Ceng352 - Database Management Systems Written Assignment 1

Furkan Doğan / 2098937

**1-)**

**a-)**

CREATE TABLE Employee (

emp\_id INT NOT NULL,

name VARCHAR(128),

surname VARCHAR(128),

salary INT,

gender VARCHAR(128),

PRIMARY KEY (emp\_id));

CREATE TABLE Department (

dept\_id INT NOT NULL,

manager\_id INT DEFAULT 101,

name VARCHAR(128),

LOCATION VARCHAR(128),

PRIMARY KEY (dept\_id),

FOREIGN KEY (manager\_id) REFERENCES employee(emp\_id) ON DELETE SET DEFAULT

);

CREATE TABLE employee\_department (

dept\_id INT,

employee\_id INT,

FOREIGN KEY (dept\_id) REFERENCES department (dept\_id),

FOREIGN KEY (employee\_id) REFERENCES employee (emp\_id) ON DELETE CASCADE

);

CREATE TABLE employee\_report (

subordinate\_id INT,

supervisor\_id INT,

FOREIGN KEY (subordinate\_id) REFERENCES employee (emp\_id),

FOREIGN KEY (supervisor\_id) REFERENCES employee (emp\_id)

);

CREATE TABLE project (

project\_id INT NOT NULL,

dept\_id INT NOT NULL,

budget INT,

state VARCHAR(128),

due\_date TIMESTAMP,

PRIMARY KEY (project\_id),

FOREIGN KEY (dept\_id) REFERENCES department (dept\_id)

);

**b-)**

CREATE ASSERTION Total CHECK (

EXISTS (

SELECT

e.emp\_id

FROM

employee e,

employee\_department d

WHERE

e.emp\_id = d.employee\_id));

**c-)**

alter table employee add constraint salaryCheck check(salary > 36000);

alter table department add constraint nameCheck check("name" LIKE '%' || "location" || '%');

**d-)**

CREATE TRIGGER BudgetProject

AFTER UPDATE budget ON project

FOR EACH ROW

WHEN (OLD.budget > budget)

UPDATE project

SET "state" = Unsuccessful

WHERE project\_id = OLD.project\_id

I didn’t refernes as old and new because postgresql is doing it by itself by referencing the old as OLD.

**2-)**

**a-)**

Since given store and given product have relation with only one store, I thought that there could be 5\*100 different given product and store so the answer is 500.

**b-)**

Since given customer and product could have relation with only specific store and sale person there could be 990\*100 different given product and customer so the answer is 99000.

**3-)**

**a-)**

1. CB 🡪 F (Given)
2. CB 🡪 B (Trivial)
3. B 🡪 E (Given)
4. CB 🡪 E (Transition on 2,3)
5. CB 🡪 FE (Combining 1,4)
6. FE 🡪 G (Given)
7. CB 🡪 G (Transition on 5,6)

**b-)**

1. AB 🡪 A (Trivial)
2. A 🡪 C (Given)
3. AB 🡪 B (Trivial)
4. AB 🡪 C (Transition on 1,2)
5. AB 🡪 CB (Combining 3,4)
6. CB 🡪 F (Given)
7. CB 🡪 B (Trivial)
8. B 🡪 E (Given)
9. CB 🡪 E (Transition on 7,8)

10-) CB 🡪 EF (Combining 6,9)

11-) AB 🡪 EF (Transition on 5,10)

**4-)**

**a-)**

{A}+ = {A, B} {B}+ = {B} {C}+ = {C} {D}+ = {C, D} {E}+ = {E, G}

{F}+ = {D, F} {G}+ = {G} {A}+ = {A,B} {A,C}+ = {A, B, C} {A, D}+ = {A, B, C, D} {A, E}+ = {A, B , E, G} {A, F}+ = {A, B, C, D, E, F} {B, C}+ = {B , C} {B, D}+ = {B, D, C} {B, E}+ = {B, E, G} {B, F}+ = {B , C, D, E, F} } {B, G}+ = {B , G} …

After writing all closure of every subset we can see that minimal key is {A, F} since we can generate all letters from these two.

**b-)**

A->B: The set closure of A+ is { A,B} therefore the left hand side is not a superkey. BCNF Violation

CD->E: The set closure of CD+ is { C, D, E, G } therefore the left hand side is not a superkey. BCNF Violation

F-> D: The set closure of F+ is { C, D, F } therefore the left hand side is not a superkey. BCNF Violation

E->G: The set closure of E+ is { E, G } therefore the left hand side is not a superkey. BCNF Violation

AC->D: The set closure of AC+ is { A, B, C, D, E, G } therefore the left hand side is not a superkey. BCNF Violation

D-> C: The set closure of D+ is { C, D } therefore the left hand side is not a superkey. BCNF Violation

So since there are BCNF violation R is not in BCNF.

**c-)**

A close up of text on a white background

Description automatically generated

**d-)**

**i-)** It is not dependency preserving since we lost AC 🡪 D dependency.

**ii-)** BCNF decomposition is always lossless**.**

**5-)**

**a-)**

A 🡪 E

A, B 🡪 C, E

A, B, C 🡪 E

A, B, C, D 🡪 E (Complete)

A, B, D 🡪 C, E (Complete)

A, B, D, E 🡪 C (Complete)

A, B, E 🡪 C

A, C 🡪 B, E

A, C, D 🡪 B, E (Complete)

A, C, D, E 🡪 B (Complete)

A, C, E 🡪 B

A, D 🡪 E

B, C 🡪 A, E

B, C, D 🡪 A, E (Complete)

B, C, D, E 🡪 A (Complete)

B, C, E 🡪 A

B, D, E 🡪 A, C (Complete)

B, E 🡪 A, C

C 🡪 A, B, E

C, D 🡪 A, B, E (Complete)

C, D, E 🡪 A, B (Complete)

C, E 🡪 A, B

D, E 🡪 A

E 🡪 A

**b-)**

CREATE TABLE AE (

A VARCHAR(256) PRIMARY KEY,

E VARCHAR(256)

);

CREATE TABLE CA (

C VARCHAR(256) PRIMARY KEY,

A VARCHAR(256),

FOREIGN KEY (A) REFERENCES AE (A)

);

CREATE TABLE CB (

C VARCHAR(256),

B VARCHAR(256),

FOREIGN KEY (C) REFERENCES CA (C)

);

CREATE TABLE CD (

C VARCHAR(256),

D VARCHAR(256),

FOREIGN KEY (C) REFERENCES CA (C)

);

**c-)**

INSERT INTO ae(a,e)SELECT t."A",t."E" FROM test t GROUP BY t."A",t."E";

INSERT INTO ca(c,a)SELECT t."C",t."A" FROM test t GROUP BY t."A",t."C";

INSERT INTO cb(c,b)SELECT t."C",t."B" FROM test t GROUP BY t."B",t."C";

INSERT INTO cd(c,d)SELECT t."C",t."D" FROM test t GROUP BY t."C",t."D";