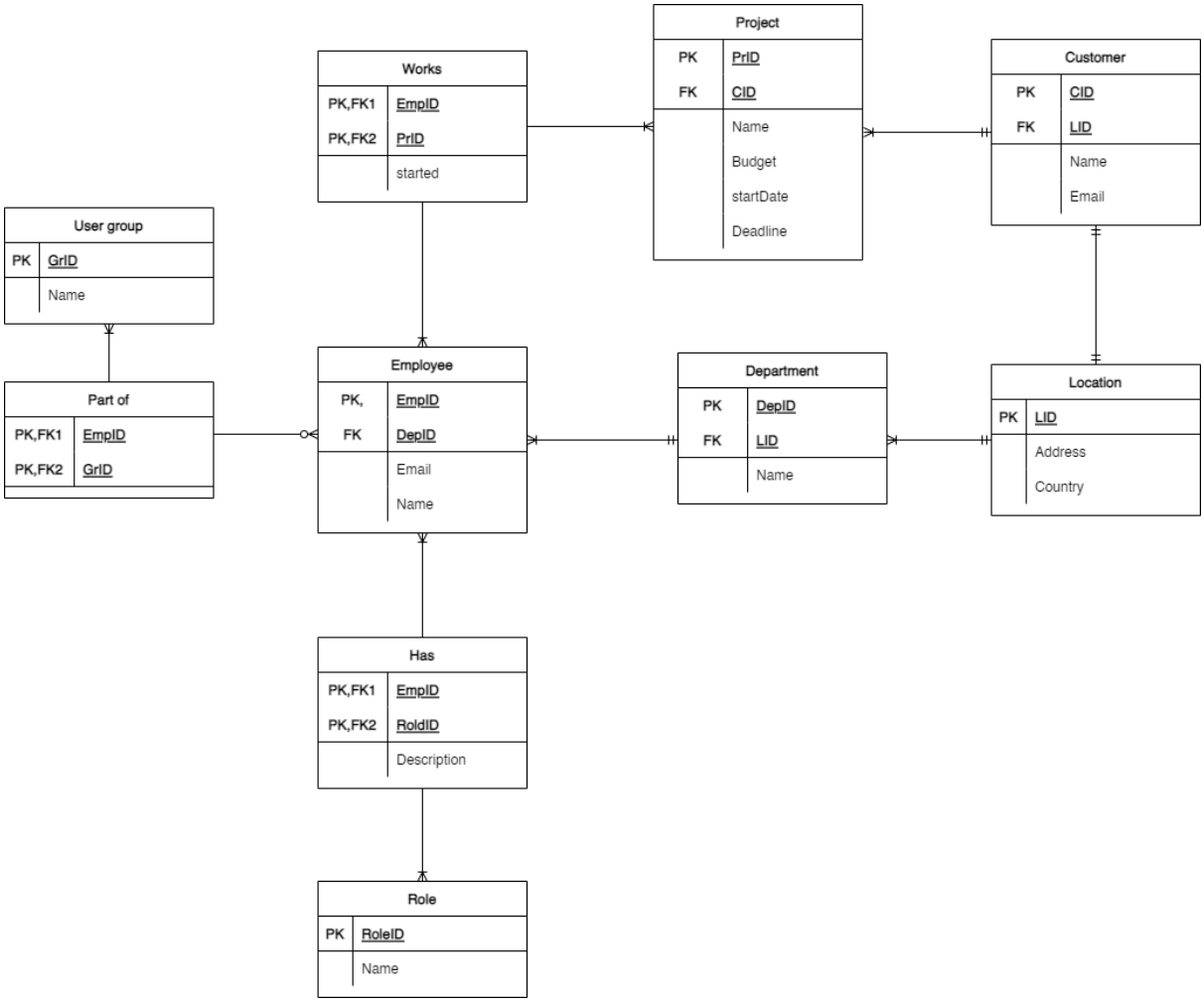
CT60A7650 Database Systems Management

FINAL PROJECT

1. Relational model based on the ER model



2. Integrity rules

a) Project:

ON DELETE: Delete all "Works" relationships connected to a "Project" when a "Project" is deleted.

