R.M.K GROUP OF ENGINEERING INSTITUTIONS

R.M.K GROUP OF INSTITUTIONS

R.M.K
GROUP OF INSTITUTIONS

R.M.K GROUP OF INSTITUTIONS

Please read this disclaimer before proceeding:

This document is confidential and intended solely for the educational purpose of RMK Group of Educational Institutions. If you have received this document through email in error, please notify the system manager. This document contains proprietary information and is intended only to the respective group / learning community as intended. If you are not the addressee you should not disseminate, distribute or copy through e-mail. Please notify the sender immediately by e-mail if you have received this document by mistake and delete this document from your system. If you are not the intended recipient you are notified that disclosing, copying, distributing or taking any action in reliance on the contents of this information is strictly prohibited.

24CS303

DATABASE MANAGEMENT SYSTEMS

Department: All Branches of B.E. / B. Tech.

Batch / Year / Sem : 2024 – 2028 / II / III

Created By : All the Subject Handling Faculty Members of RMK Group

Date: 16.05.2025

TABLE OF CONTENTS

1. COURSE OBJECTIVES

- To understand the basic concepts of Data modeling and Database Systems.
- To understand SQL and effective relational database design concepts.
- To learn relational algebra, calculus and normalization
- To know the fundamental concepts of transaction processing, concurrency control techniques, recovery procedure and data storage techniques.
- To understand query processing, efficient data querying and advanced databases.

R.M.K GROUP OF INSTITUTIONS

2. PRE REQUISITES

- **24CS101 Programming in C++ (Lab Integrated)**
- **24CS202 Java Programming (Lab Integrated)**

3. SYLLABUS

24CS303 - DATABASE MANAGEMENT SYSTEMS

UNIT I - DATABASE CONCEPTS 9+6

Concept of Database and Overview of DBMS - Characteristics of databases -Data Models, Schemas and Instances - Three-Schema Architecture - Database Languages and Interfaces- Introductions to data models types- ER Model- ER Diagrams – Enhanced ER Model - reducing ER to table Applications: ER model of University Database Application – Relational Database Design by ER- and EER-to-Relational Mapping.

List of Exercise/Experiments

Case Study using real life database applications anyone from the following list

a) Inventory Management for a EMart Grocery Shop
b) Society Financial Management
c) Cop Friendly App – Eseva
d) Property Management – eMall
e) Star Small and Medium Banking and Finance

- Build Entity Model diagram. The diagram should align with the business and functional goals stated in the application.

UNIT II - STRUCTURED QUERY LANGUAGE 9+6

SQL Data Definition and Data Types – Constraints – Queries – INSERT, UPDATE, and DELETE in SQL - Views - Integrity Procedures, Functions, Cursor and Triggers - Embedded SQL - Dynamic SQL.

List of Exercise/Experiments

Case Study using real life database applications anyone from the following list and do the following exercises.

a) Inventory Management for a EMart Grocery Shop
b) Society Financial Management
c) Cop Friendly App – Eseva
d) Property Management – eMall
e) Star Small and Medium Banking and Finance

1. Data Definition Commands, Data Manipulation Commands for inserting, deleting, updating and retrieving Tables and Transaction Control statements
2. Database Querying – Simple queries, Nested queries, Sub queries and Joins
3. Views, Sequences, Synonyms
4. Database Programming: Implicit and Explicit Cursors
5. Procedures and Functions
6. Triggers
7. Exception Handling

UNIT III - RELATIONAL ALGEBRA, CALCULUS AND NORMALIZATION 9+6

Relational Algebra – Operations - Domain Relational Calculus- Tuple Relational Calculus - Fundamental operations. Relational Database Design - Functional Dependency – Normalization (1NF, 2NF 3NF and BCNF) –Multivalued Dependency and 4NF –Joint Dependencies and 5NF - De-normalization.

List of Exercise/Experiments

1. Case Study using real life database applications anyone from the following list

- Inventory Management for a EMart Grocery Shop
- Society Financial Management
- Cop Friendly App – Eseva
- Property Management – eMall
- Star Small and Medium Banking and Finance.
- Apply Normalization rules in designing the tables in scope.

UNIT IV - TRANSACTIONS, CONCURRENCY CONTROL AND DATA STORAGE 9+6

Transaction Concepts – ACID Properties – Schedules based on Recoverability, Serializability – Concurrency Control – Need for Concurrency – Locking Protocols – Two Phase Locking – Transaction Recovery –Concepts – Deferred Update – Immediate Update.Organization of Records in Files – Unordered, Ordered – Hashing Techniques – RAID – Ordered Indexes – Multilevel Indexes - B+ tree Index Files – B tree Index Files.

List of Exercise/Experiments

Case Study using real life database applications anyone from the following list

a) Inventory Management for a EMart Grocery Shop
b) Society Financial Management
c) Cop Friendly App – Eseva
d) Property Management – eMall
e) Star Small and Medium Banking and Finance

Ability to showcase ACID Properties with sample queries with appropriate settings for the above scenario

UNIT V - QUERY OPTIMIZATION AND ADVANCED DATABASES 9+6

Query Processing Overview – Algorithms for SELECT and JOIN operations – Query optimization using Heuristics.Distributed Database Concepts – Design –Concurrency Control and Recovery – NOSQL Systems – Document-Based NOSQL Systems and MongoDB.

List of Exercise/Experiments

Case Study using real life database applications anyone from the following list

a) Inventory Management for a EMart Grocery Shop
b) Society Financial Management
c) Cop Friendly App – Eseva
d) Property Management – eMall
e) Star Small and Medium Banking and Finance

- Build PL SQL / Stored Procedures for Complex Functionalities, ex EOD Batch Processing for calculating the EMI for Gold Loan for each eligible Customer.

TOTAL:45+35=75 PERIODS

4. COURSE OUTCOMES

CO1: Map ER model to Relational model to perform database design effectively.

CO2: Implement SQL and effective relational database design concepts.

CO3: Apply relational algebra, calculus and normalization techniques in database design.

CO4: Understand the concepts of transaction processing, concurrency control, recovery procedure and data storage techniques.

CO5: Evaluate and implement transaction processing, concurrency control mechanisms, and recovery procedures to maintain data integrity.

CO6: Analyze and optimize database queries and understand the features and applications of advanced and distributed database systems, including NoSQL.

5. CO- PO/PSO MAPPING

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	2	1	1	1	1	1	1	2	2	2	2	2
CO2	3	2	2	1	1	1	1	2	2	2	2	2
CO3	2	1	1	1	1	1	1	2	2	2	2	2
CO4	2	1	1	1	1	1	1	2	2	2	2	2
CO5	2	1	1	1	1	1	1	2	2	2	2	2
CO6	2	1	1	1	1	1	1	2	2	2	2	2

	PSO1	PSO2	PSO3
CO1	2	2	2
CO2	2	3	2
CO3	2	2	2
CO4	2	2	2
CO5	2	2	2
CO6	2	2	2

6. LECTURE PLAN

S.No	Topic	No of Periods	Proposed Date	CO	Taxonomy Level	Mode of Delivery
1	SQL Data Definition and Data Types	1		CO2	K1	PPT/HANDS ON
2	Constraints	1		CO2	K1	PPT/HANDS ON
3	Queries	1		CO2	K1	PPT/HANDS ON
4	INSERT, UPDATE, and DELETE in SQL	1		CO2	K1	PPT/HANDS ON
5	Views	1		CO2	K1	PPT/HANDS ON
6	Integrity Procedures Functions	1		CO2	K1	PPT/HANDS ON
7	Cursor and Triggers	1		CO2	K1	PPT/HANDS ON
8	Embedded SQL - Dynamic SQL.	2		CO2	K1	PPT/HANDS ON

7. ACTIVITY BASED LEARNING

1. Unit 2 Activity – QUIZ - https://forms.gle/H1DZDCxfH1M2swds9

2. Peer review on database design using ER Model.

3.Student may choose any application to design the database using ER model (https://www.smartdraw.com/entity-relationship- diagram/er-diagram-tool.)

4.**Entity Identification Game (Pair Activity) Objective**: Distinguish between entities, attributes, and relationships. Provide a mixed list of entities, attributes, and relationships. Challenge pairs to classify each item correctly.

5. Implementation of relational database using SQL commands.

8. LECTURE NOTES - UNIT – II

SQL Standards:

The SQL language has several parts:

Data-definition Language (DDL): the SQL DDL provides commands for defining relation schemas, deleting relations, and modifying relation schemas.

Interactive data-manipulation language (DML): the SQL DML includes a query language based on both the relational algebra and the tuple relational calculus. It includes also commands to insert tuples into, delete tuples from, and modify tuples in the database.

View definition: the SQL DDL includes commands for defining views

Transaction control: SQL includes commands for specifying the beginning and ending of transactions.

Embedded SQL and Dynamic SQL: embedded and dynamic SQL define how SQL statements can be embedded within general-purpose programming languages such as C, C++, Java etc.,

Integrity: the SQL DDL includes commands for specifying integrity constraints that the data stored in the database must satisfy. Updates that violate integrity constraints are disallowed.

Authorization: the SQL DDL includes commands for specifying access rights to relations and views

Data types in SQL:

The SQL standard supports a variety of built-in domain types such as,

char(n). Fixed length character string, with user-specified length n.

varchar(n). Variable length character strings, with user-specified maximum length n.

int. Integer (a finite subset of the integers that is machine-dependent).

smallint. Small integer (a machine-dependent subset of the integer domain type).

numeric(p,d). Fixed point number, with user-specified precision of p digits, with n digits to the right of decimal point.

real, double precision. Floating point and double-precision floating point numbers, with machine-dependent precision.

float(n). Floating point number, with user-specified precision of at least n digits.

date: A calendar date containing a (four digit) year, month and day of the month.

time: the time of day in hours, minutes and seconds.

timestamp: a combination of date and time.

