Licence 2 Informatique-Electronique

Année universitaire 2025-2026

Présenté par : **DIALLO Aissatou Bobo**

Chargé du cours : **Mme GILLET Annabelle**

Rapport de projet

Salle de sport

Table des matières

I. Introduction

Le projet présenté dans ce rapport s'inscrit dans le cadre d'un enseignement consacré à l'introduction aux bases de données. Il consiste à concevoir, modéliser et interroger une base de données relationnelle destinée à gérer l'activité d'une salle de sport. Celle-ci doit pouvoir suivre ses clients, leurs abonnements, les prestations proposées, les séances spécialisées, les séances de coaching, les paiements associés, ainsi que la maintenance du matériel.

L'objectif est d'élaborer un modèle cohérent, de le traduire en schéma relationnel sous PostgreSQL, puis de l'illustrer à travers un ensemble de requêtes SQL et une courte interaction applicative en Java.

L'objectif de ce rapport est de présenter de manière synthétique la modélisation retenue, son implémentation sous PostgreSQL, ainsi que plusieurs requêtes illustrant les principales fonctionnalités et interrogations possibles sur la base de données.

II. Analyse fonctionnelle

Le projet vise à concevoir une base de données relationnelle permettant de gérer efficacement l'activité d'une salle de sport. Le système doit centraliser les informations sur les clients, leurs abonnements, les prestations, les séances, les paiements et la maintenance du matériel afin de pouvoir les représenter et les traiter correctement.

Gestion des clients et des abonnements

La base de données doit permettre d'enregistrer les **informations personnelles des clients** ainsi que les **abonnements** auxquels ils souscrivent.
Chaque abonnement possède un niveau, des dates de début et de fin, et un prix. Il doit être associé à un client précis.
Le système doit également gérer les **prestations incluses** dans chaque abonnement, car certains niveaux donnent accès à plus de services que d'autres.

Fonctionnalités principales :

- Enregistrer les informations des clients.
- Associer un client à un ou plusieurs abonnements.
- Définir les caractéristiques d'un abonnement (niveau, prix).
- Lister les prestations incluses dans chaque niveau d'abonnement.

Gestion des séances

La salle propose deux types d'activités : des **séances spécialisées** (cours collectifs) et des **séances de coaching individuel**.
Pour chacune, il est nécessaire de stocker la date, la durée et le coach responsable.

Séances spécialisées

Les séances spécialisées sont des cours collectifs auxquels les clients peuvent s'inscrire.
Le système doit enregistrer :

- le thème du cours (Yoga, Boxe, Pilates…) ;
- la date et la durée de la séance ;
- le coach qui l'anime ;
- les inscriptions des clients.

Chaque inscription doit être unique : un client ne peut pas s'inscrire deux fois à la même séance.

Séances de coaching

Les séances de coaching sont individuelles et payantes si ne sont pas incluses dans l'abonnement d'un client.
Elles mettent directement en relation :

- un coach,
- un client,
- une date de séance,

- une durée,
- une date d'inscription,
- Un montant pour le paiement de la séance,
- Une date de paiement.

Il n'y a pas de mécanisme d'inscription séparé : la séance elle-même représente l'engagement entre le coach et le client.

Gestion des inscriptions des clients aux séances spécialisées

Etant donné que cette séance n'est pas individuelle, plusieurs clients peuvent participer à la même séance à la fois. Mais pour y participer il leur faut s'inscrire à cette séance et payer des frais supplémentaires non incluses dans leur abonnement.

Donc on a :

- la date d'inscription
- la date du paiement,
- le montant.

