Justin Anthony Timberlake
11/22/2017
Assignment 12

**1. For D→C, CE→A, D→A, AE→ D in R(ABCDE) :**
If AE → D, D → C and D → A, then ABE → ABCDE, since one can determine all other attributes functionally with elements ABE. Similarly, BCE → ABCDE because CE → A and AE → D, and finally BDE → ABCDE because D → A and D → C. Therefore, the three keys of R(ABCDE) are ABE, BCE, and BDE.

**2. For AB→C, BC→D, CD→E, DE→A, and AE→B in R(ABCDE) :**
From the given FD's, we know :
AB → C
BC → D
Using all given FD's, we can derive :
AB → C and BC → D, so AB → D
CD → E and DE → A and AE → B, so CD → B
AB → D and BC → D, so ABC → D
BC → D and CD → A, so BCD → A
AB → D and AB → C, so ABD → C
AE → B and BC → D, so ACE → D
These are the only given or resultant functional dependencies within R(ABCDE), which will necessarily be present in S(ABCD) because they are the only ones which contain elements ABCD and not E.

**3. Of the following relations :**
**a). R(ABCD) FD's: BD → C, AB → D, AC → B, BD → A**
BD+ = AB+ = AC+ = ABCD
For each functional dependency, the closure operation of every left argument can eventually infer ABCD, so each left argument is also a key. Therefore, this set of functional dependencies is in BCNF.
**b). R(ABCD) FD's: AC → D, D → A, D → C, D → B**
AC+ = D+ = ABCD
For each functional dependency, the closure operation of every left argument can eventually infer ABCD, so each left argument is also a key. Therefore, this set of functional dependencies is in BCNF.
**c). R(ABCD) FD's: C → B, D → A, C → D, A → C**
C+ = D+ = A+ = ABCD
For each functional dependency, the closure operation of every left argument can eventually infer ABCD, so each left argument is also a key. Therefore, this set of functional dependencies is in BCNF.
**d). R(ABCD) FD's: BC → A, AD → C, CD → B, BD → C**
CD → B and BC → A, so CD → A
AD → C and CD → B, so AD → B
AD → B and AD → C, so AD → BC
BC+ = BCA
AD+ = CD+ = BD+ = ABCD
For each of the functional dependencies, all left arguments should have a closure set of ABCD and hence be keys of this relation. However, BC's closure operator only yields BCA, so D cannot be inferred from this argument and we therefore must decompose R into R1(ABC) and R2(BCD), such that for R1, BC → A with BC as the key, and for R2, CD → B with CD as the key. Since in R1, BC+ = ABC and in R2, CD+ = BCD, both R1 and R2 are in BCNF for all given funcional dependencies.

**4. For R(ABCDEF), such that AB→ CD, E → C, B → EF :**
B → EF and E → C, so B → C
AB+ = ABCDEF
E+ = CE
B+ = BCEF
With the given and derived functional dependencies, R(ABCDEF) violates BCNF because for the left arguments, E and B, both closure operations do not infer ABCDEF. However, AB+ = ABCDEF, so AB is the key. For each functional dependency, there is also no right arguments within the given or derived dependencies which are prime as none of the right arguments are subsets of the key AB.
Therefore, we need to divide R(ABCDEF) into two relations R1(BC) with B → C and R2(ABDEF) with AB → D and B → EF.
This means that for R1, we have reached a relation which now is in BCNF as the closure operation of the only key B+ = BC. However, within the functional dependencies of R2, we see that
AB+ = ABDEF
B+ = BEF
Therefore, we must further decompose R2 into two new relations R3(BEF) with B → EF and R4(ABD) with AB → D. With R3, B+ = BEF, so R3 is in BCNF. With R4, AB+ = ABD, so R4 is in BCNF as well. Any set of functional dependencies within a relation that satisfy the conditions of BCNF must also necessarily be in 3NF, therefore R1(BC), R3(BEF), and R4(ABD) are all necessarily in 3NF.

**5. SQL exercises :**
**Design and implement a database schema for employees and their dependents.**

**a. For each employee, record his/her employee id, name, age, sex, and salary**
The resulting schema would be Employee(<u>ID</u>, name, age, sex, salary)
To create such a schema as a table in PostgreSQL, one would enter the following command :
```
CREATE TABLE EMPLOYEE
( ID INT PRIMARY KEY NOT NULL,
NAME TEXT NOT NULL,
AGE INT NOT NULL,
SEX CHAR(50),
SALARY REAL );
```
Commands to add an example record would be as follows :
```
INSERT INTO EMPLOYEE VALUES (100, 'Joe', 24, 'M', 40000.00);
INSERT INTO EMPLOYEE VALUES (101, 'Cathy', 26, 'F', 51948.05);
INSERT INTO EMPLOYEE VALUES (102, 'Kyle', 29, 'M', 45000.12);
INSERT INTO EMPLOYEE VALUES (103, 'Jodie', 23, 'F', 39000.78);
```
To display all entries of the table, use SELECT * FROM EMPLOYEE;

