Étude de Cas - SQL Analytics pour Data Analyst

UniBank Haiti - Analyse de Données Client

Contexte

Vous êtes Data Analyst chez UniBank Haiti. On vous demande d'analyser les données clients et transactions pour répondre à plusieurs questions business. Toutes les réponses doivent être fournies en SQL.

Schéma de Base de Données

```sql
CREATE TABLE clients (
    client_id INT PRIMARY KEY,
    nom VARCHAR(100),
    prenom VARCHAR(100),
    date_naissance DATE,
    segment VARCHAR(20),  -- 'Mass', 'Affluent', 'Premium'
    agence_id INT,
    date_inscription DATE,
    statut VARCHAR(20)  -- 'Actif', 'Inactif', 'Fermé'
);

CREATE TABLE comptes (
    compte_id INT PRIMARY KEY,
    client_id INT,
    type_compte VARCHAR(20),  -- 'Courant', 'Epargne', 'Terme'
    solde DECIMAL(15,2),
    date_ouverture DATE,
    statut VARCHAR(20)
);

CREATE TABLE transactions (
    transaction_id INT PRIMARY KEY,
    compte_id INT,
    date_tx TIMESTAMP,
    type_tx VARCHAR(20),  -- 'Depot', 'Retrait', 'Virement', 'Frais'
    montant DECIMAL(15,2),
    canal VARCHAR(20)  -- 'Agence', 'Mobile', 'Web', 'ATM'
);

CREATE TABLE prets (
    pret_id INT PRIMARY KEY,
    client_id INT,
    montant_initial DECIMAL(15,2),
    solde_restant DECIMAL(15,2),
    taux_interet DECIMAL(5,2),
    date_octroi DATE,
    date_echeance DATE,
    jours_retard INT,
```

```
    secteur VARCHAR(50)
);

CREATE TABLE agences (
    agence_id INT PRIMARY KEY,
    nom VARCHAR(100),
    region VARCHAR(50),
    ville VARCHAR(50)
);
```

PARTIE 1: Requêtes de Base

Question 1.1

Listez les 10 clients avec le solde total le plus élevé (tous comptes confondus).

Solution 1.1

```
SELECT
    c.client_id,
    c.nom,
    c.prenom,
    c.segment,
    SUM(co.solde) as solde_total
FROM clients c
JOIN comptes co ON c.client_id = co.client_id
WHERE co.statut = 'Actif'
GROUP BY c.client_id, c.nom, c.prenom, c.segment
ORDER BY solde_total DESC
LIMIT 10;
```

Question 1.2

Calculez le nombre de clients par segment et par région.

Solution 1.2

```
SELECT
    a.region,
    c.segment,
    COUNT(DISTINCT c.client_id) as nb_clients
FROM clients c
JOIN agences a ON c.agence_id = a.agence_id
WHERE c.statut = 'Actif'
GROUP BY a.region, c.segment
ORDER BY a.region, c.segment;
```

Question 1.3

Trouvez les clients qui n'ont effectué aucune transaction depuis plus de 90 jours.

Solution 1.3

```sql
SELECT
    c.client_id,
    c.nom,
    c.prenom,
    c.segment,
    MAX(t.date_tx) as derniere_transaction
FROM clients c
JOIN comptes co ON c.client_id = co.client_id
LEFT JOIN transactions t ON co.compte_id = t.compte_id
WHERE c.statut = 'Actif'
GROUP BY c.client_id, c.nom, c.prenom, c.segment
HAVING MAX(t.date_tx) < CURRENT_DATE - INTERVAL '90 days'
   OR MAX(t.date_tx) IS NULL
ORDER BY derniere_transaction;
```

PARTIE 2: Window Functions

Question 2.1

Pour chaque client, calculez son solde total et son rang au sein de son segment.

Solution 2.1

```sql
WITH client_soldes AS (
    SELECT
        c.client_id,
        c.nom,
        c.segment,
        SUM(co.solde) as solde_total
    FROM clients c
    JOIN comptes co ON c.client_id = co.client_id
    WHERE c.statut = 'Actif' AND co.statut = 'Actif'
    GROUP BY c.client_id, c.nom, c.segment
)
SELECT
    client_id,
    nom,
    segment,
    solde_total,
    RANK() OVER (PARTITION BY segment ORDER BY solde_total DESC) as rang_segment,
    ROUND(solde_total * 100.0 / SUM(solde_total) OVER (PARTITION BY segment), 2) as pct_segment
FROM client_soldes
ORDER BY segment, rang_segment;
```

Question 2.2

Calculez le montant cumulé des transactions par jour pour le mois de janvier 2024.

