**Indian Matrimony Services**

**Domain Description**

The Indian Matrimony Services domain operates as an online platform designed to facilitate matrimonial matches for individuals seeking marriage partners. The core functionality of this domain revolves around user profiles, which serve as the primary entities. Users create detailed profiles that include personal information such as names, contact details, dates of birth, and preferences for potential partners.

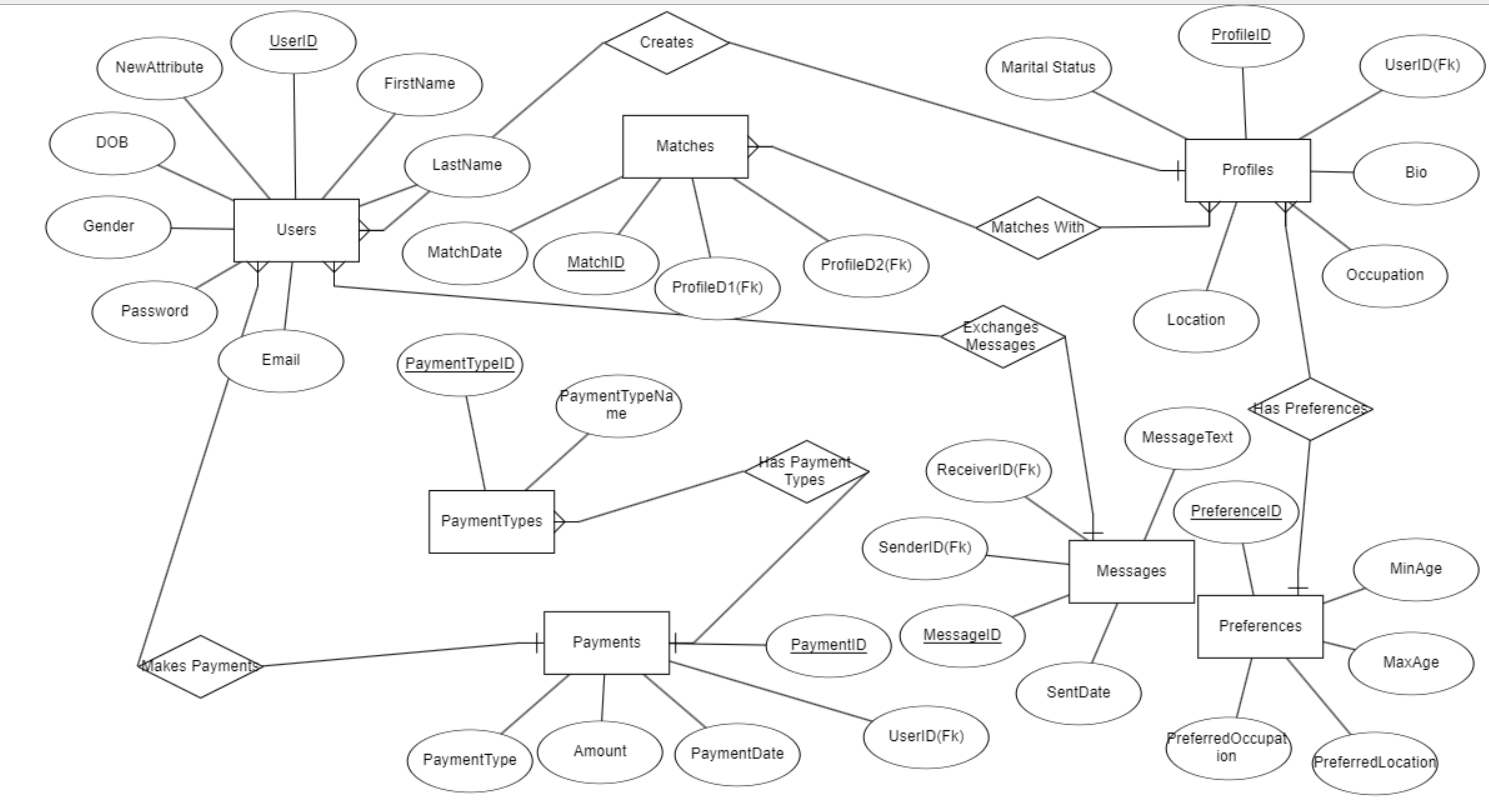
The platform incorporates a robust database management system to handle user data efficiently. Each user is assigned a unique identifier (UserID) that helps track their activities, preferences, and interactions on the platform. Users can upload photos, specify their backgrounds, interests, and expectations in a partner, which enhances the matching process.