**b. For each dependent of an employee, record his/her name, age, and sex** :
The resulting schema would be Dependent(<u>dependentOf</u>, <u>name</u>, age, sex), where dependentOf corresponds to the ID of an employee. Both the dependentOf and name attributes act as the unique identifier of a given dependent, assuming each dependent will have a different name for every set of dependents associated with each employee. To create such a schema as a table in PostgreSQL, one would enter the following command :

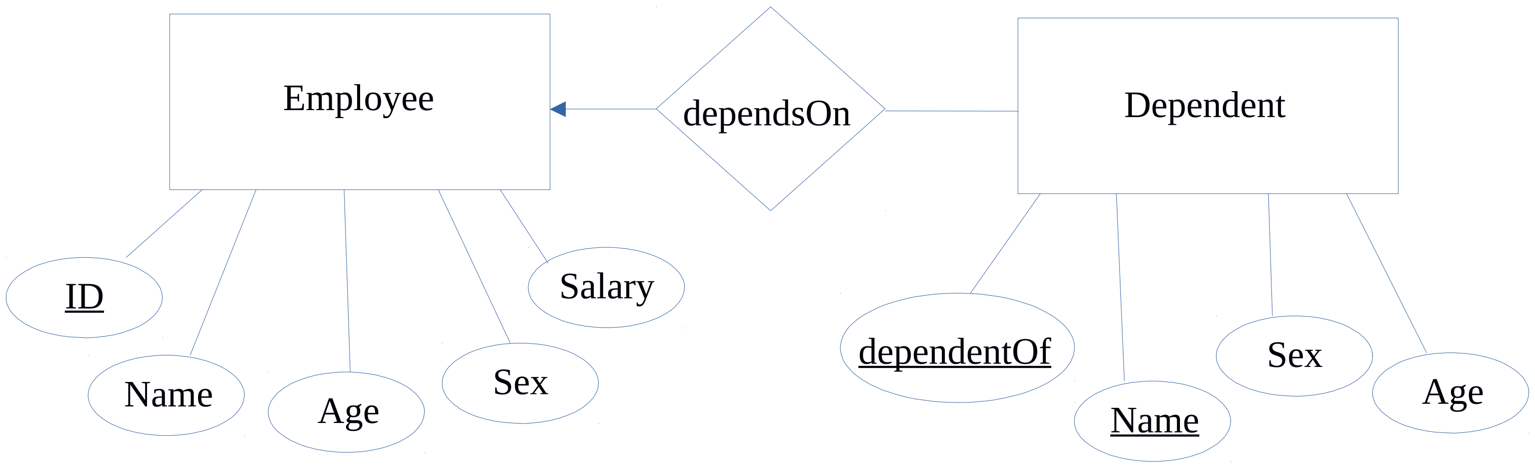
```
CREATE TABLE DEPENDENT
( DEPENDENTOF INT NOT NULL,
 NAME TEXT NOT NULL,
 AGE INT NOT NULL,
 SEX CHAR(50),
 PRIMARY KEY (DEPENDENTOF, NAME) );
```

Commands to add an example record would be as follows :

```
INSERT INTO DEPENDENT VALUES(100, 'Sally', 4, 'F');
INSERT INTO DEPENDENT VALUES(100, 'Joey', 2, 'M');
INSERT INTO DEPENDENT VALUES(101, 'Tyler', 6, 'M');
 INSERT INTO DEPENDENT VALUES(103, 'Danielle', 1, 'F');
```

To display all entries of the table, use SELECT * FROM DEPENDENT;

