

## Part 1.0: Implementation of DBs and GCP & Terminal Connection Showing

```
CLOUD SHELL
Terminal (fa24-cs411) x + v

| mysql |
| performance_schema |
| sys |
+-----+
5 rows in set (0.01 sec)

mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| cs411 |
| information_schema |
| mysql |
| performance_schema |
| sys |
+-----+
5 rows in set (0.00 sec)

mysql> 
```

```
CLOUD SHELL
Terminal (fa24-cs411) x + v

You can turn off this feature to get a c

Database changed
mysql> SHOW Tables
-> ;
+-----+
| Tables_in_cs411 |
+-----+
| CurrencyExchange |
| Inflation |
| Party |
| PartyMember |
| Spending |
| Transaction |
| User |
+-----+
7 rows in set (0.01 sec)
```

## Part 1.1 + 1.2: DDL for Tables

DDL for creating six tables in our Database.

### 1. User Table

```
CREATE TABLE User (
  UserId INT PRIMARY KEY,
  Name VARCHAR(100) NOT NULL,
  PhoneNumber VARCHAR(15) UNIQUE,
  DateCreated DATE NOT NULL
);
```

### 2. Group (Party) Table

```
CREATE TABLE Party (
  GroupId INT PRIMARY KEY,
  GroupName VARCHAR(100) NOT NULL,
  CreatedBy INT NOT NULL,
  CreateAt DATE NOT NULL,
  DeleteAt DATE,
  FOREIGN KEY (CreatedBy) REFERENCES User(UserId) ON DELETE CASCADE
);
```

### 3. Transaction Table

```
CREATE TABLE Transaction (  
  TransactionId INT PRIMARY KEY,  
  GroupId INT NOT NULL,  
  Amount DECIMAL(10, 2) NOT NULL,  
  CurrencyType VARCHAR(3) NOT NULL,  
  Date DATE NOT NULL,  
  SenderId INT NOT NULL,  
  ReceiverId INT NOT NULL,  
  FOREIGN KEY (GroupId) REFERENCES Party(GroupId) ON DELETE CASCADE,  
  FOREIGN KEY (SenderId) REFERENCES User(UserId) ON DELETE CASCADE,  
  FOREIGN KEY (ReceiverId) REFERENCES User(UserId) ON DELETE CASCADE  
);
```

### 4. Spending Table

```
CREATE TABLE Spending (  
  SpendingId INT PRIMARY KEY,  
  CurrencyType VARCHAR(3) NOT NULL,  
  Category VARCHAR(50) NOT NULL,  
  Amount DECIMAL(10, 2) NOT NULL  
);
```

### 5. CurrencyExchange Table

```
CREATE TABLE CurrencyExchange (  
  Id INT AUTO_INCREMENT PRIMARY KEY,  
  Timestamp TIMESTAMP CURRENT_TIMESTAMP,  
  SourceCurrency VARCHAR(3) NOT NULL,  
  TargetCurrency VARCHAR(3) NOT NULL,  
  Rate DECIMAL(10, 6) NOT NULL  
);
```

### 6. Inflation Table

```
CREATE TABLE Inflation (  
  Year INT NOT NULL,  
  Month INT NOT NULL,  
  Rate REAL NOT NULL,  
  PRIMARY KEY (Year, Month)  
);
```

### Part 1.3: Data Insertion and Showing

Three of our six tables contain over 1000 rows of data after insertions.

```
mysql> SELECT COUNT(*) FROM User;
+-----+
| COUNT(*) |
+-----+
|      1000 |
+-----+
1 row in set (0.01 sec)
```

```
mysql> SELECT COUNT(*) FROM CurrencyExchange;
+-----+
| COUNT(*) |
+-----+
|      6865 |
+-----+
1 row in set (0.01 sec)
```

```
mysql> SELECT COUNT(*) FROM Inflation;
+-----+
| COUNT(*) |
+-----+
|    145533 |
+-----+
1 row in set (0.04 sec)
```

```
mysql> SELECT COUNT(*) FROM Transaction;
+-----+
| COUNT(*) |
+-----+
|        15 |
+-----+
1 row in set (0.01 sec)
```

## Part 1.4: Advanced Queries for applications

### 1. (Join Multiple Relations)

**Query:** List Group Expenses Summary with Total Spending Per User

**Concepts:** Join multiple relations, Aggregation via GROUP BY, This query retrieves a summary of the total amount spent by each user within a specific group.

```
SQL Query 1: Join Multiple Relations

SELECT
  u.UserId,
  u.Name AS UserName,
  g.GroupName,
  SUM(t.Amount) AS TotalAmountSpent
FROM
  Transaction t
JOIN
  User u ON t.SenderId = u.UserId
JOIN
  Party g ON t.GroupId = g.GroupId
WHERE
  g.GroupId = 1
GROUP BY
  u.UserId, u.Name, g.GroupName;
```

