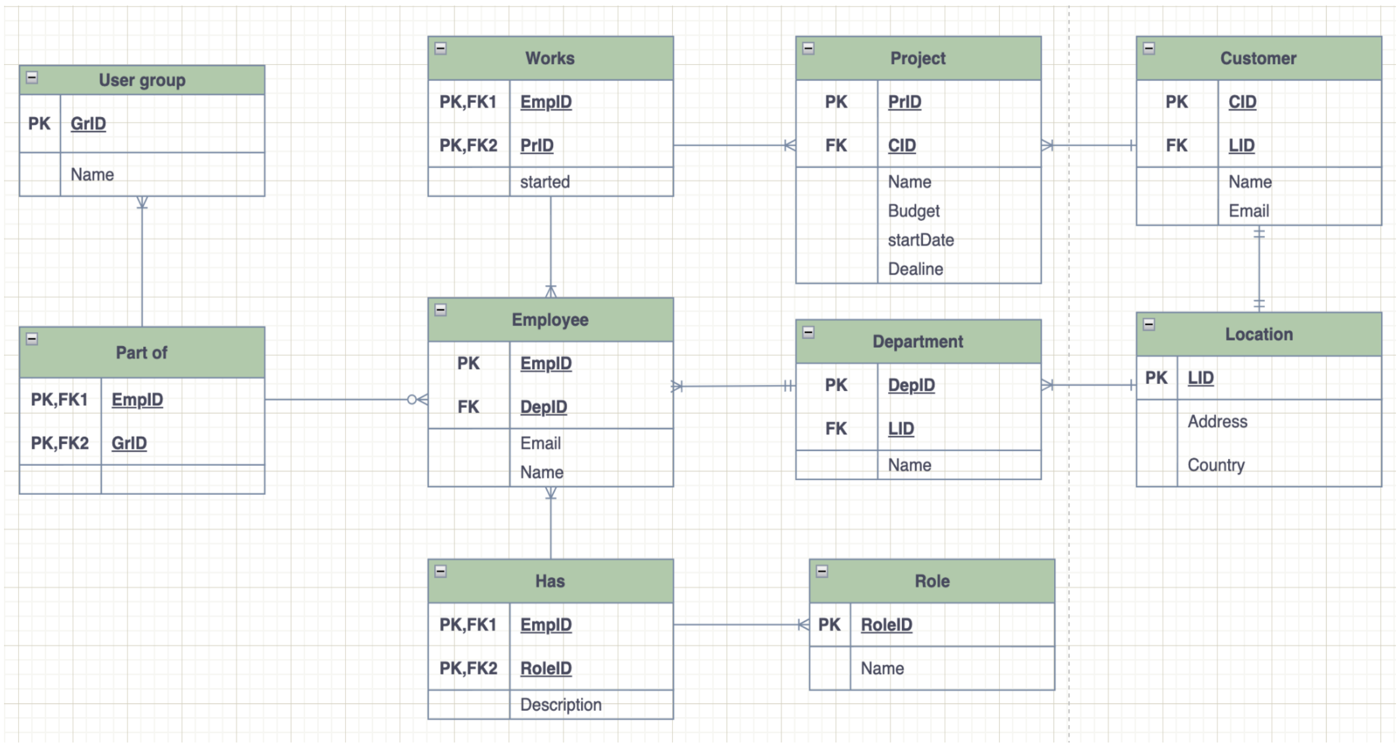
My Name: Ha Duong
My Student Number: 000938415

1. **Relational model based on the ER model**



2. **Decide on the DBMS application for BiDi**
3. **Define the availability window by assuming the cost for downtime**
    a. **Only define the window/ percentage with assumed downtime costs.**
    b. **Hint: you can refer to the SLM slide in Lecture 3 to help define availability**
4. **Partitioning**
   Partitioning would divide tables into smaller groups, which brings many advantages such as improved scalability and availability, easier maintenance, increased performance.
    a) "Employee" table partition:
       Given that the organization in question has 1000 workers, we might divide the "Employee" data according to "Name" in alphabetical order. Since we don't have to go through 1000 people each time, accessing employee information is simpler and quicker.

```
ALTER TABLE Employee PARTITION BY RANGE (LEFT(Name, 1));
CREATE TABLE Employee_Name_A_D PARTITION OF Employee FOR
VALUES IN ('A', 'B', 'C', 'D');
```

```
CREATE TABLE Employee_Name_E_H PARTITION OF Employee FOR
VALUES IN ('E', 'F', 'G', 'H');
CREATE TABLE Employee_Name_I_L PARTITION OF Employee FOR
VALUES IN ('I', 'J', 'K', 'L');
CREATE TABLE Employee_Name_M_P PARTITION OF Employee FOR
VALUES IN ('M', 'N', 'O', 'P');
CREATE TABLE Employee_Name_Q_T PARTITION OF Employee FOR
VALUES IN ('Q', 'R', 'S', 'T');
CREATE TABLE Employee_Name_U_Z PARTITION OF Employee FOR
VALUES IN ('U', 'V', 'W', 'X', 'Y', 'Z');
```

b) "Project" table partition:
Given the organization in question has dozens of projects, we might divide the "Project" data according to "Budget" in three different groups(low, middle, and high budget. Therefore, this method presents the table in a logical order making it easier for users to access budget information of projects.

```
ALTER TABLE Project PARTITION BY RANGE (Budget);
CREATE TABLE low_budget_project PARTITION OF Project FOR
VALUES FROM (0.00) TO (5000.00);
CREATE TABLE middle_budget_project PARTITION OF Project
FOR VALUES FROM (5000.00) TO (25000.00);
CREATE TABLE high_budget_project PARTITION OF Project FOR
VALUES FROM (25000.00) TO (MAXVALUE);
```

5. 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 PrID REFERENCES Project.PrID
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_Del
FOREIGN KEY EmpID REFERENCES Employee.EmpID
ON DELETE CASCADE ON UPDATE CASCADE;
ALTER TABLE “Has” ADD CONSTRAINT PartOf_Employee_Del
FOREIGN KEY EmpID REFERENCES Employee.EmpID
ON DELETE CASCADE ON UPDATE CASCADE;
ALTER TABLE “Works” ADD CONSTRAINT PartOf_Employee_Del
FOREIGN KEY EmpID REFERENCES Employee.EmpID
ON DELETE CASCADE ON UPDATE CASCADE;
```

6. Management of Values

a) Define default values:

- Define “startDate” of table “Project” as the current date, and define “started” of relationship “Works” as the current date.

```
ALTER TABLE Project ALTER startDate SET DEFAULT
current_timestamp();
ALTER TABLE Works ALTER started SET DEFAULT current_timestamp();
```

b) Define check constraints:

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

```
ALTER TABLE Project ADD CONSTRAINT validateDeadline
  CHECK (Deadline > current_timestamp() AND Deadline > startDate);
```

c) Define how to manage NULL values:
The primary key of User group, Employee, Department, Location, Customer, Project, Role 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 “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;
ALTER TABLE "Role" ALTER column RoleID SET NOT NULL;
```

7. Triggers and a Trigger Graph

8. Access Rights for Users

Let’s say we create an employee with ID “Emp1” having the role “role_1” is working on a project with ID “P001”. User group “Group1” belongs to the role “role_1”. The group “Group1” only has access to “P001”.

```
CREATE ROLE Project_1;
CREATE ROLE role_1 inherit;
GRANT Project_1 to role_1;
CREATE ROLE Group1 inherit;
GRANT role_1 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;
```

9. Security Issues and Measures

10. Design a checklist for managing changes

11. Backup and Recovery, Disaster Plan