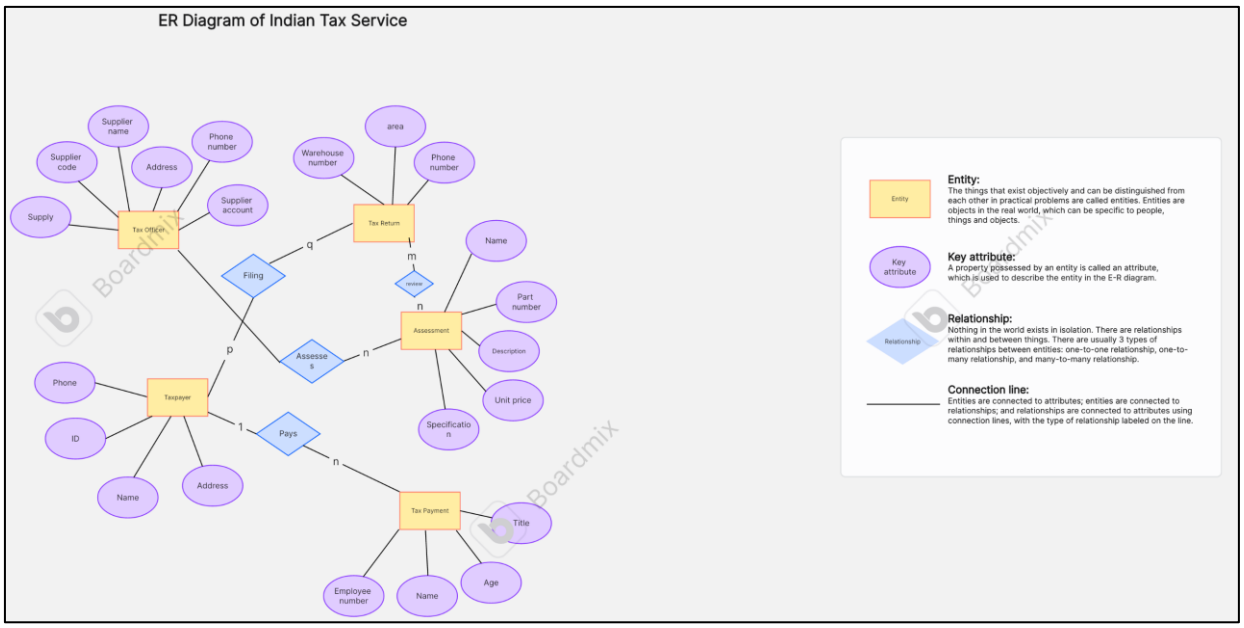
ER Diagram

The ER Diagram will model entities related to taxpayers, tax officers, tax payments, tax returns, and assessments. Below are the main entities and relationships:

1. **Entities:**
    - **Taxpayer:** Represents individuals or businesses paying taxes.
        - Attributes: TaxpayerID (PK), Name, Address, Phone, Email, PAN_Number, Taxpayer_Type (Individual/Business), Income_Level
    - **Tax Officer:** Represents officers managing tax assessments.
        - Attributes: OfficerID (PK), Name, Designation, Office_Location
    - **Tax Return:** Represents the filing of tax returns by taxpayers.
        - Attributes: ReturnID (PK), TaxpayerID (FK), Filing_Year, Filing_Date, Total_Income, Tax_Paid, Return_Status
    - **Tax Payment:** Represents the payments made by taxpayers.
        - Attributes: PaymentID (PK), TaxpayerID (FK), Payment_Date, Payment_Amount, Payment_Mode
    - **Assessment:** Represents tax assessment by a tax officer.
        - Attributes: AssessmentID (PK), OfficerID (FK), ReturnID (FK), Assessment_Date, Assessment_Result (Cleared/Discrepancy), Amount_Declared, Amount_Accepted
2. **Relationships:**
    - **Files:** Relationship between Taxpayer and Tax Return. Each taxpayer files multiple tax returns.
        - 1 Taxpayer → Many Tax Returns
    - **Makes:** Relationship between Taxpayer and Tax Payment. Each taxpayer can make multiple payments.
        - 1 Taxpayer → Many Tax Payments
    - **Assesses:** Relationship between Tax Officer and Assessment. A tax officer assesses multiple tax returns.
        - 1 Tax Officer → Many Assessments
    - **Reviews:** Relationship between Tax Return and Assessment. Each return can be reviewed by an assessment.
        - 1 Tax Return → 1 Assessment



Creating the tables

```
mysql> create database cia3;
Query OK, 1 row affected (0.05 sec)

mysql> use cia3;
Database changed
mysql> -- Creating tables for the ER diagram
mysql>
mysql> CREATE TABLE Taxpayer (
    ->     TaxpayerID INT PRIMARY KEY,
    ->     Name VARCHAR(100),
    ->     Address VARCHAR(255),
    ->     Phone VARCHAR(15),
    ->     Email VARCHAR(100),
    ->     PAN_Number VARCHAR(10) UNIQUE,
    ->     Taxpayer_Type VARCHAR(20),
    ->     Income_Level DECIMAL(10, 2)
    -> );
Query OK, 0 rows affected (0.05 sec)

mysql>
mysql> CREATE TABLE TaxOfficer (
    ->     OfficerID INT PRIMARY KEY,
    ->     Name VARCHAR(100),
    ->     Designation VARCHAR(50),
    ->     Office_Location VARCHAR(100)
    -> );
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> CREATE TABLE TaxReturn (
    ->     ReturnID INT PRIMARY KEY,
    ->     TaxpayerID INT,
    ->     Filing_Year YEAR,
    ->     Filing_Date DATE,
    ->     Total_Income DECIMAL(15, 2),
    ->     Tax_Paid DECIMAL(15, 2),
    ->     Return_Status VARCHAR(20),
    ->     FOREIGN KEY (TaxpayerID) REFERENCES Taxpayer(TaxpayerID)
    -> );
Query OK, 0 rows affected (0.03 sec)

mysql>
mysql> CREATE TABLE TaxPayment (
    ->     PaymentID INT PRIMARY KEY,
    ->     TaxpayerID INT,
    ->     Payment_Date DATE,
    ->     Payment_Amount DECIMAL(15, 2),
    ->     Payment_Mode VARCHAR(20),
    ->     FOREIGN KEY (TaxpayerID) REFERENCES Taxpayer(TaxpayerID)
    -> );
Query OK, 0 rows affected (0.03 sec)

mysql>
mysql> CREATE TABLE Assessment (
    ->     AssessmentID INT PRIMARY KEY,
    ->     OfficerID INT,
```

```
    ->     ReturnID INT,
    ->     Assessment_Date DATE,
    ->     Assessment_Result VARCHAR(50),
    ->     Amount_Declared DECIMAL(15, 2),
    ->     Amount_Accepted DECIMAL(15, 2),
    ->     FOREIGN KEY (OfficerID) REFERENCES TaxOfficer(OfficerID),
    ->     FOREIGN KEY (ReturnID) REFERENCES TaxReturn(ReturnID)
    -> );
Query OK, 0 rows affected (0.03 sec)
```

