|  |
| --- |
| Close-up image showing the leaf-sides of two oversized books side-by-side on a bookshelf, with additional books in soft focus background |
| Advanced Database Technology Laboratory  DMC6111 |
| |  |  |  | | --- | --- | --- | | Haryish Elangumaran | Central for Distance Education, Anna University, Guindy | Master of Computer Application | |

Advanced Database Technology – Laboratory

Subject CODE: DMC6111

**Master of Computer Applications**

For,

**Center for Distance Education**

**Anna University,**

**Guindy, Chennai, Tamil Nadu**

## INDEX

[INDEX 3](#_Toc141620361)

[INSTRUCTIONS BEFORE WRITING OBSERVATIONS: 3](#_Toc141620362)

[Exercise 1 4](#_Toc141620363)

[SOLUTION: 5](#_Toc141620364)

[MySQL: 5](#_Toc141620365)

[PostgreSQL: 10](#_Toc141620366)

[Oracle: 14](#_Toc141620367)

[MongoDB Ex-1 17](#_Toc141620368)

[Exercise 2 23](#_Toc141620369)

[SOLUTION: 24](#_Toc141620370)

[MySQL: 24](#_Toc141620371)

[PostgreSQL 29](#_Toc141620372)

[Oracle: 36](#_Toc141620373)

[MongoDB Ex-2 42](#_Toc141620374)

[Exercise 3 47](#_Toc141620375)

[SOLUTION 47](#_Toc141620376)

[MySQL: 47](#_Toc141620377)

[PostgreSQL: 52](#_Toc141620378)

[Oracle: 56](#_Toc141620379)

[MongoDB Ex-3 59](#_Toc141620380)

## INSTRUCTIONS BEFORE WRITING OBSERVATIONS:

* Suggested to write both sides on Observation notebook so as not to fill all pages in them and to Use 80 pages unruled notebook.
* Write the Question Description entirely, then Write the Solution as like how I mentioned and at each sub question, provide the estimated output for that
* Choose any one of the SQLs from the solution of each exercise to be written on Observation (Either MySQL, PostgreSQL, Oracle)
* MongoDB Question is mandatory that need to be written in Proper order
* Prefer to use Multiple (suggested minimum 2 data) on insert comments on MongoDB query
* Prefer not to write the codes under ‘Connect to MongoDB server’ (Red Font Colored)
* Look carefully on Question numbers with respective to Relevant query… Do not Blindly write abruptly.
* Reviewers can reach out to me anytime for questions or flaws in this booklet

# Exercise 1

Consider the following relations for an order-processing database application in a company:

CUSTOMER

(

**CUSTOMERNO** BIGINT,

CNAME VARCHAR (30),

CITY VARCHAR (30)

)

Implement a check constraint to check CUSTOMERNO starts with ‘4’ and length of CUSTOMERNO is 5.

CUST ORDER

(

ORDERNO BIGINT,

ODATE DATE,

CUSTOMERNO REFERENCES CUSTOMER,

ORD AMT BIGINT

)

Implement a check constraint to check ORDERNO starts with ‘5’ and length of ORDERNO is 5.

ITEM

(

ITEMNO BIGINT,

ITEM\_NAME VARCHAR (30),

UNIT\_PRICE NUMBER (5)

)

Implement a check constraint to check ITEMNO starts with ‘6’ and length of ITEMNO is 5.

ORDER ITEM

(

ORDERNO REFERENCES CUST\_ORDER,

ITEMNO REFERENCES ITEM,

QTY NUMBER (3)

)

Here, ORD\_AMT refers to total amount of an order (ORD\_AMT is a derived attribute);

ODATE is the date the order was placed; The primary key of each relation is underlined.

Perform the Following:

1. Develop DDL to implement the above Schema enforcing primary key, check constraints and foreign key constraints. Also Populate the database with a rich data set with 5 records.
2. Develop a SQL query to list the details of customers who have placed more than
3. three orders.
4. Develop a SQL query to list the details of items whose price is less than the average price of all items.
5. Develop a SQL query to list the order no and number of items in each order.
6. Develop a SQL query to list the details of items that are present in 25% of the orders.
7. Develop an update statement to update the value of ORD\_AMT.
8. Create a view that will keep track of the details of each customer and the number of orders placed.
9. Develop a database trigger that will not permit to insert more than six records in the CUST\_ORDER table for a particular order. (An order can contain a maximum of six items).
10. Implement CRUD operations in MONGO DB for the above relational schema.

## SOLUTION:

### MySQL:

1. DDL to implement the schema with primary key, check constraints, and foreign key constraints

CREATE TABLE CUSTOMER (

CUSTOMERNO BIGINT PRIMARY KEY,

CNAME VARCHAR (30),

CITY VARCHAR (30),

CHECK (CUSTOMERNO LIKE ‘4%’ AND LENGTH(CUSTOMERNO) = 5)

);

Output:

0 row(s) affected

CREATE TABLE CUST\_ORDER (

ORDERNO BIGINT PRIMARY KEY,

ODATE DATE,

CUSTOMERNO BIGINT,

ORD\_AMT BIGINT,

CHECK (ORDERNO LIKE ‘5\_\_\_\_’ AND LENGTH(ORDERNO) = 5),

FOREIGN KEY (CUSTOMERNO) REFERENCES CUSTOMER (CUSTOMERNO)

);

Output:

0 row(s) affected

CREATE TABLE ITEM (

ITEMNO BIGINT PRIMARY KEY,

ITEM\_NAME VARCHAR (30),

UNIT\_PRICE DECIMAL (5),

CHECK (ITEMNO LIKE ‘6\_\_\_\_’ AND LENGTH(ITEMNO) = 5)

);

Output:

0 row(s) affected

CREATE TABLE ORDER\_ITEM (

ORDERNO BIGINT,

ITEMNO BIGINT,

QTY INT (3),

FOREIGN KEY (ORDERNO) REFERENCES CUST\_ORDER (ORDERNO),

FOREIGN KEY (ITEMNO) REFERENCES ITEM (ITEMNO)

);

Output:

0 row(s) affected, 1 warning(s): 1681 Integer display width is deprecated and will be removed in a future release.

---------------------------------------------------------------

* 1. Populate the rich data set with 5 records in each:

INSERT INTO CUSTOMER (CUSTOMERNO, CNAME, CITY) VALUES

(40001, ‘John Doe’, ‘New York’),

(40002, ‘Jane Smith’, ‘Los Angeles’),

(40003, ‘Michael Johnson’, ‘Chicago’),

(40004, ‘Emily Davis’, ‘Houston’),

(40005, ‘Robert Wilson’, ‘San Francisco’);

Output:

5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0

INSERT INTO CUST\_ORDER (ORDERNO, ODATE, CUSTOMERNO, ORD\_AMT) VALUES

(50001, ‘2023-06-13’, 40001, 1000),

(50002, ‘2023-06-12’, 40002, 500),

(50003, ‘2023-06-11’, 40003, 750),

(50004, ‘2023-06-10’, 40001, 2000),

(50005, ‘2023-06-09’, 40004, 300);

Output:

5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0

INSERT INTO ITEM (ITEMNO, ITEM\_NAME, UNIT\_PRICE) VALUES

(60001, ‘Item 1’, 10),

(60002, ‘Item 2’, 20),

(60003, ‘Item 3’, 15),

(60004, ‘Item 4’, 30),

(60005, ‘Item 5’, 25);

Output:

5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0

INSERT INTO ORDER\_ITEM (ORDERNO, ITEMNO, QTY) VALUES

(50001, 60001, 2),

(50001, 60002, 3),

(50002, 60003, 1),

(50003, 60002, 2),

(50004, 60005, 4);

Output:

5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0

Displaying Records of all Tables:

Customer

|  |  |  |
| --- | --- | --- |
| 40001 | John Doe | New York |
| 40002 | Jane Smith | Los Angeles |
| 40003 | Michael Johnson | Chicago |
| 40004 | Emily Davis | Houston |
| 40005 | Robert Wilson | San Francisco |
|  |  |  |

Cust order

|  |  |  |  |
| --- | --- | --- | --- |
| 50001 | 2023-06-13 | 40001 | 1000 |
| 50002 | 2023-06-12 | 40002 | 500 |
| 50003 | 2023-06-11 | 40003 | 750 |
| 50004 | 2023-06-10 | 40001 | 2000 |
| 50005 | 2023-06-09 | 40004 | 300 |
|  |  |  |  |

Item

|  |  |  |
| --- | --- | --- |
| 60001 | Item 1 | 10 |
| 60002 | Item 2 | 20 |
| 60003 | Item 3 | 15 |
| 60004 | Item 4 | 30 |
| 60005 | Item 5 | 25 |
|  |  |  |

Order item

|  |  |  |
| --- | --- | --- |
| 50001 | 60001 | 2 |
| 50001 | 60002 | 3 |
| 50002 | 60003 | 1 |
| 50003 | 60002 | 2 |
| 50004 | 60005 | 4 |

---------------------------------------------------------------

c. SQL query to list the details of customers who have placed more than three orders:

SELECT C. CUSTOMERNO, C. CNAME, C. CITY

FROM CUSTOMER C

WHERE (

SELECT COUNT(\*)

FROM CUST\_ORDER O

WHERE O.CUSTOMERNO = C.CUSTOMERNO

) > 3;

Output:

---------------------------------------------------------------

d. SQL query to list the details of items whose price is less than the average price of all items:

SELECT ITEMNO, ITEM\_NAME, UNIT\_PRICE

FROM ITEM

WHERE UNIT\_PRICE < (

SELECT AVG(UNIT\_PRICE)

FROM ITEM

);

---------------------------------------------------------------

e. SQL query to list the orderno and number of items in each order:

SELECT ORDERNO, COUNT(\*) AS NUM\_ITEMS

FROM ORDER\_ITEM

GROUP BY ORDERNO;

---------------------------------------------------------------

f. SQL query to list the details of items that are present in 25% of the orders:

SELECT I.ITEMNO, I.ITEM\_NAME, I.UNIT\_PRICE

FROM ITEM I

WHERE (

SELECT COUNT(DISTINCT O.ORDERNO)

FROM ORDER\_ITEM O

) >= (SELECT COUNT(DISTINCT O.ORDERNO) FROM ORDER\_ITEM O) \* 0.25

AND O.ITEMNO = I.ITEMNO;

---------------------------------------------------------------

g. Update statement to update the value of ORD\_AMT:

UPDATE CUST\_ORDER

SET ORD\_AMT = <new\_amount>

WHERE ORDERNO = <order\_number>;

---------------------------------------------------------------

h. Create a view that keeps track of the details of each customer and the number of orders placed:

CREATE VIEW CUSTOMER\_ORDERS\_VIEW AS

SELECT C.CUSTOMERNO, C.CNAME, C.CITY, COUNT(O.ORDERNO) AS NUM\_ORDERS

FROM CUSTOMER C

LEFT JOIN CUST\_ORDER O ON C.CUSTOMERNO = O.CUSTOMERNO

GROUP BY C.CUSTOMERNO, C.CNAME, C.CITY;

---------------------------------------------------------------

i. Database trigger to limit the insertion of more than six records in the CUST\_ORDER table for a particular order:

CREATE OR REPLACE TRIGGER max\_items\_trigger

BEFORE INSERT ON CUST\_ORDER

FOR EACH ROW

DECLARE

total\_items NUMBER;

BEGIN

SELECT COUNT(\*) INTO total\_items

FROM ORDER\_ITEM

WHERE ORDERNO = :NEW.ORDERNO;

IF total\_items +

1 > 6 THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Maximum six items allowed per order');

END IF;

END;

---------------------------------------------------------------

### PostgreSQL:

1. DDL to implement the schema with primary key, check constraints, and foreign key constraints

CREATE TABLE CUSTOMER (

CUSTOMERNO BIGINT PRIMARY KEY,

CNAME VARCHAR(30),

CITY VARCHAR(30),

CHECK (CUSTOMERNO LIKE ‘4%’ AND LENGTH(CUSTOMERNO) = 5)

);

CREATE TABLE CUST\_ORDER (

ORDERNO BIGINT PRIMARY KEY,

ODATE DATE,

CUSTOMERNO BIGINT,

ORD\_AMT BIGINT,

CHECK (ORDERNO LIKE ‘5\_\_\_\_’ AND LENGTH(ORDERNO) = 5),

FOREIGN KEY (CUSTOMERNO) REFERENCES CUSTOMER (CUSTOMERNO)

);

CREATE TABLE ITEM (

ITEMNO BIGINT PRIMARY KEY,

ITEM\_NAME VARCHAR(30),

UNIT\_PRICE DECIMAL(5),

CHECK (ITEMNO LIKE ‘6\_\_\_\_’ AND LENGTH(ITEMNO) = 5)

);

CREATE TABLE ORDER\_ITEM (

ORDERNO BIGINT,

ITEMNO BIGINT,

QTY NUMERIC(3),

FOREIGN KEY (ORDERNO) REFERENCES CUST\_ORDER (ORDERNO),

FOREIGN KEY (ITEMNO) REFERENCES ITEM (ITEMNO)

);

---------------------------------------------------------------

1. Populate the rich data set with 5 records in each:

INSERT INTO CUSTOMER (CUSTOMERNO, CNAME, CITY) VALUES

(40001, ‘John Doe’, ‘New York’),

(40002, ‘Jane Smith’, ‘Los Angeles’),

(40003, ‘Michael Johnson’, ‘Chicago’),

(40004, ‘Emily Davis’, ‘Houston’),

(40005, ‘Robert Wilson’, ‘San Francisco’);

INSERT INTO CUST\_ORDER (ORDERNO, ODATE, CUSTOMERNO, ORD\_AMT) VALUES

(50001, ‘2023-06-13’, 40001, 1000),

(50002, ‘2023-06-12’, 40002, 500),

(50003, ‘2023-06-11’, 40003, 750),

(50004, ‘2023-06-10’, 40001, 2000),

(50005, ‘2023-06-09’, 40004, 300);

INSERT INTO ITEM (ITEMNO, ITEM\_NAME, UNIT\_PRICE) VALUES

(60001, ‘Item 1’, 10),

(60002, ‘Item 2’, 20),

(60003, ‘Item 3’, 15),

(60004, ‘Item 4’, 30),

(60005, ‘Item 5’, 25);

INSERT INTO ORDER\_ITEM (ORDERNO, ITEMNO, QTY) VALUES

(50001, 60001, 2),

(50001, 60002, 3),

(50002, 60003, 1),

(50003, 60002, 2),

(50004, 60005, 4);

c. SQL query to list the details of items whose price is less than the average price of all items:

```sql

SELECT \*

FROM ITEM

WHERE UNIT\_PRICE < (SELECT AVG(UNIT\_PRICE) FROM ITEM);

```

d. SQL query to list the orderno and number of items in each order:

```sql

SELECT ORDERNO, COUNT(\*) AS num\_items

FROM ORDER\_ITEM

GROUP BY ORDERNO;

```

e. SQL query to list the details of items that are present in 25% of the orders:

```sql

SELECT \*

FROM ITEM

WHERE ITEMNO IN (

SELECT ITEMNO

FROM ORDER\_ITEM

GROUP BY ITEMNO

HAVING COUNT(DISTINCT ORDERNO) >= (SELECT COUNT(DISTINCT ORDERNO) FROM CUST\_ORDER) \* 0.25

);

```

f. Update statement to update the value of ORD\_AMT:

```sql

UPDATE CUST\_ORDER

SET ORD\_AMT = 1500

WHERE ORDERNO = 50001;

```

g. Create a view that will keep track of the details of each customer and the number of orders placed:

```sql

CREATE VIEW CUSTOMER\_ORDERS\_VIEW AS

SELECT C.CUSTOMERNO, C.CNAME, C.CITY, COUNT(O.ORDERNO) AS num\_orders

FROM CUSTOMER C

LEFT JOIN CUST\_ORDER O ON C.CUSTOMERNO = O.CUSTOMERNO

GROUP BY C.CUSTOMERNO, C.CNAME, C.CITY;

```

h. Database trigger that will not permit inserting more than six records in the CUST\_ORDER table for a particular order:

```sql

CREATE OR REPLACE FUNCTION check\_order\_item\_count()

RETURNS TRIGGER AS $$

DECLARE

order\_count INTEGER;

BEGIN

SELECT COUNT(\*) INTO order\_count

FROM ORDER\_ITEM

WHERE ORDERNO = NEW.ORDERNO;

IF order\_count >= 6 THEN

RAISE EXCEPTION 'Maximum item count exceeded for the order.';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER limit\_order\_items

BEFORE INSERT ON ORDER\_ITEM

FOR EACH ROW

EXECUTE FUNCTION check\_order\_item\_count();

```

Note: For the trigger to work, you need to create the `ORDER\_ITEM` table and the necessary foreign key constraints before creating the trigger.

### Oracle:

1. DDL to implement the schema with primary key, check constraints, and foreign key constraints

CREATE TABLE CUSTOMER (

CUSTOMERNO NUMBER(19) PRIMARY KEY,

CNAME VARCHAR2(30),

CITY VARCHAR2(30),

CONSTRAINT CHK\_CUSTOMERNO CHECK (REGEXP\_LIKE(CUSTOMERNO, ‘^4.{4}$’))

);

CREATE TABLE CUST\_ORDER (

ORDERNO NUMBER(19) PRIMARY KEY,

ODATE DATE,

CUSTOMERNO NUMBER(19),

ORD\_AMT NUMBER(19),

CONSTRAINT CHK\_ORDERNO CHECK (REGEXP\_LIKE(ORDERNO, ‘^5.{4}$’)),

CONSTRAINT FK\_CUSTOMERNO FOREIGN KEY (CUSTOMERNO) REFERENCES CUSTOMER (CUSTOMERNO)

);

CREATE TABLE ITEM (

ITEMNO NUMBER(19) PRIMARY KEY,

ITEM\_NAME VARCHAR2(30),

UNIT\_PRICE NUMBER(5),

CONSTRAINT CHK\_ITEMNO CHECK (REGEXP\_LIKE(ITEMNO, ‘^6.{4}$’))

);

CREATE TABLE ORDER\_ITEM (

ORDERNO NUMBER(19),

ITEMNO NUMBER(19),

QTY NUMBER(3),

CONSTRAINT FK\_ORDERNO FOREIGN KEY (ORDERNO) REFERENCES CUST\_ORDER (ORDERNO),

CONSTRAINT FK\_ITEMNO FOREIGN KEY (ITEMNO) REFERENCES ITEM (ITEMNO)

);

---------------------------------------------------------------

b. Populate the rich data set with 5 records in each:

INSERT INTO CUSTOMER (CUSTOMERNO, CNAME, CITY) VALUES

(40001, ‘John Doe’, ‘New York’),

(40002, ‘Jane Smith’, ‘Los Angeles’),

(40003, ‘Michael Johnson’, ‘Chicago’),

(40004, ‘Emily Davis’, ‘Houston’),

(40005, ‘Robert Wilson’, ‘San Francisco’);

INSERT INTO CUST\_ORDER (ORDERNO, ODATE, CUSTOMERNO, ORD\_AMT) VALUES

(50001, TO\_DATE(‘2023-06-13’, ‘YYYY-MM-DD’), 40001, 1000),

(50002, TO\_DATE(‘2023-06-12’, ‘YYYY-MM-DD’), 40002, 500),

(50003, TO\_DATE(‘2023-06-11’, ‘YYYY-MM-DD’), 40003, 750),

(50004, TO\_DATE(‘2023-06-10’, ‘YYYY-MM-DD’), 40001, 2000),

(50005, TO\_DATE(‘2023-06-09’, ‘YYYY-MM-DD’), 40004, 300);

INSERT INTO ITEM (ITEMNO, ITEM\_NAME, UNIT\_PRICE) VALUES

(60001, ‘Item 1’, 10),

(60002, ‘Item 2’, 20),

(60003, ‘Item 3’, 15),

(60004, ‘Item 4’, 30),

(60005, ‘Item 5’, 25);

INSERT INTO ORDER\_ITEM (ORDERNO, ITEMNO, QTY) VALUES

(50001, 60001, 2),

(50001, 60002, 3),

(50002, 60003, 1),

(50003, 60002, 2),

(50004, 60005, 4);

c. SQL query to list the details of items whose price is less than the average price of all items:

```sql

SELECT \*

FROM ITEM

WHERE UNIT\_PRICE < (SELECT AVG(UNIT\_PRICE) FROM ITEM);

```

d. SQL query to list the orderno and number of items in each order:

```sql

SELECT ORDERNO, COUNT(\*) AS num\_items

FROM ORDER\_ITEM

GROUP BY ORDERNO;

```

e. SQL query to list the details of items that are present in 25% of the orders:

```sql

SELECT \*

FROM ITEM

WHERE ITEMNO IN (

SELECT ITEMNO

FROM ORDER\_ITEM

GROUP BY ITEMNO

HAVING COUNT(DISTINCT ORDERNO) >= (SELECT COUNT(DISTINCT ORDERNO) FROM CUST\_ORDER) \* 0.25

);

```

f. Update statement to update the value of ORD\_AMT:

```sql

UPDATE CUST\_ORDER

SET ORD\_AMT = 1500

WHERE ORDERNO = 50001;

```

g. Create a view that will keep track of the details of each customer and the number of orders placed:

```sql

CREATE VIEW CUSTOMER\_ORDERS\_VIEW AS

SELECT C.CUSTOMERNO, C.CNAME, C.CITY, COUNT(O.ORDERNO) AS num\_orders

FROM CUSTOMER C

LEFT JOIN CUST\_ORDER O ON C.CUSTOMERNO = O.CUSTOMERNO

GROUP BY C.CUSTOMERNO, C.CNAME, C.CITY;

```

h. Database trigger that will not permit inserting more than six records in the CUST\_ORDER table for a particular order:

```sql

CREATE OR REPLACE TRIGGER limit\_order\_items

BEFORE INSERT ON ORDER\_ITEM

FOR EACH ROW

DECLARE

order\_count INTEGER;

BEGIN

SELECT COUNT(\*) INTO order\_count

FROM ORDER\_ITEM

WHERE ORDERNO = :NEW.ORDERNO;

IF order\_count >= 6 THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Maximum item count exceeded for the order.');

END IF;

END;

/

```

Note: For the trigger to work, you need to create the `ORDER\_ITEM` table and the necessary foreign key constraints before creating the trigger.

