Experiment 08

Use of different SQL clauses and join

Aim:

To design and implement a relational database for managing student and sports match data, enabling analysis of student participation in various matches.

Problem Statement:

The task involves creating a database to store information about students and the matches they play in, facilitating queries to extract insights on student participation, match venues, and student demographics.

Theory:

Relational databases organize data into structured tables, allowing for efficient querying and manipulation. SQL (Structured Query Language) is utilized for data definition and manipulation. Key concepts include:

- Tables: Structures to hold data in rows and columns.
- Primary Keys: Unique identifiers for records in a table, ensuring data integrity.
- Foreign Keys: References to primary keys in other tables, establishing relationships.
- JOIN Operations: Methods to combine rows from two or more tables based on related columns.
- Aggregate Functions: Functions such as `AVG()`, `COUNT()`, and `DISTINCT` to perform calculations and return summary information.

Commands Used:

1. Database and Table Creation
2. Data Insertion

3. Data Retrieval Queries

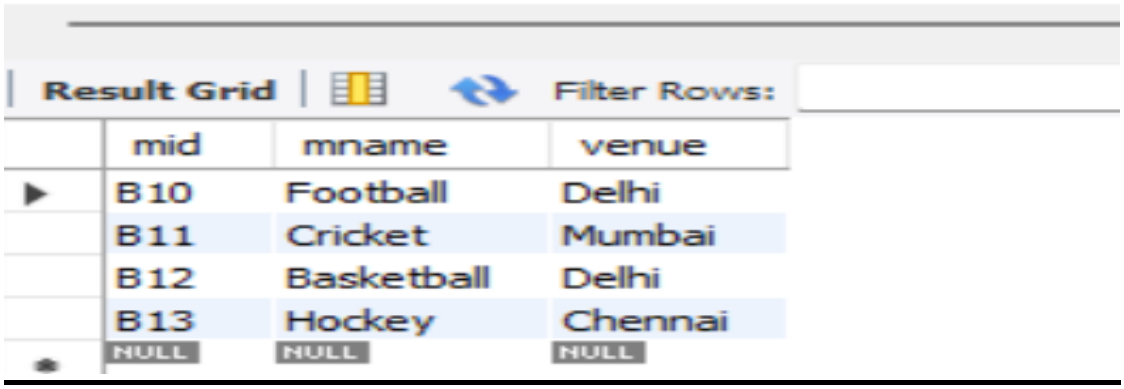
Results:

```
4   CREATE DATABASE exp9;
5   USE exp9;
6
7   -- Creating Tables:
8   CREATE TABLE Student (
9   sid INT PRIMARY KEY,
10  sname VARCHAR(50),
11  age INT
12  );
13
14  CREATE TABLE Matchh (
15  mid VARCHAR(10) PRIMARY KEY,
16  mname VARCHAR(50),
17  venue VARCHAR(50)
18  );
19  CREATE TABLE Play (
20  sid INT,
21  mid VARCHAR(10),
22  day DATE,
23  PRIMARY KEY (sid, mid),
24  FOREIGN KEY (sid) REFERENCES Student(sid),
25  FOREIGN KEY (mid) REFERENCES Matchh(mid)
26  );
27
```

```sql
-- Populating Tables:
INSERT INTO Student (sid, sname, age) VALUES
(1, 'Amit', 20),
(2, 'Ravi', 22),
(3, 'Suresh', 19),
(4, 'Priya', 21);
INSERT INTO Matchh (mid, mname, venue) VALUES
('B10', 'Football', 'Delhi'),
('B11', 'Cricket', 'Mumbai'),
('B12', 'Basketball', 'Delhi'),
('B13', 'Hockey', 'Chennai');
INSERT INTO Play (sid, mid, day) VALUES
(1, 'B10', '2024-09-01'),
(2, 'B11', '2024-09-01'),
(1, 'B12', '2024-09-02'),
(3, 'B10', '2024-09-03'),
(4, 'B11', '2024-09-04');

-- Find all information of students who have played matchh number B10.
SELECT s.*
FROM Student s
JOIN Play p ON s.sid = p.sid
WHERE p.mid = 'B10';
-- Find the name of matches played by Amit.
SELECT m.mname
FROM Student s
JOIN Play p ON s.sid = p.sid
```

```
55      JOIN Matchh m ON p.mid = m.mid
56      WHERE s.sname = 'Amit';
57      -- Find the names of students who have played a match in Delhi.
58 •    SELECT DISTINCT s.sname
59      FROM Student s
60      JOIN Play p ON s.sid = p.sid
61      JOIN Matchh m ON p.mid = m.mid
62      WHERE m.venue = 'Delhi';
63      -- Find the names of students who have played at least one match.
64 •    SELECT DISTINCT s.sname
65      FROM Student s
66      JOIN Play p ON s.sid = p.sid;
67      -- Find the ids and names of students who have played two different matches on the same day.
68 •    SELECT s.sid, s.sname
69      FROM Student s
70      JOIN Play p1 ON s.sid = p1.sid
71      JOIN Play p2 ON s.sid = p2.sid AND p1.mid != p2.mid AND
72     p1.day = p2.day;
73     -- Find the ids of students who have played a match in Delhi or Mumbai.
74 •    SELECT DISTINCT s.sid
75      FROM Student s
76      JOIN Play p ON s.sid = p.sid
77      JOIN Matchh m ON p.mid = m.mid
78      WHERE m.venue IN ('Delhi', 'Mumbai');
79      -- Find the average age of students.
80 •    SELECT AVG(age) AS average_age
81      FROM Student;
```

```
1 •    SELECT * FROM exp9.matchh;
```



