

1) ---Query details for all customers

SELECT \* FROM customers

JOIN accounts

ON customers.customer\_id = accounts.customer\_id

ORDER by customer\_id

The screenshot shows the Microsoft SQL Server Management Studio interface. The title bar indicates the connection is to 'SQLQuery1.sql - sqlserverhw.database.windows.net.CustomerAccountLoanDB (tejus (78))'. The 'Object Explorer' on the left shows the database structure, including 'CustomerAccountLoanDB' and its tables. The 'Query Editor' in the center contains the following SQL query:

```
--Query details for all customers
SELECT * FROM customers
JOIN accounts
ON customers.customer_id = accounts.customer_id
ORDER by customers.customer_id
```

The 'Results' pane at the bottom displays the output of the query as a table with 20 rows. The columns are: customer\_id, first\_name, last\_name, address, city, state, zip, account\_id, customer\_id, account\_type, and balance. The data is sorted by customer\_id.

customer_id	first_name	last_name	address	city	state	zip	account_id	customer_id	account_type	balance
1	John	Doe	123 Elm St	Toronto	ON	M4B1B3	88	1	Checking	8900.00
2	Jane	Smith	456 Maple Ave	Ottawa	ON	K1A0B1	82	2	Checking	8300.50
3	Michael	Johnson	789 Oak Dr	Montreal	QC	H1A1A1	11	3	Savings	1100.75
4	Emily	Davis	101 Pine Rd	Calgary	AB	T2A0A1	78	4	Checking	7900.50
5	David	Wilson	202 Birch Blvd	Vancouver	BC	V5K0A1	18	5	Checking	1600.50
6	Emma	Clark	505 Cedar St	Halifax	NS	B3H0A1	48	6	Checking	4900.00
7	James	Martinez	606 Spruce Ln	Winnipeg	MB	R3C0A1	28	7	Checking	2900.00
8	Olivia	Garcia	707 Fir St	Edmonton	AB	T5A0A1	68	8	Checking	6900.00
9	William	Lopez	808 Redwood Dr	Victoria	BC	V8W0A1	32	9	Checking	3300.00
10	Ava	Anderson	909 Cypress Ave	Quebec City	QC	G1A0A1	52	10	Checking	5300.00
11	Alexander	Thomas	1010 Willow Rd	St John's	NL	A1A0A1	24	11	Checking	2600.00
12	Isabella	Lee	1111 Poplar St	Fredericton	NB	E3B0A1	2	12	Checking	2500.75
13	Isabella	Lee	1111 Poplar St	Fredericton	NB	E3B0A1	64	12	Checking	6500.00
14	Daniel	Harris	1212 Ash Blvd	Charlottetown	PE	C1A0A1	44	13	Checking	4500.00
15	Sophia	Young	1313 Beech Dr	Yellowknife	NT	X1A0A1	9	14	Savings	900.25
16	Matthew	King	1414 Cedar Ln	Whitehorse	YT	Y1A0A1	38	15	Checking	3900.50
17	Charlotte	Scott	1515 Elm St	Saguenay	NB	X0A0A1	58	16	Checking	5900.50
18	Joseph	Green	1616 Maple Ave	Regina	SK	S4P0A1	72	17	Checking	7300.00
19	Amelia	Adams	1717 Oak Dr	Saskatoon	SK	S7K0A1	16	18	Checking	1400.00
20	Christopher	Baker	1818 Pine Rd	Thunder Bay	ON	P7A0A1	40	19	Checking	4100.00

The status bar at the bottom indicates 'Query executed successfully.' and '88 rows'.

