ECE 356 Lab2 Employee

Lab section 206
Group 10

Victor Yan
Lun Jing

1. Decomposition
   First, not all of the attributes are atomic, so that will be handled first to at least be in 1NF.
   The attribute ‘empName’ in “Employee” can be split into three attributes, ‘firstName’, ‘middleName’, and ‘lastName’, and ‘middleName’ can be null for employees with no middle name as it is optional.
   The attribute ‘location’ in “Department” can be split into the attributes: ‘streetNumber’, ‘streetName’, ‘cityName’, ‘province’, and ‘postalCode’.
   Now that all the attributes are atomic, the decomposition for BCNF continues based on the functional dependencies of each relation. The decomposition is split into four parts (a, b, c, d) for each of the tables.

   a. Employee
      The functional dependencies for the “Employee” table are as follows:

      empID → {empName, job, salary}

      According to the constraints on this table, empID is unique for each employee, but also, employees may belong to multiple departments. In this case, the empID is not a candidate key of the relation since it does not uniquely identify department; as it is now, multiple rows are needed using the same empID to indicate the multiple different departments that an employee belongs to. Instead, we should use tuple of empID and deptID as the candidate key. Nevertheless, by considering the functional dependencies, empID → {empName, job, salary}, BCNF is violated. The functional dependency in question is not trivial, empID is not a superkey, and neither empName, job, nor salary are a part of any candidate key.

      To solve this, we should seperate Employee into two tables, Firstly, remove the deptID from the Employee table, which we will now call EmployeeData. Since empID uniquely identifies the remaining attributes and there are no other non-trivial functional dependencies, EmployeeData will be in BCNF.

      Then, create another table called EmployeeDepartment with two variables, empID and deptID. The tuple {empID, deptID} will be the candidate key, and all functional dependencies within this relation are trivial. Here, EmployeeDepartment will be in BCNF.

Finally, one more constraint is that the middle name of employee is optional, in that case, we should move middle names out to another table, since some employees have it but some do not. So one more table called "EmployeeMiddleName" will be created with attributes 'empID' and 'middleName', then for those who does not have middle names, just have no entries in this table. This disambiguates the notion of empty strings and null values from possibly indicating a middleName as known to exist, but is simply unknown in value.

b. Project

Since projID is unique for each project, thus there exists a function dependency projID → {title, budget, funds}, and projID can be the primary key of Project. This table is already BCNF.

c. Assigned

Due to the constraints, that 'projID' is unique, 'empID' is unique, and one employee can take multiple projects or multiple roles in one project. The specified constraints do not mention multiple whether multiple employees can take on the same role in a project, so to be safe, we will assume this is possible. In this case, the only candidate key involves {empID, projID, role} → {empID, projID, role}, which is a trivial functional dependency, meaning that this relation is already in BCNF.

d. Department

The following functional dependencies are defined for this relation:

deptID → deptName
streetNumber, streetName, cityName, province → postalCode

Due to the constraints, the each 'deptID' is unique for department, but a department may have multiple locations, so we can define the first functional dependency. 'deptID' does not functionally determine the location since multiple rows with the same 'deptID' are needed to list multiple departments, meaning that 'deptID' does not uniquely identify a single location. As discussed above, the 'location' attribute was split into multiple different atomic attributes to have "Department" be in 1NF. The second functional dependency is under the assumption that 'postalCode' uniquely identifies a location.

Currently, the only candidate key of this relation is {deptID, streetNumber, streetName, cityName, province}. The second functional dependency violates BCNF as it is untrivial, the tuple {streetNumber, streetName, cityName, province} is not a superkey, and the postalCode attributeis not a part of the sole candidate key {deptID, streetNumber, streetName, cityName, province}. This means that this must be decomposed into two relations: "Department" can have the attribute 'postalCode'' removed and placed along with {streetNumber, streetName, cityName, province} in a new relation called "Address", where {streetNumber, streetName, cityName, province} is the candidate key of the relation.

