

# 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

<b>COUNT() OVER()</b>	Window function to count without collapsing rows	Query 5
<b>INTERSECT</b>	Ensures customers have both types of accounts	Query 9
<b>Nested CTEs</b>	Layers queries to organise logic	Query 11
<b>NOT EXISTS</b>	Identifies unmatched rows (anti-join pattern)	Query 8
<b>HAVING with Subquery</b>	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
```

---

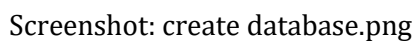
## Query 12

```
-- 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

main <ucd\_dbf> Script-23 main <ucdpa\_assignment> Script-22 x

```
nin VARCHAR UNIQUE NOT NULL,      -- National Insurance Number
address VARCHAR NOT NULL,          -- Home address
city VARCHAR NOT NULL,             -- City of residence
postcode VARCHAR NOT NULL,        -- Postcode of residence
join_date DATE NOT NULL,           -- Date customer joined the bank
credit_score INTEGER               -- Credit score of customer
);
```

--- Creating Branches Table

Statistics 1 x

Name	Value
Updated Rows	-1
Execute time	0.004s
Start time	Sat Mar 29 08:47:33 GMT 2025
Finish time	Sat Mar 29 08:47:33 GMT 2025
Query	-- Creating Customers Table CREATE TABLE uk_customers ( customer_id VARCHAR PRIMARY KEY, -- Unique ID for each customer title VARCHAR NOT NULL, -- Customer's title (Mr., Ms., etc.) first_name VARCHAR NOT NULL, -- First Name of customer last_name VARCHAR NOT NULL, -- Last Name of customer

Query Manager Database Tasks - General x

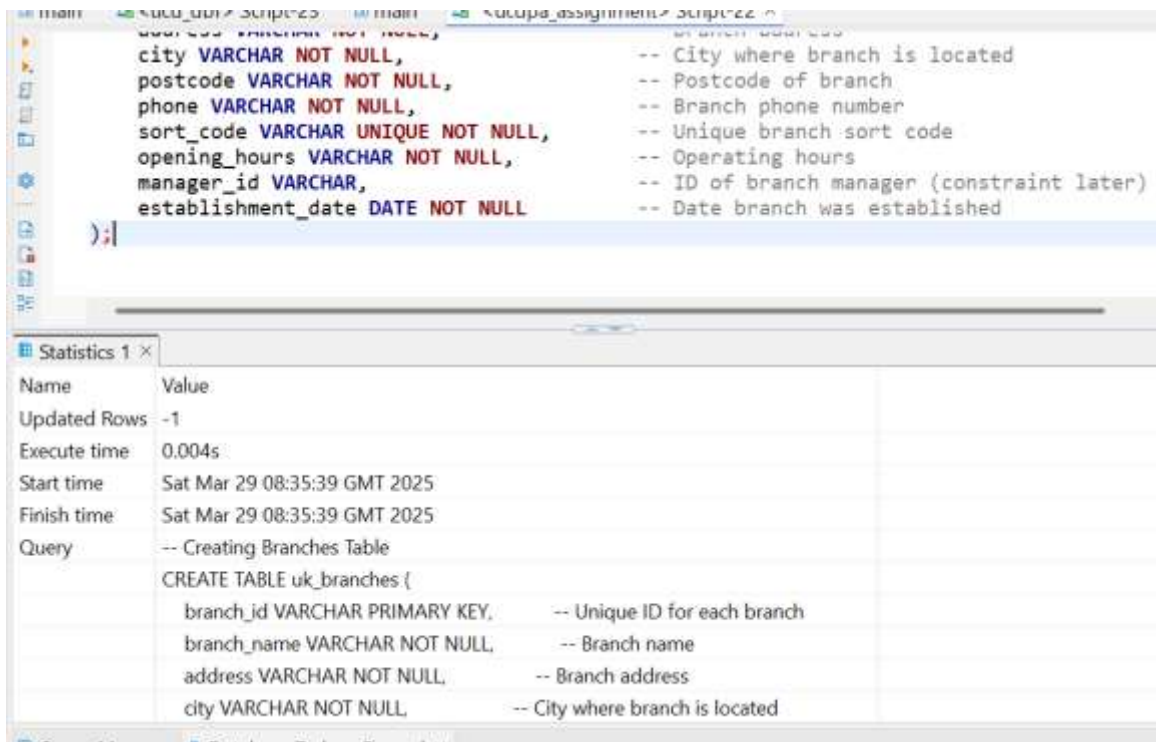
Screenshot: customer table.png

```
▲ 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 constraints)
```