### MongoDB Ex-1

j. Here's an example implementation of CRUD operations for the given relational schema in MongoDB:

1. Creating a database:

use order\_processing\_database

-----------------------------------------------------------------------------------------------------------------

Note: This command creates a new database named "order\_processing\_database" or switches to it if it already exists.

2. Creating collections with desired constraints:

Create the "customer" collection with the desired constraint on CUSTOMERNO:

db.createCollection("customer", {

validator: {

$jsonSchema: {

bsonType: "object",

required: ["CUSTOMERNO", "CNAME", "CITY"],

properties: {

CUSTOMERNO: {

bsonType: "long",

description: "Must start with '4' and have a length of 5",

pattern: "^4\\d{4}$"

},

CNAME: {

bsonType: "string",

description: "Customer name"

},

CITY: {

bsonType: "string",

description: "City"

}

}

}

}

})

-----------------------------------------------------------------------------------------------------------------

Note: The `validator` option specifies the validation rules for the collection. The `bsonType` defines the data type, and the `pattern` property specifies the regular expression pattern for CUSTOMERNO.

Create the "cust\_order" collection with the desired constraint on ORDERNO:

db.createCollection("cust\_order", {

validator: {

$jsonSchema: {

bsonType: "object",

required: ["ORDERNO", "ODATE", "CUSTOMERNO", "ORD\_AMT"],

properties: {

ORDERNO: {

bsonType: "long",

description: "Must start with '5' and have a length of 5",

pattern: "^5\\d{4}$"

},

ODATE: {

bsonType: "date",

description: "Order date"

},

CUSTOMERNO: {

bsonType: "long",

description: "Customer number"

},

ORD\_AMT: {

bsonType: "long",

description: "Order amount"

}

}

}

}

})

-----------------------------------------------------------------------------------------------------------------

Create the "item" collection with the desired constraint on ITEMNO:

db.createCollection("item", {

validator: {

$jsonSchema: {

bsonType: "object",

required: ["ITEMNO", "ITEM\_NAME", "UNIT\_PRICE"],

properties: {

ITEMNO: {

bsonType: "long",

description: "Must start with '6' and have a length of 5",

pattern: "^6\\d{4}$"

},

ITEM\_NAME: {

bsonType: "string",

description: "Item name"

},

UNIT\_PRICE: {

bsonType: "double",

description: "Unit price"

}

}

}

}

})

-----------------------------------------------------------------------------------------------------------------

Create the "order\_item" collection with references to "cust\_order" and "item" collections:

db.createCollection("order\_item", {

validator: {

$jsonSchema: {

bsonType: "object",

required: ["ORDERNO", "ITEMNO", "QTY"],

properties: {

ORDERNO: {

bsonType: "long",

description: "Reference to the ORDERNO field in the cust\_order collection",

pattern: "^5\\d{4}$"

},

ITEMNO: {

bsonType: "long",

description: "Reference to the ITEMNO field in the item collection",

pattern: "^6\\d{4}$"

},

QTY: {

bsonType: "int",

description: "Quantity"

}

}

}

}

})

-----------------------------------------------------------------------------------------------------------------

Note: In the "order\_item" collection, the `pattern` property for ORDERNO and ITEMNO ensures that they start with '5' and '6', respectively, and have a length of 5.

3. CRUD operations:

- Insert a record into the "customer" collection:

db.customer.insertOne({

CUSTOMERNO: 40001,

CNAME: "John Doe",

CITY: "New York"

})

-----------------------------------------------------------------------------------------------------------------

- Insert a record into the "cust\_order" collection:

db.cust\_order.insertOne({

ORDERNO: 50001,

ODATE: ISODate("2023-06-14"),

CUSTOMERNO: 40001,

ORD\_AMT: 1000

})

-----------------------------------------------------------------------------------------------------------------

- Insert a record into the "item" collection:

db.item.insertOne({

ITEMNO: 60001,

ITEM\_NAME: "Product 1",

UNIT\_PRICE: 10.99

})

-----------------------------------------------------------------------------------------------------------------

- Insert a record into the "order\_item" collection:

db.order\_item.insertOne({

ORDERNO: 50001,

ITEMNO: 60001,

QTY: 5

})

-----------------------------------------------------------------------------------------------------------------

- Read records from the "customer" collection:

db.customer.find({ CITY: "New York" })

-----------------------------------------------------------------------------------------------------------------

- Update a record in the "cust\_order" collection:

db.cust\_order.updateOne(

{ ORDERNO: 50001 },

{ $set: { ORD\_AMT: 1500 } }

)

-----------------------------------------------------------------------------------------------------------------

- Delete a record from the "item" collection:

db.item.deleteOne({ ITEMNO: 60001 })

-----------------------------------------------------------------------------------------------------------------

These are some basic examples of CRUD operations in MongoDB for the given schema. You can modify the queries according to your specific requirements and conditions for reading the records.

---------------------------------------------------------------

# Exercise 2

Consider the following relational schema for the office of the controller of examinations application:

STUDENT (ROLLNO, NAME, DOB, GENDER, DOA, BCODE)

Implement a check constraint for GENDER

DOA-Date of admission

BRANCH (BCODE, BNAME, DNO)

DEPARTMENT (DNO, DNAME)

COURSE (CCODE, CNAME, CREDITS, DNO)

BRANCH COURSE (BCODE, CCODE, SEMESTER)

PREREQUISITE\_COURSE (CCODE, PCCODE)

A course can have prerequisite courses. For example, Database Management Systems is a prerequisite course for Advanced Databases.

ENROLLS (ROLLNO, CCODE, SESS, GRADE)

For Example: SESS can take the values APRIL20201, NOVEMBER2020

Implement a check constraint for GRADE

VALUE SET (‘S’, ‘A’, ‘B’, ‘C’, ‘D’, ‘E’, ‘U’)

For a student to enroll for a course he/she should have completed the prerequisite courses. Students are admitted to branches. Branches are offered by departments. A branch is I offered only by one department. Each branch has a set of courses (subjects) each student must enroll during a semester. Courses are offered by departments. A course is offered only by one department. If a student is unsuccessful in a course he/she must enroll for the course during the next session. A student has successfully completed a course if a the grade student obtained is from the value set (‘S’, ‘A’, ‘B’, ‘C’, ‘D’, ‘E’). A student is unsuccessful if he/she have obtained a ‘U’ grade in a course.

The primary keys are underlined.

Perform the following:

a. Develop DDL to implement the above Schema specifying appropriate data types for each attribute enforcing primary key, check constraints and foreign key constraints.

b. Populate the database with a rich data set.

c. Develop a SQL query to list the details of departments that offer more than three branches.

d. Develop a SQL query to list the details of courses that do not have prerequisite courses.

e. Develop a SQL query to list the details of courses that are common for more than three branches.

f. Develop a SQL query to list the details of students who have got a ‘U’ grade in more than two courses during a single enrollment.

g. Create a view that will keep track of the course code, name and number of prerequisite

courses.

h. Develop a database trigger that will not permit a student to enroll for a course if he/ she have not completed the prerequisite courses.

i. Develop a procedure DISP that will accept a ROLLNO of a student as input and print the roll number, name and number of courses a student has successfully completed.

j. Develop a procedure DISP\_NOE that will accept a CCODE of a COURSE as input and print the roll number, name of students who have enrolled for the course more than twice.

k. Implement CRUD operations in MONGO DB for the above relational schema.

## SOLUTION:

### MySQL:

1. DDL to implement the schema:

CREATE TABLE STUDENT (

ROLLNO INT PRIMARY KEY,

NAME VARCHAR(30),

DOB DATE,

GENDER ENUM(‘M’, ‘F’),

DOA DATE,

BCODE INT,

CHECK (GENDER IN (‘M’, ‘F’)),

FOREIGN KEY (BCODE) REFERENCES BRANCH (BCODE)

);

CREATE TABLE BRANCH (

BCODE INT PRIMARY KEY,

BNAME VARCHAR(30),

DNO INT,

FOREIGN KEY (DNO) REFERENCES DEPARTMENT (DNO)

);

CREATE TABLE DEPARTMENT (

DNO INT PRIMARY KEY,

DNAME VARCHAR(30)

);

CREATE TABLE COURSE (

CCODE INT PRIMARY KEY,

CNAME VARCHAR(30),

CREDITS INT,

DNO INT,

FOREIGN KEY (DNO) REFERENCES DEPARTMENT (DNO)

);

CREATE TABLE BRANCH\_COURSE (

BCODE INT,

CCODE INT,

SEMESTER INT,

PRIMARY KEY (BCODE, CCODE),

FOREIGN KEY (BCODE) REFERENCES BRANCH (BCODE),

FOREIGN KEY (CCODE) REFERENCES COURSE (CCODE)

);

CREATE TABLE PREREQUISITE\_COURSE (

CCODE INT,

PCCODE INT,

PRIMARY KEY (CCODE, PCCODE),

FOREIGN KEY (CCODE) REFERENCES COURSE (CCODE),

FOREIGN KEY (PCCODE) REFERENCES COURSE (CCODE)

);

CREATE TABLE ENROLLS (

ROLLNO INT,

CCODE INT,

SESS VARCHAR(20),

GRADE ENUM(‘S’, ‘A’, ‘B’, ‘C’, ‘D’, ‘E’, ‘U’),

PRIMARY KEY (ROLLNO, CCODE, SESS),

FOREIGN KEY (ROLLNO) REFERENCES STUDENT (ROLLNO),

FOREIGN KEY (CCODE) REFERENCES COURSE (CCODE)

);