Result Grid | Filter Rows:

	mid	mname	venue
▶	B10	Football	Delhi
	B11	Cricket	Mumbai
	B12	Basketball	Delhi
	B13	Hockey	Chennai
*	NULL	NULL	NULL

```
1 •  SELECT * FROM exp9.play;
```

Result Grid | Filter Rows:

	sid	mid	day
▶	1	B10	2024-09-01
	1	B12	2024-09-02
	2	B11	2024-09-01
	3	B10	2024-09-03
	4	B11	2024-09-04
*	NULL	NULL	NULL

```
1 •  SELECT * FROM exp9.student;
```

Result Grid | Filter Rows:

	sid	sname	age
▶	1	Amit	20
	2	Ravi	22
	3	Suresh	19
	4	Priya	21
*	NULL	NULL	NULL

Conclusion:

The SQL code effectively establishes a relational database to manage student and match data. Various queries demonstrate the ability to analyze student participation in matches based on specific criteria, such as match location and student demographics. The design supports efficient data retrieval and can be expanded for additional analyses, such as tracking performance or more detailed demographic studies.

Future enhancements could involve adding more attributes to students and matches, such as performance statistics or additional match details, to provide deeper insights into student engagement in sports.

Experiment 09

To understand the concepts of Views.

Aim:

To create and manage an employee database, including functionality for inserting, updating, and deleting employee records, as well as creating views for simplified data access.

Problem Statement:

The task involves setting up a database to track employee information, such as names, dates of birth, salaries, and department affiliations. The goal is to facilitate efficient data retrieval and manipulation, while also creating views that allow for more straightforward access to specific employee data.

Theory:

Relational databases organize data in structured tables, enabling relationships and efficient querying. SQL (Structured Query Language) is used for defining and manipulating this data. Key concepts include:

- Tables: Used to store data in rows and columns.
- Primary Keys: Unique identifiers for records in a table.
- Views: Virtual tables created from SQL queries that simplify access to data.
- Data Manipulation: Using `INSERT`, `UPDATE`, and `DELETE` commands to modify data in the database.

