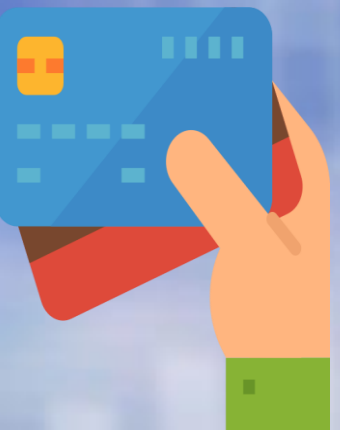
BANKING DATA ANALYSIS IN SQL



DATA SET DESCRIPTION:

Customers Table

- **customer_id**: INT (Primary Key)
- **first_name**: VARCHAR(50)
- **last_name**: VARCHAR(50)
- **date_of_birth**: DATE
- **email**: VARCHAR(100)
- **phone**: BIGINT
- **address**: VARCHAR(255)
- **city**: VARCHAR(50)
- **state**: VARCHAR(50)
- **zip_code**: INT
- **created_at**: TIMESTAMP

Accounts Table

- **account_number**: BIGINT (Primary Key)
- **customer_id**: INT (Foreign Key referencing Customers.customer_id)
- **account_type**: ENUM("Savings", "Current", "Salary", "OverDraft")
- **balance**: DECIMAL(15, 2)
- **branch_id**: INT (Foreign Key referencing Branches.branch_id)
- **created_at**: TIMESTAMP

Transactions Table

- **transaction_id**: INT (Primary Key)
- **account_number**: BIGINT (Foreign Key referencing Accounts.account_number)
- **transaction_type**: ENUM("Deposit", "Withdrawal", "Transfer")
- **amount**: DECIMAL(15, 2)
- **transaction_date**: TIMESTAMP

Branches Table

- **branch_id**: INT (Primary Key)
- **branch_name**: VARCHAR(50)
- **branch_address**: VARCHAR(255)
- **branch_location**: ENUM("Rural", "Urban")
- **city**: VARCHAR(50)
- **state**: VARCHAR(50)
- **zip_code**: INT
- **phone**: BIGINT
- **manager_id**: INT (Foreign Key referencing Employees.employee_id)

1: Identify customers who haven't made transactions in the last year and suggest strategies to re-engage them.

```sql
SELECT DISTINCT(c.customer_id)
FROM customers AS c
JOIN accounts AS A
ON c.customer_id=A.customer_id
JOIN transactions AS T
ON A.account_number=T.account_number
WHERE YEAR(transaction_date)<>2023
ORDER BY c.customer_id;
```

Result Grid | Filter Rows:

customer_id
1
2
3
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19

Result 98

Reasons for Inactivity:

- Convenience Issues
- Shift in Preferences

Here are some reactivation strategies based on the customer's region:

-Targeted Promotions: Offer location-specific discounts or deals relevant to their area.

-Local Events: If your business has a regional presence, invite them to local events or promotions.

-Personalized Emails: Use the customer's email address for personalized reactivation emails highlighting products or services relevant to their past purchases

2: Summarize the total transaction amount per account per month.

```sql
SELECT T.account_number, MONTH(t.transaction_date) AS Months,
ROUND(SUM(t.amount),2) AS total_amount
FROM accounts AS A
JOIN transactions AS T
ON A.account_number=T.account_number
GROUP BY t.account_number, Months
ORDER BY t.account_number, Months;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

account_number	Months	total_amount
1012982863	1	3510.6
1012982863	2	745.36
1012982863	4	3636.29
1012982863	5	16998.37
1012982863	7	162.98
1012982863	10	9445.09
1012982863	12	6419.55
1013036421	1	5137.06
1013036421	2	3363.92
1013036421	3	3093.25
1013036421	4	2080.01
1013036421	5	7530.37
1013036421	8	8473.86
1013036421	10	340.31
1013036421	12	17474.3
1014303562	1	1655.53
1014303562	3	3067.54
1014303562	4	3494.67
1014303562	5	5814.86

Result 99

Output

Action Output

#	Time	Action	Message
195	21:33:58	SELECT DISTINCT(c...	88 row(s) returned

3: Rank branches based on the total amount of deposits made in the last quarter.