---------------------------------------------------------------

1. Data Population:

INSERT INTO STUDENT (ROLLNO, NAME, DOB, GENDER, DOA, BCODE) VALUES

(1, ‘John Doe’, ‘2000-01-01’, ‘M’, ‘2022-01-01’, 1),

(2, ‘Jane Smith’, ‘2001-02-03’, ‘F’, ‘2022-01-01’, 2),

(3, ‘Michael Johnson’, ‘1999-05-05’, ‘M’, ‘2022-01-01’, 1),

(4, ‘Emily Davis’, ‘2002-07-10’, ‘F’, ‘2022-01-01’, 2),

(5, ‘Robert Wilson’, ‘1998-12-15’, ‘M’, ‘2022-01-01’, 1);

INSERT INTO BRANCH (BCODE, BNAME, DNO) VALUES

(1, ‘Branch 1’, 1),

(2, ‘Branch 2’, 2);

INSERT INTO DEPARTMENT (DNO, DNAME) VALUES

(1, ‘Department 1’),

(2, ‘Department 2’);

INSERT INTO COURSE (CCODE, CNAME, CREDITS, DNO) VALUES

(1, ‘Course 1’, 3, 1),

(2, ‘Course 2’, 4, 1),

(3, ‘Course 3’, 3, 2),

(4, ‘Course 4’, 4, 2),

(5, ‘Course 5’, 3, 1);

INSERT INTO BRANCH\_COURSE (BCODE, CCODE, SEMESTER) VALUES

(1, 1, 1),

(1, 2, 1),

(2, 3, 1),

(2, 4, 1),

(1, 5, 2);

INSERT INTO PREREQUISITE\_COURSE (CCODE, PCCODE) VALUES

(2, 1),

(4, 3);

INSERT INTO ENROLLS (ROLLNO, CCODE, SESS, GRADE) VALUES

(1, 1, ‘APRIL20201’, ‘A’),

(1, 2, ‘APRIL20201’, ‘B’),

(2, 1, ‘APRIL20201’, ‘C’),

(2, 3, ‘APRIL20201’, ‘D’),

(3, 1, ‘APRIL20201’, ‘S’);

---------------------------------------------------------------

1. SQL query to list the details of departments that offer more than three branches:

SELECT D.DNO, D.DNAME

FROM DEPARTMENT D

JOIN BRANCH B ON D.DNO = B.DNO

GROUP BY D.DNO, D.DNAME

HAVING COUNT(B.BCODE) > 3;

---------------------------------------------------------------

1. SQL query to list the details of courses that do not have prerequisite courses:

SELECT C.CCODE, C.CNAME

FROM COURSE C

LEFT JOIN PREREQUISITE\_COURSE PC ON C.CCODE = PC.CCODE

WHERE PC.CCODE IS NULL;

---------------------------------------------------------------

1. SQL query to list the details of courses that are common for more than three branches:

SELECT C.CCODE, C.CNAME

FROM COURSE C

JOIN BRANCH\_COURSE BC ON C.CCODE = BC.CCODE

GROUP BY C.CCODE, C.CNAME

HAVING COUNT(BC.BCODE) > 3;

---------------------------------------------------------------

1. SQL query to list the details of students who have got a ‘U’ grade in more than two courses during a single enrollment:

SELECT S.ROLLNO, S.NAME

FROM STUDENT S

JOIN ENROLLS E ON S.ROLLNO = E.ROLLNO

WHERE E.GRADE = ‘U’

GROUP BY S.ROLLNO, S.NAME

HAVING COUNT(DISTINCT E.CCODE) > 2;

---------------------------------------------------------------

1. View to keep track of the course code, name, and number of prerequisite courses:

CREATE VIEW COURSE\_PREREQUISITES AS

SELECT C.CCODE, C.CNAME, COUNT(PC.PCCODE) AS NUM\_PREREQUISITES

FROM COURSE C

LEFT JOIN PREREQUISITE\_COURSE PC ON C.CCODE = PC.CCODE

GROUP BY C.CCODE, C.CNAME;

---------------------------------------------------------------

1. Database trigger to not permit a student to enroll for a course if they have not completed the prerequisite courses:

DELIMITER //

CREATE TRIGGER CHECK\_PREREQUISITES\_BEFORE\_ENROLLMENT

BEFORE INSERT ON ENROLLS

FOR EACH ROW

BEGIN

DECLARE COUNT\_PREREQUISITES INT;

SELECT COUNT(\*) INTO COUNT\_PREREQUISITES

FROM PREREQUISITE\_COURSE

WHERE CCODE = NEW.CCODE

AND PCCODE NOT IN (

SELECT CCODE

FROM ENROLLS

WHERE ROLLNO = NEW.ROLLNO

AND GRADE IN (‘S’, ‘A’, ‘B’, ‘C’, ‘D’, ‘E’)

);

IF COUNT\_PREREQUISITES > 0 THEN

SIGNAL SQLSTATE ‘45000’ SET MESSAGE\_TEXT = ‘Cannot enroll for the course without completing prerequisite courses.’;

END IF;

END//

DELIMITER ;

---------------------------------------------------------------

1. Procedure DISP to accept a ROLLNO of a student as input and print the roll number, name, and the number of courses a student has successfully completed:

DELIMITER //

CREATE PROCEDURE DISP(IN STUDENT\_ROLLNO INT)

BEGIN

DECLARE STUDENT\_NAME VARCHAR(30);

DECLARE NUM\_COMPLETED\_COURSES INT;

SELECT NAME INTO STUDENT\_NAME

FROM STUDENT

WHERE ROLLNO = STUDENT\_ROLLNO;

SELECT COUNT(\*) INTO NUM\_COMPLETED\_COURSES

FROM ENROLLS

WHERE ROLLNO = STUDENT\_ROLLNO

AND GRADE IN (‘S’, ‘A’, ‘B’, ‘C’, ‘D’, ‘E’);

SELECT STUDENT\_ROLLNO, STUDENT\_NAME, NUM\_COMPLETED\_COURSES;

END//

DELIMITER ;

---------------------------------------------------------------

J. Procedure DISP\_NOE to accept a CCODE of a COURSE as input and print the roll number and name of students who have enrolled for the course more than twice:

DELIMITER //

CREATE PROCEDURE DISP\_NOE(IN COURSE\_CCODE INT)

BEGIN

SELECT S.ROLLNO, S.NAME

FROM STUDENT S

JOIN ENROLLS E ON S.ROLLNO = E.ROLLNO

WHERE E.CCODE = COURSE\_CCODE

GROUP BY S.ROLLNO, S.NAME

HAVING COUNT(E.CCODE) > 2;

END//

DELIMITER ;

---------------------------------------------------------------

### PostgreSQL

a. DDL to implement the above schema in PostgreSQL:

CREATE TABLE STUDENT (

ROLLNO SERIAL PRIMARY KEY,

NAME VARCHAR(50),

DOB DATE,

GENDER CHAR(1) CHECK (GENDER IN ('M', 'F')),

DOA DATE,

BCODE INTEGER,

FOREIGN KEY (BCODE) REFERENCES BRANCH (BCODE)

);

CREATE TABLE BRANCH (

BCODE INTEGER PRIMARY KEY,

BNAME VARCHAR(50),

DNO INTEGER,

FOREIGN KEY (DNO) REFERENCES DEPARTMENT (DNO)

);

CREATE TABLE DEPARTMENT (

DNO INTEGER PRIMARY KEY,

DNAME VARCHAR(50)

);

CREATE TABLE COURSE (

CCODE INTEGER PRIMARY KEY,

CNAME VARCHAR(50),

CREDITS INTEGER,

DNO INTEGER,

FOREIGN KEY (DNO) REFERENCES DEPARTMENT (DNO)

);

CREATE TABLE BRANCH\_COURSE (

BCODE INTEGER,

CCODE INTEGER,

SEMESTER INTEGER,

PRIMARY KEY (BCODE, CCODE),

FOREIGN KEY (BCODE) REFERENCES BRANCH (BCODE),

FOREIGN KEY (CCODE) REFERENCES COURSE (CCODE)

);

CREATE TABLE PREREQUISITE\_COURSE (

CCODE INTEGER,

PCCODE INTEGER,

PRIMARY KEY (CCODE, PCCODE),

FOREIGN KEY (CCODE) REFERENCES COURSE (CCODE),

FOREIGN KEY (PCCODE) REFERENCES COURSE (CCODE)

);

CREATE TABLE ENROLLS (

ROLLNO INTEGER,

CCODE INTEGER,

SESS VARCHAR(20),

GRADE CHAR(1) CHECK (GRADE IN ('S', 'A', 'B', 'C', 'D', 'E', 'U')),

PRIMARY KEY (ROLLNO, CCODE, SESS),

FOREIGN KEY (ROLLNO) REFERENCES STUDENT (ROLLNO),

FOREIGN KEY (CCODE) REFERENCES COURSE (CCODE)

);

-----------------------------------------------------------------------------------------------------------------

b. Population of the database with sample data is specific to your requirements and use case. You can insert data into the tables using `INSERT INTO` statements.

-- Insert data into the DEPARTMENT table

INSERT INTO DEPARTMENT (DNO, DNAME) VALUES

(1, 'Computer Science'),

(2, 'Electrical Engineering'),

(3, 'Mechanical Engineering');

-- Insert data into the BRANCH table

INSERT INTO BRANCH (BCODE, BNAME, DNO) VALUES

(1, 'CSE Branch', 1),

(2, 'EEE Branch', 2),

(3, 'Mech Branch', 3);

-- Insert data into the COURSE table

INSERT INTO COURSE (CCODE, CNAME, CREDITS, DNO) VALUES

(1, 'Database Management Systems', 4, 1),

(2, 'Operating Systems', 3, 1),

(3, 'Digital Electronics', 3, 2),

(4, 'Power Systems', 4, 2),

(5, 'Mechanics', 3, 3),

(6, 'Thermodynamics', 4, 3);

-- Insert data into the STUDENT table

INSERT INTO STUDENT (NAME, DOB, GENDER, DOA, BCODE) VALUES

('John Doe', '1998-05-10', 'M', '2021-09-01', 1),

('Jane Smith', '1999-07-15', 'F', '2021-09-01', 2),

('Michael Johnson', '1997-02-28', 'M', '2021-09-01', 3);

-- Insert data into the BRANCH\_COURSE table

INSERT INTO BRANCH\_COURSE (BCODE, CCODE, SEMESTER) VALUES

(1, 1, 1),

(1, 2, 2),

(2, 3, 1),

(2, 4, 2),

(3, 5, 1),

(3, 6, 2);

-- Insert data into the PREREQUISITE\_COURSE table

INSERT INTO PREREQUISITE\_COURSE (CCODE, PCCODE) VALUES

(2, 1),

(4, 3),

(6, 5);

