Vidyavardhini's
College of Engineering & Technology
Vasai Road (W)

Department of Artificial Intelligence and Data Science

Lab Manual

Semester	IV	Class	S.E
Course Code	CSL402	Academic Year	2021-22
Course Name	Database Management System		

Vidyavardhini's College of Engineering & Technology

Vision

To be a premier institution of technical education; always aiming at becoming a valuable resource for industry and society.

Mission

- To provide a technologically inspiring environment for learning.
- To promote creativity, innovation and professional activities.
- To inculcate ethical and moral values.
- To cater personal, professional and societal needs through quality education.

Department Vision:

To foster proficient artificial intelligence and data science professionals, making remarkable contributions to industry and society.

Department Mission:

- To encourage innovation and creativity with rational thinking for solving the challenges in emerging areas.
- To inculcate standard industrial practices and security norms while dealing with Data.
- To develop sustainable Artificial Intelligence systems for the benefit of various sectors.

Program Specific Outcomes (PSOs):

PSO1: Analyze the current trends in the field of Artificial Intelligence & Data Science and convey their findings by presenting / publishing at a national / international forum.

PSO2: Design and develop Artificial Intelligence & Data Science based solutions and applications for problems in the different domains catering to industry and society.

Program Outcomes (POs):

Engineering Graduates will be able to:

- **PO1. Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
- **PO2. Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
- **PO3. Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
- **PO4. Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

- **PO5. Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
- **PO6. The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
- **PO7. Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
- **PO8. Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
- **PO9. Individual and teamwork:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
- **PO10. Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
- **PO11. Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
- **PO12. Life-long learning:** Recognize the need for and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Course Objectives

1	To Develop Entity Relationship data model.
2	To develop relational Model
3	To formulate SQL queries.
4	To learn procedural interfaces to SQL queries
5	To learn the concepts of transactions and transaction processing
6	To understand how to handle concurrent transactions and able to access data through front end (using JDBC ODBC connectivity)

Course Outcomes

At the end of the course student will be able to:		Action verb	Bloom Level
CSL402.1	Design ER and EER diagrams for real life problems with software tools.	Design	Create (Level 6)
CSL402.2	Construct database tables with different DDL and DML statements and apply integrity constraints	Apply	Apply (Level 3)
CSL402.3	Apply SQL queries ,triggers for given Schema	Apply	Apply (Level 3)
CSL402.4	Apply procedure and functions for given schema	Apply	Apply (Level 3)
CSL402.5	Design ER and EER diagrams for the real life problem with software tool.	Use	Apply (Level 3)
CSL402.6	Construct database tables with different DDL and DML statements and apply integrity constraints	Construct	Apply(Level 3)

List of Experiments

Sr. No	Name of Experiments	Mode of conduction
1	Identify the case study and detailed statement of the problem. Design an Entity Relationship (ER) / Extended Entity Relationship (EER) Model.	2
2	Mapping ER/EER to Relational schema model.	2
3	Create a database using Data Definition Language (DDL) and apply integrity constraints for the specified System	2
4	Apply DML Commands for the specified system	2
5	Perform Simple queries, string manipulation operations and aggregate functions.	2
6	Implement various Join operations.	2
7	Perform DCL and TCL commands	2
8	Implementation of Views and Triggers.	2
9	Demonstrate Database connectivity	2
10	Implementation and demonstration of Transaction and Concurrency control techniques using locks	2

Mapping of Experiments with Course Outcomes

Course Modules	Course Outcomes					
	CSL402.1	CSL402.2	CSL402.3	CSL402.4	CSL402.5	CSL402.6
Identify the case study and detailed statement of the problem. Design an Entity Relationship (ER) / Extended Entity Relationship (EER) Model.	3					
Mapping ER/EER to Relational schema model.	3					
Create a database using Data Definition Language (DDL) and apply integrity constraints for the specified System		3				
Apply DML Commands for the specified system		3				
Perform Simple queries, string manipulation operations and aggregate functions.			3			
Implement various Join operations.				3		

Perform DCL and TCL commands				3		
Implementation of Views and Triggers.					3	
Demonstrate Database connectivity						3
Implementation and demonstration of Transaction and Concurrency control techniques using locks						3

Enter correlation level 1, 2 or 3 as defined below
1: Slight (Low) 2: Moderate (Medium) 3: Substatial (High)
If there is no correlation put "—".

Experiment No.1
Design an EntityRelationship (ER) / Extended Entity-Relationship (EER) Model.
Date of Performance:
Date of Submission:

Aim :- Identify the case study and detailed statement of the problem. Design an EntityRelationship (ER) / Extended Entity-Relationship (EER) Model.

Objective :- To identify and explore a real world problem, and to design an Entity Relationship (ER) / Extended Entity-Relationship (EER) Model.

Theory:

1. **Entity:**

- An entity is a real-world object or concept that exists independently and has distinguishable attributes.
- In a database context, an entity represents a table, and each row in that table represents a unique instance of that entity.
- For example, in a university database, entities could include Student, Course, Professor, Department, etc.
- Each entity has a set of attributes that describe its properties.