To sustain the platform's operations, users can make payments for premium services. The Payments table captures all financial transactions, linking each payment to a specific user. Payment types are categorized in a separate PaymentTypes table, allowing users to choose from various subscription models or one-time fees for accessing enhanced features such as personalized matchmaking services or visibility to a broader audience.

The system utilizes algorithms to match users based on compatibility criteria, such as preferences, backgrounds, and interests, while ensuring that data privacy and security measures are in place. Users can browse potential matches, send inquiries, and communicate within the platform. The matching process is supported by advanced features like filters and search criteria, which help users find partners that meet their specific needs.

In summary, the Indian Matrimony Services domain provides a structured and user-friendly environment for individuals looking to find life partners. By combining user-generated profiles, payment systems, and effective matching algorithms, the platform aims to streamline the matrimonial search process while ensuring a secure and personalized experience for its users.

**ER Diagram**



**Queries and output**

**DDL (Data Definition Language) commands are used to define and manage the structure of database objects, such as tables, indexes, and views. These commands help create, alter, and delete objects within a database.**

**The CREATE command is used to create a new database object, such as a table, index, or view.**

**The ALTER command is used to modify the structure of an existing database object, such as adding, deleting, or modifying columns in a table.**

**The DROP command deletes an existing database object, such as a table, index, or view. This is a permanent action, and the data and structure will be lost.**

**The TRUNCATE command removes all records from a table but does not delete the table structure itself. It is faster than the DELETE command because it does not log individual row deletions.**

**The RENAME command changes the name of an existing database object, such as a table.**

mysql> CREATE DATABASE MatrimonyService;

Query OK, 1 row affected (0.03 sec)

mysql> USE MatrimonyService;

Database changed

mysql>

mysql> -- Users Table

mysql> CREATE TABLE Users (

-> UserID INT PRIMARY KEY AUTO\_INCREMENT,

-> FirstName VARCHAR(50),

-> LastName VARCHAR(50),

-> Email VARCHAR(100) UNIQUE,

-> Password VARCHAR(255),

-> Gender ENUM('Male', 'Female', 'Other'),

-> DOB DATE

-> );

Query OK, 0 rows affected (0.15 sec)

mysql>

mysql> -- Profiles Table

mysql> CREATE TABLE Profiles (

-> ProfileID INT PRIMARY KEY AUTO\_INCREMENT,

-> UserID INT,

-> Bio TEXT,

-> Occupation VARCHAR(100),

-> Location VARCHAR(100),

-> MaritalStatus ENUM('Single', 'Divorced', 'Widowed'),

-> FOREIGN KEY (UserID) REFERENCES Users(UserID)

-> );

Query OK, 0 rows affected (0.15 sec)

mysql>

mysql> -- Preferences Table

mysql> CREATE TABLE Preferences (

-> PreferenceID INT PRIMARY KEY AUTO\_INCREMENT,

-> ProfileID INT,

-> MinAge INT,

-> MaxAge INT,

-> PreferredLocation VARCHAR(100),

-> PreferredOccupation VARCHAR(100),

-> FOREIGN KEY (ProfileID) REFERENCES Profiles(ProfileID)

-> );

Query OK, 0 rows affected (0.08 sec)

mysql>

mysql> -- Matches Table

mysql> CREATE TABLE Matches (

-> MatchID INT PRIMARY KEY AUTO\_INCREMENT,

-> ProfileID1 INT,

-> ProfileID2 INT,

-> MatchDate DATE,

-> FOREIGN KEY (ProfileID1) REFERENCES Profiles(ProfileID),

-> FOREIGN KEY (ProfileID2) REFERENCES Profiles(ProfileID)

-> );

Query OK, 0 rows affected (0.11 sec)

mysql>

mysql> -- Messages Table

mysql> CREATE TABLE Messages (

-> MessageID INT PRIMARY KEY AUTO\_INCREMENT,

-> SenderID INT,

-> ReceiverID INT,

-> MessageText TEXT,

-> SentDate DATETIME,

-> FOREIGN KEY (SenderID) REFERENCES Users(UserID),

-> FOREIGN KEY (ReceiverID) REFERENCES Users(UserID)

-> );

Query OK, 0 rows affected (0.10 sec)

mysql>

mysql> -- Payments Table

mysql> CREATE TABLE Payments (

-> PaymentID INT PRIMARY KEY AUTO\_INCREMENT,

-> UserID INT,

-> PaymentDate DATE,

-> Amount DECIMAL(10, 2),

-> PaymentType ENUM('Credit Card', 'Debit Card', 'Net Banking', 'UPI'),

-> FOREIGN KEY (UserID) REFERENCES Users(UserID)