2) ---Query details of specific customers

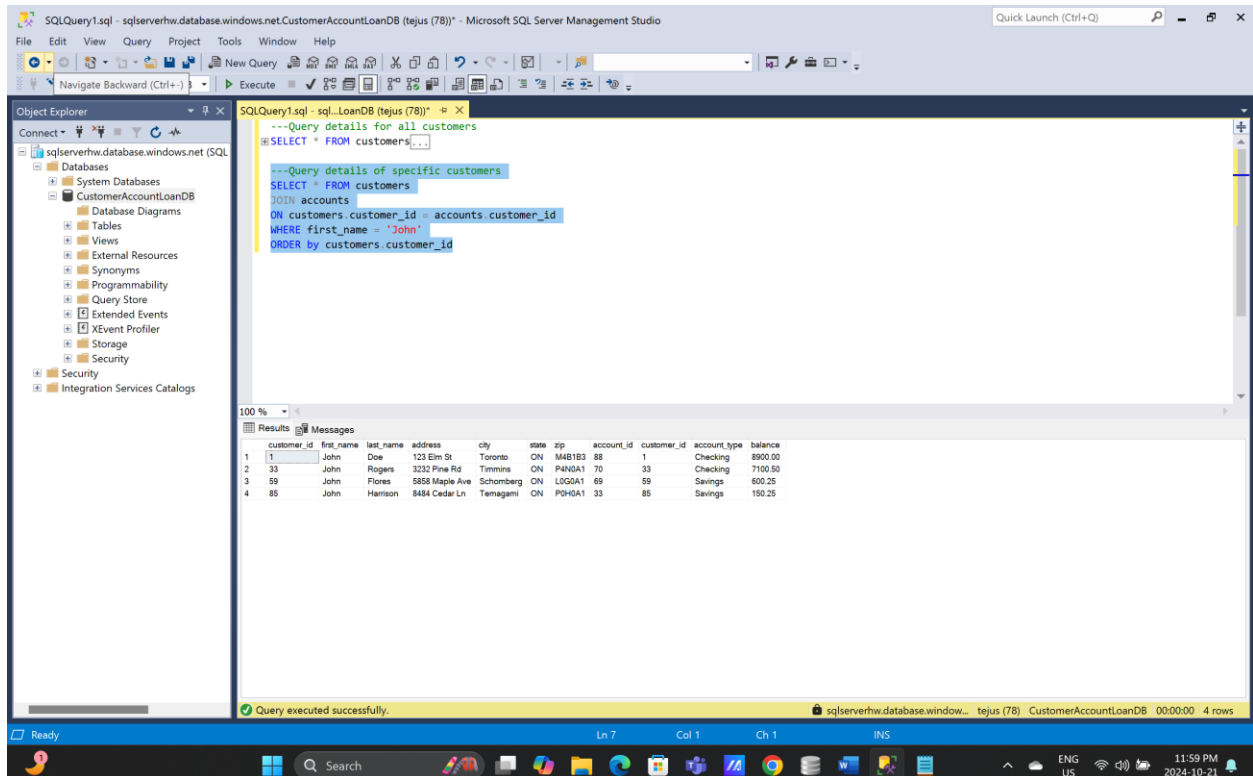
SELECT \* FROM customers

JOIN accounts

ON customers.customer\_id = accounts.customer\_id

WHERE first\_name = 'John'