2. **Attributes:**

- Attributes are the properties or characteristics that describe an entity.
- They represent the data we want to store about each instance of an entity.
- For example, attributes of a Student entity might include StudentID, Name, Age, GPA, etc.
- Attributes can be categorized as simple (atomic) attributes, which cannot be divided further, or composite attributes, which are made up of smaller sub-parts.

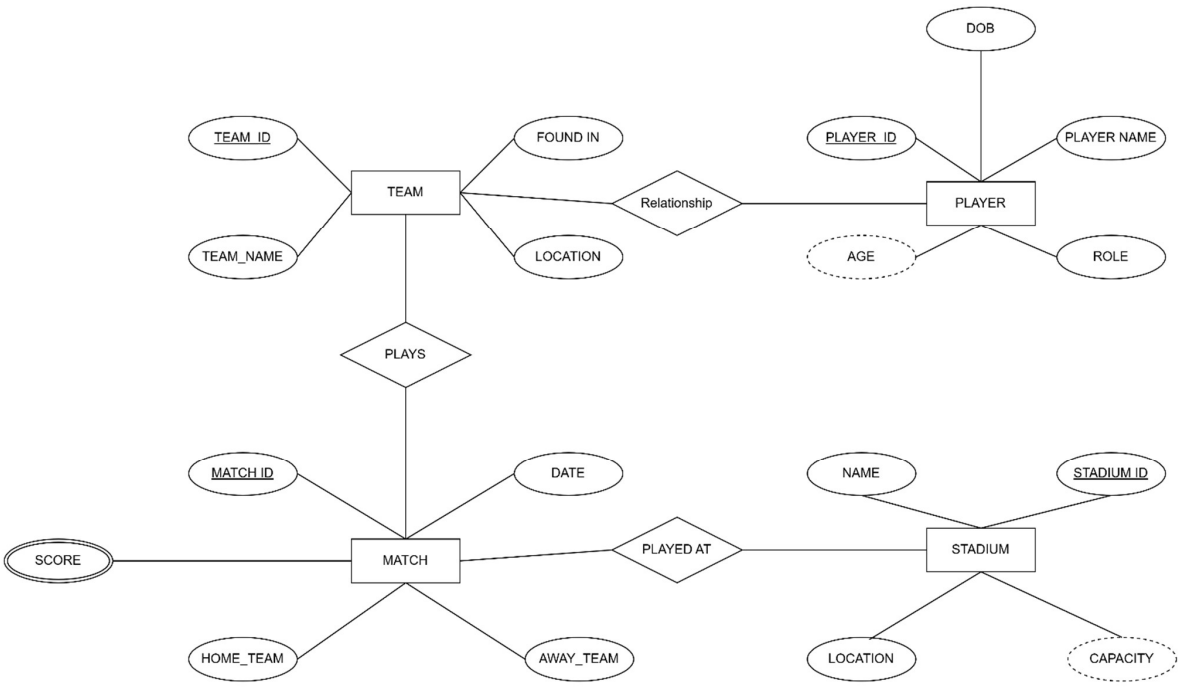
3. **Relationships:**

- Relationships describe how entities are related to each other or how they interact.
- They represent the associations between entities.
- Relationships are depicted as lines connecting related entities in the ER diagram.
- Each relationship has a degree, indicating the number of entities involved. It could be unary (involving one entity), binary (involving two entities), or ternary (involving three entities).
- Relationships also have cardinality, which defines the number of instances of one entity that can be associated with the number of instances of another entity through the relationship.

4. Cardinality:

- Cardinality specifies the number of instances of one entity that are related to the number of instances of another entity through a relationship.
- It defines the maximum and minimum number of occurrences of one entity that can be associated with the occurrences of another entity.
- Common cardinality constraints include:
  - I. One-to-One (1:1): Each instance of one entity is associated with exactly one instance of another entity, and vice versa.
  - II. One-to-Many (1:N): Each instance of one entity is associated with zero or more instances of another entity, but each instance of the second entity is associated with exactly one instance of the first entity.
  - III. Many-to-One (N:1): The reverse of One-to-Many; many instances of one entity are associated with one instance of another entity.
  - IV. Many-to-Many (N:N): Many instances of one entity can be associated with many instances of another entity.

Implementation:



Conclusion:

1. Define Entity, Attributes (also types) and Relationship between entities

Entity: Represents a real-world object or concept in a database.
Attributes: Characteristics of entities with defined data types.
Relationships: Connections between entities, defining how they interact or associate with each other.

2. Write ER/EER diagram notations

Entities: Rectangles or ovals.
Attributes: Ellipses connected to entities by lines.
Primary Key Attribute: Underlined attribute.
Composite Attribute: Attributes nested within ellipses.
Multivalued Attribute: Double oval.
Derived Attribute: Dashed ellipse.
Relationships: Diamonds or lines connecting entities.
Cardinality: Indicates the minimum and maximum number of instances in a relationship using crow's foot notation.
Participation Constraint: Total participation denoted by a double line.

Experiment No.2
Mapping ER/EER to Relational schema model.
Date of Performance:
Date of Submission:

Aim :- Prepare the schema for Relational Model with the ER/ERR diagram, drawn for the identified case study in experiment no.1.