Next, since a department can have more than one location, there will be multiple rows using the same 'deptID' but with different locations (i.e. postal codes), so we need to use the tuple {deptID, streetNumber, streetName, cityName, province} as a candidate key. Since deptID → deptName, BCNF will be violated, as this dependency is not trivial, deptID is not a superkey, and deptName is not part of the sole candidate key {deptID, streetNumber, streetName, cityName, province}. To solve this, we should separate the

‘deptName’ attribute to a new table, and add ‘deptID’ to it. This new relation/table can be called “DepartmentName”, with only ‘deptID’ and ‘deptName’. The remaining relation can be called DepartmentAddress and will contain {deptID, streetNumber, streetName, cityName, province}.

Overall, the final tables with their attributes should be like this:

EmployeeData: {‘empID’, ‘firstName’, ‘lastName’, ‘job’, ‘salary’}
EmployeeDepartment: {‘empID’, ‘deptID’}
EmployeeMiddleName: {‘empID’, ‘middleName’}
Project: {‘projID’, ‘title’, ‘budget’, ‘funds’}
Assigned: {‘empID’, ‘projID’, ‘role’}
DepartmentName: {‘deptID’, ‘deptName’}
DepartmentAddress: {‘deptID’, ‘streetNumber’, ‘streetName’, ‘cityName’, ‘province’}
Address: {‘streetNumber’, ‘streetName’, ‘cityName’, ‘province’, ‘postalCode’}

2. **Primary Keys**
   EmployeeData: {‘empID’}
   EmployeeDepartment: {‘empID’, ‘deptID’}
   EmployeeMiddleName: {‘empID’}
   Project: {‘projID’}
   Assigned: {‘empID’, ‘projID’, ‘role’}
   DepartmentName: {‘deptID’}
   DepartmentAddress: {‘deptID’, ‘streetNumber’, ‘streetName’, ‘cityName’, ‘province’}
   Address: {‘streetNumber’, ‘streetName’, ‘cityName’, ‘province’}

3. **Foreign Keys**
   EmployeeData:
   None

   EmployeeDepartment:
   ‘empID’ references Employee(‘empID’)
   ‘deptID’ references Department(‘deptID’)

   EmployeeMiddleName:
   ‘empID’ references Employee(‘empID’)

   Project:
   None

   Assigned:
   ‘empID’ references Employee(‘empID’)
   ‘projID’ references Project(‘projID’)

   DepartmentName:
   None

   DepartmentAddress:
   ‘deptID’ references Department(‘deptID’)

{‘streetNumber’, ‘streetName’, ‘cityName’, ‘province’} references Address(‘streetNumber’, ‘streetName’, ‘cityName’, ‘province’)

Address:
None

4. Queries

```
Create table EmployeeData (
      empID int(11) primary key,
      firstName varchar(100),
      lastName varchar(100),
      job varchar(100),
      salary int(11)
);

Create table EmployeeMiddleName (
      empID int(11) primary key,
      middleName varchar(100),
      foreign key(empID) references EmployeeData(empID)
);

Create table DepartmentName (
      deptID int(11) primary key,
      deptName varchar(100)
);

Create table EmployeeDepartment (
      empID int,
      deptID int,
      foreign key(empID) references EmployeeData(empID),
      foreign key(deptID) references DepartmentName(deptID)
);

Create table Project (
      projID int(11) primary key,
      Title varchar(100),
      Budget int(11),
      Funds int(11)
);

CREATE TABLE Assigned (
      empID int(11),
      projID int(11),
      role varchar(11),
```

```
      FOREIGN KEY (empID) REFERENCES EmployeeData(empID),
      FOREIGN KEY (projID) REFERENCES Project(projID));

CREATE TABLE Address (
      streetNumber int(11),
      streetName varchar(100),
      cityName varchar(100),
      province varchar(100),
      postalCode char(10),
      PRIMARY KEY (streetNumber, streetName, cityname, province));

CREATE TABLE DepartmentAddress (
      deptID int(11),
      streetNumber int(11),
      streetName varchar(100),
      cityName varchar(100),
      province varchar(100),
      FOREIGN KEY (deptID) REFERENCES DepartmentName(deptID),
      FOREIGN KEY (streetNumber, streetName, cityName, province)
REFERENCES Address(streetNumber, streetName, cityName, province));
```

