DBMS Project Report

PES University

Database Management Systems

UE18CS252

Submitted By

|  |  |  |
| --- | --- | --- |
| **SRN :**  **PES1201801471** | **Name** :  **Krupakar G M** | **Evaluation date and time:** |

**Functional Dependencies 2**

**Identifying Keys based on FDs 2**

[**Normalization**](#_wc0v2611q54c) **& testing for lossless join property 2+2**

[**DDL**](#_pn441yprtk2f)**: Table creation with all constraints 2+2**

[**Triggers**](#_a9bswny9ffj) **2**

[**SQL Queries**](#_exvna2br67rz) **2**

**Viva / modifications (Unit III/ IV concepts) 2+2**

College Database Management System

|  |
| --- |
| **Problem statement:**  Design a College Management System Database to capture the Information about Student ( identified by name, ID, DOB, Address, date of Joining), Course (identified by ID, Name),  Lecturer (identified by Lecturer\_ID, Name, Course\_ID) and Subjects (identified by Lecturer\_ID, Subject\_ID, Subject\_Name ).  Keeping the appropriate Constraints intact such as assuring the Date of Joining is after the DOB, assign the subjects to every teacher. |

# **Database Schema: (show all the tables and the constraints)**

college=# \d

List of relations

Schema | Name | Type | Owner

--------+----------+-------+----------

public | course | table | postgres

public | lecturer | table | postgres

public | student | table | postgres

public | subjects | table | postgres

(4 rows)

college=# \d student;

Table "public.student"

Column | Type | Collation | Nullable | Default

---------------------+-----------------------+-----------+----------+---------

student\_id | integer | | not null |

fname | character varying(40) | | not null |

lname | character varying(40) | | not null |

address | character varying(40) | | not null |

dob | date | | not null |

college\_joined\_date | date | | |

course\_id | character varying(40) | | |

Indexes:

"student\_pkey" PRIMARY KEY, btree (student\_id)

Check constraints:

"student\_check" CHECK (college\_joined\_date > dob)

Foreign-key constraints:

"student\_course\_id\_fkey" FOREIGN KEY (course\_id) REFERENCES course(course\_id)

college=# \d course;

Table "public.course"

Column | Type | Collation | Nullable | Default

-------------+-----------------------+-----------+----------+---------

course\_id | character varying(40) | | not null |

course\_name | character varying(40) | | |

Indexes:

"course\_pkey" PRIMARY KEY, btree (course\_id)

Referenced by:

TABLE "lecturer" CONSTRAINT "lecturer\_course\_id\_fkey" FOREIGN KEY (course\_id) REFERENCES course(course\_id) ON DELETE CASCADE

TABLE "student" CONSTRAINT "student\_course\_id\_fkey" FOREIGN KEY (course\_id) REFERENCES course(course\_id)

college=# \d subjects;

Table "public.subjects"

Column | Type | Collation | Nullable | Default

--------------+-----------------------+-----------+----------+---------

subject\_id | character varying(40) | | not null |

subject\_name | character varying(40) | | not null |

lecturer\_id | integer | | |

Indexes:

"subjects\_pkey" PRIMARY KEY, btree (subject\_id)

Foreign-key constraints:

"subjects\_lecturer\_id\_fkey" FOREIGN KEY (lecturer\_id) REFERENCES lecturer(lecturer\_id)

college=# \d lecturer;

Table "public.lecturer"

Column | Type | Collation | Nullable | Default

---------------+-----------------------+-----------+----------+---------

lecturer\_id | integer | | not null |

lecturer\_name | character varying(40) | | |

course\_id | character varying(40) | | not null |

Indexes:

"lecturer\_pkey" PRIMARY KEY, btree (lecturer\_id)

Foreign-key constraints:

"lecturer\_course\_id\_fkey" FOREIGN KEY (course\_id) REFERENCES course(course\_id) ON DELETE CASCADE

Referenced by:

TABLE "subjects" CONSTRAINT "subjects\_lecturer\_id\_fkey" FOREIGN KEY (lecturer\_id) REFERENCES lecturer(lecturer\_id)