-> );

Query OK, 0 rows affected (0.09 sec)

mysql> -- Alter table to add a new column

mysql> ALTER TABLE Users ADD PhoneNumber VARCHAR(15);

Query OK, 0 rows affected (0.09 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql>

mysql> -- Modify column size

mysql> ALTER TABLE Profiles MODIFY Occupation VARCHAR(150);

Query OK, 0 rows affected (0.05 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql>

mysql> -- Drop a column

mysql> ALTER TABLE Payments DROP COLUMN PaymentType;

Query OK, 0 rows affected (0.24 sec)

Records: 0 Duplicates: 0 Warnings: 0

**DML (Data Manipulation Language) commands are SQL (Structured Query Language) statements used to manipulate data within database tables. These commands allow users to perform actions such as inserting, updating, deleting, and retrieving data.**

**The INSERT command is used to add new records (rows) into a table.**

**The UPDATE command modifies existing records in a table. It allows you to change one or more fields in one or multiple rows.**

**The DELETE command removes records from a table. You can delete specific rows based on a condition or all rows if no condition is specified.**

**The SELECT command is used to retrieve data from a database. You can specify the columns you want to retrieve and apply conditions to filter the results.**

mysql> -- Inserting Users

mysql> INSERT INTO Users (FirstName, LastName, Email, Password, Gender, DOB)

-> VALUES

-> ('John', 'Doe', 'john@example.com', 'password123', 'Male', '1990-01-01'),

-> ('Jane', 'Doe', 'jane@example.com', 'password456', 'Female', '1992-03-12'),

-> ('Amit', 'Singh', 'amit@example.com', 'amit123', 'Male', '1985-06-15');

Query OK, 3 rows affected (0.01 sec)

Records: 3 Duplicates: 0 Warnings: 0

mysql>

mysql> -- Inserting Profiles

mysql> INSERT INTO Profiles (UserID, Bio, Occupation, Location, MaritalStatus)

-> VALUES

-> (1, 'Software Engineer looking for a match', 'Software Engineer', 'New York', 'Single'),

-> (2, 'Doctor seeking a life partner', 'Doctor', 'Los Angeles', 'Single'),

-> (3, 'Teacher looking for a partner', 'Teacher', 'Mumbai', 'Single');

Query OK, 3 rows affected (0.02 sec)

Records: 3 Duplicates: 0 Warnings: 0

mysql>

mysql> -- Inserting Preferences

mysql> INSERT INTO Preferences (ProfileID, MinAge, MaxAge, PreferredLocation, PreferredOccupation)

-> VALUES

-> (1, 25, 35, 'New York', 'Doctor'),

-> (2, 30, 40, 'Los Angeles', 'Engineer'),

-> (3, 28, 38, 'Mumbai', 'Teacher');

Query OK, 3 rows affected (0.01 sec)

Records: 3 Duplicates: 0 Warnings: 0

mysql>

mysql> -- Inserting Matches

mysql> INSERT INTO Matches (ProfileID1, ProfileID2, MatchDate)

-> VALUES

-> (1, 2, '2024-01-01'),

-> (3, 1, '2024-02-15');

Query OK, 2 rows affected (0.01 sec)

Records: 2 Duplicates: 0 Warnings: 0

mysql>

mysql> -- Inserting Messages

mysql> INSERT INTO Messages (SenderID, ReceiverID, MessageText, SentDate)

-> VALUES

-> (1, 2, 'Hi, would you like to connect?', '2024-01-02 10:00:00'),

-> (2, 1, 'Sure, let’s chat.', '2024-01-02 10:05:00');

Query OK, 2 rows affected (0.01 sec)

Records: 2 Duplicates: 0 Warnings: 0

mysql>

mysql> -- Inserting Payments

mysql> INSERT INTO Payments (UserID, PaymentDate, Amount, PaymentType)

-> VALUES

-> (1, '2024-01-01', 100.00, 'Credit Card'),

-> (2, '2024-01-02', 150.00, 'UPI');

Query OK, 2 rows affected (0.01 sec)

Records: 2 Duplicates: 0 Warnings: 0

mysql> -- Update user's phone number

mysql> UPDATE Users SET PhoneNumber = '9876543210' WHERE UserID = 1;

Query OK, 1 row affected (0.01 sec)

Rows matched: 1 Changed: 1 Warnings: 0

mysql>

mysql> -- Delete a record from the Messages table

mysql> DELETE FROM Messages WHERE MessageID = 2;

