Company Database Simulation Project

Database Design

Figure 1 shows the ER diagram that encapsulates the schema of the Company database. The diagram presents 5 strong entities: Department, Project, Employee, Location, and Dependent.

To construct a logical database, the following assumptions have been made:

1. Project leader and employee supervisor is not the same concept: every employee is supervised by a supervisor, and every employee who works on a project is under the leadership of the project leader, who may not necessarily be the same person as the employee's supervisor
2. Each department has only one manager, and one employee can only be manager of one department
3. Not only employees will have many dependents but also that dependents may have more than one employees as their relatives .For example , parents of a child may both work in this company. On the other hand , the relation between Employee and Dependent is many-to-many.
4. Employee salary is hourly
5. This database is in a world where every person's name is unique

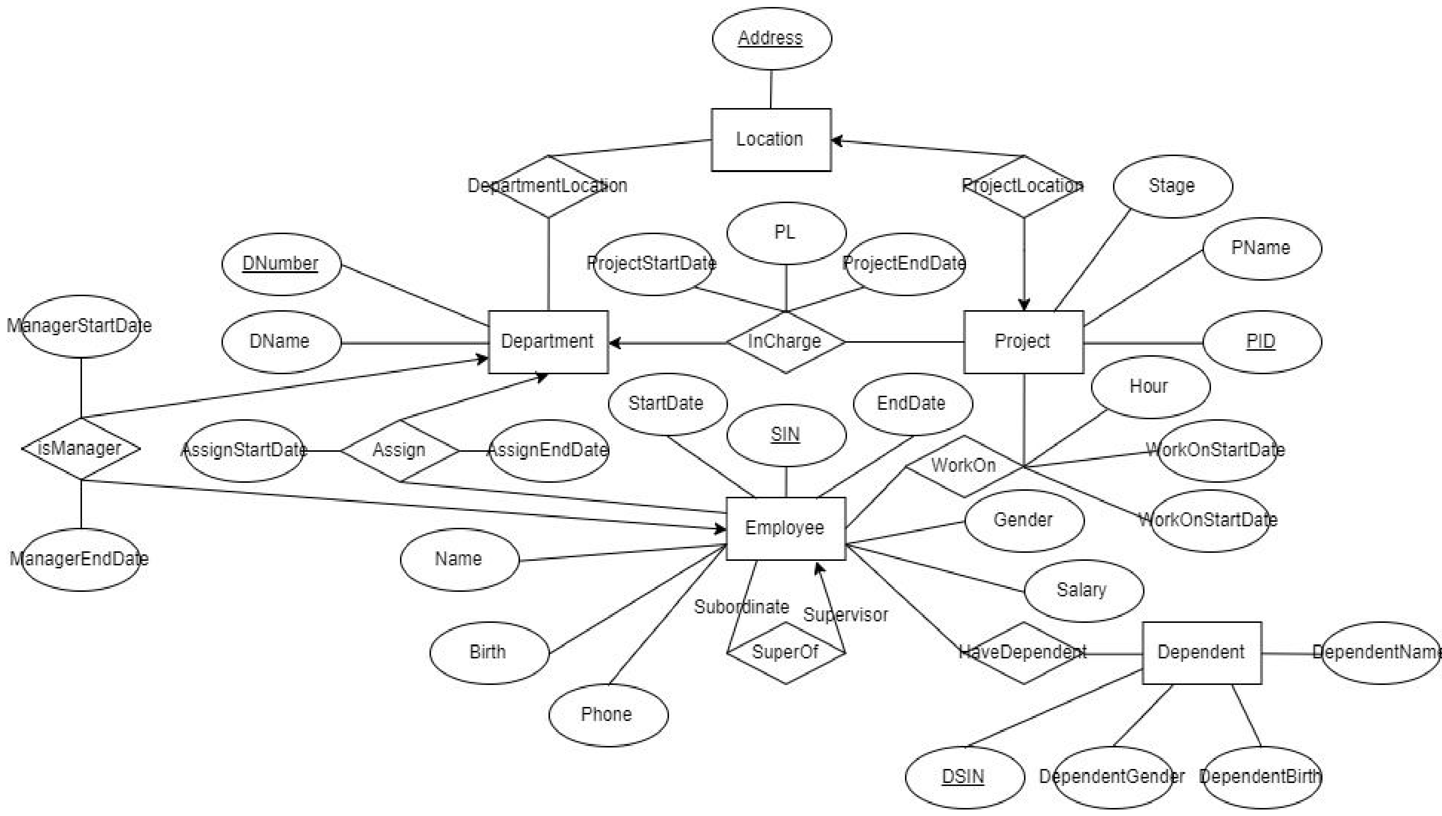

Figure 1

Database Implementation

Changes of implementation due to technical issues

Many constraints have been implemented into the database schema to maintain referential integrity, mainly in the form of FOREIGN KEY and CHECK. However, since the CHECK constraint is not implemented in the provided SQL server, the INSERT/UPDATE checking constraints will then be implemented using TRIGGER instead.
In the version of SQL provided by AITS server, the syntax of TRIGGER implementation blocks the usage of OR in TRIGGER declaration, hence the triggers on before INSERT or UPDATE are implemented separately. Also, the implementation of triggers on INSERT and UPDATE actions are placed in different action (BEFORE INSERT and AFTER UPDATE) since the SQL server does not allow multiple definitions of trigger for the same action.