```sql
SELECT
b.branch_id, ROUND(SUM(CASE WHEN t.transaction_type = 'deposit' THEN t.amount ELSE 0 END),2) AS totaldeposit,
DENSE_RANK() OVER (ORDER BY SUM(CASE WHEN t.transaction_type = 'deposit' THEN t.amount ELSE 0 END)DESC) AS branch_rank
FROM Transactions AS t
JOIN Accounts AS a
ON t.account_number=a.account_number
JOIN branches AS b
ON b.branch_id=a.branch_id
WHERE t.transaction_date>= DATE_SUB(CURDATE(), INTERVAL 3 MONTH)
GROUP BY b.branch_id
ORDER BY totaldeposit DESC;
```

Result Grid | Filter Rows: | Export:

branch_id	totaldeposit	branch_rank
27	23688.58	1
14	18448.58	2
29	16348.46	3
32	16109.1	4
17	12548.97	5
2	11969.37	6
9	10734.42	7
12	10616.46	8
31	8691.64	9
5	8457.83	10
18	8004.05	11
20	7549.68	12
6	6482.05	13
25	6249.19	14
10	6102.35	15
28	6033.37	16
30	5833.83	17
22	4955.47	18
19	4709.63	19

Result 102

Output

Action Output

#	Time	Action	Message
201	21:43:36	SELECT b.branch_id,...	34 row(s) returned

4: Find the name of the customer who has deposited the highest amount.

```
SELECT c.customer_id, concat(c.first_name, " " ,c.Last_name)
AS customer_name,
MAX(CASE WHEN t.transaction_type = 'deposit' THEN t.amount ELSE 0 END)
AS maximum_deposit
FROM customers AS c
JOIN accounts AS A
ON c.customer_id=A.customer_id
JOIN transactions AS T
ON t.account_number=a.account_number
GROUP BY customer_id, customer_name
ORDER BY maximum_deposit DESC LIMIT 1;
```

Result Grid | Filter Rows: | Export:

customer_id	customer_name	maximum_deposit
15	Gokul Sengupta	4990.4

Result 104

Output

Action Output

#	Time	Action	Message
203	21:51:39	SELECT c.customer_i...	1 row(s) returned

5: Identify any accounts that have made more than two transactions in a single day, which could indicate fraudulent activity. How

No account found who have made more than two transaction in a single day

```sql
SELECT account_number,
DATE(transaction_date) AS transaction_date,
COUNT(transaction_id) AS transaction_count
FROM Transactions
GROUP BY account_number, DATE(transaction_date)
HAVING COUNT(transaction_date) >2;
```

Result Grid | Filter Rows: | Export:

account_number	transaction_date	transaction_count

Result 106

Output

Action Output

#	Time	Action	Message
205	21:57:40	SELECT account_nu...	0 row(s) returned

6: Calculate the average number of transactions per customer per account per month over the last year.

```sql
SELECT a.customer_id, a.account_number,
YEAR(t.transaction_date) AS transaction_year,
MONTH(t.transaction_date) AS transaction_month,
    COUNT(t.transaction_id) AS transaction_count,
    COUNT(t.transaction_id) / 12 AS avg_transactions_per_month_last_year
FROM transactions t
JOIN accounts a ON t.account_number = a.account_number
WHERE t.transaction_date >= DATE_SUB(CURDATE(), INTERVAL 1 YEAR)
GROUP BY a.customer_id, a.account_number, transaction_year, transaction_month
ORDER BY a.customer_id, a.account_number, transaction_year, transaction_month;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

customer_id	account_number	transaction_year	transaction_month	transaction_count	avg_transactions_per_month_last_year
1	1032168449	2023	8	1	0.0833
1	1032168449	2023	9	1	0.0833
1	1032168449	2023	10	2	0.1667
1	1032168449	2024	2	1	0.0833
1	1032168449	2024	3	1	0.0833
1	1032168449	2024	4	1	0.0833
1	1032168449	2024	6	1	0.0833
1	1053739522	2023	7	1	0.0833
1	1053739522	2023	11	1	0.0833
1	1053739522	2024	2	2	0.1667
2	1076633469	2023	8	1	0.0833
2	1076633469	2023	9	1	0.0833
2	1076633469	2024	4	3	0.2500
3	1014303562	2023	11	2	0.1667
3	1014303562	2024	1	1	0.0833

Result 108

Output

Action Output

#	Time	Action	Message
207	22:01:03	SELECT a.customer_i...	819 row(s) returned

7: Write a query to find the daily transaction volume (total amount of all transactions) for the past month.

```
SELECT DATE(transaction_date) AS dates,
ROUND(SUM(amount),2) AS transaction_volume
FROM transactions
WHERE transaction_date>=DATE_SUB(CURDATE(), INTERVAL 1 month)
GROUP BY  dates
ORDER BY dates;
```

