DBMS – CIA 3

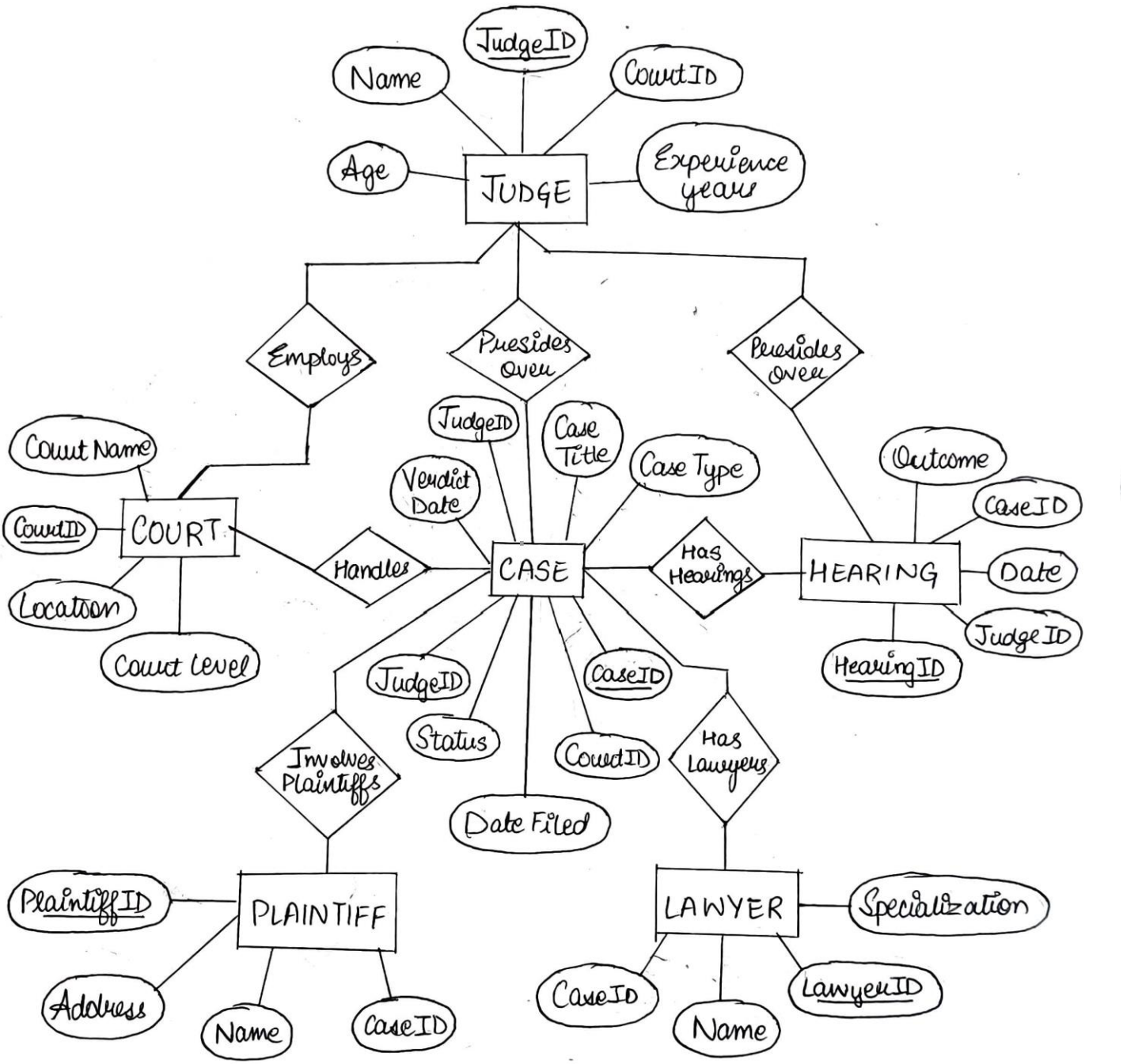
(Mishika Mittal, 3BScDM, 2341318)

Introduction to the Indian Judicial System

The Indian Judicial System is the backbone and the name of implementing the Constitution and laws of India. It consists of numerous courts which handle civil, criminal, and constitutional cases. In a judicial system, it works hierarchically. The high court is situated at the apex followed by the state-level High Court and then the district-level District Courts.

This DBMS project models the core entities of the judicial system, namely Courts, Judges, Cases, Lawyers, Plaintiffs, and Hearings, using an ER diagram along with an SQL-based implementation. The objective of this project is to design a database that represents the workings of the Indian Judicial System.

ER Diagram



Explanation of Concepts Used with Examples and Output

1. DDL Commands (Lab 1)

Data Definition Language (DDL) consists of SQL commands that define the structure of the database schema. The most commonly used DDL commands include CREATE, ALTER, and DROP. These commands are used to create new database objects such as tables, modify existing objects, or delete objects from the database.

i. **Create Table**: The CREATE TABLE statement is used to define a new table and its columns. Each column is defined with a data type and optional constraints.

Example:

```
mysql> create table court(
    -> CourtID INT AUTO_INCREMENT PRIMARY KEY,
    -> CourtName VARCHAR(100) NOT NULL,
    -> CourtLevel VARCHAR(50) CHECK (CourtLevel IN ('Supreme', 'High', 'Dist
rict')),
    -> Location VARCHAR(100),
    -> UNIQUE(CourtName, Location));
Query OK, 0 rows affected (0.07 sec)
```

Output:

```
+-------------+--------------+------+-----+---------+----------------+
| Field       | Type         | Null | Key | Default | Extra          |
+-------------+--------------+------+-----+---------+----------------+
| CourtID     | int          | NO   | PRI | NULL    | auto_increment |
| CourtName   | varchar(100) | NO   | MUL | NULL    |                |
| CourtLevel  | varchar(50)  | YES  |     | NULL    |                |
| Location    | varchar(100) | YES  |     | NULL    |                |
+-------------+--------------+------+-----+---------+----------------+
4 rows in set (0.02 sec)
```

ii. **Alter Table**: The ALTER TABLE statement is used to modify an existing table. You can use this command to add new columns, modify existing columns, or drop columns.