-- Insert data into the ENROLLS table

INSERT INTO ENROLLS (ROLLNO, CCODE, SESS, GRADE) VALUES

(1, 1, 'APRIL20201', 'A'),

(1, 2, 'APRIL20201', 'B'),

(2, 3, 'APRIL20201', 'A'),

(2, 4, 'APRIL20201', 'B'),

(3, 5, 'APRIL20201', 'B'),

(3, 6, 'APRIL20201', 'C');

----------------------------------------------------------------------------------------

c. SQL query to list the details of departments that offer more than three branches:

SELECT DNO, DNAME

FROM DEPARTMENT

WHERE DNO IN (

SELECT DNO

FROM BRANCH

GROUP BY DNO

HAVING COUNT(\*) > 3

);

-----------------------------------------------------------------------------------------------------------------

d. SQL query to list the details of courses that do not have prerequisite courses:

SELECT CCODE, CNAME

FROM COURSE

WHERE CCODE NOT IN (

SELECT CCODE

FROM PREREQUISITE\_COURSE

);

-----------------------------------------------------------------------------------------------------------------

e. SQL query to list the details of courses that are common for more than three branches:

SELECT CCODE, CNAME

FROM COURSE

WHERE CCODE IN (

SELECT CCODE

FROM BRANCH\_COURSE

GROUP BY CCODE

HAVING COUNT(DISTINCT BCODE) > 3

);

-----------------------------------------------------------------------------------------------------------------

f. SQL query to list the details of students who have received a 'U' grade in more than two courses during a single enrollment:

SELECT ROLLNO, NAME

FROM STUDENT

WHERE ROLLNO IN (

SELECT ROLLNO

FROM ENROLLS

WHERE GRADE = 'U'

GROUP BY ROLLNO, SESS

HAVING COUNT(\*) > 2

);

-----------------------------------------------------------------------------------------------------------------

g. Creating a view to keep track of the course code, name, and number of prerequisite courses:

CREATE VIEW COURSE\_DETAILS AS

SELECT C.CCODE, C.CNAME, COUNT(P.PCCODE) AS NUM\_PREREQUISITES

FROM COURSE C

LEFT JOIN PREREQUISITE\_COURSE P ON C.CCODE = P.CCODE

GROUP BY C.CCODE, C.CNAME;

-----------------------------------------------------------------------------------------------------------------

h. Creating a database trigger that will not permit a student to enroll for a course if they have not completed the prerequisite courses:

CREATE FUNCTION CHECK\_PREREQUISITES() RETURNS TRIGGER AS $$

BEGIN

IF NEW.CCODE NOT IN (

SELECT PCCODE

FROM PREREQUISITE\_COURSE

WHERE CCODE = NEW.CCODE

) THEN

RAISE EXCEPTION 'Student has not completed the prerequisite courses for this course';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER ENROLLS\_TRIGGER

BEFORE INSERT ON ENROLLS

FOR EACH ROW

EXECUTE FUNCTION CHECK\_PREREQUISITES();

-----------------------------------------------------------------------------------------------------------------

i. Creating a procedure `DISP` that accepts a ROLLNO of a student as input and prints the roll number, name, and number of courses a student has successfully completed:

CREATE OR REPLACE PROCEDURE DISP(IN STUDENT\_ROLLNO INTEGER)

LANGUAGE plpgsql

AS $$

DECLARE

STUDENT\_NAME VARCHAR(50);

COURSE\_COUNT INTEGER;

BEGIN

SELECT NAME INTO STUDENT\_NAME FROM STUDENT WHERE ROLLNO = STUDENT\_ROLLNO;

SELECT COUNT(\*) INTO COURSE\_COUNT FROM ENROLLS WHERE ROLLNO = STUDENT\_ROLLNO AND GRADE IN ('S', 'A', 'B', 'C', 'D', 'E');

RAISE NOTICE 'Roll No: %', STUDENT\_ROLLNO;

RAISE NOTICE 'Name: %', STUDENT\_NAME;

RAISE NOTICE 'Number of Courses Successfully Completed: %', COURSE\_COUNT;

END;

$$;

-----------------------------------------------------------------------------------------------------------------

j. Creating a procedure `DISP\_NOE` that accepts a CCODE of a COURSE as input and prints the roll number and name of students who have enrolled for the course more than twice:

CREATE OR REPLACE PROCEDURE DISP\_NOE(IN COURSE\_CCODE INTEGER)

LANGUAGE plpgsql

AS $$

DECLARE

STUDENT\_ROLLNO INTEGER;

STUDENT\_NAME VARCHAR(50);

BEGIN

FOR STUDENT\_ROLLNO, STUDENT\_NAME IN

SELECT E.ROLLNO, S.NAME

FROM ENROLLS E

INNER JOIN STUDENT S ON E.ROLLNO = S.ROLLNO

WHERE E.CCODE = COURSE\_CCODE

GROUP BY E.ROLLNO, S.NAME

HAVING COUNT(\*) > 2

LOOP

RAISE NOTICE 'Roll No: %', STUDENT\_ROLLNO;

RAISE NOTICE 'Name: %', STUDENT\_NAME;

END LOOP;

END;

$$;

-----------------------------------------------------------------------------------------------------------------

Note: The above SQL queries, views, and procedures are written in PostgreSQL syntax.

### Oracle:

DDL to implement the schema:

CREATE TABLE STUDENT (

ROLLNO NUMBER PRIMARY KEY,

NAME VARCHAR2(30),

DOB DATE,

GENDER CHAR(1) CHECK (GENDER IN (‘M’, ‘F’)),

DOA DATE,

BCODE NUMBER,

CONSTRAINT FK\_BRANCH FOREIGN KEY (BCODE) REFERENCES BRANCH (BCODE)

);

CREATE TABLE BRANCH (

BCODE NUMBER PRIMARY KEY,

BNAME VARCHAR2(30),

DNO NUMBER,

CONSTRAINT FK\_DEPARTMENT FOREIGN KEY (DNO) REFERENCES DEPARTMENT (DNO)

);

CREATE TABLE DEPARTMENT (

DNO NUMBER PRIMARY KEY,

DNAME VARCHAR2(30)

);

CREATE TABLE COURSE (

CCODE NUMBER PRIMARY KEY,

CNAME VARCHAR2(30),

CREDITS NUMBER,

DNO NUMBER,

CONSTRAINT FK\_DEPARTMENT FOREIGN KEY (DNO) REFERENCES DEPARTMENT (DNO)

);

CREATE TABLE BRANCH\_COURSE (

BCODE NUMBER,

CCODE NUMBER,

SEMESTER NUMBER,

CONSTRAINT PK\_BRANCH\_COURSE PRIMARY KEY (BCODE, CCODE),

CONSTRAINT FK\_BRANCH FOREIGN KEY (BCODE) REFERENCES BRANCH (BCODE),

CONSTRAINT FK\_COURSE FOREIGN KEY (CCODE) REFERENCES COURSE (CCODE)

);

CREATE TABLE PREREQUISITE\_COURSE (

CCODE NUMBER,

PCCODE NUMBER,

CONSTRAINT PK\_PREREQUISITE\_COURSE PRIMARY KEY (CCODE, PCCODE),

CONSTRAINT FK\_COURSE FOREIGN KEY (CCODE) REFERENCES COURSE (CCODE),

CONSTRAINT FK\_PREREQUISITE\_COURSE FOREIGN KEY (PCCODE) REFERENCES COURSE (CCODE)

);

CREATE TABLE ENROLLS (

ROLLNO NUMBER,

CCODE NUMBER,

SESS VARCHAR2(20),

GRADE CHAR(1) CHECK (GRADE IN (‘S’, ‘A’, ‘B’, ‘C’, ‘D’, ‘E’, ‘U’)),

CONSTRAINT PK\_ENROLLS PRIMARY KEY (ROLLNO, CCODE, SESS),

CONSTRAINT FK\_STUDENT FOREIGN KEY (ROLLNO) REFERENCES STUDENT (ROLLNO),

CONSTRAINT FK\_COURSE FOREIGN KEY (CCODE

) REFERENCES COURSE (CCODE)

);

---------------------------------------------------------------

Data Population:

INSERT INTO STUDENT (ROLLNO, NAME, DOB, GENDER, DOA, BCODE) VALUES

(1, ‘John Doe’, TO\_DATE(‘2000-01-01’, ‘YYYY-MM-DD’), ‘M’, TO\_DATE(‘2022-01-01’, ‘YYYY-MM-DD’), 1),

(2, ‘Jane Smith’, TO\_DATE(‘2001-02-03’, ‘YYYY-MM-DD’), ‘F’, TO\_DATE(‘2022-01-01’, ‘YYYY-MM-DD’), 2),

(3, ‘Michael Johnson’, TO\_DATE(‘1999-05-05’, ‘YYYY-MM-DD’), ‘M’, TO\_DATE(‘2022-01-01’, ‘YYYY-MM-DD’), 1),

(4, ‘Emily Davis’, TO\_DATE(‘2002-07-10’, ‘YYYY-MM-DD’), ‘F’, TO\_DATE(‘2022-01-01’, ‘YYYY-MM-DD’), 2),

(5, ‘Robert Wilson’, TO\_DATE(‘1998-12-15’, ‘YYYY-MM-DD’), ‘M’, TO\_DATE(‘2022-01-01’, ‘YYYY-MM-DD’), 1);

INSERT INTO BRANCH (BCODE, BNAME, DNO) VALUES

(1, ‘Branch 1’, 1),

(2, ‘Branch 2’, 2);

INSERT INTO DEPARTMENT (DNO, DNAME) VALUES

(1, ‘Department 1’),

(2, ‘Department 2’);

INSERT INTO COURSE (CCODE, CNAME, CREDITS, DNO) VALUES

(1, ‘Course 1’, 3, 1),

(2, ‘Course 2’, 4, 1),

(3, ‘Course 3’, 3, 2),

(4, ‘Course 4’, 4, 2),

(5, ‘Course 5’, 3, 1);

INSERT INTO BRANCH\_COURSE (BCODE, CCODE, SEMESTER) VALUES

(1, 1, 1),

(1, 2, 1),

(2, 3, 1),

(2, 4, 1),

(1, 5, 2);

INSERT INTO PREREQUISITE\_COURSE (CCODE, PCCODE) VALUES

(2, 1),

(4, 3);

INSERT INTO ENROLLS (ROLLNO, CCODE, SESS, GRADE) VALUES

(1, 1, ‘APRIL20201’, ‘A’),

(1, 2, ‘APRIL20201’, ‘B’),

(2, 1, ‘APRIL20201’, ‘C’),

(2, 3, ‘APRIL20201’, ‘D’),

(3, 1, ‘APRIL20201’, ‘S’);