Solution 2.2

```sql
SELECT
    DATE(date_tx) as jour,
    SUM(CASE WHEN type_tx = 'Depot' THEN montant ELSE -montant END) as flux_net,
    SUM(SUM(CASE WHEN type_tx = 'Depot' THEN montant ELSE -montant END))
        OVER (ORDER BY DATE(date_tx)) as cumul_flux
FROM transactions
WHERE date_tx >= '2024-01-01' AND date_tx < '2024-02-01'
GROUP BY DATE(date_tx)
ORDER BY jour;
```

Question 2.3

Pour chaque transaction, affichez le montant et la différence avec la transaction précédente du même client.

Solution 2.3

```sql
SELECT
    t.transaction_id,
    co.client_id,
    t.date_tx,
    t.montant,
    LAG(t.montant) OVER (
        PARTITION BY co.client_id
        ORDER BY t.date_tx
    ) as montant_precedent,
    t.montant - COALESCE(LAG(t.montant) OVER (
        PARTITION BY co.client_id
        ORDER BY t.date_tx
    ), 0) as difference
FROM transactions t
JOIN comptes co ON t.compte_id = co.compte_id
ORDER BY co.client_id, t.date_tx;
```

Question 2.4

Identifiez les 3 meilleurs agents de crédit par agence (basé sur le volume décaissé).

Solution 2.4

```sql
WITH agent_stats AS (
    SELECT
        a.agence_id,
        a.nom as agence,
```

```
        p.agent_id,  -- supposons cette colonne existe
        SUM(p.montant_initial) as total_decaisse,
        COUNT(*) as nb_prets,
        ROW_NUMBER() OVER (
            PARTITION BY a.agence_id
            ORDER BY SUM(p.montant_initial) DESC
        ) as rang
    FROM agences a
    JOIN clients c ON a.agence_id = c.agence_id
    JOIN prets p ON c.client_id = p.client_id
    WHERE p.date_octroi >= CURRENT_DATE - INTERVAL '1 year'
    GROUP BY a.agence_id, a.nom, p.agent_id
)
SELECT *
FROM agent_stats
WHERE rang <= 3
ORDER BY agence_id, rang;
```

PARTIE 3: CTEs et Analyses Complexes

Question 3.1

Analysez le comportement RFM des clients (Recency, Frequency, Monetary).

Solution 3.1

```
WITH rfm_raw AS (
    SELECT
        c.client_id,
        c.nom,
        c.segment,
        -- Recency: jours depuis dernière transaction
        CURRENT_DATE - MAX(DATE(t.date_tx)) as recency,
        -- Frequency: nombre de transactions
        COUNT(t.transaction_id) as frequency,
        -- Monetary: montant total
        SUM(t.montant) as monetary
    FROM clients c
    JOIN comptes co ON c.client_id = co.client_id
    LEFT JOIN transactions t ON co.compte_id = t.compte_id
        AND t.date_tx >= CURRENT_DATE - INTERVAL '1 year'
    WHERE c.statut = 'Actif'
    GROUP BY c.client_id, c.nom, c.segment
),
rfm_scored AS (
    SELECT
        *,
        -- R Score (5 = plus récent)
        NTILE(5) OVER (ORDER BY recency DESC) as r_score,
        -- F Score (5 = plus fréquent)
        NTILE(5) OVER (ORDER BY frequency) as f_score,
        -- M Score (5 = plus dépensé)
```

```
        NTILE(5) OVER (ORDER BY monetary) as m_score
    FROM rfm_raw
)
SELECT
    client_id,
    nom,
    segment,
    recency,
    frequency,
    monetary,
    r_score,
    f_score,
    m_score,
    r_score || f_score || m_score as rfm_segment,
    CASE
        WHEN r_score >= 4 AND f_score >= 4 AND m_score >= 4 THEN 'Champions'
        WHEN r_score >= 4 AND f_score <= 2 THEN 'Nouveaux Clients'
        WHEN r_score <= 2 AND f_score >= 4 THEN 'À Risque'
        WHEN r_score <= 2 AND f_score <= 2 THEN 'Perdus'
        ELSE 'Moyens'
    END as segment_rfm
FROM rfm_scored
ORDER BY monetary DESC;
```

Question 3.2

Calculez le NPL ratio par secteur et identifiez les secteurs problématiques.