Example:

```
mysql> ALTER TABLE Cases ADD COLUMN VerdictDate DATE;
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Output:

```
mysql> select * from Cases;
+--------+----------------------+----------+------------+---------+---------+---------+-------------+
| CaseID | CaseTitle            | CaseType | DateFiled  | Status  | CourtID | JudgeID | VerdictDate |
+--------+----------------------+----------+------------+---------+---------+---------+-------------+
|      1 | State vs John Doe    | Criminal | 2024-01-15 | Active  |       2 |       1 | NULL        |
|      2 | Smith vs Johnson     | Civil    | 2024-02-10 | Pending |       3 |       2 | NULL        |
|      3 | People vs Richard Roe | Criminal | 2023-12-22 | Closed  |       4 |       3 | NULL        |
+--------+----------------------+----------+------------+---------+---------+---------+-------------+
3 rows in set (0.00 sec)
```

iii. **Drop Table**: The DROP TABLE statement is used to completely remove a table from the database. When a table is dropped, all the data and the structure of the table are permanently deleted.

Example:

```
mysql> DROP TABLE IF EXISTS Lawyer;
Query OK, 0 rows affected (0.03 sec)
```

2. DML Commands (Lab 2)

Data Manipulation Language (DML) allows us to manage the data inside the tables. The key DML commands are INSERT (Adds new records to the table), UPDATE (Modifies existing records in the table), DELETE (Removes records from the table).

i. **Insert**: The INSERT INTO statement is used to add new data to a table. It requires specifying the table name, the columns to be populated, and the corresponding values for each column.

Example:

```
mysql> INSERT INTO Court (CourtName, CourtLevel, Location)
    -> VALUES
    -> ('Supreme Court', 'Supreme', 'New Delhi'),
    -> ('High Court Delhi', 'High', 'Delhi'),
    -> ('High Court Bombay', 'High', 'Mumbai'),
    -> ('District Court Mumbai', 'District', 'Mumbai'),
    -> ('District Court Bangalore', 'District', 'Bangalore');
Query OK, 5 rows affected (0.01 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

Output:

```
mysql> SELECT
    -> * FROM Court;
+---------+-------------------------+------------+-----------+
| CourtID | CourtName               | CourtLevel | Location  |
+---------+-------------------------+------------+-----------+
|       1 | Supreme Court           | Supreme    | New Delhi |
|       2 | High Court Delhi        | High       | Delhi     |
|       3 | High Court Bombay       | High       | Mumbai    |
|       4 | District Court Mumbai   | District   | Mumbai    |
|       5 | District Court Bangalore | District   | Bangalore |
+---------+-------------------------+------------+-----------+
5 rows in set (0.00 sec)
```

ii. **Update**: The UPDATE statement modifies the data in an existing row. You need to specify which table to update, what changes to make, and which records should be updated using a WHERE clause.

Example:

```
mysql> UPDATE Judge SET ExperienceYears = 35 WHERE JudgeID = 1;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

Output:

```
mysql> select * from Judge;
+---------+-----------------+------+---------+-----------------+
| JudgeID | Name            | Age  | CourtID | ExperienceYears |
+---------+-----------------+------+---------+-----------------+
|       1 | Justice Sharma  |   55 |       1 |              35 |
|       2 | Justice Verma   |   50 |       2 |              25 |
|       3 | Justice Rao     |   47 |       3 |              22 |
|       4 | Justice Bhaskar |   52 |       4 |              27 |
|       5 | Justice Patel   |   49 |       5 |              20 |
+---------+-----------------+------+---------+-----------------+
5 rows in set (0.00 sec)
```

iii. **Delete**: The DELETE statement removes records from a table. The WHERE clause is used to specify which records should be deleted. Without a WHERE clause, all records in the table would be deleted.

Example:

```
mysql> DELETE FROM Judge WHERE JudgeID = 5;
Query OK, 1 row affected (0.01 sec)
```

Output:

```
mysql> select * from Judge;
+---------+-----------------+------+---------+-----------------+
| JudgeID | Name            | Age  | CourtID | ExperienceYears |
+---------+-----------------+------+---------+-----------------+
|       1 | Justice Sharma  |   55 |       1 |              35 |
|       2 | Justice Verma   |   50 |       2 |              25 |
|       3 | Justice Rao     |   47 |       3 |              22 |
|       4 | Justice Bhaskar |   52 |       4 |              27 |
+---------+-----------------+------+---------+-----------------+
4 rows in set (0.00 sec)
```

3. Constraints (Lab 3)

Constraints are rules enforced on columns in a table to ensure the accuracy and reliability of the data. Common constraints include:

i. **Primary Key**: A Primary Key is a unique identifier for each record in a table. Each table should have one and only one primary key, and it ensures that no duplicate records exist for that column or set of columns.

ii. **Foreign Key**: A Foreign Key links one table to another, enforcing referential integrity. It ensures that the value in the foreign key column must match an existing value in the primary key column of the referenced table.

iii. **Not Null**: The NOT NULL constraint ensures that a column cannot contain NULL values. It is used when we want to ensure that a column always contains a value.

iv. **Check**: The CHECK constraint ensures that all values in a column meet a specific condition. It is used to enforce domain integrity.

v. **Unique**: The UNIQUE constraint ensures that all values in a column or group of columns are unique across all records in the table.

vi. **Auto_Increment**: The AUTO_INCREMENT constraint is used to automatically generate unique values for a column when a new record is inserted. It is commonly used for primary keys.

vii. Default: The DEFAULT constraint provides a default value for a column if no value is provided when a new record is inserted.

Example:

```
mysql> CREATE TABLE Judge (
    -> JudgeID INT AUTO_INCREMENT PRIMARY KEY,
    -> Name VARCHAR(100) NOT NULL,
    -> Age INT CHECK (Age > 25),
    -> CourtID INT,
    -> ExperienceYears INT DEFAULT 5,
    -> FOREIGN KEY (CourtID) REFERENCES Court(CourtID));
Query OK, 0 rows affected (0.05 sec)
```

4. SQL Clauses and Aggregate Functions (Lab 4)

SQL clauses and aggregate functions allow us to filter, group, and perform calculations on data in a database.

i. **WHERE**: The WHERE clause is used to filter records based on a condition. Only the rows that meet the specified condition are returned.

Example:

```
mysql> SELECT * FROM Cases WHERE Status = 'Active';
+--------+------------------+----------+------------+--------+---------+---------+-------------+
| CaseID | CaseTitle        | CaseType | DateFiled  | Status | CourtID | JudgeID | VerdictDate |
+--------+------------------+----------+------------+--------+---------+---------+-------------+
|      1 | State vs John Doe | Criminal | 2024-01-15 | Active |       2 |       1 | NULL        |
+--------+------------------+----------+------------+--------+---------+---------+-------------+
1 row in set (0.00 sec)
```

ii. **GROUP BY**: The GROUP BY clause groups rows that have the same values in specified columns. It is often used with aggregate functions to perform operations on each group of data.

iii. **HAVING**: The HAVING clause is used to filter groups created by the GROUP BY clause. It is similar to the WHERE clause but operates on groups rather than individual rows.

Example:

```
mysql> SELECT CourtID, COUNT(*) AS CaseCount
    -> FROM Cases
    -> GROUP BY CourtID
    -> HAVING COUNT(*) > 1;
Empty set (0.00 sec)
```

iv. **ORDER BY**: The ORDER BY clause sorts the result set by one or more columns. The default sorting order is ascending (ASC), but it can also be set to descending (DESC).

Example:

```
mysql> SELECT * FROM Judge ORDER BY Age DESC;
+---------+----------------+------+---------+-----------------+
| JudgeID | Name           | Age  | CourtID | ExperienceYears |
+---------+----------------+------+---------+-----------------+
|       1 | Justice Sharma |   55 |       1 |              35 |
|       4 | Justice Bhaskar|   52 |       4 |              27 |
|       2 | Justice Verma  |   50 |       2 |              25 |
|       3 | Justice Rao    |   47 |       3 |              22 |
+---------+----------------+------+---------+-----------------+
4 rows in set (0.00 sec)
```

v. **DISTINCT**: The DISTINCT keyword is used to return only unique values from a column, eliminating duplicates.

Example:

```
mysql> SELECT DISTINCT CourtLevel FROM Court;
+------------+
| CourtLevel |
+------------+
| Supreme    |
| High       |
| District   |
+------------+
3 rows in set (0.00 sec)
```

vi. **LIMIT**: The LIMIT keyword limits the number of rows returned by a query.

Example:

```
mysql> SELECT * FROM Cases LIMIT 2;
+--------+------------------+----------+------------+---------+---------+---------+-------------+
| CaseID | CaseTitle        | CaseType | DateFiled  | Status  | CourtID | JudgeID | VerdictDate |
+--------+------------------+----------+------------+---------+---------+---------+-------------+
|      1 | State vs John Doe | Criminal | 2024-01-15 | Active  |       2 |       1 | NULL        |
|      2 | Smith vs Johnson  | Civil    | 2024-02-10 | Pending |       3 |       2 | NULL        |
+--------+------------------+----------+------------+---------+---------+---------+-------------+
2 rows in set (0.00 sec)
```

Aggregate functions perform calculations on a set of values and return a single value. Common aggregate functions include:

- **COUNT():** Returns the number of rows.
- **SUM():** Returns the sum of values.
- **AVG():** Returns the average of values.
- **MIN():** Returns the minimum value.
- **MAX():** Returns the maximum value.

Examples:

```
mysql> SELECT COUNT(*) AS TotalCases FROM Cases;
+------------+
| TotalCases |
+------------+
|          3 |
+------------+
1 row in set (0.01 sec)

mysql> SELECT AVG(Age) FROM Judge;
+----------+
| AVG(Age) |
+----------+
|  51.0000 |
+----------+
1 row in set (0.00 sec)
```

Pattern Liking: The LIKE operator is used for pattern matching in SQL. It is often combined with wildcards (% for multiple characters, _ for a single character).

Example:

```
mysql> SELECT * FROM Cases WHERE CaseTitle LIKE 'State%';
+--------+------------------+----------+------------+--------+----------+---------+-------------+
| CaseID | CaseTitle        | CaseType | DateFiled  | Status | CourtID | JudgeID | VerdictDate |
+--------+------------------+----------+------------+--------+----------+---------+-------------+
|      1 | State vs John Doe | Criminal | 2024-01-15 | Active |       2 |       1 | NULL        |
+--------+------------------+----------+------------+--------+----------+---------+-------------+
1 row in set (0.00 sec)
```

5. Joins (Lab 5)

A join allows us to retrieve data from two or more tables based on a condition that links them. The types of joins:

i. **Inner Join**: An Inner Join returns only the rows that have matching values in both tables. Rows from either table that do not match are not included in the result.

Example:

```
mysql> SELECT Cases.CaseTitle, Judge.Name
    -> FROM Cases
    -> INNER JOIN Judge ON Cases.JudgeID = Judge.JudgeID;
+----------------------+----------------+
| CaseTitle            | Name           |
+----------------------+----------------+
| State vs John Doe    | Justice Sharma |
| Smith vs Johnson     | Justice Verma  |
| People vs Richard Roe | Justice Rao    |
+----------------------+----------------+
3 rows in set (0.00 sec)
```

ii. **Left Join**: A Left Join returns all records from the left table (the table mentioned first) and the matched records from the right table. If there is no match, NULL values are returned for columns from the right table.

Example:

```
mysql> SELECT Cases.CaseTitle, Judge.Name
    -> FROM Cases
    -> LEFT JOIN Judge ON Cases.JudgeID = Judge.JudgeID;
+----------------------+----------------+
| CaseTitle            | Name           |
+----------------------+----------------+
| State vs John Doe    | Justice Sharma |
| Smith vs Johnson     | Justice Verma  |
| People vs Richard Roe | Justice Rao    |
+----------------------+----------------+
3 rows in set (0.00 sec)
```

iii. **Right Join**: A Right Join returns all records from the right table (the second table mentioned in the query) and the matched records from the left table. If there is no match, NULL values are returned for columns from the left table.

Example:

```
mysql> SELECT Judge.Name, Cases.CaseTitle
    -> FROM Judge
    -> RIGHT JOIN Cases ON Judge.JudgeID = Cases.JudgeID;
+----------------+----------------------+
| Name           | CaseTitle            |
+----------------+----------------------+
| Justice Sharma | State vs John Doe    |
| Justice Verma  | Smith vs Johnson     |
| Justice Rao    | People vs Richard Roe |
+----------------+----------------------+
3 rows in set (0.00 sec)
```

iv. **Full Outer Join**: A Full Outer Join returns all records when there is a match in either the left or right table. If there is no match, NULL values are returned from the table that doesn't have a match.

Example:

```
mysql> SELECT Cases.CaseTitle, Judge.Name
    -> FROM Cases
    -> LEFT JOIN Judge ON Cases.JudgeID = Judge.JudgeID
    -> UNION
    -> SELECT Cases.CaseTitle, Judge.Name
    -> FROM Cases
    -> RIGHT JOIN Judge ON Cases.JudgeID = Judge.JudgeID;
+----------------------+----------------+
| CaseTitle            | Name           |
+----------------------+----------------+
| State vs John Doe    | Justice Sharma |
| Smith vs Johnson     | Justice Verma  |
| People vs Richard Roe | Justice Rao    |
| NULL                 | Justice Bhaskar |
+----------------------+----------------+
4 rows in set (0.00 sec)
```

v. **Joining Multiple Tables**: Combining more than two tables in a single query.

Example:

```
mysql> SELECT Cases.CaseTitle, Judge.Name, Court.CourtName
    -> FROM Cases
    -> JOIN Judge ON Cases.JudgeID = Judge.JudgeID
    -> JOIN Court ON Judge.CourtID = Court.CourtID;
+----------------------+----------------+-------------------+
| CaseTitle            | Name           | CourtName         |
+----------------------+----------------+-------------------+
| State vs John Doe    | Justice Sharma | Supreme Court     |
| Smith vs Johnson     | Justice Verma  | High Court Delhi  |
| People vs Richard Roe | Justice Rao   | High Court Bombay |
+----------------------+----------------+-------------------+
3 rows in set (0.00 sec)
```

6. Functions (Lab 6)

Three categories of functions:

- String Functions: Manipulate and handle string data.
- Numeric Functions: Perform calculations on numeric data.
- Date-Time Functions: Manipulate and retrieve date and time information.

i. **String Functions**: String functions allow us to manipulate text-based data. Common string functions include:

- **UPPER():** Converts a string to uppercase.
- **LOWER():** Converts a string to lowercase.
- **SUBSTRING():** Extracts a portion of a string.
- **CONCAT():** Concatenates two or more strings together.
- **LENGTH():** Returns the length of a string.

Example**:**