Data definition Language:

The SQL data definition language allows specification of not only a set of relations but also information about each relation, including

The schema for each relation.

The domain of values associated with each attribute.

Integrity constraints

The set of indices to be maintained for each relation.

Security and authorization information for each relation.

The physical storage structure of each relation on disk.

DDL Commands:

Create Table Construct:

An SQL relation is defined using the **create table** command:

Syntax:

create table r (A1 D1, A2 D2, ..., An Dn, (integrity-constraint1), ..., (integrity-constraintk))

where,

r is the name of the relation

each Ai is an attribute name in the schema of relation r

Di is the data type of values in the domain of attribute Ai

The allowed integrity constraints include

Primary key : The primary key specification says that attributes are required to be non-null and unique. That is no tuple can have a null value for a primary key attribute, and also there should not be duplicate values.

Check (P) : the check clause specifies a predicate P that must be satisfied by every tuple in the relation.

Unique : it will not allow duplicate values and null values. The difference between the primary key and the unique key is that there can be more than unique key attribute.

Not null: it will not allow null values.

Example:

create table student(name char(15) not null**,** student_id char(10), degree_level char(15), primary key(student_id), check(degree_level in ('bachelors', 'masters', 'doctorate')));

Referential Integrity:

Referential integrity is a relational database concept in which multiple tables share a relationship based on the data stored in the tables, and that relationship must remain consistent. This integrity is enforced using foreign key constraint.

There are actually 3 rules that referential integrity enforces:

We may not add record to the child table unless the foreign key for that record points to an existing record in the parent table.

If a record in the parent table is deleted all corresponding records in the child table must be deleted using a cascading delete.

If the primary key for a record in the parent table changes, all corresponding records in the child table must be modified using cascading update.

Syntax:

Create table table_name(col1 datatype,col2 datatype,…,coln datatype, foreign key(col1,col2,…,coln) references parent_table(col1,col2,…,coln));

Example:

Parent table:

Create table supplier(supid int,sname varchar(20),addr varchar(20),primary key(supid));

Child table:

Create table products(pid int,supid int,foreign key(supid) references supplier(supid));

Alter table:

Alter table command is used to add, modify or drop attributes or columns from the table.

Syntax:

alter table <table-name> add/modify <col-name1><datat ype>,….,<col-namen><data type>;

alter table <table-name> drop <col-name>;

Example:

alter table emp add year char(3);

Truncate table:

The truncate table command deletes the rows in the table but retains the table structure.

Syntax:

truncate table <table-name>;

Example:

Truncate table emp;

Droptable:

Deletes the table from the database.

Syntax:

drop table <table-name>

Example:

drop table emp;

2.4 Data Manipulation Language:

It includes also commands to insert tuples into, delete tuples from, and modify tuples in the database.

DML Commands:

Basic Query Structure:

The basic structure of an SQL expression consists of three clauses: **select, from, and where.**

The **select clause** corresponds to the projection operation of the relationalalgebra. It is used to list the attributes desired in the result of a query.

The **from clause** corresponds to the Cartesian-product operation of the relational algebra. It lists the relations to be scanned in the evaluation of the expression.

The **where clause** corresponds to the selection predicate of the relational algebra. It consists of a predicate involving attributes of the relations that appear in the from clause.

A typical SQL query has the form:

select A1, A2, ..., An
fromr1, r2, ..., rm
where P

Each Ai represents an attribute

Ri represents a relation and P is a predicate.

This query is equivalent to the relational algebra expression.

1) The Select Clause:

The **select** clause list the attributes desired in the result of a query. It corresponds to the projection operation of the relational algebra

Example: find the names of all branches in the loan relation:

Sql query: select branch_name **from** loan

Relational algebra expression: ∏branch_name (loan)

The **result** is a relation consisting of a single attribute with the heading branch-name.

Elimination of Duplicates:

To eliminate duplicates, the keyword **distinct** is inserted after select as below.

Sql query:

select distinct branch_name **from** loan

Without Duplication:

SQL uses the keyword **all** to specify explicitly that duplicates are not removed.

Sql query:

select all branch_name **from** loan

Selection of all attributes:

The * symbol can be used to denote all attributes.

Sql query:

select * from loan

Arithmetic operators:

The select clause may also contain arithmetic expressions involving the operators +,-,*, and / operating on constants or attributes of tuples.

Sql Query:

Select loan-number, branch-name, amount * 100 from loan

will return a relation that is same as the loan relation, except that the attribute amount is multiplied by 100.

The Where Clause:

Example:

Find all loan numbers for loans made at the perryridge branch with loan numbers greater then $1200.

Sql query:

Select loan-number **from** loan **where** branch-name = 'perryridge and amount >1200

SQL uses the logical connectives **and, or, and not** rather than mathematical symbols in the where clause.

Between Operator:

Sql includes a **between** comparison operator to simplify where clauses that specify that a value be less than or equal to some value and greater than or equal to some other value.

Example:

Find the loan number of those loans with loan amounts between $90,000 and $100,000.

Sql Query:

select loan-number fr**o**m loan w**h**ere amount between 90000 and 100000

Instead of

select loan-number **from** loan **where** amount <=1000000 and amount >=90000

Similarly we can use **not between** operator to display values other than this range.

The from clause:

The from clause by itself defines a Cartesian product of the relation in the clause. Since the natural join is defined in terms of a Cartesian product, a selection and a projection, it is relatively simple to write a sql expression for the natural join.

Example 1:

For all customers who have a loan from the bank, find their names, loan numbers and loan amount.

Relational algebra expression:

$\Pi_{\text{customer-name, loan-number, amount}}$ (borrower loan)

Sql query:

select customer-name,borrower.loan-number,amount **from** borrower,loan

where borrower.loan-number = loan.loan-number

Sql uses the notation relation-name.attribute-name to avoid the ambiguity where an attribute appears in the schema of more than one relation.

Example 2:

Find the customer names, loan numbers and loan amounts for all loans at perryridge branch.

Sql query

select customer-name,borrower.loan-number,amount **from** borrower,loan

where borrower.loan-number=loan.loan-number **and** branch-name = 'perryridge'

The rename operation:

Sql provides a mechanism for renaming both relations and attributes. It uses the **as** clause, taking the form:

Old-name **as** new-name

The **as** clause can appear in both **select** and **from** clauses.

Example:

Before rename:

select customer-name,borrower.loan-number,amount f**rom** borrower,loan

where borrower.loan-number=loan.loan-number

After rename:

select customer-name,borrower.loan-number **as** loan-id,amount

from borrower,loan **where** borrower.loan-number=loan.loan-number

The above query renames the loan-number as loan-id.

2) Deletion:

A delete request is expressed in much the same way as a query. Sql expresses a deletion by

deletefrom r **where** P

where P represents a predicate and r represents a relation. The **delete** statement first finds all the tuples in r for which P(t) is true, and then deletes them from r. the **where** clause can be omitted, in which case all tuples in r are deleted.

delete command operates on only one relation. To delete values from several relations **delete** command should be used for each relation separately.

Examples: delete from loan;

deletes all the tuples from the relation.

delete from account **where** branch-name = 'Perryridge';

deletes all account tuples in the Perryridge branch.

delete from loan **where** amount between 1300 and 1500;

deletes all the loan amounts between 1300 and 1500.

Delete from account **where** branch-name **in**(**select** branch-name **from** branch **where** branch-city = 'Needham');

The delete request can contain a nested select that references the relation from which tuples are to be deleted. For example, to delete the records of all accounts with balances below the average at the bank, the following query is used.

delete from account **where** balance<(**selectavg** (balance) **from** account);

The above query first tests each tuple in the relation account to check whether the account has a balance less than the average at the bank. Then, all tuples that fail the test are deleted.

3) Insertion:

To insert data into a relation, the attribute values of inserted tuples must be the members of the attribute's domain. Similarly, tuples inserted must be of the correct type.

Example1:

insertinto account **values**('a-101','perryridge',1200);

inserts the specified values into the relation account.

Example 2:

insertinto account **values(**'&aacount-no','&branch-name',&amt);

inserts more than one value with one insert statement.

Example 3:

insertinto account **select** loan-number,branch-name **from** loan **where** branch-name = 'perryridge';

Inserts tuples into account relation which is taken from loan relation.

Example 4:

insertinto depositor **select** customer-name, loan-no **from** borrower, loan **where** borrower.loan-no = loan.loan-no **and** branch-name = 'Perryridge';

Selects the tuples from borrower and loan relations whose loan numbers are equal, also whose branch is perryridge and inserts those tuples into the relation depositor.

Example 5:

insert into account **select** * from account;

4)Updates:

Update statement is used to change the values of the tuples without affecting other tuples in the relation.

Example 1:

update account **set** balance = balance * 1.05;

updates the balance as specified in the condition.

Example 2:

update account **set** balance = balance * 1.05 **where** balance >=1000;

updates the balance as specified in the condition where balance is greater than or equal to1000.

Example 3:

Sql provides a case construct, which performs two updates with a single update statement.

update account **set** balance = **case when** balance <= 10000 then balance * 1.05 **else** balance * 1.06 **end;**

2.5Data Control Language (DCL):

It is used to create roles, permissions, and referential integrity as well it is used to control access to database by securing it.