Objective :- To map the Entity Relationship (ER) / Extended Entity-Relationship (EER) Diagram to Relational Model schema and learn to incorporate various schema-based constraints.

Theory:

Mapping an Entity-Relationship (ER) model to a relational database schema involves translating the conceptual model represented in the ER diagram into tables and relationships in a relational database management system (DBMS). Here are the general rules for mapping ER to a schema in a DBMS:

1. Entities to Tables:
    a. Each entity in the ER diagram corresponds to a table in the relational schema.
    b. The attributes of the entity become the columns of the table.
    c. The primary key of the entity becomes the primary key of the table.

2. Relationships to Tables:
    a. Many-to-Many Relationships:
        i. Convert each many-to-many relationship into a new table.
        ii. Include foreign key columns in this table to reference the participating entities.
        iii. The primary key of this table may consist of a combination of the foreign keys from the participating entities.
    b. One-to-Many and One-to-One Relationships:
        i. Represented by foreign key columns in one of the participating tables.
        ii. The table on the "many" side of the relationship includes the foreign key column referencing the table on the "one" side.
        iii. The foreign key column typically references the primary key of the related table.

3. Attributes to Columns:
    a. Each attribute of an entity becomes a column in the corresponding table.
    b. Choose appropriate data types for each attribute based on its domain and constraints.
    c. Ensure that attributes participating in relationships are represented as foreign keys when needed.

4. Primary and Foreign Keys:
    a. Identify the primary key(s) of each table based on the primary key(s) of the corresponding entity.
    b. Ensure referential integrity by defining foreign keys in tables to establish relationships between them.
    c. Foreign keys should reference the primary key(s) of related tables.

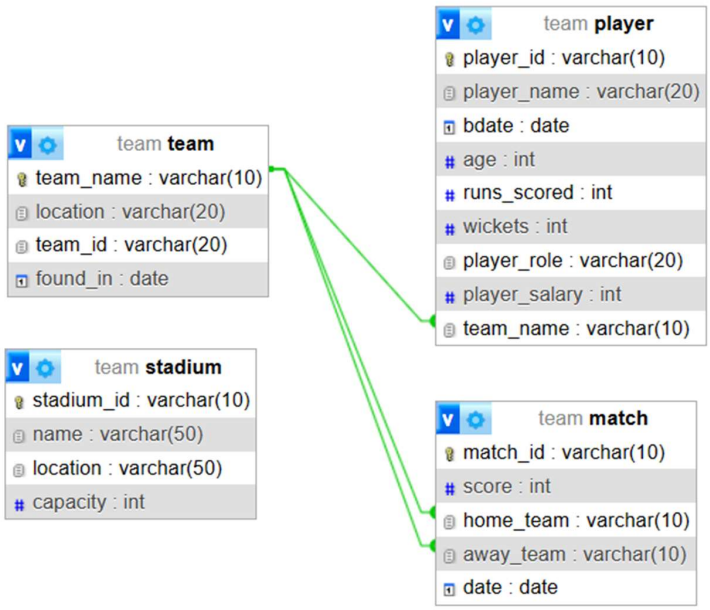
d. Ensure that foreign keys have appropriate constraints, such as ON DELETE CASCADE or ON UPDATE CASCADE, to maintain data integrity.

5. Cardinality Constraints:
   a. Use the cardinality constraints from the ER diagram to determine the multiplicity of relationships in the relational schema.
   b. Ensure that the constraints are enforced through the appropriate use of primary and foreign keys.

6. Normalization:
   a. Normalize the schema to minimize redundancy and dependency.
   b. Follow normalization rules such as First Normal Form (1NF), Second Normal Form (2NF), Third Normal Form (3NF), etc., to ensure data integrity and minimize anomalies.

7. Indexing and Optimization:
   a. Consider indexing frequently queried columns to improve query performance.
   b. Evaluate the schema design for optimization opportunities based on query patterns and performance requirements.

Implementation:



Conclusion:

1. write definition of relational schema and notations

Relational Schema and Notations:

Definition: Relational schema organizes data in a database. It consists of relation schemas, detailing attributes and types. Notations represent these schemas.

Notations:

Relation Name: Uppercase.
Attribute Names: Lowercase.
Attribute Types: Data types.
Primary Key: Underlined.
Foreign Key: Dashed underline or different font.
Referential Integrity Constraints: Arrows connecting foreign keys to primary keys.

2. write various schema-based constraints

Various Schema-based Constraints:

Primary Key Constraint: Ensures uniqueness.
Foreign Key Constraint: Maintains referential integrity.
Unique Constraint: Enforces uniqueness.
Check Constraint: Specifies conditions for validity.
Not Null Constraint: Requires non-null values.
Default Constraint: Provides default values.
Check Constraint: Limits acceptable values.
Assertion Constraint: Global condition for the database.

Experiment No.3
Create a database using Data Definition Language(DDL) and apply integrity constraints for the specified system
Date of Performance:
Date of Submission:

Aim:- Write a query to create tables for each relation in the relational schema of experiment no.2. Apply drop and alter commands on those tables.

Objective:- To learn commands of Data Definition Language(DDL) to create and define databases, and also learn to apply integrity constraints for the specified system.