Solution 3.2

```
WITH sector_stats AS (
    SELECT
        secteur,
        COUNT(*) as nb_prets,
        SUM(solde_restant) as total_encours,
        SUM(CASE WHEN jours_retard > 90 THEN solde_restant ELSE 0 END) as npl_encours,
        AVG(taux_interet) as taux_moyen
    FROM prets
    WHERE solde_restant > 0
    GROUP BY secteur
)
SELECT
    secteur,
    nb_prets,
    total_encours,
    npl_encours,
    ROUND(npl_encours * 100.0 / NULLIF(total_encours, 0), 2) as npl_ratio,
    taux_moyen,
    CASE
        WHEN npl_encours * 100.0 / NULLIF(total_encours, 0) > 7 THEN 'CRITIQUE'
        WHEN npl_encours * 100.0 / NULLIF(total_encours, 0) > 5 THEN 'ALERTE'
        ELSE 'OK'
```

```
    END as statut_risque
FROM sector_stats
ORDER BY npl_ratio DESC;
```

Question 3.3

Créez une analyse de cohorte mensuelle montrant la rétention client.

Solution 3.3

```
WITH client_cohorts AS (
    -- Cohorte = mois d'inscription
    SELECT
        client_id,
        DATE_TRUNC('month', date_inscription) as cohort_month
    FROM clients
    WHERE date_inscription >= '2023-01-01'
),
client_activity AS (
    -- Mois d'activité
    SELECT DISTINCT
        co.client_id,
        DATE_TRUNC('month', t.date_tx) as activity_month
    FROM comptes co
    JOIN transactions t ON co.compte_id = t.compte_id
    WHERE t.date_tx >= '2023-01-01'
),
cohort_data AS (
    SELECT
        cc.cohort_month,
        ca.activity_month,
        DATE_PART('month', AGE(ca.activity_month, cc.cohort_month)) as months_since_join,
        COUNT(DISTINCT cc.client_id) as active_clients
    FROM client_cohorts cc
    LEFT JOIN client_activity ca ON cc.client_id = ca.client_id
        AND ca.activity_month >= cc.cohort_month
    GROUP BY cc.cohort_month, ca.activity_month
)
SELECT
    cohort_month,
    months_since_join,
    active_clients,
    ROUND(active_clients * 100.0 /
        FIRST_VALUE(active_clients) OVER (PARTITION BY cohort_month ORDER BY months_since_join), 2
    ) as retention_rate
FROM cohort_data
WHERE months_since_join IS NOT NULL
ORDER BY cohort_month, months_since_join;
```

PARTIE 4: Requêtes d'Optimisation

Question 4.1

Expliquez comment vous optimiseriez cette requête lente:

```sql
SELECT *
FROM transactions
WHERE YEAR(date_tx) = 2024
AND client_id IN (SELECT client_id FROM clients WHERE segment = 'Premium');
```

Solution 4.1

Problèmes identifiés: 1. `SELECT *` - récupère toutes les colonnes 2. `YEAR(date_tx)` - empêche l'utilisation d'index 3. Sous-requête avec `IN` - peut être inefficace

Version optimisée:

```sql
-- Créer un index si non existant
CREATE INDEX idx_transactions_date ON transactions(date_tx);
CREATE INDEX idx_clients_segment ON clients(segment);

-- Requête optimisée
SELECT
    t.transaction_id,
    t.compte_id,
    t.date_tx,
    t.type_tx,
    t.montant
FROM transactions t
JOIN comptes co ON t.compte_id = co.compte_id
JOIN clients c ON co.client_id = c.client_id
WHERE t.date_tx >= '2024-01-01'
  AND t.date_tx < '2025-01-01'
  AND c.segment = 'Premium';
```

Améliorations: 1. Sélection de colonnes spécifiques 2. Utilisation d'un range pour la date (permet l'index) 3. JOIN au lieu de sous-requête IN 4. Index appropriés suggérés

---

Question 4.2

Proposez des index pour optimiser les requêtes fréquentes sur la table transactions.

Solution 4.2

```sql
-- Index pour les filtres par date (très courant)
CREATE INDEX idx_tx_date ON transactions(date_tx);

-- Index composite pour les analyses par compte et date
CREATE INDEX idx_tx_compte_date ON transactions(compte_id, date_tx);

-- Index pour les filtres par type de transaction
CREATE INDEX idx_tx_type ON transactions(type_tx);

-- Index pour les analyses par canal
```

```sql
CREATE INDEX idx_tx_canal ON transactions(canal);

-- Index partiel pour les grosses transactions (analyse fraude)
CREATE INDEX idx_tx_gros_montant ON transactions(montant)
WHERE montant > 50000;
```

PARTIE 5: Cas Pratiques

Question 5.1

Le département marketing veut identifier les clients susceptibles de churner. Créez une requête qui identifie les clients à risque.