GRANT - gives user's access privileges to database

REVOKE - withdraw access privileges given with the GRANT command

Grant:

Syntax:

grant<privilege list>**on**<relation name or view name>**to**<user/role list>

The privilege list allows the granting of several privileges in one command.

Example 1:

The following **grant** statement grants users U1, U2 and U3 **select** authorization on the account relation.

grant selecton account **to** U1,U2,U3

Example 2:

This **grant** statement gives users U1,U2, and U3 update authorization on the amount attribute of the loan relation:

grant update(amount) **on** loan **to** U1,U2,U3

Roles:

Roles can be created in SQL as follows:

Example:

Create role teller

Roles can then be granted privileges just as the users can, as illustrated in the below statement.

grant select on account **to** teller

Roles can be assigned to the users, as well as to some other roles, as the below statements show:

grant teller **to** john

createrole manager

grant teller **to** manager

grant manager **to** mary

The privilege to grant Privileges:

By default, a user/role that is granted a privilege is not authorized to grant privilege to another user/role.

If we wish to grant a privilege and to allow the recipient to pass the privilege on other users, we append **with grant option** clause to the appropriate grant command.

For example, if wish to allow U1 the **select** privilege on branch and allow U1 to grant privilege to others, the query written is

grant select on branch to U1 with grant option

Revoke :

To revoke an authorization, the revoke statement is used. It takes the form almost identical to that grant format.

revoke<privilege list>**on**<relation name or view name>

from<user/role list> [**restrict | cascade**]

Thus, to revoke the privileges that were granted previously, the queries are written as below:

revoke select on branch **from** U1,U2,U3

revoke update (amount) **on** loan **from** U1,U2,U3

The revocation of a privilege from a user/role may cause other users/roles also to lose privilege. This behavior is called **cascading of the revoke**.

revoke select on branch **from** U1,U2,U3 **restrict**

The system returns an error if there are any cascading revokes, and does not carry out the revoke action. The following **revoke** statement revokes only the grant option, rather than the actual select privilege.

revoke grant option for select on branch from U1

Transaction Control Language (TCL):

These statements are used to manage the changes made by DML statements. It allows statements to be grouped together into logical transactions.

COMMIT - save work done

SAVEPOINT - identify a point in a transaction to which you can later roll back

ROLLBACK - restore database to original since the last COMMIT

Example:

```
SQL>select * from borrower;

CNAME                LNO
----------------------  ----------
adams                11
smith                21
jones                31
SQL> savepoint a;    Savepoint created.
SQL> delete from borrower;
3 rows deleted.

SQL> rollback to a;

Rollback complete.
SQL> select * from borrower;
CNAME                LNO
----------------------  ----------
adams                11
smith                21
jones                31
```

Additional Operations in SQL:

String Operations:

Like:

The most commonly used operation on strings is pattern matching using the operator like. In sql the patterns are usually described using two special characters.

Percent (%): the % character matched any substring.

Underscore (_): the _ character matches any character.

Patterns are case-sensitive; that is uppercase characters do not match lower-case characters, or vice-versa. The following examples illustrate the pattern matching.

'Perry%' matches any string beginning with "Perry".

'%idge%' matches any string containing "idge" as a substring, for example, 'Perryridge', 'Rock Ridge', 'Mianus Bridge', and 'Ridgeway'.

'_ _ _' matches any string of exactly three characters.

'_ _ _%' matches any string of at least three characters.

Sql expresses patterns using the **like** comparison operator.

Example: Find the names of all customers whose street address includes the substring 'Main'.

Sql Query:

select customer-name **from** customer **where** customer-street like '%Main%'

Sql also allows finding mismatches by using **not like** operator instead of **like**.

Ordering the Display of Tuples:

Sql offers the user some control over the order in which tuples in a relation are displayed. The order by clause causes the tuples in the result of a query to appear in soted order.

Example:

List in alphabetical order all customers who have a loan at the perryridge branch

Sql query:

select distinct customer-name **from** borrower, loan **where** borrower.loan-number = loan.loan-number **and** branch-name = 'perryridge' **order by** customer-name

By default the order by clause lists items in ascending order or **asc** can be used. To specify the sort order, in descending it is necessary to specify **desc.**

Ordering can also be performed on multiple attributes as below.

Sql query:

select * **from** laon **order by** amount **desc**, loan-number **asc**

Set Operations:

The Sql operations **union, intersect, and except** operate on relations.

Union operation:

Example:

Find all customers having a loan, an account, or both at the bank.

Sql query:

(**select** customer-name **from** depositor) **union** (**select** customer-name **from** borrower)

The union operation automatically eliminates duplicates.

Sql query with duplications:

The **union all** is used to retain the duplicates.

(**select** customer-name **from** depositor) **union all** (**select** customer-name **from** borrower)

The Intersect Operation

Example:

Find all customers who have both a loan and an account at the bank.

Sql Query:

(**select** customer-name **from** depositor) **intersect** (**select** customer-name **from** borrower)

The intersect operation removes all duplicates automatically.

Sql query with duplications:

The **intersect all** is used to retain the duplicates.

(**select** customer-name **from** depositor) **intersect all** (**select** customer-name **from** borrower)

The Except

OperationExample:

Find all customers who have an account but no loan at the bank.

Sql Query:

(select customer-name from depositor except (select customer-name

from borrower)

The except operation removes all duplicates automatically.

Sql query with duplications:

The except all is used to retain the duplicates.

(select customer-name from depositor) except all (select customer-name from borrower)

Aggregate Functions:

Aggregate functions are functions that take a collection of values as input and return a single value. Sql offers five built-in aggregate functions.

Average: **avg,** finds the average

Minimum: **min,** finds the minimum value

Maximum: **max,** finds the maximum value

Total: **sum,** calculates the sum

Count: **count,** finds the number of rows or tuples in a relation

The input to **sum** and **avg** must be a collection of numbers.

Example:

Find the average account balance at the perryridge branch.

Sql query:

select avg (balance) **from** account **where** branch-name = 'Perryridge'

The result of this query is a relation with a single attribute, containing a single row or tuple corresponding to the average balance at the Perryridge branch.

group by clause:

The attribute or attributes given in the **group by** clause are used to form groups. Tuples or rows with the same value on all attributes in the **group by** clause are placed in one group. The **group by** clause is usually associated with aggregate functions.

Example:

Find the average account balance at each branch

Sql Query:

select branch-name, **avg**(balance) **from** account **group by** branch-name

having clause:

The having clause is usually used along with group by clause.

Example:

Find the account balance of those branches where the average account balance is more than $1200.

This condition does not apply to a single row or tuple. Rather, it applies to each group constructed by the **group by** clause. To express such query the **having clause** is used.

Sql Query:

select branch-name, **avg**(balance) **from** account **group by** branch-name
having avg(balance)>1200

Nested Subqueries:

Sql provides a mechanism for nesting Subqueries. A subquery is a **select-from-where** expression that is nested within another query. A common use of subqueries is to perform tests for set membership, make set comparisons, and determine set cardinality.

Set Membership:

The **in** connective tests for the set membership, where the set is a collection of values produced by a **select** clause. The **not in** connective tests for the absence of set membership.

Example 1:

Find all customers who have both a loan and an account at the bank.

The intersection operation is used for this query previously.

Sql Query using in:

select distinct customer-name **from** borrower

where customer-name **in (select** customer-name **from** depositor)

This example shows that it is possible to write the same query several ways in sql.

Example 2:

Find all customers who have both an account and a loan at the perryridge branch

Sql Query using in:

select distinctcustomer-name f**rom** borrower, loan **where** borrower.loan-number = loan.loan-number **and b**ranch-name = 'Perryridge' **and** (branch-name, customer-name)**in (select** branch-name, customer-name **from** depositor, account **where** depositor.account-number = account.account-number)

Example 3:

Find all customers who have a loan at the bank but do not have an account at the bank

Sql Query using not in:

select distinct customer-name **from** borrower **where** customer-name**not in (select** customer-name **from** depositor**)**

Example 4:

Find the names of customers who have a loan at the bank, and whose names are neither smith nor jones.

Sql query:

select distinct customer-name **from** borrower **where** customer-name **not in** ('smith','jones')

Set comparison:

The set comparisons are performed using **>some** and **>all** connectives.

Example 1:

Finds the names of all branches that have assets greater than those of at least one branch located in Brooklyn.

Sql query using some:

selectbranch-name **from** branch **where** assets>**some** **(select** assets **from** branch **where**branch-city = 'Brooklyn')

The subquery,

(select assets **from** branch **where**branch-city = 'Brooklyn')

generates the set of all asset values for all branches in Brooklyn. The **>some** comparison in the where clause of the outer select is true if the assets value of the tuple is greater than at least one member of the set of all asset values for branches in Brooklyn.

Sql also allows **< some**, **<=some**, **>=some** and **<> some** comparisons.

Example 2:

Finds the names of all branches that have an asset value greater than that of each branch located in Brooklyn.

Sql query using all:

select branch-name **from** branch **where** assets**>all (select** assets **from** branch **where** branch-city **= 'Brooklyn')**

Sql also allows **< all**, **<=all**, **>=all** and **<> all** comparisons

Views:

There are two kinds of tables namely base tables and views. Base table is a physical table that is not defined in terms of other tables. That is it exists on its own right. A view on the other hand does not exist on its own right. It is a named table that is represented not by its own physically separate stored data, but by its definition in terms of other named tables. When users see a view, they see the same data that is in the base tables, but perhaps with a different perspective.