Result Grid | Filter Rows: | Export:

dates	transaction_volume
2024-06-11	7513.08
2024-06-12	19277.54
2024-06-14	14557.73
2024-06-15	8140.17
2024-06-16	4627.48
2024-06-17	7027.12
2024-06-18	9822.37
2024-06-19	1134.2
2024-06-20	14478.25
2024-06-21	8019.87
2024-06-22	4839.31
2024-06-23	1212.67
2024-06-24	6299.43
2024-06-25	13606.42
2024-06-26	2701.95
2024-06-27	15909.99
2024-06-28	4933.52

Result 110

Output

Action Output

#	Time	Action	Message
209	22:08:09	SELECT DATE(transa...	27 row(s) returned

8: Calculate the total transaction amount performed by each age group in the past year. (Age groups: 0-17, 18-30, 31-60, 60+)

```sql
SELECT
CASE WHEN TIMESTAMPDIFF(YEAR,date_of_birth,CURRENT_DATE())<=17 THEN "0-17"
     WHEN TIMESTAMPDIFF(YEAR,date_of_birth,CURRENT_DATE()) BETWEEN 18 AND 30 THEN "18-30"
     WHEN TIMESTAMPDIFF(YEAR,date_of_birth,CURRENT_DATE()) BETWEEN 31 AND 60 THEN "31-60"
     ELSE "60+" END AS Age_group,
     ROUND(SUM(t.amount),2) AS total_transaction_amount
FROM Customers AS c
JOIN Accounts AS a ON c.customer_id = a.customer_id
JOIN Transactions AS t ON a.account_number = t.account_number
WHERE t.transaction_date >= DATE_SUB(CURRENT_DATE(), INTERVAL 1 YEAR)
GROUP BY Age_group
ORDER BY AGE_group;
```

Result Grid | Filter Rows: | Export:

Age_group	total_transaction_amount
18-30	520538.68
31-60	1166461.8
60+	1031761.96

Result 112

Output

Action Output

#	Time	Action	Message
211	22:11:11	SELECT CASE WHE...	3 row(s) returned

9: Find the branch with the highest average account balance.

```
SELECT branch_id, AVG(balance) AS average_balance
FROM accounts
GROUP BY Branch_id
ORDER BY average_balance DESC limit 1;
```

Result Grid | Filter Rows: | Export:

Age_group	total_transaction_amount
18-30	520538.68
31-60	1166461.8
60+	1031761.96

Result 112

Output

Action Output

#	Time	Action	Message
211	22:11:11	SELECT CASE WHE...	3 row(s) returned

10: Calculate the average balance per customer at the end of each month in the last year.

```sql
SELECT
    c.customer_id,
    YEAR(t.transaction_date) AS year,
    MONTH(t.transaction_date) AS month,
    ROUND(AVG(a.balance), 3) AS avg_balance_per_customer
FROM accounts as a
INNER JOIN customers c ON a.customer_id = c.customer_id
INNER JOIN transactions t ON t.account_number = a.account_number
WHERE
    t.transaction_date >= DATE_SUB(CURDATE(), INTERVAL 1 YEAR) -- > 2023-7-11
    AND t.transaction_date = (
        SELECT MAX(t2.transaction_date)
        FROM transactions t2
        WHERE t2.account_number = t.account_number
        AND YEAR(t2.transaction_date) = YEAR(t.transaction_date)
        AND MONTH(t2.transaction_date) = MONTH(t.transaction_date))
GROUP BY
    c.customer_id,
    YEAR(t.transaction_date),
    MONTH(t.transaction_date)
    order by  MONTH(t.transaction_date), YEAR(t.transaction_date);
```

Result Grid | Filter Rows: | Export:

customer_id	year	month	avg_balance_per_customer
25	2024	1	4588.59
87	2024	1	7024.7
64	2024	1	5551.215
59	2024	1	7095.29
13	2024	1	6622.175
37	2024	1	3034.67
48	2024	1	8142.66
60	2024	1	8333.12
53	2024	1	5433.47
19	2024	1	5919.6
56	2024	1	8456.54
41	2024	1	4226.905
16	2024	1	7032.18
89	2024	1	4543.66
68	2024	1	3267.93
14	2024	1	9970.33
79	2024	1	6603.365
99	2024	1	6145.61
94	2024	1	7218.86

Result 116

Output

Action Output

#	Time	Action		Message
215	22:18:10	SELECT	c.custom...	625 row(s) returned