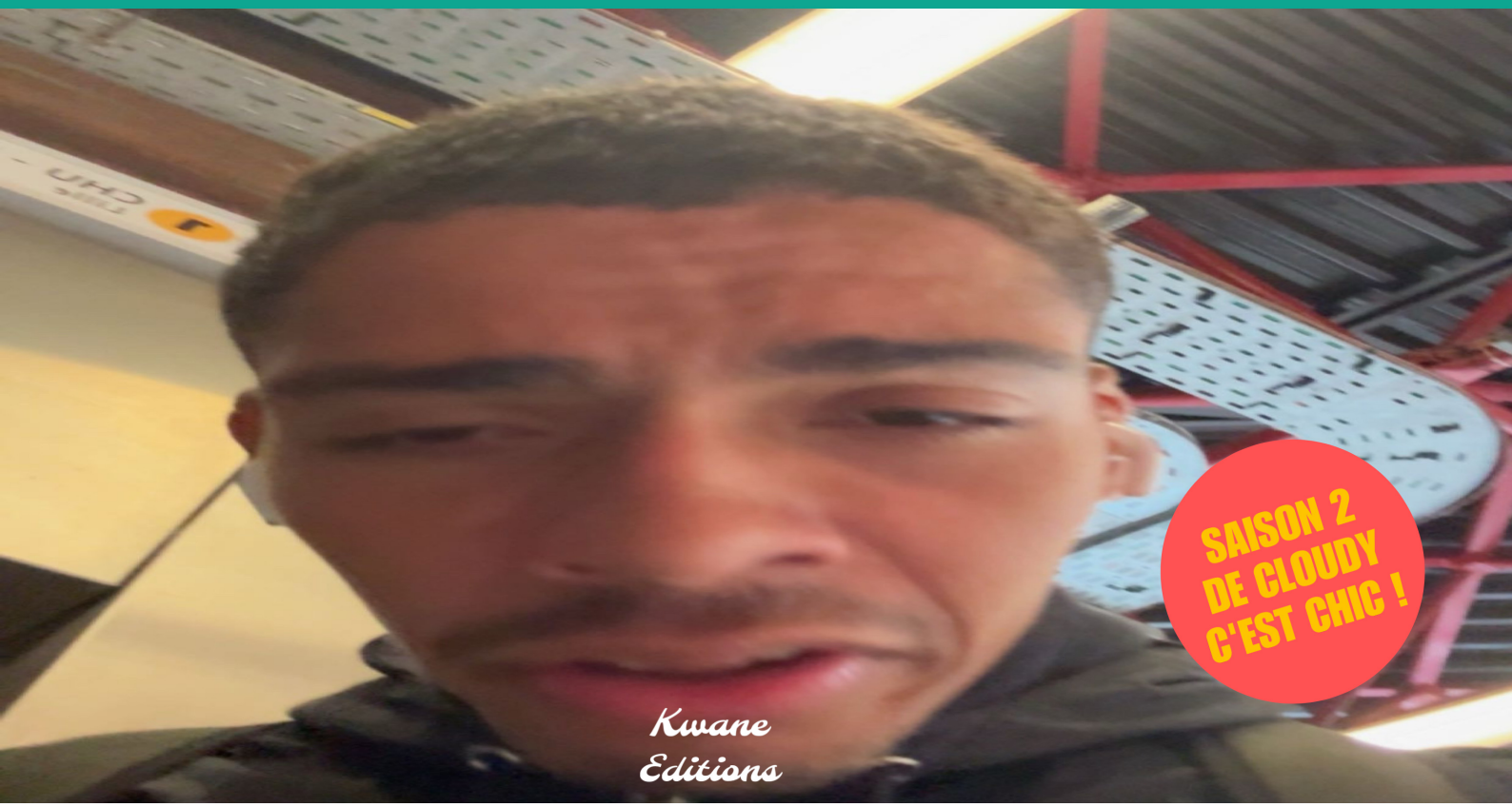
PATRICK CLOUDY

BONJOUR C'EST ENCORE MOI, PHILIPPE MATHIEU



Kwane
Editions

Allons-y en douceur…

1. Création de tables (DDL - Data Definition Language)

Concept

- La création de tables en SQL se fait avec la commande `CREATE TABLE`.
- Chaque table doit avoir une **clé primaire** (`PRIMARY KEY`), et les types de données doivent être spécifiés pour chaque colonne.

Exemple

```
CREATE TABLE abonne (
  ano SERIAL PRIMARY KEY,        -- Identifiant unique
  nom VARCHAR(50),               -- Nom de l'abonné
  prenom VARCHAR(50),            -- Prénom de l'abonné
  date_naissance DATE,           -- Date de naissance au
format AAAA-MM-JJ
  adresse VARCHAR(100)           -- Adresse
);
```

Clés importantes :

- **SERIAL** : génère un identifiant unique automatiquement.
- **VARCHAR(n)** : chaîne de caractères de longueur maximale `n`.
- **DATE** : format de date (AAAA-MM-JJ).
- **PRIMARY KEY** : identifie chaque enregistrement de façon unique.

