UCDPA Final Assignment – Sam Breen

Part 1 – Database Setup and Design

This assignment required the creation of a relational banking database in DuckDB consisting of seven interrelated tables. These tables capture customer data, account activities, employees, loans, transactions, and card usage.

Database and Table Creation

A new DuckDB database named `UCDPA_Assignment` was created. The tables were defined using `CREATE TABLE` statements and established with relevant primary keys and foreign key constraints to maintain data integrity.

Tables Created:
- customers
- branches
- employees
- accounts
- transactions
- cards
- loans

Data Insertion

CSV files were imported using the `read_csv_auto()` function in DuckDB combined with INSERT INTO statements.

Data Verification

SELECT statements and a UNION ALL query were used to verify table population. A manager view was also created and validated.

Part 2 – SQL Queries and Advanced Techniques

Twelve queries were designed using advanced SQL, including window functions, CTEs, subqueries, and set operations.

Advanced SQL Techniques Used

Technique	Description	Applied In
PERCENT_RANK()	Percentile rank for bottom 10% selection	Query 2
LEAD()	Compare current credit score to next higher score	Query 2
Correlated Subquery	Select with outer query value reference	Query 4

COUNT() OVER()	Window function to count without collapsing rows	Query 5
INTERSECT	Ensures customers have both types of accounts	Query 9
Nested CTEs	Layers queries to organise logic	Query 11
NOT EXISTS	Identifies unmatched rows (anti-join pattern)	Query 8
HAVING with Subquery	Confirms customers meet all categorical requirements	Query 12

SQL Query Implementation (Queries 1-12)

Query 1

```
-- Query 1: Employees in London Branches (INNER JOIN)
-- Advanced Technique Explanation:
-- INNER JOIN precisely matches employees to their respective branches by branch_id. Chosen for its efficiency in directly linking records, improving query performance and accuracy when filtering by city.
SELECT e.first_name, e.last_name, e.salary
FROM uk_employees e
JOIN uk_branches b ON e.branch_id = b.branch_id
WHERE b.city = 'London'
```

Query 2

```
-- Query 2: Customers in Lowest 10% Credit Scores (Window Functions - PERCENT_RANK & LEAD)
-- Advanced Technique Explanation:
-- Utilises PERCENT_RANK() to segment customers into percentile ranks and LEAD() to identify gaps between consecutive credit scores. This approach highlights customers in the lowest decile and provides insights into the distribution of low credit scores.
WITH RankedCustomers AS (
  SELECT customer_id, first_name, last_name, credit_score,
      PERCENT_RANK() OVER (ORDER BY credit_score ASC) AS pct_rank,
      LEAD(credit_score) OVER (ORDER BY credit_score ASC) AS next_higher_score
  FROM uk_customers
)
SELECT customer_id, first_name, last_name, credit_score,
    (next_higher_score - credit_score) AS gap_to_next_score
FROM RankedCustomers
WHERE pct_rank <= 0.10
```

Query 3

```
-- Query 3: Account Type Summary (Aggregation & HAVING)
-- Advanced Technique Explanation:
-- Aggregates accounts by type using COUNT and AVG, with a HAVING clause to filter only significant account types. Demonstrates
```

efficiency and clarity in summarising data and filtering aggregated results.

```
SELECT account_type,
    COUNT(*) AS total_accounts,
    ROUND(AVG(balance), 2) AS avg_balance
FROM uk_accounts
GROUP BY account_type
HAVING COUNT(*) > 1
```

Query 4

```
-- Query 4: Branches with High-Value Accounts (Correlated Subquery)
-- Advanced Technique Explanation:
-- Employs a correlated subquery, directly linking each branch to the account counts individually. Clearly demonstrates proficiency in advanced SQL constructs, enhancing clarity and data retrieval accuracy.
SELECT b.branch_name,
    (SELECT COUNT(*) FROM uk_accounts a
     WHERE a.branch_id = b.branch_id AND a.balance > 10000) AS high_value_accounts
FROM uk_branches b
```

Query 5

```
-- Query 5: Transactions per Account Type (Window Function)
-- Advanced Technique Explanation:
-- Utilises window function COUNT() OVER PARTITION BY for efficiently summarising transaction counts per account type, eliminating the need for a GROUP BY clause and showcasing advanced analytical SQL skills.
SELECT DISTINCT a.account_type,
    COUNT(t.transaction_id) OVER (PARTITION BY a.account_type) AS total_transactions
FROM uk_transactions t
JOIN uk_accounts a ON t.account_id = a.account_id
```

Query 6

```
-- Query 6: Branches with High Credit-Score Customers (DISTINCT & JOIN)
-- Technique Explanation:
```

```
-- DISTINCT with JOIN operations clearly identifies unique branches
serving customers with high credit scores, eliminating duplication
from multiple customer accounts per branch.
SELECT DISTINCT b.branch_name
FROM uk_branches b
JOIN uk_accounts a ON b.branch_id = a.branch_id
JOIN uk_customers c ON a.customer_id = c.customer_id
WHERE c.credit_score > 800
```

Query 7

```
-- Query 7: Customers with Mortgage Loans (Multiple JOINs)
-- Advanced Technique Explanation:
-- Clearly demonstrates sequential JOIN operations for accurate
matching between customers, loans, and accounts, filtering
explicitly for mortgage loans.
SELECT DISTINCT c.first_name, c.last_name, a.account_type
FROM uk_customers c
JOIN uk_loans l ON c.customer_id = l.customer_id
JOIN uk_accounts a ON c.customer_id = a.customer_id
WHERE l.loan_type = 'Mortgage'
```

Query 8

```
-- Query 8: Branches without Credit Card Customers (NOT EXISTS)
-- Advanced Technique Explanation:
-- Utilises NOT EXISTS as an anti-join method, clearly and efficiently
identifying branches without customers holding credit cards.
SELECT b.branch_name,
    e.first_name AS manager_first_name,
    e.last_name AS manager_last_name
FROM uk_branches b
LEFT JOIN uk_employees e ON b.manager_id = e.employee_id
WHERE NOT EXISTS (
  SELECT 1
  FROM uk_accounts a
  JOIN uk_cards c ON a.account_id = c.account_id
  WHERE c.card_type = 'Credit' AND a.branch_id = b.branch_id
)
```

Query 9

```
-- Query 9: Customers with Both Current and Savings Accounts
(INTERSECT)
-- Advanced Technique Explanation:
-- Employs INTERSECT set operation to clearly identify customers
holding both Current and Savings accounts
SELECT first_name, last_name
FROM uk_customers
WHERE customer_id IN (
  SELECT customer_id FROM uk_accounts WHERE account_type =
'Current'
  INTERSECT
  SELECT customer_id FROM uk_accounts WHERE account_type =
'Savings'
)
```

Query 10

```
-- Query 10: Loan Amount Range per Loan Type (Aggregation)
-- Advanced Technique Explanation:
-- Aggregates data to compute loan amount ranges by type, clearly
presenting variability insights within loan categories.
SELECT loan_type,
    MAX(amount) - MIN(amount) AS loan_amount_range
FROM uk_loans
GROUP BY loan_type
```

Query 11

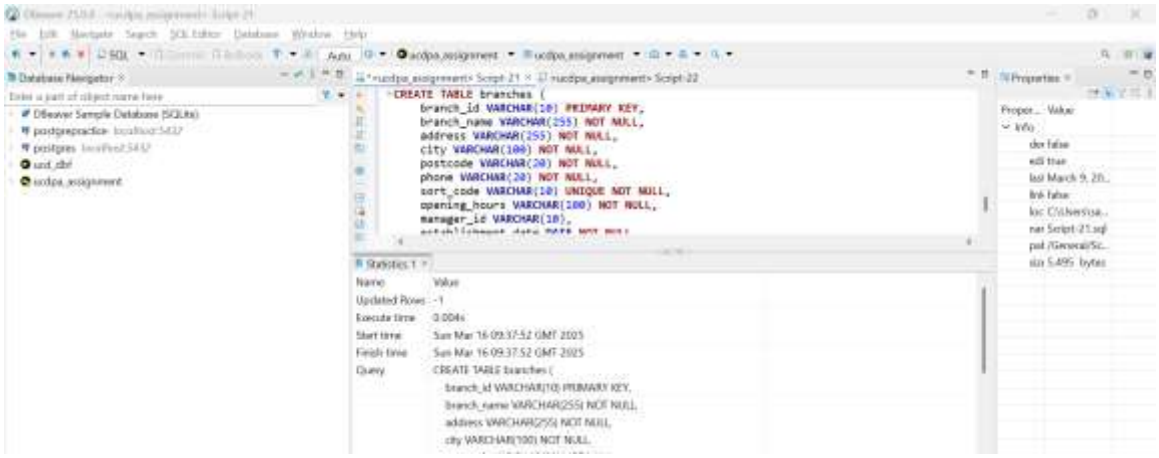
```
-- Query 11: Average Loan Amount by Branch (Nested CTEs)
-- Advanced Technique Explanation:
-- Implements nested Common Table Expressions for organised,
extraction and aggregation.
WITH AccountLoans AS (
  SELECT a.branch_id, l.amount
  FROM uk_accounts a
  JOIN uk_loans l ON a.customer_id = l.customer_id
),
BranchAggregates AS (
  SELECT b.branch_name, AVG(al.amount) AS average_amount
  FROM AccountLoans al
  JOIN uk_branches b ON al.branch_id = b.branch_id
```

```
    GROUP BY b.branch_name
)
SELECT branch_name, ROUND(average_amount, 2) AS
average_loan_amount
FROM BranchAggregates
```

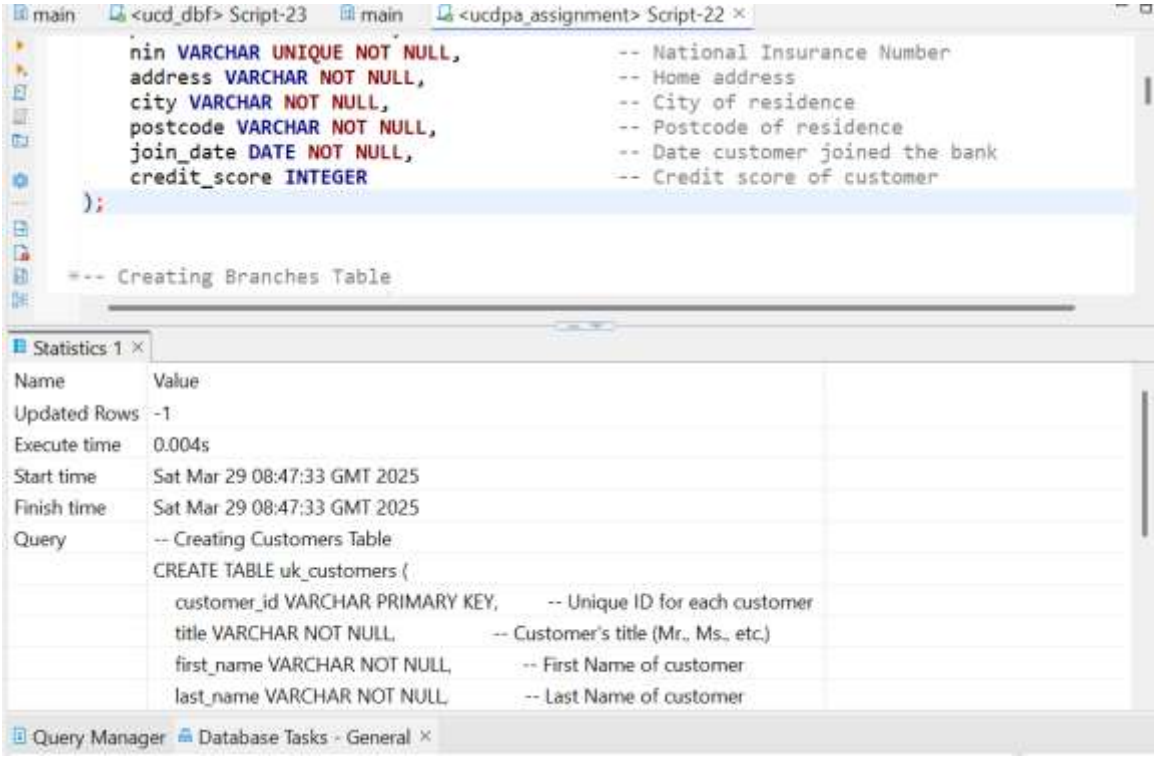
Query 12

```
-- Query 12: Customers with All Account Types (HAVING & Subquery)
--Technique Explanation:
-- Uses HAVING clause and subquery to clearly confirm customers
possessing every available account type
SELECT c.customer_id, c.first_name, c.last_name, c.credit_score
FROM uk_customers c
JOIN uk_accounts a ON c.customer_id = a.customer_id
GROUP BY c.customer_id, c.first_name, c.last_name, c.credit_score
HAVING COUNT(DISTINCT a.account_type) = (
    SELECT COUNT(DISTINCT account_type) FROM uk_accounts
)
```

