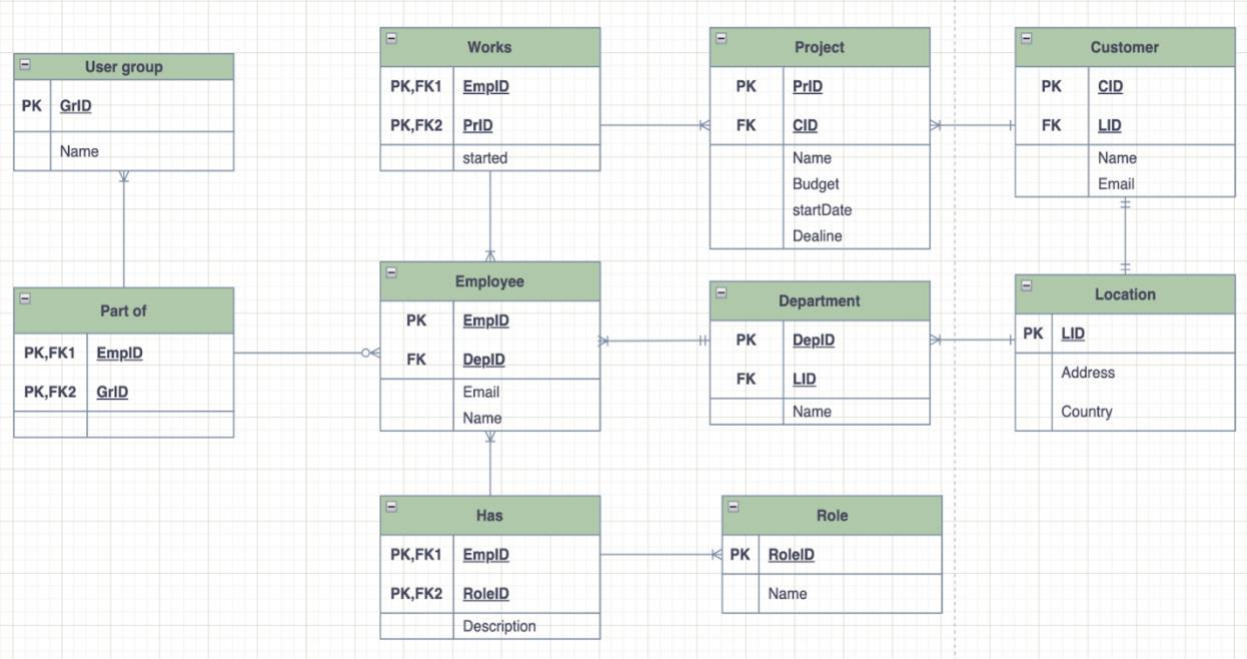
CT60A7650 Database Systems Management FINAL PROJECT

My Name: Ha Duong

My Student ID: 000938415

1. Design a Relational Model or database schema based on the ER model



2. Decide on the DBMS application for BiDi

As a DBA, I have considered many potential DBMS vendors based on different decision factors, especially price. The best option among these DBMS vendors is the Microsoft SQL server. There are the following reasons why I believe this would be the suitable and best choice for our company:

Operating System Support: The Microsoft SQL server supports various systems such as Windows Server, Linux, and Docker. Additionally, virtualization technologies such as Hyper-v and VMware or cloud platforms such as AWS also are compatible with this DBMS. This is a huge advantage for the Microsoft SQL Server, which simplifies the process of integration of the database system management.

Scalability and Performance: Microsoft SQL Server is created to be scalable, handle large workloads, and have high performance. It would come in handy when in the future when the need for saving the information of the customer and employee is quite large.

Cost of Ownership: The cost of ownership for Microsoft SQL Server can vary depending on several elements such as the licensing fees, hardware and infrastructure cost, administration and support cost, and training and development costs,… But in general, it's relatively affordable compared to Oracle Database Enterprise Edition and IBM Db2 Enterprise Editio, making it a suitable option for the initial budget of €100,000.