Theory:

DDL Commands & Syntax:

Data Definition Language (DDL) is a subset of SQL and a part of DBMS(Database Management System). DDL consist of Commands to commands like CREATE, ALTER, TRUNCATE and DROP. These commands are used to create or modify the tables in SQL.

DDL Commands:

1. Create
2. Alter
3. truncate
4. drop
5. Rename

CREATE:

This command is used to create a new table in SQL. The user must give information like table name, column names, and their data types.

Syntax-

CREATE TABLE table name(

column 1 datatype,

column_2 datatype,

column_ 3 datatype,

….

);

ALTER:

This command is used to add, delete or change columns in the existing table. The user needs to know the existing table name and can add, delete, or modify tasks easily.

Syntax-

ALTER TABLE table_name

ADD column name datatype;

TRUNCATE :
This command is used to remove all rows from the table, but the structure of the table still exists.

Syntax_

TRUNCATE TABLE table_name;

DROP :

This command is used to remove an existing table along with its structure from the Database.

Syntax-

DROP TABLE table name;

RENAME :

It is possible to change name of table with or without data in it using simple RENAME command. We can rename any table object at any point of time.

Syntax-

RENAME TABLE <Table_Name> To <New_Table_Name>;

Implementation:

Database:

```
create database teams;
use teams;
```

```
14  17:40:54  create database teams
15  17:40:54  use teams
```

Table:

```
create table player (
player_name varchar(20),
bdate date ,
age int,
runs int,
wickets int,
player_role varchar (29),
player_salary varchar(10)
);
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

player_name	bdate	age	runs	wickets	player_role	player_salary

Result Grid

Alter:

```
alter table player
add player_id varchar(10);
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

player_name	bdate	age	runs	wickets	player_role	player_salary	player_id

Result Grid

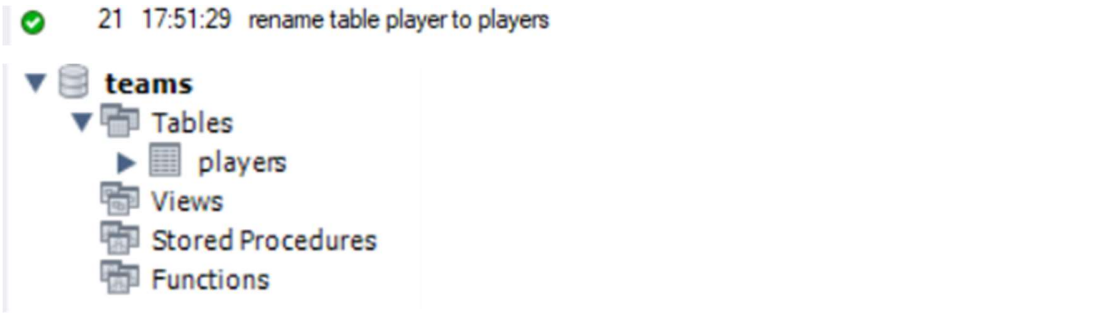
Truncate:

```
truncate table player;
```

```
20  17:49:26  truncate table player
```

Rename:

```
rename table player to players;
```



Drop:

Conclusion:

1. Explain the concept of constraints in DDL. How are constraints used to enforce data integrity?

   Constraints in DDL (Data Definition Language) are rules or conditions applied to the columns in a database table to enforce data integrity. They ensure that the data stored in the database meets certain criteria, preventing invalid or inconsistent data from being entered. Constraints can enforce various rules such as uniqueness, referential integrity, and data validity. For example, a primary key constraint ensures that each row in a table has a unique identifier, while a foreign key constraint ensures that values in a column match values in another table's column. By enforcing these constraints, data integrity is maintained, ensuring the accuracy, consistency, and reliability of the database.

2. What is the significance of data types in DDL? Provide examples of commonly used data types in DDL.

Data types in DDL specify the type of data that can be stored in a column of a database table. They define the format and range of values that can be assigned to a column, ensuring proper storage and manipulation of data. Commonly used data types in DDL include:

INTEGER: Used for storing whole numbers.

VARCHAR(n): Variable-length character string with a maximum length of 'n' characters.

DATE: Used for storing dates in the format 'YYYY-MM-DD'.

DECIMAL(p, s): Exact numeric data type with 'p' total digits and 's' digits to the right of the decimal point.

BOOLEAN: Used for storing true/false or 1/0 values.

TIMESTAMP: Used for storing date and time values in the format 'YYYY-MM-DD HH:MM:SS'.

CHAR(n): Fixed-length character string with a length of 'n' characters.

FLOAT: Approximate numeric data type used for storing floating-point numbers.

TEXT: Variable-length character string for storing large blocks of text. These data types allow database designers to define the structure of the database tables and ensure the appropriate handling of data.

Experiment No.4
Apply DML commands for the specified system
Date of Performance:
Date of Submission:

Aim :- Write insert query to insert rows for each table created of your database management system. Use update and delete commands to manipulate the inserted values in the table.

Objective :- To learn commands of Data Manipulation Language(DML) to insert, update or delete the values in the database system.