Create Table Queries

Tables of entities and relations are created as the following, the key attributes are underlined:

1. Department(<u>DNumber</u>, DName)
FDs:

DName->DNumber
DNumber->DName

SQL table creation query:

```
Create table Department (
DNumber INT PRIMARY KEY AUTO_INCREMENT ,
DName VARCHAR(30)  NOT NULL UNIQUE
);
```

Analysis:

- Entity Department has attributes DNumber and DName, DNumber being the unique ID of the department represented by an integer, and DName is the unique name of the department represented by a char array.
- Both DName and DNumber are candidate keys. In this case, DNumber is chosen as primary key. DName is however given the properties of unique and not null.
- For every FD in this schema, the left hand side is a super key as either of DName and DNumber is able to uniquely identify a tuple, thus this schema is in 3NF.

2. Employee(<u>SIN</u>,Name, Birth, HomeAddress, Phone, Gender, Salary, StartDate, EndDate)

FDs:

```
SIN->Name,Birth, HomeAddress, Phone,Gender,Salary, StartDate, EndDate
Name->SIN,Birth, HomeAddress, Phone,Gender,Salary, StartDate, EndDate
```

SQL table creation query:

```
CREATE TABLE Employee(
SIN INT PRIMARY KEY CHECK(SIN>=100000000 AND SIN <=999999999),
Name VARCHAR(100) NOT NULL,
Birth DATE DEFAULT '0000-00-00',
HomeAddress VARCHAR(100) DEFAULT 'UNKNOWN',
Phone INT CHECK(Phone>99 AND Phone<=9999999999),
Gender CHAR(1) DEFAULT '?' CHECK(Gender='F' OR Gender='M' OR Gender='?'),
Salary DOUBLE(5,2) UNSIGNED DEFAULT 0.00,
StartDate DATE NOT NULL,
EndDate DATE CHECK (EndDate > StartDate),
);
```

CHECK implemented using TRIGGER:

-
  ```
  delimiter //
  CREATE TRIGGER sin_check_insert
  BEFORE INSERT ON Employee
  FOR EACH ROW
  BEGIN
  IF (NEW.SIN<100000000 OR NEW.SIN>999999999) THEN
  SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'SIN must be exactly 9 digits';
  ELSEIF (NEW.Phone <= 99 OR NEW.Phone > 9999999999) THEN
  SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Phone must be valid ';
  ELSEIF (NEW.Gender != 'F' AND NEW.Gender != 'M' AND NEW.Gender != '?')THEN
  SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Gender must be valid ';
  ELSEIF (NEW.EndDate < NEW.StartDate) THEN
  SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'EndDate must be later than
  StartDate ';
  END IF;
  END;//
  delimiter ;
  ```

-
  ```
  delimiter //
  CREATE TRIGGER sin_check_update
  AFTER UPDATE ON Employee
  FOR EACH ROW
  BEGIN
  IF (NEW.SIN<100000000 OR NEW.SIN>999999999) THEN
  UPDATE Employee SET SIN=OLD.SIN, Name=OLD.Name, Birth=OLD.Birth,
  HomeAddress=OLD.HomeAddress,
  ```

```
Phone=OLD.Phone, Gender=OLD.Gender, Salary=OLD.Salary, StartDate=OLD.StartDate,
EndDate=OLD.EndDate
WHERE SIN=NEW.SIN;
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'SIN must be exactly 9 digits';
ELSEIF (NEW.Phone <= 99 OR NEW.Phone > 9999999999) THEN
UPDATE Employee SET SIN=OLD.SIN, Name=OLD.Name, Birth=OLD.Birth,
HomeAddress=OLD.HomeAddress,
Phone=OLD.Phone, Gender=OLD.Gender, Salary=OLD.Salary, StartDate=OLD.StartDate,
EndDate=OLD.EndDate
WHERE SIN=NEW.SIN;
ELSEIF (NEW.Gender != 'F' AND NEW.Gender != 'M' AND NEW.Gender != '?') THEN
UPDATE Employee SET SIN=OLD.SIN, Name=OLD.Name, Birth=OLD.Birth,
HomeAddress=OLD.HomeAddress,
Phone=OLD.Phone, Gender=OLD.Gender, Salary=OLD.Salary, StartDate=OLD.StartDate,
EndDate=OLD.EndDate
WHERE SIN=NEW.SIN;
ELSEIF (NEW.EndDate < NEW.StartDate) THEN
UPDATE Employee SET SIN=OLD.SIN, Name=OLD.Name, Birth=OLD.Birth,
HomeAddress=OLD.HomeAddress,
Phone=OLD.Phone, Gender=OLD.Gender, Salary=OLD.Salary, StartDate=OLD.StartDate,
EndDate=OLD.EndDate
WHERE SIN=NEW.SIN;
END IF;
END;//
delimiter ;
```

Analysis:

- The entity Employee has attributes SIN, Name, Birth, HomeAddress, Phone, Gender, Salary, StartDate, EndDate.
- Since both SIN and Name are unique, both are candidate keys for this schema. In this case, SIN is chosen as primary key. Name is set to unique not null.
- We set the constraint on SIN, Phone, Gender and EndDate are implemented by triggers:
  - SIN: unique not null integer of exactly 9 digits
  - Phone: integer between 3 to 9 digits
  - Gender: single character with value from ‘M’, ‘F’, ‘?’
  - EndDate: date where employee leaves company, cannot precede employee’s StartDate
- Default values of Birth, HomeAddress, Gender, and Salary have been set since these are not required
- For every FD in this schema, the left hand side is a super key, either SIN or Name is able to uniquely identify a row. Thus this schema is in 3NF.