Gestion du matériel et de la maintenance

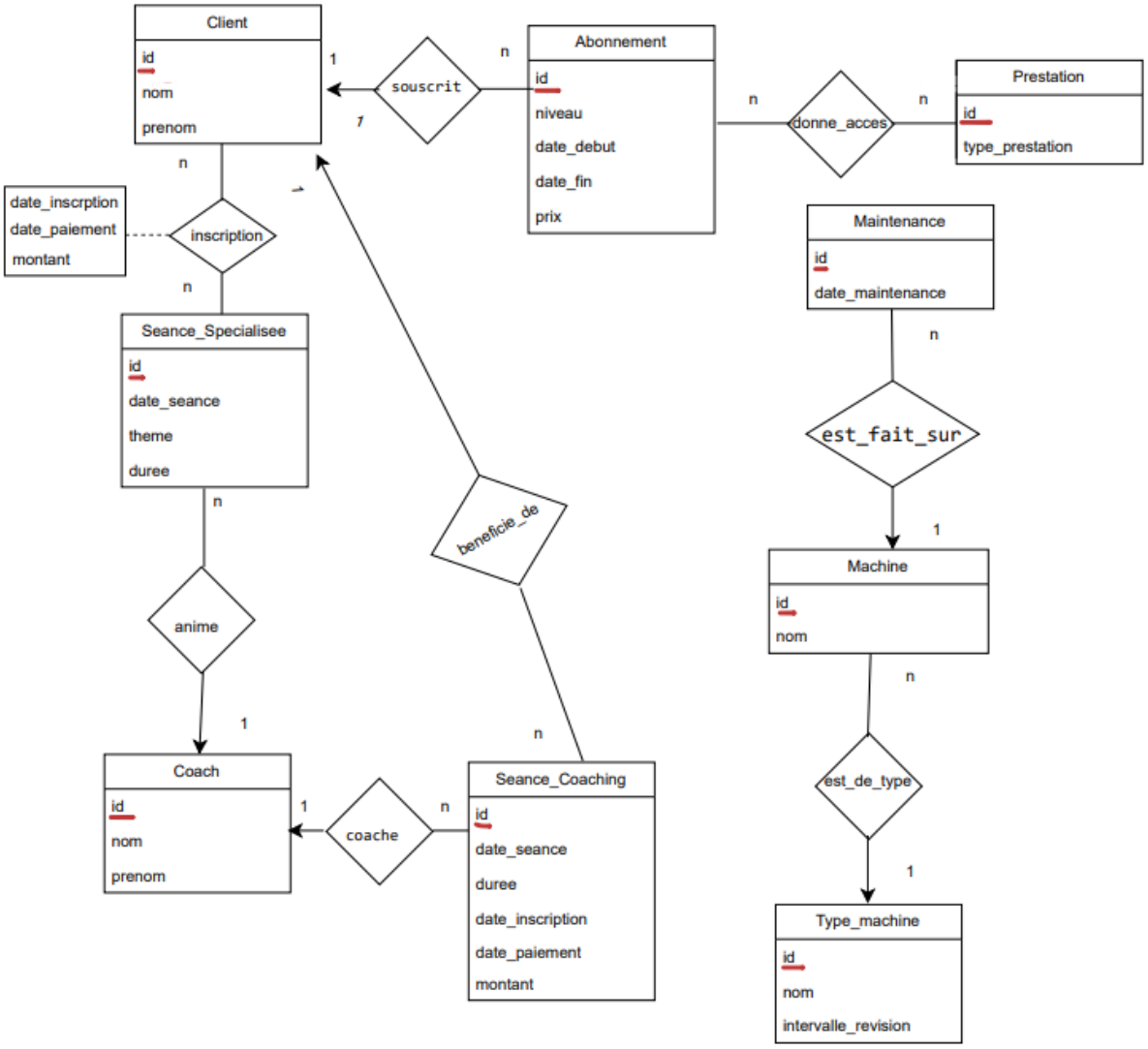
La salle de sport dispose de plusieurs machines réparties en types (tapis de course, vélo elliptique, etc.), chaque type possédant un **intervalle de révision** exprimé en jours.
Le système doit suivre l'historique des maintenances afin de déterminer si une machine doit prochainement être révisée.

Fonctionnalités principales :

- Enregistrer les types de machines avec leur intervalle de révision.
- Associer chaque machine à un type.
- Enregistrer les opérations de maintenance (date + machine concernée).
- Permettre d'identifier les machines qui approchent leur prochaine révision.

III. Modélisation

La modélisation réalisée à l'aide du modèle Entité–Association ci-dessous permet de représenter la structure de notre base de données. Elle met en évidence les différentes entités ainsi que les relations qui les lient.



IV. Justification de la modélisation

Relation Client – Abonnement (1–n)

• Un client peut souscrire plusieurs abonnements, tandis qu'un abonnement n'est associé qu'à un seul client. Cette organisation correspond à une cardinalité de type 1–n. On aurait pu mettre une relation 1-1 pour avoir « un client souscrit à un abonnement » ça n'aurait pas posé de problème mais je suis partie dans l'idée que dans une année par exemple un client peut souscrire à plusieurs abonnement mais un à la fois.
• Une alternative aurait été de modéliser cette relation en n–n, ce qui impliquerait que plusieurs clients puissent partager un même abonnement. Cette option ne correspond pas à la réalité, car un abonnement est personnel.
• La relation 1–n garantit une structure simple, logique et conforme au fonctionnement d'une salle de sport.