Statistics 1 ×	
name	Value
Updated Rows	500
Execute time	0.022s
Start time	Sat Mar 29 08:39:12 GMT 2025
Finish time	Sat Mar 29 08:39:12 GMT 2025
Query	-- 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 first)
Statistics 1 x	
me	Value
dated Rows	20
ecute 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
	)
	SFI FCT

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

```
opening_date DATE NOT NULL,      -- Account opening date
balance DECIMAL(15,2) NOT NULL,  -- Current account balance
iban VARCHAR UNIQUE,             -- IBAN for international transactions
status VARCHAR NOT NULL,         -- Status (Active, Closed, etc.)
interest_rate DECIMAL(5,2) NOT NULL, -- Interest rate applied
FOREIGN KEY (customer_id) REFERENCES uk_customers(customer_id),
FOREIGN KEY (branch_id) REFERENCES uk_branches(branch_id)
);
```

Statistics 1 ×	Value
Time	
Updated Rows	-1
Execute time	0.004s
Start time	Sat Mar 29 08:37:09 GMT 2025
Finish time	Sat Mar 29 08:37:09 GMT 2025
Query	-- Creating Accounts Table
	CREATE TABLE uk_accounts (
	account_id VARCHAR PRIMARY KEY, -- Unique account ID
	customer_id VARCHAR NOT NULL, -- Customer who owns the account
	branch_id VARCHAR NOT NULL, -- Branch managing the account
	account_type VARCHAR NOT NULL, -- Type of account (Savings, Current, etc.)

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,

```

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
	)
	<b>SELECT</b>
	loan_id, customer_id, loan_type, amount, interest_rate, term_years, start_date, end_date, status, payment_amount
	<b>FROM</b> read_csv_auto('C:\Users\sambr\OneDrive\Desktop\ugd\loans.csv');
	-- Inserting card data from CSV (depends on accounts table)
	<b>INSERT INTO</b> uk_cards (
Statistics 1 x	
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

```
-- Create 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
    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 x

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

	cvv, issue_date, status
	)
	SELECT
	card_id, account_id, card_type, card_number, expiry_date,
	cvv, issue_date, status
	FROM read_csv_auto('C:\Users\sambr\OneDrive\Desktop\ucd\cards.csv');
	-- Inserting transaction data from CSV (depends on accounts table)
	INSERT INTO uk_transactions (
	transaction id, account id, transaction type, recipient account,
Statistics 1	
Name	Value
Updated Rows	400
Execute time	0.036s
Start time	Sat Mar 29 08:51:33 GMT 2025
Finish time	Sat Mar 29 08:51:33 GMT 2025
Query	-- Inserting card data from CSV (depends on accounts table)
	INSERT INTO uk_cards (
	card_id, account_id, card_type, card_number, expiry_date,
	cvv, issue_date, status
	)
	SELECT

Screenshot: insertcards.png

```
CREATE TABLE uk_transactions (
    transaction_id VARCHAR(255) NOT NULL,
    transaction_date DATE NOT NULL,
    transaction_time TIME NOT NULL,
    description TEXT,
    FOREIGN KEY (account_id) REFERENCES uk_accounts(account_id),
    FOREIGN KEY (recipient_account) REFERENCES uk_accounts(account_id)
);
```

-- Creating Cards Table

```
CREATE TABLE uk_cards (
```

---

Statistics 1 x

Name	Value
Updated Rows	-1
Execute time	0.004s
Start time	Sat Mar 29 08:37:27 GMT 2025
Finish time	Sat Mar 29 08:37:27 GMT 2025
Query	-- Creating Transactions Table
	CREATE TABLE uk_transactions (
	transaction_id VARCHAR PRIMARY KEY, -- Unique transaction ID
	account_id VARCHAR NOT NULL, -- Account linked to transaction
	transaction_type VARCHAR NOT NULL, -- Type of transaction
	recipient_account VARCHAR, -- Recipient account (optional)