3. Project(<u>PID</u>, PName, Stage)

FDs:

PName->PID, Stage
PID->PName, Stage

SQL table creation query:

```
	CREATE TABLE Project(
	PID INT PRIMARY KEY AUTO_INCREMENT,
	PName VARCHAR(30) NOT NULL UNIQUE,
	Stage VARCHAR(20) default 'Preliminary' CHECK(Stage = 'Preliminary' OR Stage =
'Intermediate' OR Stage = 'Advanced' OR Stage = 'Complete'
	);
```

CHECK implemented using TRIGGER:

-
  ```
  delimiter //
  CREATE TRIGGER Stage_check_insert
  BEFORE INSERT ON  Project
  FOR EACH ROW
  IF (NEW.Stage != 'Preliminary' AND NEW.Stage != 'Intermediate' AND NEW.Stage !=
  'Advanced' AND NEW.Stage != 'Complete') THEN
  SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Stage must be in four states.';
  END IF;
  END;//
  delimiter ;
  ```

-
  ```
  delimiter //
  CREATE TRIGGER Stage_check_update
  AFTER UPDATE ON Project
  FOR EACH ROW
  BEGIN
  IF (NEW.Stage != 'Preliminary' AND NEW.Stage != 'Intermediate' AND NEW.Stage !=
  'Advanced' AND NEW.Stage != 'Complete') THEN
  UPDATE Project SET PID=OLD.PID, PName=OLD.PName, Stage=OLD.Stage
  WHERE Project.PID=NEW.PID;
  SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Stage must be in four states.';
  END IF;
  END;//
  delimiter ;
  ```

Analysis:

- The entity Project has attributes PID, PName, and Stage.
- Both PID and PName are candidate keys of this table, PID is chosen as primary key. PName is set to unique and not null.
- Constraint has been implemented on the attribute Stage using triggers (data input into Stage can only be chosen from ‘Preliminary’, ‘Intermediate’, ‘Advanced’, ‘Complete’).

- For every FD in this schema, the left hand side is a super key, that is to say either of PName and PID can uniquely identify a row. Thus this schema is in 3NF.

4. Location(<u>Address</u>)

SQL table creation query:

```
    CREATE TABLE Location (
    Address VARCHAR(100) PRIMARY KEY
    );
```

Analysis:

- The entity Location has only one attribute, Address. Since it is the sole attribute, it is the primary key.
- Since there is only one attribute in this table, no FD is present. 3NF/BCNF analysis does not apply.

5. DepartmentLocation(<u>Address, DNumber</u>)

SQL table creation query:

```
    CREATE TABLE DepartmentLocation(
    Address VARCHAR(100),
    FOREIGN KEY(Address) REFERENCES Location(Address) ON DELETE CASCADE ON
UPDATE CASCADE,
    DNumber INT,
    FOREIGN KEY(DNumber) REFERENCES Department(DNumber) ON DELETE
CASCADE ON UPDATE CASCADE,
    PRIMARY KEY(Address, DNumber)
    );
```

Analysis:

- This table represents the many-to-many relation between Department and Location. Since the relation is many-to-many, Address and DNumber are used as keyset.
- Address is a foreign key references from entity Location, DNumber is the foreign key from Department.
- The policy on delete and update for both foreign keys is to cascade, any changes of parent's key will reflect accordingly on the child's key.
- Since the only attributes of the table, Address and DNumber, compose the keyset, the FDs in this schema are trivial. The schema is on 3NF.

6. ProjectLocation(Address, <u>PID</u>)

FDs:

PID->Address

SQL table creation query:

```
        CREATE TABLE ProjectLocation(
        Address VARCHAR(100),
        PID INT PRIMARY KEY,
        FOREIGN KEY(Address) REFERENCES Location(Address) ON DELETE CASCADE
ON UPDATE CASCADE,
        FOREIGN KEY(PID) REFERENCES Project(PID) ON DELETE CASCADE ON UPDATE
CASCADE
        );
```

Analysis:

- This table represents the many-to-one relation between Project and Location. Since the relation is many-to-one (one Project can only be related to one Location, but a Location can be related to many Projects), only PID can uniquely identify a tuple, hence it is chosen as the primary key.
- PID and Address are attributes taken from Project and Location respectively, hence their values are bounded by foreign key constraints.
- The policy on delete and update for both foreign keys is to cascade, any changes of parent's key will reflect accordingly on the child's key.
- This schema is in 3NF since the left hand side of the FD is the schema's superkey.

7.Dependent(<u>DSIN</u>,DependentName, DependentGender, DependentBirth)

FDs:

DSIN->DependentName, DependentGender, DependentBirth
DependentName->DSIN, DependentGender, DependentBirth

SQL table creation query:

```
       CREATE TABLE Dependent(
       DSIN INT UNIQUE CHECK(SIN>=100000000 AND SIN <=999999999),
       DependentName VARCHAR(100) NOT NULL,
       DependentGender CHAR(1) DEFAULT '?' CHECK(Gender='F' OR Gender='M' OR
Gender='?'),
       DependentBirth DATE DEFAULT '0000-00-00',
       PRIMARY KEY(DSIN)
       );
```

CHECK implemented using TRIGGER:

-
  ```
  delimiter //
  CREATE TRIGGER dsingender_check_insert
  BEFORE INSERT ON Dependent
  FOR EACH ROW
    BEGIN
  ```

```
IF (NEW.DSIN<100000000 OR NEW.DSIN>999999999) THEN
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Dependent SIN must be exactly 9
digits';
ELSEIF (NEW.DependentGender != 'F' AND NEW.DependentGender != 'M' AND
NEW.DependentGender != '?')THEN
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'DependentGenderGender must be
valid ';
END IF;
END;//
delimiter ;
```

-
  ```
  delimiter //
  CREATE TRIGGER dsingender_check_update
  AFTER UPDATE ON Dependent
  FOR EACH ROW
  BEGIN
  IF (NEW.DSIN<100000000 OR NEW.DSIN>999999999) THEN
  UPDATE Dependent SET DSIN=OLD.DSIN, DependentName=OLD.DependentName,
  DependentBirth=OLD.DependentBirth, DependentGender=OLD.DependentGender
  WHERE DSIN=NEW.DSIN;
  SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'SIN must be exactly 9 digits';
  ELSEIF (NEW.DependentGender != 'F' AND NEW.DependentGender != 'M' AND
  NEW.DependentGender != '?') THEN
  UPDATE Dependent SET DSIN=OLD.DSIN, DependentName=OLD.DependentName,
  DependentBirth=OLD.DependentBirth, DependentGender=OLD.DependentGender
  WHERE DSIN=NEW.DSIN;
  SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'DependentGenderGender must be
  valid';
  END IF;
  END;//
  delimiter ;
  ```