---------------------------------------------------------------

SQL query to list the details of departments that offer more than three branches:

SELECT D.DNO, D.DNAME

FROM DEPARTMENT D

JOIN BRANCH B ON D.DNO = B.DNO

GROUP BY D.DNO, D.DNAME

HAVING COUNT(B.BCODE) > 3;

---------------------------------------------------------------

SQL query to list the details of courses that do not have prerequisite courses:

SELECT C.CCODE, C.CNAME

FROM COURSE C

LEFT JOIN PREREQUISITE\_COURSE PC ON C.CCODE = PC.CCODE

WHERE PC.CCODE IS NULL;

---------------------------------------------------------------

SQL query to list the details of courses that are common for more than three branches:

SELECT C

.CCODE, C.CNAME

FROM COURSE C

JOIN BRANCH\_COURSE BC ON C.CCODE = BC.CCODE

GROUP BY C.CCODE, C.CNAME

HAVING COUNT(BC.BCODE) > 3;

---------------------------------------------------------------

SQL query to list the details of students who have got a ‘U’ grade in more than two courses during a single enrollment:

SELECT S.ROLLNO, S.NAME

FROM STUDENT S

JOIN ENROLLS E ON S.ROLLNO = E.ROLLNO

WHERE E.GRADE = ‘U’

GROUP BY S.ROLLNO, S.NAME

HAVING COUNT(DISTINCT E.CCODE) > 2;

---------------------------------------------------------------

View to keep track of the course code, name, and number of prerequisite courses:

CREATE VIEW COURSE\_PREREQUISITES AS

SELECT C.CCODE, C.CNAME, COUNT(PC.PCCODE) AS NUM\_PREREQUISITES

FROM COURSE C

LEFT JOIN PREREQUISITE\_COURSE PC ON C.CCODE = PC.CCODE

GROUP BY C.CCODE, C.CNAME;

---------------------------------------------------------------

Database trigger to not permit a student to enroll for a course if they have not completed the prerequisite courses:

CREATE OR REPLACE TRIGGER CHECK\_PREREQUISITES\_BEFORE\_ENROLLMENT

BEFORE INSERT ON ENROLLS

FOR EACH ROW

DECLARE

COUNT\_PREREQUISITES NUMBER;

BEGIN

SELECT COUNT(\*)

INTO COUNT\_PREREQUISITES

FROM PREREQUISITE\_COURSE

WHERE CCODE = :NEW.CCODE

AND PCCODE NOT IN (

SELECT CCODE

FROM ENROLLS

WHERE ROLLNO = :NEW.ROLLNO

AND GRADE IN (‘S’, ‘A’, ‘B’, ‘C’, ‘D’, ‘E’)

);

IF COUNT\_PREREQUISITES > 0 THEN

Raise\_application\_error(-20001, ‘Cannot enroll for the course without completing prerequisite courses.’);

END IF;

END;

/

---------------------------------------------------------------

Procedure DISP to accept a ROLLNO of a student as input and print the roll number, name, and the number of courses a student has successfully completed:

CREATE OR REPLACE PROCEDURE DISP(STUDENT\_ROLLNO IN NUMBER) AS

STUDENT\_NAME VARCHAR2(30);

NUM\_COMPLETED\_COURSES NUMBER;

BEGIN

SELECT NAME INTO STUDENT\_NAME

FROM STUDENT

WHERE ROLLNO = STUDENT\_ROLLNO;

SELECT COUNT(\*)

INTO NUM\_COMPLETED\_COURSES

FROM ENROLLS

WHERE ROLLNO = STUDENT\_ROLLNO

AND GRADE IN (‘S’, ‘A’, ‘B’, ‘C’, ‘D’, ‘E’);

DBMS\_OUTPUT.PUT\_LINE(‘Roll Number: ‘ || STUDENT\_ROLLNO);

DBMS\_OUTPUT.PUT\_LINE(‘Name: ‘ || STUDENT\_NAME);

DBMS\_OUTPUT.PUT\_LINE(‘Number of Courses Completed: ‘ || NUM\_COMPLETED\_COURSES);

END;

/

---------------------------------------------------------------

Procedure DISP\_NOE to accept a CCODE of a COURSE as input and print the roll number and name of students who have enrolled for the course more than twice:

CREATE OR REPLACE PROCEDURE DISP\_NOE(COURSE\_CCODE IN NUMBER) AS

BEGIN

FOR R IN (

SELECT S.ROLLNO, S.NAME

FROM STUDENT S

JOIN ENROLLS E ON S.ROLLNO = E.ROLLNO

WHERE E.CCODE = COURSE\_CCODE

GROUP BY S.ROLLNO, S.NAME

HAVING COUNT(E.CCODE) > 2

) LOOP

DBMS\_OUTPUT.PUT\_LINE(‘Roll Number: ‘ || R.ROLLNO);

DBMS\_OUTPUT.PUT\_LINE(‘Name: ‘ || R.NAME);

END LOOP;

END;

/

---------------------------------------------------------------

### MongoDB Ex-2

To implement CRUD operations in MongoDB for the above relational schema, you would need to design the document structure and define the appropriate collections to represent each entity in the schema. Since MongoDB is a NoSQL database, the schema design and data modeling approach will be different compared to the relational database.

Here's how you can implement CRUD operations in MongoDB for the given relational schema:

1. Connect to the MongoDB server:

const MongoClient = require('mongodb').MongoClient;

const url = 'mongodb://localhost:27017'; // Update with your MongoDB connection URL

MongoClient.connect(url, function(err, client) {

if (err) throw err;

console.log('Connected to MongoDB server');

const db = client.db('examinations'); // Replace 'examinations' with your database name

// Perform CRUD operations here

client.close();

});

-----------------------------------------------------------------------------------------------------------------

2. Create the collections and define the constraints:

db.createCollection('student', {

validator: {

$jsonSchema: {

bsonType: 'object',

required: ['ROLLNO', 'NAME', 'DOB', 'GENDER', 'DOA', 'BCODE'],

properties: {

ROLLNO: {

bsonType: 'int',

description: 'Student Roll Number',

},

NAME: {

bsonType: 'string',

description: 'Student Name',

},

DOB: {

bsonType: 'date',

description: 'Date of Birth',

},

GENDER: {

bsonType: 'string',

description: 'Gender',

enum: ['M', 'F'],

},

DOA: {

bsonType: 'date',

description: 'Date of Admission',

},

BCODE: {

bsonType: 'int',

description: 'Branch Code',

},

},

},

},

});

db.createCollection('branch', {

validator: {

$jsonSchema: {

bsonType: 'object',

required: ['BCODE', 'BNAME', 'DNO'],

properties: {

BCODE: {

bsonType: 'int',

description: 'Branch Code',

},

BNAME: {

bsonType: 'string',

description: 'Branch Name',

},

DNO: {

bsonType: 'int',

description: 'Department Number',

},

},

},

},

validationLevel: 'strict', // Optional, sets the validation level

validationAction: 'error', // Optional, sets the validation action

});

// Similarly, create collections for DEPARTMENT, COURSE, BRANCH\_COURSE, PREREQUISITE\_COURSE, and ENROLLS

-----------------------------------------------------------------------------------------------------------------

Note: The above code snippet demonstrates creating a collection and defining the validation rules using JSON Schema. You can customize the constraints and validation options as per your requirements.

3. CRUD operations:

- Insert a record into the "student" collection:

db.student.insertOne({

ROLLNO: 10001,

NAME: 'John Doe',

DOB: new Date('1995-01-01'),

GENDER: 'M',

DOA: new Date('2023-01-01'),

BCODE: 20001,

});

-----------------------------------------------------------------------------------------------------------------

- Insert a record into the "branch" collection:

db.branch.insertOne({

BCODE: 20001,

BNAME: 'Computer Science',

DNO: 30001,

});

-----------------------------------------------------------------------------------------------------------------

- Insert a record into the "course" collection:

db.course.insertOne({

CCODE: 40001,

CNAME: 'Database Management Systems',

CREDITS: 3,

DNO: 30001,

});

-----------------------------------------------------------------------------------------------------------------

- Insert a record into the "branch\_course" collection:

db.branch\_course.insertOne({

BCODE: 20001,

CCODE: 40001,

SEMEST

ER: 'Spring 2023',

});

-----------------------------------------------------------------------------------------------------------------

- Insert a record into the "prerequisite\_course" collection:

db.prerequisite\_course.insertOne({

CCODE: 40002,

PCCODE: 40001,

});

-----------------------------------------------------------------------------------------------------------------

- Insert a record into the "enrolls" collection:

db.enrolls.insertOne({

ROLLNO: 10001,

CCODE: 40001,

SESS: 'APRIL20201',

GRADE: 'A',

});

-----------------------------------------------------------------------------------------------------------------

- Read records from a collection:

// Find all students

db.student.find().toArray();

// Find all courses offered by a specific department

db.course.find({ DNO: 30001 }).toArray();

// Find enrolled courses for a student

db.enrolls.find({ ROLLNO: 10001 }).toArray();

-----------------------------------------------------------------------------------------------------------------

- Update a record:

// Update the name of a student

db.student.updateOne({ ROLLNO: 10001 }, { $set: { NAME: 'Jane Doe' } });

-----------------------------------------------------------------------------------------------------------------

- Delete a record:

// Delete a student record

db.student.deleteOne({ ROLLNO: 10001 });

-----------------------------------------------------------------------------------------------------------------

These are just basic examples to demonstrate CRUD operations. You can modify them according to your specific needs and add error handling, query conditions, and other necessary fields as required.

# Exercise 3

Consider the following relational schema for a banking database application:

CUSTOMER (CID, CNAME)

ACCOUNT (ANO, ATYPE, BALANCE, CID)

An account can be a savings account or a current account. Check ATYPE in ‘S’ or ‘C’.

A customer can have both types of accounts.

TRANSACTION (TID, ANO, TTYPE, TDATE, TAMOUNT)

TTYPE CAN BE ‘D’ OR ‘W’

D- Deposit; W — Withdrawal

The primary key of each relation is: CUSTOMER(CID),ACCOUNT(ANO),TRANSACTION(TID,ANO)

Perform the following:

a. Develop DDL to implement the above Schema specifying appropriate data each attribute enforcing primary key, check constraints and foreign key constraints.

b. Populate the database with a rich data set.

c. Develop a SQL query to list the details of customers who have a savings account and a current account.

d. Develop a SQL query to list the details of customers who have balance less than the average balance of all customers.

e. Develop a SQL query to list the details of customers with the sum of balance in their account (s)

f. Develop a SQL query to list the details of customers who have performed three

transactions on a day.

g. Create a view that will keep track of customer details and the number of accounts each customer has.

h. Develop a database trigger that will not permit a customer to perform more than three transactions on a day.

i. Develop a database procedure that will accept transaction id, account number,

transaction type as input and insert a record into TRANSACTION table subject to the following conditions:

If TTYPE=‘D’ the value of BALANCE in the ACCOUNT table must

Be incremented by the value of TAMOUNT

II. If TTYPE=‘W’ the value of BALANCE in the ACCOUNT table must

Be decremented by the value of TAMOUNT if a minimum balance of Rs. 2000/- will be maintained for a savings account and a minimum balance of Rs. 5000/- will be maintained for a current account else appropriate messages may be displayed.