Relation Abonnement – Prestation (n–n)

• Le modèle retenu considère que plusieurs prestations peuvent appartenir à un même abonnement et qu'une prestation peut apparaître dans plusieurs niveaux d'abonnement. La structure n–n est

donc la plus adaptée.
• Une autre possibilité aurait été d'intégrer les prestations directement comme attributs dans l'entité Abonnement. Cette solution est toutefois rigide, car l'ajout d'une nouvelle prestation impliquerait une modification de la structure de la table.
• Le modèle n–n est plus flexible, respecte les règles de normalisation et évite toute redondance d'information.

Séances

• Les séances spécialisées et les séances de coaching sont séparées car elles correspondent à deux fonctionnements différents : les premières sont collectives, les secondes individuelles.
• Une entité unique aurait créé des cardinalités incohérentes, alors que la séparation permet de conserver des relations simples (n–n pour les séances spécialisées, 1–n pour les séances de coaching).

Relation Client – Séance spécialisée (n–n)

• Un client peut participer à plusieurs séances spécialisées, et une séance peut accueillir plusieurs clients. La cardinalité n–n est donc nécessaire. Et en ayant cette cardinalité on doit créer une table d'association.
• Une relation 1–n aurait été incorrecte car elle supposerait qu'une séance ne possède qu'un seul participant, ce qui contredit la notion de cours collectif.
• La relation n–n permet de représenter avec justesse les inscriptions aux séances spécialisées.

La date d'inscription dépend du lien entre un client et une séance, et non d'une seule des deux entités. Elle doit donc être placée dans la table d'association pour permettre à chaque client d'avoir une date d'inscription différente pour chaque séance.

Relation Séance spécialisée – Coach (n–1)

• Un coach peut animer plusieurs séances spécialisées, tandis qu'une séance est encadrée par un seul coach. La relation est donc de type 1–n.
• Une modélisation en n–n aurait supposé qu'une séance puisse être animée par plusieurs coachs, ce qui n'est pas demandé et rendrait le modèle inutilement complexe.

Relation Séance de coaching – Client (n–1)

• Une séance de coaching individuel concerne un seul client, mais un client peut participer à plusieurs séances. La relation retenue est donc 1–n. Ici la date d'inscription appartient directement à la séance, car il n'y a qu'un seul client par séance et aucune table d'association n'est nécessaire.
• Une relation n–n aurait impliqué qu'une séance individuelle puisse concerner plusieurs clients, ce qui n'a pas de sens dans le cadre d'un coaching individuel.

Relation Séance de coaching – Coach (n–1)

• Un coach peut encadrer plusieurs séances de coaching, tandis qu'une séance individuelle implique toujours un seul coach. La cardinalité 1–n est donc la plus appropriée. D'ailleurs « un client est suivi par un coach attitré » à travers cette table seance_coaching. Mais on aurait également pu mettre directement un lien entre coach et client on aurait une relation 1-n qui dit que un coach a plusieurs clients et un client est suivi par un coach attitré.

Relation Type de machine – Machine (1–n)

• Un type de machine (par exemple un tapis de course) peut correspondre à plusieurs machines physiques différentes, tandis qu'une machine appartient toujours à un seul type. La structure 1–n est donc naturelle.
• Supprimer l'entité Type_machine pour intégrer ses attributs directement dans Machine aurait entraîné une duplication des informations, notamment concernant l'intervalle de révision.

Relation Machine – Maintenance (1–n)

• Une machine peut recevoir plusieurs opérations de maintenance au fil du temps, tandis qu'une maintenance ne concerne qu'une seule machine. La relation retenue est donc 1–n.
• Cette structure permet d'assurer un suivi clair de l'historique des interventions et de déterminer facilement les machines qui doivent être révisées.