5.

```
CREATE VIEW Employee
AS SELECT EData.empID, CONCAT(EData.firstName, ' ',
IFNULL(EMN.middleName, ''), ' ', EData.lastName) AS empName,
EData.job, EDept.deptID, EData.salary
FROM EmployeeData AS EData
LEFT OUTER JOIN EmployeeMiddleName AS EMN ON (EData.empID = EMN.empID)
LEFT OUTER JOIN EmployeeDepartment AS EDept ON (EData.empID =
EDept.empID);


CREATE VIEW Department
AS SELECT DN.deptID, DN.deptName, CONCAT(DA.streetNumber, ' ',
DA.streetName, ', ', DA.cityName, ', ', DA.cityName, ', ',
DA.province) as location
FROM DepartmentName AS DN
LEFT OUTER JOIN DepartmentAddress AS DA ON (DN.deptID = DA.deptID)
INNER JOIN Address AS A ON (DA.streetNumber = A.streetNumber AND
DA.streetName = A.streetName AND DA.cityName = A.cityName AND
DA.province = A.province);
```

6.

```
DROP PROCEDURE IF EXISTS payRaise;

DELIMITER //

CREATE PROCEDURE payRaise(IN inEmpID int(11), IN inPercentageRaise
double(4, 2), OUT errorCode int(11))
BEGIN
      IF inPercentageRaise > 0.10 OR inPercentageRaise < 0 THEN
      SET errorCode = -1;
      ELSEIF NOT EXISTS(SELECT * FROM EmployeeData WHERE empID =
inEmpID) THEN
      SET errorCode = -2;
      ELSE
      UPDATE EmployeeData SET salary = ROUND(salary * (1.0 +
inPercentageRaise))
      WHERE empID = inEmpID;
      SET errorCode = 0;
      END IF;
END//

CREATE PROCEDURE updateWaterlooSalaries()
BEGIN
      DECLARE i INT(11) DEFAULT 0;

      DROP TEMPORARY TABLE IF EXISTS WaterlooEmployees;
      CREATE TEMPORARY TABLE WaterlooEmployees
      (SELECT DISTINCT(empID)
      FROM EmployeeData
      LEFT OUTER JOIN EmployeeDepartment USING (empID)
      INNER JOIN DepartmentAddress USING (deptID)
      WHERE cityName='Waterloo');

      SET i = 0;
      SELECT COUNT(*) INTO @size FROM WaterlooEmployees;
      WHILE i < @size DO
      SELECT empID INTO @empID FROM WaterlooEmployees LIMIT i,1;
      CALL payRaise(@empID, 0.05, @out_value);
      SET i = i + 1;
      END WHILE;

      DROP TEMPORARY TABLE WaterlooEmployees;
END//
```

```
DELIMITER ;
```

TEMPORARY ITEMS

```
INSERT INTO EmployeeData
VALUES     (1, 'Bob', 'Marley', 'Music Artist', 123456),
       (2, 'Sam', 'Qing', 'Random Guy', 9001),
       (3, 'Victor', 'Yan', 'Student', 1000),
       (4, 'George', 'Washington', 'POTUS', 999999);


INSERT INTO DepartmentName
VALUES     (1, 'Icon'),
       (2, 'Blair'),
       (3, 'KW4Dept');


INSERT INTO EmployeeMiddleName
VALUES     (2, 'Ling'),
       (3, 'Li');


INSERT INTO EmployeeDepartment
VALUES     (1, 1),
       (2, 2),
       (3, 3),
       (4, 2),
       (4, 1);

INSERT INTO Address
```

```
VALUES     (1, 'Main Street', 'Waterloo', 'Ontario', 'A0A0A0'),
           (100, 'First Street', 'Ottawa', 'Ontario', 'B1B1B1'),
           (21, 'Yonge Street', 'Toronto', 'Ontario', 'C1C1C1'),
           (4, 'Mainer Street', 'Waterloo', 'Ontario', 'A1A3A4');

INSERT INTO DepartmentAddress
VALUES     (1, 1, 'Main Street', 'Waterloo', 'Ontario'),
           (2, 100, 'First Street', 'Ottawa', 'Ontario'),
           (3, 21, 'Yonge Street', 'Toronto', 'Ontario'),
           (3, 4, 'Mainer Street', 'Waterloo', 'Ontario');
```