Sql defines a view using the **create view** command. To define a view we must give the view a name and must sate the query that computes the view. The form of the create view command is

create view v **as** <query expression>

where <query expression> is any legal query expression. The view name is represented by v.

Example:

Consider a **book table.** It has six columns namely, **id, title, author, publisher, year** and **price.** If certain users need to see only the **id, title, publisher, year and price** columns and that too of **books published after 1995,** then the view can be created as follows:

create view bookv (id, title, publisher, year, price)

asselect id, title, publisher, year, price

from book **where** year > 1995;

The effect of the above sql statement is the creation of a view called bookv. This view will be based on the book table.

Data manipulation with views:

Once the view has been created, the user can perform data manipulation with the created views.

Example:

Query1:

Display the id, title, publisher, year, price of book whose published year is greater than 1995 and the publisher name is McGraw-Hill.

To display the above details the view **bookv** can be used. The view already has the necessary details except publisher name. So the query can be written as follows.

```
select *
from bookv
where publisher = "McGraw-Hill";
```

Query 2:

Increase the price of the book to 12% whose year is greater than 1995 and whose publisher name is McGraw-Hill.

The sql statement for the above query can be written as below.

```
update bookv
set price = price * 1.12
where publisher = "McGraw-Hill;
```

The insert and delete operation can also be performed to the created view.

Views involving multiple tables:

The views can also be created by taking the columns from the multiple tables. For example, consider the **book** and **publisher** tables**.** The **book** tablehas the columns, **id, title, author, publisher, year,** and **price.** The **publisher** tablehas the columns **publisher-id, name, city, country.** A **single view** can be created by **joining these two tables** as below.

createview bookpub

asselect id, title, author, publisher, year, price, city, country

from book, publisher

where book.publisher = publuisher.name;

Here the publisher name is omitted in the publisher table to avoid the repetition. Now the sql statements can be issued on this view as if it were a single table.

Example:

Query:

Display title, author, publisher, city, country of the books whose publisher country is India.

The sql statement for the above query can be written as below:

select title, author, publisher, city, country

from bookpub

where publisher.country = "India";

Conditions regarding updates of a view:

- It must be derived from a single table.
- If the definition of the view involves either a group by or having clause then the view is not updateable.
- If the definition of the view includes a nested subquery then the view is not updateable.
- If the column of a view is derived from an aggregate function then the view is not updateable.

Advantages of view:

- Data security – views allow to set up different security levels for the same base table, thus shielding or protecting certain data from people who do not have proper authority.
- The views allow the same data to be seen by different users in different ways at the same time.
- Views can be used to hide complex queries.

PROCEDURES

Procedures are the subprograms which can be created and saved in the database as database objects. Just as you can in other languages, you can create and drop procedures in SQL as well.

Syntax of procedures in SQL

The following illustrates the basic syntax of creating a procedure in SQL

```
CREATE [ OR REPLACE] PROCEDURE procedure_name [
(parameter_name [IN | OUT | IN OUT] type [ ])]
{IS | AS }
BEGIN [declaration_section]
executable_section
//SQL statement used in the stored procedure
END
GO
```

Parameter

A parameter is a variable that holds a value of any valid SQL datatype through which the subprogram can exchange the values with the main code. In other words, parameters are used to pass values to the procedure. There are 3 different types of parameters, which are as follows:

- IN: This is the Default Parameter, which always receives the values from the calling program. It is a read-only variable inside the subprograms and its value cannot be changed inside the subprogram.
- OUT: It is used for getting output from the subprograms.
- IN OUT: This parameter is used for both giving input and for getting output from the subprograms.

Procedure in SQL: Examples

Example1

The following example creates a simple procedure that displays the welcome message on the screen when executed. Then, the procedure will be:

```
CREATE OR REPLACE PROCEDURE welcome_msg
(para1_name IN VARCHAR2)
IS
BEGIN
    dbms_output.put_line (‘Hello World! '||
para1_name);
END;
/
```

Execute the stored procedure. A standalone procedure can be called in two ways -

- Using the **EXECUTE** keyword
- Calling the name of the procedure from a SQL block

Advantages of procedures in SQL

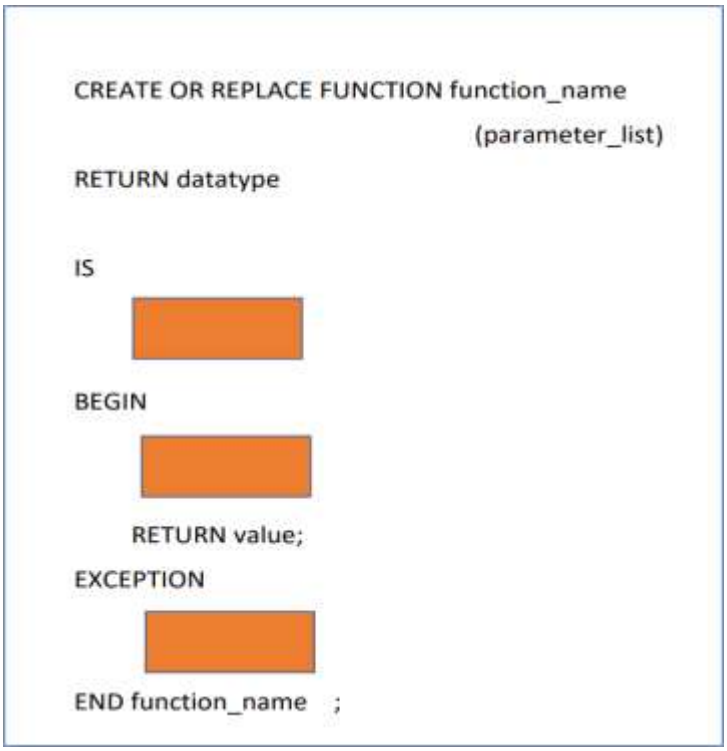
The main purpose of stored procedures in SQL is to hide direct SQL queries from the code and improve the performance of database operations such as select, update, and delete data. Other advantages of procedure in SQL are:

- Reduces the amount of information sent to the database server. It can become a more important benefit when the bandwidth of the network is less.
- Enables the reusability of code
- Enhances the security since you can grant permission to the user for executing the Stored procedure instead of giving permission on the tables used in the Stored procedure.
- Support nested procedure calls to other SQL procedures or procedures implemented in other languages

.

FUNCTIONS

Functions is a subprogram that runs on its own. Functions, like procedures, have a unique name by which they can be identified



Some of the qualities of functions are listed below.

- Functions are a type of standalone block that is mostly used to do calculations.
- The value is returned using the **RETURN** keyword, and the datatype is defined at the time of creation.
- Return is required in functions since they must either return a value or raise an exception.
- Functions that do not require DML statements can be called directly from SELECT queries, whereas functions that require DML operations can only be invoked from other PL/SQL blocks.
- It can be defined and nested inside other blocks or packages, or it can have nested blocks.
- The parameters can be used to pass values into the function or to retrieve values from the process.

- In addition to utilizing RETURN, a PLSQL function can return the value using OUT parameters.
- Because it always returns the value, it is always used in **conjunction** with the assignment operator to populate the variables in the calling statement.

```
CREATE OR REPLACE FUNCTION Customer_moduel (
                                customer_id IN NUMBER,
                                customer_Balance OUT NUMBER)
RETURN VARCHAR2;
IS
    name VARCHAR2(30)
    address VARCHAR2(30)

    CURSOR get_customer(customer_id NUMBER) IS
            SELECT customer_name,customer_address, balance
            FROM Customer_table
            WHERE customer_ID = customer_id;
BEGIN

OPEN get_customer(2);

LOOP
  FETCH get_customer INTO name, address, customer_Balance'

   IF get_customer%NOTFOUND:
   DBMS_OUTPUT.PUT_LINE("No data available");
   END IF;

  END LOOP;

  CLOSE get_customer;

  RETURN name;
END Customer_moduel;
```

Cursors in PLSQL

The cursor is a temporary working area created in the system memory when your SQL statement is executed. Cursors contain information on the select statement and the data which was accessed by the particular select statement. In simple words, it works like a pointer to the result set of the SQL query in PLSQL. There are two types of cursors.

1. Implicit Cursors

Oracle automatically creates implicit cursors when a SQL statement is executed, and there is no explicit cursor for the statement. As a result, programmers have no control over implicit cursors or the data they contain.

When a DML(Data Manipulation Language) statement (INSERT, UPDATE, or DELETE) is issued, it is accompanied by an implicit cursor. The cursor holds the data that needs to be inserted for INSERT operations. In addition, the cursor identifies the rows that will be affected by UPDATE and DELETE actions.

The most recent implicit cursor is the SQL cursor in PL/SQL, and it always has attributes like **%FOUND**, **%ISOPEN**, **%NOTFOUND**, and **%ROWCOUNT**.

```
DECLARE
   customer_idA NUMBER;

BEGIN
    SELECT customer_id
    INTO customer_idA
    FROM customer_table

 END
```

Explicit Cursors

Explicit cursors are cursors that have been programmed to give the user more control over the context area. In the PLSQL Block's declaration section, an explicit cursor should be defined. It's based on a SELECT statement that returns multiple rows. The following code snippet shows the way to declare the explicit cursor.

```
DECLARE
  name VARCHAR2(30);
  address VARCHAR2(30)

  CURSOR get_customer(cus_id NUMBER) IS
      SELECT customer_name, address  FROM
      customer_table
      WHERE customer_id =cus_id;

BEGIN                          --Pass the parameter for cus_id
     OPEN get_customer(10);
     variable
   LOOP
     FETCH get_customer INTO name, address;     -- We fectched
   the data into name and address variables
     EXIT WHEN get_customer%NOTFOUND    -- used the %NOTFOUND
   attribute to validate the result set
   END LOOP;

   CLOSE get_customer;

   END;
```

Triggers:

A trigger is a statement that the system executes automatically as a side effect ofa modification to the database. A trigger has three basic parts:

- A triggering event or statement
- A trigger restriction
- A trigger action

Each of these parts is explained below:

Triggering Event or Statement:

It is a SQL statement that causes a trigger to be fired. It can be insert, update, or delete statement for a specific table

Trigger Restriction:

A trigger restriction specifies a Boolean expression that must be true for the trigger to fire. It is an option available for triggers that are fired for each row. Its function is to conditionally control the execution of a trigger. A trigger restriction isspecified using a **WHEN** clause.