Screenshot: transactions table.png

```

INSERT INTO uk_transactions (
    transaction_id, account_id, transaction_type, recipient_account,
    amount, transaction_date, transaction_time, description
)
SELECT
    transaction_id, account_id, transaction_type, recipient_account,
    amount, transaction_date, transaction_time, description
FROM read_csv_auto('C:\Users\sambr\OneDrive\Desktop\ucd\transactions.csv');

-- Verify the data was imported correctly by counting rows in each table
SELECT 'Customers' AS table_name, COUNT(*) AS record_count FROM uk_customers

```

Statistics 1	Value
Time	
Updated Rows	10000
Execute time	0.145s
Start time	Sat Mar 29 08:52:10 GMT 2025
Finish time	Sat Mar 29 08:52:10 GMT 2025
Query	-- Inserting transaction data from CSV (depends on accounts table)
	INSERT INTO uk_transactions (
	transaction_id, account_id, transaction_type, recipient_account,
	amount, transaction_date, transaction_time, description
	)
	SELECT

Query Manager: Database Tools - General

Screenshot: insert transactions.png

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

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

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

Results 1 x

SELECT \* FROM uk\_customers

	customer_id	title	first_name	last_name	date_of_birth
1	C00001	Ms	Amanda	Sutton	1966-01-18 01
2	C00002	Mrs	Bethany	Griffiths	1973-03-09 01
3	C00003	Prof	Timothy	Payne	1956-09-08 01
4	C00004	Mrs	Diane	Smith	1973-07-22 01
5	C00005	Ms	Adam	Douglas	1960-07-31 01

Screenshot: verify customers.png

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

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

Results 1 x

SELECT \* FROM uk\_branches | Enter a SQL expression to filter results (use Ctrl+Space)

	branch_id	branch_name	address	city	postcode
1	B001	London - Royal	Studio 60N	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 00N	Glasgow	DN67 5WH

Screenshot: verify branches.png



-- Verify data inserted into uk\_employees  
**SELECT \* FROM uk\_employees LIMIT 5;**

Results 1 ×

SELECT \* FROM uk\_employee | Enter a SQL expression to filter results (use Ctrl+Space)

	employee_id	first_name	last_name	role	Value
1	E001	Annette	Cooke	Financial Advisor	B01
2	E002	Joe	Bartlett	Investment Specialist	B01
3	E003	Martyn	Smart	Financial Advisor	B01
4	E004	June	Turner	Customer Service	B01
5	E005	William	Dobson	Investment Specialist	B01

Refresh | Save | Cancel | Export data | 200 | 5 |

Screenshot: verify employees.png

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

-- Verify data inserted into uk_loans
SELECT * FROM uk_loans LIMIT 5;

-- Verify data inserted into uk_cards
SELECT * FROM uk_cards LIMIT 5;

-- Verify data inserted into uk_transactions
```

Results 1 x

SELECT \* FROM uk\_accounts | *Enter a SQL expression to filter results (use Ctrl+Space)*

	account_id	customer_id	branch_id	account_type	ac
1	A000001	C00058	B014	Current	9871
2	A000002	C00058	B014	Savings	8403
3	A000003	C00058	B014	Business	5494
4	A000004	C00058	B014	Premium	4695
5	A000005	C00058	B014	Student	7836

Refresh Save Cancel Export data 200 5

Screenshot: verify accounts.png

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

-- Verify data inserted into uk_cards
SELECT * FROM uk_cards LIMIT 5;

-- Verify data inserted into uk_transactions
```

Results 1 x

SELECT \* FROM uk\_loans LIMIT 5; Enter a SQL expression to filter results (use Ctrl+Space)

	loan_id	customer_id	loan_type	amount	interest_rate
1	L0001	C00108	Business	198,644.1	
2	L0002	C00366	Personal	22,916.01	
3	L0003	C00469	Mortgage	422,723.74	
4	L0004	C00424	Business	105,952.64	
5	L0005	C00015	Personal	29,407.57	

Value x L0001

Refresh Save Cancel 16 100 Export data 200 5

Screenshot: verify loans.png

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

```
-- Verify data inserted into uk_transactions
```