ORDER by customers.customer\_id



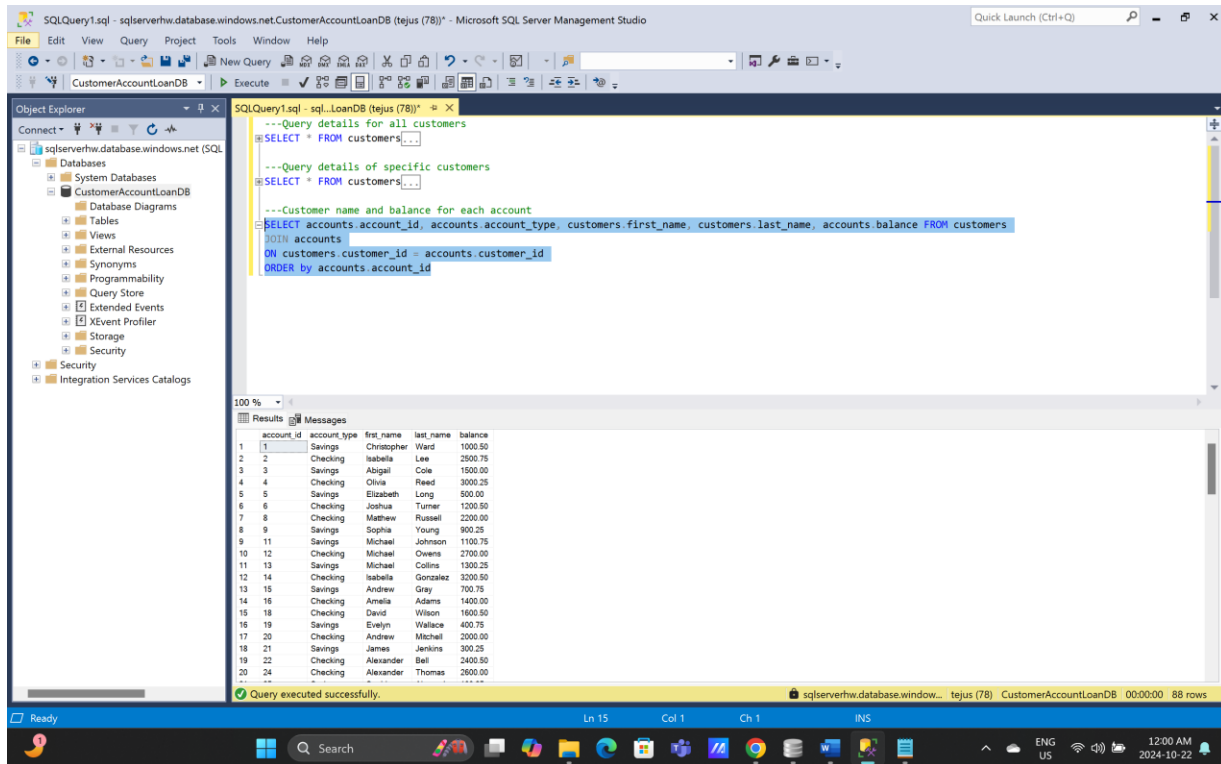
3) ---Customer name and balance for each account

SELECT accounts.account\_id, accounts.account\_type, customers.first\_name, customers.last\_name, accounts.balance FROM customers

JOIN accounts

ON customers.customer\_id = accounts.customer\_id

ORDER by accounts.account\_id



#### 4) ---Analyze customer loan balances

```

SELECT c.customer_id,c.first_name,c.last_name, l.loan_id, l.loan_amount, l.interest_rate, l.loan_term,
((l.loan_amount+((l.loan_amount*l.interest_rate*(l.loan_term/12))/100)) as [repayment_amount],
lp.payment_amount, lp.payment_date,
((l.loan_amount+((l.loan_amount*l.interest_rate*(l.loan_term/12))/100))-lp.payment_amount) as
[loan_balance] FROM customers as c

```

JOIN loans as l

ON c.customer\_id = l.customer\_id

JOIN loan\_payments as lp

ON l.loan\_id = lp.loan\_id

ORDER by c.customer\_id

The screenshot shows the Microsoft SQL Server Management Studio interface. The query editor contains the following SQL code:

```
--Query details for all customers
SELECT * FROM customers

---Analyze customer loan balances
SELECT c.customer_id, c.first_name, c.last_name, l.loan_id, l.loan_amount, l.interest_rate, l.loan_term, (l.loan_amount*((1.loan_amount*1.interest_rate*(1.loan_t
JOIN loans as l
ON c.customer_id = l.customer_id
JOIN loan_payments as lp
ON l.loan_id = lp.loan_id
ORDER by c.customer_id
```

The Results pane displays the following data:

customer_id	first_name	last_name	loan_id	loan_amount	interest_rate	loan_term	repayment_amount	payment_amount	payment_date	loan_balance
1	John	Doe	81	27500	3	24	29150	900	2024-01-17	28250
2	Jane	Smith	82	20000.5	4.5	48	23600.59	3600	2024-03-11	20000.59
3	Michael	Johnson	11	10000.75	6	60	13000.975	550	2024-01-10	12450.975
4	Emily	Davis	78	27500.5	4	48	31900.58	450	2024-01-08	31450.58
5	David	Wilson	18	27500.5	4.5	48	32450.59	2400	2024-02-16	30950.59
6	Emma	Clark	48	27500	3	24	29150	3900	2024-03-17	25250
7	James	Martinez	28	27500	3.5	24	29425	2900	2024-02-26	26525
8	Olivia	Garcia	68	27500	3.5	24	29425	4900	2024-04-06	24525
9	William	Lopez	32	20000	3	24	21200	1100	2024-01-21	20100
10	Ava	Anderson	52	20000	3.5	24	21400	2100	2024-02-10	19300
11	Alexander	Thomas	24	30000	3	24	31800	4700	2024-04-02	27100
12	Isabella	Lee	2	20000.75	4.5	48	23600.885	4600	2024-03-31	19000.885
13	Isabella	Lee	64	30000	3	24	31800	1700	2024-02-02	30100
14	Daniel	Harris	44	30000	3.5	24	32100	700	2024-01-13	31400
15	Sophia	Young	9	32500.25	5.5	36	37862.79125	1450	2024-01-28	36412.79125
16	Matthew	King	38	27500.5	4	48	31900.58	3400	2024-03-07	28500.58
17	Charlotte	Scott	58	27500.5	4.5	48	32450.59	4400	2024-03-27	28550.59
18	Joseph	Green	72	20000	3	24	21200	3100	2024-03-01	18100
19	Amelia	Adams	16	17500	3	24	18550	3300	2024-03-05	15250
20	Christopher	Baker	40	37500	3	24	39750	2500	2024-02-18	37250

The status bar at the bottom indicates: Query executed successfully. sqlserverhw.database.window... tejus (78) CustomerAccountLoanDB 00:00:00 88 rows

5) --All customers with transactions in March 2024

SELECT customers.customer\_id, customers.first\_name, customers.last\_name,  
transactions.transaction\_date FROM customers

JOIN accounts

ON customers.customer\_id = accounts.customer\_id

JOIN transactions

ON accounts.account\_id = transactions.account\_id

WHERE transactions.transaction\_date >= '20240301'

AND transactions.transaction\_date < '20240401'

ORDER by customers.customer\_id

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left displays the database structure for 'CustomerAccountLoanDB'. The central query window contains the following SQL code:

```

JOIN loan_payments as lp
ON l.loan_id = lp.loan_id
ORDER by c.customer_id

--All customers with transactions in March 2024
SELECT customers.customer_id, customers.first_name, customers.last_name, transactions.transaction_date FROM customers
JOIN accounts
ON customers.customer_id = accounts.customer_id
JOIN transactions
ON accounts.account_id = transactions.account_id
WHERE transactions.transaction_date >= '20240301'
AND transactions.transaction_date < '20240401'
ORDER by customers.customer_id

```

The Results pane at the bottom displays the output of the query, showing a list of customers and their transaction dates. The data is as follows:

customer_id	first_name	last_name	transaction_date
2	Jane	Smith	2024-03-23 00:00:00.000
10	Ava	Anderson	2024-03-01 00:00:00.000
12	Isabella	Lee	2024-03-22 00:00:00.000
15	Matthew	King	2024-03-30 00:00:00.000
19	Christopher	Baker	2024-03-24 00:00:00.000
25	Daniel	Campbell	2024-03-06 00:00:00.000
32	Sophia	Morrison	2024-03-30 00:00:00.000
33	John	Rogers	2024-03-21 00:00:00.000
34	Olivia	Reed	2024-03-18 00:00:00.000
37	Alexander	Ball	2024-03-16 00:00:00.000
40	Sophia	Rivers	2024-03-03 00:00:00.000
42	Charlotte	Richardson	2024-03-29 00:00:00.000
45	Christopher	Ward	2024-03-28 00:00:00.000
48	Harper	James	2024-03-07 00:00:00.000
52	Abigail	Henderson	2024-03-19 00:00:00.000
53	James	Jenkins	2024-03-26 00:00:00.000
59	John	Flores	2024-03-05 00:00:00.000
62	Ava	Simmons	2024-03-02 00:00:00.000
63	Alexander	Foster	2024-03-19 00:00:00.000
67	Matthew	Russell	2024-03-08 00:00:00.000

The status bar at the bottom indicates that the query was executed successfully, returning 28 rows.