ON UPDATE: Update the attribute "PrID" of relationship "Works" when "PrID" is updated.

```
alter table Works add constraint Works_Project_Del
foreign key EmpID references Project.EmpID
on delete cascade on update cascade;
```

b) Customer:

ON DELETE: Delete all "Project" connected to a "Customer" when a "Customer" is deleted.

ON UPDATE: Update the attribute "CID" of table "Project" when "CID" is updated.

```
alter table Project add constraint Project_Customer_Del
foreign key CID references Customer.CID
on delete cascade on update cascade;
```

c) User group:

ON DELETE: Delete all "Part of" relationships connected to "User group" when "User group" is deleted.

ON UPDATE: Update the attribute "GrID" of relationship "Part of" when "GrID" is updated.

```
alter table "Part of" add constraint PartOf_UserGroup_Del
foreign key GrID references "User group".GrID
on delete cascade on update cascade;
```

d) Employee:
ON DELETE: Delete all “Part of”, “Works”, and “Has” relationships connected to “Employee” when “Employee” is deleted.
ON UPDATE: Update the attribute “EmpID” of relationship “Part of”, “Works”, and “Has” when “EmpID” is updated.

```
alter table "Part of" add constraint PartOf_Employee
foreign key EmpID references Employee.EmpID
on delete cascade on update cascade;
alter table "User group" add constraint UserGroup_Employee
foreign key EmpID references Employee.EmpID
on delete cascade on update cascade;
alter table Works add constraint Works_Employee
foreign key EmpID references Employee.EmpID
on delete cascade on update cascade;
```

3. Partitioning

Partitioning would divide tables into smaller groups, which brings many benefits such as improved scalability and availability, easier maintenance, increased performance.

a) “Employee” table partition:
The company in question has 100 employees; therefore, we could partition the “Employee” table based on “Name” in alphabetic order. Accessing information of employees is easier and faster as we do not need to go through 100 employees every time.

```
alter table Employee partition by list (left(Name,1));
create table af partition of Employee for values in ('A', 'B', 'C', 'D', 'E',
'F');
create table gl partition of Employee for values in ('G', 'H', 'I', 'J', 'K',
'L');
create table mr partition of Employee for values in ('M', 'N', 'O', 'P', 'Q',
'R');
create table sz partition of Employee for values in ('S', 'T', 'U', 'V', 'W',
'X', 'Y', 'Z');
```

b) “Project” table partition:
The company in question has dozens of projects; therefore, we could partition the “Project” table based on “startDate”, based on a quarterly matter. This method presents the table in a logical order, from the project that starts in the beginning of the year, to the one that starts before Christmas.

```
alter table Project partition by range (startDate);
create table 12022 partition of Project for values from ('2022-01-01') to ('2022-
03-31');
create table 22022 partition of Project for values from ('2022-04-01') to ('2022-
06-30');
create table 32022 partition of Project for values from ('2022-07-01') to ('2022-
09-30');
create table 42022 partition of Project for values from ('2022-10-01') to ('2022-
12-31');
```

4. Access rights for users

For example, an employee with ID “Emp1” having role “Emp1” is working on project with ID “P001”. User group “Group1” belongs to role “Emp1”. The group “Group1” only has access to “P001”.

```
create role Project1;
create role Emp1 inherit;
grant Project1 to Emp1;
create role Group1 inherit;
grant Emp1 to Group1;
create policy all_view ON Project for select using (true);
create policy accrts ON Project for update to Project1
  using (PID = 'P001')
  with check (PID = 'P001');
grant select, update all ON Project TO public;
```

5. Management of values

a) Define default values:

- Define “startDate” of table “Project” as the current date, and the “Deadline” is 6 months from the current date.

```
alter table Project alter startDate set default current_timestamp();
alter table Project alter Deadline set default current_timestamp() + interval '6
months';
```

- Define “started” of relationship “Works” as the current date.

```
alter table Works alter started set default current_timestamp();
```

b) Define check constraints:

- Define “Deadline” of table “Project” should be should be larger than current date and larger than “startDate”, and “Budget” should be larger than 0.

```
alter table Project add constraint deadlineCheck
   check (Deadline > current_timestamp()
      AND Deadline > startDate);
```

- Define “Budget” of table “Project” should be larger than 0.

```
alter table Project add constraint budgetCheck
   check (Budget > 0);
```

c) Define how to manage NULL values:

All IDs of the database should not be NULL.