Trigger Action:

A trigger action is the PL/SQL code to be executed when a triggering statement is encountered and if any trigger restriction evaluates to TRUE. The PL/SQL block can contain SQL and PL/SQL statements, can define PL/SQL language constructs and can call stored procedures. Additionally, for row triggers, the keywords :old and :new are used to access the old and new values.

Need for Triggers:

Triggers are useful mechanisms for alerting humans or for starting certain tasks automatically when certain conditions are met.

Types of triggers:

The types of triggers are given below:

Row-level triggers:

Row-level triggers trigger once for each row in a transaction. These types of triggers are useful, to track the modifications made to the data for each row in the table. A **row level trigger** is created by using the **for each row clause** in the **create trigger** command.

Statement-level Triggers:

Statement-level triggers execute once for each transaction. For example, if a single transaction inserted 700 rows into a table then a statement-level trigger on that table will be executed only once. Statement-level triggers are not often used for data related activities. **Statement-level triggers** are default type of triggers created using **create trigger** command.

Before and After triggers:

Since triggers occur because of events, they may set to occur immediately before or after those events. Since the events that execute triggers are database transactions, triggers can be executed immediately before or after **Inserts, Updates and Deletes.**

Within a trigger, it is possible to reference the old and new values involved in the transaction.

The access required for the old and new data may determine which type of trigger is needed.

OLD refers to the data, as it existed prior to the transaction. Updates and deletes usually reference old values.

New values are the data values that the transaction creates. They are referred by the keyword NEW.

To set a column value in an inserted row via a trigger, then before insert trigger should be used in order to access the new values.

An after insert trigger would not allow to set the inserted value, since the row will already have been inserted into the table.

Syntax:

The syntax for the create trigger command is shown below.

```
CREATE[OR REPLACE] TRIGGER TRIGGER_NAME
  [BEFORE|AFTER]
  [DELETE|INSERT|UPDATRE[OF COLUMN_NAME]]
  ON [TABLE_NAME]
  [FOR EACH ROW] [WHEN CONDITION]
  [PL/SQL BLOCK];
```

The REPLACE keyword replaces the trigger if it already exists.

TRIGGER_NAME is the name of the trigger created.

The BEFORE and AFTER keywords indicate whether the trigger should be executed before or after the triggering event.

The DELETE, INSERT AND UPDATE keywords indicate the type of data manipulation that will constitute the triggering event.

When the, FOR EACH ROW clause is used, the trigger will be a row-level trigger, otherwise, it will be a statement-level trigger.

The WHEN clause is used to further restrict when the trigger is executed. The restrictions enforced in the when clause may include checks of old and new data values.

Deleting a trigger:

Syntax:

Drop trigger <trigger_name>;

deletes the created trigger.

Raise_application_error procedure:

The oracle engine provides a procedure named raise_application_error that allows programmers to issue user-defined error messages.

Syntax:

raise_application_error (<errorno>,<message>);

where errorno is a negative number in the range -20000 to -20999 and message is a character string up to 2048 bytes in length.

Example 1:

Question:

Raise a trigger such that the insert and update is not possible if the quantity on hand is less than 50 on the itemfile table.

Trigger:

CREATE OR REPLACE TRIGGER QTY

BEFORE INSERT OR UPDATE ON ITEMFILE FOR EACH ROW

```
WHEN (NEW.QTY_HAND<=50)
BEGIN
RAISE_APPLICATION_ERROR(-20001,'CANNTO INSERT OR UPDATE');
END;
```

Example 2:

Question:

Raise a trigger if the entered amount is negative or zero.

Trigger:

```
CREATE OR REPLACE TRIGGER TRANS_CHECK
BEFORE INSERT ON TRANS FOR EACH ROW
BEGIN

IF: NEW.AMT<=0 THEN
RAISE_APPLICATION_ERROR (-20001,'CANNOT BE NEGATIVE OR ZERO');
END IF;

END;
```

Embedded SQL

SQL provides a powerful declarative query language. Writing queries in SQL is usually much easier than coding the same queries in a general-purpose language. However, a programmer must have access to a database from ageneral purpose programming language for at least two reasons:

Not all queries can be expressed in SQL, since SQL does not provide the full expressive power of a general purpose language. That is, there exist queries that can be expressed in a language such as C, java, or COBOL that cannot be expressed in SQL. To write such queries, we can embed SQL within a more powerful language.

Non declarative actions-such as printing a report, interacting with a user, or sending the results of a query to a graphical user interface cannot be done from within SQL.

The SQL standard defines embeddings of SQL in a variety of programming languages such as Pascal, PL/I, FORTRAN, C, and COBOL.

A language to which SQL queries are embedded is referred to as a host language, and the SQL structures permitted in the host language comprise embedded SQL.

Programs written in the host language can use the embedded SQL syntax to access and update data stored in a database.

This embedded form of SQL extends the programmer's ability to manipulate the database even further.

In embedded SQL, all query processing is performed by the database system, which then makes the result of the query available to the programmer one tupleat a time.

An embedded SQL program must be processed by a special preprocessor prior to compilation.

The preprocessor replaces embedded SQL requests with host-language declarations and procedure calls that allow run-time execution of the database accesses.

Then, the resulting program is compiled by the host language compiler. To identify embedded SQL requests to the preprocessor, we use the EXEC SQL statement.

It has the form

EXEC SQL <embedded SQL statement> END-EXEC

The exact syntax for embedded SQL requests depends on the language in which SQL is embedded.

For instance, a semicolon is used instead of END-EXEC when SQL is embedded in C.

The java embedding of SQL called (SQLJ) uses the syntax

#SQL{<embedded SQL statement>}

The statement SQL INCLUDE is placed in the program to identify the place where the preprocessor should insert the special variables used for communication between the program and the database system.

Variables of the host language can be used within embedded SQL statements, but they must be preceded by a colon (:) to distinguish them from SQL variables.

Before executing any SQL statements, the program must first connect to the database. This is done using

EXEC SQL connectto server user user-name END-EXEC

Here server identifies the server to which a connection is to be established.

Database implementations may require a password to be provided in addition to a user name.

To write a relational query, the declare cursor statement is used. The program must use the open and fetch commands to obtain the result tuples.

Consider the banking schema. Assume that we have a host language variable amount, and that we wish to find the names and cities of residence of customers who have more than amount dollars in any account. The query can be written as follows:

EXEC SQL

Declare c cursor for
select customer-name, customer-city
from depositor, customer, account
where depositor.customer-name = customer.customer-name and
depositor account-number = account.account-number
and account.balance > :amount

END-EXEC

- The open statement causes the query to be evaluated and to save the result in the temporary relation.
- EXEC SQL **open** c END-EXEC
- If the SQL statement results in an error the database system stores an error diagnostic in the SQL communication area (SQLCA) variables, whose declarations are inserted by the SQL include statement.
- The **fetch** statement causes the values of one tuple in the query result to be placed on host language variables.

EXEC SQL fetch **c** into ***:cn, :cc END-EXEC***

The fetch statement requires one host-language variable for each attribute of the result relation. In the example query cn ***holds the*** customer-name ***and*** cc ***holds the*** customer-city.

To obtain all tuples of the result, the program must contain a loop to iterate over tuples. When the program executes an open statement on a cursor, the cursor is set to point to the first tuple of the result. Each time it executes a fetch statement, the cursor is updated to point to the next tuple of the result. When no further tuples remain to be processed, the variable called SQLSTATE in the SQL communication area (SQLCA) gets set to '02000' to indicate no more data is available.

The **close** statement causes the database system to delete the temporary relation that holds the result of the query.

EXEC SQL **close**c END-EXEC

Updates through Cursors

Embedded SQL can update tuples fetched by cursor by declaring that the cursor isfor update as shown below:

```
declare c cursor for
    select *  from account  where branch-name = 'Perryridge'
    for update
```

We then iterate through the tuples by performing fetch operations on the cursor, and after fetching each tuple the following code is executed.

```
update account  set balance = balance + 100 where current of c
```

Dynamic SQL:

- The dynamic SQL component allows programs to construct and submit SQL queries at run time.
- In contrast, embedded SQL statements must be completely present at compile time.
- Using dynamic SQL, programs can create SQL queries as strings at run time and can either have them executed immediately or have them prepared for subsequent use.
- Preparing dynamic SQL statement compiles it, and subsequent uses of the prepared statement use the compiled version
- SQL defines standards for embedding dynamic SQL calls in a host language, such as C, as in the following example.
- char * sqlprog = "update account
  set balance = balance * 1.05
  where account-number = ?"
  EXEC SQL prepare dynprog from :sqlprog;
  char account [10] = "A-101";
  EXEC SQL execute dynprog using :account;
- The dynamic SQL program contains a ?, which is a place holder for a value that is provided when the SQL program is executed

Limitation of Dynamic SQL:

- We cannot use some of the SQL statements Dynamically.
  Performance of these statements is poor as compared to Static SQL.

