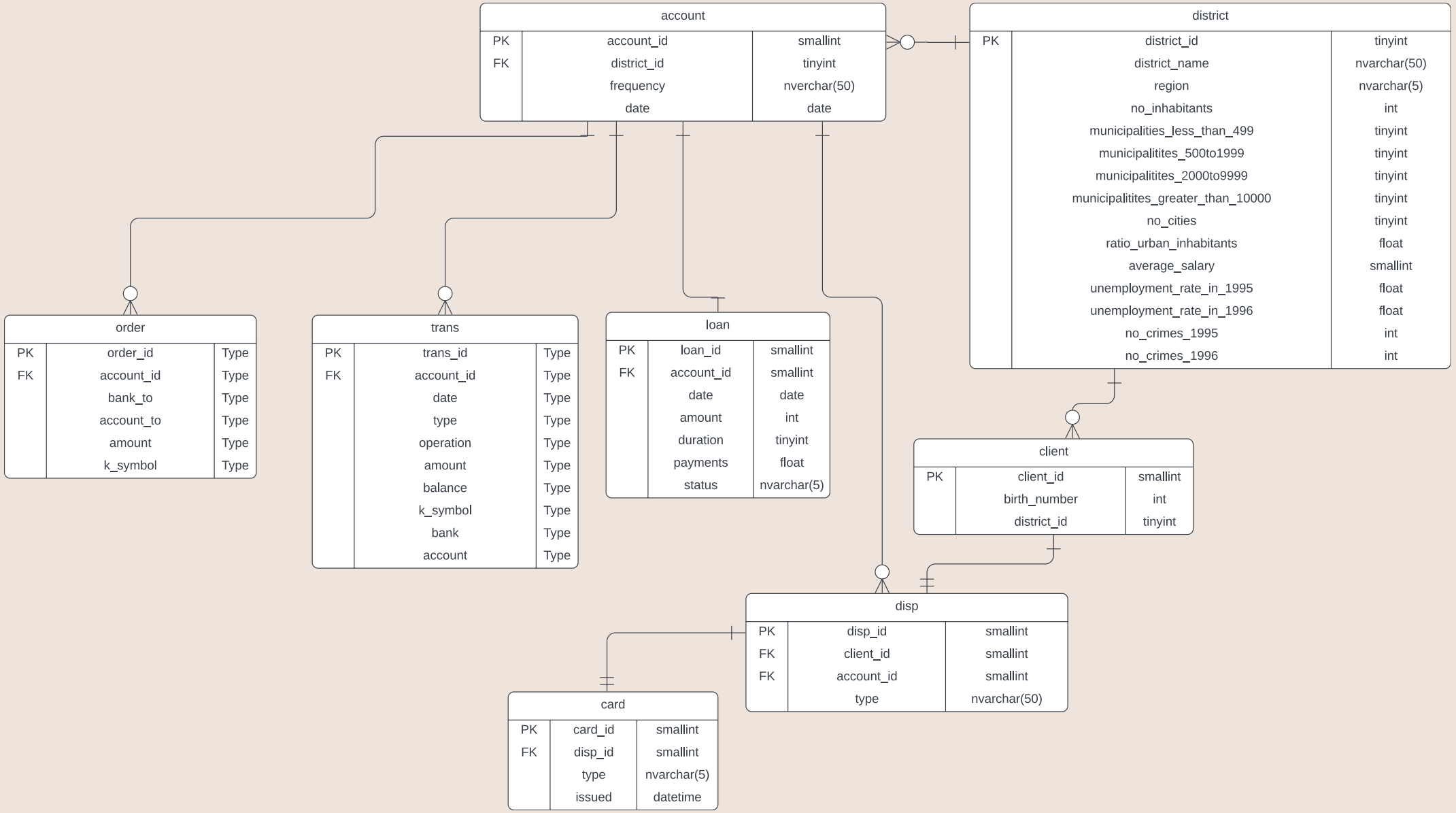
Development of SQL-Based Loan Scoring System to Optimise Loan Approval: A Scenario-Based Project Using Test Data

Alice Mythen

ENTITY RELATIONSHIP DIAGRAM



Finding the client_id associated with each account _id:

- Credit score calculator relies on the client_id as the input.
- Loan table uses account_id not client_id.

```
WITH client_account
AS
(SELECT c.client_id, a.account_id
FROM client AS c
LEFT JOIN disp AS d ON c.client_id = d.client_id
LEFT JOIN account AS a ON d.account_id = a.account_id
WHERE type = 'OWNER'),
```

Simple join of the client table, disposition table and account table in a CTE to be used later to join onto the loan table.

WHERE clause to identify just the OWNERS of the account as only the OWNER can ask for a loan.

Results | Messages

	client_id	account_id
1	1	1
2	2	2
3	4	3
4	6	4
5	7	5
6	8	6
7	9	7
8	10	8
9	12	9
10	13	10

Finding the clients account balance before the loan was granted:

```sql
SELECT l.account_id, ca.client_id, l.status
      ,l.amount AS loan_amount
      ,a.date AS account_start_date
      ,DATEADD(day,-1,l.date) AS day_before_loan  --- day before loan used as looking for the balance before the laon was granted --
      ,t.date AS transaction_date
      ,t.amount AS trans_amount, t.balance AS balance_after_transaction
      ,LAG(t.date) OVER (PARTITION BY l.account_id ORDER BY t.date) AS previous_trans_date
      ,LAG(t.balance) OVER (PARTITION BY l.account_id ORDER BY t.date) AS previous_balance
FROM loan AS l
LEFT JOIN account AS a ON l.account_id = a.account_id
LEFT JOIN trans AS t ON l.account_id = t.account_id
LEFT JOIN client_account as ca ON ca.account_id = l.account_id
```

Joined the transactions table as this gives all the transactions from a clients account with the date of transaction and balance after.