Results 1 x

SELECT \* FROM uk\_cards LIMIT 5

	card_id	account_id	card_type	card_number	expiry
1	CD0001	A000732	Student	0251668440709744	09/29
2	CD0002	A000449	Business	9536702046593707	05/29
3	CD0003	A000833	Platinum	4872101740325769	03/26
4	CD0004	A000959	Debit	4737968144972989	01/26
5	CD0005	A001411	Credit	0167716503328805	07/25

Refresh Save Cancel Export data 200 5

Screenshot: verify cards.png

-- Verify data inserted into uk\_transactions  
SELECT \* FROM uk\_transactions LIMIT 5;

Results 1 x

SELECT \* FROM uk\_transactio | Enter a SQL expression to filter results (use Ctrl+Space)

	transaction_id	account_id	transaction_type	recipient_accou	Value x
1	T0000001	A000249	Transfer	A000718	T0000001
2	T0000002	A000403	Refund	[NULL]	
3	T0000003	A000435	Payment	[NULL]	
4	T0000004	A001066	Fee	[NULL]	
5	T0000005	A000094	Transfer	A001492	

Refresh Save Cancel Export data 200 5

Screenshot: verify transactions.png

SQL Script Editor

```
-- Verify the data was imported correctly by counting rows in each table
SELECT 'Customers' AS table_name, COUNT(*) AS record_count FROM uk_customers
UNION ALL
SELECT 'Branches' AS table_name, COUNT(*) AS record_count FROM uk_branches
UNION ALL
SELECT 'Employees' AS table_name, COUNT(*) AS record_count FROM uk_employees
UNION ALL
SELECT 'Accounts' AS table_name, COUNT(*) AS record_count FROM uk_accounts
UNION ALL
SELECT 'Loans' AS table_name, COUNT(*) AS record_count FROM uk_loans
UNION ALL
```

Results 1 x

SELECT 'Customers' AS table\_name, COUNT(\*) AS record\_count FROM uk\_customers

Grid	table_name	record_count
2	Branches	20
3	Employees	50
4	Accounts	1,500
5	Loans	150
6	Cards	400
7	Transactions	10,000

Refresh Save Cancel Export data 200 7

7 row(s) fetched - 0.007s, on 2025-03-29 at 08:22:58

Query Manager Database Tasks - General x

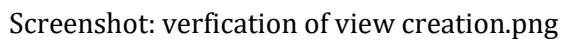
Tasks: type a part of task name here

Name	Last Run (System time)	Last Result	Type
------	------------------------	-------------	------

Task executions: type a part of task name here

System time	Duration
-------------	----------

Screenshot: verification of tables and data through union.png



```
-- Creating view to enforce manager_id constraint
CREATE VIEW vw_branch_manager_check AS
SELECT b.branch_id, b.branch_name, b.manager_id, e.employee_id
FROM uk_branches b
LEFT JOIN uk_employees e ON b.manager_id = e.employee_id
WHERE b.manager_id IS NOT NULL AND e.employee_id IS NULL;

-- To verify constraint:
SELECT * FROM vw_branch_manager_check;
```

Statistics 1 X

SQL Error: Catalog Error: View with name "vw\_branch\_manager\_check" already exists!

```
-- Creating view to enforce manager_id con
CREATE VIEW vw_branch_manager_check AS
SELECT b.branch_id, b.branch_name, b.manag
FROM uk_branches b
LEFT JOIN uk_employees e ON b.manager_id =
WHERE b.manager_id IS NOT NULL AND e.emplo
```

Screenshot: manager view creation.png



-- Query 1: Employees in London Branches (INNER JOIN)  
-- Advanced Technique Explanation:  
-- INNER JOIN precisely matches employees to their respective branches by branch\_id. Cho:  
**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: Customers in Lowest 10% Credit Scores (Window Functions - PERCENT\_RANK & LEAD)  
-- Advanced Technique Explanation:

Results 1 x

SELECT e.first\_name, e.last\_name, e.salary

	first_name	last_name	salary
1	Joe	Bartlett	64,528
2	Stuart	Dickinson	45,563

Value x  
Joe

Screenshot: query 1.png

```

-- 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
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

```

Screenshot: query2.png

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: Account Type Summary (Aggregation & HAVING)  
-- Advanced Technique Explanation:

Results 1 x

WITH RankedCustomers AS (Enter a SQL expression to filter results (use Ctrl+Space))