1) --Calculate the total balance across all accounts for each customer

SELECT customers.customer\_id, customers.first\_name, customers.last\_name, (sum(accounts.balance))  
as [total\_balance] FROM customers

JOIN accounts

ON customers.customer\_id = accounts.customer\_id

GROUP BY customers.first\_name, customers.customer\_id, customers.last\_name

ORDER BY customers.customer\_id

SQLQuery1.sql - sqlserverhw.database.windows.net.CustomerAccountLoanDB (tejus (78)) - Microsoft SQL Server Management Studio

Object Explorer: sqlserverhw.database.windows.net (SQL)

Query: SQLQuery1.sql - sql...LoanDB (tejus (78))

```
--Calculate the total balance across all accounts for each customer
SELECT customers.customer_id, customers.first_name, customers.last_name, (sum(accounts.balance)) as [total_balance] FROM customers
JOIN accounts
ON customers.customer_id = accounts.customer_id
GROUP BY customers.first_name, customers.customer_id, customers.last_name
ORDER BY customers.customer_id
```

Results: Messages

customer_id	first_name	last_name	total_balance
1	John	Doe	8900.00
2	Jane	Smith	8300.50
3	Michael	Johnson	1100.75
4	Emily	Davis	7900.50
5	David	Wilson	1600.50
6	Emma	Clark	4900.00
7	James	Martinez	2900.00
8	Olivia	Garcia	6900.00
9	William	Lopez	3300.00
10	Ava	Anderson	5300.00
11	Alexander	Thomas	2600.00
12	Isabella	Lee	9000.75
13	Daniel	Harris	4600.00
14	Sophia	Young	900.25
15	Matthew	King	3900.50
16	Charlotte	Scott	5900.50
17	Joseph	Green	7300.00
18	Amelia	Adams	1400.00
19	Christopher	Baker	4100.00
20	Mia	Nelson	6100.00

Query executed successfully.

- 2) --Calculate the average loan amount for each loan term:
- ```
SELECT c.customer_id,c.first_name,c.last_name, l.loan_id, l.loan_amount, l.loan_term ,
(l.loan_amount/l.loan_term) as [avg_loan_amount_per__term] FROM customers as c
JOIN loans as l
ON c.customer_id = l.customer_id
JOIN loan_payments as lp
ON l.loan_id = lp.loan_id
ORDER by c.customer_id
```

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left displays the database structure for 'CustomerAccountLoanDB'. The query window in the center contains the following SQL code:

```

SELECT customers.customer_id, customers.first_name, customers.last_name, (sum(accounts.balance)) as [total_balance] FROM customers
JOIN accounts
ON customers.customer_id = accounts.customer_id
GROUP BY customers.first_name, customers.customer_id, customers.last_name
ORDER BY customers.customer_id

--Calculate the average loan amount for each loan term:
SELECT c.customer_id, c.first_name, c.last_name, l.loan_id, l.loan_amount, l.loan_term, (l.loan_amount/l.loan_term) as [avg_loan_amount_per_term]
FROM customers c
JOIN loans l ON c.customer_id = l.customer_id
JOIN loan_payments lp ON l.loan_id = lp.loan_id
ORDER BY c.customer_id

--Find the total loan amount and interest across all loans:
SELECT (sum(loan_amount)) as [total_loan_amount], sum(((loan_amount*interest_rate*(loan_term/12))/100)) as [total_interest_amount] FROM loans

--Find the most frequent transaction type

```

The Results pane at the bottom shows the output of the first query, displaying columns: customer\_id, first\_name, last\_name, loan\_id, loan\_amount, loan\_term, and avg\_loan\_amount\_per\_term. The status bar indicates 'Query executed successfully' and '88 rows'.