Inserting the data

```
mysql> -- Insert into Taxpayer table
mysql> INSERT INTO Taxpayer (TaxpayerID, Name, Address, Phone, Email,
PAN_Number, Taxpayer_Type, Income_Level)
    -> VALUES
    -> (1, 'John Doe', '123 Elm St', '9876543210', 'john@example.com',
'ABCDE1234F', 'Individual', 750000),
    -> (2, 'Jane Smith', '456 Oak St', '9876543220', 'jane@example.com',
'XYZDE6789K', 'Individual', 550000),
    -> (3, 'XYZ Corp', '789 Maple St', '9123456780', 'contact@xyz.com',
'LMNOP2345B', 'Business', 4500000),
    -> (4, 'Amit Verma', '321 Pine St', '8741234560', 'amit@example.com',
'QRSTU3456C', 'Individual', 1250000),
    -> (5, 'Shreya Sharma', '654 Cedar St', '9987654321',
'shreya@example.com', 'UVWXY5678D', 'Individual', 600000),
    -> (6, 'Global Traders', '890 Walnut St', '9988776655',
'info@globaltraders.com', 'ABCDF1234G', 'Business', 8000000),
    -> (7, 'Rohan Patel', '987 Birch St', '9871234567',
'rohan@example.com', 'OPQRW5678H', 'Individual', 750000),
    -> (8, 'Green Ventures', '654 Elm St', '9123123456',
'info@greenventures.com', 'JKLMN7890I', 'Business', 2500000),
    -> (9, 'Meera Rao', '111 Fir St', '9543217890', 'meera@example.com',
'QRSTY1234J', 'Individual', 850000),
    -> (10, 'Akash Gupta', '222 Palm St', '9874563210',
'akash@example.com', 'LMNOP3456K', 'Individual', 950000);
Query OK, 10 rows affected (0.01 sec)
Records: 10  Duplicates: 0  Warnings: 0

mysql>
mysql> -- Insert into TaxOfficer table
mysql> INSERT INTO TaxOfficer (OfficerID, Name, Designation,
Office_Location)
    -> VALUES
    -> (101, 'Alice Smith', 'Senior Officer', 'New Delhi'),
    -> (102, 'Rahul Singh', 'Junior Officer', 'Mumbai'),
    -> (103, 'Neha Kapoor', 'Chief Officer', 'Chennai'),
    -> (104, 'Rakesh Mehta', 'Officer', 'Bangalore'),
    -> (105, 'Priya Joshi', 'Officer', 'Hyderabad'),
    -> (106, 'Suresh Kumar', 'Senior Officer', 'Delhi'),
    -> (107, 'Vandana Rao', 'Junior Officer', 'Kolkata'),
    -> (108, 'Ravi Iyer', 'Officer', 'Pune'),
    -> (109, 'Anjali Mishra', 'Chief Officer', 'Ahmedabad'),
    -> (110, 'Deepak Sharma', 'Officer', 'Jaipur');
Query OK, 10 rows affected (0.00 sec)
Records: 10  Duplicates: 0  Warnings: 0
```

```
mysql>
mysql> -- Insert into TaxReturn table
mysql> INSERT INTO TaxReturn (ReturnID, TaxpayerID, Filing_Year,
Filing_Date, Total_Income, Tax_Paid, Return_Status)
    -> VALUES
    -> (1001, 1, 2023, '2024-06-10', 750000, 50000, 'Pending'),
    -> (1002, 2, 2023, '2024-06-15', 550000, 35000, 'Cleared'),
    -> (1003, 3, 2023, '2024-06-18', 4500000, 400000, 'Pending'),
    -> (1004, 4, 2023, '2024-06-20', 1250000, 100000, 'Cleared'),
    -> (1005, 5, 2023, '2024-06-22', 600000, 50000, 'Pending'),
    -> (1006, 6, 2023, '2024-06-24', 8000000, 700000, 'Cleared'),
    -> (1007, 7, 2023, '2024-06-26', 750000, 50000, 'Pending'),
    -> (1008, 8, 2023, '2024-06-28', 2500000, 200000, 'Cleared'),
    -> (1009, 9, 2023, '2024-06-30', 850000, 75000, 'Cleared'),
    -> (1010, 10, 2023, '2024-07-02', 950000, 80000, 'Pending');
Query OK, 10 rows affected (0.01 sec)
Records: 10  Duplicates: 0  Warnings: 0

mysql>
mysql> -- Insert into TaxPayment table
mysql> INSERT INTO TaxPayment (PaymentID, TaxpayerID, Payment_Date,
Payment_Amount, Payment_Mode)
    -> VALUES
    -> (2001, 1, '2024-05-15', 25000, 'Online'),
    -> (2002, 2, '2024-05-20', 20000, 'Cheque'),
    -> (2003, 3, '2024-05-25', 100000, 'Cash'),
    -> (2004, 4, '2024-05-30', 50000, 'Online'),
    -> (2005, 5, '2024-06-01', 30000, 'Cheque'),
    -> (2006, 6, '2024-06-05', 150000, 'Online'),
    -> (2007, 7, '2024-06-10', 25000, 'Cash'),
    -> (2008, 8, '2024-06-15', 50000, 'Online'),
    -> (2009, 9, '2024-06-20', 50000, 'Cheque'),
    -> (2010, 10, '2024-06-25', 30000, 'Cash');
Query OK, 10 rows affected (0.00 sec)
Records: 10  Duplicates: 0  Warnings: 0

mysql>
mysql> -- Insert into Assessment table
mysql> INSERT INTO Assessment (AssessmentID, OfficerID, ReturnID,
Assessment_Date, Assessment_Result, Amount_Declared, Amount_Accepted)
    -> VALUES
    -> (3001, 101, 1001, '2024-08-01', 'Cleared', 750000, 750000),
    -> (3002, 102, 1002, '2024-08-03', 'Cleared', 550000, 550000),
    -> (3003, 103, 1003, '2024-08-05', 'Discrepancy', 4500000, 4400000),
    -> (3004, 104, 1004, '2024-08-07', 'Cleared', 1250000, 1250000),
    -> (3005, 105, 1005, '2024-08-09', 'Discrepancy', 600000, 550000),
    -> (3006, 106, 1006, '2024-08-11', 'Cleared', 8000000, 8000000),
    -> (3007, 107, 1007, '2024-08-13', 'Cleared', 750000, 750000),
    -> (3008, 108, 1008, '2024-08-15', 'Cleared', 2500000, 2500000),
    -> (3009, 109, 1009, '2024-08-17', 'Cleared', 850000, 850000),
    -> (3010, 110, 1010, '2024-08-19', 'Discrepancy', 950000, 900000);
Query OK, 10 rows affected (0.01 sec)
Records: 10  Duplicates: 0  Warnings: 0
```