Limitations of Static SQL:

- They do not change at runtime thus are hard-coded into applications.

9. ASSIGNMENT : UNIT – II

Set Number (Category)	Questions
Set - 1 (Toppers)	**Consider the following Employee Database** Emp(ename,street,city) Works(ename,companyname,salary) Company(companyname,city) Manages(ename,managername) **Give the expression in SQL to express each of the following queries:** 1. Find the names of all employees who work for First Bank Corporation. 2. Find the names, street address, and cities of residence of all employees who work for first Bank Corporation and earn more than 200000 per annum. 4. Find the names of all employees in this database who live in the same city as the companies for which they work. 5. Find the names of all employees who live in the same city and on the same street as do their managers. 6. Find the names of all employees who earn more than every employees of small bank corporation **(CO2, K3)**
Set - 2 (Above Average)	**Consider the following schema:** Catalog(sid: integer, pid: integer, cost: real) Suppliers(sid: integer, sname: string, address: string) Parts(pid: integer, pname: string, color: string) The key fields are underlined, and the domain of each field is listed after the field name. Therefore sidis the key for Suppliers, pidis the key for Parts and sidand pidtogether form the key for Catalog. The Catalog relation lists the prices charged for parts by Suppliers. Write the following queries in relational algebra and SQL. 1. Find the sids of suppliers who supply some red or green part 2. Find the sids of suppliers who supply every part. 3. Find the sids of suppliers who supply every red part or supply every green part. 4. Find sailors who've reserved at least one boat **(CO2, K3)**
Set - 3 (Average)	**Consider the following relation:** Emp(ename,street,city) Works(ename,companyname,salary) Company(companyname,city) Manages(ename,managername) Give the SQL DDL definition of this database. Identify referential integrity constraints that should hold and include them in the DDL definition. **(CO2, K3)**

9. ASSIGNMENT : UNIT – II

Set Number (Category)	Questions
Set - 4 **(Below Average)**	**Write a trigger to ensure that no employee of age less than 25 can be inserted in the database.** **Explanation:** Whenever we want to insert any tuple to table 'employee', then before inserting this tuple to the table, trigger named 'Check_age' will be executed. This trigger will check the age attribute. If it is greater than 25 then this tuple will be inserted into the tuple otherwise an error message will be printed stating "ERROR: AGE MUST BE ATLEAST 25 YEARS!" **(CO2, K3)**
Set - 5 **(Slow Learners)**	**Create a trigger which will work before deletion in employee table and create a duplicate copy of the record in another table employee_backup.** Explanation: We want to create a backup table that holds the value of those employees who are no more the employee of the institution. So, we create a trigger named Backup that will be executed before the deletion of any Tuple from the table employee. Before deletion, the values of all the attributes of the table employee will be stored in the table employee_backup. **(CO2, K3)**

10. Part A Question & Answer

S.No	Question and Answers	CO	K
1	**What is the purpose of the UNIQUE constraint in SQL?** Answer: The UNIQUE constraint ensures that all values in a column are distinct, preventing duplicate entries.	CO2	K1
2	**Define the term "data type" in SQL.** Answer: A data type specifies the kind of data that can be stored in a column, such as INTEGER, VARCHAR, or DATE	CO2	K1
3	**What is the function of the AUTO_INCREMENT attribute in SQL?** Answer: The AUTO_INCREMENT attribute automatically generates a unique value for a column, typically used for primary keys.	CO2	K1
4	**What does the NULL value represent in SQL?** Answer: A NULL value represents the absence of a value or an unknown value in a column.	CO2	K1
5	**What is a composite key?** Answer: A composite key is a combination of two or more columns in a table that uniquely identifies a row.	CO2	K1
6	**Explain the difference between CHAR and VARCHAR data types.** Answer: CHAR is a fixed-length string data type, while VARCHAR is a variable-length string data type that can store up to a specified length	CO2	K2
7	**What is the purpose of the FOREIGN KEY constraint?** Answer: The FOREIGN KEY constraint establishes a link between two tables, ensuring referential integrity by requiring that the value in the foreign key column matches a value in the primary key column of another table.	CO2	K2
8	**Describe the role of a view in SQL.** Answer: A view is a virtual table created by a query that simplifies complex queries and enhances security by restricting access to specific data.	CO2	K2
9	**What is the difference between DELETE and TRUNCATE commands?** Answer: DELETE removes rows from a table based on a condition and can be rolled back, while TRUNCATE removes all rows from a table without logging individual row deletions and cannot be rolled back.	CO2	K2
10	**Explain the purpose of the CHECK constraint.** Answer: The CHECK constraint enforces a condition on the values in a column, ensuring that only valid data is entered	CO2	K2

S.No	Question and Answers	CO	K
11	**Write an SQL statement to create a table named Students with columns for StudentID, Name, and Age.** Answer: CREATE TABLE Students (StudentID INT PRIMARY KEY, Name VARCHAR(100) NOT NULL, Age INT CHECK (Age >= 0));	CO2	K3
12	**How would you insert a new record into the Students table?** Answer: INSERT INTO Students (StudentID, Name, Age) VALUES (1, 'Alice', 20);	CO2	K3
13	**Write an SQL statement to update the age of a student with StudentID 1. (CO2, K3)** Answer: UPDATE Students SET Age = 21 WHERE StudentID = 1;	CO2	K3
14	**How can you delete a student record with StudentID 1 from the Students table? (CO2, K3)** Answer: DELETE FROM Students WHERE StudentID = 1;	CO2	K3
15	**Write an SQL query to create a view named AdultStudents that shows students aged 18 and above. (CO2, K3)** Answer: CREATE VIEW AdultStudents AS SELECT * FROM Students WHERE Age >= 18;	CO2	K3
16	**Given the following tables, Orders and Customers, how would you find all orders made by customers from a specific country? (CO2, K4)** Answer: SELECT Orders.* FROM Orders INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID WHERE Customers.Country = 'SpecificCountry';	CO2	K4
17	**Analyze the following SQL statement and identify any potential issues: (CO2, K4)** SELECT * FROM Students WHERE Age > 18 OR Age < 10; Answer: The query may return students who are not within the expected age range, which could be an issue if the business rule states that students must be between 10 and 18 years old.	CO2	K4
18	**How would you identify duplicate records in a table named Products based on the ProductName column? (CO2, K4)** Answer: SELECT ProductName, COUNT(*) FROM Products GROUP BY ProductName HAVING COUNT(*) > 1;	CO2	K4
19	**Given a table Sales, how would you calculate the total sales amount for each product? (CO2, K4)** Answer: SELECT ProductID, SUM(Amount) AS TotalSales FROM Sales GROUP BY ProductID;	CO2	K4

S.No	Question and Answers	CO	K
20	**How can you find the maximum salary of employees in each department from the Employees table? (CO2, K4)** Answer: SELECT DepartmentID, MAX(Salary) AS MaxSalary FROM Employees GROUP BY DepartmentID;	CO2	K4
21	**Evaluate the following SQL statement for correctness: (CO2, K4)** SELECT * FROM Students WHERE Age = 20 AND Name LIKE 'A%'; Answer: The statement is correct; it retrieves students who are 20 years old and whose names start with 'A'.	CO2	K4
22	**Assess the impact of using a DELETE statement without a WHERE clause on a table. (CO2, K4)** Answer: Using a DELETE statement without a WHERE clause will remove all records from the table, leading to potential data loss.	CO2	K4
23	**Critique the use of triggers for enforcing business rules in a database. (CO2, K4)** Answer: Triggers can effectively enforce business rules automatically, but they can also lead to complexity and performance issues if not managed properly.	CO2	K4
24	**Evaluate the following SQL statement for potential performance issues: (CO2, K4)** Answer: This query may have performance issues due to the leading wildcard %, which prevents the use of indexes, resulting in a full table scan.	CO2	K4
25	**Assess the use of dynamic SQL in applications. What are the pros and cons? (CO2, K4)** Answer: Pros include flexibility and the ability to construct complex queries at runtime. Cons include increased risk of SQL injection attacks and potential performance overhead. .	CO2	K4

26	**Create an SQL function that returns the full name of a student given their StudentID. (CO2, K4)**
	
	Answer:
	CREATE FUNCTION GetFullName(StudentID INT) RETURNS VARCHAR(200) AS
	BEGIN
	DECLARE FullName VARCHAR(200);
	SELECT CONCAT(FirstName, ' ', LastName) INTO FullName
	FROM Students WHERE StudentID = StudentID;
	RETURN FullName;
	END;
		CO2	K4

S.No	Question and Answers	CO	K

27	**Write an SQL procedure to increase the age of all students by 1 year. (CO2, K6)**
	
	Answer:
	CREATE PROCEDURE IncreaseAge()
	BEGIN
	UPDATE Students SET Age = Age + 1;
	END;
		CO2	K6

28	**Create a trigger that logs changes to the Students table into a ChangeLog table. (CO2, K6)**
	
	Answer:
	CREATE TRIGGER LogStudentChanges
	AFTER UPDATE ON Students
	FOR EACH ROW
	BEGIN
	INSERT INTO ChangeLog (StudentID, ChangeDate, OldAge, NewAge)
	VALUES (OLD.StudentID, NOW(), OLD.Age, NEW.Age);
	END;
		CO2	K6