Grid	customer_id	first_name	last_name	credit_score
1	C00013	Luke	Graham	301
2	C00323	Francis	Singh	304
3	C00209	Nathan	Bradley	306
4	C00312	Marcus	Lewis	306
5	C00498	Abdul	Begum	306
6	C00211	Dorothy	Sullivan	308

Value x  
C00013

Refresh Save Cancel Export data 200 50

Screenshot: query2-1.png

-- 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 those with more than one account.

```

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 3: Account Type Summary (Aggregation & HAVING)

Results 1 x

SELECT account\_type, COUNT(\*) AS total\_accounts, ROUND(AVG(balance), 2) AS avg\_balance

Grid	account_type	total_accounts	avg_balance
1	Business	228	23,259.31
2	Savings	215	25,613.23
3	Premium	226	23,666.26
4	Joint	208	26,809.65
5	Student	207	25,136.4
6	ISA	207	26,316.51
7	Current	209	24,854.21

Refresh | Save | Cancel | Export data | 200 | 7

Screenshot: query3.png

```
-- Query 4: Branches with High-Value Accounts (Correlated Subquery)
-- Advanced Technique Explanation:
-- Employs a correlated subquery, directly linking each branch to the account counts inside the subquery.
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;
```

Results 1 x

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

	branch_name	high_value_accounts
1	London - Royal	64
2	Edinburgh - Central	49
3	Newcastle - Royal	65
4	Bristol - East	63
5	Glasgow - City Centre	67
6	Aberdeen - Market Square	60
7	Liverpool - City Centre	53

Value x  
London - Royal

Refresh Save Cancel Export data 200 20

Screenshot: query4.png

```
-- Query 5: Transactions per Account Type (Window Function)
-- Advanced Technique Explanation:
-- Utilises window function COUNT() OVER PARTITION BY for efficiently summarising transac
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: Branches with High Credit-Score Customers (DISTINCT & JOIN)
-- Advanced Technique Explanation:
```

Results 1 x

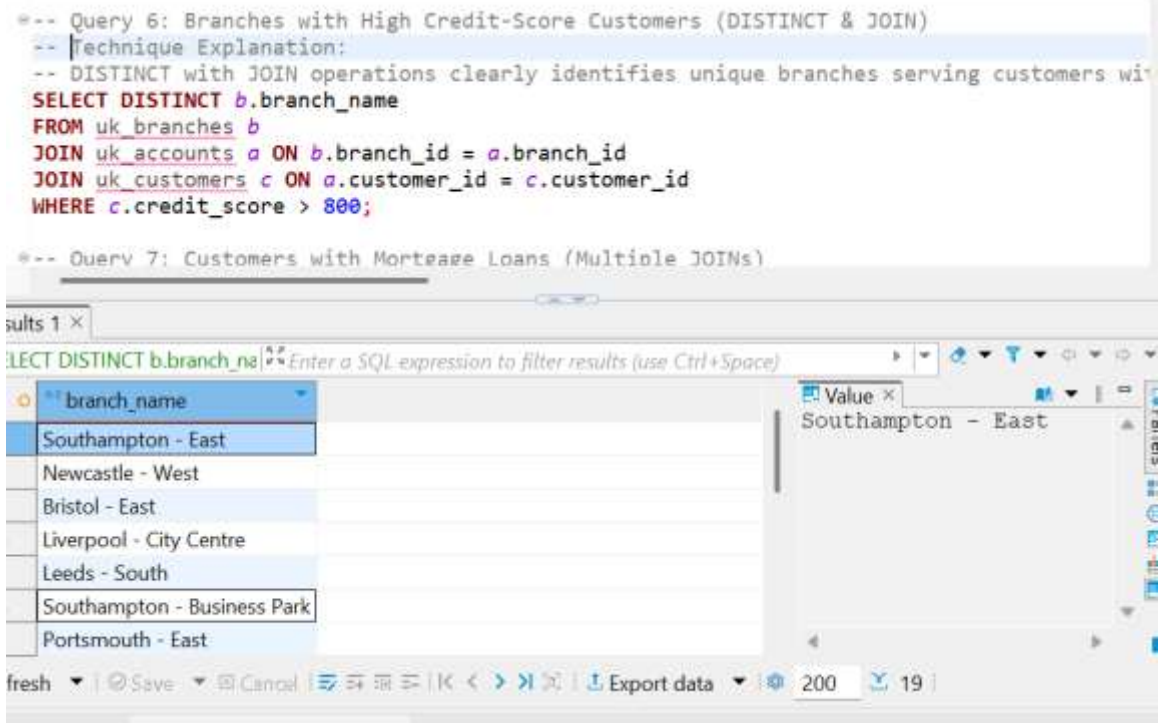
SELECT DISTINCT a.account\_type

	account_type	total_transactions
1	Premium	1,528
2	Business	1,509
3	Current	1,385
4	Savings	1,416
5	ISA	1,374
6	Joint	1,402
7	Student	1,386

Value x  
Premium

Refresh Save Cancel Export data 200 7

Screenshot: query5.png



Screenshot: query6.png

The screenshot shows a SQL IDE with a query editor at the top and a results pane at the bottom. The query editor contains two queries. Query 7 is a SELECT DISTINCT statement with three columns: first\_name, last\_name, and account\_type. It joins uk\_customers (c) with uk\_loans (l) and uk\_accounts (a) on customer\_id, filtering for mortgage loans. Query 8 is a comment about branches without credit card customers. The results pane shows the output of Query 7, displaying 7 rows of customer data. A tooltip for the first row shows the value 'Marion' for the first\_name column.