#### Output

UserId	UserName	GroupName	TotalAmountSpent
1	Kimmie Woodham	Trip to Japan	850.00
2	Reeba Malatalant	Trip to Japan	1100.00
3	Cherish Holton	Trip to Japan	1350.00
4	Jermaine Sands	Trip to Japan	1600.00
5	Brittne Braffington	Trip to Japan	1850.00
37	Andrea Faithfull	Trip to Japan	123.34
48	Thibaut Groome	Trip to Japan	497.23
98	Richard Dockerty	Trip to Japan	234.29
122	Gabi Ortet	Trip to Japan	944.23
125	Jocelin Ashpital	Trip to Japan	124.93
234	Gayelord Pascow	Trip to Japan	425.85
245	Layne Casserley	Trip to Japan	123.00
336	Dorthy Schulken	Trip to Japan	506.52
392	Yolanthe Bowmer	Trip to Japan	123.12
442	Farrell Gripton	Trip to Japan	345.97

15 rows in set (0.00 sec)

### 2.(Subquery + Set Operations)

**Query:** Find Users with No Activity in Any

**Concepts:** This query identifies users who are registered but have not participated in any transactions.

#### Output:

```
SQL Query 2: Subqueries/ Set Operations x

SELECT UserId, Name, DateCreated
FROM User
WHERE UserId NOT IN
(
  SELECT SenderId
  FROM Transaction
  UNION
  SELECT ReceiverId
  FROM Transaction
) AND Name like '%C%' AND DateCreated LIKE '2024%';
```

```
mysql> SELECT UserId, Name
-> FROM User
-> WHERE UserId NOT IN
-> (
->   SELECT SenderId
->   FROM Transaction
->   UNION
->   SELECT ReceiverId
->   FROM Transaction
-> ) limit 15;

+----+-----+
| UserId | Name                |
+----+-----+
| 6      | Kelley Labone       |
| 7      | Brett Rainsden     |
| 8      | Marcellus Gendrich |
| 9      | Arnie Pummery      |
| 10     | Judie Frudd        |
| 11     | Webb Ewenson       |
| 12     | Clarence Inker     |
| 13     | Amanda Parry       |
| 14     | Rodolph Bielfeld   |
| 15     | Winifred Gillett   |
| 16     | Kearney Boyde      |
| 17     | Sandy Bauman       |
| 18     | Marika Isaaksohn   |
| 19     | Gregorius Airey    |
| 20     | Virgie Albery      |
+----+-----+
```

### 3. (Set Operation)

**Query:** Find users who are not involved in any transactions:

**Output**

```
SELECT u.PhoneNumber, u.DateCreated
FROM User u
LEFT JOIN Transaction t1 ON u.PhoneNumber = t1.SenderId
LEFT JOIN Transaction t2 ON u.PhoneNumber = t2.ReceiverId
WHERE t1.SenderId IS NULL AND t2.ReceiverId IS NULL AND u.PhoneNumber LIKE '1%'
AND u.DateCreated LIKE '2024%';
```

```
mysql> SELECT PhoneNumber
+-----+
| PhoneNumber |
+-----+
| 1002634196 |
| 1003798908 |
| 1009209719 |
| 1027508872 |
| 1045859738 |
| 1047967575 |
| 1051643186 |
| 1054965466 |
| 1066883474 |
| 1071962448 |
| 1079461173 |
| 1089622005 |
| 1091332112 |
| 1091501256 |
| 1107147455 |
+-----+
15 rows in set (0.00 sec)
```