Analysis:

- The entity Dependent has attributes DSIN, DependentName, DependentGender, DependentBirth. Both DSIN and DependentName are unique and either can uniquely identify a tuple, hence these are both candidate keys. In this case, DSIN is chosen as primary key, while setting DependentName as unique and not null.
- Attribute DependentGender is bounded by constraint that limits its value to ‘M’, ‘F’, and ‘?’. The constraint is implemented using trigger.
- Attribute DSIN is bounded by a constraint that limits its data to exactly 9 digits
- Attribute DependentBirth has a default value set in case user does not want to input this value.
- For each FD in this schema, the left hand side is a superkey. Hence this schema is in 3NF.

8.HaveDependent (<u>DSIN,SIN</u>)

SQL table creation query:

```
      CREATE TABLE HaveDependent(
      SIN INT,
     FOREIGN KEY(SIN) REFERENCES Employee(SIN) ON DELETE CASCADE ON
UPDATE CASCADE,
     DSIN INT,
      FOREIGN KEY (DSIN) REFERENCES Dependent(DSIN) ON DELETE CASCADE ON
UPDATE CASCADE,
     PRIMARY KEY(SIN, DSIN)
     );
```

Analysis:

- This table represents the many-to-many relation between Employee and Dependent, with DSIN and SIN as attributes. Due to the many-to-many nature, this relations needs both DSIN and SIN as keyset.
- DSIN is a foreign attribute taken from table Dependent, and SIN is another foreign attribute taken from Employee. The referential integrity is enforced using foreign key constraints.
- Policy on foreign key update and deletion is to cascade
- The keyset DSIN and SIN are also the only attributes of this schema, the 3NF condition is satisfied since all FDs are trivial.

9. Assign(DNumber,<u>SIN</u>, AssignStartDate, AssignEndDate)

FDs:

SIN->DNumber, AssignStartDate, AssignEndDate

SQL table creation query:

```
      CREATE TABLE Assign(
      SIN INT PRIMARY KEY,
      DNumber INT,
      AssignStartDate DATE NOT NULL ,
      AssignEndDate DATE CHECK(AssignEndDate>AssignStartDate ),
      FOREIGN KEY(SIN) REFERENCES Employee(SIN) ON DELETE CASCADE ON
UPDATE CASCADE,
      FOREIGN KEY(DNumber) REFERENCES Department(DNumber) ON DELETE
CASCADE ON UPDATE CASCADE
      );
```

CHECK implemented using TRIGGER:

-
  ```
  delimiter //
  CREATE TRIGGER AssignStartEndDate_check_insert
  BEFORE INSERT ON Assign
  FOR EACH ROW
  BEGIN
  IF (NEW.AssignStartDate < (SELECT StartDate FROM Employee WHERE
  Employee.SIN = NEW.SIN) OR NEW.AssignStartDate > (SELECT EndDate FROM
  Employee WHERE Employee.SIN = NEW.SIN)) THEN
  SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'AssignStartDate is incorrect.';
  ELSEIF (NEW.AssignEndDate < NEW.AssignStartDate OR NEW.AssignEndDate >
  (SELECT EndDate FROM Employee WHERE Employee.SIN = NEW.SIN))THEN
  SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'AssignEndDate is incorrect ';
  END IF;
  END;//
  delimiter ;
  ```

-
  ```
  delimiter //
  CREATE TRIGGER AssignStartEndDate_check_update
  AFTER UPDATE ON Assign
  FOR EACH ROW
  BEGIN
  IF (NEW.AssignStartDate < (SELECT StartDate FROM Employee WHERE
  Employee.SIN = NEW.SIN) OR NEW.AssignEndDate < NEW.AssignStartDate OR
  NEW.AssignStartDate > (SELECT EndDate FROM Employee WHERE Employee.SIN =
  NEW.SIN)) THEN
  UPDATE Assign SET AssignStartDate=OLD.AssignStartDate
  WHERE Assign.SIN=NEW.SIN;
  SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'AssignStartDate is incorrect.';
  ELSEIF (NEW.AssignEndDate < NEW.AssignStartDate OR NEW.AssignEndDate >
  (SELECT EndDate FROM Employee WHERE Employee.SIN = NEW.SIN)) THEN
  UPDATE Assign SET AssignEndDate=OLD.AssignEndDate,
  AssignStartDate=OLD.AssignStartDate
  WHERE Assign.SIN=NEW.SIN;
  SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'AssignEndDate is incorrect';
  END IF;
  END;//
  delimiter ;
  ```

Analysis:

- This table represents the many-to-one relation between Employee and Department, with attributes SIn and DNumber taken from table Employee and table Department, and local attributes AssignStartDate and AssignEndDate.

- Since the relation is many-to-one (one employee can only be assigned to one department, a department on the other hand can receive assignment from multiple employees), only SIN is used as primary key.
- Foreign key constraints are implemented to reinforce the integrity of SIN and DNumber attributes.
- Policy on update and deletion of foreign key is to cascade.
- Constraints are imposed onto AssignStartDate and AssignEndDate to ensure the assignment period of an employee to department is within the time range which the employee has been working in the company:
    - AssignStartDate cannot precede the employee's StartDate found in the table Employee
    - AssignEndDate cannot precede the AssignStartDate, but cannot be later than the employee's EndDate found in table Employee.
- For the FD in this schema, the left hand side is a superkey to the schema. Thus this schema is in 3NF.