Query OK, 1 row affected (0.01 sec)

mysql>

mysql> -- Insert a new payment record

mysql> INSERT INTO Payments (UserID, PaymentDate, Amount) VALUES (3, '2024-03-01', 200.00);

Query OK, 1 row affected (0.01 sec)

**In MySQL, constraints are rules that ensure the accuracy, integrity, and consistency of the data in a table. The NOT NULL constraint enforces that a column must always contain a value, meaning it cannot be left empty. The UNIQUE constraint ensures that all the values in a column are different, preventing duplicate entries. The PRIMARY KEY constraint uniquely identifies each record in a table and combines both the NOT NULL and UNIQUE properties, making sure each entry is distinct and not empty. The FOREIGN KEY constraint is used to link two tables together, enforcing referential integrity by ensuring that a value in one table must correspond to an existing value in another table. The CHECK constraint validates that the data in a column meets a specific condition or rule. Lastly, the DEFAULT constraint provides a default value for a column when no explicit value is supplied during an insert operation. All these constraints help maintain data quality and define relationships between tables in a MySQL database.**

mysql> -- Adding a UNIQUE constraint on Email column

mysql> ALTER TABLE Users ADD CONSTRAINT unique\_email UNIQUE (Email);

Query OK, 0 rows affected, 1 warning (0.10 sec)

Records: 0 Duplicates: 0 Warnings: 1

mysql>

mysql> -- Adding a CHECK constraint to ensure age preferences are valid

mysql> ALTER TABLE Preferences ADD CONSTRAINT check\_age\_range CHECK (MinAge < MaxAge);

Query OK, 3 rows affected (0.25 sec)

Records: 3 Duplicates: 0 Warnings: 0

**The WHERE clause in SQL is used to filter records based on specific conditions, allowing you to retrieve, update, or delete only the rows that meet the criteria defined. It is commonly used in conjunction with SELECT, UPDATE, DELETE, and INSERT statements. The conditions within the WHERE clause can be based on comparisons.**

mysql> -- WHERE Clause

mysql> SELECT \* FROM Users WHERE Gender = 'Male';

+--------+-----------+----------+------------------+-------------+--------+------------+-------------+

| UserID | FirstName | LastName | Email | Password | Gender | DOB | PhoneNumber |

+--------+-----------+----------+------------------+-------------+--------+------------+-------------+

| 1 | John | Doe | john@example.com | password123 | Male | 1990-01-01 | 9876543210 |

| 3 | Amit | Singh | amit@example.com | amit123 | Male | 1985-06-15 | NULL |

+--------+-----------+----------+-

**GROUP BY Clause**

**The GROUP BY clause is used to group rows that have the same values in specified columns into summary rows, often in combination with aggregate functions (like COUNT(), SUM(), AVG(), etc.). It allows you to perform operations on each group of data rather than on individual rows. For example, grouping sales data by product type to get total sales per product.**

**HAVING Clause**

**The HAVING clause is used to filter the results of a GROUP BY query. It works similarly to the WHERE clause but is applied after the data has been grouped. You use HAVING to filter groups based on aggregate conditions (e.g., showing only product categories with total sales greater than 10,000).**

mysql>

mysql> -- GROUP BY and HAVING

mysql> SELECT Location, COUNT(\*) AS TotalProfiles FROM Profiles GROUP BY Location HAVING COUNT(\*) > 1;

Empty set (0.01 sec)

mysql>

mysql> -- ORDER BY

mysql> SELECT \* FROM Profiles ORDER BY Location ASC;

+-----------+--------+---------------------------------------+-------------------+-------------+---------------+

| ProfileID | UserID | Bio | Occupation | Location | MaritalStatus |

+-----------+--------+---------------------------------------+-------------------+-------------+---------------+

| 2 | 2 | Doctor seeking a life partner | Doctor | Los Angeles | Single |

| 3 | 3 | Teacher looking for a partner | Teacher | Mumbai | Single |

| 1 | 1 | Software Engineer looking for a match | Software Engineer | New York | Single |

+-----------+--------+---------------------------------------+-------------------+-------------+---------------+

3 rows in set (0.00 sec)

**ORDER BY Clause**

**The ORDER BY clause is used to sort the result set of a query by one or more columns in either ascending (ASC) or descending (DESC) order. For instance, sorting a list of employees by salary or sorting products by price from highest to lowest.**

mysql> -- ORDER BY

mysql> SELECT \* FROM Profiles ORDER BY Location ASC;

