PART 1: A Professorial ERD to Tables

```
CREATE TABLE Conference (
conName         CHAR (40),
conYear         INTEGER,
attendance      INTEGER,
location        CHAR (40),
employeeID      CHAR (9) NOT NULL,
title            CHAR (40) NOT NULL,
FOREIGN KEY (employeeID) REFERENCES Professor,
FOREIGN KEY (title) REFERENCES Research Paper,
PRIMARY KEY (conName, conYear)
)
```

Conference Table:
employeeID and title are foreign keys and to show the participation constraint that conference has with aggregated entity we should use key word NOT NULL. conYear and conName are primary keys

```
CREATE TABLE Writes (
employeeID      CHAR (9),
title            CHAR (40),
billing         INTEGER,
FOREIGN KEY (employeeID) REFERENCES Professor,
FOREIGN KEY (title) REFERENCES Research Paper,
PRIMARY KEY (employeeID, title)
)
```

Writes Table:
Since each professor with their specific research paper can publish one conference at most, having a table for publishes is unnecessary, but because aggregated entity doesn’t have total participation and writes has its descriptive attribute, we need a table for writes, including Professor and Research paper primary keys as its foreign key.

```
CREATE TABLE Research Paper (
title            CHAR (40),
field           CHAR (10),
employeeID      CHAR (9) NOT NULL,
FOREIGN KEY (employeeID) REFERENCES Professor,
PRIMARY KEY (title)
)
```

Research Paper Table:
It has participation constraint, so we need Professor’s primary key to be NOT NULL.

```
CREATE TABLE Professor (
employeeID    CHAR (9),
name          CHAR (40) NOT NULL,
tenureDeadline DATETIME,
title          CHAR (40) NOT NULL,
FOREIGN KEY (title) REFERENCES Research Paper,
PRIMARY KEY (employeeID)
)
```

Professor Table:

It has participation constraint with Research Paper, so we need Research Paper's primary key to be NOT NULL, but it doesn't have any constraint with course.

```
CREATE TABLE Course (
Capacity            INTEGER,
courseNumber INTEGER,
termID              CHAR (9),
department          CHAR (4),
year                INTEGER,
employeeID    CHAR (9) NOT NULL,
studentID           CHAR (9) NOT NULL,
FOREIGN KEY (employeeID) REFERENCES Professor,
FOREIGN KEY (studentID) REFERENCES TA,
PRIMARY KEY (courseNumber, termID, department)
)
```

Course Table:

Course has participation constraint with professor and TA, so we need both of their primary keys in course table as foreign key while they are NOT NULL. Course doesn't have any constraint with grade. Also course primary key is compound, so we must mention all three of them.

```
CREATE TABLE Supports (
Pay                 REAL,
courseNumber INTEGER,
termID              CHAR (9),
department          CHAR (4),
studentID           CHAR (9),
FOREIGN KEY (courseNumber, termID, department) REFERENCES Course,
FOREIGN KEY (studentID) REFERENCES TA,
PRIMARY KEY (courseNumber, termID, department, studentID)
)
```

Supports Table:

Supports is a many-to-many relationship, so we need to capture both entities primary keys. Note that course primary key is compound.

```
CREATE TABLE TA (
studentID           CHAR (9),
name                CHAR (40) NOT NULL,
```

courseNumber INTEGER NOT NULL,
termID CHAR (9) NOT NULL,
department CHAR (4) NOT NULL,
FOREIGN KEY (courseNumber, termID, department) REFERENCES Course,
PRIMARY KEY (studentID)
)
TA Table:
TA has participation constraint with course, so course's primary key is a foreign key in TA that is NOT NULL.

CREATE TABLE Grade (
Email CHAR (20),
studentName CHAR (40),
finalGrade CHAR (2),
courseNumber INTEGER,
termID CHAR (9),
department CHAR (4),
CONSTRAINT unique_email UNIQUE (email),
FOREIGN KEY (courseNumber, termID, department) REFERENCES Course ON DELETE CASCADE,
PRIMARY KEY (email, courseNumber, termID, department)
)
Grade Table:
Garde is a weak entity so the primary key of course is needed, and we should mention the key word ON DELETE CASCADE when we're mentioning the foreign key. As we know based on characteristics of weak entities it has a partial key and we need course primary key to be included in grade primary key.

Part 2: Relational Algebra Queries

1. π firstName, lastName (σ birthdate < 1994-05-01 ∧ income > 94000 (customer))
2. π customerID, lastName, birthdate (Customer ⋈ Owns ⋈ Account ⋈ σ budget > 2,300,000 (Branch))
3. π SIN, Employee.firstName, Employee.lastName, startDate ((PersonalBanker ⋈ Employee) ⋈ Branch)
4. π customerID, accNumber (σ type = "joint" (Account) ⋈ Owns)
5. π SIN, salary σ manager.salary < Employee.salary ∧ Employee.branchNumber = manager.BranchNumber (ρ manager ((σ Employee.Sin=Branch.managerSIN (Branch) ⋈ Employee)) X Employee)
6. π branchName(σ lastName= "Wilson" ∧ lastName= "Carson" (Branch ⋈ Employee))
7. π Customer.firstName, Customer.lastName, birthDate (Customer ⋈ Owns ⋈ Account ⋈ σ branchName= "Lonsdale" (Branch)) ∪ π Employee.firstName, Employee.lastName, startDate (Employee ⋈ σ branchName= "Lonsdale" (Branch))

8. π CustomerID, birthDate (Personal Banker ⋈ Employee ⋈ σ branchName= “Kitsilano” v branchName= “Marine” (Branch))
9. π CustomerID (Owns ⋈ Account ⋈ σ amount => 20000 v amount <= -20000 (Transaction))
10. π CustomerID, income (Account ⋈ Customer ⋈ Owns) ÷ π Type (Account)
11. π SIN, Employee.firstName, Employee.lastName (σ Employee.branchNumber = Account.branchNumber (Employee ⋈ Branch ⋈ Account))
12. {t | $\exists\ s \in Customer(s.birthDate < 1994-05-01\ \wedge s.Income > 94{,}000\ \wedge t.firstName = s.firstName \wedge t.lastName = s.lastName)$}
13. {t | $\exists\ e \in Employee\ \exists\ p \in PersonalBanker\ \exists\ b \in Branch(p.Sin = b.managerSin \wedge p.Sin = e.Sin\ \wedge b.managerSin = e.Sin\ \wedge t.firstName = e.firstName\ \wedge t.lastName = e.lastName\ \wedge\ t.startDate = e.startDate)$}