10. IsManager(DNumber, <u>SIN</u>, ManagerStartDate, ManagerEndDate)

FDs:

SIN->DNumber, ManagerStartDate, ManagerEndDate

SQL table creation query:

```
        CREATE TABLE IsManager(
        SIN INT PRIMARY KEY CHECK(SIN IN (SELECT SIN FROM Assign WHERE
Assign.DNumber=IsManager.DNumber)),
         DNumber INT UNIQUE,
         ManagerStartDate DATE NOT NULL ,
         ManagerEndDate DATE CHECK (ManagerEndDate  > ManagerStartDate ),
         FOREIGN KEY(SIN) REFERENCES Assign(SIN) ON DELETE CASCADE ON UPDATE
CASCADE,
         FOREIGN KEY(DNumber) REFERENCES Department(DNumber) ON DELETE
CASCADE ON UPDATE CASCADE
         );
```

CHECK implemented using TRIGGER:

-
  ```
  delimiter //
  CREATE TRIGGER  IsManagerSINStartEndDate_check_insert
  BEFORE INSERT ON IsManager
  FOR EACH ROW
  BEGIN
  IF (NEW.SIN NOT IN (SELECT Assign.SIN FROM Assign,IsManager WHERE
  Assign.DNumber = NEW.DNumber)) THEN
  ```

```
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'This employee can not be
manager in different department.';
ELSEIF(NEW.ManagerStartDate < (SELECT AssignStartDate FROM Assign WHERE
Assign.SIN = NEW.SIN) OR NEW.ManagerStartDate > (SELECT AssignEndDate FROM
Assign WHERE Assign.SIN = NEW.SIN)) THEN
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'IsManagerStartDate is
incorrect.';
ELSEIF (NEW.ManagerEndDate < NEW.ManagerStartDate OR NEW.ManagerEndDate
> (SELECT AssignEndDate FROM Assign WHERE Assign.SIN = NEW.SIN))THEN
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'ManagerEndDate is incorrect ';
END IF;
END;//
delimiter ;
```

-
  ```
  delimiter //
  CREATE TRIGGER IsManagerSINStartEndDate_check_update
  AFTER UPDATE ON IsManager
  FOR EACH ROW
  BEGIN
  IF (NEW.SIN NOT IN (SELECT Assign.SIN FROM Assign,IsManager WHERE
  Assign.DNumber = NEW.DNumber)) THEN
  UPDATE IsManager SET SIN = OLD.SIN , DNumber = OLD.DNumber,
  ManagerStartDate = OLD.ManagerStartDate, ManagerEndDate =
  OLD.ManagerEndDate
  WHERE IsManager.SIN = NEW.SIN ;
  SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'This employee can not be
  manager in different department.';
  ELSEIF(NEW.ManagerStartDate < (SELECT AssignStartDate FROM Assign WHERE
  Assign.SIN = NEW.SIN) OR NEW.ManagerEndDate < NEW.ManagerStartDate OR
  NEW.ManagerStartDate > (SELECT AssignEndDate FROM Assign WHERE Assign.SIN
  = NEW.SIN)) THEN
  UPDATE IsManager SET SIN = OLD.SIN , DNumber = OLD.DNumber,
  ManagerStartDate = OLD.ManagerStartDate, ManagerEndDate =
  OLD.ManagerEndDate
  WHERE IsManager.SIN = NEW.SIN;
  SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'IsManagerStartDate is
  incorrect.';
  ELSEIF (NEW.ManagerEndDate < NEW.ManagerStartDate OR NEW.ManagerEndDate
  > (SELECT AssignEndDate FROM Assign WHERE Assign.SIN = NEW.SIN)) THEN
  UPDATE IsManager SET SIN = OLD.SIN , DNumber = OLD.DNumber,
  ManagerStartDate = OLD.ManagerStartDate, ManagerEndDate =
  OLD.ManagerEndDate
  WHERE IsManager.SIN = NEW.SIN;
  ```

```
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'ManagerEndDate is incorrect';
END IF;
END;//
delimiter ;
```

Analysis:

- *This table is created from the relation between Department and Employee.*
    - *Primary Key : SIN in this table is the SIN of people who is manager in Employee .SIN or DNumber can be primary Key for this table instead of both SIN and DNumber because it is one to one relation between Employee(who is manager) and Department and one attribute as primary key is enough. We choose SIN as primary key in this table. We also set DNumber is unique.*
    - *Foreign Key :SIN and DNumber is Primary Key for Employee and Department, so they should be the foreign key for this table. SIN is the foreign key references Employee and DNumber is the foreign key references from Department .Employee and Department are two entities which are related by Assign . We use delete cascade and update cascade for both two foreign key , which means that when the data in the foreign key is delete or update in the parent table , it should be delete or update in the child table.*
- *Constraints* are imposed onto ManagerStartDate and ManagerEndDate to make sure the time range is logical *:*
    - ManagerStartDate*(the date that an employee manager in a department must be later than the date that he assign in this department and ealier than the date he leave this appartment)but we can not add this check because the SQL do not support this in this version (implement by trigger).*
    - ManagerEndDate *(the date that an employee manage in a department must be earlier than the date that he stop managing in this department and the date he stop managing the this department must be earlier than the date he stop assigning this department.) .*
- *We have two triggers and the situation is similar as we refer before.*
- *FDs:We set* ManagerStartDate *not null .SIN is the super key. According to the FDs, the schema is on 3NF.*

11. WorkOn(<u>SIN,PID</u>,Hour,WorkOnStartDate,WorkOnEndDate)

FDs:

SIN,PID->Hour
SIN,PID->WorkOnStartDate
SIN,PID->WorkOnEndDate

SQL table creation query:

```
CREATE TABLE WorkOn(
SIN INT,
WorkOnStartDate DATE NOT NULL,
FOREIGN KEY(SIN) REFERENCES Employee(SIN) ON DELETE CASCADE ON
UPDATE CASCADE,
PID INT,
FOREIGN KEY(PID) REFERENCES Project(PID) ON DELETE CASCADE ON UPDATE
CASCADE,
Hour INT UNSIGNED DEFAULT 0,
PRIMARY KEY(SIN, PID)
);
```

CHECK implemented using TRIGGER:

```
delimiter //
CREATE TRIGGER WorkOnStartEndDate_check_insert
BEFORE INSERT ON WorkOn
FOR EACH ROW
BEGIN
IF (NEW.WorkOnStartDate < (SELECT StartDate FROM Employee WHERE Employee.SIN
= NEW.SIN) OR NEW.WorkOnStartDate < (SELECT ProjectStartDate FROM InCharge
WHERE InCharge.PID = NEW.PID)) THEN
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'WorkOnStartDate is incorrect.';
ELSEIF (NEW.WorkOnEndDate < NEW.WorkOnStartDate)THEN
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'WorkOnEndDate is incorrect ';
END IF;
END;//
delimiter ;

delimiter //
CREATE TRIGGER WorkOnStartEndDate_check_update
AFTER UPDATE ON WorkOn
FOR EACH ROW
BEGIN
IF (NEW.WorkOnStartDate < (SELECT StartDate FROM Employee WHERE Employee.SIN
= NEW.SIN) OR NEW.WorkOnStartDate < (SELECT ProjectStartDate FROM InCharge
WHERE InCharge.PID = NEW.PID)) THEN
UPDATE WorkOn SET SIN =OLD.SIN,PID =OLD.PID,Hour =OLD.Hour,WorkOnStartDate
=OLD.WorkOnStartDate,WorkOnEndDate =OLD.WorkOnEndDate
WHERE WorkOn.PID = NEW.PID AND WorkOn.SIN = NEW.SIN;
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'WorkOnStartDate is incorrect.';
ELSEIF (NEW.WorkOnEndDate < NEW.WorkOnStartDate) THEN
UPDATE WorkOn SET SIN =OLD.SIN,PID =OLD.PID,Hour =OLD.Hour,WorkOnStartDate
=OLD.WorkOnStartDate,WorkOnEndDate =OLD.WorkOnEndDate
WHERE WorkOn.PID = NEW.PID AND WorkOn.SIN = NEW.SIN;
```

```
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'WorkOnEndDate is incorrect';
END IF;
END;//
delimiter ;
```

Analysis:

- *This table is created from the relation between Project and Employee. SIN in this table is the SIN of people who is working on the project .*
    - *Primary Key :Both SIN and PID are primary Key for this table because it is many to many relation between Employee and Project . Only a couple attributes can be primary key .*
    - *Foreign key : SIN and PID is Primary Key for Employee and Project , so they should be the foreign key for this table. SIN is the foreign key references Employee and PID is the foreign key references from Project . Employee and Project are two entities which are related by WorkOn . We use delete cascade and update cascade for both two foreign key , which means that when the data in the foreign key is delete or update in the parent table , it should be delete or update in the child table.*
- *Constraints* are imposed onto WorkOnStartDate and *WorkOn*EndDate to make sure the time range is logical *:*
    - WorkOnStartDate*(the date that an employee WorkOn in a project must be later than the date that he join this company and later than the date this project start)but we can not add this check because the SQL do not support this in this version (implement by trigger).*
    - *WorkOn*EndDate *(the date that an employee work on a project must be earlier than the date that he start working on this project) .*
- *Triggers :We have two triggers and the situation is similar as we refer before.*
- *FDs :We set* WorkOnStartDate *not null.SIN and PID is super key. According to the FDs, the schema is on 3NF.*

12. SuperOf (SupervisorSIN, <u>SubordinateSIN</u>).

FDs:

SubordinateSIN-> SupervisorSIN

SQL table creation query:

```
CREATE TABLE SuperOf(
SupervisorSIN INT,
SubordinateSIN INT PRIMARY KEY,
FOREIGN KEY(SupervisorSIN) REFERENCES Employee(SIN) ON DELETE CASCADE
ON UPDATE CASCADE,
FOREIGN KEY(SubordinateSIN) REFERENCES Employee(SIN) ON DELETE
CASCADE ON UPDATE CASCADE
);
```

- *This table is the self-relation in Employee.*
- *Primary Key :An employee can have many subordinate and only have one supervisor so it is many to one relation. We set SubordinateSIN as primary key .*
- *Foreign Key: SubordinateSIN and SuperviorSIN are foreigen key references Employee(SIN).We use delete cascade and update cascade for both two foreign key , which means that when the data in the foreign key is delete or update in the parent table , it should be delete or update in the child table.*
- *FDs :SubordinateSIN is super key . According to the FDs, the schema is on 3NF.*

13. InCharge(DNumber, PID, PL, ProjectStartDate, ProjectEndDate)

FDs:

PID->DNumber, PL, StartDate, EndDate

SQL table creation query:

```
CREATE TABLE InCharge(
DNumber INT,
PID INT PRIMARY KEY,
PL INT  NOT NULL CHECK(PL IN (SELECT SIN FROM Assign WHERE
Assign.DNumber = InCharge.DNumber)),
ProjectStartDate DATE NOT NULL,
ProjectEndDate DATE CHECK (ProjectEndDate > ProjectStartDate),
FOREIGN KEY(DNumber) REFERENCES Department(DNumber) ON DELETE
CASCADE ON UPDATE CASCADE,
FOREIGN KEY(PID) REFERENCES Project(PID) ON DELETE CASCADE ON UPDATE
CASCADE
foreign key (PL) references Employee(SIN) ON DELETE CASCADE ON UPDATE
CASCADE;
);
```