### 4.(Subqueries +Join + Aggregation)

**Query:** Find the latest exchange rate for each currency pair:

**Output**

```
SELECT
    ce.SourceCurrency,
    ce.TargetCurrency,
    ce.Rate,
    ce.Timestamp
FROM
    CurrencyExchange ce
INNER JOIN (
    SELECT
        SourceCurrency,
        TargetCurrency,
        MAX(Timestamp) AS MaxTimestamp
    FROM
        CurrencyExchange
    GROUP BY
        SourceCurrency, TargetCurrency
) latest
ON
    ce.SourceCurrency = latest.SourceCurrency
    AND ce.TargetCurrency = latest.TargetCurrency
    AND ce.Timestamp = latest.MaxTimestamp;
```

```
mysql> SELECT
    ce.SourceCurrency, ce.TargetCurrency, ce.Rate,
    MAX(Timestamp) AS MaxTimestamp FROM
    CurrencyExchange
    AND ce.TargetCurrency = latest.TargetCurrency AND ce.Timestamp = latest.MaxTimestamp;
+-----+-----+-----+-----+
| SourceCurrency | TargetCurrency | Rate | Timestamp |
+-----+-----+-----+-----+
| USD | AED | 3.6725 | 2024-10-30 18:08:32 |
| USD | AFN | 67.192 | 2024-10-30 18:08:32 |
| USD | ALL | 91.2453 | 2024-10-30 18:08:32 |
| USD | AMD | 387.042 | 2024-10-30 18:08:32 |
| USD | ANG | 1.79 | 2024-10-30 18:08:33 |
| USD | AOA | 916.734 | 2024-10-30 18:08:33 |
| USD | ARS | 989.25 | 2024-10-30 18:08:33 |
| USD | AUD | 1.5242 | 2024-10-30 18:08:33 |
| USD | AWG | 1.79 | 2024-10-30 18:08:33 |
| USD | AZN | 1.7001 | 2024-10-30 18:08:33 |
| USD | BAM | 1.8095 | 2024-10-30 18:08:33 |
| USD | BBD | 2 | 2024-10-30 18:08:33 |
| USD | BDT | 119.5898 | 2024-10-30 18:08:33 |
| USD | BGN | 1.8102 | 2024-10-30 18:08:33 |
| USD | BHD | 0.376 | 2024-10-30 18:08:33 |
| USD | BIF | 2911.5126 | 2024-10-30 18:08:33 |
| USD | BMD | 1 | 2024-10-30 18:08:33 |
| USD | BND | 1.3245 | 2024-10-30 18:08:33 |
| USD | BOB | 6.9295 | 2024-10-30 18:08:33 |
| USD | USD | 1 | 2024-10-30 18:08:32 |
+-----+-----+-----+-----+
20 rows in set (0.03 sec)

mysql>
```

## Part 2.1: Advanced Queries Indexing

### Part 1 Query # 4: Find the latest exchange rate for each currency pair

#### Before applying indexing: COST = 32498.25

```
-----+
| -> Nested loop inner join (cost=32498.25 rows=0) (actual time=49.556..67.316 rows=6804 loops=1)
|   -> Table scan on ce (cost=1273.25 rows=12490) (actual time=0.063..4.546 rows=13692 loops=1)
|   -> Covering index lookup on latest using <auto_key0> (SourceCurrency=ce.SourceCurrency, TargetCurrency=ce.TargetCurrency, MaxTimestamp=ce.`Timestamp`) (actual time=0.004..0.004 rows=0 loops=13692)
|     -> Materialize (cost=0.00..0.00 rows=0) (actual time=35.236..35.236 rows=6804 loops=1)
|       -> Table scan on <temporary> (actual time=21.652..22.697 rows=6804 loops=1)
|         -> Aggregate using temporary table (actual time=21.649..21.649 rows=6804 loops=1)
|           -> Table scan on CurrencyExchange (cost=1273.25 rows=12490) (actual time=0.024..4.360 rows=13692 loops=1)
|
|-----+
1 row in set (0.07 sec)
```

#### Index Set 1: Applying index on (Timestamp, SourceCurrency, TargetCurrency); **COST = 5779.12**

CREATE INDEX idx\_currency\_timestamp ON CurrencyExchange (Timestamp, SourceCurrency, TargetCurrency);

```
-----+
| -> Nested loop inner join (cost=5779.12 rows=1249) (actual time=26.376..57.165 rows=6804 loops=1)
|   -> Filter: (latest.MaxTimestamp is not null) (cost=0.11..1407.62 rows=12490) (actual time=26.317..28.249 rows=6804 loops=1)
|     -> Table scan on latest (cost=2.50..2.50 rows=0) (actual time=26.314..27.622 rows=6804 loops=1)
|       -> Materialize (cost=0.00..0.00 rows=0) (actual time=26.312..26.312 rows=6804 loops=1)
|         -> Table scan on <temporary> (actual time=21.662..22.571 rows=6804 loops=1)
|           -> Aggregate using temporary table (actual time=21.658..21.658 rows=6804 loops=1)
|             -> Table scan on CurrencyExchange (cost=1273.25 rows=12490) (actual time=0.046..4.300 rows=13692 loops=1)
|               -> Filter: (ce.SourceCurrency = latest.SourceCurrency) (cost=0.25 rows=0.1) (actual time=0.003..0.004 rows=1 loops=6804)
|                 -> Index lookup on ce using idx_currency_timestamp (Timestamp=latest.MaxTimestamp, TargetCurrency=latest.TargetCurrency) (cost=0.25 rows=1) (actual time=0.003..0.004 rows=1 loops=6804)
|
|-----+
1 row in set (0.07 sec)
```