Theory:

Data Manipulation Language (DML) is a subset of SQL (Structured Query Language) used for managing data within relational database management systems (RDBMS). DML commands are used to perform operations such as inserting, updating, and deleting data from database tables.

1. Inserting Data

The INSERT statement is used to add new rows of data into a table. It specifies the table to insert data into and provides values or expressions for each column in the new row. If a column list is not specified, values must be provided for all columns in the table in the order they were defined.

Syntax:-

INSERT INTO table_name (column1, column2, column3) VALUES (value1, value2,

value3); 2. Updating Data

The UPDATE statement is used to modify existing data within a table. It allows you to change the values of one or more columns in one or more rows based on specified conditions. If no condition is specified, all rows in the table will be updated.

Syntax:

UPDATE table_name SET column1 = value1, column2 = value2 WHERE

condition; 3. Deleting Data

The DELETE statement is used to remove one or more rows from a table based on specified conditions. If no condition is specified, all rows in the table will be deleted.

Syntax:

DELETE FROM table_name WHERE condition;

Implementation:

1. INSERT:

```sql
INSERT INTO Team (team_name, location, team_id, found_in)
VALUES
    ('Blasters', 'Mumbai', 'MI', '2020-01-01'),
    ('Fixers', 'Chennai', 'CSK', '2019-03-15'),
    ('Goldens', 'Bengaluru', 'RCB', '2021-07-10');

INSERT INTO Player (player_id, player_name, bdate, age, runs_scored, wickets, player_role, player_salary, team_name)
VALUES
    ('P45', 'Rohit Sharma', '1995-05-20', 27, 500, 20, 'Batsman', 50000, 'Blasters'),
    ('P07', 'MS Dhoni', '1998-08-15', 24, 300, 15, 'Wicket-Keeper', 48000, 'Fixers'),
    ('P18', 'Virat Kohli', '1993-12-10', 28, 700, 10, 'All-Rounder', 55000, 'Goldens');

INSERT INTO `Match` (match_id, score, home_team, away_team, date)
VALUES
    ('M001', 250, 'Blasters', 'Fixers', '2024-04-15'),
    ('M002', 300, 'Fixers', 'Goldens', '2024-04-18'),
    ('M003', 200, 'Goldens', 'Blasters', '2024-04-20');

INSERT INTO Stadium (stadium_id, name, location, capacity)
VALUES
    ('S001', 'Wankhede', 'Mumbai', 50000),
    ('S002', 'Stadium un Thala', 'Chennai', 60000),
    ('S003', 'Stad de EducatedSarr', 'Bengaluru', 70000);
```

	player_id	player_name	bdate	age	runs_scored	wickets	player_role	player_salary	team_name
Edit Copy Delete	P07	MS Dhoni	1998-08-15	24	300	15	Wicket-Keeper	48000	Fixers
Edit Copy Delete	P18	Virat Kohli	1993-12-10	28	700	10	All-Rounder	55000	Goldens
Edit Copy Delete	P45	Rohit Sharma	1995-05-20	27	500	20	Batsman	50000	Blasters

	match_id	score	home_team	away_team	date
Edit Copy Delete	M001	250	Blasters	Fixers	2024-04-15
Edit Copy Delete	M002	300	Fixers	Goldens	2024-04-18
Edit Copy Delete	M003	200	Goldens	Blasters	2024-04-20

2. UPDATE:

```sql
UPDATE Player SET player_name = 'Abhishek Sharma' WHERE player_id = 'P18';
```

	player_id	player_name	bdate	age	runs_scored	wickets	player_role	player_salary	team_name
Edit Copy Delete	P07	MS Dhoni	1998-08-15	24	300	15	Wicket-Keeper	48000	Fixers
Edit Copy Delete	P18	Abhishek Sharma	1993-12-10	28	700	10	All-Rounder	55000	Goldens
Edit Copy Delete	P45	Rohit Sharma	1995-05-20	27	500	20	Batsman	50000	Blasters

3. DELETE:

```
1 DELETE FROM `Match` WHERE match_id = 'M002';
```

	match_id	score	home_team	away_team	date
Edit Copy Delete	M001	250	Blasters	Fixers	2024-04-15
Edit Copy Delete	M003	200	Goldens	Blasters	2024-04-20

Conclusion:

1. Explain the role of database constraints in enforcing data integrity during DML operations.

   Database constraints play a crucial role in enforcing data integrity during DML (Data Manipulation Language) operations by imposing rules and conditions on the data stored in the database tables. These constraints ensure that the data conforms to certain standards and requirements, preventing the insertion, modification, or deletion of data that could compromise its integrity. Constraints such as primary key, foreign key, unique, and check constraints help maintain consistency, accuracy, and reliability in the database by preventing invalid or inconsistent data from being introduced or manipulated.

2. How do you update multiple columns in a table using a single UPDATE statement?

   To update multiple columns in a table using a single UPDATE statement, you specify the column names and their corresponding new values separated by commas within the SET clause of the UPDATE statement. For example:

   UPDATE table_name
   SET column1 = value1, column2 = value2, column3 = value3
   WHERE condition;

   This statement updates the values of column1, column2, and column3 in the specified table with the provided values, subject to the specified condition.