+-----------+--------+---------------------------------------+-------------------+-------------+---------------+

| ProfileID | UserID | Bio | Occupation | Location | MaritalStatus |

+-----------+--------+---------------------------------------+-------------------+-------------+---------------+

| 2 | 2 | Doctor seeking a life partner | Doctor | Los Angeles | Single |

| 3 | 3 | Teacher looking for a partner | Teacher | Mumbai | Single |

| 1 | 1 | Software Engineer looking for a match | Software Engineer | New York | Single |

+-----------+--------+---------------------------------------+-------------------+-------------+---------------+

3 rows in set (0.00 sec)

**DISTINCT Keyword**

**The DISTINCT keyword is used to return only unique values, eliminating duplicates from the result set. It is often used when retrieving records where duplicates are not needed, such as getting a list of unique customer cities.**

mysql> -- DISTINCT

mysql> SELECT DISTINCT Location FROM Profiles;

+-------------+

| Location |

+-------------+

| New York |

| Los Angeles |

| Mumbai |

+-------------+

3 rows in set (0.00 sec)

**LIMIT Keyword**

**The LIMIT keyword is used to restrict the number of rows returned by a query. For example, if you want to retrieve only the top 5 results from a query, you can use LIMIT**

mysql> -- LIMIT

mysql> SELECT \* FROM Users LIMIT 2;

+--------+-----------+----------+------------------+-------------+--------+------------+-------------+

| UserID | FirstName | LastName | Email | Password | Gender | DOB | PhoneNumber |

+--------+-----------+----------+------------------+-------------+--------+------------+-------------+

| 1 | John | Doe | john@example.com | password123 | Male | 1990-01-01 | 9876543210 |

| 2 | Jane | Doe | jane@example.com | password456 | Female | 1992-03-12 | NULL |

+--------+-----------+----------+------------------+-------------+--------+------------+-------------+

2 rows in set (0.00 sec)

**Aggregate Functions**

**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 total sum of a numeric column.**

**AVG(): Returns the average value of a numeric column.**

**MIN() and MAX(): Return the smallest and largest values, respectively, from a column.**

mysql> -- Aggregate Functions

mysql> SELECT AVG(Amount) AS AvgPayment, SUM(Amount) AS TotalPayment FROM Payments;

+------------+--------------+

| AvgPayment | TotalPayment |

+------------+--------------+

| 150.000000 | 450.00 |

+------------+--------------+

1 row in set (0.01 sec)

**Pattern Matching using LIKE**

**The LIKE keyword is used in a WHERE clause to search for a specified pattern in a column. It supports two wildcards:**

**%: Represents zero or more characters (e.g., LIKE 'A%' matches any value starting with "A").**

**\_: Represents a single character (e.g., LIKE '\_b%' matches any value where the second character is "b"). LIKE is often used for flexible text searches in databases.**

**These SQL concepts are essential for querying, filtering, and analyzing data efficiently in relational databases.**

mysql> -- Pattern Matching using LIKE

mysql> SELECT \* FROM Users WHERE FirstName LIKE 'J%';

+--------+-----------+----------+------------------+-------------+--------+------------+-------------+

| UserID | FirstName | LastName | Email | Password | Gender | DOB | PhoneNumber |

+--------+-----------+----------+------------------+-------------+--------+------------+-------------+

| 1 | John | Doe | john@example.com | password123 | Male | 1990-01-01 | 9876543210 |

| 2 | Jane | Doe | jane@example.com | password456 | Female | 1992-03-12 | NULL |

+--------+-----------+----------+------------------+-------------+--------+------------+-------------+

2 rows in set (0.01 sec)

**Joins are used in SQL to combine rows from two or more tables based on related columns.**

**An INNER JOIN returns only the rows that have matching values in both tables; if there is no match, those rows are excluded.**

**A LEFT JOIN (or LEFT OUTER JOIN) returns all rows from the left table and the matching rows from the right table. If there is no match, the result will include NULL values for columns from the right table.**

**A RIGHT JOIN (or RIGHT OUTER JOIN) is the opposite; it returns all rows from the right table and the matching rows from the left table, with NULL values for columns from the left table where there is no match.**

**A FULL OUTER JOIN returns all rows when there is a match in either table, and if no match is found, it shows NULL for non-matching columns from either side.**

mysql> SELECT Users.FirstName, Profiles.Bio FROM Users

-> INNER JOIN Profiles ON Users.UserID = Profiles.UserID;

+-----------+---------------------------------------+

| FirstName | Bio |

+-----------+---------------------------------------+

| John | Software Engineer looking for a match |

| Jane | Doctor seeking a life partner |

| Amit | Teacher looking for a partner |

+-----------+---------------------------------------+

3 rows in set (0.00 sec)

mysql> SELECT Users.FirstName, Payments.Amount FROM Users

-> LEFT JOIN Payments ON Users.UserID = Payments.UserID

-> LEFT JOIN Profiles ON Users.UserID = Profiles.UserID;