```
mysql> SELECT UPPER(Name) AS UpperName FROM Judge;
+-----------------+
| UpperName       |
+-----------------+
| JUSTICE SHARMA  |
| JUSTICE VERMA   |
| JUSTICE RAO     |
| JUSTICE BHASKAR |
+-----------------+
4 rows in set (0.00 sec)

mysql> SELECT CONCAT(CaseTitle, ' - ', Status) AS CaseDetails FROM Cases;
+--------------------------------+
| CaseDetails                    |
+--------------------------------+
| State vs John Doe - Active     |
| Smith vs Johnson - Pending     |
| People vs Richard Roe - Closed |
+--------------------------------+
3 rows in set (0.00 sec)

mysql> SELECT SUBSTRING(CourtName, 1, 5) AS ShortCourtName FROM Court;
+---------------+
| ShortCourtName |
+---------------+
| Distr          |
| Distr          |
| High           |
| High           |
| Supre          |
+---------------+
5 rows in set (0.00 sec)
```

ii. **Numeric Functions**: Numeric functions are used to perform mathematical operations on numeric data. Common numeric functions include:

- **ROUND():** Rounds a number to the specified number of decimal places.
- **CEIL():** Returns the smallest integer greater than or equal to a number.
- **FLOOR():** Returns the largest integer less than or equal to a number.
- **ABS():** Returns the absolute value of a number.
- **MOD():** Returns the remainder of a division operation.

Example:

```
mysql> SELECT ROUND(AVG(ExperienceYears), 2) FROM Judge;
+--------------------------------+
| ROUND(AVG(ExperienceYears), 2) |
+--------------------------------+
|                          27.25 |
+--------------------------------+
1 row in set (0.00 sec)

mysql> SELECT CEIL(ExperienceYears) AS CeilingExperience, FLOOR(ExperienceYears) AS
FloorExperience FROM Judge;
+-------------------+-----------------+
| CeilingExperience | FloorExperience |
+-------------------+-----------------+
|                35 |              35 |
|                25 |              25 |
|                22 |              22 |
|                27 |              27 |
+-------------------+-----------------+
4 rows in set (0.00 sec)

mysql> SELECT CaseTitle, YEAR(DateFiled) AS FilingYear, MONTH(DateFiled) AS FilingMo
nth FROM Cases;
+----------------------+------------+-------------+
| CaseTitle            | FilingYear | FilingMonth |
+----------------------+------------+-------------+
| State vs John Doe    |       2024 |           1 |
| Smith vs Johnson     |       2024 |           2 |
| People vs Richard Roe |       2023 |          12 |
+----------------------+------------+-------------+
```

iii. **Date-Time Function**: Date-time functions are used to manipulate date and time values. Common date-time functions include:

- **NOW():** Returns the current date and time.
- **CURDATE():** Returns the current date.
- **YEAR():** Extracts the year from a date.
- **MONTH():** Extracts the month from a date.
- **DAY():** Extracts the day from a date.
- **DATEDIFF():** Returns the difference between two dates.

Example:

```
mysql> SELECT CURDATE() AS CurrentDate;
+-------------+
| CurrentDate |
+-------------+
| 2024-10-17  |
+-------------+
1 row in set (0.00 sec)

mysql> SELECT DATEDIFF(NOW(), DateFiled) AS DaysSinceFiled FROM Cases;
+----------------+
| DaysSinceFiled |
+----------------+
|            276 |
|            250 |
|            300 |
+----------------+
3 rows in set (0.00 sec)
```

7. Nested Queries (Lab 7)

A nested query (also known as a subquery) is a query within another query. The results of the inner query (the nested query) are used by the outer query to retrieve or filter data.

i. **Self-Query**: A self-query is a query that refers to the same table more than once within the same query, often used to find hierarchical or recursive relationships within the data.

Example:

```
mysql> SELECT Name FROM Judge WHERE JudgeID IN (SELECT JudgeID F
ROM Cases WHERE Status = 'Active');
+----------------+
| Name           |
+----------------+
| Justice Sharma |
+----------------+
1 row in set (0.00 sec)
```

ii. **Subquery with Multiple Tables**: Subqueries can involve multiple tables to perform complex queries and retrieve data across related tables.

Example:

```
mysql> SELECT Name FROM Judge WHERE Age > ALL (SELECT Age FROM J
udge WHERE CourtID = 1);
Empty set (0.00 sec)
```

iii. **Subquery with Relational Operators**: Subqueries can be used with relational operators like >, <, =, IN, NOT IN, and more. These operators allow us to compare values between the subquery results and the outer query.

Example:

```
mysql> SELECT Name FROM Judge WHERE JudgeID IN (SELECT JudgeID F
ROM Cases WH
ERE DateFiled < '2024-01-01');
+-------------+
| Name        |
+-------------+
| Justice Rao |
+-------------+
1 row in set (0.00 sec)
```

iv. **Subquery with Conditional Operators**: Conditional operators like ANY and ALL allow comparisons to be made between a value and a set of results.

- **ANY**: Returns true if any one value in the subquery satisfies the condition.
- **ALL**: Returns true only if all values in the subquery satisfy the condition.

Example:

```
mysql> SELECT Name FROM Judge WHERE Age > ALL (SELECT Age FROM J
udge WHERE CourtID = 1);
Empty set (0.00 sec)
```

iv. **Set Operations Without Minus Operator**: The MINUS operator is often used to find the difference between two sets. In databases that don't support MINUS, we can achieve the same result using a combination of LEFT JOIN and WHERE.

Example:

```
mysql> SELECT Judge.Name FROM Judge
    -> LEFT JOIN Cases ON Judge.JudgeID = Cases.JudgeID
    -> WHERE Cases.JudgeID IS NULL;
+-----------------+
| Name            |
+-----------------+
| Justice Bhaskar |
+-----------------+
1 row in set (0.00 sec)
```

8. Relational Algebra Operations (Lab 8)

Relational algebra operations are mathematical operations that help in manipulating and retrieving data from relational databases. In SQL, these operations are translated into SQL queries.

i. **Cartesian Product**: The Cartesian Product (also known as the cross join) returns all possible combinations of rows from two tables. The result set contains every row from the first table paired with every row from the second table.

Example:

```
mysql> SELECT Judge.Name, Court.CourtName FROM Judge, Court;
+-----------------+--------------------------+
| Name            | CourtName                |
+-----------------+--------------------------+
| Justice Bhaskar | District Court Bangalore |
| Justice Rao     | District Court Bangalore |
| Justice Verma   | District Court Bangalore |
| Justice Sharma  | District Court Bangalore |
| Justice Bhaskar | District Court Mumbai    |
| Justice Rao     | District Court Mumbai    |
| Justice Verma   | District Court Mumbai    |
| Justice Sharma  | District Court Mumbai    |
| Justice Bhaskar | High Court Bombay        |
| Justice Rao     | High Court Bombay        |
| Justice Verma   | High Court Bombay        |
| Justice Sharma  | High Court Bombay        |
| Justice Bhaskar | High Court Delhi         |
| Justice Rao     | High Court Delhi         |
| Justice Verma   | High Court Delhi         |
| Justice Sharma  | High Court Delhi         |
| Justice Bhaskar | Supreme Court            |
| Justice Rao     | Supreme Court            |
| Justice Verma   | Supreme Court            |
| Justice Sharma  | Supreme Court            |
+-----------------+--------------------------+
20 rows in set (0.00 sec)
```

ii. **Division**: Division is a more complex relational algebra operation. It is used to answer queries that involve the "all" condition, such as finding entities that are related to all entities in another set. SQL does not support the division operation directly, but it can be emulated using a combination of JOIN, GROUP BY, and HAVING.

Example:

```
mysql> SELECT Judge.Name
    -> FROM Judge
    -> JOIN Cases ON Judge.JudgeID = Cases.JudgeID
    -> WHERE CaseType IN ('Criminal', 'Civil')
    -> GROUP BY Judge.Name
    -> HAVING COUNT(DISTINCT CaseType) = 2;
Empty set (0.00 sec)
```

iii. **Rename**: The Rename operation is used to give a column or table a temporary name in the result set. This is particularly useful for making query results more readable.

Example:

```
mysql> SELECT Judge.Name AS JudgeName, Court.CourtName AS Assign
edCourt
    -> FROM Judge
    -> JOIN Court ON Judge.CourtID = Court.CourtID;
+-----------------+----------------------+
| JudgeName       | AssignedCourt        |
+-----------------+----------------------+
| Justice Sharma  | Supreme Court        |
| Justice Verma   | High Court Delhi     |
| Justice Rao     | High Court Bombay    |
| Justice Bhaskar | District Court Mumbai |
+-----------------+----------------------+
4 rows in set (0.00 sec)
```

9. TCL Commands (Lab 9)

Transaction Control Language (TCL) commands are used to manage changes made by DML operations (like INSERT, UPDATE, and DELETE). These commands allow us to group SQL statements into transactions that are either saved permanently (committed) or undone (rolled back) in case of an error.

i. **START TRANSACTION/BEGIN**: The START TRANSACTION or BEGIN command begins a new transaction. Once a transaction is started, SQL statements are executed as part of the transaction, but changes are not saved to the database until a COMMIT is issued.

ii. **COMMIT**: The COMMIT command permanently saves all the changes made in the current transaction. Once a COMMIT is issued, all operations within the transaction become permanent and cannot be rolled back.

iii. **ROLLBACK**: The ROLLBACK command undoes any changes made in the current transaction. If a transaction encounters an error or if you want to undo the changes for any reason, you can issue a ROLLBACK to revert to the state before the transaction began.

iv. **SAVEPOINT**: A SAVEPOINT creates a point within a transaction to which you can roll back without affecting the entire transaction. This allows more granular control within a transaction.

v. **Auto Commit (ON/OFF):** In SQL, by default, every DML statement (like INSERT, UPDATE, or DELETE) is committed automatically. You can control this behavior by enabling or disabling auto-commit mode.

- Auto Commit ON: Every SQL statement is automatically committed after execution.
- Auto Commit OFF: You need to explicitly commit a transaction using COMMIT

Example:

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> UPDATE Cases SET Status = 'Closed' WHERE CaseID = 1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SAVEPOINT BeforeExperienceUpdate;
Query OK, 0 rows affected (0.00 sec)

mysql> UPDATE Judge SET ExperienceYears = 40 WHERE JudgeID = 1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> ROLLBACK TO BeforeExperienceUpdate;
Query OK, 0 rows affected (0.00 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> SET autocommit = 0;
Query OK, 0 rows affected (0.00 sec)

mysql> BEGIN;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO Judge (Name, Age, CourtID, ExperienceYears)
    -> VALUES ('Justice Desai', 48, 2, 20);
Query OK, 1 row affected (0.00 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.01 sec)

mysql> SET autocommit = 1;
Query OK, 0 rows affected (0.00 sec)
```

10. VDL Commands (Lab 10)

View Definition Language (VDL) commands are used to create, update, and manage views in SQL. A view is a virtual table that consists of data retrieved by a query. Views do not store data themselves but display data from the underlying base tables.

i. Create a View with All Fields.

```
mysql> CREATE VIEW AllCases AS  SELECT * FROM Cases;
Query OK, 0 rows affected (0.01 sec)
```

ii. Create a View with Selected Fields.

```
mysql> CREATE VIEW SeniorJudges AS
    -> SELECT Name, Age  FROM Judge
    -> WHERE Age > 50;
Query OK, 0 rows affected (0.01 sec)
```

iii. Create a View Using Nested Queries.

```
mysql> CREATE VIEW DelhiCases AS
    -> SELECT CaseTitle, CourtID
    -> FROM Cases
    -> WHERE CourtID IN (SELECT CourtID FROM Court WHERE Locatio
n = 'Delhi');
Query OK, 0 rows affected (0.01 sec)
```

iv. Create a View Using Joins.

```
mysql> CREATE VIEW JudgeCases AS
    -> SELECT Judge.Name AS JudgeName, Cases.CaseTitle
    -> FROM Judge
    -> JOIN Cases ON Judge.JudgeID = Cases.JudgeID;
Query OK, 0 rows affected (0.01 sec)
```

v. Update a View.

```
mysql> UPDATE SeniorJudges
    -> SET Age = 60
    -> WHERE Name = 'Justice Verma';
Query OK, 0 rows affected (0.01 sec)
Rows matched: 0  Changed: 0  Warnings: 0
```

vi. Insert Records into a View.

```
mysql> INSERT INTO SeniorJudges (Name, Age) VALUES ('Justice Des
ai', 55);
Query OK, 1 row affected (0.00 sec)
```

vii. Update the View Using Conditional Operators (IF-ELSE-LIKE Conditions).

```
mysql> UPDATE SeniorJudges
    -> SET Age = CASE
    -> WHEN Age < 60 THEN Age + 1
    -> ELSE Age
    -> END;
Query OK, 3 rows affected (0.00 sec)
Rows matched: 3  Changed: 3  Warnings: 0
```