Implement CRUD operations in MONGO DB for the above relational uu.

## **SOLUTION**

### MySQL:

DDL to implement the schema in MySQL:

CREATE TABLE CUSTOMER (

CID INT PRIMARY KEY,

CNAME VARCHAR(30)

);

CREATE TABLE ACCOUNT (

ANO INT PRIMARY KEY,

ATYPE ENUM(‘S’, ‘C’),

BALANCE DECIMAL(10, 2),

CID INT,

FOREIGN KEY (CID) REFERENCES CUSTOMER(CID)

);

CREATE TABLE TRANSACTION (

TID INT PRIMARY KEY,

ANO INT,

TTYPE ENUM(‘D’, ‘W’),

TDATE DATE,

TAMOUNT DECIMAL(10, 2),

FOREIGN KEY (ANO) REFERENCES ACCOUNT(ANO)

);

---------------------------------------------------------------

Populating the database with a rich data set:

Inserting sample data into CUSTOMER table

INSERT INTO CUSTOMER (CID, CNAME) VALUES

(1, ‘John Doe’),

(2, ‘Jane Smith’),

(3, ‘Mike Johnson’);

Inserting sample data into ACCOUNT table

INSERT INTO ACCOUNT (ANO, ATYPE, BALANCE, CID) VALUES

(101, ‘S’, 5000.00, 1),

(102, ‘C’, 10000.00, 1),

(103, ‘S’, 3000.00, 2),

(104, ‘C’, 8000.00, 2),

(105, ‘S’, 2000.00, 3);

Inserting sample data into TRANSACTION table

INSERT INTO TRANSACTION (TID, ANO, TTYPE, TDATE, TAMOUNT) VALUES

(1, 101, ‘D’, ‘2023-01-01’, 1000.00),

(2, 101, ‘W’, ‘2023-01-02’, 500.00),

(3, 102, ‘D’, ‘2023-01-01’, 2000.00),

(4, 103, ‘D’, ‘2023-01-01’, 1500.00),

(5, 104, ‘W’, ‘2023-01-02’, 1000.00),

(6, 105, ‘D’, ‘2023-01-02’, 500.00);

---------------------------------------------------------------

SQL query to list the details of customers who have a savings account and a current account:

SELECT C.CID, C.CNAME

FROM CUSTOMER C

INNER JOIN ACCOUNT A1 ON C.CID = A1.CID

INNER JOIN ACCOUNT A2 ON C.CID = A2.CID

WHERE A1.ATYPE = ‘S’ AND A2.ATYPE = ‘C’;

---------------------------------------------------------------

SQL query to list the details of customers who have a balance less than the average balance of all customers:

SELECT C.CID, C.CNAME

FROM CUSTOMER C

INNER JOIN ACCOUNT A ON C.CID = A.CID

WHERE A.BALANCE < (SELECT AVG(BALANCE) FROM ACCOUNT);

---------------------------------------------------------------

SQL query to list the details of customers with the sum of balance in their account(s):

SELECT C.CID, C.CNAME, SUM(A.BALANCE) AS TOTAL\_BALANCE

FROM CUSTOMER C

INNER JOIN ACCOUNT A ON C.CID = A.CID

GROUP BY C.CID, C.CNAME;

---------------------------------------------------------------

SQL query to list the details of customers who have performed three transactions on a day:

SELECT C.CID, C.CNAME

FROM CUSTOMER C

INNER JOIN ACCOUNT A ON C.CID = A.CID

INNER JOIN TRANSACTION T ON A.ANO = T.ANO

WHERE T.TDATE = ‘2023-01

-01’

GROUP BY C.CID, C.CNAME

HAVING COUNT(T.TID) = 3;

---------------------------------------------------------------

Creating a view to keep track of customer details and the number of accounts each customer has:

CREATE VIEW CUSTOMER\_ACCOUNT\_COUNT AS

SELECT C.CID, C.CNAME, COUNT(A.ANO) AS ACCOUNT\_COUNT

FROM CUSTOMER C

LEFT JOIN ACCOUNT A ON C.CID = A.CID

GROUP BY C.CID, C.CNAME;

---------------------------------------------------------------

Database trigger to not permit a customer to perform more than three transactions on a day:

DELIMITER //

CREATE TRIGGER limit\_transaction

BEFORE INSERT ON TRANSACTION

FOR EACH ROW

BEGIN

DECLARE transaction\_count INT;

SELECT COUNT(\*)

INTO transaction\_count

FROM TRANSACTION

WHERE ANO = NEW.ANO AND TDATE = NEW.TDATE;

IF transaction\_count >= 3 THEN

SIGNAL SQLSTATE ‘45000’

SET MESSAGE\_TEXT = ‘Maximum transaction limit reached for the day.’;

END IF;

END //

DELIMITER ;

---------------------------------------------------------------

Database procedure to insert a record into the TRANSACTION table with specific conditions:

DELIMITER //

CREATE PROCEDURE INSERT\_TRANSACTION(

IN p\_TID INT,

IN p\_ANO INT,

IN p\_TTYPE CHAR(1),

IN p\_TAMOUNT DECIMAL(10, 2)

)

BEGIN

DECLARE v\_BALANCE DECIMAL(10, 2);

SELECT BALANCE

INTO v\_BALANCE

FROM ACCOUNT

WHERE ANO = p\_ANO;

IF p\_TTYPE = ‘D’ THEN

UPDATE ACCOUNT

SET BALANCE = BALANCE + p\_TAMOUNT

WHERE ANO = p\_ANO;

ELSEIF p\_TTYPE = ‘W’ THEN

IF (v\_BALANCE – p\_TAMOUNT) >= 2000 AND (v\_BALANCE – p\_TAMOUNT) >= 5000 THEN

UPDATE ACCOUNT

SET BALANCE = BALANCE – p\_TAMOUNT

WHERE ANO = p\_ANO;

ELSE

SIGNAL SQLSTATE ‘45000’

SET MESSAGE\_TEXT = ‘Minimum balance not maintained.’;

END IF;

END IF;

INSERT INTO TRANSACTION (TID, ANO, TTYPE, TDATE, TAMOUNT)

VALUES (p\_TID, p\_ANO, p\_TTYPE, CURDATE(), p\_TAMOUNT);

END //

DELIMITER ;

---------------------------------------------------------------

### PostgreSQL:

DDL to implement the schema with appropriate constraints:

CREATE TABLE CUSTOMER (

CID INT PRIMARY KEY,

CNAME VARCHAR(30)

);

CREATE TABLE ACCOUNT (

ANO INT PRIMARY KEY,

ATYPE CHAR(1) CHECK (ATYPE IN (‘S’, ‘C’)),

BALANCE DECIMAL(10, 2),

CID INT,

FOREIGN KEY (CID) REFERENCES CUSTOMER (CID)

);

CREATE TABLE TRANSACTION (

TID INT PRIMARY KEY,

ANO INT,

TTYPE CHAR(1) CHECK (TTYPE IN (‘D’, ‘W’)),

TDATE DATE,

TAMOUNT DECIMAL(10, 2),

FOREIGN KEY (ANO) REFERENCES ACCOUNT (ANO)

);

---------------------------------------------------------------

Populate the database with a rich data set:

INSERT INTO CUSTOMER (CID, CNAME) VALUES

(1, ‘John Doe’),

(2, ‘Jane Smith’),

(3, ‘Michael Johnson’);

INSERT INTO ACCOUNT (ANO, ATYPE, BALANCE, CID) VALUES

(101, ‘S’, 5000.00, 1),

(102, ‘C’, 10000.00, 1),

(201, ‘S’, 3000.00, 2),

(202, ‘C’, 15000.00, 2),

(301, ‘S’, 8000.00, 3),

(302, ‘C’, 12000.00, 3);

INSERT INTO TRANSACTION (TID, ANO, TTYPE, TDATE, TAMOUNT) VALUES

(1, 101, ‘D’, ‘2023-01-01’, 1000.00),

(2, 101, ‘W’, ‘2023-01-02’, 500.00),

(3, 102, ‘D’, ‘2023-01-03’, 2000.00),

(4, 201, ‘D’, ‘2023-01-01’, 1500.00),

(5, 301, ‘W’, ‘2023-01-02’, 1000.00),

(6, 302, ‘D’, ‘2023-01-03’, 3000.00);

---------------------------------------------------------------

SQL query to list details of customers with savings and current accounts:

SELECT C.CID, C.CNAME

FROM CUSTOMER C

WHERE EXISTS (

SELECT 1

FROM ACCOUNT A

WHERE A.CID = C.CID AND A.ATYPE = ‘S’

) AND EXISTS (

SELECT 1

FROM ACCOUNT A

WHERE A.CID = C.CID AND A.ATYPE = ‘C’

);

---------------------------------------------------------------

SQL query to list details of customers with balance less than average balance:

SELECT C.CID, C.CNAME

FROM CUSTOMER C

JOIN ACCOUNT A ON C.CID = A.CID

GROUP BY C.CID, C.CNAME

HAVING AVG(A.BALANCE) > A.BALANCE;

---------------------------------------------------------------

SQL query to list details of customers with the sum of balance in their accounts:

SELECT C.CID, C.CNAME, SUM(A.BALANCE) AS TOTAL\_BALANCE

FROM CUSTOMER C

JOIN ACCOUNT A ON C.CID = A.CID

GROUP BY C.CID, C.CNAME;

---------------------------------------------------------------

SQL query to list details of customers with three transactions on a day:

SELECT C.CID

, C.CNAME

FROM CUSTOMER C

JOIN ACCOUNT A ON C.CID = A.CID

JOIN TRANSACTION T ON A.ANO = T.ANO

WHERE T.TDATE = ‘2023-01-03’

GROUP BY C.CID, C.CNAME

HAVING COUNT(T.TID) = 3;

---------------------------------------------------------------

Create a view to track customer details and the number of accounts:

CREATE VIEW CUSTOMER\_ACCOUNTS AS

SELECT C.CID, C.CNAME, COUNT(A.ANO) AS NUM\_ACCOUNTS

FROM CUSTOMER C

LEFT JOIN ACCOUNT A ON C.CID = A.CID

GROUP BY C.CID, C.CNAME;

---------------------------------------------------------------

PostgreSQL does not support triggers on select statements, so it’s not possible to develop a trigger for the given requirement.