#### Index Set 2: Applying index on (SourceCurrency, TargetCurrency); **COST = 11954.57**

CREATE INDEX idx\_currency\_timestamp ON CurrencyExchange (SourceCurrency, TargetCurrency);

```
-----+
| -> Nested loop inner join (cost=11954.57 rows=2293) (actual time=30.409..91.727 rows=6804 loops=1)
|   -> Table scan on latest (cost=3771.26..3929.88 rows=12490) (actual time=30.365..32.400 rows=6804 loops=1)
|     -> Materialize (cost=3771.26..3929.88 rows=12490) (actual time=30.361..30.361 rows=6804 loops=1)
|       -> Group aggregate: max(CurrencyExchange.`Timestamp`) (cost=2522.25 rows=12490) (actual time=0.287..25.230 rows=6804 loops=1)
|         -> Index scan on CurrencyExchange using idx_currency_timestamp (cost=1273.25 rows=12490) (actual time=0.268..18.629 rows=13692 loops=1)
|           -> Filter: (ce.`Timestamp` = latest.MaxTimestamp) (cost=0.46 rows=0.2) (actual time=0.008..0.009 rows=1 loops=6804)
|             -> Index lookup on ce using idx_currency_timestamp (SourceCurrency=latest.SourceCurrency, TargetCurrency=latest.TargetCurrency) (cost=0.46 rows=2) (actual time=0.007..0.008 rows=2 loops=6804)
|
|-----+
1 row in set (0.07 sec)
```

#### Index Set 3: Applying index on (TimeStamp, TargetCurrency); **COST = 5779.12**

CREATE INDEX idx\_currency\_timestamp ON CurrencyExchange (Timestamp, TargetCurrency);

```
-----+
| -> Nested loop inner join (cost=5779.12 rows=1249) (actual time=26.376..57.165 rows=6804 loops=1)
|   -> Filter: (latest.MaxTimestamp is not null) (cost=0.11..1407.62 rows=12490) (actual time=26.317..28.249 rows=6804 loops=1)
|     -> Table scan on latest (cost=2.50..2.50 rows=0) (actual time=26.314..27.622 rows=6804 loops=1)
|       -> Materialize (cost=0.00..0.00 rows=0) (actual time=26.312..26.312 rows=6804 loops=1)
|         -> Table scan on <temporary> (actual time=21.662..22.571 rows=6804 loops=1)
|           -> Aggregate using temporary table (actual time=21.658..21.658 rows=6804 loops=1)
|             -> Table scan on CurrencyExchange (cost=1273.25 rows=12490) (actual time=0.046..4.300 rows=13692 loops=1)
|               -> Filter: (ce.SourceCurrency = latest.SourceCurrency) (cost=0.25 rows=0.1) (actual time=0.003..0.004 rows=1 loops=6804)
|                 -> Index lookup on ce using idx_currency_timestamp (Timestamp=latest.MaxTimestamp, TargetCurrency=latest.TargetCurrency) (cost=0.25 rows=1) (actual time=0.003..0.004 rows=1 loops=6804)
|
|-----+
1 row in set (0.07 sec)
```

**Part1: Query # 3:****Before applying indexing: COST = 128.43**

```
-----+
| -> Filter: (t2.ReceiverId is null) (cost=128.43 rows=111) (actual time=0.211..1.407 rows=99 loops=1)
|   -> Nested loop antijoin (cost=128.43 rows=111) (actual time=0.211..1.396 rows=99 loops=1)
|     -> Filter: (t1.SenderId is null) (cost=108.04 rows=37) (actual time=0.200..1.161 rows=99 loops=1)
|       -> Nested loop antijoin (cost=108.04 rows=37) (actual time=0.199..1.149 rows=99 loops=1)
|         -> Filter: ((u.PhoneNumber like '1%') and (u.DateCreated like '2024%')) (cost=101.25 rows=12) (actual time=0.176..0.856 rows=99 loops=1)
|           -> Table scan on u (cost=101.25 rows=1000) (actual time=0.066..0.509 rows=1000 loops=1)
|             -> Filter: (cast(u.PhoneNumber as double) = cast(t1.SenderId as double)) (cost=0.27 rows=3) (actual time=0.003..0.003 rows=0 loops=99)
|               -> Covering index lookup on t1 using idx_transaction_senderid (SenderId=u.PhoneNumber) (cost=0.27 rows=3) (actual time=0.003..0.003 rows=0 loops=99)
|             -> Filter: (cast(u.PhoneNumber as double) = cast(t2.ReceiverId as double)) (cost=0.26 rows=3) (actual time=0.002..0.002 rows=0 loops=99)
|               -> Covering index lookup on t2 using idx_transaction_receiver (ReceiverId=u.PhoneNumber) (cost=0.26 rows=3) (actual time=0.002..0.002 rows=0 loops=99)
|
+-----+
```