29	**Write an SQL statement to create a dynamic SQL query that selects all columns from a table based on a variable table name. (CO2, K6)** Answer: SET @tableName = 'Students'; SET @sql = CONCAT('SELECT * FROM ', @tableName); PREPARE stmt FROM @sql; EXECUTE stmt; DEALLOCATE PREPARE stmt;	CO2	K6
30	**Create a view that combines data from two tables, Courses and Students, showing course names and the names of students enrolled in those courses. (CO2, K6)** Answer: CREATE VIEW CourseEnrollments AS SELECT Courses.CourseName, Students.Name FROM Courses JOIN Enrollments ON Courses.CourseID = Enrollments.CourseID JOIN Students ON Enrollments.StudentID = Students.StudentID;	CO2	K6

S.No	Question and Answers	CO	K
31	**Name the Categories of SQL Commands?** SQL commands are divided in to the following categories: 1. Data definition language 2. Data manipulation language 3. Data Query language 4. Data control language 5. Data administration statements 6. Transaction control statements	CO2	K1
32	**What is data definition language?** Data Definition Language: A data definition language or data description language (DDL) is a syntax for defining the data structures using imperative verbs, especially database schemas.	CO2	K1
33	**What is the syntax for creating a table in SQL?** Syntax: CREATE TABLE [table name] ([column definitions]) [table parameters] Example: The command to create a table named employees with a few sample columns would be: CREATE TABLE employees (id INTEGER PRIMARY KEY, first_name VARCHAR(50) NULL, last_name VARCHAR(75) NOT NULL, fname VARCHAR(50) NOT NULL, dateofbirth DATE NULL);	CO2	K1
34	**What is the syntax for removing or deleting a table in SQL?** **Drop - To destroy an existing** database, table, index, or view. Syntax: DROP objecttype objectname. Example: The command to drop a table named employees would be: DROP employees; The Drop statement would remove the entire table (employees) from the database	CO2	K1
35	**How to modify or alter an existing table in SQL.** Alter - To modify an existing database object. An ALTER statement in SQL changes the properties of an object inside of a relational database management system (RDBMS). Syntax: ALTER objecttype objectname parameters. Example: The command to add (then remove) a column named bubbles for an existing table named sink would be: ALTER TABLE sink ADD bubbles INTEGER; ALTER TABLE sink DROP COLUMN bubbles;	CO2	K1
36	**What is a PROJECT operation?** The project operation is a unary operation that returns its argument relation with certain attributes left out. Projection is denoted by pie	CO2	K1

S.No	Question and Answers	CO	K
37	**How to rename an existing table in SQL?** Syntax: Rename - to rename the table. Example RENAME TABLE old_name TO new_name;	CO2	K1
38	**Define Data Manipulation Language.** A data manipulation language (DML) is a set of commands permitting users to manipulate data in a database. This manipulation involves inserting data into database tables, retrieving existing data, deleting data from existing tables and modifying existing data.	CO2	K1
39	**List the DML commands in SQL.** SELECT: This command is used to retrieve rows from a table. The select UPDATE: This command modifies data of one or more records. The update command INSERT: This command adds one or more records to a database table. DELETE: This command removes one or more records from a table according to specified conditions	CO2	K1
40	**List the aggregation functions in SQL.** • COUNT returns the number of rows in a specified column. • SUM returns the sum of the values in a specified column. • AVG returns the average of the values in a specified column. • MIN returns the smallest value in a specified column. • MAX returns the largest value in a specified column.	CO2	K1
41	**List the Types of subquery** • A scalar subquery returns a single column and a single row (singlevalue). • A row subquery returns multiple columns, but a single row. • A table subquery returns one or more columns and multiple rows.	CO2	K1
42	**How to modify a data in a table in SQL?** Modifying Data in the DB (UPDATE) Syntax UPDATE table_name SET column_name1 = data_value1 [, column_namei = data_valuei ...] [WHERE search_condition]	CO2	K1

S.No	Question and Answers	CO	K
43	**What is DCL?** The Data Control Language (DCL) component of the SQL language is used to create privileges to allow users access to, and manipulation of, the database. There are two main commands: GRANT to grant a privilege to a user REVOKE to revoke (remove) a privilege from a user	CO2	K1
44	**Write about REVOKE command in SQL.** SQL REVOKE Command: The REVOKE command removes user access rights or privileges to the database objects. The Syntax for the REVOKE command is: REVOKE privilege_name ON object_name FROM {user_name |PUBLIC |role_name}	CO2	K1
45	**What is the syntax for dropping a role in SQL?** The Syntax to drop a role from the database: DROP ROLE role_name; Example: To drop a role called developer, you can write: DROP ROLE testing;	CO2	K1
46	**Define TCL.** TCL - Transactional Control Language. It is used to manage different transactions occurring within a database.	CO2	K1
47	**Define ROLLBACK statement in SQL.** To undo work done in the current transaction. You can also use this command to manually und the work done by an in-doubt distributed transaction.	CO2	K1
48	**What is embedded SQL?** Embedded SQL statements are SQL statements written inline with the program source code of the host language. The embedded SQL statements are parsed by an embedded SQL preprocessor and replaced by host-language calls to a code library.	CO2	K1
49	**What is static SQL?** Static SQL The source form of a static SQL statement is embedded within an application program written in a host language such as COBOL. The statement is prepared before the program is executed and the operational form of the statement persists beyond the execution of the program.	CO2	K1

S.No	Question and Answers	CO	K
50	**What is dynamic SQL?** Programs that contain embedded dynamic SQL statements must be precompiled like those that contain static SQL, but unlike static SQL, the dynamic statements are constructed and prepared at run time.	CO2	K1
51	**What are triggers in SQL?** Triggers are the SQL codes that are automatically executed in response to certain events on a particular table. These are used to maintain the integrity of the data. A trigger in SQL works similar to a real-world trigger.	CO2	K1
52	**What is Cursor in SQL ?** **Cursor** is a Temporary Memory or Temporary Work Station. It is Allocated by Database Server at the Time of Performing DML(Data Manipulation Language) operations on Table by User. Cursors are used to store Database Tables.	CO2	K1
53	**What is Trigger in SQL ?** **Trigger** is a statement that a system executes automatically when there is any modification to the database. In a trigger, we first specify when the trigger is to be executed and then the action to be performed when the trigger executes.	CO2	K1
54	**List the Types of Triggers** We can define 6 types of triggers for each table: 1. **AFTER INSERT** activated after data is inserted into the table. 2. **AFTER UPDATE:** activated after data in the table is modified. 3. **AFTER DELETE:** activated after data is deleted/removed from the table. 4. **BEFORE INSERT:** activated before data is inserted into the table. 5. **BEFORE UPDATE:** activated before data in the table is modified. 6. **BEFORE DELETE:** activated before data is deleted/removed from the table.	CO2	K1
55	**What is a procedure in SQL?** A procedure in SQL (often referred to as stored procedure), is a reusable unit that encapsulates the specific business logic of the application. A SQL procedure is a group of SQL statements and logic, compiled and stored together to perform a specific task. .	CO2	K1

11. Part B Questions

S.No	Question and Answers	CO	K
1	**Design a database schema for a library management system that includes tables for Books, Members, and Loans. Specify the relationships between the tables and include appropriate constraints. (CO2, K6)**	CO2	K6
2	**Write an SQL query that retrieves the top 5 employees with the highest salaries from each department. Explain how you would implement this using window functions. (CO2, K3)**	CO2	K3
3	**Create a stored procedure that accepts a department ID and returns the average salary of employees in that department. Include error handling for cases where the department does not exist. (CO2, K6)**	CO2	K6
4	**Discuss the implications of using cascading deletes in foreign key relationships. Provide an example of a scenario where this could lead to unintended data loss. (CO2, K4)**	CO2	K4
5	**Write an SQL statement to create a view that combines data from Employees and Departments to show the total salary expenditure for each department. Explain how this view can be used for reporting. (CO2, K3)**	CO2	K3
6	**Develop a trigger that automatically updates the LastUpdated timestamp column in an Employees table whenever a record is modified. Explain the importance of maintaining this timestamp. (CO2, K6)**	CO2	K6
7	**Construct a dynamic SQL query that allows users to filter results from a Products table based on various optional criteria (e.g., category, price range). Discuss how to prevent SQL injection in this scenario. (CO2, K3)**	CO2	K3
8	**Analyze the performance implications of using subqueries versus joins in SQL. Provide examples of when one might be preferred over the other. (CO2, K4)**	CO2	K4
9	**Create a function that calculates the total number of orders placed by a customer given their CustomerID. Discuss how this function can be integrated into other queries. (CO2, K6)**	CO2	K6
10	**Write an SQL statement to implement a check constraint that ensures the EndDate of an event is always after the StartDate. Discuss how this constraint helps maintain data integrity. (CO2, K3)**	CO2	K3
11	**Propose a method to implement version control for records in a Products table. Discuss how you would structure the table and manage updates to maintain historical data. (CO2, K6)**	CO2	K6

12. Part B Questions

S.No	Question and Answers	CO	K
12	**Create a complex SQL query that retrieves the names of customers who have placed orders in the last 30 days, along with the total amount spent by each customer. Explain the logic behind your query. (CO2, K3)**	**CO2**	**K3**
13	**Discuss the role of embedded SQL in application development. Provide an example of how embedded SQL can be used to enhance application performance and security. (CO2, K4)**	**CO2**	**K4**
14	**Evaluate the use of materialized views in SQL databases. Discuss their advantages and disadvantages, and provide an example of a scenario where a materialized view would be beneficial. (CO2, K5)** **.**	**CO2**	**K5**
15	**Design a cursor-based solution to iterate through all records in a Sales table and update the Status column based on specific business logic. Explain the advantages and disadvantages of using cursors. (CO2, K6)**	**CO2**	**K6**