Experiment No.5
Perform simple queries, string manipulation operations and aggregate functions.
Date of Performance:
Date of Submission:

Aim :- Write simple query to manipulate string operations and perform aggregate functions like (MIN, MAX, SUM, AVERAGE, COUNT).

Objective :- To apply aggregate functions and string manipulation functions to perform simple queries in the database system

Theory:

Simple Queries in SQL:

In SQL, a simple query is a request for data from a database table or tables. It allows users to retrieve specific information by specifying the columns they want to retrieve and any conditions for filtering rows based on certain criteria. Simple queries are the backbone of interacting with databases, enabling users to extract the data they need for analysis, reporting, or further processing.

String Manipulation Operations:

String manipulation operations in SQL involve modifying or transforming string values stored in database columns. These operations are crucial for tasks such as formatting data, combining strings, converting case, or extracting substrings. By using string functions and operators, users can manipulate text data to suit their requirements, whether it's for display purposes or for further analysis.

Aggregate Functions:

Aggregate functions in SQL are used to perform calculations on sets of values and return a single result. These functions allow users to summarize data across multiple rows, providing insights into the overall characteristics of the dataset. Common aggregate functions include calculating counts, sums, averages, minimums, and maximums of numerical values. They are essential tools for data analysis, enabling users to derive meaningful insights from large datasets.

Benefits of Understanding These Concepts:

- Data Retrieval: Simple queries allow users to fetch specific data from databases, facilitating data retrieval for various purposes.
- Data Transformation: String manipulation operations enable users to format and transform text data according to their needs, improving data consistency and readability.
- Data Analysis: Aggregate functions help users summarize and analyze large datasets, providing valuable insights into trends, patterns, and statistical measures.
- Data Reporting: By combining simple queries, string manipulation operations, and aggregate functions, users can generate reports and visualizations that communicate key findings effectively.

Implementation:

```sql
use team;
SELECT MIN(age) AS min_age FROM Player;
SELECT MAX(runs_scored) AS max_runs_scored FROM Player;
SELECT SUM(runs_scored) AS total_runs_scored FROM Player;
SELECT AVG(player_salary) AS average_salary FROM Player;
SELECT COUNT(*) AS total_matches FROM `Match`;
SELECT CONCAT(player_name, ' - ', player_role) AS player_info FROM Player;

```

Conclusion:

1. Write syntax and explanation for each of the five aggregate functions

Aggregate functions are SQL functions used to perform calculations on sets of values and return a single result. Here's a brief explanation of each aggregate function along with the syntax and an example of its usage:

- MIN: Finds the smallest value in a set.

Syntax: MIN(column_name)

Example: SELECT MIN(age) AS min_age FROM Player;

- MAX: Finds the largest value in a set.

Syntax: MAX(column_name)

Example: SELECT MAX(runs_scored) AS max_runs_scored FROM Player;

- SUM: Calculates the sum of values in a set.

Syntax: SUM(column_name)

Example: SELECT SUM(runs_scored) AS total_runs_scored FROM Player;

- AVG: Calculates the average value in a set.

Syntax: AVG(column_name)

Example: SELECT AVG(player_salary) AS average_salary FROM Player;

- COUNT: Counts the number of rows in a set.

Syntax: COUNT(*) or COUNT(column_name)

Example: SELECT COUNT(*) AS total_matches FROM Match;

2. Show results of operations performed.

min_age
24

max_runs_scored
700

total_runs_scored
1500

average_salary
51000.0000

total_matches
2

player_info
MS Dhoni - Wicket-Keeper
Abhishek Sharma - All-Rounder
Rohit Sharma - Batsman

Experiment No.6
Implement various join operations
Date of Performance:
Date of Submission:

Aim :- Write simple query to implement join operations(equi join, natural join, inner join, outer joins).

Objective :- To apply different types of join to retrieve queries from the database management system.

Theory:

SQL Join statement is used to combine data or rows from two or more tables based on a common field between them. Different types of Joins are as follows:

- INNER JOIN
- LEFT JOIN
- RIGHT JOIN
- FULL JOIN

A. INNER JOIN

The INNER JOIN keyword selects all rows from both the tables as long as the condition is satisfied. This keyword will create the result-set by combining all rows from both the tables where the condition satisfies i.e value of the common field will be the same.

Syntax:

SELECT table1.column1,table1.column2,table2.column1,....

FROM table1

INNER JOIN table2

ON table1.matching_column = table2.matching_column;

table1: First table.

table2: Second table

matching_column: Column common to both the tables.

B. LEFT JOIN

This join returns all the rows of the table on the left side of the join and matches rows for the table on the right side of the join. For the rows for which there is no matching row on the right side, the result-set will contain *null*. LEFT JOIN is also known as LEFT OUTER JOIN.

Syntax:

SELECT table1.column1,table1.column2,table2.column1,....

FROM table1

LEFT JOIN table2

ON table1.matching_column = table2.matching_column;

table1: First table.

table2: Second table

matching_column: Column common to both the tables.

C. RIGHT JOIN

RIGHT JOIN is similar to LEFT JOIN. This join returns all the rows of the table on the right side of the join and matching rows for the table on the left side of the join. For the rows for which there is no matching row on the left side, the result-set will contain *null*. RIGHT JOIN is also known as RIGHT OUTER JOIN.