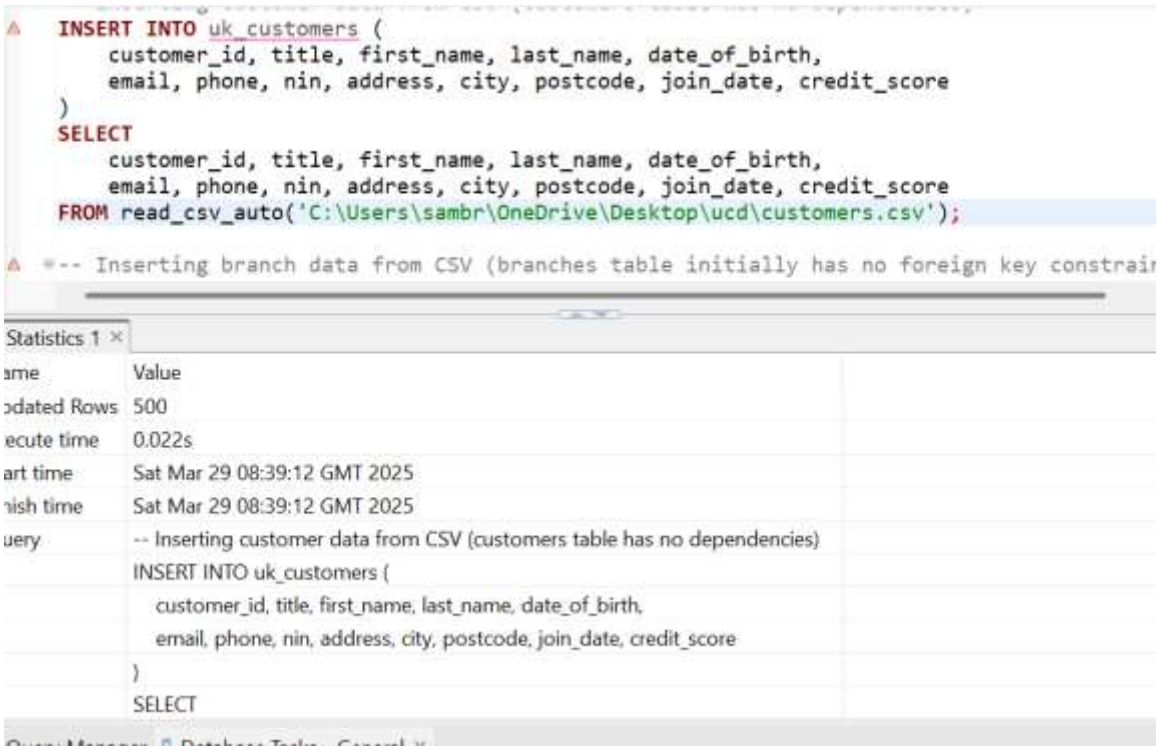
Appendix – Query & Verification Screenshots



Screenshot: create database.png



Screenshot: customer table.png



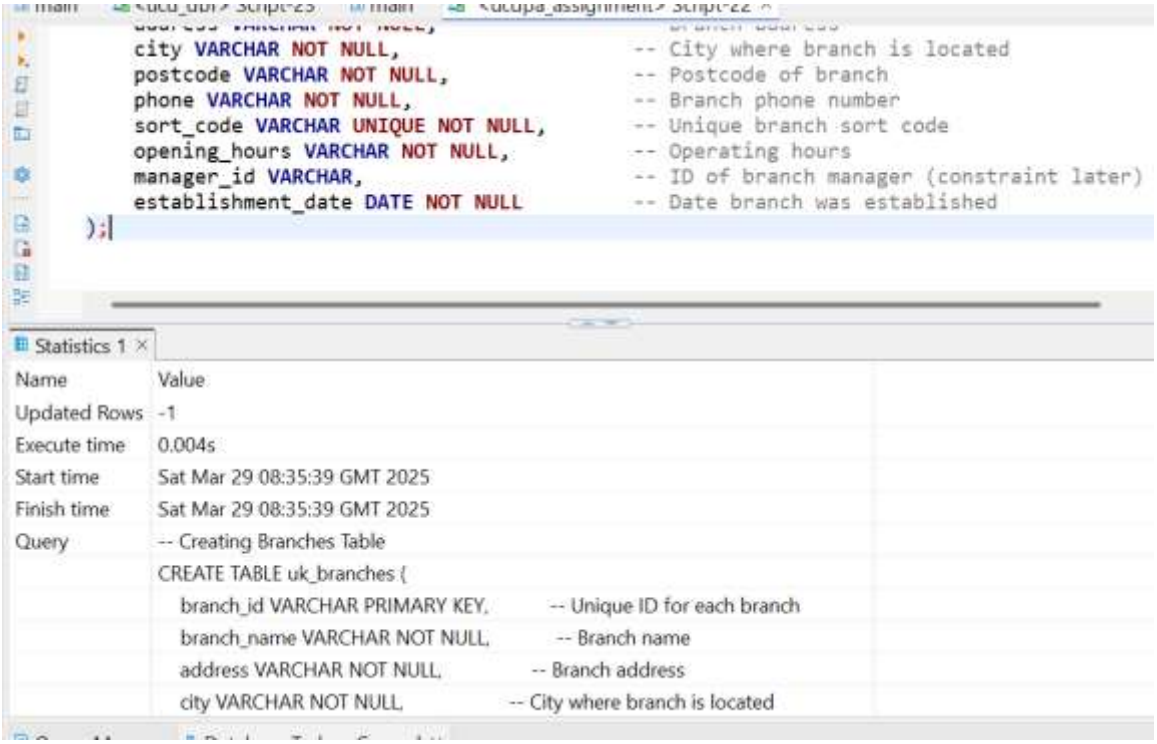
```
INSERT INTO uk_customers (
    customer_id, title, first_name, last_name, date_of_birth,
    email, phone, nin, address, city, postcode, join_date, credit_score
)
SELECT
    customer_id, title, first_name, last_name, date_of_birth,
    email, phone, nin, address, city, postcode, join_date, credit_score
FROM read_csv_auto('C:\Users\sambr\OneDrive\Desktop\ucd\customers.csv');

-- Inserting branch data from CSV (branches table initially has no foreign key constrai
```

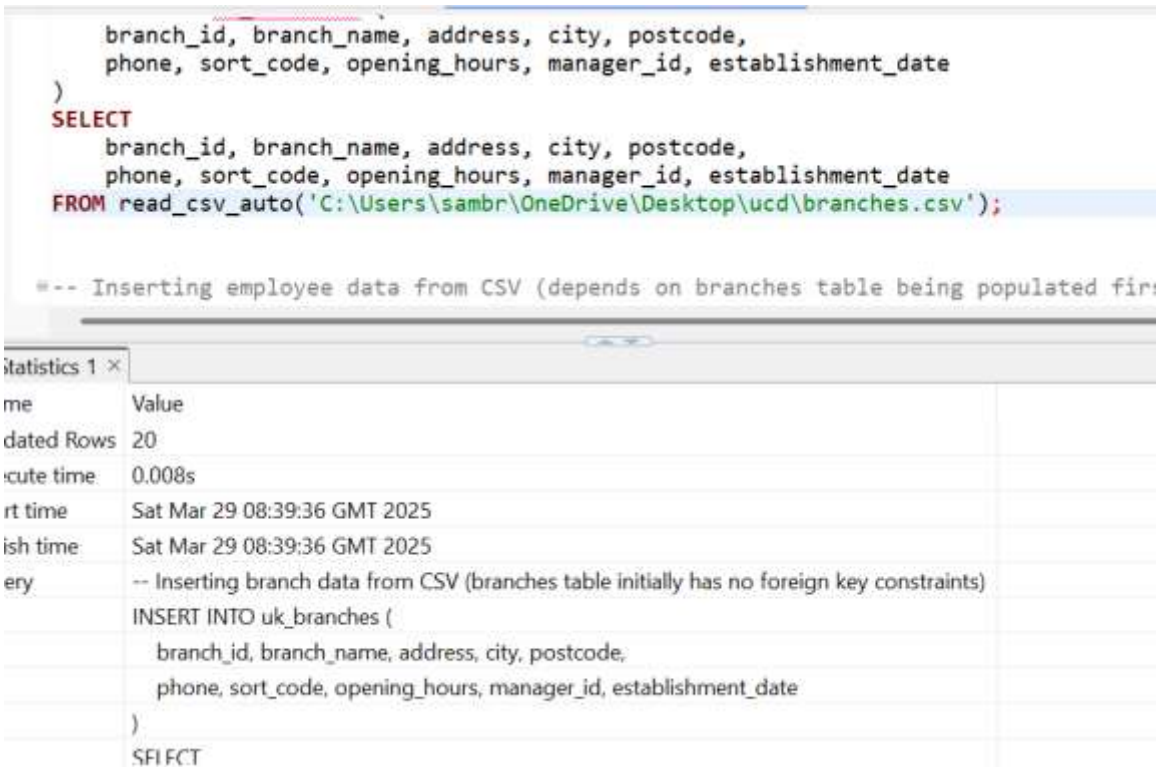
Statistics 1

ame	Value
odated Rows	500
ecute time	0.022s
art time	Sat Mar 29 08:39:12 GMT 2025
nish time	Sat Mar 29 08:39:12 GMT 2025
uery	-- Inserting customer data from CSV (customers table has no dependencies)
	INSERT INTO uk_customers (
	customer_id, title, first_name, last_name, date_of_birth,
	email, phone, nin, address, city, postcode, join_date, credit_score
	)
	SELECT

Screenshot: insert customers.png



Screenshot: branches table.png



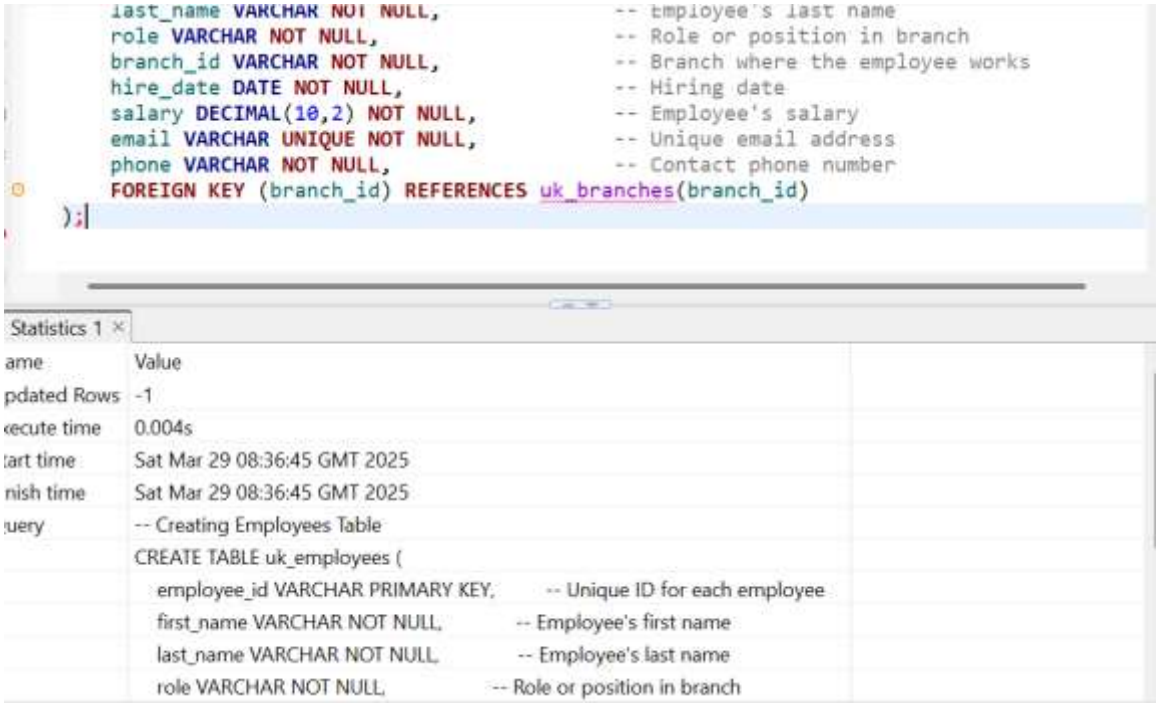
```
    branch_id, branch_name, address, city, postcode,
    phone, sort_code, opening_hours, manager_id, establishment_date
)
SELECT
    branch_id, branch_name, address, city, postcode,
    phone, sort_code, opening_hours, manager_id, establishment_date
FROM read_csv_auto('C:\Users\sambr\OneDrive\Desktop\ucd\branches.csv');


-- Inserting employee data from CSV (depends on branches table being populated fir
```

tatistics 1 ×

me	Value
dated Rows	20
cute time	0.008s
rt time	Sat Mar 29 08:39:36 GMT 2025
ish time	Sat Mar 29 08:39:36 GMT 2025
ery	-- Inserting branch data from CSV (branches table initially has no foreign key constraints)
	INSERT INTO uk_branches (
	branch_id, branch_name, address, city, postcode,
	phone, sort_code, opening_hours, manager_id, establishment_date
	)
	SELECT