3) --Find the total loan amount and interest across all loans:

```

SELECT (sum(loan_amount)) as [total_loan_amount],
sum(((loan_amount*interest_rate*(loan_term/12))/100)) as [total_interest_amount] FROM loans

```

The screenshot shows the Microsoft SQL Server Management Studio interface. The query window contains the following SQL code:

```

--Calculate the total balance across all accounts for each customer
SELECT customers.customer_id, customers.first_name, customers.last_name, (sum(accounts.balance)) as [total_balance] FROM customers
JOIN accounts
ON customers.customer_id = accounts.customer_id
GROUP BY customers.first_name, customers.customer_id, customers.last_name
ORDER BY customers.customer_id

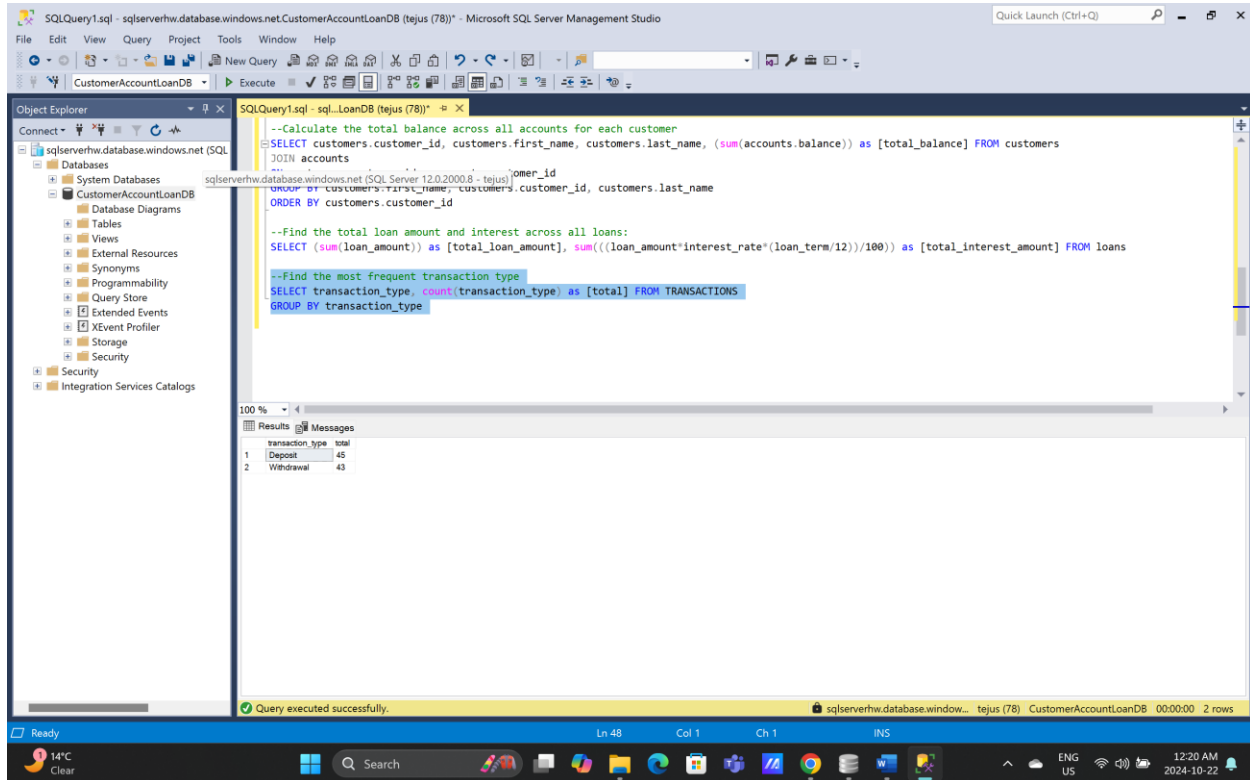
--Find the total loan amount and interest across all loans:
SELECT (sum(loan_amount)) as [total_loan_amount], sum(((loan_amount*interest_rate*(loan_term/12))/100)) as [total_interest_amount] FROM loans

```

The Results pane shows the output of the second query, displaying columns: total\_loan\_amount and total\_interest\_amount. The status bar indicates 'Query executed successfully' and '1 rows'.