Group by and Having

The GROUP BY clause in SQL is used to group rows that have the same values in specified columns into summary rows, like "total sales per customer" or "average income per department." It is typically used in conjunction with aggregate functions (COUNT(), SUM(), AVG(), MAX(), MIN()) to perform operations on each group of data.

The HAVING clause is used to filter groups created by the GROUP BY clause, based on a specified condition. It's similar to the WHERE clause, but HAVING is used to apply conditions on aggregate functions (which can't be used in WHERE).

```
mysql> -- Example 1: Total tax paid by each taxpayer
mysql> SELECT TaxpayerID, SUM(Tax_Paid) AS TotalTaxPaid
    -> FROM TaxReturn
    -> GROUP BY TaxpayerID;
+------------+--------------+
| TaxpayerID | TotalTaxPaid |
+------------+--------------+
|          1 |     50000.00 |
|          2 |     35000.00 |
|          3 |    400000.00 |
|          4 |    100000.00 |
|          5 |     50000.00 |
|          6 |    700000.00 |
|          7 |     50000.00 |
|          8 |    200000.00 |
|          9 |     75000.00 |
|         10 |     80000.00 |
+------------+--------------+
10 rows in set (0.01 sec)

mysql>
mysql> -- Example 2: Number of payments made by each taxpayer
mysql> SELECT TaxpayerID, COUNT(*) AS PaymentCount
    -> FROM TaxPayment
    -> GROUP BY TaxpayerID
    -> HAVING COUNT(*) > 1;
Empty set (0.00 sec)

mysql>
mysql> -- Example 3: Average income of taxpayers grouped by taxpayer type
mysql> SELECT Taxpayer_Type, AVG(Income_Level) AS AvgIncome
    -> FROM Taxpayer
    -> GROUP BY Taxpayer_Type
    -> HAVING AVG(Income_Level) > 1000000;
+---------------+----------------+
| Taxpayer_Type | AvgIncome      |
+---------------+----------------+
| Business      | 5000000.000000 |
+---------------+----------------+
1 row in set (0.01 sec)
```

Joins

In SQL, a **JOIN** is used to combine rows from two or more tables based on a related column between them. Joins help in retrieving data from multiple tables by finding relationships between them, usually through **foreign keys.**

There are different types of joins depending on how you want to retrieve data and what kind of relationship exists between the tables.

INNER JOIN Returns only the matching rows from both tables.

LEFT JOIN Returns all rows from the left table, and matching rows from the right table (NULL if no match).

RIGHT JOIN Returns all rows from the right table, and matching rows from the left table (NULL if no match).

FULL OUTER JOIN Returns all rows where there is a match in either table (NULL if no match).

CROSS JOIN Returns the Cartesian product of both tables (all combinations).

```
mysql> -- Example 1: Join taxpayers and their tax returns
mysql> SELECT t.Name, tr.Filing_Year, tr.Total_Income, tr.Tax_Paid
    -> FROM Taxpayer t
    -> JOIN TaxReturn tr ON t.TaxpayerID = tr.TaxpayerID;
+----------------+-------------+--------------+-----------+
| Name           | Filing_Year | Total_Income | Tax_Paid  |
+----------------+-------------+--------------+-----------+
| John Doe       |        2023 |    750000.00 |  50000.00 |
| Jane Smith     |        2023 |    550000.00 |  35000.00 |
| XYZ Corp       |        2023 |   4500000.00 | 400000.00 |
| Amit Verma     |        2023 |   1250000.00 | 100000.00 |
| Shreya Sharma  |        2023 |    600000.00 |  50000.00 |
| Global Traders |        2023 |   8000000.00 | 700000.00 |
| Rohan Patel    |        2023 |    750000.00 |  50000.00 |
| Green Ventures |        2023 |   2500000.00 | 200000.00 |
| Meera Rao      |        2023 |    850000.00 |  75000.00 |
| Akash Gupta    |        2023 |    950000.00 |  80000.00 |
+----------------+-------------+--------------+-----------+
10 rows in set (0.00 sec)

mysql>
mysql> -- Example 2: Join tax officers and assessments they made
mysql> SELECT o.Name, a.Assessment_Result, a.Assessment_Date
    -> FROM TaxOfficer o
    -> JOIN Assessment a ON o.OfficerID = a.OfficerID;
+---------------+-------------------+-----------------+
| Name          | Assessment_Result | Assessment_Date |
+---------------+-------------------+-----------------+
| Alice Smith   | Cleared           | 2024-08-01      |
| Rahul Singh   | Cleared           | 2024-08-03      |
| Neha Kapoor   | Discrepancy       | 2024-08-05      |
| Rakesh Mehta  | Cleared           | 2024-08-07      |
| Priya Joshi   | Discrepancy       | 2024-08-09      |
| Suresh Kumar  | Cleared           | 2024-08-11      |
| Vandana Rao   | Cleared           | 2024-08-13      |
| Ravi Iyer     | Cleared           | 2024-08-15      |
| Anjali Mishra | Cleared           | 2024-08-17      |
| Deepak Sharma | Discrepancy       | 2024-08-19      |
+---------------+-------------------+-----------------+
10 rows in set (0.00 sec)

mysql>
mysql> -- Example 3: Join tax returns and their corresponding payments
```

```
mysql> SELECT tr.ReturnID, tp.Payment_Amount, tp.Payment_Mode
    -> FROM TaxReturn tr
    -> JOIN TaxPayment tp ON tr.TaxpayerID = tp.TaxpayerID;
+----------+----------------+--------------+
| ReturnID | Payment_Amount | Payment_Mode |
+----------+----------------+--------------+
|     1001 |       25000.00 | Online       |
|     1002 |       20000.00 | Cheque       |
|     1003 |      100000.00 | Cash         |
|     1004 |       50000.00 | Online       |
|     1005 |       30000.00 | Cheque       |
|     1006 |      150000.00 | Online       |
|     1007 |       25000.00 | Cash         |
|     1008 |       50000.00 | Online       |
|     1009 |       50000.00 | Cheque       |
|     1010 |       30000.00 | Cash         |
+----------+----------------+--------------+
10 rows in set (0.00 sec)
```