+-----------+--------+

| FirstName | Amount |

+-----------+--------+

| John | 100.00 |

| Jane | 150.00 |

| Amit | 200.00 |

+-----------+--------+

3 rows in set (0.00 sec)

mysql> SELECT Payments.UserID, Profiles.Bio FROM Payments

-> RIGHT JOIN Profiles ON Payments.UserID = Profiles.UserID

-> RIGHT JOIN Users ON Profiles.UserID = Users.UserID

-> RIGHT JOIN Messages ON Users.UserID = Messages.SenderID;

+--------+---------------------------------------+

| UserID | Bio |

+--------+---------------------------------------+

| 1 | Software Engineer looking for a match |

+--------+---------------------------------------+

1 row in set (0.01 sec)

mysql> SELECT \*

-> FROM Profiles

-> LEFT JOIN Matches ON Profiles.ProfileID = Matches.ProfileID1

->

-> UNION

->

-> SELECT \*

-> FROM Profiles

-> RIGHT JOIN Matches ON Profiles.ProfileID = Matches.ProfileID1;

+-----------+--------+---------------------------------------+-------------------+-------------+---------------+---------+------------+------------+------------+

| ProfileID | UserID | Bio | Occupation | Location | MaritalStatus | MatchID | ProfileID1 | ProfileID2 | MatchDate |

+-----------+--------+---------------------------------------+-------------------+-------------+---------------+---------+------------+------------+------------+

| 1 | 1 | Software Engineer looking for a match | Software Engineer | New York | Single | 1 | 1 | 2 | 2024-01-01 |

| 2 | 2 | Doctor seeking a life partner | Doctor | Los Angeles | Single | NULL | NULL | NULL | NULL |

| 3 | 3 | Teacher looking for a partner | Teacher | Mumbai | Single | 2 | 3 | 1 | 2024-02-15 |

+-----------+--------+---------------------------------------+-------------------+-------------+---------------+---------+------------+------------+------------+

3 rows in set (0.01 sec)

**In SQL, functions help manipulate and retrieve data in various ways, with string, numeric, and date-time functions being commonly used. String functions operate on text data. For example, UPPER() converts a string to uppercase, LOWER() to lowercase, LENGTH() returns the number of characters in a string, SUBSTRING() extracts part of a string, and CONCAT() combines multiple strings. Numeric functions deal with numerical data. Common examples include ROUND() to round a number to a specific number of decimal places, ABS() to return the absolute value of a number, POWER() for raising a number to a power, and CEIL() or FLOOR() to round a number up or down to the nearest integer. Date-time functions manipulate dates and times. For instance, NOW() returns the current date and time, DATE() extracts the date part from a date-time value, DATEDIFF() calculates the difference between two dates, and DATE\_ADD() or DATE\_SUB() adds or subtracts a specified interval from a date. These functions are crucial for performing calculations and formatting data in SQL queries.**

mysql> SELECT ROUND(AVG(Amount), 2) FROM Payments;

+-----------------------+

| ROUND(AVG(Amount), 2) |

+-----------------------+

| 150.00 |

+-----------------------+

1 row in set (0.00 sec)

mysql> SELECT CONCAT(FirstName, ' ', LastName) AS FullName FROM Users;

+------------+

| FullName |

+------------+

| John Doe |

| Jane Doe |

| Amit Singh |

+------------+

3 rows in set (0.01 sec)

mysql> SELECT YEAR(DOB) AS BirthYear FROM Users;

+-----------+

| BirthYear |

+-----------+

| 1990 |

| 1992 |

| 1985 |

+-----------+

3 rows in set (0.00 sec)

**Nested queries and subqueries in SQL refer to queries that are placed inside another query to help retrieve specific data based on complex conditions. A subquery is an inner query enclosed within parentheses that provides input to the outer query. It can appear in the SELECT, FROM, WHERE, or HAVING clause of the main query. Subqueries are often used to compare values against a set of results or to filter data dynamically based on the result of the inner query. For example, you might use a subquery to find employees whose salary is above the average salary (where the average is calculated by the inner query). Nested queries are essentially the same concept, where one query depends on the result of another, but these can involve multiple layers of queries. There are two types: correlated subqueries, where the inner query depends on values from the outer query for each row, and non-correlated subqueries, where the inner query runs independently and provides a result that the outer query can use. Subqueries simplify complex filtering and comparison tasks in SQL by allowing queries to be built in layers.**

mysql> SELECT FirstName, UserID

-> FROM Users

-> WHERE UserID IN (

-> SELECT UserID

-> FROM Payments

-> WHERE Amount > (SELECT AVG(Amount) FROM Payments)