2. Insertion de données (DML - Data Manipulation Language)

Concept

- La commande `INSERT INTO` permet d'ajouter des lignes dans une table.
- Il est possible d'ajouter des données soit pour toutes les colonnes, soit pour une sélection de colonnes.

Exemple

```
INSERT INTO abonne (nom, prenom, date_naissance, adresse)
VALUES ('Dupont', 'Jean', '1990-05-14', '123 Rue de Lille');
```

Points clés :

- **INSERT INTO table (colonnes)** : spécifie les colonnes dans lesquelles insérer des données.
- **VALUES (valeurs)** : les valeurs à insérer dans chaque colonne, dans le même ordre.

3. Requêtes de sélection (DQL - Data Query Language)

Concept

- La commande `SELECT` permet de récupérer des données d'une table.
- `WHERE` filtre les résultats en fonction de certaines conditions.

Exemple

```
SELECT nom, prenom
FROM abonne
WHERE date_naissance < '2000-01-01';
```

Points clés :

- **SELECT** : indique les colonnes à afficher.
- **FROM** : spécifie la table d'où proviennent les données.
- **WHERE** : filtre les résultats selon des conditions.

4. Mise à jour de données

Concept

- La commande `UPDATE` modifie les valeurs existantes dans une table.
- On utilise `SET` pour indiquer les nouvelles valeurs et `WHERE` pour spécifier quelles lignes sont concernées.

Exemple

```
UPDATE abonne
SET adresse = '456 Avenue des Sports'
WHERE nom = 'Dupont' AND prenom = 'Jean';
```

Points clés :

- **UPDATE table** : indique la table à modifier.
- **SET colonne = nouvelle_valeur** : définit la nouvelle valeur.
- **WHERE** : filtre les lignes à modifier.

5. Suppression de données

Concept

- La commande `DELETE` permet de supprimer des lignes d'une table.
- `WHERE` est crucial pour éviter de supprimer toutes les lignes !

Exemple

```
DELETE FROM abonne
WHERE nom = 'Dupont' AND prenom = 'Jean';
```

Points clés :

- **DELETE FROM table** : indique la table dans laquelle supprimer des lignes.
- **WHERE** : filtre les lignes à supprimer.

6. Jointures entre tables

Concept

- Les jointures permettent de combiner des données provenant de plusieurs tables en une seule requête.
- La jointure la plus courante est la **jointure interne** (`INNER JOIN`), qui récupère les lignes ayant des correspondances dans les deux tables.

Exemple

```
SELECT abonne.nom, creneau.jour, terrain.tnom
FROM abonne
INNER JOIN reservation ON abonne.ano = reservation.ano
INNER JOIN creneau ON reservation.cno = creneau.cno
INNER JOIN terrain ON creneau.tno = terrain.tno;
```

Points clés :

- **INNER JOIN** : combine les lignes avec correspondance dans les deux tables.
- **ON** : spécifie les colonnes à utiliser pour faire la correspondance.

7. Requêtes complexes (Aggrégats, GROUP BY, HAVING)

Concept

- Les fonctions d'agrégats (`COUNT`, `SUM`, `AVG`, etc.) permettent de réaliser des calculs sur des ensembles de données.
- `GROUP BY` permet de regrouper les lignes ayant les mêmes valeurs dans certaines colonnes.
- `HAVING` est utilisé pour filtrer après un `GROUP BY`.

Exemple

```
SELECT terrain.tnom, COUNT(*) AS nb_reservations
FROM reservation
INNER JOIN terrain ON reservation.tno = terrain.tno
GROUP BY terrain.tnom
```

```
HAVING COUNT(*) > 5;
```

Points clés :

- **COUNT()** : compte le nombre de lignes.
- **GROUP BY** : regroupe les résultats par une ou plusieurs colonnes.
- **HAVING** : filtre après un `GROUP BY`.

8. Transactions

Concept

- Les transactions permettent de regrouper plusieurs opérations en une seule unité logique.
- On utilise `BEGIN`, `COMMIT` pour valider les changements, et `ROLLBACK` pour annuler une transaction en cas d'erreur.

Exemple

```
BEGIN;
UPDATE abonne SET adresse = '789 Rue du Sport' WHERE nom =
'Durand';
DELETE FROM reservation WHERE ano = 5;
COMMIT;
```

Points clés :

- **BEGIN** : démarre une transaction.
- **COMMIT** : valide les changements.
- **ROLLBACK** : annule les changements depuis `BEGIN`.