String functions

String functions in SQL are used to manipulate and handle string data (text). They help in performing operations like finding the length of a string, concatenating strings, extracting substrings, changing letter case, and more.

```
mysql> -- Example 1: Convert taxpayer names to uppercase
mysql> SELECT UPPER(Name) AS UpperCaseName FROM Taxpayer;
+----------------+
| UpperCaseName  |
+----------------+
| JOHN DOE       |
| JANE SMITH     |
| XYZ CORP       |
| AMIT VERMA     |
| SHREYA SHARMA  |
| GLOBAL TRADERS |
| ROHAN PATEL    |
| GREEN VENTURES |
| MEERA RAO      |
| AKASH GUPTA    |
+----------------+
10 rows in set (0.00 sec)

mysql>
mysql> -- Example 2: Get the length of each taxpayer's name
mysql> SELECT Name, LENGTH(Name) AS NameLength FROM Taxpayer;
+----------------+------------+
| Name           | NameLength |
+----------------+------------+
| John Doe       |          8 |
| Jane Smith     |         10 |
| XYZ Corp       |          8 |
| Amit Verma     |         10 |
| Shreya Sharma  |         13 |
```

```
| Global Traders |         14 |
| Rohan Patel    |         11 |
| Green Ventures |         14 |
| Meera Rao      |          9 |
| Akash Gupta    |         11 |
+----------------+------------+
10 rows in set (0.00 sec)

mysql>
mysql> -- Example 3: Concatenate name and PAN number of taxpayers
mysql> SELECT CONCAT(Name, ' - ', PAN_Number) AS TaxpayerInfo FROM
Taxpayer;
+-----------------------------+
| TaxpayerInfo                |
+-----------------------------+
| John Doe - ABCDE1234F       |
| Jane Smith - XYZDE6789K     |
| XYZ Corp - LMNOP2345B       |
| Amit Verma - QRSTU3456C     |
| Shreya Sharma - UVWXY5678D  |
| Global Traders - ABCDF1234G |
| Rohan Patel - OPQRW5678H    |
| Green Ventures - JKLMN7890I |
| Meera Rao - QRSTY1234J      |
| Akash Gupta - LMNOP3456K    |
+-----------------------------+
10 rows in set (0.00 sec)
```

Numeric functions

Numeric functions in SQL are used to perform operations on numerical data. These functions are essential for performing mathematical calculations, manipulating numeric values, and extracting useful data insights.

```
mysql> -- Example 1: Maximum and minimum tax paid
mysql> SELECT MAX(Tax_Paid) AS MaxTax, MIN(Tax_Paid) AS MinTax FROM
TaxReturn;
+-----------+----------+
| MaxTax    | MinTax   |
+-----------+----------+
| 700000.00 | 35000.00 |
+-----------+----------+
1 row in set (0.00 sec)

mysql>
mysql> -- Example 2: Average income level of taxpayers
mysql> SELECT AVG(Income_Level) AS AvgIncome FROM Taxpayer;
+----------------+
| AvgIncome      |
+----------------+
| 2070000.000000 |
+----------------+
1 row in set (0.00 sec)

mysql>
mysql> -- Example 3: Round total income values to nearest 1000
mysql> SELECT ROUND(Total_Income, -3) AS RoundedIncome FROM TaxReturn;
+---------------+
| RoundedIncome |
```

```
+---------------+
|        750000 |
|        550000 |
|       4500000 |
|       1250000 |
|        600000 |
|       8000000 |
|        750000 |
|       2500000 |
|        850000 |
|        950000 |
+---------------+
10 rows in set (0.00 sec)
```

Date-Time functions

SQL **date-time functions** are used to manipulate and extract information from date and time data types. These functions allow you to perform various operations such as getting the current date, extracting specific parts of a date, and performing date calculations.

```
mysql> -- Example 1: Calculate days between assessment and filing date
mysql> SELECT a.AssessmentID, DATEDIFF(a.Assessment_Date, tr.Filing_Date)
AS DaysBetween
    -> FROM Assessment a
    -> JOIN TaxReturn tr ON a.ReturnID = tr.ReturnID;
+--------------+-------------+
| AssessmentID | DaysBetween |
+--------------+-------------+
|         3001 |          52 |
|         3002 |          49 |
|         3003 |          48 |
|         3004 |          48 |
|         3005 |          48 |
|         3006 |          48 |
|         3007 |          48 |
|         3008 |          48 |
|         3009 |          48 |
|         3010 |          48 |
+--------------+-------------+
10 rows in set (0.01 sec)

mysql>
mysql> -- Example 2: Extract the month from payment date
mysql> SELECT PaymentID, MONTH(Payment_Date) AS PaymentMonth FROM
TaxPayment;
+-----------+--------------+
| PaymentID | PaymentMonth |
+-----------+--------------+
|      2001 |            5 |
|      2002 |            5 |
|      2003 |            5 |
|      2004 |            5 |
|      2005 |            6 |
|      2006 |            6 |
|      2007 |            6 |
|      2008 |            6 |
```

```
|      2009 |            6 |
|      2010 |            6 |
+-----------+--------------+
10 rows in set (0.00 sec)

mysql>
mysql> -- Example 3: Add 10 days to the filing date
mysql> SELECT ReturnID, Filing_Date, DATE_ADD(Filing_Date, INTERVAL 10
DAY) AS NewFilingDate FROM TaxReturn;
+----------+-------------+---------------+
| ReturnID | Filing_Date | NewFilingDate |
+----------+-------------+---------------+
|     1001 | 2024-06-10  | 2024-06-20    |
|     1002 | 2024-06-15  | 2024-06-25    |
|     1003 | 2024-06-18  | 2024-06-28    |
|     1004 | 2024-06-20  | 2024-06-30    |
|     1005 | 2024-06-22  | 2024-07-02    |
|     1006 | 2024-06-24  | 2024-07-04    |
|     1007 | 2024-06-26  | 2024-07-06    |
|     1008 | 2024-06-28  | 2024-07-08    |
|     1009 | 2024-06-30  | 2024-07-10    |
|     1010 | 2024-07-02  | 2024-07-12    |
+----------+-------------+---------------+
10 rows in set (0.00 sec)
```