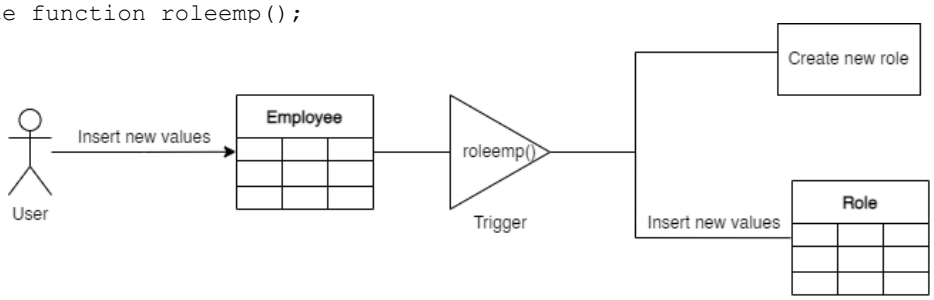
```
alter table "User group" alter column GrID set not null;
alter table Employee alter column EmpID set not null;
alter table Role alter column RoleID set not null;
alter table Department alter column DepID set not null;
alter table Location alter column LID set not null;
alter table Customer alter column CID set not null;
alter table Project alter column PrID set not null;
```

6. Triggers and trigger graph

The following triggers should be created so the access rights (see 4. Access rights for users) could be implemented faster:

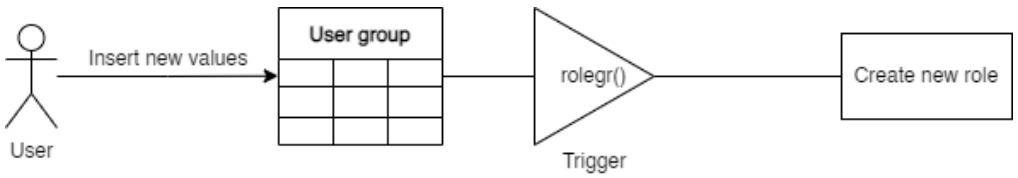
- Automatically create a role with the same name as “EmpID” when user inserts new employee; automatically insert into table “Role” this role “RoleID” and “Name”.

```
create function roleemp()
returns trigger
language plpgsql
as $$
begin
create role (new.EmpID);
insert into Role values (new.EmpID, new.Name);
$$;
create trigger roleemp_trigger before insert on employee for each row
execute function roleemp();
```



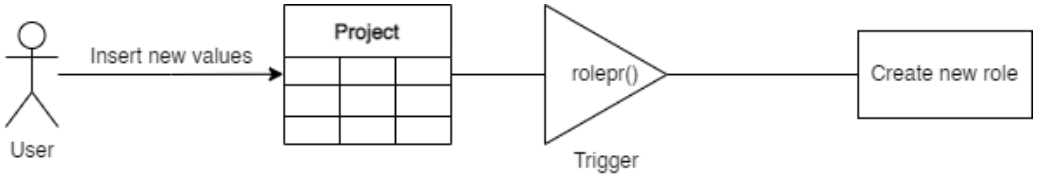
- Automatically create a role with the same name as “GrID” when user inserts a new user group.

```
create function rolegr()
returns trigger
language plpgsql
as $$
begin
 create role (new.GrID);
end;
$$;
create trigger rolegr_trigger before insert on “User group” for each row
execute function rolegr();
```



