CMR
INSTITUTE
OF TECHNOLOGY

SCHEME OF EVALUATION

Internal Assessment Test 2– July 2024

Sub:	**DATABASE MANAGEMENT SYSTEMS**	**Code:**	BCS403
Date: 10/7/2024 Duration: 90 mins Max Marks: 50	**Sem:** **4**	**Branch:**	AIML

Note: ***Answer any five full questions.*** ***(5 X 10 = 50)***

Question No.		**Description**	**Marks Split up**		**Total Marks**
1.	a	✓ Informal design guidlines	6	6M	10M
	b	✓ Trivial and non trivial ✓ Prime and non prime	2 2	4M	
2.	a)	✓ Explation of 1NF,2NF,3NF ✓ Finding candidate key ✓ Finding normal form	6 2 2	10M	10M
3.	a)	✓ syntax of SELECT command with 6 clauses.	6	10M	10M
	b)	aggregate functions in SQL 2 marks Importance aggregate functions in SQL 2 marks	4		
4.	A)	5 Diffrence between nested query and correlated query	5	10M	10M
	B)	Defination of view (1 M) Syntax(2 M) Example (2M)	5		

5.	a)	Defination (2M)
		Syntax(2M)
		Example(2M)
			6M	10M	10M

	b)	3 valued logic	4M		
6.		✓ Each query with 2 marks	2M*5		10M

USN | | | | | | | | | | |

CMRIT
CMR INSTITUTE OF TECHNOLOGY, BENGALURU.
ACCREDITED WITH A++ GRADE BY NAAC

Internal Assessment Test 2 – July 2024

Sub:	**Database Management System**					Sub Code:	**BCS403**	Branch:	**AIML /CSE(AIML)**
Date:	**10/07/24**	Duration:	**90 minutes**	Max Marks:	**50**	Sem/Sec:	**IV**		**OBE**

		Answer any FIVE FULL Questions	**MARKS**	**CO**	**RBT**
1	a	Explain the informal design guidelines used as measures to determine the quality of relation schema design.	6	3	L2
	b	Differentiate between the following by taking suitable examples, i)Trivial and Non-Trivial Dependency ii)Prime and Non-prime attributes	4	3	L2
2	a	Explain 1NF,2NF and 3NF.Consider the relation schema R (A, B, C, D, E, F, G, H, I,J) and the functional dependencies {AB->C, A->DE, B->F, F->GH, D->IJ}. Determine the candidate key and the highest normal form for the above relation.	10	3	L3
3	a	Explain the syntax of SELECT command with 6 clauses.	6	3	L2
	b	What are the aggregate functions in SQL and its need?	4	3	L1
4	a	Differentiate between nested query and correlated query with suitable examples.	5	4	L2
	b	What is a view in SQL? How to create a view in SQL?	5	4	L1
5	a	Explain Specifying Constraints as Assertions in SQL with an example.	6	4	L2
	b	Explain three-valued logic in SQL.	4	4	L2
6	a	Consider the following relation schema and write SQL queries, employee (person-name, street, city) works (person-name, company-name, salary) company (company-name, city) i. Find the names, street address, and cities of residence for all employees who work for 'First Bank Corporation' and earn more than Rs.10,000. ii. Find the names of all employees in the database who do not work for 'First Bank Corporation'. iii. Find the names of all employees in the database who earn more than every employee of 'Small Bank Corporation'. iv. Update the salary of people working in ‘Infosys’ by 15%. v. Display company wise, average salary and maximum salary paid to the employees.	10	4	L3

Q1. Explain the informal design guidelines used as measures to determine the quality of
relation schema design.

Ans. **1 Informal Design Guidelines for Relational Databases**

1.1Semantics of the Relation Attributes

1.2 Redundant Information in Tuples and Update Anomalies

1.3 Null Values in Tuples

1.4 Spurious Tuples

1.1 Semantics of the Relation Attributes

Informally, each tuple in a relation should represent one entity or relationship instance. (Applies to individual relations and their attributes).

- Attributes of different entities (EMPLOYEEs, DEPARTMENTs, PROJECTs) should not be mixed in the same relation
- Only foreign keys should be used to refer to other entities
- Entity and relationship attributes should be kept apart as much as possible.

1.2 Redundant Information in Tuples and Update Anomalies

- Mixing attributes of multiple entities may cause problems
- Information is stored redundantly wasting storage
- Problems with update anomalies
  - Insertion anomalies
  - Deletion anomalies

    Modification anomalies

1.3 Null Values in Tuples

- Attributes that are NULL frequently could be placed in separate relations (with the primary key)
- Reasons for nulls:
  - attribute not applicable or invalid
  - attribute value unknown (may exist)
  - value known to exist, but unavailable

1.4 Spurious Tuples

- Bad designs for a relational database may result in erroneous results for certain JOIN operations
- The "lossless join" property is used to guarantee meaningful results for join operations

There are two important properties of decompositions:

(a) non-additive or losslessness of the corresponding join

(b) preservation of the functional dependencies.