Screenshot: insert branch.png

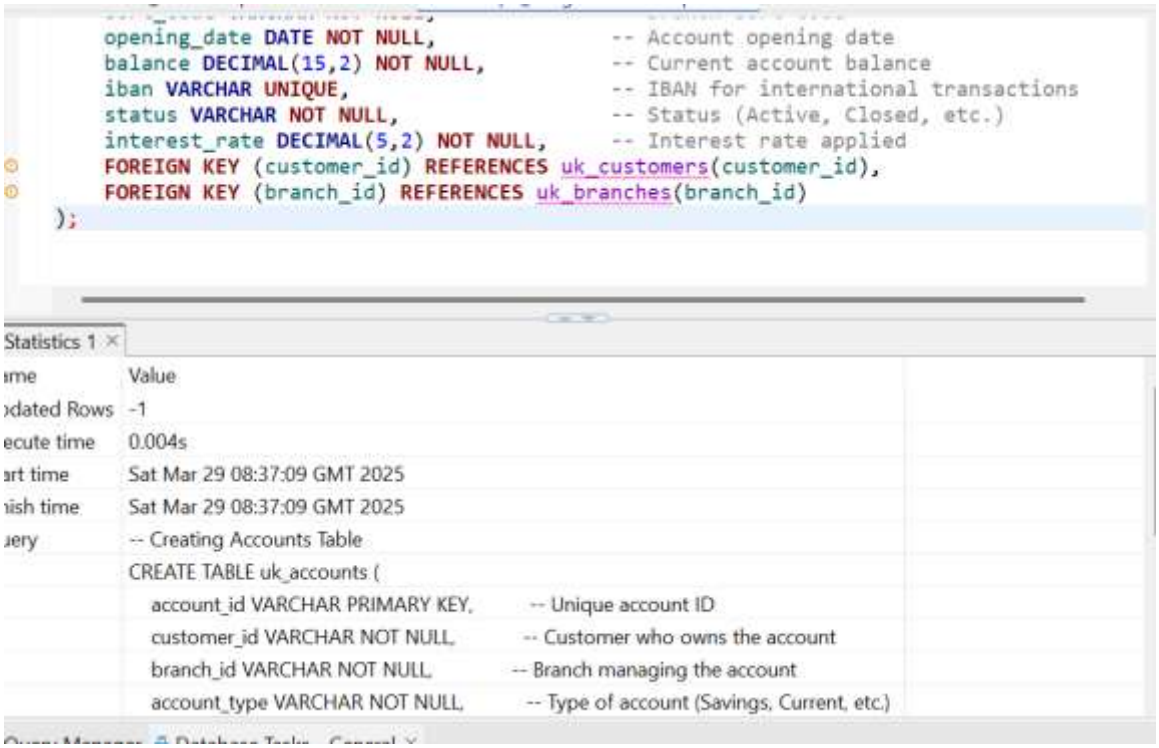


```
    last_name VARCHAR NOT NULL,              -- Employee's last name
    role VARCHAR NOT NULL,                   -- Role or position in branch
    branch_id VARCHAR NOT NULL,              -- Branch where the employee works
    hire_date DATE NOT NULL,                 -- Hiring date
    salary DECIMAL(10,2) NOT NULL,           -- Employee's salary
    email VARCHAR UNIQUE NOT NULL,           -- Unique email address
    phone VARCHAR NOT NULL,                  -- Contact phone number
    FOREIGN KEY (branch_id) REFERENCES uk_branches(branch_id)
);
```

Statistics 1

ame	Value
pdated Rows	-1
ecute time	0.004s
art time	Sat Mar 29 08:36:45 GMT 2025
nish time	Sat Mar 29 08:36:45 GMT 2025
uery	-- Creating Employees Table
	CREATE TABLE uk_employees (
	employee_id VARCHAR PRIMARY KEY, -- Unique ID for each employee
	first_name VARCHAR NOT NULL, -- Employee's first name
	last_name VARCHAR NOT NULL, -- Employee's last name
	role VARCHAR NOT NULL, -- Role or position in branch

Screenshot: employees table.png



Screenshot: accounts table.png

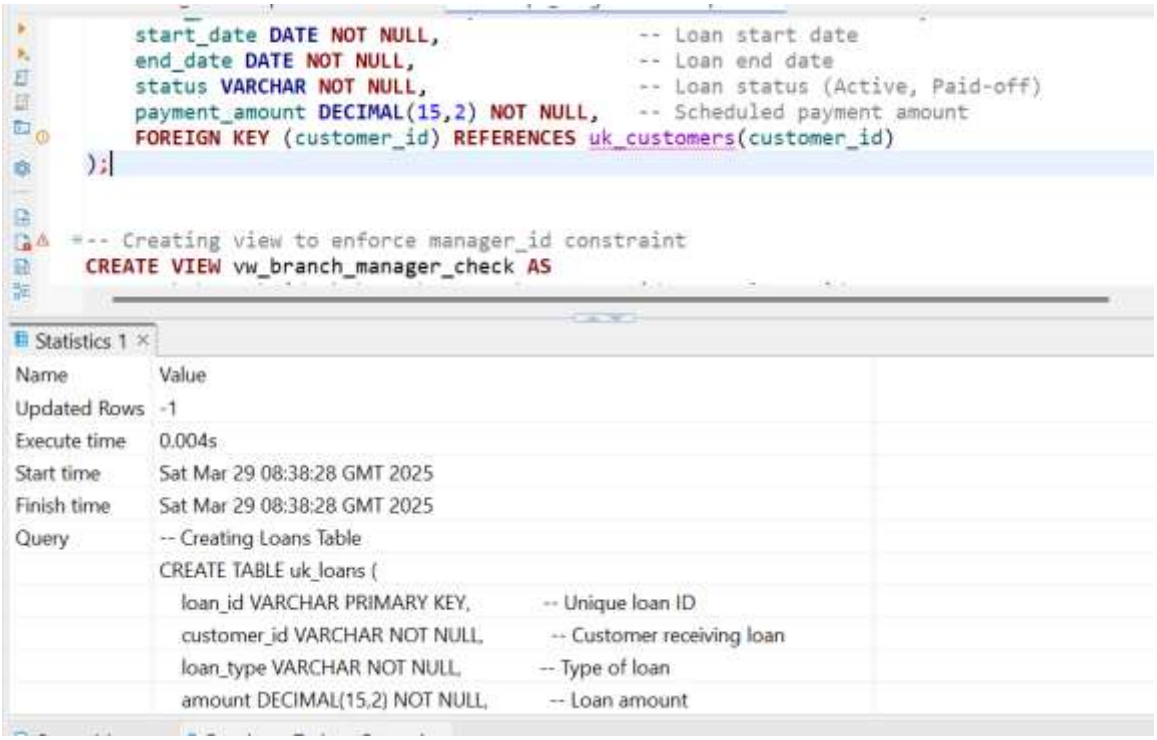
```
    sort_code, opening_date, balance, iban, status, interest_rate
)
SELECT
    account_id, customer_id, branch_id, account_type, account_number,
    sort_code, opening_date, balance, iban, status, interest_rate
FROM read_csv_auto('C:\Users\sambr\OneDrive\Desktop\ucd\accounts.csv');

-- Inserting loan data from CSV (depends on customers table)
INSERT INTO uk_loans (
    loan_id, customer_id, loan_type, amount, interest_rate,
```

Statistics 1

Name	Value
Updated Rows	1500
Execute time	0.062s
Start time	Sat Mar 29 08:40:25 GMT 2025
Finish time	Sat Mar 29 08:40:25 GMT 2025
Query	-- Inserting account data from CSV (depends on customers and branches tables)
	INSERT INTO uk_accounts (
	account_id, customer_id, branch_id, account_type, account_number,
	sort_code, opening_date, balance, iban, status, interest_rate
	)
	SELECT

Screenshot: insert accounts.png



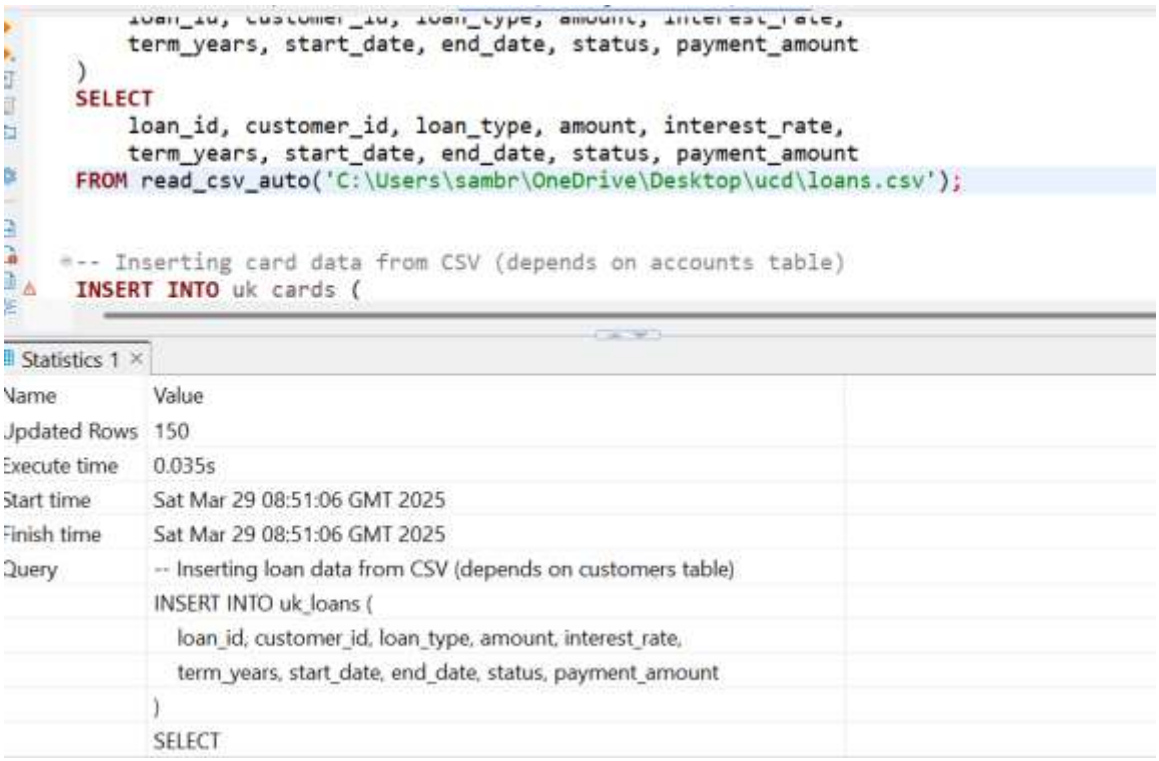
```
    start_date DATE NOT NULL,                -- Loan start date
    end_date DATE NOT NULL,                  -- Loan end date
    status VARCHAR NOT NULL,                 -- Loan status (Active, Paid-off)
    payment_amount DECIMAL(15,2) NOT NULL,   -- Scheduled payment amount
    FOREIGN KEY (customer_id) REFERENCES uk_customers(customer_id)
);

-- Creating view to enforce manager_id constraint
CREATE VIEW vw_branch_manager_check AS
```

Statistics 1

Name	Value
Updated Rows	-1
Execute time	0.004s
Start time	Sat Mar 29 08:38:28 GMT 2025
Finish time	Sat Mar 29 08:38:28 GMT 2025
Query	-- Creating Loans Table
	CREATE TABLE uk_loans (
	loan_id VARCHAR PRIMARY KEY, -- Unique loan ID
	customer_id VARCHAR NOT NULL, -- Customer receiving loan
	loan_type VARCHAR NOT NULL, -- Type of loan
	amount DECIMAL(15,2) NOT NULL, -- Loan amount

Screenshot: loans table.png



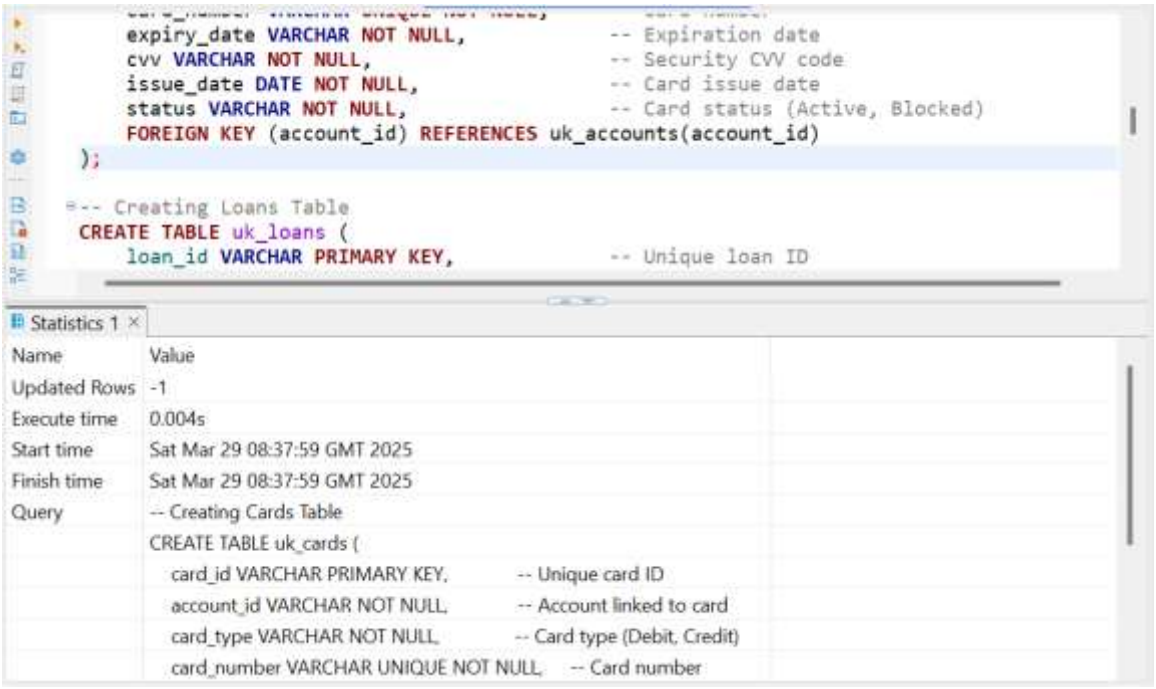
```
    loan_id, customer_id, loan_type, amount, interest_rate,
    term_years, start_date, end_date, status, payment_amount
)
SELECT
    loan_id, customer_id, loan_type, amount, interest_rate,
    term_years, start_date, end_date, status, payment_amount
FROM read_csv_auto('C:\Users\sambr\OneDrive\Desktop\ucd\loans.csv');


-- Inserting card data from CSV (depends on accounts table)
INSERT INTO uk cards (
```