Used the LAG function to find the balance prior to the transaction and the associated date.

Problem – not every client had a transaction on the day before they were granted their loan.

Finding the loan to balance ratio and assigning a score to each one:

```
SELECT @loan_ratio_score =
    CASE WHEN loan_amount/AVG(previous_balance) IS NULL THEN 250
         WHEN loan_amount/AVG(previous_balance) > 200 THEN 250
         WHEN loan_amount/AVG(previous_balance) > 20 THEN CAST(((loan_amount/(AVG(previous_balance)))*(-5/6)+800) AS INT)
         ELSE CAST(((loan_amount/(AVG(previous_balance)))*(-30)+1000) AS INT)
    END
    FROM balance_date AS bd
    WHERE day_before_loan BETWEEN previous_trans_date AND transaction_date
    AND bd.client_id = @p_client_id
    GROUP BY account_id, bd.loan_amount;
```

The BETWEEN function allowed me to find where the date before the loan fell between transaction two dates.

AVG function resolved issues with multiple transactions in one day – GROUP BY account_id required.

$$\text{WHERE } 0 > x > 20$$
$$y = -30x + 1000$$

$$\text{WHERE } 20 > x > 200$$
$$y = (-5/6)x + 800$$

Limitations with linear relationship – Python to be used in the future for exponential relationship.

Assigning a score based on the status of each loan:

CASE WHEN function to assign a score to each account depending on their status.

```
SELECT @status_score =
      CASE WHEN l.status = 'A' THEN 1000
           WHEN l.status = 'B' THEN 750
           WHEN l.status = 'C' THEN 250
           WHEN l.status = 'D' THEN 500
       END
    FROM loan AS l
    LEFT JOIN disp AS d ON d.account_id = l.account_id
    LEFT JOIN client AS c ON c.client_id = d.client_id
    WHERE type = 'OWNER' AND c.client_id = @p_client_id

SET @overall_loan_score = (@status_score + @loan_ratio_score)/2;

    IF NOT EXISTS (SELECT * FROM loan AS l
    LEFT JOIN disp AS d ON d.account_id = l.account_id
    LEFT JOIN client AS c ON c.client_id = d.client_id
    WHERE type = 'OWNER' AND c.client_id = @p_client_id)
    SET @overall_loan_score = 250
```

Overall loan score calculated by dividing the sum of the status score and loan ratio score by 2.

IF NOT EXISTS was used to assign a score of 250 to any client without a loan.

Limitation: generalised status - not considering clients payment behaviour i.e. late payments.

Development for the future – assigning a score based on late

```
WITH payment_history
AS
(
    SELECT l.account_id, l.status, l.date AS loan_start_date,  t.date as transaction_date, l.duration, c.client_id
    ,LEAD(t.date) OVER (PARTITION BY l.account_id ORDER BY t.date) as next_payment
    FROM loan as l
    LEFT JOIN [order] AS o ON l.account_id = o.account_id
    LEFT JOIN trans AS t ON l.account_id = t.account_id
    AND o.account_to = t.account
    AND o.k_symbol = t.k_symbol
    LEFT JOIN disp AS d ON d.account_id = l.account_id
    LEFT JOIN client AS c ON c.client_id = d.client_id
    WHERE o.K_symbol = 'UVER' AND d.type = 'OWNER'), late_payments
AS
(
    SELECT *
    ,DATEDIFF(month, transaction_date, next_payment) AS months_between_payments
    FROM payment_history as ph
    WHERE DATEDIFF(month, transaction_date, next_payment)  > 1
), num_late_payments
AS
(
    SELECT DISTINCT lp.client_id
    ,COUNT(lp.client_id) OVER (PARTITION BY lp.client_id) AS number_of_late_payments
    FROM late_payments AS lp)

SELECT @late_payment_score =
          CASE WHEN number_of_late_payments = 1 THEN 600
               WHEN number_of_late_payments = 2 THEN 500
               WHEN number_of_late_payments = 3 THEN 450
               WHEN number_of_late_payments = 4 THEN 400
               WHEN number_of_late_payments = 5 THEN 350
               WHEN number_of_late_payments = 6 THEN 300
               WHEN number_of_late_payments > 6 THEN 250
END
FROM num_late_payments AS nlp
WHERE nlp.client_id = @p_client_id

END ;
```

We can view client's loan payment history by joining the orders and transactions tables and filtering transaction type
(UVER = loan payment).

DATEDIFF function was used to show the number of months between each payment

COUNT function provided the number of late payments which was then used to assign scores (higher late payments = lower score)

	account_id	transaction_date	next_payment	months_between_payments	num_of_late_payments
86	6118	1995-10-12	1995-12-12	2	2
87	6473	1994-12-12	1995-02-12	2	1
88	6609	1996-06-12	1997-01-12	7	2
89	6609	1997-02-12	1997-07-12	5	2
90	6715	1998-03-12	1998-05-12	2	1