Full SQL Code and Output:

```
mysql> create database IndianJudicialSystem;
Query OK, 1 row affected (0.01 sec)

mysql> use IndianJudicialSystem;
Database changed
mysql> create table court(
    -> CourtID INT AUTO_INCREMENT PRIMARY KEY,
    -> CourtName VARCHAR(100) NOT NULL,
    -> CourtLevel VARCHAR(50) CHECK (CourtLevel IN ('Supreme', 'High',
'District')),
    -> Location VARCHAR(100),
    -> UNIQUE(CourtName, Location));
Query OK, 0 rows affected (0.07 sec)

mysql> DESC Court;
+------------+-------------+------+-----+---------+----------------+
| Field      | Type        | Null | Key | Default | Extra          |
+------------+-------------+------+-----+---------+----------------+
| CourtID    | int         | NO   | PRI | NULL    | auto_increment |
| CourtName  | varchar(100) | NO   | MUL | NULL    |                |
| CourtLevel | varchar(50)  | YES  |     | NULL    |                |
| Location   | varchar(100) | YES  |     | NULL    |                |
+------------+-------------+------+-----+---------+----------------+
4 rows in set (0.02 sec)

mysql> INSERT INTO Court (CourtName, CourtLevel, Location)
    -> VALUES
    -> ('Supreme Court', 'Supreme', 'New Delhi'),
    -> ('High Court Delhi', 'High', 'Delhi'),
    -> ('High Court Bombay', 'High', 'Mumbai'),
    -> ('District Court Mumbai', 'District', 'Mumbai'),
    -> ('District Court Bangalore', 'District', 'Bangalore');
Query OK, 5 rows affected (0.01 sec)
```

```
Records: 5  Duplicates: 0  Warnings: 0

mysql> SELECT
    -> * FROM Court;
+---------+-------------------------+------------+-----------+
| CourtID | CourtName               | CourtLevel | Location  |
+---------+-------------------------+------------+-----------+
|       1 | Supreme Court           | Supreme    | New Delhi |
|       2 | High Court Delhi        | High       | Delhi     |
|       3 | High Court Bombay       | High       | Mumbai    |
|       4 | District Court Mumbai   | District   | Mumbai    |
|       5 | District Court Bangalore | District   | Bangalore |
+---------+-------------------------+------------+-----------+
5 rows in set (0.00 sec)

mysql> CREATE TABLE Judge (
    -> JudgeID INT AUTO_INCREMENT PRIMARY KEY,
    -> Name VARCHAR(100) NOT NULL,
    -> Age INT CHECK (Age > 25),
    -> CourtID INT,
    -> ExperienceYears INT DEFAULT 5,
    -> FOREIGN KEY (CourtID) REFERENCES Court(CourtID));
Query OK, 0 rows affected (0.05 sec)

mysql> INSERT INTO Judge (Name, Age, CourtID, ExperienceYears)VALUES
    -> ('Justice Sharma', 55, 1, 30),
    -> ('Justice Verma', 50, 2, 25),
    -> ('Justice Rao', 47, 3, 22),
    -> ('Justice Bhaskar', 52, 4, 27),
    -> ('Justice Patel', 49, 5, 20);
Query OK, 5 rows affected (0.01 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> CREATE TABLE Cases (
    -> CaseID INT AUTO_INCREMENT PRIMARY KEY,
```

```
    -> CaseTitle VARCHAR(255),
    -> CaseType VARCHAR(50),
    -> DateFiled DATE,
    -> Status VARCHAR(20),
    -> CourtID INT,
    -> JudgeID INT,
    -> FOREIGN KEY (CourtID) REFERENCES Court(CourtID),
    -> FOREIGN KEY (JudgeID) REFERENCES Judge(JudgeID));
Query OK, 0 rows affected (0.04 sec)

mysql> INSERT INTO Cases (CaseTitle, CaseType, DateFiled, Status, CourtID,
JudgeID)VALUES
    -> ('State vs John Doe', 'Criminal', '2024-01-15', 'Active', 2, 1),
    -> ('Smith vs Johnson', 'Civil', '2024-02-10', 'Pending', 3, 2),
    -> ('People vs Richard Roe', 'Criminal', '2023-12-22', 'Closed', 4, 3);
Query OK, 3 rows affected (0.02 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> CREATE TABLE Lawyer (
    -> LawyerID INT AUTO_INCREMENT PRIMARY KEY,
    -> Name VARCHAR(100) NOT NULL,
    -> Specialization VARCHAR(100),
    -> CaseID INT,
    -> FOREIGN KEY (CaseID) REFERENCES Cases(CaseID));
Query OK, 0 rows affected (0.04 sec)

mysql> INSERT INTO Lawyer (Name, Specialization, CaseID)VALUES
    -> ('Lawyer Kumar', 'Criminal', 1),
    -> ('Lawyer Joshi', 'Civil', 2),
    -> ('Lawyer Desai', 'Criminal', 3);
Query OK, 3 rows affected (0.01 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> CREATE TABLE Plaintiff (
    -> PlaintiffID INT AUTO_INCREMENT PRIMARY KEY,
    -> Name VARCHAR(100) NOT NULL,
```

```
    -> Address VARCHAR(255),
    -> CaseID INT,
    -> FOREIGN KEY (CaseID) REFERENCES Cases(CaseID));
Query OK, 0 rows affected (0.06 sec)

mysql> INSERT INTO Plaintiff (Name, Address, CaseID) VALUES
    -> ('John Doe', '123 Main St, Delhi', 1),
    -> ('Jane Smith', '456 Market St, Mumbai', 2),
    -> ('Richard Roe', '789 Park Ave, Bangalore', 3);
Query OK, 3 rows affected (0.01 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> CREATE TABLE Hearing (
    -> HearingID INT AUTO_INCREMENT PRIMARY KEY,
    -> Date DATE,
    -> Outcome VARCHAR(255),
    -> JudgeID INT,
    -> CaseID INT,
    -> FOREIGN KEY (JudgeID) REFERENCES Judge(JudgeID),
    -> FOREIGN KEY (CaseID) REFERENCES Cases(CaseID));
Query OK, 0 rows affected (0.05 sec)

mysql> INSERT INTO Hearing (Date, Outcome, JudgeID, CaseID) VALUES
    -> ('2024-02-25', 'Verdict Pending', 1, 1),
    -> ('2024-03-10', 'Adjourned', 2, 2),
    -> ('2024-01-18', 'Guilty', 3, 3);
Query OK, 3 rows affected (0.01 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> ALTER TABLE Cases ADD COLUMN VerdictDate DATE;
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> select * from Cases;
+--------+---------------------+----------+------------+---------+---------+---------+-------------+
| CaseID | CaseTitle           | CaseType | DateFiled  | Status  | CourtID | JudgeID | VerdictDate |
```

```
+--------+----------------------+----------+------------+---------+---------+---------+-------------+
|      1 | State vs John Doe    | Criminal | 2024-01-15 | Active  |       2 |       1 | NULL        |
|      2 | Smith vs Johnson     | Civil    | 2024-02-10 | Pending |       3 |       2 | NULL        |
|      3 | People vs Richard Roe | Criminal | 2023-12-22 | Closed  |       4 |       3 | NULL        |
+--------+----------------------+----------+------------+---------+---------+---------+-------------+
3 rows in set (0.00 sec)

mysql> DROP TABLE IF EXISTS Lawyer;
Query OK, 0 rows affected (0.03 sec)

mysql> UPDATE Judge SET ExperienceYears = 35 WHERE JudgeID = 1;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from Judge;
+---------+----------------+------+---------+-----------------+
| JudgeID | Name           | Age  | CourtID | ExperienceYears |
+---------+----------------+------+---------+-----------------+
|       1 | Justice Sharma |   55 |       1 |              35 |
|       2 | Justice Verma  |   50 |       2 |              25 |
|       3 | Justice Rao    |   47 |       3 |              22 |
|       4 | Justice Bhaskar |   52 |       4 |              27 |
|       5 | Justice Patel  |   49 |       5 |              20 |