- Automatically create a role with the same name as “PrID” when user inserts a new project.

```
create function rolepr()
returns trigger
language plpgsql
as $$
begin
 create role (new.PrID);
end;
$$;
create trigger rolepr_trigger before insert on “User group” for each row
execute function rolepr();
```



7. Security issues and measures

The company in question is a medium-size enterprise. Therefore, the following security breaches are possible:

- Unauthorized disclosure of information: Employees disclose sensitive data, such as project or customer details. → Enforce strict confidentiality policies, implement reprimands for employees who violate these policies, and adopt clear authorization hierarchy.
- Cyberattack: The database's security system is breached by a malicious third party, and data is lost or damaged. → Maintain an up-to-date software and system, train employees to watch out for possible suspicious activities, and have multiple backups of the database.
- Lack of maintenance: Regular maintenance is not performed properly, leading to a database malfunction and a leak of sensitive information. → Have an efficient and clear maintenance schedule, use appropriate maintenance techniques, and give regular training to maintenance team.

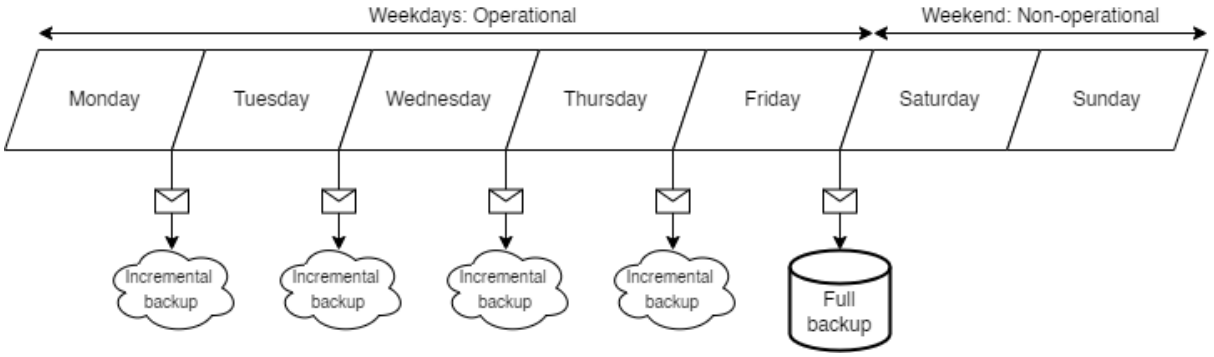
8. Backup and recovery, database disaster plan

a) Backup method and schedule:

The database should have both full- and incremental backup.

A full backup should be performed at the end of every week, on the last weekday (Friday). A full backup will allow a fast and comprehensive recovery of data, direct access to the most recent version, and a minimal time required to restore business operations. However, this type of backup also uses a large amount of storage space and takes a long time; therefore, it should occur only once a week.

An incremental backup should be done at the end of each working day. An incremental backup will allow recovery for minor errors such as incorrect data insertions or accidental delete of data. This type of backup ensures smooth database operations and leaves a "return path" if an unfortune accident happens.



b) Disasters and recovery methods:

- Natural disasters: Natural phenomenon such as earthquakes, storms, wildfires, etc., severely impacting database operations or destroying data centers.
- Hardware errors: Damaged hard drives or server bottleneck creating malfunctions and data loss.
- Software errors: Software bugs, corrupted data, or cyberattacks damaging data and bringing system offline.
- Geopolitical conflicts (extremely unlikely): Wars and conflicts causing unprecedented and irreversible consequences, with the possibility to permanently disrupt database operations.

→ Database recovery solutions:

- Follow backup schedule and maintain backup carefully;
- Have multiple copies of one backup, spread across different sites;
- Use the cloud as one of the storage solutions;
- Establish an emergency headquarter at one of the company's offices, with a server than could be turned online in a short amount of time;
- Train an emergency response team to quickly control the damage;
- Update software according to schedule;
- Upgrade hardware when they reach intended lifespan.

***Note:** As there is no actual database to test the solutions and possible methods, and the information provided about the company is inadequate, all details provided in this final project are based on assumptions.