Statistics 1

Name	Value
Updated Rows	150
Execute time	0.035s
Start time	Sat Mar 29 08:51:06 GMT 2025
Finish time	Sat Mar 29 08:51:06 GMT 2025
Query	-- Inserting loan data from CSV (depends on customers table)
	INSERT INTO uk_loans (
	loan_id, customer_id, loan_type, amount, interest_rate,
	term_years, start_date, end_date, status, payment_amount
	)
	SELECT

Screenshot: insert loans.png



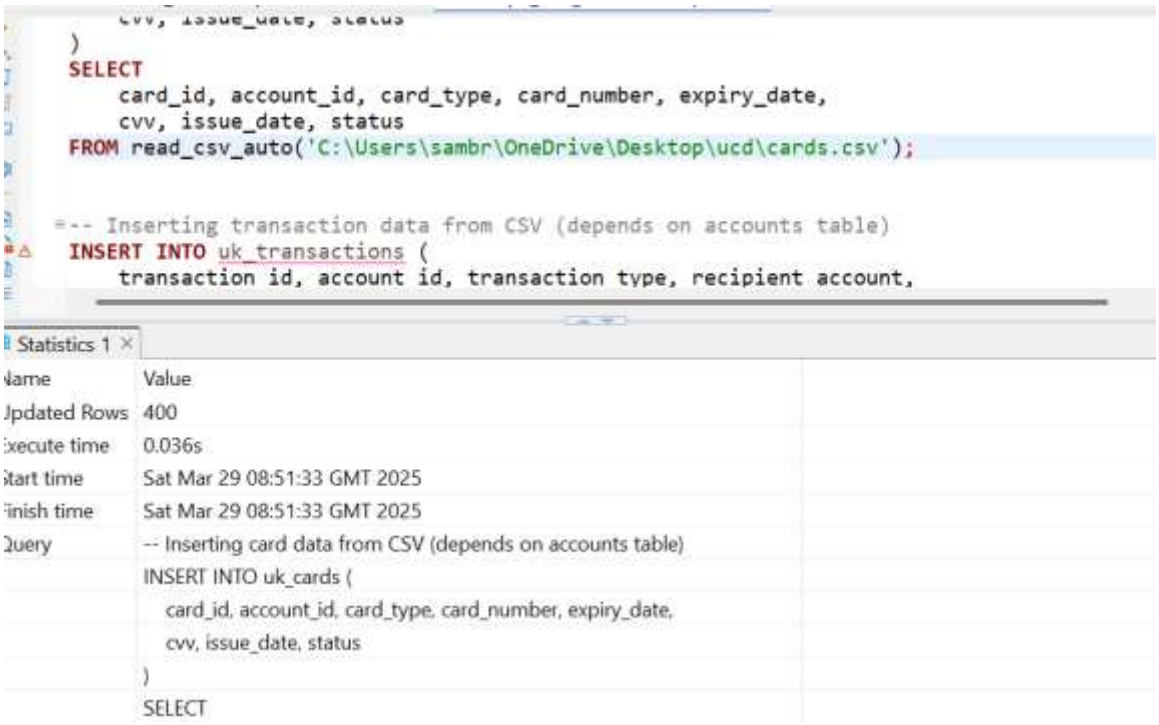
```
    expiry_date VARCHAR NOT NULL,              -- Expiration date
    cvv VARCHAR NOT NULL,                      -- Security CVV code
    issue_date DATE NOT NULL,                  -- Card issue date
    status VARCHAR NOT NULL,                   -- Card status (Active, Blocked)
    FOREIGN KEY (account_id) REFERENCES uk_accounts(account_id)
);

-- Creating Loans Table
CREATE TABLE uk_loans (
    loan_id VARCHAR PRIMARY KEY,               -- Unique loan ID
```

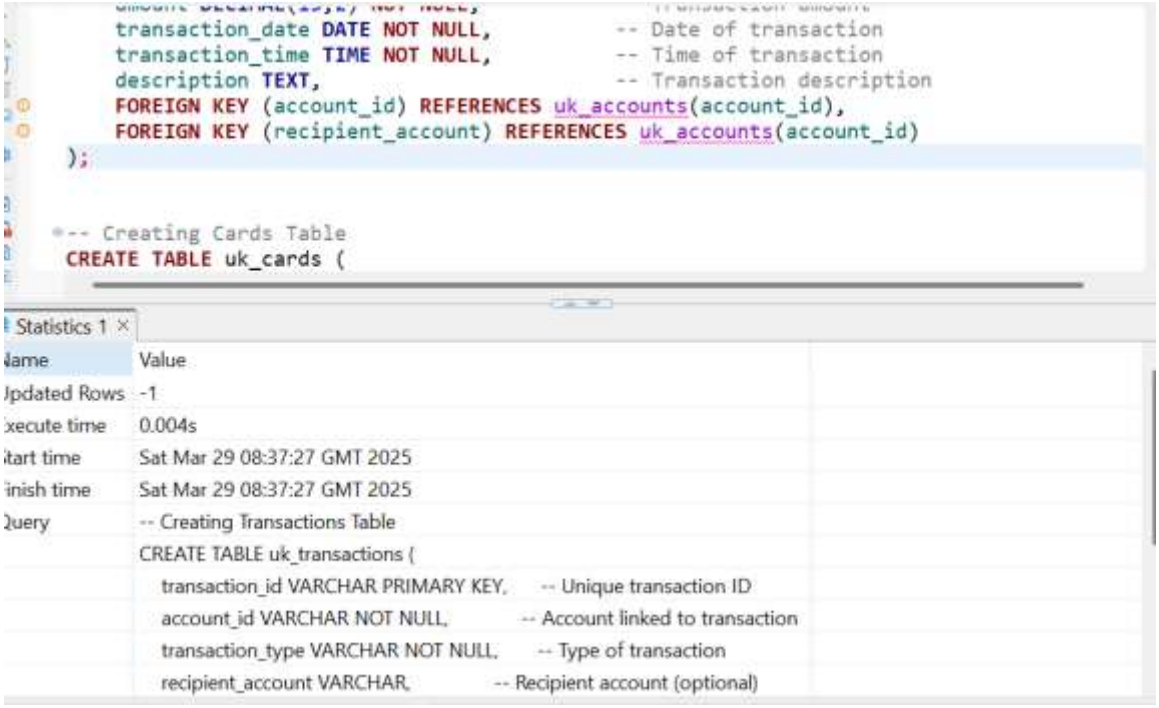
Statistics 1

Name	Value
Updated Rows	-1
Execute time	0.004s
Start time	Sat Mar 29 08:37:59 GMT 2025
Finish time	Sat Mar 29 08:37:59 GMT 2025
Query	-- Creating Cards Table
	CREATE TABLE uk_cards (
	card_id VARCHAR PRIMARY KEY, -- Unique card ID
	account_id VARCHAR NOT NULL, -- Account linked to card
	card_type VARCHAR NOT NULL, -- Card type (Debit, Credit)
	card_number VARCHAR UNIQUE NOT NULL, -- Card number

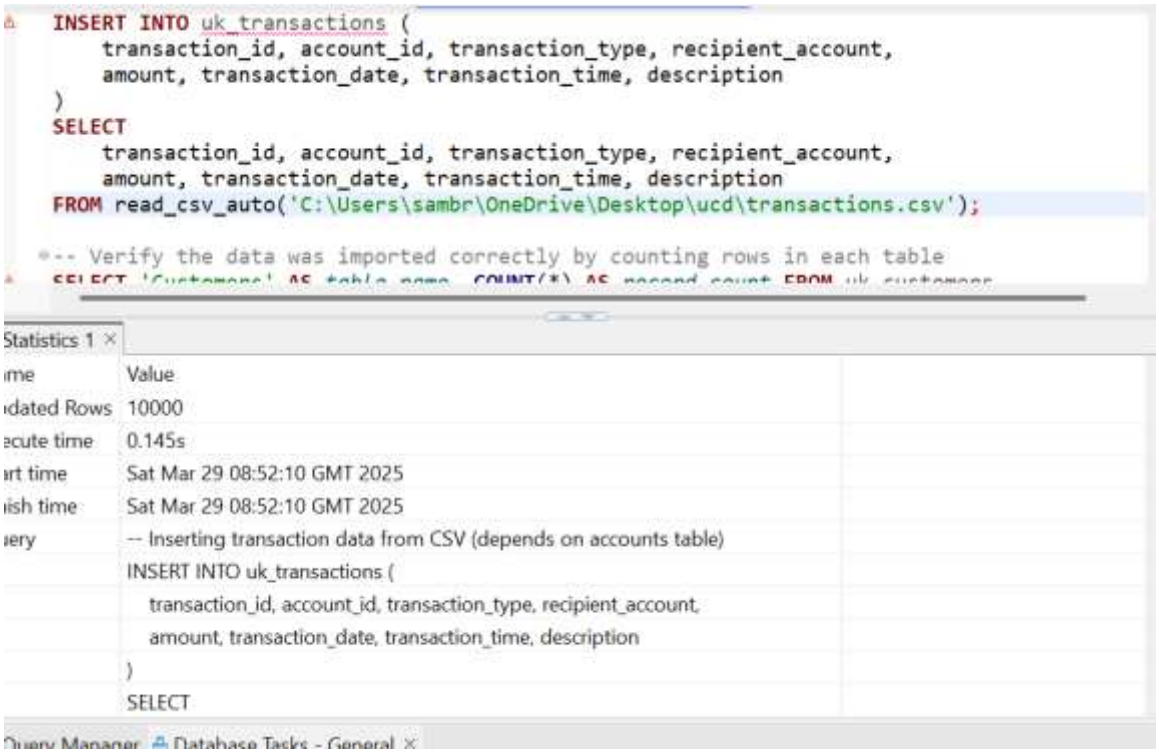
Screenshot: cards table.png



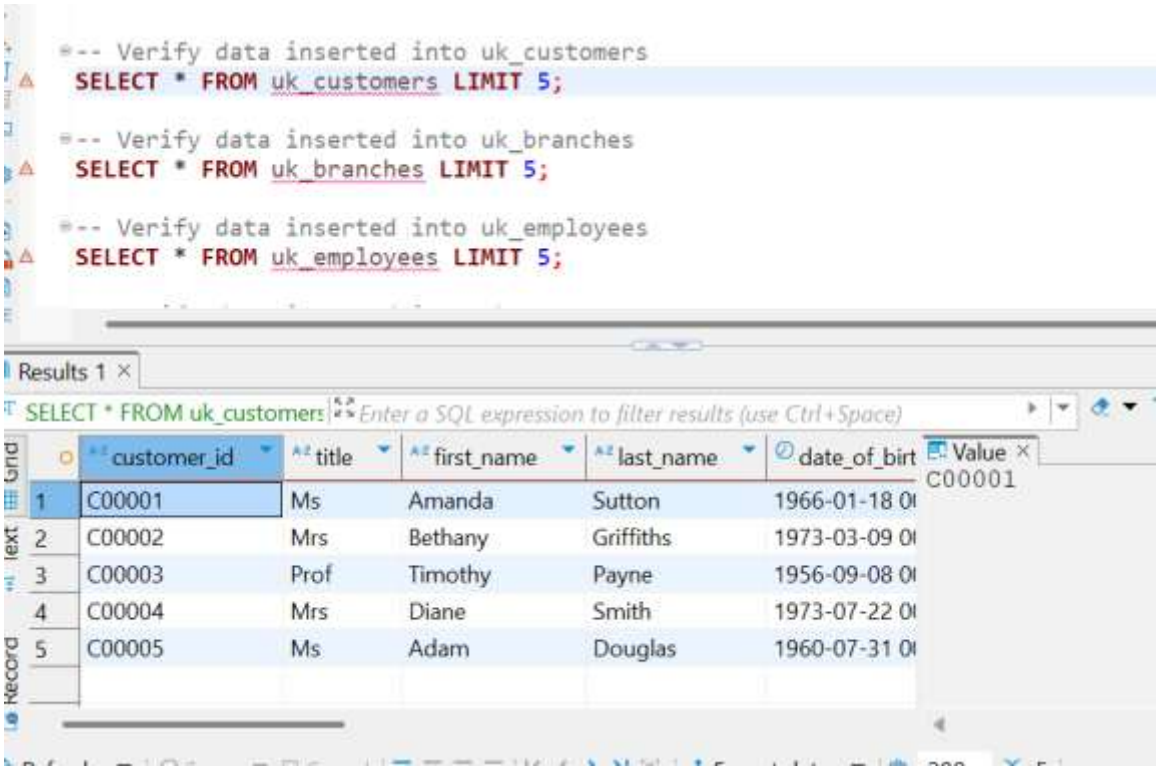
Screenshot: insertcards.png



Screenshot: transactions table.png



Screenshot: insert transactions.png



Screenshot: verify customers.png

```sql
-- Verify data inserted into uk_branches
SELECT * FROM uk_branches LIMIT 5;

-- Verify data inserted into uk_employees
SELECT * FROM uk_employees LIMIT 5;
```