Nested queries and subqueries

Nested Queries, also known as **Subqueries**, are queries embedded within another query. These subqueries are executed first, and their result is used by the outer query. They are useful when you need to retrieve data based on the result of another query.

Types of Subqueries

1. **Single-row Subqueries**: Returns only one row.
2. **Multiple-row Subqueries**: Returns more than one row.
3. **Correlated Subqueries**: Depends on the outer query to complete its execution.

```
mysql> -- Example 1: Taxpayers who paid above average tax
mysql> SELECT Name
    -> FROM Taxpayer
    -> WHERE TaxpayerID IN (SELECT TaxpayerID FROM TaxReturn WHERE
Tax_Paid > (SELECT AVG(Tax_Paid) FROM TaxReturn));
+----------------+
| Name           |
+----------------+
| XYZ Corp       |
| Global Traders |
| Green Ventures |
+----------------+
3 rows in set (0.00 sec)

mysql>
mysql> -- Example 2: Taxpayers who have filed more than one return
mysql> SELECT Name
    -> FROM Taxpayer
```

```
    -> WHERE TaxpayerID IN (SELECT TaxpayerID FROM TaxReturn GROUP BY
TaxpayerID HAVING COUNT(*) > 1);
Empty set (0.00 sec)

mysql>
mysql> -- Example 3: Officers who assessed returns with discrepancies
mysql> SELECT Name
    -> FROM TaxOfficer
    -> WHERE OfficerID IN (SELECT OfficerID FROM Assessment WHERE
Assessment_Result = 'Discrepancy');
+---------------+
| Name          |
+---------------+
| Neha Kapoor   |
| Priya Joshi   |
| Deepak Sharma |
+---------------+
3 rows in set (0.00 sec)
```

Cartesian Product

The **Cartesian Product**, also known as the **Cross Join**, is a type of join that returns all possible combinations of rows from two or more tables. When you perform a Cartesian product between two tables, every row in the first table is paired with every row in the second table.
This can result in a very large number of rows in the result set, especially if both tables have many rows.

```
mysql> -- Example 1: Cartesian product of taxpayers and officers
mysql> SELECT t.Name AS Taxpayer, o.Name AS Officer
    -> FROM Taxpayer t, TaxOfficer o;
+----------------+---------------+
| Taxpayer       | Officer       |
+----------------+---------------+
| Akash Gupta    | Alice Smith   |
| Meera Rao      | Alice Smith   |
| Green Ventures | Alice Smith   |
| Rohan Patel    | Alice Smith   |
| Global Traders | Alice Smith   |
| Shreya Sharma  | Alice Smith   |
| Amit Verma     | Alice Smith   |
| XYZ Corp       | Alice Smith   |
| Jane Smith     | Alice Smith   |
| John Doe       | Alice Smith   |
| Akash Gupta    | Rahul Singh   |
| Meera Rao      | Rahul Singh   |
| Green Ventures | Rahul Singh   |
| Rohan Patel    | Rahul Singh   |
| Global Traders | Rahul Singh   |
| Shreya Sharma  | Rahul Singh   |
| Amit Verma     | Rahul Singh   |
| XYZ Corp       | Rahul Singh   |
| Jane Smith     | Rahul Singh   |
| John Doe       | Rahul Singh   |
| Akash Gupta    | Neha Kapoor   |
```