CHECK implemented using TRIGGER:

```
delimiter //
CREATE TRIGGER  InChargePlEndDate_check_insert
BEFORE INSERT ON InCharge
FOR EACH ROW
BEGIN
IF (NEW.PL NOT IN (SELECT Assign.SIN FROM Assign,InCharge WHERE
Assign.DNumber = NEW.DNumber)) THEN
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'This employee can not be Project
Leader in different department.';
```

```
ELSEIF (NEW.ProjectEndDate < NEW.ProjectStartDate )THEN
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'ProjectEndDate is incorrect ';
END IF;
END;//
delimiter ;

delimiter //
CREATE TRIGGER InChargePlEndDate_check_update
AFTER UPDATE ON InCharge
FOR EACH ROW
BEGIN
IF (NEW.PL NOT IN (SELECT Assign.SIN FROM Assign,InCharge WHERE
Assign.DNumber = NEW.DNumber)) THEN
UPDATE InCharge SET DNumber = OLD.DNumber, PID = OLD.PID, PL = OLD.PL,
ProjectStartDate = OLD.ProjectStartDate, ProjectEndDate = OLD.ProjectEndDate
WHERE InCharge.PID = NEW.PID ;
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'This employee can not be
manager in different department.';
ELSEIF (NEW.ProjectEndDate < NEW.ProjectStartDate ) THEN
UPDATE InCharge SET DNumber = OLD.DNumber, PID = OLD.PID, PL = OLD.PL,
ProjectStartDate = OLD.ProjectStartDate, ProjectEndDate = OLD.ProjectEndDate
WHERE InCharge.PID = NEW.PID;
SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'InChargeEndDate is incorrect';
END IF;
END;//
delimiter ;
```

- *This table is created from the relation between Project and Department.*
- *Primanry key : PID can be primary Key for this table instead of both PID and DNumber because it is one to many relation(one department can have many Project) between Project and Department.*
- *Foreign key :*
    - *PID and DNumber : PID and DNumber is Primary Key for Project and Department, so they should be the foreign key for this table. Project and Department are two entities which are related by InCharge . PID is the foreign key references Project and DNumber is the foreign key references from Department .*
    - *PL : PL is the SIN of an employee who is the leader of the Project so PL is foreign key references Employee. The WorkOnStartDate of PL must earlier than the project End date and the assignEndDate of PL must later than the project start date.This situation is already considered in the trigger in WorkOn table part.*
    - *We use delete cascade and update cascade for PID,DNumber and PL, which means that when the data in the foreign key is delete or update in the parent table , it should be delete or update in the child table.*

- *Constraints* are imposed on to *PL* and *ProjectEndDate* to make sure the time range is logical *:*
  - *PL : an employe who is the leader of a Project must belongs to the department which is in charge this Project.*
  - *ProjectEndDate : the date that an project end must be later than the date this project start.*
- *Triggers :We have two triggers and the situation is similar as we refer before.*
- *FDs:We set ProjectStartDate not null.PID is the super key. According to the FDs, the schema is on 3NF.*

DBMS Implementation

Queries used for main functions:

Department Management

- Show all existing department:

SELECT Department.DNumber AS DepartmentNumber, DName AS DepartmentName, IsManager.SIN AS ManagerSIN, Name AS ManagerName, ManagerStartDate, ManagerEndDate,
(SELECT COUNT(*) FROM Assign WHERE Assign.DNumber=Department.DNumber) AS NoOfMembers
FROM Department LEFT JOIN IsManager ON Department.DNumber=IsManager.DNumber LEFT JOIN Employee ON IsManager.SIN=Employee.SIN ORDER BY DepartmentNumber

- Search by attributes:

Base query: SELECT DNumber AS DepartmentNumber, DName AS DepartmentName, SIN AS ManagerSIN, Name AS ManagerName, ManagerStartDate, ManagerEndDate,
(SELECT COUNT(*) FROM Assign WHERE Assign.DNumber=Department.DNumber) AS NoOfMembers
FROM Department LEFT JOIN IsManager ON Department.DNumber=IsManager.DNumber LEFT JOIN Employee ON IsManager.SIN=Employee.SIN WHERE

Dynamic query: The WHERE conditions are appended dynamically based on user selections

- Delete Department

DELETE FROM Department WHERE DNumber=$DNumber

- Update Department and department related attributes

Transactions: the following queries are wrapped in transaction. Before executing the queries below, "START TRANSACTION;" is first sent to SQL server. Once the following queries finish execution, the DBMS checks status of their execution, and only COMMIT transaction if all execution return successful. Otherwise, DBMS will ROLLBACK transaction.

UPDATE Department SET DName='$DName' WHERE DNumber=$DNumber
INSERT INTO IsManager(SIN, DNumber, ManagerStartDate, ManagerEndDate) VALUES ($ManagerSIN, $DNumber, NOW(), $ManagerEndDate)
UPDATE IsManager SET SIN=$ManagerSIN, ManagerStartDate=NOW(), ManagerEndDate=$ManagerEndDate WHERE DNumber=$DNumber

- Single department expenses

SELECT PID, PName, SUM(Hour) AS TotalHour FROM InCharge NATURAL JOIN Project NATURAL JOIN WorkOn WHERE DNumber=$DNumber GROUP BY PID

Calculations are performed based on data from the above query.