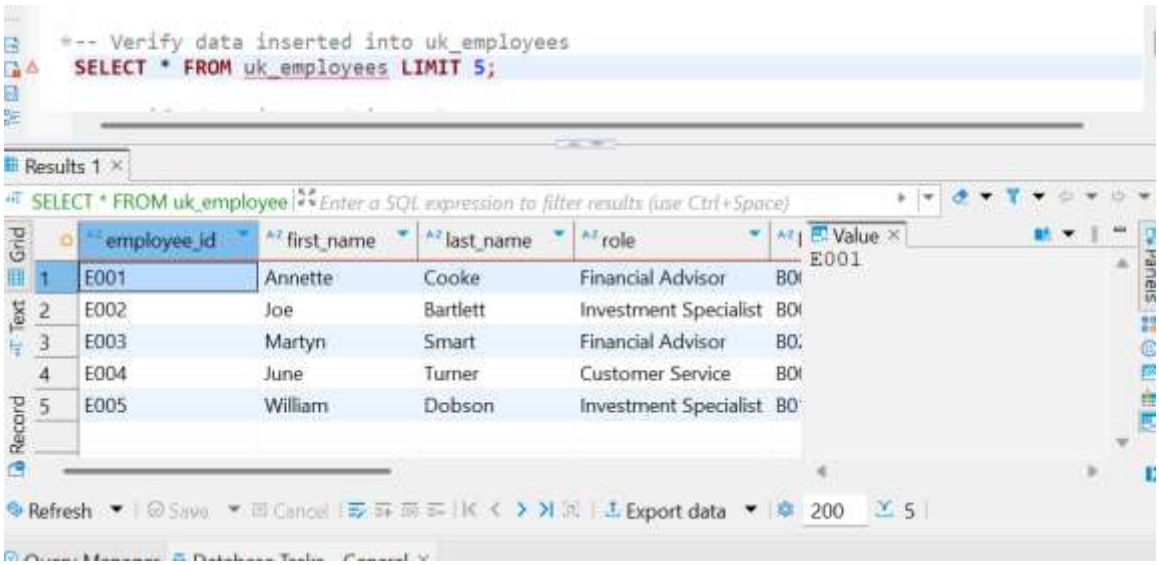
esults 1 ×

ELECT * FROM uk_branches l | Enter a SQL expression to filter results (use Ctrl+Space)

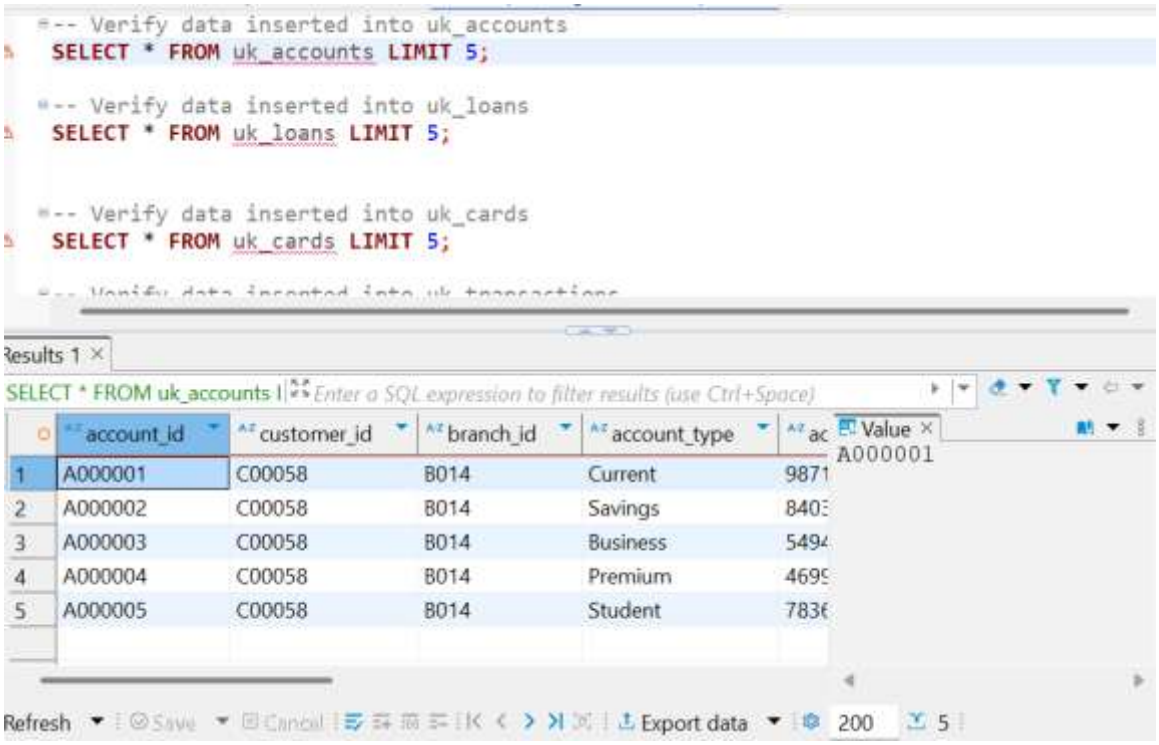
	branch_id	branch_name	address	city	postcode
1	B001	London - Royal	Studio 60N¶Nich	London	ME34 6NM
2	B002	Edinburgh - Central	9 Blake forge	Edinburgh	MK4 9WK
3	B003	Newcastle - Royal	03 Norris alley	Newcastle	GL87 6GB
4	B004	Bristol - East	576 Barnes vista	Bristol	PH57 5RE
5	B005	Glasgow - City Centre	Flat 00¶Nicola m	Glasgow	DN67 5WH