```
-- Query 7: Customers with Mortgage Loans (Multiple JOINS)
-- Advanced Technique Explanation:
-- Clearly demonstrates sequential JOIN operations for accurate matching between customer
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: Branches without Credit Card Customers (NOT EXISTS)
```

Results 1 x

SELECT DISTINCT c.first\_name

	first_name	last_name	account_type
1	Marion	Holt	Student
2	Jason	Nixon	Premium
3	Marion	Holt	ISA
4	Bruce	Thompson	Premium
5	Sean	Clark	Student
6	Jason	Nixon	Business
7	Judith	Bennett	Student

Value x  
Marion

Screenshot: query7.png



```
-- 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
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

The screenshot shows a SQL query in an IDE. The query is as follows:

```
SELECT  
    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  
);
```

Below the query editor, the 'Results: 1' window is open, displaying a table with the following data:

	branch_name	manager_first_name	manager_last_name
1	Portsmouth - East	David	Gardiner

To the right of the table, a 'Value' column shows the text 'Portsmouth - East'.

Screenshot: query8-1.png

```
-- 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
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'
);
```

Results 1 x

SELECT first\_name, last\_name *Enter a SQL expression to filter results (use Ctrl+Space)*

	first_name	last_name
1	Sheila	Brown
2	Jenna	Smith
3	Natasha	Holden
4	Stanley	Patterson
5	Ann	Hill
6	Sharon	Hamilton
7	Kevin	Stewart

Value x  
Sheila

Refresh Save Cancel Export data 200 67

Screenshot: query9.png



```
-- 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

```

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;

```

Results 1 x

WITH AccountLoans AS ( SELECT *Enter a SQL expression to filter results (use Ctrl+Space)* )

	branch_name	average_loan_amount
1	Manchester - City Centre	67,542.38
2	Portsmouth - East	80,260.74
3	Southampton - Business Park	92,494.33
4	Newcastle - West	35,365.43
5	Southampton - East	71,317.12
6	Portsmouth - Central	49,836.28
7	Sheffield - South	83,667.66

Value x  
Manchester - City Centi

Screenshot: query11-1.png

```

-- Query 12: Customers with All Account Types (HAVING & Subquery)
--Technique Explanation:
-- Uses HAVING clause and subquery to clearly confirm customers possessing every available
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
);

```

results 1 x

SELECT c.customer\_id, c.first\_name, c.last\_name, c.credit\_score

	customer_id	first_name	last_name	credit_score
1	C00076	Deborah	O'Brien	522
2	C00095	Francesca	Gibbs	671
3	C00410	Martin	Wright	394
4	C00094	Rita	Green	535
5	C00120	Conor	Lewis	448
6	C00238	Ruth	Patel	848
7	C00129	Brenda	Taylor	318

Value x: C00076

refresh Save Cancel Export data 200 31

Screenshot: query12 remaining comments.png