V. Traduction de la modélisation en script SQL

Eléments techniques principaux

À partir du modèle Entité–Association présenté précédemment, nous avons construit le schéma relationnel implémenté dans PostgreSQL à l'aide d'un script SQL. Chaque entité du MCD a été traduite en table, et les relations ont été représentées par des clés étrangères ou par des tables d'association selon leur cardinalité.

```
CREATE SCHEMA sds;
```

Pour commencer j'ai créé un schéma dans lequel contiendra mes différentes tables. Chaque entité du modèle entité-association sera transformé en table et en les créant je dois précéder leur nom par le nom de schéma.

Par exemple la création de la table client :

```
create table sds.client(
    id integer primary key,
    nom text,
    prenom text
);
```

Chaque table doit avoir des attributs et leurs types et doit avoir aussi une **clé primaire(**primary key) qui est un identifiant de la table qui est *unique* et *non nulle*.

Pour les tables ou on a des **relation 1-n** on parle de **clé étrangère** (Foreign key), on doit mettre la clé étrangère dans la table ou ya le n(plusieurs) avec l'expression **machine_id integer REFERENCES sds.machine(id));** de la table maintenance par exemple.

```
create table sds.Maintenance(
   id integer primary key,
   date_maintenance Date,
   machine_id integer references sds.machine(id)
   );
```

Dans les cas où on a une **relation n-n** notamment nos tables client et seance-specialisee et aussi les tables prestation et abonnement on doit créer une table d'association qui contiendra l'id des deux tables associées comme *clé étrangère* et aussi des fois comme *clé primaire* (mais ce n'est pas obligatoire par contre on serait obligé de mettre pour ces deux attributs une contrainte d'unicité qui dit que ce couple ne doit pas se dupliquer dans notre table). La colonne date_inscription est placée dans cette table d'association car elle dépend du couple (client, séance) et non de l'une des deux tables seules, et aussi on a le montant et la date de paiement de la séance comme ça on peut savoir quel paiement a effectué quel client et pour quelle séance.

```sql
CREATE TABLE sds.inscription_seance_specialisee (
    client_id              INTEGER,
    seance_specialisee_id  INTEGER,
    date_inscription       DATE,
    date_paiement          DATE,
    montant                INTEGER,
    PRIMARY KEY (client_id, seance_specialisee_id),
    FOREIGN KEY (client_id)REFERENCES sds.client(id),
    FOREIGN KEY (seance_specialisee_id)REFERENCES sds.seance_specialisee(id)
);
```

VI. Explication du code java

Le code Java permet d'interagir avec la base de données PostgreSQL afin de répondre aux questions du sujet. Voici une explication des éléments techniques principaux :

Connexion à la base de données :

Le programme charge le driver PostgreSQL, ouvre une connexion à la base avec les identifiants fournis (URL, utilisateur, mot de passe), et confirme si la connexion réussit.

```java
// ======== Configuration connexion ========
private static final String URL = "jdbc:postgresql://kafka.iem/ad057513";
private static final String USER = "ad057513";
private static final String PASSWORD = "ad057513";

Run | Debug
public static void main(String[] args) {

    try (Scanner sc = new Scanner(System.in)) {

        // Etape 1 : charger le driver
        Class.forName(className: "org.postgresql.Driver");

        // Etape 2 : etablir la connexion (1 seule connexion pour tout le menu)
        try (Connection connexion = DriverManager.getConnection(URL, USER, PASSWORD)) {
            System.out.println(x: "Connexion reussie !");
```

Le Menu :

Cette méthode affiche le menu principal du programme dans la console afin de présenter les différentes fonctionnalités accessibles à l'utilisateur. Elle permet de guider l'utilisateur dans la navigation du programme en lui proposant des choix numérotés traités par des switch et des cases.

```
// ================= MENU =================
private static void afficherMenu() {
    System.out.println(x: "===== MENU =====");
    System.out.println(x: "1: A propos de la maintenance des machine");
    System.out.println(x: "2: Liste des seances qu'un client a beneficie");
    System.out.println(x: "3: Liste des clients qui n'ont pas participe ou a peu de seances")
    System.out.println(x: "4: Liste des clients qui auraient ete financierement avantageux en
    System.out.println(x: "5: Liste des coachs les plus sollicites");
    System.out.println(x: "0: Quitter");
    System.out.println(x: "-----------------------");
}
```

```
switch (choix) {
    case 1:
        executerQuestion1(connexion);
        break;
    case 2:
        executerQuestion2(connexion, sc);
        break;
```

Exemple d'exécution des requêtes :

Requête sans paramètre :

