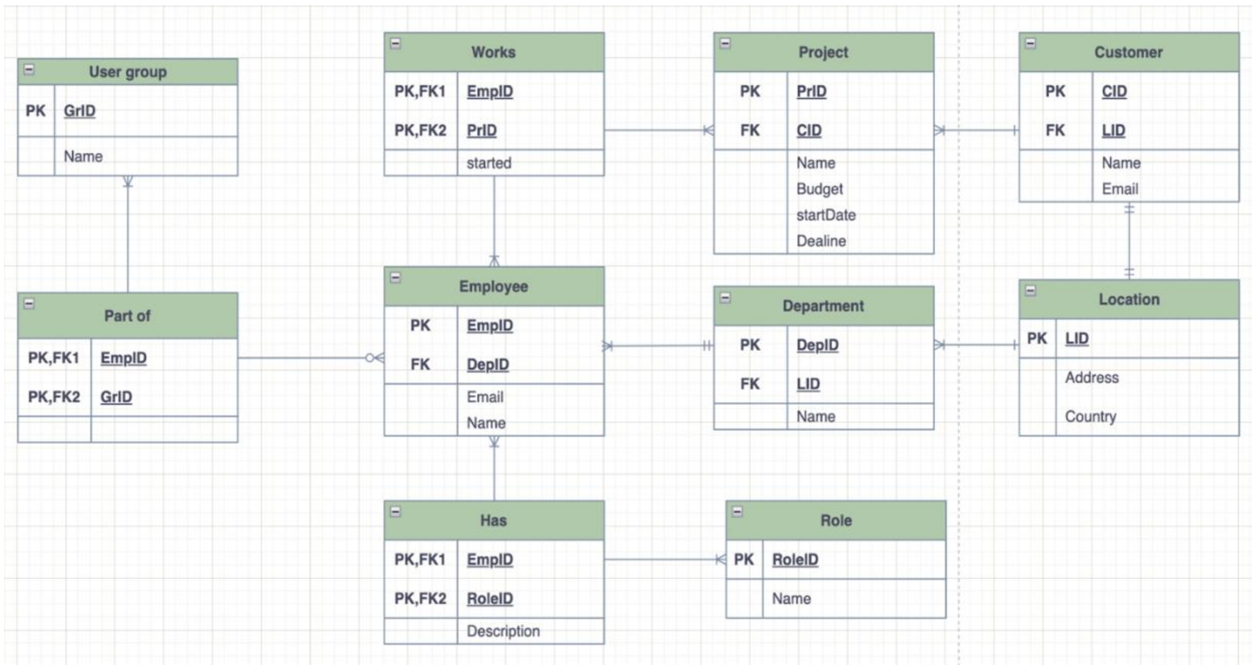
CT60A7650 Database Systems Management FINAL PROJECT

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 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.

Technicians: Whenever we run into any database problems, we can receive prompt assistance from the Microsoft community which has a sizable pool of knowledgeable professionals. Also, there are forums or groups that have a sizable number of active users who can give developers weather of tools to use.

Release Schedule: Using the Microsoft SQL Server can make sure that our server always stays up to date with the latest releases and has clear guidance and tools to help us implement upgrades to new SQL Server releases. The upgrade normally happens for a set period of time, including mainstream support and extended support.

Reference Customers: The Microsoft SQL Server has a large customer base ranging from small businesses to well-known corporations, such as Adobe, Volvo, Siemens, and Dell technologies. These companies have proven the credibility of this SQL server so we do not need to worry about this.

3. Define the availability window by assuming the cost of 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 a medium-sized enterprise, 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);
```

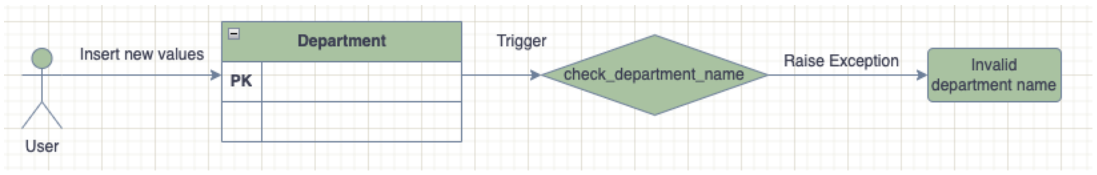
c) Define how to manage NULL values:
The primary key of the User group, Employee, Department, Location, Customer, Project, Role should not be NULL

```
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)