Syntax:

SELECT table1.column1,table1.column2,table2.column1,....

FROM table1

RIGHT JOIN table2

ON table1.matching_column = table2.matching_column;

table1: First table.

table2: Second table

matching_column: Column common to both the tables.

D. FULL JOIN

FULL JOIN creates the result-set by combining results of both LEFT JOIN and RIGHT JOIN. The result-set will contain all the rows from both tables. For the rows for which there is no matching, the result-set will contain NULL values.

Syntax:

SELECT table1.column1,table1.column2,table2.column1,....

FROM table1

FULL JOIN table2

ON table1.matching_column = table2.matching_column;

table1: First table.

table2: Second table

matching_column: Column common to both the tables.

Implementation:

```
use team;
-- INNER JOIN
SELECT Player.player_name, Team.team_name
FROM Player
INNER JOIN Team ON Player.team_name = Team.team_name;
-- LEFT JOIN
SELECT Player.player_name, `Match`.match_id
FROM Player
LEFT JOIN `Match` ON Player.player_id = `Match`.match_id;
-- RIGHT JOIN
SELECT Player.player_name, `Match`.match_id
FROM Player
RIGHT JOIN `Match` ON Player.player_id = `Match`.match_id;
```

player_name	team_name
MS Dhoni	Fixers
Abhishek Sharma	Goldens
Rohit Sharma	Blasters

player_name	match_id
MS Dhoni	*NULL*
Abhishek Sharma	*NULL*
Rohit Sharma	*NULL*

player_name	match_id
NULL	M001
NULL	M003

Conclusion:

1. Illustrate how to perform natural join for the joining attributes with different names with a suitable example.

   Performing a natural join with joining attributes having different names requires explicitly specifying the join condition. Here's a concise example:

   Example:

   SELECT *
   FROM Employees
   NATURAL JOIN Departments
   ON Employees.dept_id = Departments.department_id;

   In this example, Employees and Departments tables have different column names (dept_id and department_id). The ON clause specifies the common columns for the natural join.

2. Illustrate significant differences between natural join, equi-join and inner join.

Differences Between Natural Join, Equi Join, and Inner Join:

Natural Join: Automatically matches columns with the same name but can produce unexpected results.

Equi Join: Specifies join conditions explicitly, allowing joining attributes with different names.

Inner Join: Returns rows that satisfy the join condition specified in the ON clause, providing control over the join condition.

Experiment No.7
Perform DCL and TCL commands
Date of Performance:
Date of Submission:

Aim :- Write a query to implement Data Control Language(DCL) and Transaction Control Language(TCL) commands

Objective :- To learn DCL commands like Grant and Revoke privileges to the user and TCL commands to commit the transactions and recover it using rollback and save points.

Theory:

Data Control Language:

DCL commands are used to grant and take back authority from any database user.

- Grant
- Revoke

a. Grant: It is used to give user access privileges to a database.

Example

1. GRANT SELECT, UPDATE ON MY_TABLE TO SOME_USER, ANOTHER_USER;

b. Revoke: It is used to take back permissions from the user.

Example

1. REVOKE SELECT, UPDATE ON MY_TABLE FROM USER1, USER2;

Transaction Control Language

TCL commands can only use with DML commands like INSERT, DELETE and UPDATE only.

These operations are automatically committed in the database that's why they cannot be used while creating tables or dropping them.

Here are some commands that come under TCL:

- COMMIT
- ROLLBACK
- SAVEPOINT

a. Commit: Commit command is used to save all the transactions to the database.

Syntax:

```
1. COMMIT;
```

Example:

```
1. DELETE FROM CUSTOMERS
2. WHERE AGE = 25;
3. COMMIT;
```

b. Rollback: Rollback command is used to undo transactions that have not already been saved to the database.

Syntax:

```
1. ROLLBACK;
```

Example:

```
1. DELETE FROM CUSTOMERS
2. WHERE AGE = 25;
3. ROLLBACK;
```

c. SAVEPOINT: It is used to roll the transaction back to a certain point without rolling back the entire transaction.

Syntax:

```
2.  SAVEPOINT SAVEPOINT_NAME;
```

Implementation:

```
use team;
CREATE TABLE IF NOT EXISTS my_table (
    id INT AUTO_INCREMENT PRIMARY KEY,
    column1 VARCHAR(50),
    column2 VARCHAR(50)
);
-- Grant privileges to the user
GRANT SELECT, INSERT, UPDATE, DELETE ON my_table TO 'root'@'localhost';

-- Revoke privileges from the user
REVOKE SELECT, INSERT, UPDATE, DELETE ON my_table FROM 'root'@'localhost';

-- Begin a transaction
START TRANSACTION;

-- Insert data into a table within the transaction
INSERT INTO my_table (column1, column2) VALUES ('value1', 'value2');

-- Savepoint in the transaction
SAVEPOINT savepoint_name;

-- Rollback to a savepoint
ROLLBACK TO savepoint_name;

-- Commit the transaction
COMMIT;

-- Rollback the entire transaction
ROLLBACK;

```

Conclusion:

1. Explain about issues faced during rollback in mysql and how it got resolved.

During rollback in MySQL, issues can arise if there are concurrent transactions or if the rollback process encounters errors such as deadlocks. These issues are resolved by ensuring proper transaction management, handling deadlock situations, and using appropriate isolation levels to minimize conflicts between transactions.