**c. Draw an E/R diagram, convert it into relational schema, and implement it with PostgreSQL :**



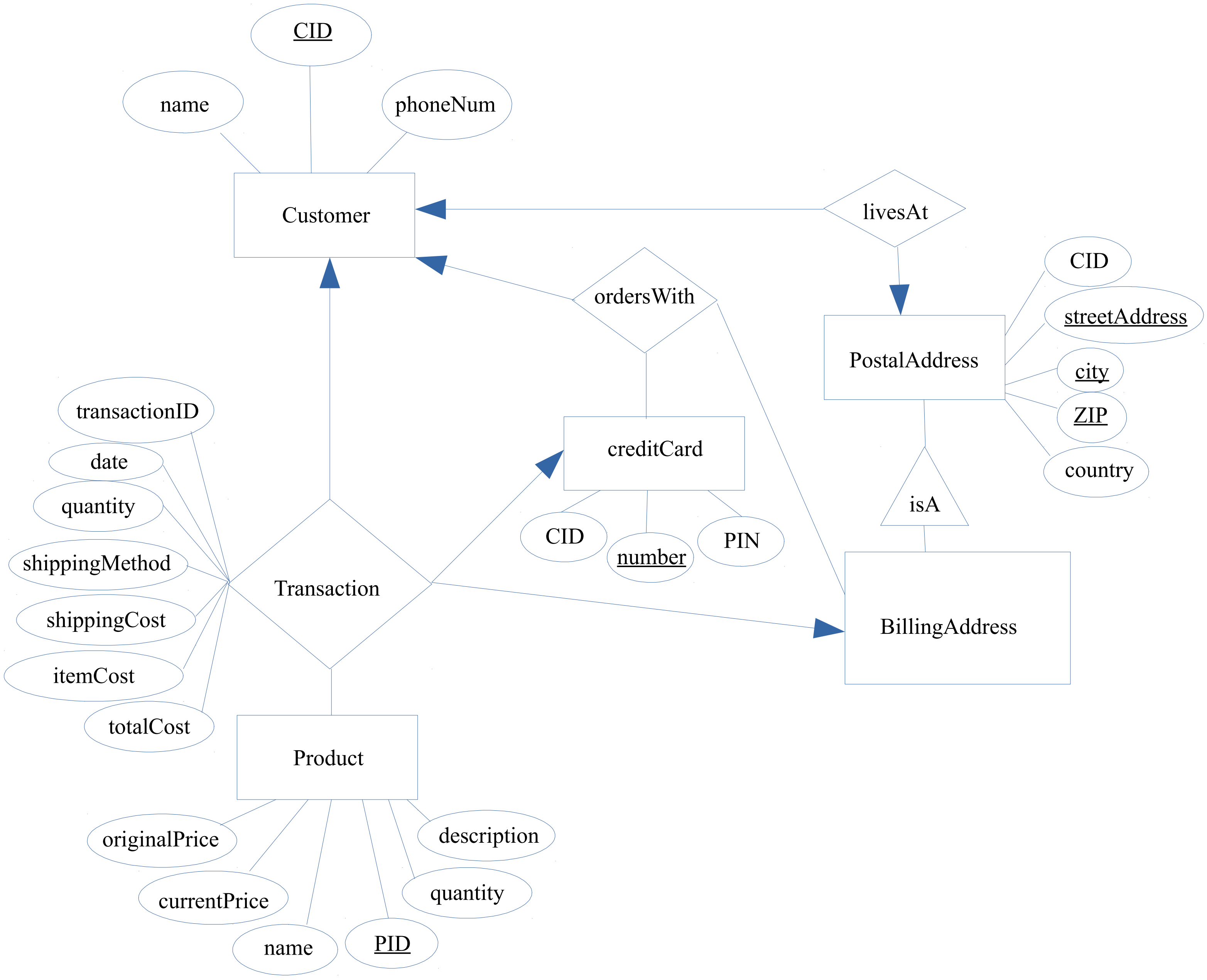
The resultant relational schema dependsOn(<u>Provider_ID</u>, Provider_name, <u>Name</u>, Sex, Age) can be represented in PostGreSQL by the following command :

```
SELECT E.ID AS PROVIDER_ID, E.NAME AS PROVIDER_NAME, D.NAME, D.SEX, D.AGE
FROM EMPLOYEE E JOIN DEPENDENT D
ON E.ID = D.DEPENDENTOF;
```

This will show the ID’s and names of the employees who each dependent depends on as well as each dependent’s attributes, only for those employees which have dependents. In the example of the tuples I provided, Kyle (ID 102) would only show up in the table with provider ID and name and null values if this query were changed to a LEFT OUTER JOIN. Every Dependent has exactly one provider. Each tuple is uniquely identified by the provider’s ID and the dependent’s name.

**d. Each relation/table must have a primary key.**
They do.
**e. Create at least one single-attribute index and one multi-attribute index.**
An index for all lowercase spellings of Employee names :
```
CREATE INDEX lowercase_names ON EMPLOYEE((lower(name)));
```
An index for all the names and ages of dependents :
```
CREATE INDEX name_age ON DEPENDENT(NAME, AGE);
```
**f. Implement referential integrity constraints (if there is any) using foreign keys in PostgreSQL.**
To add a foreign key constraint that disallows adding any dependents whose dependentOf attribute cannot be matched with some Employee ID already existing within the Employee table, execute the following comand :
```
ALTER TABLE Dependent
ADD CONSTRAINT fk_ID
FOREIGN KEY (DEPENDENTOF)
REFERENCES EMPLOYEE;
```
Therefore, when trying to add another record to Dependent with an invalid Employee ID number, an error message will appear as follows :
```
INSERT INTO DEPENDENT VALUES (105, 'Jenna', 9, 'F');
ERROR:  insert or update on table "dependent" violates foreign key constraint "fk_id"
DETAIL:  Key (dependentof)=(105) is not present in table "employee".
```

**BONUS :**



*Any given ZIP code is unique for a specific area in the world, so the key for PostalAddress need only include streetAddress, city, and ZIP and the country can be functionally determined ZIP → country.
**While exactly one creditCard and BillingAddress is used with every transaction, the online database still stores multiple instances of each with exactly one Customer with his/her CID. Exactly one customer lives at exactly one PostalAddress, however.
***The list of items for each transaction is determined by the transactionID associated with any number of prdoucts being purchased.

**I, Justin Anthony Timberlake, declare that I have completed this assignment completely and entirely on my own, without any consultation with others. I understand that any breach of the UAB Academic Honor Code may result in severe penalties.**