Meera Rao	Neha Kapoor
Green Ventures	Neha Kapoor
Rohan Patel	Neha Kapoor
Global Traders	Neha Kapoor
Shreya Sharma	Neha Kapoor
Amit Verma	Neha Kapoor
XYZ Corp	Neha Kapoor
Jane Smith	Neha Kapoor
John Doe	Neha Kapoor
Akash Gupta	Rakesh Mehta
Meera Rao	Rakesh Mehta
Green Ventures	Rakesh Mehta
Rohan Patel	Rakesh Mehta
Global Traders	Rakesh Mehta
Shreya Sharma	Rakesh Mehta
Amit Verma	Rakesh Mehta
XYZ Corp	Rakesh Mehta
Jane Smith	Rakesh Mehta
John Doe	Rakesh Mehta
Akash Gupta	Priya Joshi
Meera Rao	Priya Joshi
Green Ventures	Priya Joshi
Rohan Patel	Priya Joshi
Global Traders	Priya Joshi
Shreya Sharma	Priya Joshi
Amit Verma	Priya Joshi
XYZ Corp	Priya Joshi
Jane Smith	Priya Joshi
John Doe	Priya Joshi
Akash Gupta	Suresh Kumar
Meera Rao	Suresh Kumar
Green Ventures	Suresh Kumar
Rohan Patel	Suresh Kumar
Global Traders	Suresh Kumar
Shreya Sharma	Suresh Kumar
Amit Verma	Suresh Kumar
XYZ Corp	Suresh Kumar
Jane Smith	Suresh Kumar
John Doe	Suresh Kumar
Akash Gupta	Vandana Rao
Meera Rao	Vandana Rao
Green Ventures	Vandana Rao
Rohan Patel	Vandana Rao
Global Traders	Vandana Rao
Shreya Sharma	Vandana Rao
Amit Verma	Vandana Rao
XYZ Corp	Vandana Rao
Jane Smith	Vandana Rao
John Doe	Vandana Rao
Akash Gupta	Ravi Iyer
Meera Rao	Ravi Iyer
Green Ventures	Ravi Iyer
Rohan Patel	Ravi Iyer
Global Traders	Ravi Iyer
Shreya Sharma	Ravi Iyer
Amit Verma	Ravi Iyer
XYZ Corp	Ravi Iyer
Jane Smith	Ravi Iyer

```
| John Doe        | Ravi Iyer     |
| Akash Gupta     | Anjali Mishra |
| Meera Rao       | Anjali Mishra |
| Green Ventures  | Anjali Mishra |
| Rohan Patel     | Anjali Mishra |
| Global Traders  | Anjali Mishra |
| Shreya Sharma   | Anjali Mishra |
| Amit Verma      | Anjali Mishra |
| XYZ Corp        | Anjali Mishra |
| Jane Smith      | Anjali Mishra |
| John Doe        | Anjali Mishra |
| Akash Gupta     | Deepak Sharma |
| Meera Rao       | Deepak Sharma |
| Green Ventures  | Deepak Sharma |
| Rohan Patel     | Deepak Sharma |
| Global Traders  | Deepak Sharma |
| Shreya Sharma   | Deepak Sharma |
| Amit Verma      | Deepak Sharma |
| XYZ Corp        | Deepak Sharma |
| Jane Smith      | Deepak Sharma |
| John Doe        | Deepak Sharma |
+-----------------+---------------+
100 rows in set (0.01 sec)

mysql>
mysql> -- Example 2: Cartesian product of taxpayers and returns
mysql> SELECT t.Name, tr.Filing_Year
    -> FROM Taxpayer t, TaxReturn tr;
+-----------------+-------------+
| Name            | Filing_Year |
+-----------------+-------------+
| Akash Gupta     |        2023 |
| Meera Rao       |        2023 |
| Green Ventures  |        2023 |
| Rohan Patel     |        2023 |
| Global Traders  |        2023 |
| Shreya Sharma   |        2023 |
| Amit Verma      |        2023 |
| XYZ Corp        |        2023 |
| Jane Smith      |        2023 |
| John Doe        |        2023 |
| Akash Gupta     |        2023 |
| Meera Rao       |        2023 |
| Green Ventures  |        2023 |
| Rohan Patel     |        2023 |
| Global Traders  |        2023 |
| Shreya Sharma   |        2023 |
| Amit Verma      |        2023 |
| XYZ Corp        |        2023 |
| Jane Smith      |        2023 |
| John Doe        |        2023 |
| Akash Gupta     |        2023 |
| Meera Rao       |        2023 |
| Green Ventures  |        2023 |
| Rohan Patel     |        2023 |
| Global Traders  |        2023 |
| Shreya Sharma   |        2023 |
| Amit Verma      |        2023 |
```

XYZ Corp	2023
Jane Smith	2023
John Doe	2023
Akash Gupta	2023
Meera Rao	2023
Green Ventures	2023
Rohan Patel	2023
Global Traders	2023
Shreya Sharma	2023
Amit Verma	2023
XYZ Corp	2023
Jane Smith	2023
John Doe	2023
Akash Gupta	2023
Meera Rao	2023
Green Ventures	2023
Rohan Patel	2023
Global Traders	2023
Shreya Sharma	2023
Amit Verma	2023
XYZ Corp	2023
Jane Smith	2023
John Doe	2023
Akash Gupta	2023
Meera Rao	2023
Green Ventures	2023
Rohan Patel	2023
Global Traders	2023
Shreya Sharma	2023
Amit Verma	2023
XYZ Corp	2023
Jane Smith	2023
John Doe	2023
Akash Gupta	2023
Meera Rao	2023
Green Ventures	2023
Rohan Patel	2023
Global Traders	2023
Shreya Sharma	2023
Amit Verma	2023
XYZ Corp	2023
Jane Smith	2023
John Doe	2023
Akash Gupta	2023
Meera Rao	2023
Green Ventures	2023
Rohan Patel	2023
Global Traders	2023
Shreya Sharma	2023
Amit Verma	2023
XYZ Corp	2023
Jane Smith	2023
John Doe	2023
Akash Gupta	2023
Meera Rao	2023
Green Ventures	2023
Rohan Patel	2023
Global Traders	2023

```
| Shreya Sharma  |        2023 |
| Amit Verma     |        2023 |
| XYZ Corp       |        2023 |
| Jane Smith     |        2023 |
| John Doe       |        2023 |
| Akash Gupta    |        2023 |
| Meera Rao      |        2023 |
| Green Ventures |        2023 |
| Rohan Patel    |        2023 |
| Global Traders |        2023 |
| Shreya Sharma  |        2023 |
| Amit Verma     |        2023 |
| XYZ Corp       |        2023 |
| Jane Smith     |        2023 |
| John Doe       |        2023 |
+----------------+-------------+
100 rows in set (0.00 sec)

mysql>
mysql> -- Example 3: Cartesian product of payments and assessments
mysql> SELECT tp.Payment_Amount, a.Assessment_Result
    -> FROM TaxPayment tp, Assessment a;
+----------------+-------------------+
| Payment_Amount | Assessment_Result |
+----------------+-------------------+
|       30000.00 | Cleared           |
|       50000.00 | Cleared           |
|       50000.00 | Cleared           |
|       25000.00 | Cleared           |
|      150000.00 | Cleared           |
|       30000.00 | Cleared           |
|       50000.00 | Cleared           |
|      100000.00 | Cleared           |
|       20000.00 | Cleared           |
|       25000.00 | Cleared           |
|       30000.00 | Cleared           |
|       50000.00 | Cleared           |
|       50000.00 | Cleared           |
|       25000.00 | Cleared           |
|      150000.00 | Cleared           |
|       30000.00 | Cleared           |
|       50000.00 | Cleared           |
|      100000.00 | Cleared           |
|       20000.00 | Cleared           |
|       25000.00 | Cleared           |
|       30000.00 | Discrepancy       |
|       50000.00 | Discrepancy       |
|       50000.00 | Discrepancy       |
|       25000.00 | Discrepancy       |
|      150000.00 | Discrepancy       |
|       30000.00 | Discrepancy       |
|       50000.00 | Discrepancy       |
|      100000.00 | Discrepancy       |
|       20000.00 | Discrepancy       |
|       25000.00 | Discrepancy       |
|       30000.00 | Cleared           |
|       50000.00 | Cleared           |
|       50000.00 | Cleared           |
```