Value ×
B001

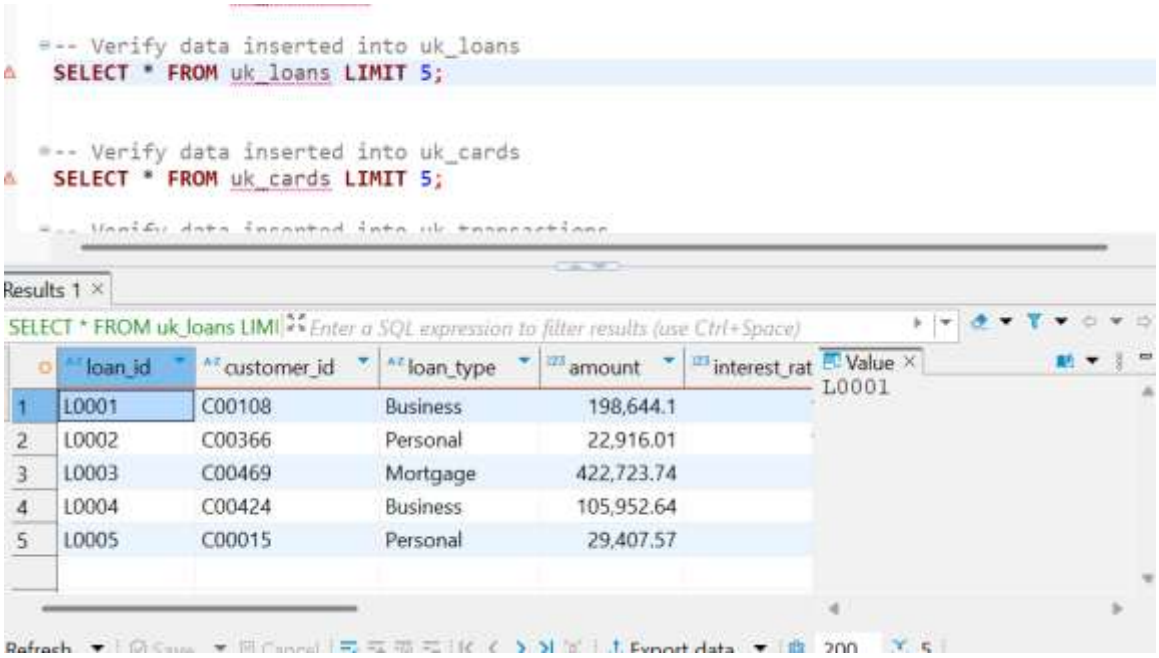
Screenshot: verify branches.png



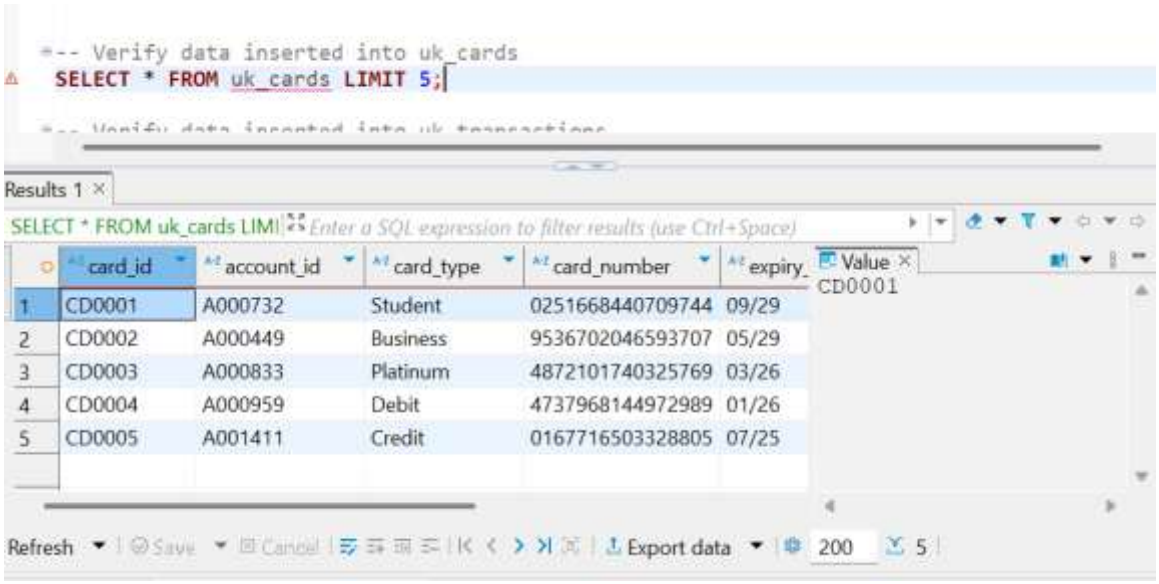
Screenshot: verify employees.png



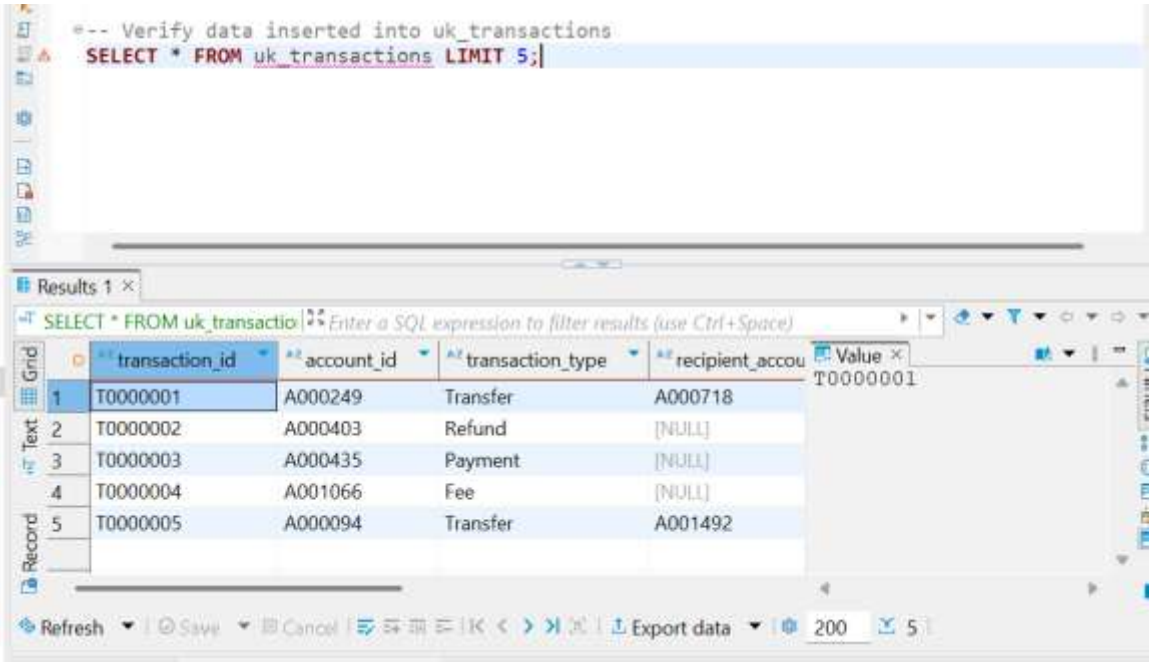
Screenshot: verify accounts.png



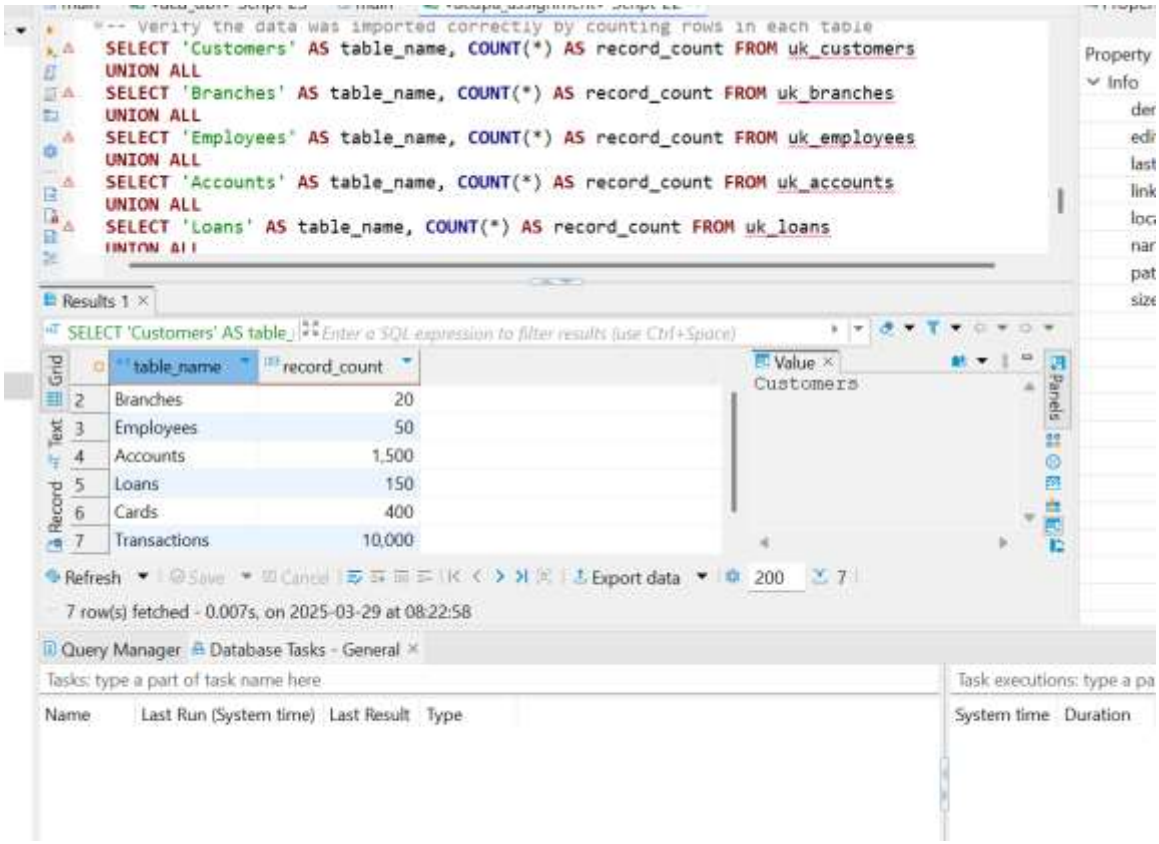
Screenshot: verify loans.png



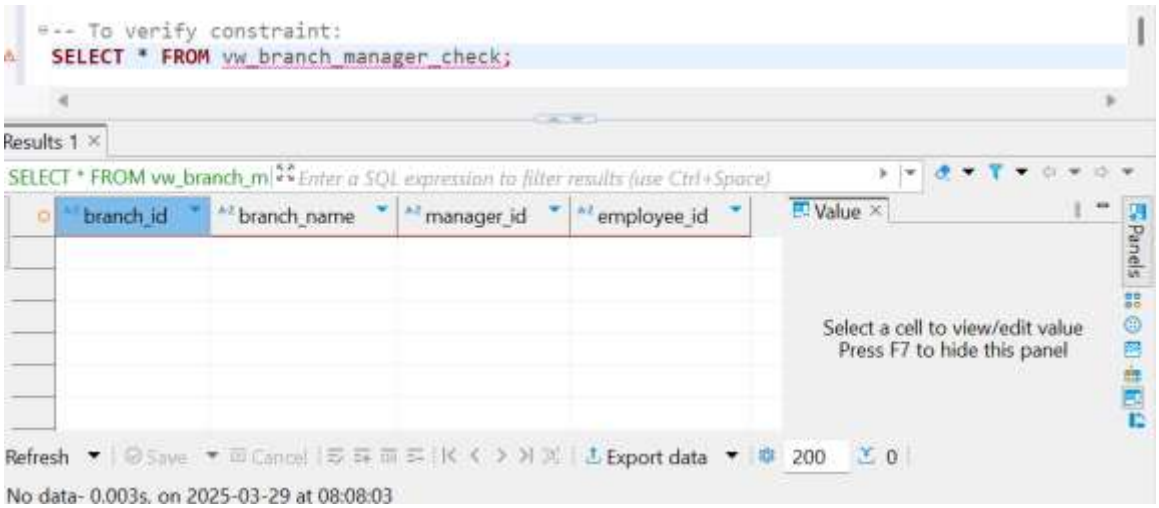
Screenshot: verify cards.png



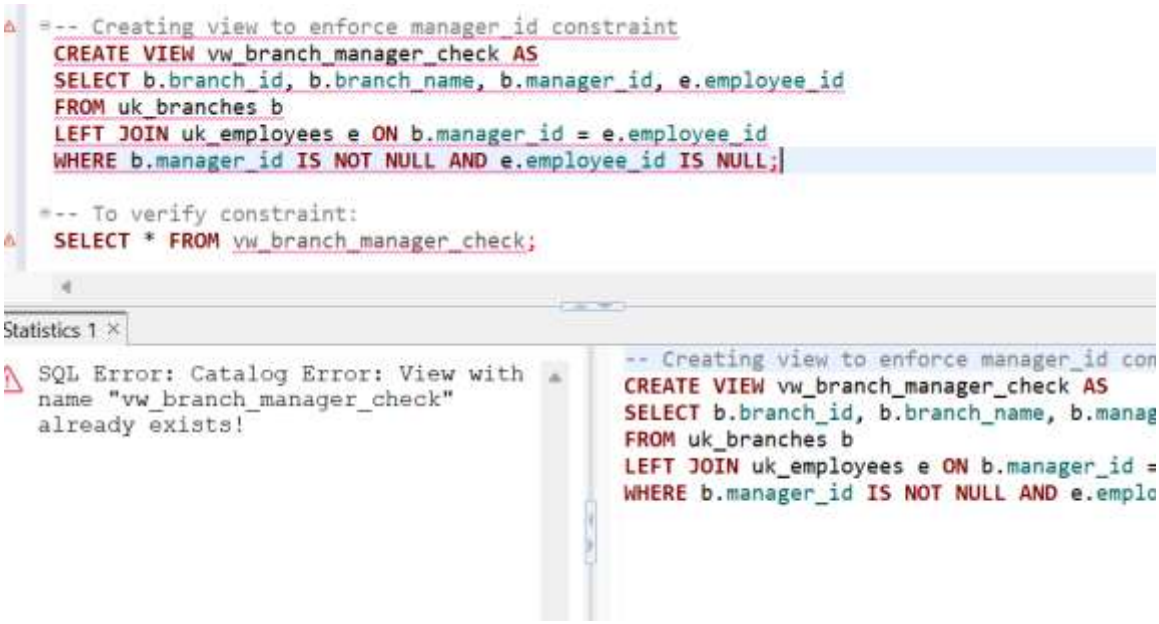
Screenshot: verify transactions.png



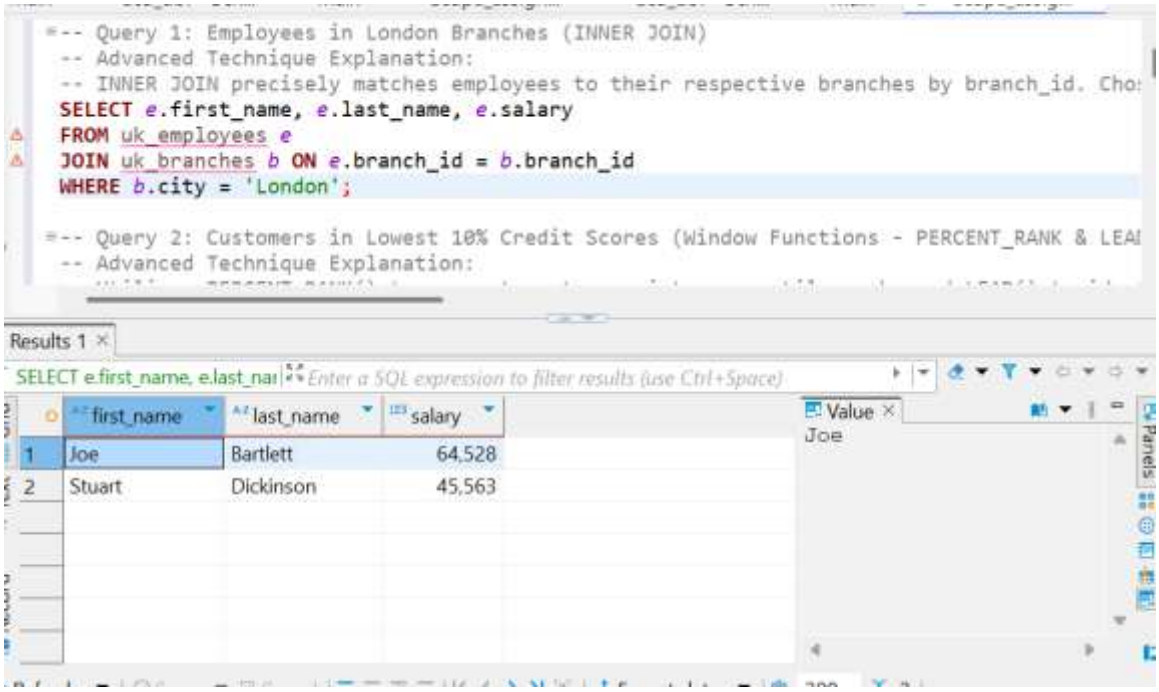
Screenshot: verfication of tables and data through union.png



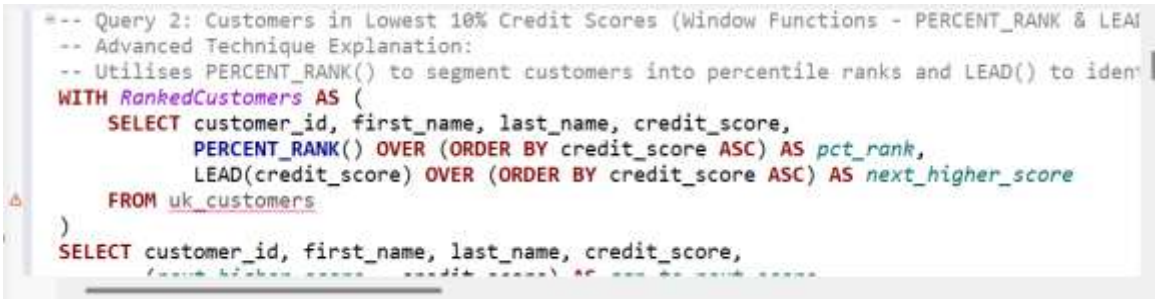
Screenshot: verfication of view creation.png



