Homework Assignment 7

1. E - Both a and b are correct.
2. D – rollback
3. A - INSERT INTO
4. C - DELETE FROM homework10
5. A - SELECT * FROM homework10 WHERE col2 IS NULL FOR UPDATE
6. B - INSERT INTO homework10 (col3, col1, col2) VALUES (NULL, ‘A’, ‘B’);
7. A - &
8. B - ‘O’ ’hara’ (two single quotes following the O)
9. B - LOCK TABLE homework10 IN EXCLUSIVE MODE;
10. D - None of the above would cause the command to fail.
11. E - all of the above
12. C - Other users can view the new orders as soon as you exit the system or execute a COMMIT command.
13. A - DELETE FROM orders WHERE orderdate < ‘01-APR-09’;
14. A – 1
15. D - None of the above restores the deleted orders.

Advance Challenge

Currently, the contents of the Category column in the BOOKS table are the actual name for each category. This structure presents a problem if one user enters COMPUTER for the Computer category and another user enters COMPUTERS. To avoid this and other problems that might occur, the database designers have decided to create a CATEGORY table containing a code and description for each category. The structure for the CATEGORY table should be as follows

Column Name	Datatype	Width	Constraints
CATCODE	VARCHAR2	3	PRIMARY KEY
CATDESC	VARCHAR2	11	NOT NULL

The data for the CATEGORY table is as follows:

CATCODE	CATDESC
BUS	BUSINESS
CHN	CHILDREN
COK	COOKING
COM	COMPUTER
FAL	FAMILY LIFE
FIT	FITNESS
SEH	SELF HELP
LIT	LITERATURE

Required:

- Create the CATEGORY table and populate it with the given data. Save the changes permanently.
- Add a column to the BOOKS table called Catcode.
- Add a FOREIGN KEY constraint that requires all category codes entered in the BOOKS table to already exist in the CATEGORY table. Set the Catcode values for the existing rows in the BOOKS table, based on each book's current Category value.
- Verify that the correct categories have been assigned in the BOOKS table, and save the changes permanently.
- Delete the Category column from the BOOKS table.

SOLUTION:-

Step 1: Create the CATEGORY Table

Create the CATEGORY table with CATCODE as the primary key and CATDESC as a NOT NULL column.

```
CREATE TABLE CATEGORY (
   CATCODE VARCHAR2(3) PRIMARY KEY,
   CATDESC VARCHAR2(11) NOT NULL
);
```

Step 2: Populate the CATEGORY Table with Given Data

Insert the provided data into the CATEGORY table.

```
INSERT INTO CATEGORY (CATCODE, CATDESC) VALUES ('BUS', 'BUSINESS');
INSERT INTO CATEGORY (CATCODE, CATDESC) VALUES ('CHN', 'CHILDREN');
INSERT INTO CATEGORY (CATCODE, CATDESC) VALUES ('COK', 'COOKING');
INSERT INTO CATEGORY (CATCODE, CATDESC) VALUES ('COM', 'COMPUTER');
INSERT INTO CATEGORY (CATCODE, CATDESC) VALUES ('FAL', 'FAMILY LIFE');
INSERT INTO CATEGORY (CATCODE, CATDESC) VALUES ('FIT', 'FITNESS');
INSERT INTO CATEGORY (CATCODE, CATDESC) VALUES ('SEH', 'SELF HELP');
INSERT INTO CATEGORY (CATCODE, CATDESC) VALUES ('LIT', 'LITERATURE');
```

Step 3: Add Catcode Column to the BOOKS Table

Add a new column called Catcode to the BOOKS table to store the category code for each book.

```
ALTER TABLE BOOKS
ADD Catcode VARCHAR2(3);
```