-> );

+-----------+--------+

| FirstName | UserID |

+-----------+--------+

| Amit | 3 |

+-----------+--------+

1 row in set (0.00 sec)

mysql> SELECT ProfileID, Location FROM Profiles P1 WHERE EXISTS (

-> SELECT \* FROM Profiles P2 WHERE P1.Location = P2.Location AND P1.ProfileID != P2.ProfileID

-> );

Empty set (0.00 sec)

**In SQL, a Cartesian Product occurs when two or more tables are combined without any conditions or joins, meaning every row from the first table is paired with every row from the second table. This results in a large result set where the total number of rows is the product of the number of rows in each table. For example, if one table has 5 rows and another has 4, the Cartesian product will produce 20 rows (5 \* 4). While this operation can be useful in certain cases, it often produces large and unintended results if not properly constrained by conditions like WHERE or JOIN clauses.**

**Division in relational algebra (not directly available in SQL but can be simulated) is used when you want to find all records in one set that are related to all records in another set. This is typically used in cases where you're looking for a relationship that must exist across all possible combinations. For instance, if you want to find students who have completed all required courses, you would use a division-like query. Simulating division in SQL typically involves using a combination of GROUP BY, HAVING, and COUNT to ensure that the condition holds across all elements of a subset.**

**Both Cartesian product and division are fundamental concepts in relational algebra, but in practical SQL use, they need careful handling to ensure that results are meaningful and efficient.**

mysql> SELECT \*

-> FROM Users

-> CROSS JOIN Profiles;

+--------+-----------+----------+------------------+-------------+--------+------------+-------------+-----------+--------+---------------------------------------+-------------------+-------------+---------------+

| UserID | FirstName | LastName | Email | Password | Gender | DOB | PhoneNumber | ProfileID | UserID | Bio | Occupation | Location | MaritalStatus |

+--------+-----------+----------+------------------+-------------+--------+------------+-------------+-----------+--------+---------------------------------------+-------------------+-------------+---------------+

| 3 | Amit | Singh | amit@example.com | amit123 | Male | 1985-06-15 | NULL | 1 | 1 | Software Engineer looking for a match | Software Engineer | New York | Single |

| 2 | Jane | Doe | jane@example.com | password456 | Female | 1992-03-12 | NULL | 1 | 1 | Software Engineer looking for a match | Software Engineer | New York | Single |

| 1 | John | Doe | john@example.com | password123 | Male | 1990-01-01 | 9876543210 | 1 | 1 | Software Engineer looking for a match | Software Engineer | New York | Single |

| 3 | Amit | Singh | amit@example.com | amit123 | Male | 1985-06-15 | NULL | 2 | 2 | Doctor seeking a life partner | Doctor | Los Angeles | Single |

| 2 | Jane | Doe | jane@example.com | password456 | Female | 1992-03-12 | NULL | 2 | 2 | Doctor seeking a life partner | Doctor | Los Angeles | Single |

| 1 | John | Doe | john@example.com | password123 | Male | 1990-01-01 | 9876543210 | 2 | 2 | Doctor seeking a life partner | Doctor | Los Angeles | Single |

| 3 | Amit | Singh | amit@example.com | amit123 | Male | 1985-06-15 | NULL | 3 | 3 | Teacher looking for a partner | Teacher | Mumbai | Single |

| 2 | Jane | Doe | jane@example.com | password456 | Female | 1992-03-12 | NULL | 3 | 3 | Teacher looking for a partner | Teacher | Mumbai | Single |

| 1 | John | Doe | john@example.com | password123 | Male | 1990-01-01 | 9876543210 | 3 | 3 | Teacher looking for a partner | Teacher | Mumbai | Single |

+--------+-----------+----------+------------------+-------------+--------+------------+-------------+-----------+--------+---------------------------------------+-------------------+-------------+---------------+

9 rows in set (0.00 sec)

mysql> CREATE TABLE PaymentTypes (

-> PaymentTypeID INT PRIMARY KEY AUTO\_INCREMENT,

-> PaymentType VARCHAR(50)

-> );

Query OK, 0 rows affected (0.06 sec)

mysql>

mysql> INSERT INTO PaymentTypes (PaymentType) VALUES

-> ('Credit Card'),

-> ('Debit Card'),

-> ('UPI'),

-> ('Net Banking');

Query OK, 4 rows affected (0.01 sec)