**Index Set 1: Applying index on (PhoneNumber);****COST = 82.92****CREATE INDEX idx\_user\_phone ON User (PhoneNumber);**

```
-----+
| -> Filter: (t2.ReceiverId is null) (cost=82.92 rows=119) (actual time=0.157..0.821 rows=99 loops=1)
|   -> Nested loop antijoin (cost=82.92 rows=119) (actual time=0.157..0.813 rows=99 loops=1)
|     -> Filter: (t1.SenderId is null) (cost=61.09 rows=40) (actual time=0.150..0.660 rows=99 loops=1)
|       -> Nested loop antijoin (cost=61.09 rows=40) (actual time=0.150..0.651 rows=99 loops=1)
|         -> Filter: (u.DateCreated like '2024%') (cost=53.81 rows=13) (actual time=0.130..0.489 rows=99 loops=1)
|           -> Index range scan on u using idx_user_phone over ('1' <= PhoneNumber <= '1????????????????????????????????????????'), with index condition: (u.PhoneNu
mber like '1%') (cost=53.81 rows=119) (actual time=0.090..0.444 rows=119 loops=1)
|             -> Filter: (cast(u.PhoneNumber as double) = cast(t1.SenderId as double)) (cost=0.27 rows=3) (actual time=0.001..0.001 rows=0 loops=99)
|               -> Covering index lookup on t1 using idx_transaction_senderid (SenderId=u.PhoneNumber) (cost=0.27 rows=3) (actual time=0.001..0.001 rows=0 loops=99)
|             -> Filter: (cast(u.PhoneNumber as double) = cast(t2.ReceiverId as double)) (cost=0.26 rows=3) (actual time=0.001..0.001 rows=0 loops=99)
|               -> Covering index lookup on t2 using idx_transaction_receiver (ReceiverId=u.PhoneNumber) (cost=0.26 rows=3) (actual time=0.001..0.001 rows=0 loops=99)
|
+-----+
```

**Index Set 2: Applying index on (DateCreated);****COST = 82.92****CREATE INDEX DateCreate ON User (DateCreated);**

```
-----+
| -> Filter: (t2.ReceiverId is null) (cost=82.92 rows=119) (actual time=0.157..0.821 rows=99 loops=1)
|   -> Nested loop antijoin (cost=82.92 rows=119) (actual time=0.157..0.813 rows=99 loops=1)
|     -> Filter: (t1.SenderId is null) (cost=61.09 rows=40) (actual time=0.150..0.660 rows=99 loops=1)
|       -> Nested loop antijoin (cost=61.09 rows=40) (actual time=0.150..0.651 rows=99 loops=1)
|         -> Filter: (u.DateCreated like '2024%') (cost=53.81 rows=13) (actual time=0.130..0.489 rows=99 loops=1)
|           -> Index range scan on u using idx_user_phone over ('1' <= PhoneNumber <= '1????????????????????????????????????????'), with index condition: (u.PhoneNu
mber like '1%') (cost=53.81 rows=119) (actual time=0.090..0.444 rows=119 loops=1)
|             -> Filter: (cast(u.PhoneNumber as double) = cast(t1.SenderId as double)) (cost=0.27 rows=3) (actual time=0.001..0.001 rows=0 loops=99)
|               -> Covering index lookup on t1 using idx_transaction_senderid (SenderId=u.PhoneNumber) (cost=0.27 rows=3) (actual time=0.001..0.001 rows=0 loops=99)
|             -> Filter: (cast(u.PhoneNumber as double) = cast(t2.ReceiverId as double)) (cost=0.26 rows=3) (actual time=0.001..0.001 rows=0 loops=99)
|               -> Covering index lookup on t2 using idx_transaction_receiver (ReceiverId=u.PhoneNumber) (cost=0.26 rows=3) (actual time=0.001..0.001 rows=0 loops=99)
|
+-----+
```