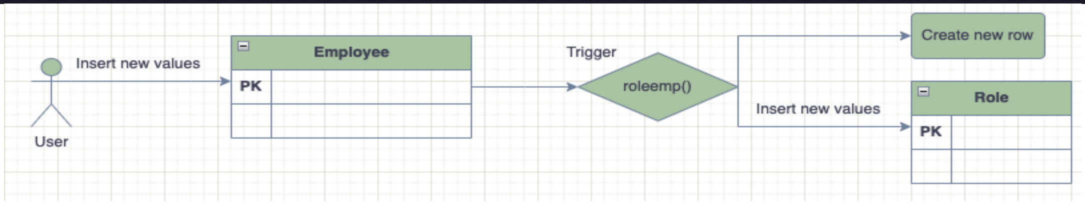
```
CREATE OR REPLACE FUNCTION check_department_name()
RETURNS TRIGGER AS $$
BEGIN
IF NEW.name NOT IN ('HR', 'Software', 'Data', 'ICT', 'Customer support') THEN
RAISE EXCEPTION 'Invalid department name.';
END IF;
RETURN NEW;
END;
$$ LANGUAGE plpgsql;

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.

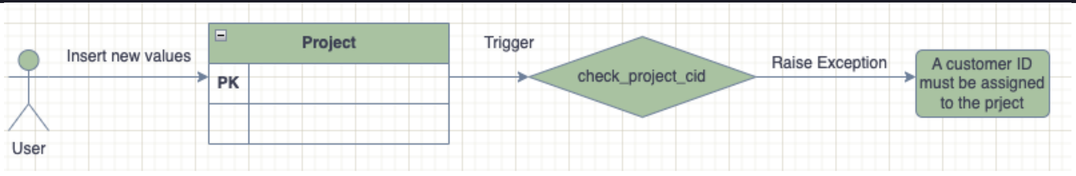
```
CREATE OR REPLACE FUNCTION roleemp()
RETURNS TRIGGER AS $$
BEGIN
CREATE ROLE (new.EmpID);
```

```
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

a) Three different security issues and a countermeasure for each:

- Lack of maintenance: Employees do not maintain the system regularly and correctly, causing a database error and the hacker can take advantage of that to steal private data. => Solution: Make a productive and transparent maintenance plan, using suitable maintenance practices, and giving ongoing training for employees.

- Cyberattack: The system is vulnerable to being attacked by meticulous code, which affects severely on company’s business when creating downtime and loss of private information => educating staff to be on the lookout for any unusual activity and maintaining numerous database backups are all recommended.

- Unauthorized disclosure of information: Employees reveal sensitive information, such as projects or customer details => Solution: Making a clear authorization hierarchy, implementation of punishment for employees who violate these regulations, and enforcement of stringent confidentiality requirements.

b) It is essential for both staff and managers to follow some regulatory standards. Especially, when it comes to medical support fields, the industries and types of organizations are subject to regulatory standards that require compliance with specific security measures to protect private information, for example, the personal or financial data of the customer.

10. Design a checklist for managing changes

There are the following steps to manage changes:

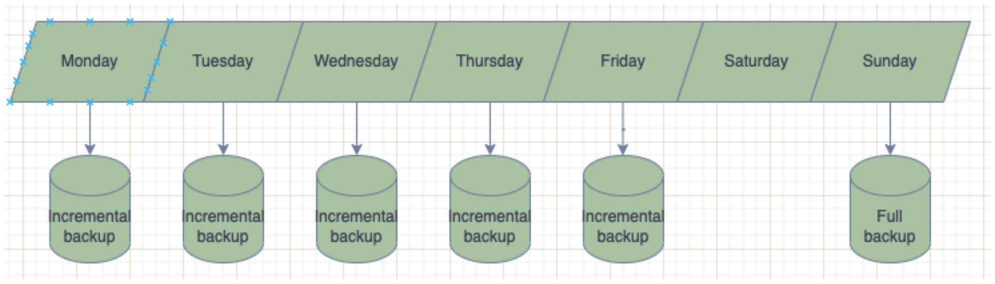
1. Define the change request: We should give the reason why the application is needed to be changed and define the change request and its impact.
2. Analyze the impact of the change: Consider its impact on the database and any associated applications and look at any dependencies and potential dangers.

3. Develop a test plan: Scheduling a test plan covering all aspects of the change, including both functional and non-functional testing.
4. Get approval: Make sure that we get approval from all stakeholders
5. Implement the change: Based on the test plan, we implement all the changes
6. Perform testing: Test the modification, both functionally and non-functionally, to ensure that it complies with the necessary requirements.
7. Deploy the change and rollback plan: We should deploy the change to production, and ensure that is properly monitored and supported, and develop a rollback plan in case some changes are wrong

11. Backup and Recovery, Disaster Plan

a) Backup method and schedule:

The database should have both full- and incremental backups.
Every week, on the last day of the week(Sunday), a complete backup should be run, therefore; it ensures that the work of that week can not be lost if something wrong happens. A complete backup will enable rapid and thorough data recovery, immediate access to the most recent version, and a short turnaround time for business activities restoration. This kind of backup should only be performed once a week because it takes a long time and consumes a lot of storage space.
Every working day should finish with an incremental backup. Minor mistakes like improper data insertions or unintentional data deletions can be recovered with an incremental backup. This kind of backup guarantees efficient database operations and provides a "return path" in the event of an unpleasant incident.



b) Disasters and recovery methods:

- Natural disasters: Natural disasters like earthquakes, hurricanes, wildfires, etc., can have a significant influence on database operations or even completely destroy data centers.
- Hardware errors: Damaged hard drives or server bottlenecks creating malfunctions and data loss.
- Software errors: Damaged data, software flaws, or hackers destroying data and taking the system down.
- Geopolitical conflicts (extremely unlikely): Wars and conflicts with the potential to permanently impair database operations might have unprecedented and irrevocable effects.

=> Database recovery solutions:

- Follow the backup schedule and maintain backup carefully;
- Have multiple backup methods, spread across different sites;
- Use the cloud as one of the storage solutions;
- Schedule a plan to upgrade the software
- Upgrade the hardware when it reaches the end of the lifespan
- Prepare an emergency response team to contain the damage immediately;
- Create an emergency headquarters in one of the company's offices, with a server there that could be brought up quickly;