12. SUPPORTIVE ONLINE CERTIFICATION COURSES

S.No.	Name of the Institute	Name of the Course	Website Link
1.	coursera	Database Management Essentials	https://www.coursera.org/learn/database-management
2.	coursera	Database systems Specialization	https://www.coursera.org/specializations/database-systems
3.	Udemy	Introduction to Database Engineering	https://www.udemy.com/course/database-engines-crash-course/
4.	Udemy	Relational Database Design	https://www.udemy.com/course/relational-database-design/
5.	Udemy	Database Design	https://www.udemy.com/cour se/database-design/
6.	Udemy	Database Design Introduction	https://www.udemy.com/course/cwdatabase-design-introduction/
7.	Udemy	The Complete Database Design & Modeling Beginners Tutorial	https://www.udemy.com/cour se/the complete-database-modeling-and-design- beginners-tutorial/
8.	Udemy	Database Design and MySQL	https://www.udemy.com/cour se/calebthevideomaker2-database-and-mysql-classes/
9.	NPTEL	Data Base Management System	https://onlinecourses.nptel.ac .in/noc21_cs04/preview
10.	Udemy	The Complete SQL Bootcamp 2021: Go from Zero to Hero	https://www.udemy.com/cour se/the-complete-sql-bootcamp/
11	Udemy	The Complete Oracle SQL Certification Course	https://www.udemy.com/cour se/the-complete-oracle-sql-certification-course/
12	Udemy	Microsoft SQL for Beginners	https://www.udemy.com/cour se/microsoft-sql-for-beginners/
13	Udemy	SQL for Data Analysis: Weekender Crash Course for Beginners	https://www.udemy.com/cour se/sql-for-newbs/

13. REAL TIME APPLICATIONS IN DAY TO DAY LIFE AND TO INDUSTRY

Application and Uses of Database Management System (DBMS)

- Railway Reservation System.
- Library Management System
- Banking System
- Universities and colleges Management Systems
- Credit card transactions.
- Social Media Sites
- Telecommunications
- Finance Applications

14. Contents beyond the Syllabus

With SQL being the de facto query language for traditional databases, it has probably made its place in almost all tech stacks currently being used to deploy production-grade applications. Even when traditional databases (read relational databases) are not in use, SQL inevitably shows itself in the system. Cassandra has SQL-like CQL, Spark has SparkSQL, and even JIRA has JQL, which resembles basic SQL clauses like `WHERE`, `IN`, `BETWEEN` , etc. These clauses don't just belong to SQL, they are also part of almost all programming languages and are used in looping and conditional constructs

MySQL

For the longest time, MySQL didn't support analytic and window functions. It just supported basic aggregations along with `GROUP BY`. However, drop-in replacements of MySQL, like Percona Server and MariaDB, had released these features even before MySQL.

Everything you can do from a SQL client or a SQL terminal shall be considered SQL. For instance, in MySQL, you can run a `SHOW SLAVE STATUS` command to check the current status of the slave and if the replication is working or not, how far it has progressed, and so on. This is not part of standard SQL and isn't supported similarly in the other databases. To get to know the SQL-92 standard, you can go through this 500-page document.

Some of the features unique of MySQL are

- Support for native complex replication topologies
- Session and global variables to emulate analytic functions and PL/SQL functionality
- Storage engines — not many databases support these many storage engines

PostgreSQL

Usually called the programmer's database, PostgreSQL has established itself as a real competitor to MySQL in the open-source space with native support for advanced abstract data structures like arrays. For instance, to check the replication status, you'd go and check this table — `pg_stat_replication`. Some of the features that make PostgreSQL unique are

- Extensible — you can add extensions to the database to enhance its functionality. PostGIS is a popular extension to PostgreSQL, a library for doing geospatial SQL queries.
- Extensive support for abstract data types, especially helpful to match usual data types in an application programming language.
- Unlike MySQL, PostgreSQL does extend itself to include a procedural language PL/pgSQL, similar to Oracle's PL/SQL

Oracle

What can be said about the Oracle database? It's probably the most resilient and durable database — which is why it continues to reign in time-critical businesses requiring the strongest ACIDity and the guarantees that come with that.
Some of the unique features of Oracle databases are

- Extremely strong ACID compliance, high reliability, high durability, and resiliency.
- Great support for hierarchical queries with `CONNECT BY ... LEVEL`.
- Full support of analytical and window functions for the longest time.

The one reason why MySQL and PostgreSQL have become more popular over the years is that they're free to use — they're open source. Oracle is the definition of closed-source. The licensing cost could kill a small company. Every business cannot afford proprietary databases like Oracle and SQL Server.

SQL Server

The ultimate database from Microsoft also boasts similar features and levels of ACID support, etc., just like Oracle. Again, just like Oracle, it's too costly for most companies.
Some of the unique features of SQL Server are

- Support for T-SQL, Microsoft's own procedural
- Advanced query statistics using Query Store for optimizing database and query performance
- Full support for analytic and window functions

These are the four main relational databases. I'm not counting DB/2, SQLite, etc., anymore because there are not enough new users for those databases. Wherever DB/2 is being used is probably a legacy system. SQLite is not feature-rich and is unsuitable for production-grade applications.

CQL

CQL is Cassandra's answer to SQL. It's very much like SQL, but again, like all the other databases you'll see from now on, this one, too, has additional features about the architectural implementation, which, in turn, necessitates additional features in SQL. For instance, CQL has something called `ALLOW FILTERING`

SparkSQL

Initially, Spark didn't have support for SQL queries, and the queries would have to be written with the Scala or Python APIs for DataFrames. SparkSQL provides another abstraction layer to make it easy for the SQL-educated workforce to fully use their knowledge.

HiveQL

With the advent of the Hadoop ecosystem, it quickly got very difficult to analyze huge amounts of data because everyone wasn't familiar with programming languages, and one would have had to write many lines of code to analyze the data. Facebook had a lot of data to analyze, so they developed this SQL-like query engine that ran on top of Hadoop's storage system. Hive is also meant to query only structured distributed data.

Hive query looks like:

```
CREATE TABLE sample_bucket (code STRING,
                            description STRING,
                            total_emp INT,
                            salary INT)
 CLUSTERED BY (code)
   SORTED BY (salary)
     INTO 5 BUCKETS;
```

InfluxQL

InfluxDB is a time-series database for applications that require real-time querying. It's a use case that traditional databases haven't supported fully. Some of the new-age streaming pipeline solutions provide this feature with some limitations.

```
CREATE CONTINUOUS QUERY "cq_basic_rp" ON "transportation"
BEGIN
  SELECT mean("passengers")
    INTO "transportation"."three_weeks"."average_passengers"
    FROM "bus_data"
   GROUP BY time(1h)
END
```

15. ASSESSMENT SCHEDULE

- **Tentative schedule for the Assessment During 2025-2026 Odd Semester:**

S.NO	Name of the Assessment	Start Date	End Date	Portion
1	Unit Test 1			UNIT 1
2	Internal Assessment Test 1			UNIT 1 & 2
3	Unit Test 2			UNIT 3
4	Internal Assessment Test 2			UNIT 3 & 4
5	Model Examination			ALL 5 UNITS

16. PRESCRIBED TEXT BOOKS &REFERENCE BOOKS

TEXT BOOKS:

1. Elmasri R. and S. Navathe, "Fundamentals of Database Systems", PearsonEducation, 7th Edition, 2016.

2. Abraham Silberschatz, Henry F.Korth, "Database System Concepts",TataMcGraw Hill , 7th Edition, 2021.

REFERENCES:

1. Raghu Ramakrishnan, Gehrke "Database Management Systems", MCGraw Hill, 3rd Edition 2014.
2. Plunkett T., B. Macdonald, "Oracle Big Data Hand Book" , McGraw Hill, First Edition, 2013
3. Gupta G K , "Database Management Systems" , Tata McGraw Hill Education Private Limited, New Delhi, 2011.
4. C. J. Date, A.Kannan, S. Swamynathan, "An Introduction to Database Systems", Eighth Edition, Pearson Education, 2015.

17. MINI PROJECT SUGGESTION

- **Objective:**

This module facilitate hands-on skills of the students (from the practical courses more effectively) and they can try the following mini projects for deep understanding in Compiler Design

Planning:

- This method is mostly used to improve the ability of students in application domain and also to reinforce knowledge imparted during the lecture.
- Being a technical institute, this method is extensively used to provide empirical evidence of theory learnt.
- Students are asked to prepare mini projects involving application of the concepts, principles or laws learnt.
- The faulty guides the students at various stages of developing the project and gives timely inputs for the development of the model.

Projects:

Set Number (Category)	Mini Project Title
Set - 1 (Toppers)	**University Course Enrollment System** Manage departments, Courses, Enrollments, Prerequisites and grades. (CO2, K6)
Set - 2 (Above Average)	**Crime Record Analysis System** Store and analyze crime data by region, type, and date. (CO2, K6)
Set - 3 (Average)	**Hospital Management Database** Store patient records, doctor schedules, medical histories, and billing. (CO2, K6)
Set - 4 (Below Average)	**Banking System Database** Manage accounts, transactions, loans, and customer information. (CO2, K6)
Set - 5 (Slow Learners)	**Student Record System** Maintain student details, marks, attendance, and class schedules.(CO2, K6)

Thank you