4) --Find the most frequent transaction type

```
SELECT transaction_type, count(transaction_type) as [total] FROM TRANSACTIONS  
GROUP BY transaction_type
```



5) --Analyze transactions by account and transaction type:

```
SELECT sum(t.transaction_amount) as [transaction_totals], count(t.transaction_type) as  
[no_of_transactions], t.transaction_type, avg(t.transaction_amount) as [avg_transaction_amt],  
a.account_type FROM TRANSACTIONS as t
```

```
JOIN accounts as a
```

```
ON t.account_id = a.account_id
```

```
--GROUP BY t.transaction_type
```

```
GROUP BY a.account_type, t.transaction_type
```



The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left displays the database structure for 'CustomerAccountLoanDB'. The main query window contains the following SQL code:

```
--Find the most frequent transaction type
SELECT transaction_type, count(transaction_type) as [total] FROM TRANSACTIONS
GROUP BY transaction_type

--Analyze transactions by account and transaction type:
SELECT sum(t.transaction_amount) as [transaction_totals], count(t.transaction_type) as [no_of_transactions], t.transaction_type, avg(t.transaction_amount) as [avg_transaction_amt]
FROM accounts as a
JOIN transactions as t ON t.account_id = a.account_id
GROUP BY t.transaction_type, a.account_type, t.transaction_type
```

The Results pane at the bottom shows the output of the query, which is a table with 5 columns: transaction\_totals, no\_of\_transactions, transaction\_type, avg\_transaction\_amt, and account\_type. The data is as follows:

| transaction_totals | no_of_transactions | transaction_type | avg_transaction_amt | account_type |
|--------------------|--------------------|------------------|---------------------|--------------|
| 5153.5             | 23                 | Deposit          | 224.065217391304    | Checking     |
| 4305.5             | 22                 | Deposit          | 195.704545454545    | Savings      |
| 4708.75            | 25                 | Withdrawal       | 268.35              | Checking     |
| 5067.5             | 18                 | Withdrawal       | 280.972222222222    | Savings      |

The status bar at the bottom indicates 'Query executed successfully.' and '4 rows'.

1) --Create a view of active loans with payments greater than \$1000:

CREATE VIEW active\_loans AS

SELECT loan\_id, payment\_amount

FROM loan\_payments

WHERE payment\_amount>1000

SELECT \* FROM active\_loans

The screenshot shows the Microsoft SQL Server Management Studio interface. The Object Explorer on the left displays the database structure for 'CustomerAccountLoanDB'. The query window on the right contains the following SQL code:

```
--Create a view of active loans with payments greater than $1000:
CREATE VIEW active_loans AS
SELECT loan_id, payment_amount
FROM loan_payments
WHERE payment_amount > 1000

SELECT * FROM active_loans

--Create an index on `transaction_date` in the `transactions` table for performance optimization:
```

The Results window below the query window displays the data from the 'active\_loans' view:

| loan_id | payment_amount |
|---------|----------------|
| 1       | 21             |
| 2       | 32             |
| 3       | 43             |
| 4       | 54             |
| 5       | 65             |
| 6       | 76             |
| 7       | 88             |
| 8       | 9              |
| 9       | 20             |
| 10      | 31             |
| 11      | 42             |
| 12      | 53             |
| 13      | 64             |
| 14      | 75             |
| 15      | 86             |
| 16      | 8              |
| 17      | 19             |
| 18      | 30             |
| 19      | 41             |
| 20      | 52             |

The status bar at the bottom indicates 'Query executed successfully.' and '72 rows'.

2) --Create an index on `transaction\_date` in the `transactions` table for performance optimization:

CREATE INDEX transaction\_date

ON transactions (transaction\_date)

The screenshot shows the Microsoft SQL Server Management Studio interface after executing the index creation query. The query window on the right contains the following SQL code:

```
--Create a view of active loans with payments greater than $1000:
CREATE VIEW active_loans AS
SELECT loan_id, payment_amount
FROM loan_payments
WHERE payment_amount > 1000

SELECT * FROM active_loans

--Create an index on `transaction_date` in the `transactions` table for performance optimization:
CREATE INDEX transaction_date
ON transactions (transaction_date)
```

The Messages window below the query window displays the following message:

```
Message 1093, Level 16, State 1, Line 1
The index 'transaction_date' is already present on the table 'transactions'.
```

The status bar at the bottom indicates 'Query executed successfully.' and '0 rows'.