- **GUIDELINE 1:** Design a schema that can be explained easily relation by relation. The semantics of attributes should be easy to interpret.
- **GUIDELINE 2:** Design a schema that does not suffer from the insertion, deletion and update anomalies. If there are any present, then note them so that applications can be made to take them into account
- **GUIDELINE 3:** Relations should be designed such that their tuples will have as few NULL values as possible
- **GUIDELINE 4:** The relations should be designed to satisfy the lossless join condition. No spurious tuples should be generated by doing a natural-join of any relations.

Q2. Differentiate between the following by taking suitable examples,

i)Trivial and Non-Trivial Dependency

ii)Prime and Non-prime attributes

Ans.) Trivial and Non-Trivial Dependency

Trivial Dependency:

- A functional dependency is considered trivial if it is satisfied by all relations.
- This occurs when an attribute (or a set of attributes) functionally determines itself or a subset of itself.

Example:

- Consider a relation R(A,B)R(A, B)R(A,B) and a functional dependency A→AA \rightarrow AA→A. This is a trivial dependency because AAA obviously determines itself.

Non-Trivial Dependency:

- A functional dependency is considered non-trivial if it is not automatically satisfied by all relations.
- This occurs when an attribute (or a set of attributes) functionally determines another attribute (or set of attributes) that is not a subset of the original.

Example:

- Consider a relation R(A,B)R(A, B)R(A,B) and a functional dependency A→BA \rightarrow BA→B. This is a non-trivial dependency because AAA does not trivially determine BBB; this dependency must be explicitly enforced.

ii) Prime and Non-Prime Attributes

Prime Attribute:

- An attribute that is part of any candidate key of a relation.
- Candidate keys are the minimal sets of attributes that can uniquely identify a tuple in a relation.

Example:

- Consider a relation R(A,B,C)R(A, B, C)R(A,B,C) with candidate keys (A,B)(A, B)(A,B) and (A,C)(A, C)(A,C). Here, AAA, BBB, and CCC are all prime attributes because they are part of some candidate key.

Non-Prime Attribute:

- An attribute that is not part of any candidate key of a relation.
- These attributes do not play a role in uniquely identifying a tuple in the relation.

Example:

- Consider a relation R(A,B,C,D)R(A, B, C, D)R(A,B,C,D) with a candidate key (A,B)(A, B)(A,B). Here, AAA and BBB are prime attributes, while CCC and DDD are non-prime attributes because they are not part of any candidate key.

2. Explain 1NF,2NF and 3NF.Consider the relation schema R (A, B, C, D, E, F, G, H, I,J) and the functional dependencies

{AB->C, A->DE, B->F, F->GH, D->IJ}. Determine the candidate key and the highest normal form for the above relation.

Ans. Explanation of Normal Forms

1NF (First Normal Form)

A relation is in First Normal Form if:

- All the values in the relation are atomic (indivisible).
- Each column contains unique values.
- There are no repeating groups or arrays.

Example: If a table contains a list of phone numbers for a person in a single column, it is not in 1NF. To convert it to 1NF, each phone number should be in a separate row or a separate column.

2NF (Second Normal Form)

A relation is in Second Normal Form if:

- It is in 1NF.
- All non-prime attributes are fully functionally dependent on the entire primary key (no partial dependency).

Example: Consider a table with attributes {StudentID, CourseID, StudentName}. If the primary key is {StudentID, CourseID}, and StudentName is dependent only on StudentID, it is a partial dependency and violates 2NF. To achieve 2NF, StudentName should be moved to a separate table where StudentID is the primary key.

3NF (Third Normal Form)

A relation is in Third Normal Form if:

- It is in 2NF.
- There are no transitive dependencies, meaning non-prime attributes are not dependent on other non-prime attributes.

Example:

Consider a table with attributes {StudentID, CourseID, ProfessorID, ProfessorName}. If the primary key is {StudentID, CourseID}, and ProfessorName is dependent on ProfessorID (a non-prime attribute), it violates 3NF. To achieve 3NF, ProfessorName should be moved to a separate table where ProfessorID is the primary key.

Q3. Explain the syntax of SELECT command with 6 clauses.

Ans. SQL SELECT has different clauses to manage the data output. They are: FROM, AS, GROUP BY, HAVING, INTO, ORDER BY, * (asterisk). Let's see how we can use each clause within the SELECT syntax.