3. Define the availability window by assuming the cost for downtime

If the system goes down during the availability window, users may not be able to:
- access Project information such as name, budget, and deadline, which may result in missed project deadlines
- access Employee information leading to a delay in payroll processing

- ask or contact their customer about the requirement in medical support systems, which makes it harder to implement their product
=> However, the need for 24/7 availability is not necessary because the company is still new, and the business of the company is not significantly affected by downtime. Therefore 98% is enough for the availability of the database.

4. Partitioning

Partitioning would divide tables into smaller groups, which brings many advantages such as improved scalability and availability, easier maintenance, and 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” into three different groups(low, middle, and high budget. Therefore, this method presents the table logically, making it easier for users to access project budget information.

```
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 table “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” relationships 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 the 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

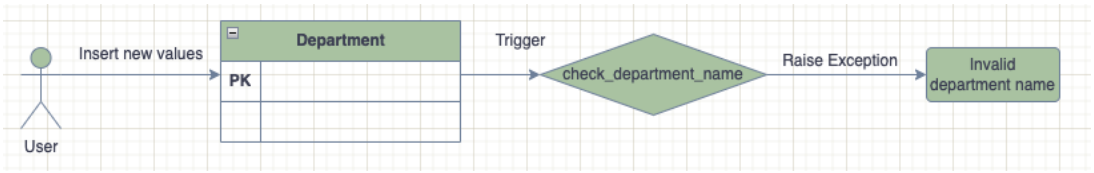
```
ALTER TABLE “User group” ALTER column GrID SET NOT NULL;
ALTER TABLE “Employee” ALTER column EmpID SET NOT NULL;
ALTER TABLE “Department” ALTER column DepID SET NOT NULL;
ALTER TABLE “Location” ALTER column LID SET NOT NULL;
ALTER TABLE “Customer” ALTER column CID SET NOT NULL;
ALTER TABLE “Project” ALTER column PrID SET NOT NULL;
ALTER TABLE "Role" ALTER column RoleID SET NOT NULL;
```

7. Triggers and a Trigger Graph

- Trigger to ensure that before users insert information into table “Department”, the “name” attribute is one of "HR", "Software", "Data", "ICT", or "Customer support" (Because BiDi company just has only the departments above)

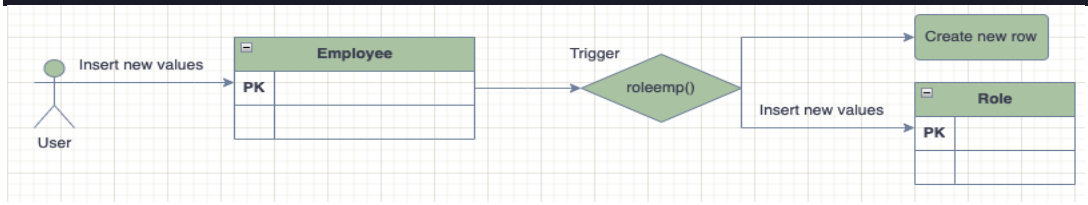
```
CREATE OR REPLACE FUNCTION check_department_name()
RETURNS TRIGGER AS $$
BEGIN
```

```
IF NEW.name NOT IN ('HR', 'Software', 'Data', 'ICT', 'Customer support') THEN
RAISE EXCEPTION 'Invalid department name.';
END IF;
RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_department_name
BEFORE INSERT OR UPDATE ON Department
FOR EACH ROW EXECUTE FUNCTION check_department_name();
```



- Trigger to ensure that when inserting new information about an employee into the “Employee” table, the database application automatically creates a role and inserts it into the “Role” table.

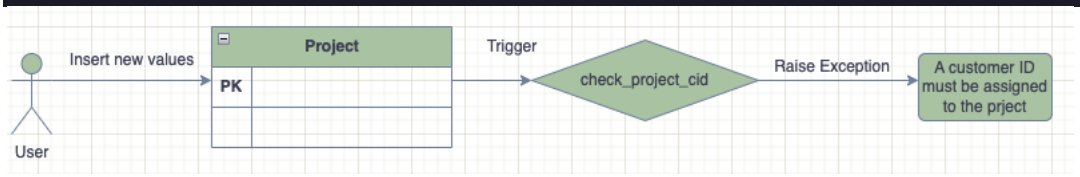
```
CREATE OR REPLACE FUNCTION roleemp()
RETURNS TRIGGER AS $$
BEGIN
CREATE ROLE (new.EmpID);
INSERT INTO Role value (new.EmpID, new.Name);
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER roleemp_trigger BEFORE INSERT ON employee FOR EACH ROW
EXECUTE FUNCTION roleemp();
```



- Trigger to ensure that a customer is assigned to the project.