25000.00	Cleared
150000.00	Cleared
30000.00	Cleared
50000.00	Cleared
100000.00	Cleared
20000.00	Cleared
25000.00	Cleared
30000.00	Discrepancy
50000.00	Discrepancy
50000.00	Discrepancy
25000.00	Discrepancy
150000.00	Discrepancy
30000.00	Discrepancy
50000.00	Discrepancy
100000.00	Discrepancy
20000.00	Discrepancy
25000.00	Discrepancy
30000.00	Cleared
50000.00	Cleared
50000.00	Cleared
25000.00	Cleared
150000.00	Cleared
30000.00	Cleared
50000.00	Cleared
100000.00	Cleared
20000.00	Cleared
25000.00	Cleared
30000.00	Cleared
50000.00	Cleared
50000.00	Cleared
25000.00	Cleared
150000.00	Cleared
30000.00	Cleared
50000.00	Cleared
100000.00	Cleared
20000.00	Cleared
25000.00	Cleared
30000.00	Cleared
50000.00	Cleared
50000.00	Cleared
25000.00	Cleared
150000.00	Cleared
30000.00	Cleared
50000.00	Cleared
100000.00	Cleared
20000.00	Cleared
25000.00	Cleared
30000.00	Cleared
50000.00	Cleared
50000.00	Cleared
25000.00	Cleared
150000.00	Cleared
30000.00	Cleared
50000.00	Cleared
100000.00	Cleared
20000.00	Cleared
25000.00	Cleared
30000.00	Discrepancy

```
|       50000.00 | Discrepancy       |
|       50000.00 | Discrepancy       |
|       25000.00 | Discrepancy       |
|      150000.00 | Discrepancy       |
|       30000.00 | Discrepancy       |
|       50000.00 | Discrepancy       |
|      100000.00 | Discrepancy       |
|       20000.00 | Discrepancy       |
|       25000.00 | Discrepancy       |
+----------------+-------------------+
100 rows in set (0.00 sec)
```

Division

Division in SQL is a relational operation that is used to find a set of values from one table that are associated with all values in another table. This operation is often required in scenarios where you want to find items that meet a certain condition for all values in a subset.

```
mysql> -- Example 1: Taxpayers who made payments in all modes
mysql> SELECT TaxpayerID
    -> FROM TaxPayment tp1
    -> WHERE NOT EXISTS (SELECT pm.Payment_Mode FROM (SELECT DISTINCT
Payment_Mode FROM TaxPayment) pm WHERE NOT EXISTS(SELECT 1 FROM
TaxPayment tp2 WHERE tp1.TaxpayerID = tp2.TaxpayerID AND tp2.Payment_Mode
= pm.Payment_Mode));
Empty set (0.01 sec)

mysql>
mysql> -- Example 2: Taxpayers who filed returns for all years
mysql> SELECT TaxpayerID
    -> FROM TaxReturn tr1
    -> WHERE NOT EXISTS (SELECT fy.Filing_Year FROM (SELECT DISTINCT
Filing_Year FROM TaxReturn) fy WHERE NOT EXISTS (SELECT 1 FROM TaxReturn
tr2 WHERE tr1.TaxpayerID = tr2.TaxpayerID AND tr2.Filing_Year =
fy.Filing_Year));
+------------+
| TaxpayerID |
+------------+
|          1 |
|          2 |
|          3 |
|          4 |
|          5 |
|          6 |
|          7 |
|          8 |
|          9 |
|         10 |
+------------+
10 rows in set (0.00 sec)

mysql>
mysql> -- Example 3: Tax officers who assessed all types of results
mysql> SELECT OfficerID
    -> FROM Assessment a1
```

```
    -> WHERE NOT EXISTS (SELECT ar.Assessment_Result FROM (SELECT
DISTINCT Assessment_Result FROM Assessment) ar WHERE NOT EXISTS (SELECT 1
FROM Assessment a2 WHERE a1.OfficerID = a2.OfficerID AND
a2.Assessment_Result = ar.Assessment_Result));
Empty set (0.00 sec)
```

Rename

The **RENAME** operation in SQL is used to change the name of a table or a column within a table. While SQL does not have a specific command called RENAME, it achieves this functionality through the ALTER TABLE statement for tables and columns.

```
mysql> -- Example 1: Rename Taxpayer table to Taxpayer_Details
mysql> ALTER TABLE Taxpayer RENAME TO Taxpayer_Details;
Query OK, 0 rows affected (0.03 sec)

mysql>
mysql> -- Example 2: Rename Assessment table to Tax_Assessment
mysql> ALTER TABLE Assessment RENAME TO Tax_Assessment;
Query OK, 0 rows affected (0.01 sec)

mysql>
mysql> -- Example 3: Rename TaxPayment to Payment_Details
mysql> ALTER TABLE TaxPayment RENAME TO Payment_Details;
Query OK, 0 rows affected (0.01 sec)
```

Transaction Control Language

Transaction Control Language (TCL) is a subset of SQL that is used to manage transactions in a database. Transactions are sequences of operations performed as a single logical unit of work. TCL commands ensure the integrity of data by allowing you to commit or roll back transactions.