Employee Management

- Show Existing Employees

SELECT SIN, Name, DNumber AS DID, DName AS Department, AssignStartDate AS AssignedOn, SupervisorSIN, Birth AS BirthDate, HomeAddress,
 Phone AS PhoneNumber, Gender, Salary, StartDate AS HiredOn, EndDate AS LeftOn FROM Employee NATURAL JOIN Assign

```
  NATURAL JOIN Department LEFT JOIN SuperOf ON SIN=SuperOf.SubordinateSIN
  UNION
  SELECT SIN, Name, -1, 'UNASSIGNED', '0000-00-00', SupervisorSIN, Birth AS BirthDate, HomeAddress,
  Phone AS PhoneNumber, Gender, Salary, StartDate AS HiredOn, EndDate AS LeftOn FROM Employee LEFT
JOIN SuperOf ON SIN=SuperOf.SubordinateSIN
  WHERE SIN NOT IN
  (SELECT SIN FROM Employee NATURAL JOIN Assign NATURAL JOIN Department)
  ORDER BY SIN
```

- Search Employee by attribute

Base query, first half: `SELECT SIN, Name, DNumber as DID, DName AS Department, AssignStartDate AS AssignedOn,Birth AS BirthDate, HomeAddress,`
`  Phone AS PhoneNumber, Gender, Salary, StartDate AS HiredOn, EndDate AS LeftOn FROM Employee NATURAL JOIN Assign`
`  NATURAL JOIN Department WHERE`

Base query, second half: `SELECT SIN, Name, 'UNASSIGNED', '0000-00-00',Birth AS BirthDate, HomeAddress,`
`  Phone AS PhoneNumber, Gender, Salary, StartDate AS HiredOn, EndDate AS LeftOn FROM Employee WHERE SIN NOT IN`
`  (SELECT SIN FROM Employee NATURAL JOIN Assign NATURAL JOIN Department) AND`

Dynamic query: WHERE conditions are appended to both base queries according to dynamic user selections, then the base queries are linked using UNION

- Delete Employee

```
DELETE FROM Employee WHERE SIN=$SIN
```

- Update Employee

Transaction: The following queries for updating Employee and related tables are wrapped inside SQL transaction.

```
INSERT INTO Assign(SIN, DNumber, AssignStartDate) VALUES ($SIN, $DID, NOW())
UPDATE Assign SET DNumber=$DID, AssignStartDate=NOW() WHERE SIN=$SIN
INSERT INTO SuperOf(SupervisorSIN, SubordinateSIN) VALUES ($SupervisorSIN, $SIN)
UPDATE SuperOf SET SupervisorSIN=$SupervisorSIN WHERE SubordinateSIN=$SIN
UPDATE Employee SET Name='$Name', Birth='$BirthDate', HomeAddress='$HomeAddress',
Phone=$PhoneNumber, Gender='$Gender', Salary=$Salary, EndDate=$LeftOn WHERE SIN=$SIN
```

- Single employee total salary calculation

Hour/project distribution: `SELECT WorkOn.PID, PName, Hour, WorkOnStartDate, WorkOnEndDate FROM WorkOn LEFT JOIN Project ON WorkOn.PID = Project.PID WHERE SIN=$SIN`

Total hour: `SELECT SUM(Hour) AS TotalHour FROM WorkOn WHERE SIN=$SIN`

Total salary: `SELECT Salary FROM Employee WHERE SIN=$SIN`

Calculations are performed based on data gathered from above queries

Project Management

- Show Existing Projects

```
SELECT Project.PID, PName, InCharge.DNumber AS DID, DName AS DepartmentInCharge, Stage, PL AS
ProjectLeaderSIN, Name AS ProjectLeaderName,
(SELECT COUNT(*) FROM WorkOn WHERE WorkOn.PID=Project.PID) AS NoOfMembers,
(SELECT SUM(Hour) FROM WorkOn WHERE WorkOn.PID=Project.PID) AS TotalWorkHours, Address AS Location
```

```
FROM Project LEFT JOIN InCharge ON Project.PID=InCharge.PID
LEFT JOIN Department ON InCharge.DNumber=Department.DNumber
LEFT JOIN Employee ON PL=SIN LEFT JOIN ProjectLocation ON Project.PID=ProjectLocation.PID
ORDER BY PID
```

- Search by attributes

Base query:

```
SELECT Project.PID, PName, InCharge.DNumber AS DID, DName AS DepartmentInCharge,
Stage, PL AS ProjectLeaderSIN, Name AS ProjectLeaderName,
(SELECT COUNT(*) FROM WorkOn WHERE WorkOn.PID=Project.PID) AS NoOfMembers,
(SELECT SUM(Hour) FROM WorkOn WHERE WorkOn.PID=Project.PID) AS TotalWorkHours, Address AS Location
FROM Project LEFT JOIN InCharge ON Project.PID=InCharge.PID
LEFT JOIN Department ON InCharge.DNumber=Department.DNumber
LEFT JOIN Employee ON PL=SIN LEFT JOIN ProjectLocation ON Project.PID=ProjectLocation.PID
WHERE PL IS NULL
ORDER BY PID
```

Dynamic query: WHERE condition is appended based on user selections

- Delete Project

```
DELETE FROM Project WHERE PID=$PID
```

- Update Project and related data

Transaction: queries used to update Project tuple and related tuple are wrapped inside SQL transaction.

```
UPDATE Project SET PName = '$PName', Stage = '$Stage' WHERE PID=$PID
INSERT INTO InCharge (DNumber, PID, PL, ProjectStartDate) VALUES ($DID, $PID, $ProjectLeaderSIN, NOW())
UPDATE InCharge SET DNumber=$DID, PL=$ProjectLeaderSIN WHERE PID=$PID
UPDATE ProjectLocation SET Address='$Location' WHERE PID=$PID
```

- Single project expense calculation

```
SELECT WorkOn.SIN, Name, Hour, Salary FROM WorkOn NATURAL JOIN Employee WHERE PID=$PID
```

Calculations are performed based on data gathered from the query above.