Step 4: Update Catcode in BOOKS Table Based on Category Column

Map the existing Category values in the BOOKS table to the appropriate Catcode values. For example:

```
UPDATE BOOKS SET Catcode = 'BUS' WHERE Category = 'BUSINESS';

UPDATE BOOKS SET Catcode = 'CHN' WHERE Category = 'CHILDREN';

UPDATE BOOKS SET Catcode = 'COK' WHERE Category = 'COOKING';

UPDATE BOOKS SET Catcode = 'COM' WHERE Category = 'COMPUTER';

UPDATE BOOKS SET Catcode = 'FAL' WHERE Category = 'FAMILY LIFE';

UPDATE BOOKS SET Catcode = 'FIT' WHERE Category = 'FITNESS';

UPDATE BOOKS SET Catcode = 'SEH' WHERE Category = 'SELF HELP';

UPDATE BOOKS SET Catcode = 'LIT' WHERE Category = 'LITERATURE';
```

Step 5: Add Foreign Key Constraint

Ensure that the Catcode values in the BOOKS table reference valid CATCODE values in the CATEGORY table.

```
ALTER TABLE BOOKS

ADD CONSTRAINT fk_books_category

FOREIGN KEY (Catcode) REFERENCES CATEGORY(CATCODE);
```

Step 6: Verify the Updates (Optional)

Run a query to verify that the correct Catcode values have been assigned.

```
SELECT * FROM BOOKS;
```

Step 7: Delete the Category Column from BOOKS Table

Once verification is complete, remove the old Category column from the BOOKS table.

```
ALTER TABLE BOOKS

DROP COLUMN Category;
```

1. Question **14.5. What is a functional dependency? What are the possible sources of the information that defines the functional dependencies that hold among the attributes of a relation schema?**

Solution:-

A **functional dependency** in database management is a constraint that specifies a relationship between two sets of attributes in a relation. If attribute XXX determines attribute YYY (denoted as X→YX \rightarrow YX→Y), it means that each unique value of XXX is associated with precisely one value of YYY in that relation. This concept is central to normalization, which organizes databases to minimize redundancy and dependency issues .

Sources of information for functional dependencies in a relation schema include:

1. **Business Rules**: These are constraints defined by the organization's operational practices. For instance, in an employee database, it may be a business rule that each employee ID uniquely identifies an employee name.
2. **Data Analysis**: Reviewing existing data can reveal patterns indicating dependencies, such as how certain attributes correlate or consistently match others.
3. **Domain Knowledge**: Understanding the context or domain, such as industry standards, can highlight which attributes should be dependent on others.

These sources guide database designers in identifying and implementing functional dependencies to maintain data integrity and support efficient query processing.

2. Question 14. 6: **Why can we not infer a functional dependency automatically from a particular relation state?**

Solution:-

We cannot automatically infer a functional dependency from a specific relation state because a functional dependency represents a general constraint on the data that should hold across all possible states of a relation, not just a single instance. Here's why relying on one state is insufficient:

1. **Dependence on Data Population**: A functional dependency is intended to express a rule about how attributes relate to one another in the entire schema, regardless of the specific data present at any given time. Observing one particular state of data

may show patterns that appear coincidental or temporary rather than universally true. For example, in one instance, each employee may have a unique office number, but this may change as the organization grows, invalidating the dependency.

2. **Possibility of Coincidence**: The values in one specific relation state might coincidentally satisfy what appears to be a dependency. For instance, if a subset of data shows each employee in a department has the same manager, this could imply a dependency (e.g., department → manager). However, without further data or domain rules, we can't be sure this dependency will hold for all states.
3. **Absence of Generality**: Functional dependencies are meant to capture essential relationships dictated by business logic or domain rules, not transient patterns in one data snapshot. A dependency inferred from a single relation state may not represent the true business logic that should govern the data over time.
4. **Data Changes Over Time**: As more data is added to the database, initial patterns that appeared in one relation state might not persist, and the inferred dependency may no longer hold.

Therefore, determining functional dependencies typically requires examining data across various states, considering business rules, and leveraging domain knowledge rather than relying on just one instance of the data.