Common TCL Commands

1. **COMMIT**
2. **ROLLBACK**
3. **SAVEPOINT**
4. **SET TRANSACTION**

```
mysql> -- Example 1: Using COMMIT after an insertion in the renamed
Payment_Details table
mysql> BEGIN TRANSACTION;
ERROR 1064 (42000): You have an error in your SQL syntax; check the
manual that corresponds to your MySQL server version for the right syntax
to use near 'TRANSACTION' at line 1
mysql> INSERT INTO Payment_Details (PaymentID, TaxpayerID, Payment_Date,
Payment_Amount, Payment_Mode)
    -> VALUES (2011, 10, '2024-07-15', 40000, 'Online');
Query OK, 1 row affected (0.00 sec)

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

mysql>
mysql> -- Example 2: Using ROLLBACK to undo a transaction in the renamed
Taxpayer_Details table
```

```
mysql> BEGIN TRANSACTION;
ERROR 1064 (42000): You have an error in your SQL syntax; check the
manual that corresponds to your MySQL server version for the right syntax
to use near 'TRANSACTION' at line 1
mysql> UPDATE Taxpayer_Details SET Income_Level = Income_Level + 100000
WHERE TaxpayerID = 1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

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

mysql>
mysql> -- Example 3: Using SAVEPOINT in the renamed Tax_Assessment and
Taxpayer_Details tables
mysql> BEGIN TRANSACTION;
ERROR 1064 (42000): You have an error in your SQL syntax; check the
manual that corresponds to your MySQL server version for the right syntax
to use near 'TRANSACTION' at line 1
mysql> INSERT INTO Tax_Assessment (AssessmentID, OfficerID, ReturnID,
Assessment_Date, Assessment_Result, Amount_Declared, Amount_Accepted)
    -> VALUES (3011, 101, 1001, '2024-09-01', 'Cleared', 750000, 750000);
Query OK, 1 row affected (0.00 sec)

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

mysql> UPDATE Taxpayer_Details SET Income_Level = 850000 WHERE TaxpayerID
= 1;
Query OK, 0 rows affected (0.00 sec)
Rows matched: 1  Changed: 0  Warnings: 0

mysql> ROLLBACK TO before_update;
ERROR 1305 (42000): SAVEPOINT before_update does not exist
mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
```

View Definition Language

View Definition Language (VDL) is a subset of SQL that is primarily concerned with defining and managing views within a database. A view is a virtual table that is based on the result of a SQL query. It does not store data physically; instead, it dynamically pulls data from the underlying tables whenever it is accessed.

```
mysql> CREATE VIEW HighIncomeTaxpayers AS
    -> SELECT TaxpayerID, Name, Income_Level
    -> FROM Taxpayer_Details
    -> WHERE Income_Level > 1000000;
Query OK, 0 rows affected (0.03 sec)

mysql> select * from HighIncomeTaxpayers;
+------------+----------------+--------------+
| TaxpayerID | Name           | Income_Level |
+------------+----------------+--------------+
|          3 | XYZ Corp       |   4500000.00 |
|          4 | Amit Verma     |   1250000.00 |
|          6 | Global Traders |   8000000.00 |
```

```
|          8 | Green Ventures |   2500000.00 |
+------------+----------------+--------------+
4 rows in set (0.01 sec)

mysql> CREATE VIEW ClearedTaxReturns AS
    -> SELECT ReturnID, TaxpayerID, Total_Income, Tax_Paid
    -> FROM TaxReturn
    -> WHERE Return_Status = 'Cleared';
Query OK, 0 rows affected (0.01 sec)

mysql> select * from ClearedTaxReturns;
+----------+------------+--------------+-----------+
| ReturnID | TaxpayerID | Total_Income | Tax_Paid  |
+----------+------------+--------------+-----------+
|     1002 |          2 |    550000.00 |  35000.00 |
|     1004 |          4 |   1250000.00 | 100000.00 |
|     1006 |          6 |   8000000.00 | 700000.00 |
|     1008 |          8 |   2500000.00 | 200000.00 |
|     1009 |          9 |    850000.00 |  75000.00 |
|     1011 |          1 |    900000.00 |  60000.00 |
+----------+------------+--------------+-----------+
6 rows in set (0.09 sec)

mysql> CREATE VIEW OfficerAssessments AS
    -> SELECT o.Name AS OfficerName, a.Assessment_Date, t.Name AS
TaxpayerName, a.Assessment_Result
    -> FROM Tax_Assessment a
    -> JOIN TaxOfficer o ON a.OfficerID = o.OfficerID
    -> JOIN Taxpayer_Details t ON t.TaxpayerID = a.ReturnID;
Query OK, 0 rows affected (0.07 sec)

mysql> select * from OfficerAssessments;
Empty set (0.01 sec)
```

Data Definition Language (DDL)

DDL is a subset of SQL used to define and manage all database objects, including tables, schemas, indexes, and views. The primary purpose of DDL commands is to create, modify, and delete these structures. DDL commands do not manipulate the data itself but instead deal with the schema and structure of the database. Common DDL commands include:

- **CREATE**: Used to create new database objects, such as tables and indexes.
- **ALTER**: Used to modify existing database objects, allowing changes to structure, constraints, or properties.
- **DROP**: Used to delete database objects, permanently removing them from the database.
- **TRUNCATE**: Used to remove all records from a table without removing the table structure.

Data Manipulation Language (DML)

DML is a subset of SQL used for manipulating and retrieving data within the database. DML commands allow users to insert, update, delete, and retrieve data. These commands directly affect the data stored in database tables. Common DML commands include:

- **SELECT**: Used to query and retrieve data from one or more tables.
- **INSERT**: Used to add new records to a table.
- **UPDATE**: Used to modify existing records in a table.
- **DELETE**: Used to remove records from a table.

Transaction Control Language (TCL)

TCL is a subset of SQL used to manage transactions in a database. Transactions are sequences of operations that are executed as a single logical unit. TCL commands ensure data integrity by allowing users to commit or roll back changes made during a transaction. Common TCL commands include:

- **COMMIT**: Used to save all changes made during the current transaction permanently to the database.

- **ROLLBACK**: Used to undo changes made during the current transaction, reverting the database to its last committed state.
- **SAVEPOINT**: Used to create a point within a transaction that can be rolled back to without affecting the entire transaction.
- **SET TRANSACTION**: Used to configure transaction properties, such as isolation levels.

View Definition Language (VDL)

VDL is a subset of SQL focused on defining and managing views in a database. A view is a virtual table that represents a specific subset of data from one or more tables based on a query. VDL commands allow users to create, modify, and delete views, providing a simplified and secure way to access data. Common VDL commands include:

- **CREATE VIEW**: Used to define a new view based on a query.
- **ALTER VIEW**: Used to modify an existing view's definition.
- **DROP VIEW**: Used to remove a view from the database.

Github account link - https://github.com/N3thannn