# **Functional Dependencies: (List based on your application constraints)**

**Student Table**

1. student\_id -> fname, course\_id, lname, address, dob, college\_joined\_date
2. Dob -> fname, student\_id

**Course Table**

1. Course\_id -> course\_name
2. Course\_name -> course\_id

**Subjects Table**

1. Subject\_id -> subject\_name
2. Subject\_name -> lecturer\_id
3. Lecturer\_id -> subject\_id

**Lecturer Table**

1. Lecturer\_id -> lecturer\_name, course\_id

# **Candidate keys: (Justify how did you get these as keys)**

**Student Table =>** student\_id , dob

**Course Table =>** course\_id , course\_name

**Subjects Table** **=>** subject\_id, subject\_name, lecturer\_id

**Lecturer Table =>** lecturer\_id

# **Normalization and testing for lossless join property:**

<You will get a normalized set of relations if you develop using ER Model and converting ER to schema approach.

Still discuss when a 2nd normal form may be violated.

For example, In company schema-if you add employee name to works-on relationship table, 2NF is violated because ssn->name but the key is ssn, project\_number

2NF is also violated if you add a project name.

The functional dependency is in 1NF if and only if the domain of **each attribute contains only atomic values** and the value of each attribute contains only a single value from that domain.

2NF form will be violated if we have a partial dependency.

Partial dependency occurs when we can derive a non-prime attribute from a prime attribute.

**CONDITION FOR 2NF**

* Table should be in 1NF (First normal form)
* No non-prime attribute is dependent on the proper subset of any candidate key of table.

3NF form will be violated if we have a transitive dependency.

Transitive dependency occurs when we can derive a non-prime attribute from a non-prime

Attribute.

**CONDITION FOR 3NF**

* Table must be in 2NF
* [Transitive functional dependency](https://beginnersbook.com/2015/04/transitive-dependency-in-dbms/) of non-prime attribute on any super key should be removed.

If it doesn’t satisfy the 2NF condition then no need to check this condition since if it

is not in 2NF then it cannot be in 3NF.

BCNF form: It’s a Stronger Version of 3NF

**CONDITION FOR BCNF**

* It Should be in 3NF
* For each Non-Trivial Functional Dependency (X -> Y), X must be Super Key.

As we have understood the different forms, let’s implement it on our Database

**Student Table**

It’s 2NF as Prime Attribute (student\_id) is derived from Prime Attribute(dob).

It’s 3NF as Non Prime Attribute (address) is derived from Prime Attribute(dob).

It’s BCNF (X -> Y) as X is Candidate Key I,e Student\_id and dob.

* Therefore, Student Table in **BCNF**

**Course Table**

It’s 2NF as Prime Attribute (course\_name) is derived from Prime Attribute(course\_id).

It’s 3NF as Prime Attribute (course\_id) is derived from Prime Attribute(course\_name).

It’s BCNF (X -> Y) as X is Candidate Key I,e course\_id and course\_name.

* Therefore, Course Table in **BCNF**

**Subjects Table**

It’s 2NF as Prime Attribute (lecturer\_id) is derived from Prime Attribute(subject\_name).

It’s 3NF as Prime Attribute (subject\_name) is derived from Prime Attribute(subject\_id).

It’s BCNF (X -> Y) as X is Candidate Key I,e subject\_id , subject\_name and lecturer\_d.

* Therefore, Subjects Table in **BCNF**

**Lecturer Table**

It’s not 2NF as Non Prime Attribute (couse\_id, lecturer\_name) is derived from Prime Attribute(lecturer\_id).

* Therefore, Lecturer Table in **1NF** only.

**To check for lossless join decomposition using FD set, following conditions must**

**hold:**

1. Union of Attributes of R1 and R2 must be equal to attribute of R. Each attribute of R

must be either in R1 or in R2.