L'image ci-dessous correspond à la **déclaration de la méthode** qui exécute le traitement de la question 1, ("q1" contient la requête) . Elle reçoit une connexion JDBC en paramètre afin de pouvoir interroger directement la base de données PostgreSQL.

```
private static void executerQuestion1(Connection connexion) {
    String q1 = "SELECT "
```

L'image ci-dessous correspond à **l'exécution de la requête SQL**. Le programme crée un objet Statement, exécute la requête définie précédemment et récupère les résultats sous forme d'un ResultSet.

```
try (Statement statement = connexion.createStatement();
     ResultSet rs = statement.executeQuery(q1)) {
```

L'image ci-dessous correspond au traitement **l'affichage des résultats**. Le programme parcourt les lignes retournées par la requête et affiche, pour chaque machine concernée, son identifiant, son nom, la date de la dernière maintenance et la date estimée de la prochaine maintenance.

```
while (rs.next()) {
    System.out.printf(
            format: "%-10d | %-12s | %-20s | %-22s%n",
            rs.getInt(columnLabel: "machine_id"),
            rs.getString(columnLabel: "machine_nom"),
            rs.getDate(columnLabel: "derniere_maintenance"),
            rs.getTimestamp(columnLabel: "prochaine_maintenance")
    );
}
```

L'image ci-dessous correspond à la gestion **des erreurs SQL éventuelles**. Si un problème survient lors de l'exécution de la requête ou de l'accès à la base, l'exception est capturée et affichée pour faciliter le débogage.

```
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

Requête avec paramètre :

Cette méthode a un paramètre de plus que la première, c'est une méthode paramétrée, elle donne l'occasion à l'utilisateur de saisir des informations pour interroger la base de donnée.

```
private static void executerQuestion2(Connection connexion, Scanner sc) {
```

Le code ci-dessous prépare une requête SQL sécurisée et y insère l'identifiant du client saisi par l'utilisateur afin d'exécuter la requête avec les données correspondant uniquement à ce client. Dans la requête on a des points d'interrogations à la place de l'id, donc c'est là-bas que l'id qui a été saisi par l'utilisateur va être placer(1 pour le premier point d'interrogation et 2 pour le deuxième).

```
String q2 = "SELECT 'specialisee' AS type, ss.date_seance, ss.theme, ss.duree " +
        "FROM sds.inscription_seance_specialisee iss " +
        "JOIN sds.seance_specialisee ss ON ss.id = iss.seance_specialisee_id " +
        "WHERE iss.client_id = ? " +
        "UNION ALL " +
        "SELECT 'coaching' AS type, sc.date_seance, 'Coaching' AS theme, sc.duree " +
        "FROM sds.seance_coaching sc " +
        "WHERE sc.client_id = ? " +
        "ORDER BY date_seance;";
```

```
try (PreparedStatement ps = connexion.prepareStatement(q2)) {
    ps.setInt(parameterIndex: 1, clientId);
    ps.setInt(parameterIndex: 2, clientId);
```

VIII. Requêtes SQL

Les requêtes SQL suivantes répondent aux différentes questions définies dans

le sujet. Voici une description des éléments techniques principaux :

- **Machines qui doivent subir une opération de maintenance sous un délai de 1 mois :**

```sql
SELECT
    m.id AS machine_id,
    m.nom AS machine_nom,
    MAX(maint.date_maintenance) AS derniere_maintenance,
    MAX(maint.date_maintenance) + tm.intervalle_revision * INTERVAL '1 day'
        AS prochaine_maintenance
FROM sds.machine m
JOIN sds.type_machine tm
    ON m.type_machine_id = tm.id
JOIN sds.maintenance maint
    ON maint.machine_id = m.id
GROUP BY m.id, m.nom, tm.intervalle_revision
HAVING
    (MAX(maint.date_maintenance) + tm.intervalle_revision * INTERVAL '1 day')
        BETWEEN DATE '2025-12-12' AND DATE '2026-01-12'
ORDER BY m.id;
```

- La requête joint les tables machine, type_machine et maintenance pour exploiter les informations de maintenance par machine.
- MAX(date_maintenance) permet d'identifier la **dernière maintenance** effectuée pour chaque machine.
- La **prochaine maintenance** est calculée en ajoutant l'intervalle de révision (en jours) à la dernière date de maintenance.
- Le GROUP BY permet d'appliquer les fonctions d'agrégation par machine.
- La clause HAVING permet de filtrer les machines dont la prochaine maintenance est prévue dans une période donnée. Cette période a été fixée afin d'être cohérente avec les dates présentes dans la base de données et de garantir un résultat lors de l'exécution de la requête. Une alternative aurait été d'utiliser la **date courante,** mais cela aurait pu ne retourner aucun résultat lors de la correction si les dates de maintenance stockées ne correspondent pas à la date d'exécution.
- ORDER BY organise l'affichage des résultats par identifiant de machine.

➢ **Pour un client donné en paramètre, lui retourner la liste des séances spécialisées et des séances de coaching spécialisées dont il a bénéficié.**