1. FROM is used to specify a table name where a necessary column with data is located.

Syntax:SELECT <column><table>;

2. AS is used to create a temporary name for the column headings. This method lets create more clear column headings. AS is optional and can be present in the query for readability purposes.

Syntax:

```
SELECT <column> [ AS <new_column> ] FROM <table> [ AS <new_table> ];
```

OR

```
SELECT <column> [ <new_column> ]FROM <table> [ <new_table> ];
```

3. GROUP BY is used to group results with similar data. There are some important things you should know about the clause:

- GROUP BY displays one record for each group.
- GROUP BY is used with aggregate functions COUNT, MAX, MIN, SUM, AVG etc.
- GROUP BY follows the WHERE clause, but precedes the ORDER BY clause in a query.

Syntax:

```
SELECT <column1>, SUM(<column2>) FROM <table> GROUP BY <grouping_column>;
```

4. HAVING is used to define a search condition. The clause is used in combination with GROUP BY.

Syntax:

```
SELECT <column1>, SUM(<column2>)FROM <table>GROUP BY <grouping_column>HAVING <condition>;
```

5. INTO is used to create a new table and copy the retrieved results into it.

Syntax:

```
SELECT <column> INTO <new_table>FROM <table>WHERE <condition>;
```

6. ORDER BY is used to filter retrieved results. The sorting can be organized in ascending (a default one) and descending order.

Syntax:

```
SELECT <column1>FROM <table>ORDER BY <column2>;
```

Q3b. What are the aggregate functions in SQL and its need?

Ans. An aggregate function is a function that performs a calculation on a set of values, and returns a single value.

Aggregate functions are often used with the GROUP BY clause of the SELECT statement. The GROUP BY clause splits the result-set into groups of values and the aggregate function can be used to return a single value for each group.

The most commonly used SQL aggregate functions are:

- MIN() - returns the smallest value within the selected column
- MAX() - returns the largest value within the selected column
- COUNT() - returns the number of rows in a set
- SUM() - returns the total sum of a numerical column
- AVG() - returns the average value of a numerical column

Aggregate functions ignore null values (except for COUNT()).

Q4a. Differentiate between nested query and correlated query with suitable examples.

Ans. **Nested Subqueries**
A subquery is nested when you are having a subquery in the where or having clause of another subquery.

Get the result of all the students who are enrolled in the same course as the student with ROLLNO 12.

```
Select *
From result
where rollno in (select rollno
            from student
            where courseid = (select courseid
                              from student
                              where rollno = 12));
```

The innermost subquery will be executed first and then based on its result the next subquery will be executed and based on that result the outer query will be executed. The levels to which you can do the nesting is implementation-dependent.

Correlated Subquery

A **Correlated Subquery** is one that is executed after the outer query is executed. So correlated subqueries take an approach opposite to that of normal subqueries. The correlated subquery execution is as follows:

-The outer query receives a row.
-For each candidate row of the outer query, the subquery (the correlated subquery) is executed once.
-The results of the correlated subquery are used to determine whether the candidate row should be part of the result set.
-The process is repeated for all rows.

Correlated Subqueries differ from the normal subqueries in that the nested SELECT statement referes back to the table in the first SELECT statement.

To find out the names of all the students who appeared in more than three papers of their opted course, the SQL will be

```
Select name
from student A
Where 3 < (select count (*)
      from result b
      where b.rollno = a.rollno);
```

Q4(b). What is a view in SQL? How to create a view in SQL?

Ans. In SQL, a view is a virtual table based on the result-set of an SQL statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

You can add SQL statements and functions to a view and present the data as if the data were coming from one single table.

A view is created with the CREATE VIEW statement.

CREATE VIEW Syntax

CREATE VIEW *view_name* AS SELECT *column1*, *column2*, ...FROM *table_name* WHERE *condition*;

SQL Updating a View

A view can be updated with the CREATE OR REPLACE VIEW statement.

SQL CREATE OR REPLACE VIEW Syntax

CREATE OR REPLACE VIEW *view_name* AS SELECT *column1*, *column2*, ...
FROM *table_name* WHERE *condition*;

SQL Dropping a View

A view is deleted with the DROP VIEW statement.