Att(R1) U Att(R2) = Att(R)

2. Intersection of Attributes of R1 and R2 must not be NULL.

Att(R1) ∩ Att(R2) ≠ Φ

3. Common attribute must be a key for at least one relation (R1 or R2)

Att(R1) ∩ Att(R2) -> Att(R1) or Att(R1) ∩ Att(R2) -> Att(R2)

**TESTING LOSSLESS JOIN PROPERTY FOR EVERY TABLE**

**Student Table**

R1 ( fname, lname, college\_joined\_date, **student\_id**) R2 (**student\_id**, dob, address, course\_id)

The Above Decomposed Relations **obey** Lossless Join Property as all the above 3 Conditions are met.

**Course Table**

This Table **doesn’t obey** Lossless Join Property as the first condition fails

i.e Att(R1) U Att(R2) = Att(R)

**Subjects Table**

R1 ( Subject\_name, **subject\_id**) R2 (**subject\_id**, subject\_name)

The Above Decomposed Relations **obey** Lossless Join Property as all the above 3 Conditions are met.

**Lecturer Table**

R1 ( lecturer\_name, **lecturer\_id**) R2 (**lecturer\_id**, course\_id)

The Above Decomposed Relations **obey** Lossless Join Property as all the above 3 Conditions are met.

**DDL:**

Create table scripts here. Ensure integrity constraints are defined.

# Add sample insert statements as well, that you would be using for demo.

# CREATE DATABASE college;

\c college;

CREATE TABLE Student(

student\_id int NOT NULL PRIMARY KEY,

fname varchar(40) NOT NULL,

lname varchar(40) NOT NULL,

Address varchar(40) NOT NULL,

DOB date NOT NULL,

college\_joined\_date DATE CHECK ( college\_joined\_date > DOB)

);

CREATE TABLE Course(

course\_id varchar(40) NOT NULL PRIMARY KEY,

course\_name varchar(40)

);

ALTER TABLE student ADD COLUMN course\_id varchar(40) references course(course\_id);

CREATE TABLE Lecturer(

lecturer\_id int NOT NULL PRIMARY KEY,

lecturer\_name varchar(40),

course\_id varchar(40) NOT NULL

);

CREATE TABLE Subjects(

subject\_id varchar(40) NOT NULL PRIMARY KEY,

subject\_name varchar(40) NOT NULL,

lecturer\_id int references lecturer(lecturer\_id)

);

ALTER TABLE lecturer ADD FOREIGN KEY (course\_id) references course(course\_id) ON DELETE CASCADE;

# **Triggers:**

1. Identify a constraint to implement as a trigger and write the English statement for that.

For Storing the Info if any Modification in Address or Branch of the Student

1. Write the trigger creation statement along with any stored procedures/functions involved.

CREATE TABLE tracking\_table(  
 old\_student\_id int NOT NULL,

Old\_course\_id varchar(40) NOT NULL,

New\_course\_id varchar(40) NOT NULL,  
 Old\_ address varchar(40) NOT NULL,

New\_address varchar(40) NOT NULL,  
 modified\_time TIMESTAMP

);

CREATE TRIGGER tracker

BEFORE UPDATE ON Student

FOR EACH ROW

insert into tracking\_table VALUES(OLD.student\_id, OLD.course-id, NEW.course\_id, OLD.address, NEW.address, NOW() );

**SQL Queries:**

<Write a few English sentences and SQL queries for them. Ensure at least 2 correlated-nested Advanced and 2 aggregate queries. >

1. Write a Query to display the names of those who have taken Computer Science and belong to Bangalore.

SELECT fname AS name , course\_name, Address

FROM student NATURAL JOIN course

WHERE course\_name = ‘Computer Science’ AND Address = ‘Bangalore’

1. Write a Query to display the Age of all the students of Mechanical and Computer science.

SELECT fname as NAME, AGE ( NOW() , DOB ) AS Age

FROM student;

1. Write a Query to find the total number of Lecturers in every Branch.

SELECT

SUM (course\_id) AS Count, Course\_ID

FROM lecturer

GROUP BY course\_id;

1. Write a Query to display to count the Total number of Students in the College.

SELECT COUNT (fname) as TOTAL

FROM student;