```
SELECT
  'specialisee' AS type,
  ss.date_seance,
  ss.theme,
  ss.duree
FROM sds.inscription_seance_specialisee iss
JOIN sds.seance_specialisee ss
  ON ss.id = iss.seance_specialisee_id
WHERE iss.client_id = ?

UNION ALL

SELECT
  'coaching' AS type,
  sc.date_seance,
  'Coaching' AS theme,
  sc.duree
FROM sds.seance_coaching sc
WHERE sc.client_id = ?
ORDER BY date_seance;
```

- La requête combine les **séances spécialisées** et les **séances de coaching** d'un même client à l'aide de UNION ALL.
- Le premier SELECT récupère les séances spécialisées suivies par le client via la table d'inscription et la jointure avec seance_specialisee.
- Le second SELECT récupère les séances de coaching associées directement au client.
- Une colonne constante ('specialisee' ou 'coaching') permet d'identifier le type de séance dans le résultat.
- Le paramètre **?** permet de filtrer dynamiquement les résultats pour un client donné (requête préparée).
- Le ORDER BY date_seance trie l'ensemble des séances par ordre chronologique, tous types confondus.

- **les clients abonnés qui ne participent pas (ou participent peu) aux séances spécialisées disponibles dans leur abonnement :**

```sql
SELECT
    c.id,
    c.nom,
    c.prenom,
    COUNT(ss.id) AS nb_seances_inclues
FROM sds.client c
JOIN sds.abonnement a
    ON a.client_id = c.id
JOIN sds.abonnement_prestation ap
    ON ap.id_abonnement = a.id
JOIN sds.prestation p
    ON p.id = ap.id_prestation
   AND p.type_prestation = 'Seances specialisees'
LEFT JOIN sds.inscription_seance_specialisee iss
    ON iss.client_id = c.id
LEFT JOIN sds.seance_specialisee ss
    ON ss.id = iss.seance_specialisee_id
   AND ss.date_seance BETWEEN a.date_debut AND a.date_fin
   AND iss.montant = 0
GROUP BY c.id, c.nom, c.prenom
HAVING COUNT(ss.id) <= 1
ORDER BY c.id;
```

- La requête sélectionne les clients dont l'abonnement donne accès à la prestation **« Séances spécialisées »** (jointure abonnement , abonnement_prestation et prestation).
- Elle relie ensuite ces clients à leurs inscriptions via inscription_seance_specialisee et aux séances via seance_specialisee (LEFT JOIN pour conserver aussi ceux sans inscription).
- Elle ne compte que les séances **incluses dans l'abonnement** grâce aux conditions ss.date_seance BETWEEN a.date_debut AND a.date_fin et iss.montant = 0.
- Le COUNT(ss.id) calcule le nombre de séances spécialisées incluses réellement utilisées par chaque client.
- La clause HAVING COUNT(ss.id) <= 1 filtre les clients qui ne participent pas ou participent peu (0 ou 1 séance).
- ORDER BY c.id trie l'affichage des résultats par identifiant de client.

Le script SQL, l'ensemble des requêtes et le code java fournis en annexe.

IX. Conclusion

Ce projet m'a permis d'appliquer concrètement les notions de modélisation et d'interrogation de bases de données relationnelles à travers le langage SQL et l'utilisation de PostgreSQL. La conception du schéma et l'écriture des requêtes ont permis de répondre à des problématiques réelles liées à la gestion d'une salle de sport.

L'intégration de la base de données avec une application Java via JDBC m'a également aidé à mieux comprendre le lien entre une application et une base de données.

Ce travail m'a enfin permis de renforcer mon autonomie dans l'analyse d'un sujet et dans la mise en œuvre de solutions cohérentes et fonctionnelles.