Commands Used:

1. Database and Table Creation:
2. Data Insertion:
3. Creating Views:

4. Modifying Table Structure:

5. Updating Records via a View:

6. Deleting Records via a View:

7. Creating Another View:

8. Viewing the Data:

9. Dropping a View:

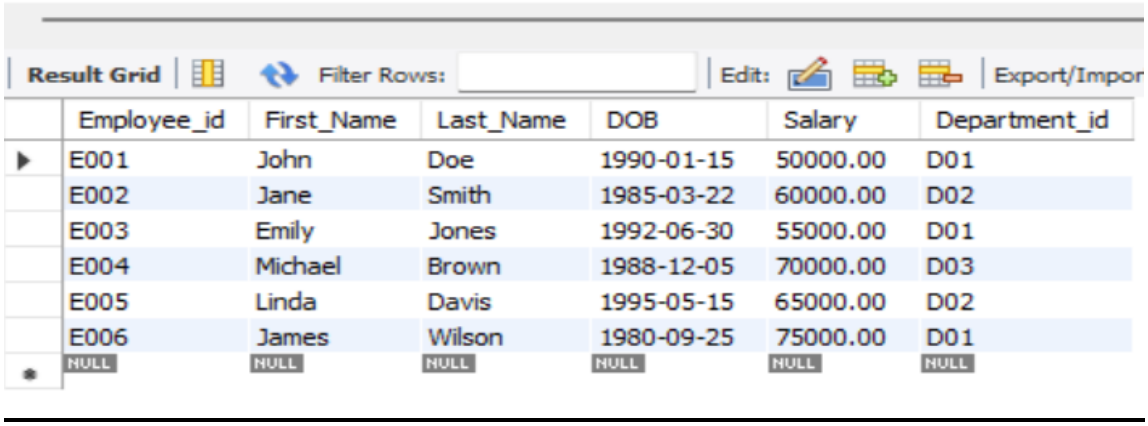
Results:

```sql

CREATE DATABASE exp10;
USE exp10;

-- Creating the EMPLOYEES Table
CREATE TABLE EMPLOYEES (
 Employee_id CHAR(10) PRIMARY KEY,
 First_Name CHAR(30) NOT NULL,
 Last_Name CHAR(30) NOT NULL,
 DOB DATE,
 Salary DECIMAL(10, 2) NOT NULL,  -- Using DECIMAL to handle salaries with two decimal places
 Department_id CHAR(10)
);

-- Inserting values into the EMPLOYEES table
INSERT INTO EMPLOYEES (Employee_id, First_Name, Last_Name, DOB, Salary, Department_id) VALUES
('E001', 'John', 'Doe', '1990-01-15', 50000.00, 'D01'),
('E002', 'Jane', 'Smith', '1985-03-22', 60000.00, 'D02'),
('E003', 'Emily', 'Jones', '1992-06-30', 55000.00, 'D01'),
('E004', 'Michael', 'Brown', '1988-12-05', 70000.00, 'D03'),
('E005', 'Linda', 'Davis', '1995-05-15', 65000.00, 'D02'),
('E006', 'James', 'Wilson', '1980-09-25', 75000.00, 'D01');

-- Creating a View named emp_view
CREATE VIEW emp_view AS
SELECT Employee_id, Last_Name, Salary, Department_id
```

```sql
FROM EMPLOYEES;

-- You cannot directly insert into a view like this unless you are inserting into an updatable view that maps directly to a base table.
-- Remove the insert into view since it will cause errors.

-- If you need to modify the Salary column to allow NULL values, you'd do the following:
ALTER TABLE EMPLOYEES MODIFY Salary DECIMAL(10, 2) NULL;

-- Now, you can insert a row with a NULL salary
INSERT INTO EMPLOYEES (Employee_id, First_Name, Last_Name, DOB, Salary, Department_id) VALUES
('E007', 'Chris', 'Anderson', NULL, NULL, 'D01');

-- Update operations on the View (affects the base table EMPLOYEES)
UPDATE emp_view
SET Salary = 80000.00
WHERE Employee_id = 'E001';

-- Delete an employee from the view (and consequently from the EMPLOYEES table)
DELETE FROM emp_view
WHERE Employee_id = 'E003';

SELECT * FROM emp_view;
-- Dropping the emp_view
DROP VIEW emp_view;

-- Create a View named salary_view to show annual salary for employees in Department D02
CREATE VIEW salary_view AS
SELECT Employee_id, Last_Name, Salary * 12 AS Annual_Salary
```

```sql
SELECT Employee_id, Last_Name, Salary * 12 AS Annual_Salary
FROM EMPLOYEES
WHERE Department_id = 'D02';

-- View the salary_view
SELECT * FROM salary_view;
```