3. Question 14.19: Suppose we have the following requirements for a university database that is used to keep track of students' transcripts:

i. The university keeps track of each student's name (SNAME), student number (SNUM), social security number (SSSN), current address (SCADDR) and phone (SCPHONE), permanent address (SPADDR) and phone (SPPHONE), birthdate (BDATE), sex (SEX), class (CLASS) (freshman, sophomore, ..., graduate), major department (MAJORDEPTCODE), minor department (MINORDEPTCODE) (if any), and degree program (PROG) (B.A., B.S., ..., Ph.D.). Both ssn and student number have unique values for each student.

ii. Each department is described by a name (DEPTNAME), department code (DEPTCODE), office number (DEPTOFFICE), office phone (DEPTPHONE), and college (DEPTCOLLEGE). Both name and code have unique values for each department.

iii. Each course has a course name (CNAME), description (CDESC), code number (CNUM), number of semester hours (CREDIT), level (LEVEL), and offering department (CDEPT). The value of code number is unique for each course.

iv. Each section has an instructor (INSTUCTORNAME), semester (SEMESTER), year (YEAR), course (SECCOURSE), and section number (SECNUM). Section numbers distinguish different sections of the same course that are taught during the same semester/year; its values are 1, 2, 3, ...; up to the number of sections taught during each semester.

v. A grade record refers to a student (Ssn), refers to a particular section, and grade (GRADE).

Design an relational database schema for this database application. First show all the functional dependencies that should hold among the attributes. Then, design relation schemas for the database that are each in 3NF or BCNF. Specify the key attributes of each relation. Note any unspecified requirements, and make appropriate assumptions to make the specification complete.

Solution:-

To design a relational database schema for the university database, we'll break down the requirements and determine the functional dependencies (FDs) for each entity. Based on these FDs, we'll create relation schemas in Third Normal Form (3NF) or Boyce-Codd Normal Form (BCNF).

Step 1: Define Functional Dependencies (FDs)

We'll go through each entity and determine the FDs based on unique identifiers and relationships described in the requirements.

1. Student Entity

Attributes:

- **SNAME, SNUM, SSSN, SCADDR, SCPHONE, SPADDR, SPPHONE, BDATE, SEX, CLASS, MAJORDEPTCODE, MINORDEPTCODE, PROG**

Functional Dependencies:

1. **SSSN →** all other student attributes (since SSSN uniquely identifies each student)
2. **SNUM →** all other student attributes (since SNUM also uniquely identifies each student)

Keys: **SSSN** and **SNUM**

2. Department Entity

Attributes:

- **DEPTNAME, DEPTCODE, DEPTOFFICE, DEPTPHONE, DEPTCOLLEGE**

Functional Dependencies:

1. **DEPTCODE →** DEPTNAME, DEPTOFFICE, DEPTPHONE, DEPTCOLLEGE
2. **DEPTNAME →** DEPTCODE, DEPTOFFICE, DEPTPHONE, DEPTCOLLEGE

Keys: **DEPTCODE** and **DEPTNAME**

3. Course Entity

Attributes:

- **CNAME, CDESC, CNUM, CREDIT, LEVEL, CDEPT**

Functional Dependencies:

1. **CNUM →** CNAME, CDESC, CREDIT, LEVEL, CDEPT

Key: **CNUM**

4. Section Entity

Attributes:

- **INSTRUCTORNAME, SEMESTER, YEAR, SECCOURSE, SECNUM**

Functional Dependencies:

1. **(SECCOURSE, SEMESTER, YEAR, SECNUM) →** INSTRUCTORNAME

Key: **(SECCOURSE, SEMESTER, YEAR, SECNUM)**

5. Grade Record Entity

Attributes:

- **SSN (student), SECCOURSE, SEMESTER, YEAR, SECNUM, GRADE**

Functional Dependencies:

1. **(SSN, SECCOURSE, SEMESTER, YEAR, SECNUM) →** GRADE

Key: **(SSN, SECCOURSE, SEMESTER, YEAR, SECNUM)**

Step 2: Design Relation Schemas

We will design the relation schemas based on the FDs we have identified and ensure they satisfy 3NF or BCNF.

