(1)(2)(3)

Employee		Project	Assigned		Department			
EmployeeInfo	EmployeeDept	Project	EmployeeRole	ProjectRoles	DeptNames	DeptLocation	PostCode	LocationInfo
empID **int(11)**	**empID** **int(11)**	**projID** **int(11)**	**roleID** **int(11)**	**roleID** **int(11)**	**deptID** **int(11)**	**locationID** **int(11)**	**postalCode** varchar(100)	**locationID** **int(11)**
firstName varchar(100)	**deptID** **int(11)**	title varchar(100)	**empID** **int(11)**	projID int(11)	deptName varchar(100)	**deptID** **int(11)**	city varchar(100)	postalCode varchar(100)
lastName varchar(100)		budget int(11)		role varchar(100)			province varchar(100)	streetNumber varchar(100)
middleName varchar(100)		funds int(11)						streetName varchar(100)
job varchar(100)								
salary int(11)								

Bolded values are the primary keys for that table.

Foreign keys:

EmployeeDept.empID -> EmployeeInfo.empID

EmployeeDept.deptID -> DeptNames.deptID

EmployeeRole.roleID -> ProjectRoles.roleID

EmployeeRole.empID -> EmployeeInfo.empID

ProjectRoles.projID -> Project.projID

DeptLocation.locationID -> LocationInfo.locationID

DeptLocation.deptID -> DeptNames.deptID

LocationInfo.postalCode -> PostCode.postalCode

(4)

Create tables

```
create table EmployeeInfo(empID int(11) primary key, firstName varchar(100),
                          lastName varchar(100), middleName varchar(100),
                          job varchar(100), deptID int(11), salary int(11));

create table EmployeeDept(empID int(11), deptID int(11),
                          primary key(empID, deptID),
```

```
                        foreign key(empID) references EmployeeInfo(empID),
                        foreign key(deptID) references DeptNames(deptID)
                      );

create table Project(projID int(11) primary key, title varchar(100),
                     budget int(11), funds int(11));

create table EmployeeRole(roleID int(11) primary key, empID int(11),
                        foreign key(empID) references EmployeeInfo(empID),
                        foreign key(roleID) references ProjectRoles(roleID)
                      );

create table ProjectRoles(roleID int(11) primary key, projID int(11),
                        role varchar(100),
                        foreign key(projID) references Project(projID)
                      );

create table DeptNames(deptID int(11) primary key, deptName varchar(100));

create table DeptLocation(locationID int(11), deptID int(11),
                        primary key(locationID, deptID),
                        foreign key(locationID) references
                        LocationInfo(locationID),
                        foreign key(deptID) references DeptNames(deptID)
                      );

create table PostCode(postalCode varchar(100) primary key, city varchar(100),
                      province varchar(100));

create table LocationInfo(locationID int(11) primary key,
                        postalCode varchar(100), streetNumber varchar(100),
                        streetName varchar(100),
                        foreign key(postalCode) references
                        PostCode(postalCode),
                      );
```

(5)

The Employee, Assigned and Department tables were broken due to converting it to BCNF, so we must create the views down here.

```
create view Employee as select empID, CONCAT(firstName, " ", lastName, " ",
       middleName) AS empName, job, deptID, salary from EmployeeInfo
       left join EmployeeDept using(empID);
```

```
create view Assigned as select empID, projID, role from EmployeeRole left join
       ProjectRoles using(roleID);

create view Department as select deptID, deptName, CONCAT(streetNumber, " ",
       streetName, " ", city, " ", province, " ", postalCode)
       AS location from DeptNames left join DeptLocation using (deptID) left join
       LocationInfo using(locationID) natural join PostCode;
```

(6)

The following is a stored procedure that takes two input parameters “inEmpID” (Int) and “inPercentageRaise” (double 4,2) and one output parameter “errorCode” (int). In normal operation the procedure will raise the salary of the associated employee by the input percentage and return an errorCode of 0. However, if the payRaise is by more than 10% or less than 0% (i.e., it is a pay cut), it will return -1. If the employee does not exist, it will return an errorCode of -2.

```
DELIMITER @@
create procedure payRaise(IN inEmpID int, IN inPercentageRaise double(4,2), OUT
errorCode int)

BEGIN

DECLARE stmt VARCHAR(255);
DECLARE SQL3 VARCHAR(255);

IF (inPercentageRaise > 10.00 OR inPercentageRaise) < 0.00 THEN
SET errorCode = -1;
ELSEIF (select exists (select empID from Employee WHERE empID=inEmpID)) THEN SET
errorCode = 2;
ELSE
SET errorCode = 0;
END IF;

SET SQL3 = CONCAT('UPDATE Employee SET salary = sum(salary, salary / 100 * ',
inPercentageRaise, ') WHERE empID =', inEmpID);

PREPARE stmt FROM 'select * from Employee';
EXECUTE stmt;
DEALLOCATE PREPARE stmt;
SELECT errorCode;
END@@
DELIMITER ;
```

The following is a query to increase the salary of all employees at Waterloo location by 5%.

```
create view `waterlooOnly` as select empID,salary from Employee inner join
Department using (deptID) where location=`Waterloo`;

DELIMITER @@
create procedure waterlooPayRaise()
BEGIN
DECLARE n INT;
DECLARE i INT;
DECLARE inEmpIDWaterloo INT;
DECLARE inPercentageRaiseWaterloo double(4,2);
DECLARE errorCodeWaterloo INT;

SET n = 0;
SET i = 0;
SET inPercentageRaiseWaterloo = 5.00;
SELECT COUNT(*) FROM EmployeeInfo INTO n;
SET i=0;
WHILE i<n DO
   SELECT empID FROM waterlooOnly LIMIT i,1 INTO inEmpIDWaterloo;
   CALL payRaise(inEmpIDWaterloo, inPercentageRaiseWaterloo, errorCodeWaterloo);
   SET i = i + 1;
END WHILE;
END@@
DELIMITER ;
```