2. Explain how to create a user in sql.

To create a user in SQL, you typically use the CREATE USER statement followed by the username and password. Optionally, you can specify additional parameters such as permissions and privileges. For example:

CREATE USER 'username'@'hostname' IDENTIFIED BY 'password';

Experiment No.8
Implementation of Views and Triggers
Date of Performance:
Date of Submission:

Aim :- Write a SQL query to implement views and triggers

Objective :- To learn about virtual tables in the database and also PLSQL constructs

Theory:

SQL Views:

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;
```

Trigger: A trigger is a stored procedure in the database which automatically invokes whenever a special event in the database occurs. For example, a trigger can be invoked when a row is inserted into a specified table or when certain table columns are being updated.

Syntax:

create trigger [trigger_name]

[before | after]

{insert | update | delete}

on [table_name]

[for each row]

[trigger_body]

Explanation of syntax:

1. create trigger [trigger_name]: Creates or replaces an existing trigger with the trigger_name.
2. [before | after]: This specifies when the trigger will be executed.
3. {insert | update | delete}: This specifies the DML operation.
4. on [table_name]: This specifies the name of the table associated with the trigger. 5. [for each row]: This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected.
6. [trigger_body]: This provides the operation to be performed as trigger is fired

Conclusion:

1. Brief about the benefits for using views and triggers.

   Views simplify queries, enhance security, abstract table structures, and optimize performance. Triggers enforce data integrity, audit changes, enforce business logic, and support replication.

2. Explain different strategies to update views

   Updating views can be done directly, by updating base tables, using triggers, or by recreating views. These methods offer varying degrees of control and are applied based on the view's complexity and update requirements.

Experiment No.9
Demonstrate Database connectivity
Date of Performance:
Date of Submission:

Aim :- Write a java program to connect Java application with the MySQL database

Objective :- To learn database connectivity

Theory:

Database used : MySql

1. Driver class: The driver class for the mysql database is com.mysql.jdbc.Driver. 2. Connection URL: The connection URL for the mysql database is jdbc:mysql://localhost:3306/loan management where jdbc is the API, mysql is the database, localhost is the server name on which mysql is running, can also use IP address, 3306 is the port number and loan management is the database name. 3. Username: The default username for the mysql database is Hiren.
4. Password: It is the password given by the user at the time of installing the mysql database. Password used is " ".

To connect a Java application with the MySQL database, follow the following steps.

● First create a database and then create a table in the mysql database. ● To connect java application with the mysql database, mysqlconnector.jar file is required to be loaded.

● download the jar file mysql-connector.jar
● add the jar file to the same folder as the java program.
● Compile and run the java program to retrieve data from the database.

Conclusion: Data has been retrieved successfully from a table by establishing database connectivity of java program with mysql database.

2. Explain steps to connect a java application with the MySQL database

- Download and install MySQL Connector/J.
- Include the MySQL Connector/J JAR file in your Java project.
- Import the necessary classes from the `java.sql` package.
- Use the `DriverManager.getConnection()` method to establish a connection to the MySQL database.
- Provide the JDBC URL, username, and password for authentication.
- Perform database operations using `Statement` or `PreparedStatement` objects.
- Close the connection and resources after completing the database operations.

Experiment No.10
Implementation and demonstration of Transaction and Concurrency control techniques using locks
Date of Performance:
Date of Submission:

Aim :- Write a query to lock and unlock a table for transaction and concurrency control.

Objective :- To learn locking of tables for transaction processing and concurrency control. **Theory:**

A lock is a mechanism associated with a table used to restrict the unauthorized access of the data in a table. MySQL allows a client session to acquire a table lock explicitly to cooperate with other sessions to access the table's data. MySQL also allows table locking to prevent unauthorized modification into the same table during a specific period.

Table Locking in MySQL is mainly used to solve concurrency problems. It will be used while running a transaction, i.e., first read a value from a table (database) and then write it into the table (database).

MySQL provides two types of locks onto the table, which are:

READ LOCK: This lock allows a user to only read the data from a table. WRITE

LOCK: This lock allows a user to do both reading and writing into a table. The

following is the syntax that allows us to acquire a table lock explicitly: LOCK

TABLES table_name [READ | WRITE];

The following is the syntax that allows us to release a lock for a table in MySQL:

UNLOCK TABLES;

Conclusion: Locking and unlocking of tables is achieved and verified using insert command in the same table of a database system.

1. Explain Transaction and Concurrency control techniques using locks.

Transactions ensure that a series of database operations are executed as a single unit of work, either all succeed or none do. Concurrency control techniques using locks manage access to shared resources in a multi-user environment. Locks can be applied at various levels, such as database, table, or row, to prevent conflicts between transactions.

In short, transactions ensure atomicity, consistency, isolation, and durability of database operations, while concurrency control techniques using locks prevent data inconsistency and ensure data integrity by managing concurrent access to shared resources.