**Index Set 3: Applying index on (PhoneNumber, DateCreate);****COST = 54.17****CREATE INDEX PhoneNumber, DateCreate ON User (PhoneNumber, DateCreate);**

```
-----+
| -> Filter: (t2.ReceiverId is null) (cost=54.17 rows=119) (actual time=0.035..0.534 rows=99 loops=1)
|   -> Nested loop antijoin (cost=54.17 rows=119) (actual time=0.035..0.525 rows=99 loops=1)
|     -> Filter: (t1.SenderId is null) (cost=32.34 rows=40) (actual time=0.030..0.311 rows=99 loops=1)
|       -> Nested loop antijoin (cost=32.34 rows=40) (actual time=0.029..0.302 rows=99 loops=1)
|         -> Filter: ((u.PhoneNumber like '1%') and (u.DateCreated like '2024%')) (cost=25.06 rows=13) (actual time=0.020..0.124 rows=99 loops=1)
|           -> Covering index range scan on u using idx_user_phone over ('1' <= PhoneNumber <= '1????????????????????????????????????????') (cost=25.06 rows=119) (
actual time=0.012..0.073 rows=119 loops=1)
|             -> Filter: (cast(u.PhoneNumber as double) = cast(t1.SenderId as double)) (cost=0.27 rows=3) (actual time=0.002..0.002 rows=0 loops=99)
|               -> Covering index lookup on t1 using idx_transaction_senderid (SenderId=u.PhoneNumber) (cost=0.27 rows=3) (actual time=0.002..0.002 rows=0 loops=99)
|             -> Filter: (cast(u.PhoneNumber as double) = cast(t2.ReceiverId as double)) (cost=0.26 rows=3) (actual time=0.002..0.002 rows=0 loops=99)
|               -> Covering index lookup on t2 using idx_transaction_receiver (ReceiverId=u.PhoneNumber) (cost=0.26 rows=3) (actual time=0.002..0.002 rows=0 loops=99)
|
+-----+
```



**Part1: Query # 2:**  
**Before applying indexing: COST = 104.06**

```

| -> Filter: (<in_optimizer>('User'.UserId,<exists>(select #2) is false) and ('User'.DateCreated like '2024%') and ('User'.PhoneNumber like '1%')) (cost=104.06 rows=12) (actual time=0.129
..4.871 rows=93 loops=1)
  -> Table scan on User (cost=104.06 rows=1000) (actual time=0.051..0.365 rows=1000 loops=1)
  -> Select #2 (subquery in condition; dependent)
    -> Limit: 1 row(s) (cost=2.29..2.29 rows=1) (actual time=0.004..0.004 rows=0 loops=1000)
    -> Table scan on <union temporary> (cost=2.29..3.55 rows=2) (actual time=0.003..0.003 rows=0 loops=1000)
    -> Union materialize with deduplication (cost=1.03..1.03 rows=2) (actual time=0.003..0.003 rows=0 loops=1000)
    -> Limit table size: 1 unique row(s)
    -> Limit: 1 row(s) (cost=0.37 rows=1) (actual time=0.002..0.002 rows=0 loops=1000)
    -> Covering index lookup on Transaction using idx_transaction_senderid (SenderId=<cache>('User'.UserId)) (cost=0.37 rows=1) (actual time=0.001..0.001 rows=0 lo
ops=1000)
    -> Limit table size: 1 unique row(s)
    -> Limit: 1 row(s) (cost=0.45 rows=1) (actual time=0.001..0.001 rows=0 loops=959)
    -> Covering index lookup on Transaction using idx_transaction_receiver (ReceiverId=<cache>('User'.UserId)) (cost=0.45 rows=2) (actual time=0.001..0.001 rows=0
loops=959)
|

```

**Index Set 1: Applying index on (Name);**  
**COST = 74.73**  
 CREATE INDEX idx\_user\_name ON User (Name);

[illegible]

**Index Set 2: Applying index on (Name, DateCreated);**  
**COST = 74.73**  
 CREATE INDEX idx\_user\_name\_datecreated ON User (Name, DateCreated);

[illegible]