+---------+----------------+------+---------+-----------------+
5 rows in set (0.00 sec)

mysql> DELETE FROM Judge WHERE JudgeID = 5;
Query OK, 1 row affected (0.01 sec)

mysql> select * from Judge;
+---------+----------------+------+---------+-----------------+
| JudgeID | Name           | Age  | CourtID | ExperienceYears |
+---------+----------------+------+---------+-----------------+
|       1 | Justice Sharma |   55 |       1 |              35 |
|       2 | Justice Verma  |   50 |       2 |              25 |
|       3 | Justice Rao    |   47 |       3 |              22 |
```

```
|       4 | Justice Bhaskar |   52 |       4 |               27 |
+---------+-----------------+------+---------+-------------------+
4 rows in set (0.00 sec)

mysql> SELECT * FROM Cases WHERE Status = 'Active';
+--------+------------------+----------+------------+--------+---------+---------+-------------+
| CaseID | CaseTitle        | CaseType | DateFiled  | Status | CourtID | JudgeID | VerdictDate |
+--------+------------------+----------+------------+--------+---------+---------+-------------+
|      1 | State vs John Doe | Criminal | 2024-01-15 | Active |       2 |       1 | NULL        |
+--------+------------------+----------+------------+--------+---------+---------+-------------+
1 row in set (0.00 sec)

mysql> SELECT CourtID, COUNT(*) AS CaseCount
    -> FROM Cases
    -> GROUP BY CourtID
    -> HAVING COUNT(*) > 1;
Empty set (0.00 sec)

mysql> SELECT * FROM Judge ORDER BY Age DESC;
+---------+-----------------+------+---------+-------------------+
| JudgeID | Name            | Age  | CourtID | ExperienceYears |
+---------+-----------------+------+---------+-------------------+
|       1 | Justice Sharma  |   55 |       1 |               35 |
|       4 | Justice Bhaskar |   52 |       4 |               27 |
|       2 | Justice Verma   |   50 |       2 |               25 |
|       3 | Justice Rao     |   47 |       3 |               22 |
+---------+-----------------+------+---------+-------------------+
4 rows in set (0.00 sec)

mysql> SELECT DISTINCT CourtLevel FROM Court;
+------------+
| CourtLevel |
+------------+
| Supreme    |
| High       |
| District   |
```

```
+------------+
3 rows in set (0.00 sec)

mysql> SELECT * FROM Cases LIMIT 2;
+--------+------------------+----------+------------+--------+---------+---------+-------------+
| CaseID | CaseTitle        | CaseType | DateFiled  | Status | CourtID | JudgeID | VerdictDate |
+--------+------------------+----------+------------+--------+---------+---------+-------------+
|      1 | State vs John Doe | Criminal | 2024-01-15 | Active |       2 |       1 | NULL        |
|      2 | Smith vs Johnson | Civil    | 2024-02-10 | Pending |       3 |       2 | NULL        |
+--------+------------------+----------+------------+--------+---------+---------+-------------+
2 rows in set (0.00 sec)

mysql> SELECT COUNT(*) AS TotalCases FROM Cases;
+------------+
| TotalCases |
+------------+
|          3 |
+------------+
1 row in set (0.01 sec)

mysql> SELECT AVG(Age) FROM Judge;
+----------+
| AVG(Age) |
+----------+
|  51.0000 |
+----------+
1 row in set (0.00 sec)

mysql> SELECT * FROM Cases WHERE CaseTitle LIKE 'State%';
+--------+------------------+----------+------------+--------+---------+---------+-------------+
| CaseID | CaseTitle        | CaseType | DateFiled  | Status | CourtID | JudgeID | VerdictDate |
+--------+------------------+----------+------------+--------+---------+---------+-------------+
|      1 | State vs John Doe | Criminal | 2024-01-15 | Active |       2 |       1 | NULL        |
+--------+------------------+----------+------------+--------+---------+---------+-------------+
1 row in set (0.00 sec)

mysql> SELECT Cases.CaseTitle, Judge.Name
    -> FROM Cases
```

```
    -> INNER JOIN Judge ON Cases.JudgeID = Judge.JudgeID;
+----------------------+----------------+
| CaseTitle            | Name           |
+----------------------+----------------+
| State vs John Doe    | Justice Sharma |
| Smith vs Johnson     | Justice Verma  |
| People vs Richard Roe | Justice Rao    |
+----------------------+----------------+
3 rows in set (0.00 sec)

mysql> ^C
mysql> SELECT Cases.CaseTitle, Judge.Name
    -> FROM Cases
    -> LEFT JOIN Judge ON Cases.JudgeID = Judge.JudgeID;
+----------------------+----------------+
| CaseTitle            | Name           |
+----------------------+----------------+
| State vs John Doe    | Justice Sharma |
| Smith vs Johnson     | Justice Verma  |
| People vs Richard Roe | Justice Rao    |
+----------------------+----------------+
3 rows in set (0.00 sec)

mysql> SELECT Judge.Name, Cases.CaseTitle
    -> FROM Judge
    -> RIGHT JOIN Cases ON Judge.JudgeID = Cases.JudgeID;
+----------------+----------------------+
| Name           | CaseTitle            |
+----------------+----------------------+
| Justice Sharma | State vs John Doe    |
| Justice Verma  | Smith vs Johnson     |
| Justice Rao    | People vs Richard Roe |
+----------------+----------------------+
3 rows in set (0.00 sec)
```

```
mysql> SELECT Cases.CaseTitle, Judge.Name, Court.CourtName
    -> FROM Cases
    -> JOIN Judge ON Cases.JudgeID = Judge.JudgeID
    -> JOIN Court ON Judge.CourtID = Court.CourtID;
+----------------------+---------------+------------------+
| CaseTitle            | Name          | CourtName        |
+----------------------+---------------+------------------+
| State vs John Doe    | Justice Sharma | Supreme Court    |
| Smith vs Johnson     | Justice Verma  | High Court Delhi  |
| People vs Richard Roe | Justice Rao    | High Court Bombay |
+----------------------+---------------+------------------+
3 rows in set (0.00 sec)

mysql> SELECT UPPER(Name) AS UpperName FROM Judge;
+----------------+
| UpperName      |
+----------------+
| JUSTICE SHARMA  |
| JUSTICE VERMA   |
| JUSTICE RAO     |
| JUSTICE BHASKAR |
+----------------+
4 rows in set (0.00 sec)

mysql> SELECT CONCAT(CaseTitle, ' - ', Status) AS CaseDetails FROM Cases;
+-----------------------------+
| CaseDetails                 |
+-----------------------------+
| State vs John Doe - Active     |
| Smith vs Johnson - Pending     |
| People vs Richard Roe - Closed |
+-----------------------------+
3 rows in set (0.00 sec)

mysql> SELECT SUBSTRING(CourtName, 1, 5) AS ShortCourtName FROM Court;
```

```
+---------------+
| ShortCourtName |
+---------------+
| Distr          |
| Distr          |
| High           |
| High           |
| Supre          |
+---------------+
5 rows in set (0.00 sec)


mysql> SELECT ROUND(AVG(ExperienceYears), 2) FROM Judge;
+-------------------------------+
| ROUND(AVG(ExperienceYears), 2) |
+-------------------------------+
|                         27.25 |