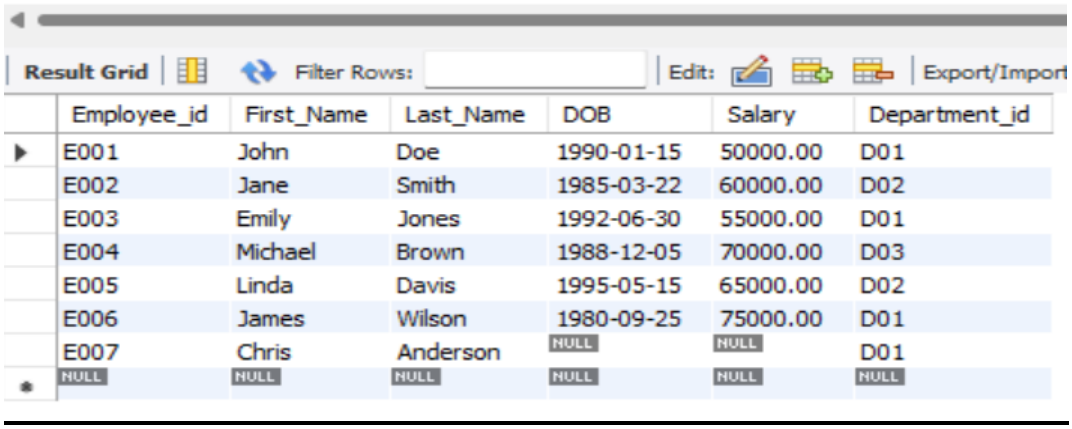
```
1 •  SELECT * FROM exp10.employees;
```



Result Grid | Filter Rows: | Edit: | Export/Impor

Employee_id	First_Name	Last_Name	DOB	Salary	Department_id
E001	John	Doe	1990-01-15	50000.00	D01
E002	Jane	Smith	1985-03-22	60000.00	D02
E003	Emily	Jones	1992-06-30	55000.00	D01
E004	Michael	Brown	1988-12-05	70000.00	D03
E005	Linda	Davis	1995-05-15	65000.00	D02
E006	James	Wilson	1980-09-25	75000.00	D01
NULL	NULL	NULL	NULL	NULL	NULL

```
1 •  SELECT * FROM exp10.employees;
```



Result Grid | Filter Rows: | Edit: | Export/Import

Employee_id	First_Name	Last_Name	DOB	Salary	Department_id
E001	John	Doe	1990-01-15	50000.00	D01
E002	Jane	Smith	1985-03-22	60000.00	D02
E003	Emily	Jones	1992-06-30	55000.00	D01
E004	Michael	Brown	1988-12-05	70000.00	D03
E005	Linda	Davis	1995-05-15	65000.00	D02
E006	James	Wilson	1980-09-25	75000.00	D01
E007	Chris	Anderson	NULL	NULL	D01
NULL	NULL	NULL	NULL	NULL	NULL

```
1 •  SELECT * FROM exp10.emp_view;
```

Result Grid | Filter Rows: | Expor

Employee_id	Last_Name	Salary	Department_id
E001	Doe	80000.00	D01
E002	Smith	60000.00	D02
E004	Brown	70000.00	D03
E005	Davis	65000.00	D02
E006	Wilson	75000.00	D01
E007	Anderson	NULL	D01

```
1 •  SELECT * FROM exp10.salary_view;
```

Result Grid | Filter Rows: | Expo

Employee_id	Last_Name	Annual_Salary
E002	Smith	720000.00
E005	Davis	780000.00

Conclusion:

The SQL code effectively sets up a relational database to manage employee information, including functionalities for inserting, updating, and deleting records. The creation of views enhances data accessibility by allowing users to easily retrieve specific employee data without dealing with the underlying table directly.

The use of a view for annual salaries provides a clear example of how to present calculated data, facilitating reporting and analysis. Overall, this database design

supports efficient employee management and can be expanded further with additional features such as more complex views or stored procedures for automated reporting.

Future enhancements could include adding indexes for faster querying, more detailed employee attributes, or implementing stored procedures for common operations to streamline data management tasks.