PostgreSQL does not support the creation of stored procedures. Instead, you can use functions. Here’s an example of a function that inserts a record into the TRANSACTION table:

CREATE FUNCTION INSERT\_TRANSACTION(

P\_tid INT,

P\_ano INT,

P\_ttype CHAR(1),

P\_tdate DATE,

P\_tamount DECIMAL(10, 2)

)

RETURNS VOID AS

$$

BEGIN

IF p\_ttype = ‘D’ THEN

UPDATE ACCOUNT SET BALANCE = BALANCE + p\_tamount WHERE ANO = p\_ano;

ELSIF p\_ttype = ‘W’ THEN

UPDATE ACCOUNT SET BALANCE = BALANCE – p\_tamount WHERE ANO = p\_ano;

Add additional logic to check minimum balance and display appropriate messages.

END IF;

INSERT INTO TRANSACTION (TID, ANO, TTYPE, TDATE, TAMOUNT)

VALUES (p\_tid, p\_ano, p\_ttype, p\_tdate, p\_tamount);

END;

$$

LANGUAGE plpgsql;

---------------------------------------------------------------

PostgreSQL does not support triggers on select statements, so it’s not possible to develop a trigger for the given requirement.

### Oracle:

DDL to implement the schema with appropriate constraints:

CREATE TABLE CUSTOMER (

CID NUMBER PRIMARY KEY,

CNAME VARCHAR2(30)

);

CREATE TABLE ACCOUNT (

ANO NUMBER PRIMARY KEY,

ATYPE CHAR(1) CHECK (ATYPE IN (‘S’, ‘C’)),

BALANCE NUMBER(10, 2),

CID NUMBER,

FOREIGN KEY (CID) REFERENCES CUSTOMER (CID)

);

CREATE TABLE TRANSACTION (

TID NUMBER PRIMARY KEY,

ANO NUMBER,

TTYPE CHAR(1) CHECK (TTYPE IN (‘D’, ‘W’)),

TDATE DATE,

TAMOUNT NUMBER(10, 2),

FOREIGN KEY (ANO) REFERENCES ACCOUNT (ANO)

);

---------------------------------------------------------------

Populate the database with a rich data set:

INSERT INTO CUSTOMER (CID, CNAME) VALUES

(1, ‘John Doe’),

(2, ‘Jane Smith’),

(3, ‘Michael Johnson’);

INSERT INTO ACCOUNT (ANO, ATYPE, BALANCE, CID) VALUES

(101, ‘S’, 5000.00, 1),

(102, ‘C’, 10000.00, 1),

(201, ‘S’, 3000.00, 2),

(202, ‘C’, 15000.00, 2),

(301, ‘S’, 8000.00, 3),

(302, ‘C’, 12000.00, 3);

INSERT INTO TRANSACTION (TID, ANO, TTYPE, TDATE, TAMOUNT) VALUES

(1, 101,

‘D’, TO\_DATE(‘2023-01-01’, ‘YYYY-MM-DD’), 1000.00),

(2, 101, ‘W’, TO\_DATE(‘2023-01-02’, ‘YYYY-MM-DD’), 500.00),

(3, 102, ‘D’, TO\_DATE(‘2023-01-03’, ‘YYYY-MM-DD’), 2000.00),

(4, 201, ‘D’, TO\_DATE(‘2023-01-01’, ‘YYYY-MM-DD’), 1500.00),

(5, 301, ‘W’, TO\_DATE(‘2023-01-02’, ‘YYYY-MM-DD’), 1000.00),

(6, 302, ‘D’, TO\_DATE(‘2023-01-03’, ‘YYYY-MM-DD’), 3000.00);

---------------------------------------------------------------

SQL query to list details of customers with savings and current accounts:

SELECT C.CID, C.CNAME

FROM CUSTOMER C

WHERE EXISTS (

SELECT 1

FROM ACCOUNT A

WHERE A.CID = C.CID AND A.ATYPE = ‘S’

) AND EXISTS (

SELECT 1

FROM ACCOUNT A

WHERE A.CID = C.CID AND A.ATYPE = ‘C’

);

---------------------------------------------------------------

SQL query to list details of customers with balance less than average balance:

SELECT C.CID, C.CNAME

FROM CUSTOMER C

JOIN ACCOUNT A ON C.CID = A.CID

GROUP BY C.CID, C.CNAME

HAVING AVG(A.BALANCE) > A.BALANCE;

---------------------------------------------------------------

SQL query to list details of customers with the sum of balance in their accounts:

SELECT C.CID, C.CNAME, SUM(A.BALANCE) AS TOTAL\_BALANCE

FROM CUSTOMER C

JOIN ACCOUNT A ON C.CID = A.CID

GROUP BY C.CID, C.CNAME;

---------------------------------------------------------------

SQL query to list details of customers with three transactions on a day:

SELECT C.CID, C.CNAME

FROM CUSTOMER C

JOIN ACCOUNT A ON C.CID = A.CID

JOIN TRANSACTION T ON A.ANO = T.ANO

WHERE T.TDATE = TO\_DATE(‘2023-01-03’, ‘YYYY-MM-DD’)

GROUP BY C.CID, C.CNAME

HAVING COUNT(T.TID) = 3;

---------------------------------------------------------------

Create a view to track customer details and the number of accounts:

CREATE VIEW CUSTOMER\_ACCOUNTS AS

SELECT C.CID, C.CNAME, COUNT(A.ANO) AS NUM\_ACCOUNTS

FROM CUSTOMER C

LEFT JOIN ACCOUNT A ON C.CID = A.CID

GROUP BY C.CID, C.CNAME;

---------------------------------------------------------------

Oracle does not support triggers on select statements, so it’s not possible to develop a trigger for the given requirement.

Oracle does support stored procedures. Here’s an example of a procedure that inserts a record into the TRANSACTION table:

CREATE OR REPLACE PROCEDURE INSERT\_TRANSACTION(

P\_tid IN NUMBER,

P\_ano IN NUMBER,

P\_ttype IN CHAR,

P\_tdate IN DATE,

P\_tamount IN NUMBER

)

IS

BEGIN

IF p\_ttype = ‘D’ THEN

UPDATE ACCOUNT SET BALANCE = BALANCE + p\_tamount WHERE ANO = p\_ano;

ELSIF p\_ttype = ‘W’ THEN

UPDATE ACCOUNT SET BALANCE = BALANCE – p\_tamount WHERE ANO = p\_ano;

Add additional logic to check minimum balance and display Appropriate messages.

END IF;

INSERT INTO TRANSACTION (TID, ANO, TTYPE, TDATE, TAMOUNT)

VALUES (p\_tid, p\_ano, p\_ttype, p\_tdate, p\_tamount);

COMMIT;

END;

/

---------------------------------------------------------------

Oracle does not support triggers on select statements, so it’s not possible to develop a trigger for the given requirement.

### MongoDB Ex-3

Sure! Here's how you can implement CRUD operations in MongoDB for the given relational schema:

1. Connect to the MongoDB server:

const MongoClient = require('mongodb').MongoClient;

const url = 'mongodb://localhost:27017'; // Update with your MongoDB connection URL

MongoClient.connect(url, function(err, client) {

if (err) throw err;

console.log('Connected to MongoDB server');

const db = client.db('banking'); // Replace 'banking' with your database name

// Perform CRUD operations here

client.close();

});

-----------------------------------------------------------------------------------------------------------------

2. Create the collections and define the constraints:

db.createCollection('customer', {

validator: {

$jsonSchema: {

bsonType: 'object',

required: ['CID', 'CNAME'],

properties: {

CID: {

bsonType: 'int',

description: 'Customer ID',

},

CNAME: {

bsonType: 'string',

description: 'Customer Name',

},

},

},

},

});

db.createCollection('account', {

validator: {

$jsonSchema: {

bsonType: 'object',

required: ['ANO', 'ATYPE', 'BALANCE', 'CID'],

properties: {

ANO: {

bsonType: 'int',

description: 'Account Number',

},

ATYPE: {

enum: ['S', 'C'],

description: 'Account Type (Savings or Current)',

},

BALANCE: {

bsonType: 'double',

minimum: 0,

description: 'Account Balance',

},

CID: {

bsonType: 'int',

description: 'Customer ID',

},

},

},

},

});

db.createCollection('transaction', {

validator: {

$jsonSchema: {

bsonType: 'object',

required: ['TID', 'ANO', 'TTYPE', 'TDATE', 'TAMOUNT'],

properties: {

TID: {

bsonType: 'int',

description: 'Transaction ID',

},

ANO: {

bsonType: 'int',

description: 'Account Number',

},

TTYPE: {

enum: ['D', 'W'],

description: 'Transaction Type (Deposit or Withdrawal)',

},

TDATE: {

bsonType: 'date',

description: 'Transaction Date',

},

TAMOUNT: {

bsonType: 'double',

minimum: 0,

description: 'Transaction Amount',

},

},

},

},

});

-----------------------------------------------------------------------------------------------------------------

Note: The above code snippet demonstrates creating a collection and defining the validation rules using JSON Schema. You can customize the constraints and validation options as per your requirements.

3. CRUD operations:

- Insert a record into the "customer" collection:

db.customer.insertOne({

CID: 1001,

CNAME: 'John Doe',

});

-----------------------------------------------------------------------------------------------------------------

- Insert a record into the "account" collection:

db.account.insertOne({

ANO: 2001,

ATYPE: 'S',

BALANCE: 1000,

CID: 1001,

});

-----------------------------------------------------------------------------------------------------------------

- Insert a record into the "transaction" collection:

db.transaction.insertOne({

TID: 3001,

ANO: 2001,

TTYPE: 'D',

TDATE: new Date(),

TAMOUNT: 500,

});

-----------------------------------------------------------------------------------------------------------------

- Read records from a collection:

// Find all customers

db.customer.find().toArray();

// Find all accounts of a specific customer

db.account.find({ CID:

1001 }).toArray();

// Find transactions for an account

db.transaction.find({ ANO: 2001 }).toArray();

-----------------------------------------------------------------------------------------------------------------

- Update a record:

// Update the balance of an account

db.account.updateOne({ ANO: 2001 }, { $inc: { BALANCE: -200 } });

-----------------------------------------------------------------------------------------------------------------

- Delete a record:

// Delete a customer record

db.customer.deleteOne({ CID: 1001 });

-----------------------------------------------------------------------------------------------------------------

These are basic examples to demonstrate CRUD operations. You can modify them according to your specific needs and add error handling, query conditions, and other necessary fields as required.