+-------------------------------+
1 row in set (0.00 sec)


mysql> SELECT CEIL(ExperienceYears) AS CeilingExperience, FLOOR(ExperienceYears)
AS FloorExperience FROM Judge;
+------------------+-----------------+
| CeilingExperience | FloorExperience |
+------------------+-----------------+
|                35 |              35 |
|                25 |              25 |
|                22 |              22 |
|                27 |              27 |
+------------------+-----------------+
4 rows in set (0.00 sec)


mysql> SELECT CaseTitle, YEAR(DateFiled) AS FilingYear, MONTH(DateFiled) AS
FilingMonth FROM Cases;
+----------------------+-----------+-------------+
| CaseTitle             | FilingYear | FilingMonth |
+----------------------+-----------+-------------+
```

```
| State vs John Doe     |       2024 |           1 |
| Smith vs Johnson      |       2024 |           2 |
| People vs Richard Roe |       2023 |          12 |
+-----------------------+------------+-------------+
3 rows in set (0.00 sec)


mysql> SELECT CURDATE() AS CurrentDate;
+-------------+
| CurrentDate |
+-------------+
| 2024-10-17  |
+-------------+
1 row in set (0.00 sec)


mysql> SELECT DATEDIFF(NOW(), DateFiled) AS DaysSinceFiled FROM Cases;
+----------------+
| DaysSinceFiled |
+----------------+
|            276 |
|            250 |
|            300 |
+----------------+
3 rows in set (0.00 sec)


mysql> SELECT Name FROM Judge WHERE Age > ALL (SELECT Age FROM Judge WHERE
CourtID = 1);
Empty set (0.00 sec)


mysql> SELECT Name FROM Judge WHERE JudgeID IN (SELECT JudgeID FROM Cases WHERE
Status = 'Active');
+----------------+
| Name           |
+----------------+
| Justice Sharma |
+----------------+
1 row in set (0.00 sec)
```

```
mysql> SELECT Name FROM Judge WHERE JudgeID IN (SELECT JudgeID FROM Cases WHERE
DateFiled < '2024-01-01');
+-------------+
| Name        |
+-------------+
| Justice Rao |
+-------------+
1 row in set (0.00 sec)


mysql> SELECT Judge.Name FROM Judge
    -> LEFT JOIN Cases ON Judge.JudgeID = Cases.JudgeID
    -> WHERE Cases.JudgeID IS NULL;
+-----------------+
| Name            |
+-----------------+
| Justice Bhaskar |
+-----------------+
1 row in set (0.00 sec)


mysql> SELECT Judge.Name, Court.CourtName FROM Judge, Court;
+-----------------+-------------------------+
| Name            | CourtName               |
+-----------------+-------------------------+
| Justice Bhaskar | District Court Bangalore |
| Justice Rao     | District Court Bangalore |
| Justice Verma   | District Court Bangalore |
| Justice Sharma  | District Court Bangalore |
| Justice Bhaskar | District Court Mumbai    |
| Justice Rao     | District Court Mumbai    |
| Justice Verma   | District Court Mumbai    |
| Justice Sharma  | District Court Mumbai    |
| Justice Bhaskar | High Court Bombay        |
| Justice Rao     | High Court Bombay        |
| Justice Verma   | High Court Bombay        |
| Justice Sharma  | High Court Bombay        |
```

```
| Justice Bhaskar | High Court Delhi         |
| Justice Rao     | High Court Delhi         |
| Justice Verma   | High Court Delhi         |
| Justice Sharma  | High Court Delhi         |
| Justice Bhaskar | Supreme Court            |
| Justice Rao     | Supreme Court            |
| Justice Verma   | Supreme Court            |
| Justice Sharma  | Supreme Court            |
+-----------------+--------------------------+
20 rows in set (0.00 sec)

mysql> SELECT Cases.CaseTitle, Judge.Name
    -> FROM Cases
    -> LEFT JOIN Judge ON Cases.JudgeID = Judge.JudgeID
    -> UNION
    -> SELECT Cases.CaseTitle, Judge.Name
    -> FROM Cases
    -> RIGHT JOIN Judge ON Cases.JudgeID = Judge.JudgeID;
+----------------------+-----------------+
| CaseTitle            | Name            |
+----------------------+-----------------+
| State vs John Doe    | Justice Sharma  |
| Smith vs Johnson     | Justice Verma   |
| People vs Richard Roe | Justice Rao    |
| NULL                 | Justice Bhaskar |
+----------------------+-----------------+
4 rows in set (0.00 sec)

mysql> SELECT Judge.Name
    -> FROM Judge
    -> JOIN Cases ON Judge.JudgeID = Cases.JudgeID
    -> WHERE CaseType IN ('Criminal', 'Civil')
    -> GROUP BY Judge.Name
    -> HAVING COUNT(DISTINCT CaseType) = 2;
Empty set (0.00 sec)
```

```
mysql> SELECT Judge.Name AS JudgeName, Court.CourtName AS AssignedCourt
    -> FROM Judge
    -> JOIN Court ON Judge.CourtID = Court.CourtID;
+----------------+----------------------+
| JudgeName      | AssignedCourt        |
+----------------+----------------------+
| Justice Sharma | Supreme Court        |
| Justice Verma  | High Court Delhi     |
| Justice Rao    | High Court Bombay    |
| Justice Bhaskar | District Court Mumbai |
+----------------+----------------------+
4 rows in set (0.00 sec)

mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> UPDATE Cases SET Status = 'Closed' WHERE CaseID = 1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SAVEPOINT BeforeExperienceUpdate;
Query OK, 0 rows affected (0.00 sec)

mysql> UPDATE Judge SET ExperienceYears = 40 WHERE JudgeID = 1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> ROLLBACK TO BeforeExperienceUpdate;
Query OK, 0 rows affected (0.00 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.01 sec)

mysql> SET autocommit = 0;
```

```
Query OK, 0 rows affected (0.00 sec)

mysql> BEGIN;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO Judge (Name, Age, CourtID, ExperienceYears)
    -> VALUES ('Justice Desai', 48, 2, 20);
Query OK, 1 row affected (0.00 sec)

mysql> COMMIT;
Query OK, 0 rows affected (0.01 sec)

mysql> SET autocommit = 1;
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE VIEW AllCases AS  SELECT * FROM Cases;
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE VIEW SeniorJudges AS
    -> SELECT Name, Age  FROM Judge
    -> WHERE Age > 50;
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE VIEW DelhiCases AS
    -> SELECT CaseTitle, CourtID
    -> FROM Cases
    -> WHERE CourtID IN (SELECT CourtID FROM Court WHERE Location = 'Delhi');
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE VIEW JudgeCases AS
    -> SELECT Judge.Name AS JudgeName, Cases.CaseTitle
    -> FROM Judge
    -> JOIN Cases ON Judge.JudgeID = Cases.JudgeID;
Query OK, 0 rows affected (0.01 sec)
```

```
mysql> UPDATE SeniorJudges
    -> SET Age = 60
    -> WHERE Name = 'Justice Verma';
Query OK, 0 rows affected (0.01 sec)
Rows matched: 0  Changed: 0  Warnings: 0

mysql> INSERT INTO SeniorJudges (Name, Age) VALUES ('Justice Desai', 55);
Query OK, 1 row affected (0.00 sec)

mysql> UPDATE SeniorJudges
    -> SET Age = CASE
    -> WHEN Age < 60 THEN Age + 1
    -> ELSE Age
    -> END;
Query OK, 3 rows affected (0.00 sec)
Rows matched: 3  Changed: 3  Warnings: 0
```