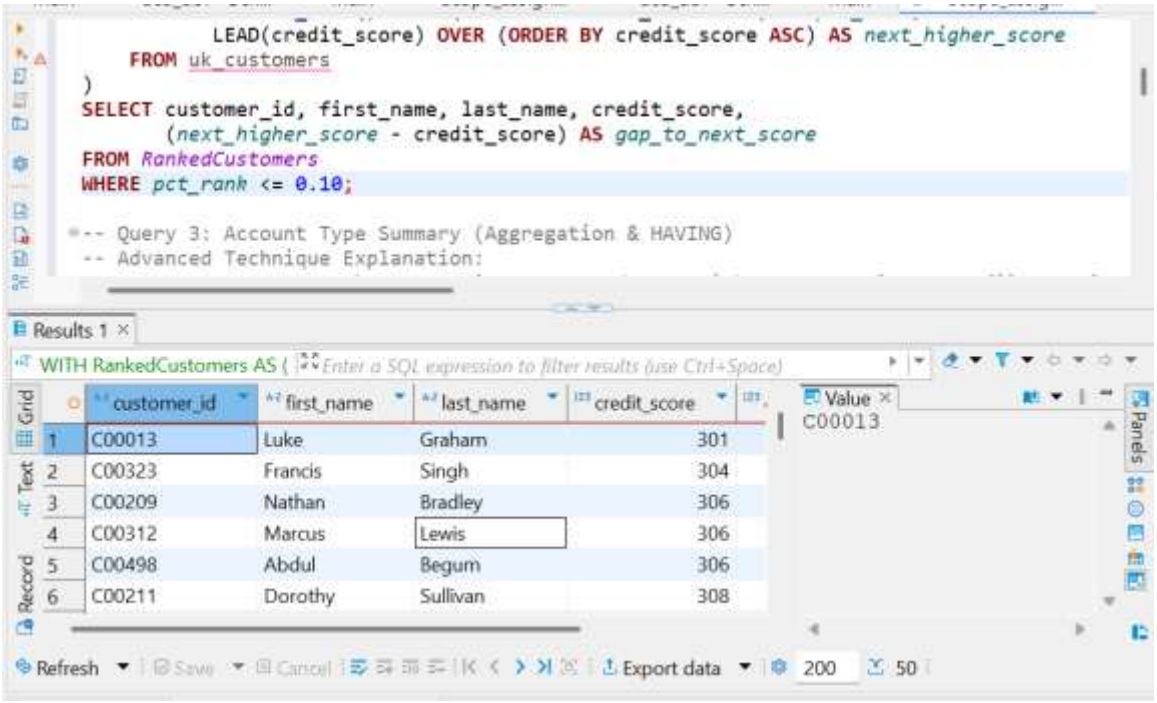
Screenshot: manager view creation.png



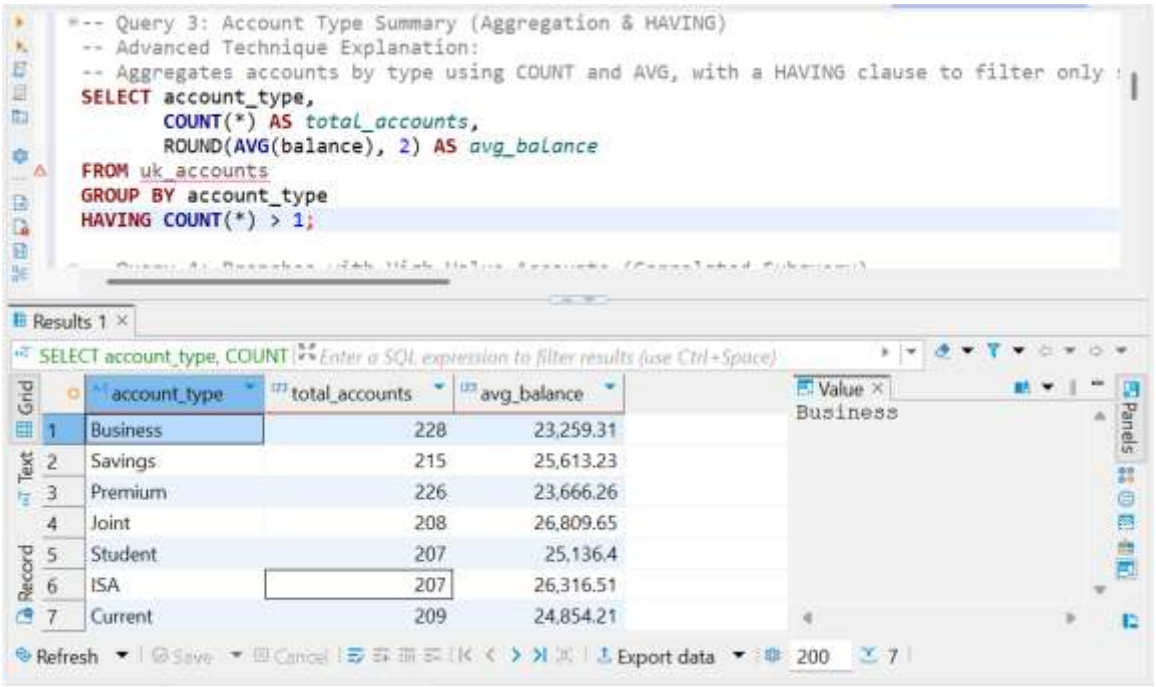
Screenshot: query 1.png



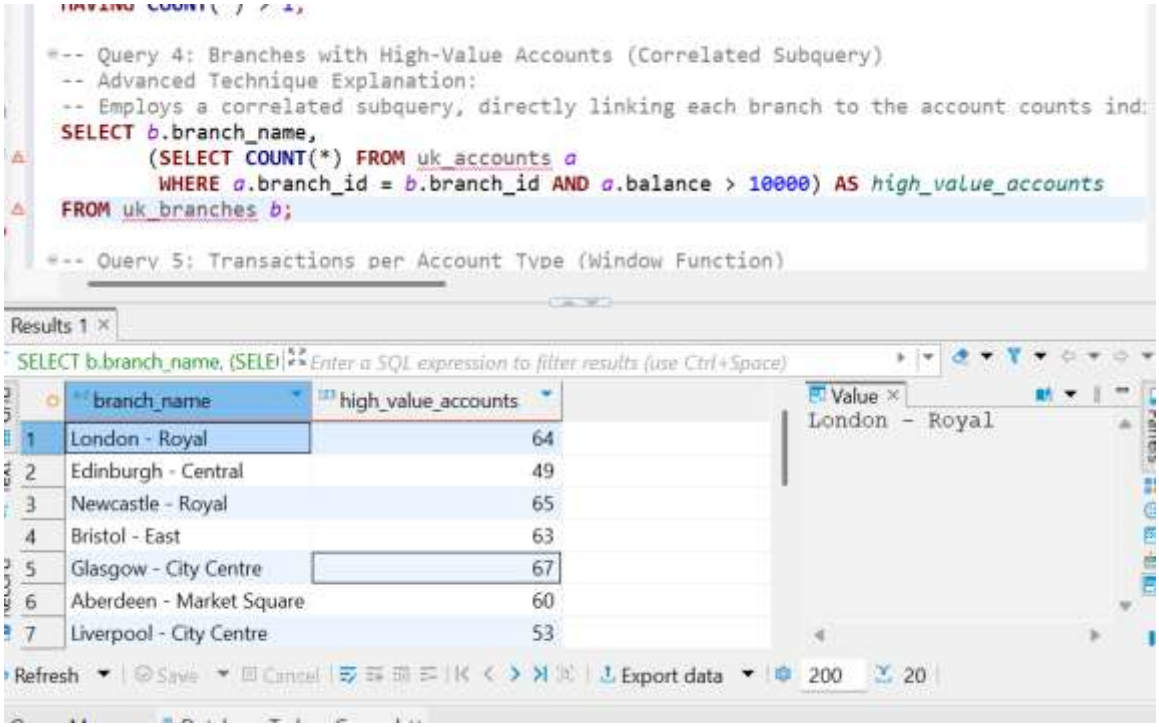
Screenshot: query2.png



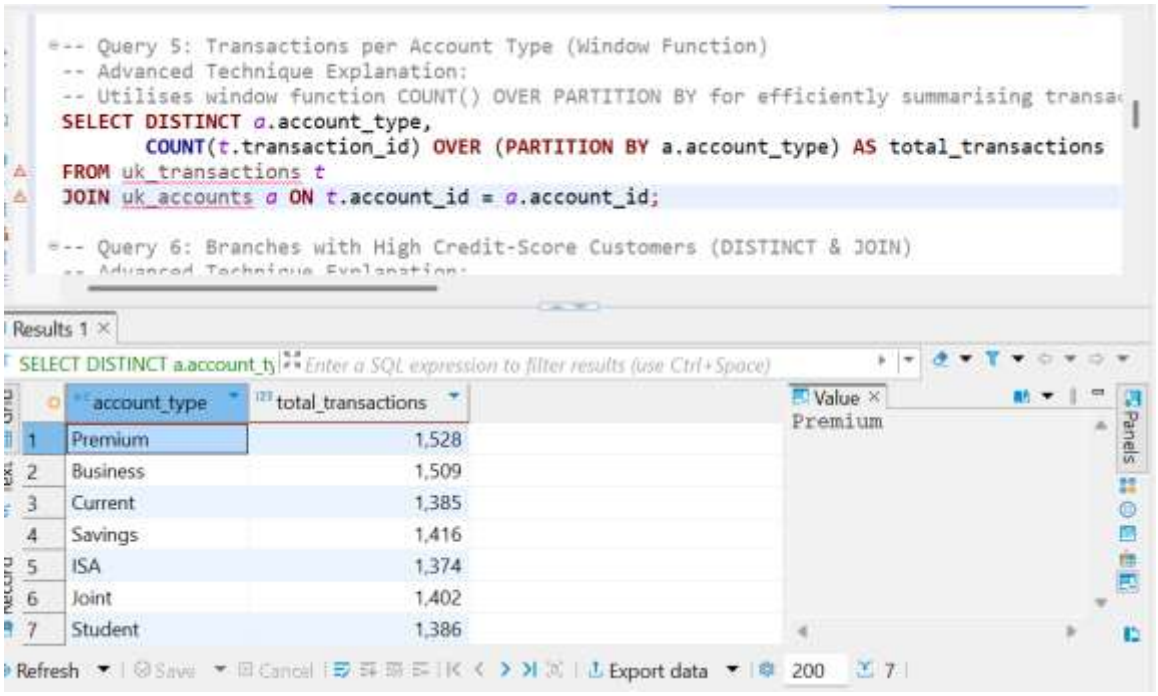
Screenshot: query2-1.png



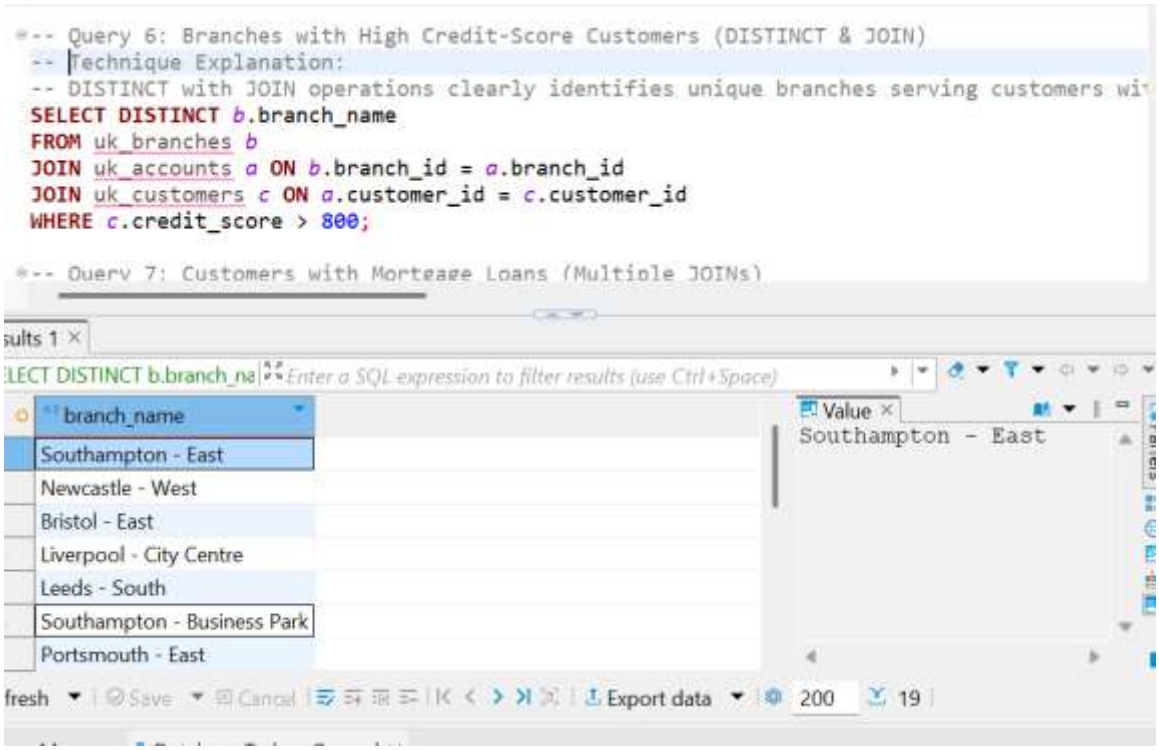
Screenshot: query3.png



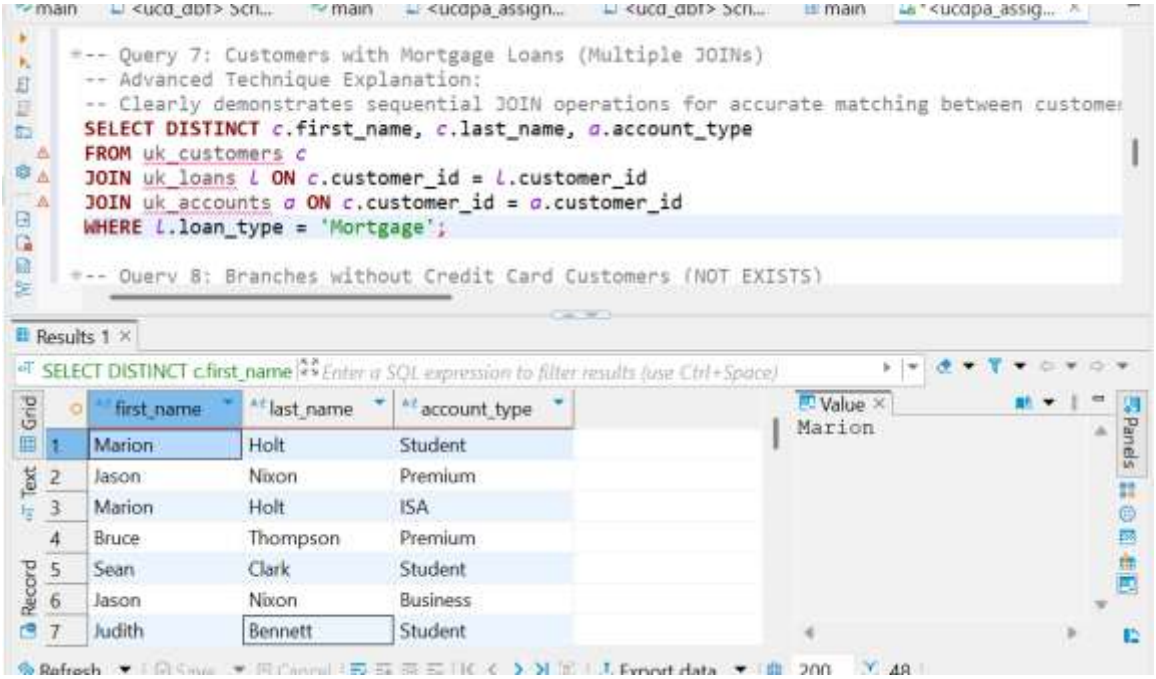
Screenshot: query4.png



Screenshot: query5.png



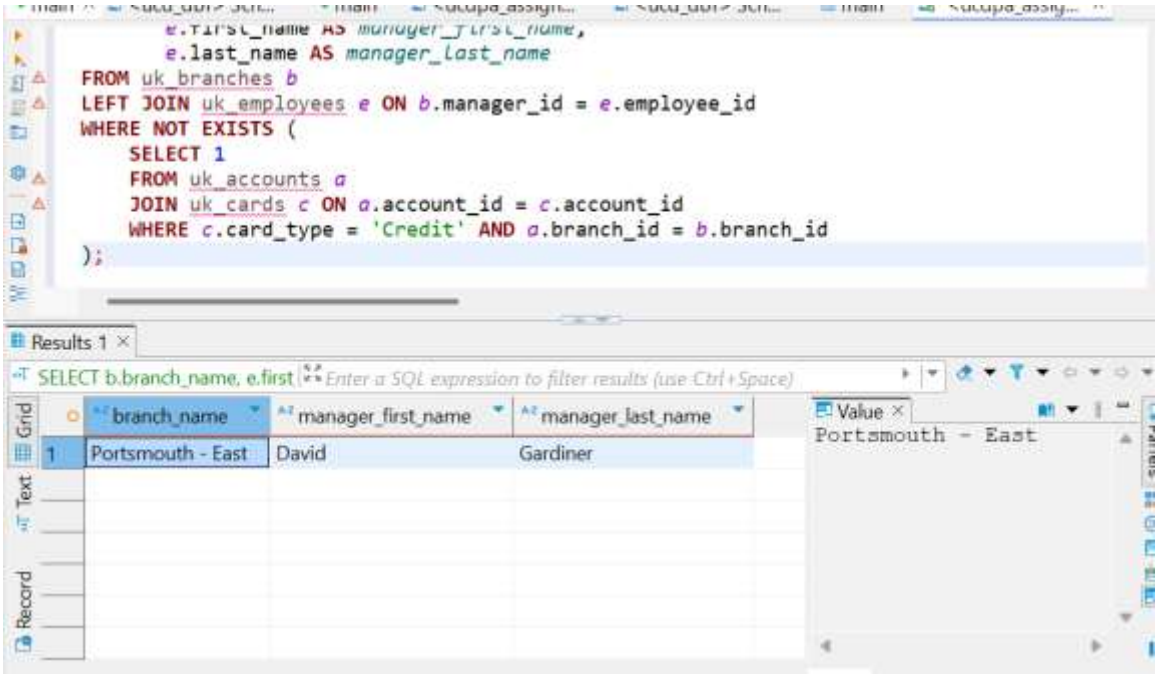
Screenshot: query6.png



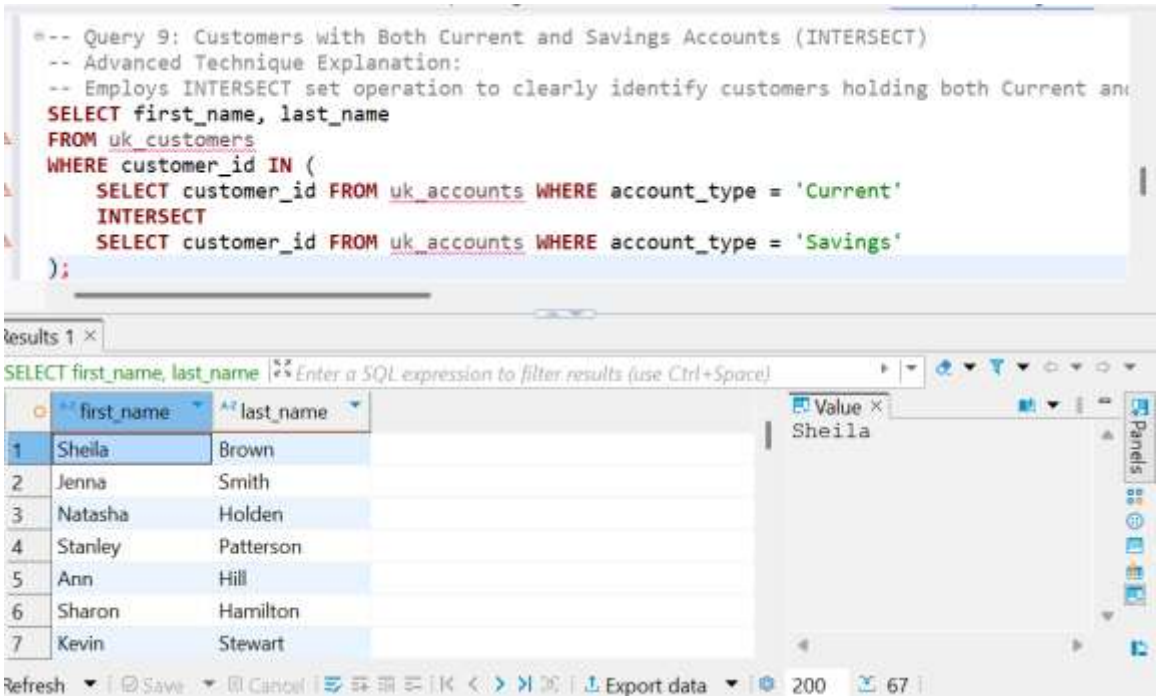
Screenshot: query7.png

```sql
-- Query 8: Branches without Credit Card Customers (NOT EXISTS)
-- Advanced Technique Explanation:
-- Utilises NOT EXISTS as an anti-join method, clearly and efficiently identifying branc
SELECT b.branch_name,
       e.first_name AS manager_first_name,
       e.last_name AS manager_last_name