```
CREATE OR REPLACE FUNCTION check_project_cid()
RETURNS TRIGGER AS $$
```

```
BEGIN
IF NEW.CID IS NULL THEN
RAISE EXCEPTION 'A customer must be assigned to the project.';
END IF;
RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_project_cid
BEFORE INSERT OR UPDATE ON Project
FOR EACH ROW EXECUTE FUNCTION check_project_cid();
```



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 SLECT 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

The company in question is a medium-sized 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 a clear authorization hierarchy.
- - Cyberattack: The database's security system is breached by a malicious third party, and data is lost or damaged. → Maintain 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 the maintenance team.

10. Design a checklist for managing changes

There are the following steps to manage changes:

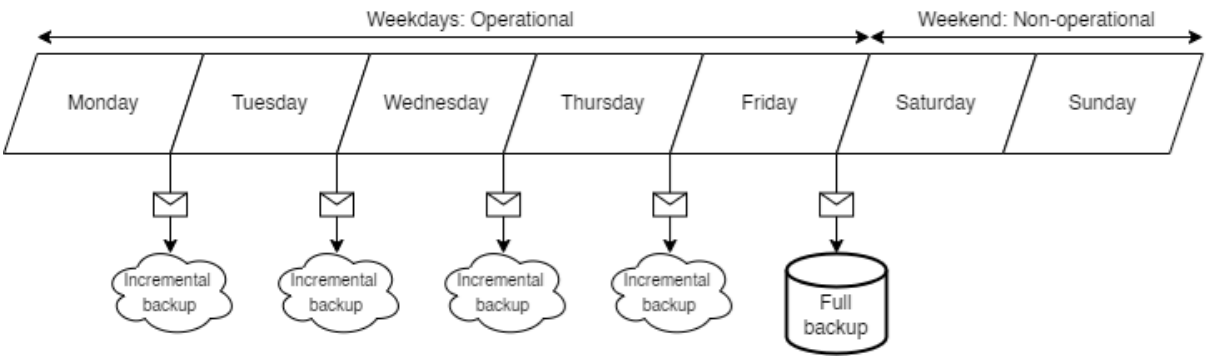
1. Define the change request: Clearly define the change request and its scope, and document the reasons for the change.
2. Analyze the impact of the change: Analyze the impact of the change on the database and its related applications, and identify any potential risks and dependencies.
3. Develop a test plan: Develop a test plan for the change, including both functional and non-functional testing, and ensure that it covers all aspects of the change.
4. Get approval: Obtain approval from all stakeholders, including management, database administrators, and application owners.
5. Implement the change: Implement the change according to the test plan, and document any deviations from the plan.
6. Perform testing: Perform testing on the change, including functional and non-functional testing, and verify that it meets the required specifications.
7. Rollback plan: Develop a rollback plan in case the change does not work as intended, and ensure that all stakeholders are aware of the plan.
8. Deploy the change: Deploy the change to production, and ensure that it is properly monitored and supported.

11. Backup and Recovery, Disaster Plan

a) Backup method and schedule:

The database should have both full- and incremental backups.
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 deletion of

data. This type of backup ensures smooth database operations and leaves a “return path” if an unfortunate accident happens.



b) Disasters and recovery methods:

- - Natural disasters: Natural phenomena such as earthquakes, storms, wildfires, etc., severely impacting database operations or destroying data centers.
- - Hardware errors: Damaged hard drives or server bottlenecks creating malfunctions and data loss.
- - Software errors: Software bugs, corrupted data, or cyberattacks damaging data and bringing the 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 the 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

  then 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.