Records: 4 Duplicates: 0 Warnings: 0

mysql> ALTER TABLE Payments

-> ADD PaymentType VARCHAR(50);

Query OK, 0 rows affected (0.07 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> INSERT INTO Payments (UserID, PaymentDate, Amount, PaymentType) VALUES

-> (1, '2024-01-01', 100.00, 'Credit Card'),

-> (1, '2024-01-02', 150.00, 'Debit Card'),

-> (2, '2024-01-03', 200.00, 'Credit Card'),

-> (2, '2024-01-04', 250.00, 'UPI'),

-> (3, '2024-01-05', 300.00, 'Debit Card'),

-> (3, '2024-01-06', 350.00, 'Net Banking');

Query OK, 6 rows affected (0.01 sec)

Records: 6 Duplicates: 0 Warnings: 0

mysql> SELECT U.UserID

-> FROM Users U

-> WHERE NOT EXISTS (

-> SELECT PT.PaymentType

-> FROM PaymentTypes PT

-> WHERE NOT EXISTS (

-> SELECT P.PaymentType

-> FROM Payments P

-> WHERE P.UserID = U.UserID

-> AND P.PaymentType = PT.PaymentType

-> )

-> );

Empty set (0.00 sec)

**TCL (Transaction Control Language) commands in SQL are used to manage and control transactions in a database. Transactions are a sequence of operations that are executed as a single unit of work. TCL commands ensure that these operations either all succeed together or fail together, maintaining the integrity and consistency of the database. TCL includes key commands such as COMMIT, ROLLBACK, SAVEPOINT, and SET TRANSACTION.**

mysql> -- Start a transaction

mysql> START TRANSACTION;

Query OK, 0 rows affected (0.00 sec)

mysql>

mysql> -- Insert a payment

mysql> INSERT INTO Payments (UserID, PaymentDate, Amount) VALUES (1, '2024-03-05', 250.00);

Query OK, 1 row affected (0.00 sec)

mysql>

mysql> -- Commit the transaction

mysql> COMMIT;

Query OK, 0 rows affected (0.01 sec)

mysql>

mysql> -- Rollback example

mysql> DELETE FROM Payments WHERE PaymentID = 3;

Query OK, 1 row affected (0.01 sec)

mysql> ROLLBACK;

Query OK, 0 rows affected (0.00 sec)

**VDL (View Definition Language) is a part of SQL used to define and manage views in a database. A view is a virtual table that is derived from one or more underlying tables or queries but does not physically store data. VDL allows users to create, modify, and delete views using statements like CREATE VIEW, ALTER VIEW, and DROP VIEW. Views are often used to simplify complex queries, present data in a specific format, or restrict access to certain data by providing a filtered view of a table. They enable abstraction and improve security by allowing users to interact with data without exposing the actual tables. Through VDL, database administrators and developers can create reusable, queryable data representations that serve different purposes in the application while keeping the database schema clean and organized.**

mysql> CREATE VIEW FilteredProfiles AS

-> SELECT FirstName, LastName, Location FROM Users

-> JOIN Profiles ON Users.UserID = Profiles.UserID

-> WHERE Location = 'New York';

Query OK, 0 rows affected (0.03 sec)

mysql> CREATE VIEW UserPayments AS

-> SELECT U.FirstName, P.Amount FROM Users U

-> JOIN Payments P ON U.UserID = P.UserID

-> WHERE P.Amount > (SELECT AVG(Amount) FROM Payments);

Query OK, 0 rows affected (0.01 sec)

mysql> CREATE VIEW JoinedView AS

-> SELECT U.FirstName, M.MessageText FROM Users U

-> JOIN Messages M ON U.UserID = M.SenderID;

Query OK, 0 rows affected (0.02 sec)

mysql> UPDATE UserProfile SET FirstName = 'Johnny' WHERE UserID = 1;

Query OK, 1 row affected (0.01 sec)

Rows matched: 1 Changed: 1 Warnings: 0

mysql> DELETE FROM Messages WHERE SenderID = 1;

Query OK, 1 row affected (0.01 sec)

mysql> SELECT \* FROM JoinedView;

Empty set (0.00 sec)

mysql> INSERT INTO FilteredProfiles (FirstName, LastName, Location) VALUES ('Alex', 'Smith', 'New York');

ERROR 1393 (HY000): Can not modify more than one base table through a join view 'matrimonyservice.filteredprofiles'

mysql> UPDATE FilteredProfiles SET Location = 'California' WHERE FirstName = 'Alex';

Query OK, 0 rows affected (0.00 sec)

Rows matched: 0 Changed: 0 Warnings: 0

**GitHub Link**