SQL DROP VIEW Syntax

DROP VIEW *view_name*;

Q5(a) Explain Specifying Constraints as Assertions in SQL with an example.
Ans. When a constraint involves 2 (or) more tables, the table constraint mechanism is sometimes hard and results may not come as expected. To cover such situation SQL supports the creation of assertions that are constraints not associated with only one table. And an assertion statement should ensure a certain condition will always exist in the database. DBMS always checks the assertion whenever modifications are done in the corresponding table.
Syntax –

```
CREATE ASSERTION  [ assertion_name ]

CHECK ( [ condition ] );
```

Example –

```
CREATE TABLE sailors (sid int,sname varchar(20), rating int,primary key(sid),

CHECK(rating >= 1 AND rating <=10)

CHECK((select count(s.sid) from sailors s) + (select count(b.bid)from boats b)<100) );
```

In the above example, we enforcing CHECK constraint that the number of boats and sailors should be less than 100. So here we are able to CHECK constraints of two tablets simultaneously

Q5(b) Explain three-valued logic in SQL.
Ans. n SQL there may be some records in a table that do not have values or data for every field and those fields are termed as a NULL value.

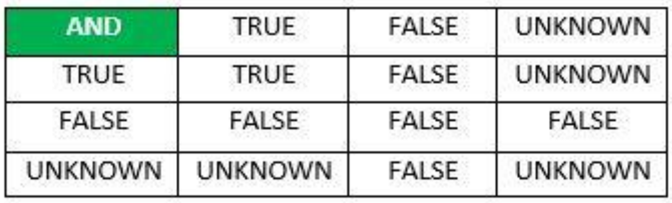
NULL values could be possible because at the time of data entry information is not available. So SQL supports a special value known as NULL which is used to represent the values of attributes that may be unknown or not apply to a tuple. SQL places a NULL value in the field in the absence of a user-defined value. For example, the Apartment_number attribute of an address applies only to addresses that are in apartment buildings and not to other types of residences.
So, **NULL** values are those values in which there is no data value in the particular field in the table.
Importance of NULL Value

- It is important to understand that a NULL value differs from a zero value.
- A NULL value is used to represent a missing value, but it usually has one of three different interpretations:
    - The value unknown (value exists but is not known)
    - Value not available (exists but is purposely withheld)
    - Attribute not applicable (undefined for this tuple)
- It is often not possible to determine which of the meanings is intended. Hence, SQL does not distinguish between the different meanings of NULL.

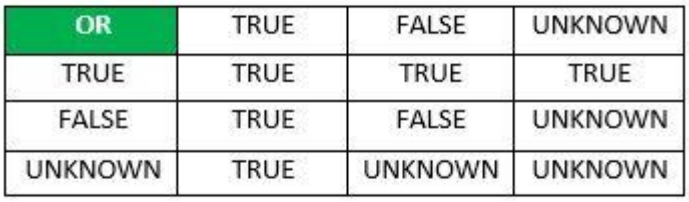
Principles of NULL values

- Setting a NULL value is appropriate when the actual value is unknown, or when a value is not meaningful.
- A NULL value is not equivalent to a value of ZERO if the data type is a number and is not equivalent to spaces if the data type is a character.
- A NULL value can be inserted into columns of any data type.
- A NULL value will evaluate NULL in any expression.
- Suppose if any column has a NULL value, then UNIQUE, FOREIGN key, and CHECK constraints will ignore by SQL.

In general, each NULL value is considered to be different from every other NULL in the database. When a NULL is involved in a comparison operation, the result is considered to be UNKNOWN. Hence, SQL uses a three-valued logic with values **True, False**, and **Unknown.** It is, therefore, necessary to define the results of three-valued logical expressions when the logical connectives AND, OR, and NOT are used.



AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN



OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

Q6. Consider the following relation schema and write SQL queries,
employee (person-name, street, city)
works (person-name, company-name, salary)
company (company-name, city)

1.Find the names, street address, and cities of residence for all employees who work for 'First Bank Corporation' and earn more than Rs.10,000.

2.Find the names of all employees in the database who do not work for 'First Bank Corporation'.

3.Find the names of all employees in the database who earn more than every employee of 'Small Bank Corporation'.
4.Update the salary of people working in ‘Infosys’ by 15%.
5.Display company wise, average salary and maximum salary paid to the employees.

Ans. 1. SELECT e.person_name, e.street, e.city FROM employee e JOIN works w ON e.person_name = w.person_name WHERE w.company_name = 'First Bank Corporation' AND w.salary > 10000;

2. SELECT DISTINCT e.person_name FROM employee e WHERE e.person_name NOT IN (SELECT w.person_name FROM works w WHERE w.company_name = 'First Bank Corporation');

3. SELECT e.person_name FROM works e WHERE e.salary > (SELECT MAX(w.salary)FROM works w
WHERE w.company_name = 'Small Bank Corporation');
4. UPDATE works SET salary = salary * 1.15 WHERE company_name = 'Infosys';
5. SELECT company_name, AVG(salary) AS average_salary, MAX(salary) AS max_salary FROM works
GROUP BY company_name;