9. JDBC et interaction avec Postgres/H2

Concept

- JDBC est une API Java permettant de se connecter à des bases de données et d'exécuter des requêtes SQL.
- Postgres et H2 sont des bases de données couramment utilisées en développement.
- Exemple de connexion JDBC :

```
Connection conn =
DriverManager.getConnection("jdbc:postgresql://
localhost:5432/ma_base", "utilisateur", "motdepasse");
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery("SELECT * FROM abonne");
```

Points clés :

- **Connection** : se connecte à la base de données.
- **Statement** : prépare et exécute des requêtes SQL.
- **ResultSet** : récupère les résultats des requêtes `SELECT`.

Pratiquons… (D'après le DS de 2022)

1. **Exercice 1 : Compter les personnes entre 20 et 40 ans à Lille ou Lens**
   Programme en Java :

```
public class Exo1 {
    public static void main(String[] args) {
        try {
            Connection conn = DriverManager.getConnection(url, user, password);
            PreparedStatement stmt = conn.prepareStatement(
                "SELECT COUNT(*) FROM personne WHERE age BETWEEN 20 AND 40 AND (ville = 'Lille' OR ville = 'Lens')"
            );
            ResultSet rs = stmt.executeQuery();
            if (rs.next()) {
                System.out.println("Il y a " + rs.getInt(1) + " personnes dans cette situation.");
            }
            rs.close();
            stmt.close();
            conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

   - **Explication** : On utilise une requête préparée pour sélectionner les personnes dont l'âge est compris entre 20 et 40 ans et vivant à Lille ou Lens.

2. **Exercice 2 : Affichage en HTML des personnes triées par ville et par âge décroissant**
   Programme en Java :

```
public class Exo2 {
    public static void main(String[] args) {
        try {
            Connection conn = DriverManager.getConnection(url, user, password);
            PreparedStatement stmt = conn.prepareStatement(
```

```
                "SELECT * FROM personne ORDER BY ville ASC, age DESC"
            );
            ResultSet rs = stmt.executeQuery();
            String lastCity = "";
            while (rs.next()) {
                String city = rs.getString("ville");
                if (!city.equals(lastCity)) {
                    System.out.println("<h2>" + city + "</h2>");
                    lastCity = city;
                }
                System.out.println(
                    rs.getInt("pno") + " <td>" +
                    rs.getString("nom") + " <td>" +
                    rs.getString("prenom") + " <td>" +
                    rs.getInt("age")
                );
            }
            rs.close();
            stmt.close();
            conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

- **Explication** : Les résultats sont triés par ville (ordre alphabétique) et âge (ordre décroissant), et affichés sous une forme simple simulant du HTML.

Exercice 3 : Transaction pour dispatcher les lignes de la table dans des tables par ville
Programme en Java :

```
public class Exo3 {

    public static void main(String[] args) {
        try {
            Connection conn = DriverManager.getConnection(url, user, password);
            conn.setAutoCommit(false); // Démarrer la transaction

            PreparedStatement stmtSelect = conn.prepareStatement("SELECT * FROM personne");
```

```
59.             ResultSet rs = stmtSelect.executeQuery();
60.
61.             while (rs.next()) {
62.                 String city = rs.getString("ville");
63.                 PreparedStatement stmtInsert =
    conn.prepareStatement(
64.                         "INSERT INTO " + city + " (pno, nom,
    prenom, age) VALUES (?, ?, ?, ?)"
65.                 );
66.                 stmtInsert.setInt(1, rs.getInt("pno"));
67.                 stmtInsert.setString(2,
    rs.getString("nom"));
68.                 stmtInsert.setString(3,
    rs.getString("prenom"));
69.                 stmtInsert.setInt(4, rs.getInt("age"));
70.                 stmtInsert.executeUpdate();
71.                 stmtInsert.close();
72.             }
73.
74.             conn.commit(); // Valider la transaction
75.             rs.close();
76.             stmtSelect.close();
77.             conn.close();
78.         } catch (SQLException e) {
79.             e.printStackTrace();
80.             try {
81.                 conn.rollback(); // Annuler la
    transaction en cas d'erreur
82.             } catch (SQLException ex) {
83.                 ex.printStackTrace();
84.             }
85.         }
86.     }
87. }
88.
```

- **Explication** : On insère les données dans les tables spécifiques à chaque ville. La transaction est "Tout ou Rien" grâce à l'utilisation de `commit()` et `rollback()`.

Bonnes Pratiques en SQL et JDBC

- **Optimiser les requêtes** pour minimiser les lectures et transferts réseau.
- **Utiliser les transactions** pour assurer l'intégrité des données en cas d'erreur.
- **Fermeture des ressources** (connexion, statement, resultset) pour éviter les fuites de mémoire.

Allez, bonne chance pour ce ds, dure dure semaine, révisez bien vos 284 points de cours pour la crypto !!