1. Student Relation

Attributes:

- **SNUM (PK), SSSN (Unique), SNAME, SCADDR, SCPHONE, SPADDR, SPPHONE, BDATE, SEX, CLASS, MAJORDEPTCODE, MINORDEPTCODE, PROG**

Since **SNUM** and **SSSN** both uniquely identify students, we can make **SNUM** the primary key and **SSSN** a unique constraint.

Schema:

Student(SNUM (PK), SSSN (Unique), SNAME, SCADDR, SCPHONE, SPADDR, SPPHONE, BDATE, SEX, CLASS, MAJORDEPTCODE, MINORDEPTCODE, PROG)

2. Department Relation

Attributes:

- **DEPTCODE (PK), DEPTNAME (Unique), DEPTOFFICE, DEPTPHONE, DEPTCOLLEGE**

DEPTCODE and **DEPTNAME** are both unique, so **DEPTCODE** can be the primary key and **DEPTNAME** a unique constraint.

Schema:

Department(DEPTCODE (PK), DEPTNAME (Unique), DEPTOFFICE, DEPTPHONE, DEPTCOLLEGE)

3. Course Relation

Attributes:

- **CNUM (PK), CNAME, CDESC, CREDIT, LEVEL, CDEPT**

Since **CNUM** uniquely identifies each course, it is the primary key.

Schema:

Course(CNUM (PK), CNAME, CDESC, CREDIT, LEVEL, CDEPT (FK references Department(DEPTCODE)))

4. Section Relation

Attributes:

- **SECCOURSE (FK references Course(CNUM)), SEMESTER, YEAR, SECNUM, INSTRUCTORNAME**

Since each section is uniquely identified by **(SECCOURSE, SEMESTER, YEAR, SECNUM)**, this composite attribute set is the primary key.

Schema:

Section(SECCOURSE (FK), SEMESTER, YEAR, SECNUM, INSTRUCTORNAME, (PK: SECCOURSE, SEMESTER, YEAR, SECNUM))

5. GradeRecord Relation

Attributes:

- **SSN (FK references Student(SSSN)), SECCOURSE, SEMESTER, YEAR, SECNUM, GRADE**

Each grade record is uniquely identified by **(SSN, SECCOURSE, SEMESTER, YEAR, SECNUM)**.

Schema:

GradeRecord(SSN (FK), SECCOURSE, SEMESTER, YEAR, SECNUM, GRADE, (PK: SSN, SECCOURSE, SEMESTER, YEAR, SECNUM))

Step 3: Summary of the Database Schema

The schema is designed to be in BCNF as all non-trivial FDs have a left-hand side that is a superkey in each relation.

1. **Student(SNUM (PK), SSSN (Unique), SNAME, SCADDR, SCPHONE, SPADDR, SPPHONE, BDATE, SEX, CLASS, MAJORDEPTCODE, MINORDEPTCODE, PROG)**
2. **Department(DEPTCODE (PK), DEPTNAME (Unique), DEPTOFFICE, DEPTPHONE, DEPTCOLLEGE)**
3. **Course(CNUM (PK), CNAME, CDESC, CREDIT, LEVEL, CDEPT (FK))**

4. **Section(SECCOURSE (FK), SEMESTER, YEAR, SECNUM, INSTRUCTORNAME, (PK: SECCOURSE, SEMESTER, YEAR, SECNUM))**
5. **GradeRecord(SSN (FK), SECCOURSE, SEMESTER, YEAR, SECNUM, GRADE, (PK: SSN, SECCOURSE, SEMESTER, YEAR, SECNUM))**

Assumptions

- **Minor Department** is optional, allowing NULL values.
- No other constraints were specified for student phone numbers or addresses, so we assume each student has one current and one permanent address and phone number.
- **CNUM** in the Course entity is unique and should be enforced as a primary key.
- **Semester and year** uniquely identify a time frame, allowing sections of courses to be distinguished based on semester and year.