**Index Set 3: Applying index on (Name, PhoneNumber, DateCreated);**  
**COST = 18.3**  
 CREATE INDEX idx\_user\_name\_phone\_datecreated ON User (Name, PhoneNumber, DateCreated);

```
| -> Filter: (<in optimizer>('User'.UserId,<exists>(select #2) is false) and ('User'.Name like 'c%') and ('User.DateCreated like '2024%') and ('User.PhoneNumber like '1%')) (cost=18.3  
6 rows=1) (actual time=0.099..0.495 rows=9 loops=1)  
|   |  
|   |   -> Covering index range scan on User using idx_user_phone over ('c' <= Name <= 'c?????????????????????????????????????????????????????????????????  
????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????  
????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????  
????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????????  
????') (cost=18.36 rows=73) (actual time=0.019..0.069 rows=73 loops=1)  
|   |   |  
|   |   |   -> Select #2 (subquery in condition; dependent)  
|   |   |   |  
|   |   |   |   -> Limit: 1 row(s) (cost=2.29..2.29 rows=1) (actual time=0.004..0.004 rows=0 loops=73)  
|   |   |   |   |  
|   |   |   |   |   -> Table scan on <union temporary> (cost=2.29..3.55 rows=2) (actual time=0.004..0.004 rows=0 loops=73)  
|   |   |   |   |   |  
|   |   |   |   |   |   -> Union materialize with deduplication (cost=1.03..1.03 rows=2) (actual time=0.004..0.004 rows=0 loops=73)  
|   |   |   |   |   |   |  
|   |   |   |   |   |   |   -> Limit table size: 1 unique row(s)  
|   |   |   |   |   |   |   |  
|   |   |   |   |   |   |   |   -> Limit: 1 row(s) (cost=0.37 rows=1) (actual time=0.002..0.002 rows=0 loops=73)  
|   |   |   |   |   |   |   |   |  
|   |   |   |   |   |   |   |   |   -> Covering index lookup on Transaction using idx_transaction_senderid (SenderId=<cache>('User'.UserId)) (cost=0.37 rows=1) (actual time=0.002..0.002 rows=0 lo  
ops=73)  
|   |   |   |   |   |   |   |   |   |  
|   |   |   |   |   |   |   |   |   |   -> Limit table size: 1 unique row(s)  
|   |   |   |   |   |   |   |   |   |   |  
|   |   |   |   |   |   |   |   |   |   |   -> Limit: 1 row(s) (cost=0.45 rows=1) (actual time=0.002..0.002 rows=0 loops=70)  
|   |   |   |   |   |   |   |   |   |   |   |  
|   |   |   |   |   |   |   |   |   |   |   |   -> Covering index lookup on Transaction using idx_transaction_receiver (ReceiverId=<cache>('User'.UserId)) (cost=0.45 rows=2) (actual time=0.001..0.001 rows=0  
loops=70)  
|   |  
|
```



## Part1: Query # 1:

**Before applying indexing: COST = 18.61**

```
-----+
| -> Table scan on <temporary> (actual time=0.412..0.418 rows=26 loops=1)
|   -> Aggregate using temporary table (actual time=0.411..0.411 rows=26 loops=1)
|     -> Nested loop inner join (cost=18.61 rows=13) (actual time=0.060..0.318 rows=36 loops=1)
|       -> Nested loop inner join (cost=6.69 rows=13) (actual time=0.050..0.100 rows=36 loops=1)
|         -> Filter: (g.GroupName = 'Trip to Japan') (cost=4.35 rows=4) (actual time=0.034..0.053 rows=1 loops=1)
|           -> Table scan on g (cost=4.35 rows=41) (actual time=0.029..0.040 rows=42 loops=1)
|             -> Covering index lookup on t using idx_transaction_group_sender_amount (GroupId=g.GroupId) (cost=0.33 rows=3) (actual time=0.014..0.041 rows=36 loops=1)
|             -> Single-row index lookup on u using PRIMARY (UserId=t.SenderId) (cost=0.82 rows=1) (actual time=0.002..0.003 rows=1 loops=36)
|
|-----+
```

## **Index Set 1: Applying index on (Name);**

**COST = 18.61**

**CREATE INDEX idx\_user\_name ON User (Name);**

```
-----+
| -> Table scan on <temporary> (actual time=0.412..0.418 rows=26 loops=1)
|   -> Aggregate using temporary table (actual time=0.411..0.411 rows=26 loops=1)
|     -> Nested loop inner join (cost=18.61 rows=13) (actual time=0.060..0.318 rows=36 loops=1)
|       -> Nested loop inner join (cost=6.69 rows=13) (actual time=0.050..0.100 rows=36 loops=1)
|         -> Filter: (g.GroupName = 'Trip to Japan') (cost=4.35 rows=4) (actual time=0.034..0.053 rows=1 loops=1)
|           -> Table scan on g (cost=4.35 rows=41) (actual time=0.029..0.040 rows=42 loops=1)
|             -> Covering index lookup on t using idx_transaction_group_sender_amount (GroupId=g.GroupId) (cost=0.33 rows=3) (actual time=0.014..0.041 rows=36 loops=1)
|             -> Single-row index lookup on u using PRIMARY (UserId=t.SenderId) (cost=0.82 rows=1) (actual time=0.002..0.003 rows=1 loops=36)
|
|-----+
```

## **Index Set 2: Applying index on (GroupName);**

**COST = 3.83**

**CREATE INDEX idx\_groupname ON Party (GroupName);**

```
-----+
| -> Table scan on <temporary> (actual time=0.130..0.131 rows=3 loops=1)
|   -> Aggregate using temporary table (actual time=0.128..0.128 rows=3 loops=1)
|     -> Nested loop inner join (cost=3.83 rows=0.4) (actual time=0.065..0.106 rows=3 loops=1)
|       -> Nested loop inner join (cost=0.92 rows=3) (actual time=0.026..0.045 rows=36 loops=1)
|         -> Covering index lookup on g using idx_groupname (GroupName='Trip to Japan') (cost=0.35 rows=1) (actual time=0.015..0.016 rows=1 loops=1)
|         -> Covering index lookup on t using idx_transaction_group_sender_amount (GroupId=g.GroupId) (cost=0.57 rows=3) (actual time=0.009..0.025 rows=36 loops=1)
|       -> Filter: (u.'Name' like 'G%') (cost=0.82 rows=0.1) (actual time=0.002..0.002 rows=0 loops=36)
|         -> Single-row index lookup on u using PRIMARY (UserId=t.SenderId) (cost=0.82 rows=1) (actual time=0.001..0.001 rows=1 loops=36)
|
|-----+
```