FROM uk_branches b
LEFT JOIN uk_employees e ON b.manager_id = e.employee_id
WHERE NOT EXISTS (
    SELECT 1
```

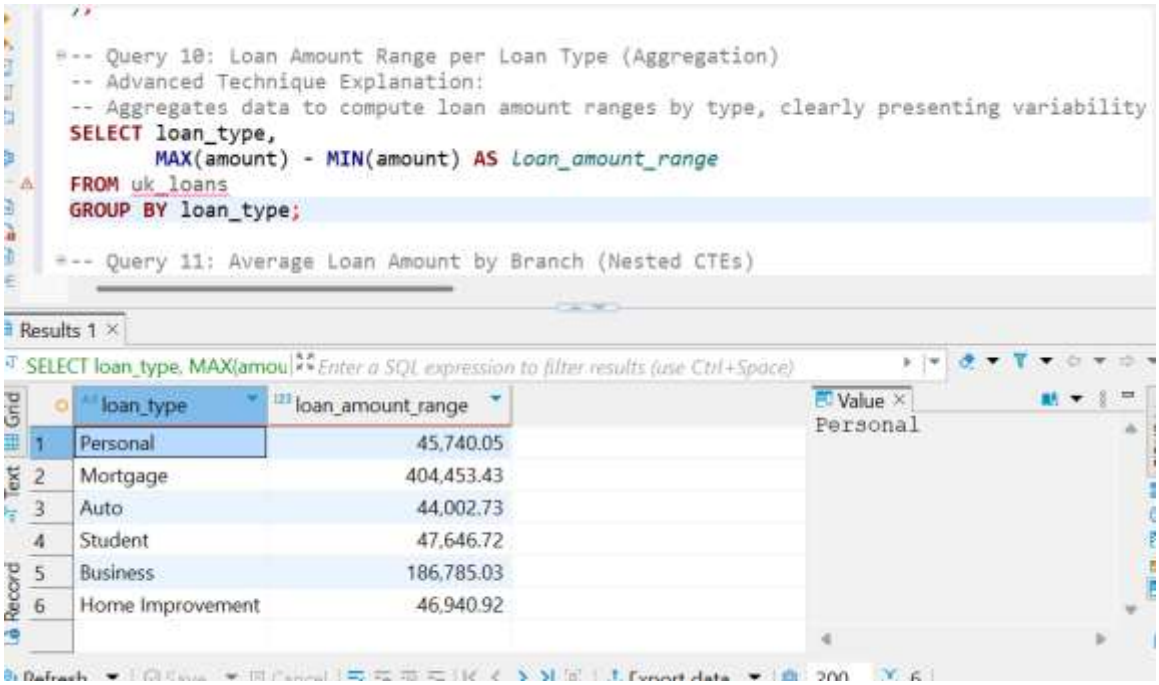
Screenshot: query8.png



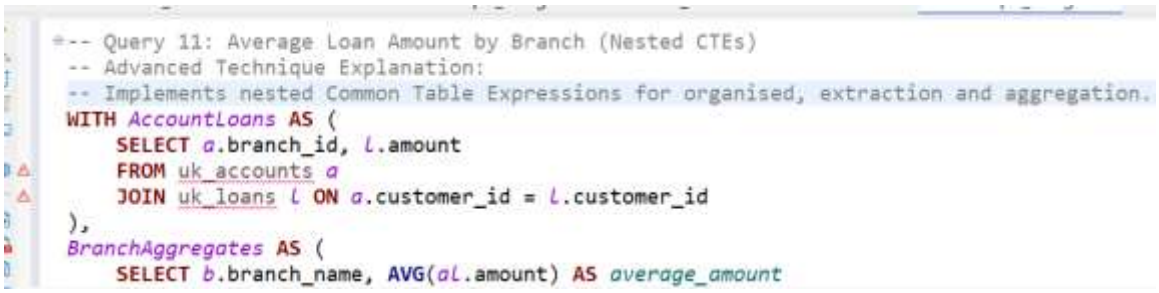
Screenshot: query8-1.png



Screenshot: query9.png

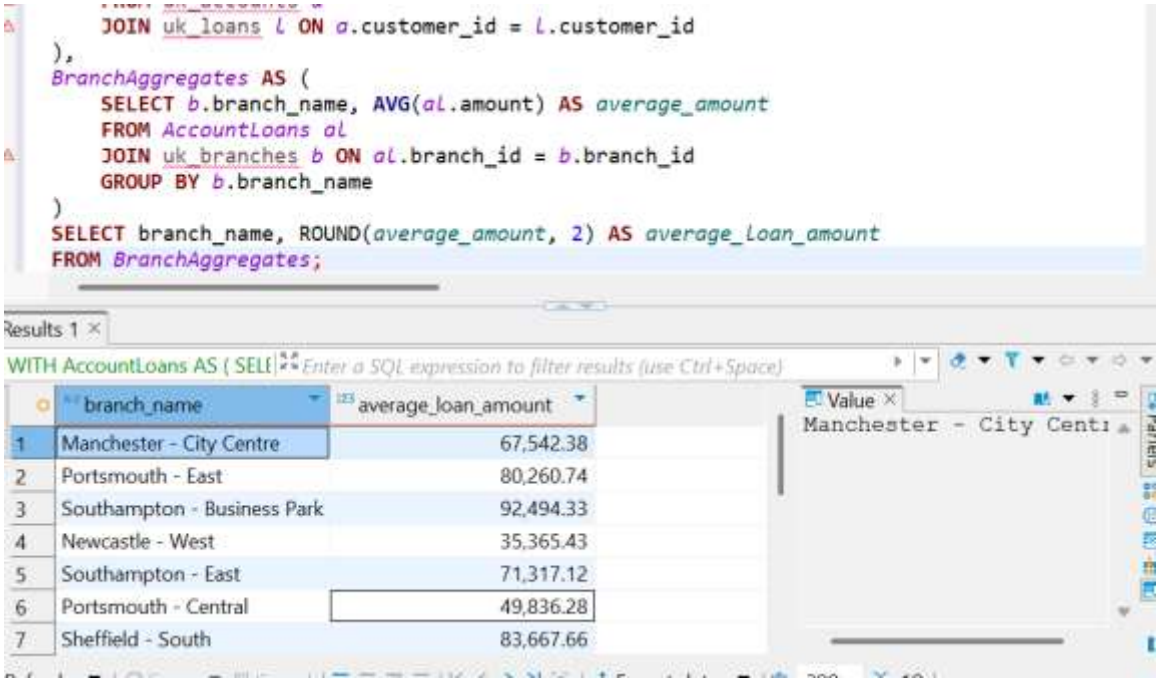


Screenshot: query10.png

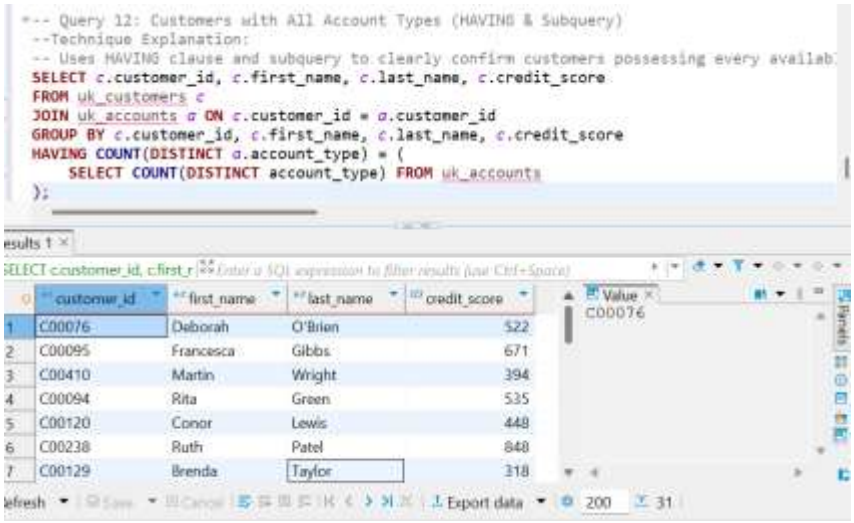


```sql
-- Query 11: Average Loan Amount by Branch (Nested CTEs)
-- Advanced Technique Explanation:
-- Implements nested Common Table Expressions for organised, extraction and aggregation.
WITH AccountLoans AS (
    SELECT a.branch_id, l.amount
    FROM uk_accounts a
    JOIN uk_loans l ON a.customer_id = l.customer_id
),
BranchAggregates AS (
    SELECT b.branch_name, AVG(al.amount) AS average_amount
```

Screenshot: query11.png



Screenshot: query11-1.png



Screenshot: query12 remaining comments.png