Solution 5.1

```sql
WITH client_metrics AS (
    SELECT
        c.client_id,
        c.nom,
        c.segment,
        c.date_inscription,
        -- Dernière activité
        MAX(t.date_tx) as derniere_tx,
        CURRENT_DATE - MAX(DATE(t.date_tx)) as jours_inactif,
        -- Tendance d'activité
        COUNT(CASE WHEN t.date_tx >= CURRENT_DATE - INTERVAL '30 days' THEN 1 END) as tx_30j,
        COUNT(CASE WHEN t.date_tx >= CURRENT_DATE - INTERVAL '90 days'
                   AND t.date_tx < CURRENT_DATE - INTERVAL '30 days' THEN 1 END) as tx_30_90j,
        -- Solde
        SUM(co.solde) as solde_total,
        -- Nombre de produits
        COUNT(DISTINCT co.compte_id) as nb_comptes
    FROM clients c
    JOIN comptes co ON c.client_id = co.client_id
    LEFT JOIN transactions t ON co.compte_id = t.compte_id
    WHERE c.statut = 'Actif'
    GROUP BY c.client_id, c.nom, c.segment, c.date_inscription
)
SELECT
    client_id,
    nom,
    segment,
    jours_inactif,
    tx_30j,
    tx_30_90j,
    solde_total,
    nb_comptes,
    -- Score de risque de churn
    CASE
        WHEN jours_inactif > 90 THEN 'TRÈS ÉLEVÉ'
        WHEN jours_inactif > 60 OR (tx_30j = 0 AND tx_30_90j > 5) THEN 'ÉLEVÉ'
        WHEN jours_inactif > 30 OR tx_30j < tx_30_90j / 2 THEN 'MODÉRÉ'
```

```
        ELSE 'FAIBLE'
    END as risque_churn
FROM client_metrics
WHERE jours_inactif > 30 OR tx_30j < tx_30_90j / 2
ORDER BY
    CASE
        WHEN jours_inactif > 90 THEN 1
        WHEN jours_inactif > 60 THEN 2
        WHEN jours_inactif > 30 THEN 3
        ELSE 4
    END,
    solde_total DESC;
```

Question 5.2

Créez un rapport mensuel de performance des agences.

Solution 5.2

```
WITH monthly_stats AS (
    SELECT
        a.agence_id,
        a.nom as agence,
        a.region,
        DATE_TRUNC('month', t.date_tx) as mois,
        -- Volume transactions
        COUNT(t.transaction_id) as nb_transactions,
        SUM(t.montant) as volume_transactions,
        -- Dépôts nets
        SUM(CASE WHEN t.type_tx = 'Depot' THEN t.montant ELSE 0 END) -
        SUM(CASE WHEN t.type_tx = 'Retrait' THEN t.montant ELSE 0 END) as flux_net,
        -- Clients actifs
        COUNT(DISTINCT co.client_id) as clients_actifs
    FROM agences a
    JOIN clients c ON a.agence_id = c.agence_id
    JOIN comptes co ON c.client_id = co.client_id
    JOIN transactions t ON co.compte_id = t.compte_id
    WHERE t.date_tx >= DATE_TRUNC('month', CURRENT_DATE) - INTERVAL '12 months'
    GROUP BY a.agence_id, a.nom, a.region, DATE_TRUNC('month', t.date_tx)
)
SELECT
    mois,
    agence,
    region,
    nb_transactions,
    volume_transactions,
    flux_net,
    clients_actifs,
    -- Comparaison au mois précédent
    LAG(nb_transactions) OVER (PARTITION BY agence_id ORDER BY mois) as tx_mois_prec,
    ROUND((nb_transactions - LAG(nb_transactions) OVER (PARTITION BY agence_id ORDER BY mois)) * 100.0 /
        NULLIF(LAG(nb_transactions) OVER (PARTITION BY agence_id ORDER BY mois), 0), 2) as var_tx_pct,
```

```
    -- Rang par région
    RANK() OVER (PARTITION BY region, mois ORDER BY volume_transactions DESC) as rang_regional
FROM monthly_stats
ORDER BY mois DESC, region, rang_regional;
```

Critères d'Évaluation

Critère	Points
Exactitude des requêtes	30%
Utilisation appropriée des fonctionnalités SQL	25%
Optimisation et performance	20%
Clarté et lisibilité du code	15%
Interprétation business	10%

Conseils

1. **Testez vos requêtes** avec des cas limites
2. **Commentez** les parties complexes
3. **Utilisez des CTEs** pour améliorer la lisibilité
4. **Pensez aux index** pour les requêtes fréquentes
5. **Vérifiez les NULL** dans vos calculs