## **Index Set 3: Applying index on (Name, GroupName);**

**COST = 3.83**

**CREATE INDEX idx\_groupname ON Party (GroupName);**

**CREATE INDEX idx\_user\_name ON User (Name);**

```
-----+
| -> Table scan on <temporary> (actual time=0.126..0.126 rows=3 loops=1)
|   -> Aggregate using temporary table (actual time=0.125..0.125 rows=3 loops=1)
|     -> Nested loop inner join (cost=3.83 rows=0.2) (actual time=0.058..0.102 rows=3 loops=1)
|       -> Nested loop inner join (cost=0.92 rows=3) (actual time=0.019..0.038 rows=36 loops=1)
|         -> Covering index lookup on g using idx_groupname (GroupName='Trip to Japan') (cost=0.35 rows=1) (actual time=0.010..0.011 rows=1 loops=1)
|         -> Covering index lookup on t using idx_transaction_group_sender_amount (GroupId=g.GroupId) (cost=0.57 rows=3) (actual time=0.008..0.022 rows=36 loops=1)
|       -> Filter: (u.'Name' like 'G%') (cost=0.81 rows=0.05) (actual time=0.002..0.002 rows=0 loops=36)
|         -> Single-row index lookup on u using PRIMARY (UserId=t.SenderId) (cost=0.81 rows=1) (actual time=0.001..0.001 rows=1 loops=36)
|
|-----+
```

## Part 2.2: Advanced Queries Indexing Analysis and Explanation

### **Query 1 Analysis:**

Index Set 1: Cost of query remains at 18.61.

Effect: The single-column index on Name provides minimal optimization, as other fields in the query (GroupName) still require scanning, resulting in limited performance improvement.

Index Set 2: Cost of query decreased to 3.83.

Effect: The index on GroupName effectively reduces the cost by optimizing the filtering of groups, covering more of the query's conditions and reducing the need for additional scans.

Index Set 3: Cost remains at 3.83.

Result: Applying indexes on both Name and GroupName does not further reduce the cost because GroupName already covered the major filtering requirement. Adding Name doesn't provide additional optimization beyond what Index Set 2 achieved.

### **Query 2 Analysis :**

Index Set 1: Cost of a query decreased from 104.06 to 74.73.

Effect: The single-column index Name helps optimize the performance of LIKE 'c%' query, but the performance gain is limited because only one field is indexed and the database still needs to scan the other fields (DateCreated and PhoneNumber).

Index Set 2: The Cost of query remains at 74.73, the same as Index Set 1.

Effect: Although the index of DateCreated is added, the index cannot cover all the conditions of the query because the query condition still involves PhoneNumber. The database still needs to scan the PhoneNumber column, resulting in no significant cost reduction.

Index Set 3: The cost of a query is significantly reduced to 18.3.

Result: The combined index on Name, PhoneNumber, and DateCreated covers all of the query's filters, allowing the database to scan for rows that satisfy the conditions directly through the indexes without the need for an additional full table scan."

### **Query 3 Analysis :**

The (PhoneNumber,DateCreated) compound index is beneficial because it covers both columns used in the WHERE clause, allowing the query to benefit from index coverage. This compound index optimizes retrieval for users created in specific date ranges (like those created in '2024') and further improves lookup performance on PhoneNumber, which is necessary for the left join conditions. Therefore, the (PhoneNumber,DateCreated) index is the best choice as it minimizes query execution costs and aligns well with the query's filtering and joining needs.

### **Query 4 Analysis:**

The query requires retrieving the latest Timestamp for each combination of SourceCurrency and TargetCurrency. The index on (Timestamp, SourceCurrency, TargetCurrency) is particularly beneficial as it allows efficient ordering and filtering by Timestamp to identify the latest entry. It

also supports lookups on SourceCurrency and TargetCurrency, enabling the grouping and join conditions to match currency pairs with their latest timestamp efficiently. The (Timestamp, SourceCurrency, TargetCurrency) index is optimal for this query as it provides the necessary ordering and grouping for efficient retrieval of the latest exchange rate per currency pair.