

Documents de cours interdits sauf 1 feuille recto-verso.

Répondre de manière claire et concise. Le barème est indicatif.

Répondre exclusivement sur les feuilles réponses fournies avec le sujet.

I. Contexte

Un ensemble d'amis décide de partager ses gouts musicaux via une petite application indépendante, et de stocker les informations dans une base de données. Cette base décrit les utilisateurs de l'application, leurs playlists composés de chansons. Chaque chanson appartient à un album qui est, lui-même, lié à un artiste.

USER (NAMEU, MAIL)

$\{(n, m) \in \text{USER} \Leftrightarrow \text{Un utilisateur est identifié par son nom } n \text{ et possède un email } m.\}$

ARTIST (NAMEA, BIOGRAPHY, TYPE)

$\{(n, b, g) \in \text{ARTIST} \Leftrightarrow \text{Un artiste est identifié par son nom } n, \text{ il possède une biographie } b, \text{ et est rattaché à un genre musical } g.\}$

SONG (TITLES, DURATION, TITLEA)

$\{(t, d, a) \in \text{SONG} \Leftrightarrow \text{Une chanson est identifiée par son titre } t \text{ et possède une durée } d \text{ en secondes. Une chanson est associée à un album de titre } a.\}$

PLAYLIST (NAMEP, DESCRIPTION, DURATION, NAMEU)

$\{(n, d, t, u) \in \text{PLAYLIST} \Leftrightarrow \text{Une liste de lecture est identifiée par nom nom } n, \text{ possède une description } d \text{ et une durée totale } t \text{ (la somme des durées des chansons qui la compose). Une playlist appartient à un utilisateur de nom } u.\}$

ALBUM (TITLEA, RELEASEDATE, NAMEA)

$\{(t, rd, a) \in \text{ALBUM} \Leftrightarrow \text{Un album est identifié par son titre } t, \text{ possède une date de parution } rd \text{ et est associé à un artiste de nom } a.\}$

PLAYLIST_SONG(NAMEP, TITLES)

$\{(n, t) \in \text{PLAYLIST_SONG} \Leftrightarrow \text{Une chanson identifiée par son titre } t \text{ est rattachée à la playlist identifiée par son nom } n. \text{ Une chanson peut être rattachée à plusieurs playlist.}\}$

MUSICALTYPE (TYPE)

$\{(t) \in \text{MUSICALTYPE} \Leftrightarrow \text{un genre musical est identifié par son type } t.\}$

domaine(NAMEA, NAMEP, NAMEU, TITLEA, TITLES, MAIL, BIOGRAPHY, DESCRIPTION) =

Chaine de caractères

domaine(TYPE) = {'R&B','RAP','SLAM','VARIETY','CLASSIC'}

domaine(DURATION) = entier > 0

domaine(REALLEASEDATE) = date

$\text{SONG}[\text{TITLEA}] \subseteq \text{ALBUM}[\text{TITLEA}]$

$\text{PLAYLIST}[\text{NAMEU}] \subseteq \text{USER}[\text{NAMEU}]$

$\text{ALBUM}[\text{NAMEA}] \subseteq \text{ARTIST}[\text{NAMEA}]$

$\text{ALBUM}[\text{TYPE}] \subseteq \text{MUSICALTYPE}[\text{TYPE}]$

$\text{PLAYLIST_SONG}[\text{NAMEP}] \subseteq \text{PLAYLIST}[\text{NAMEP}]$

$\text{PLAYLIST_SONG}[\text{TITLES}] \subseteq \text{SONG}[\text{TITLES}]$

II. Requêtes (6 points)

Q1) Quelles sont les utilisateurs qui partagent les mêmes artistes que l'utilisateur Nina dans leurs playlist (ils peuvent en avoir d'autres mais au moins tous ceux de Nina) ? Le résultat attendu est une liste de NAMEU. Vous donnerez la version algébrique de cette requête qui doit être pensée comme une division.

Q2) Quel(s) genre musical(s) apparaît(sent) le plus dans les playlists ? (Une chanson récupère le genre musical de l'artiste associé à l'album auquel appartient la chanson). Le résultat attendu est une liste de MUSICALTYPE. Donner la requête SQL qui renvoie un résultat exact sans clause LIMIT et ORDER BY.

Q3) Pour chaque utilisateur, donner le nombre de chansons dans ses playlists pour chaque genre musical. Le résultat sera de la forme (NAMEU, MUSICALTYPE, NB). Dans le cas où un genre musical n'a aucune chanson associée dans ses playlists, alors le résultat renvoyé doit être de la forme (NAMEU, MUSICALTYPE, 0). Pour construire cette requête on décomposera le problème en 4 étapes: 1) construire l'association de tous les utilisateurs à tous les types musicaux (produit cartésien), 2) construire l'association de tous les utilisateurs aux types musicaux qui correspondent aux artistes des albums des chansons de leurs playlists (jointures internes classiques), 3) combiner les deux ensembles en privilégiant le premier (jointure externe), 4) comptabiliser les chansons par utilisateur et par type musicaux (agrégation avec groupement). Vous compléterez la requête proposée.

III. Normalisation (2 points)

Les informations de l'album sont enrichies de nouvelles données et le schéma suivant est proposé :

ALBUM2 (TITLEA, RELEASEDATE, NAMEA, VERSION, NAMEED, RSED, NINSEE, ADED)

$\{(t, rd, a, v, ne, rs, n, ad) \in \text{ALBUM} \Leftrightarrow \text{Un album est identifié par son titre } t, \text{ possède une date de parution } rd, \text{ est associé à un artiste de nom } a, \text{ correspond à la version } v, \text{ est édité par l'éditeur de nom } ne, \text{ de raison social } rs, \text{ de matricule } n, \text{ d'adresse } ad.\}$

Avec les dépendances fonctionnelles suivantes :

TITLEA → NAMEA

TITLEA, RELEASEDATE → VERSION

NAMEED → RSED, NINSEE, ADED

NINSEE → NAMEED

TITLEA, VERSION → RELEASEDATE, NAMEED

Q4) Donner le graphe minimal de dépendance et préciser la (ou les) clé(s) de la relation ALBUM2.

Q5) Donner en justifiant la forme normale de ALBUM2.

Q6) Proposer une décomposition optimale de cette relation (aucune table résultante ne peut être décomposée).

IV. SPRING - JPA (12 points)

Cette section est indépendante des sections I, II et III.

Q7) QCM: 10 questions (5 points)

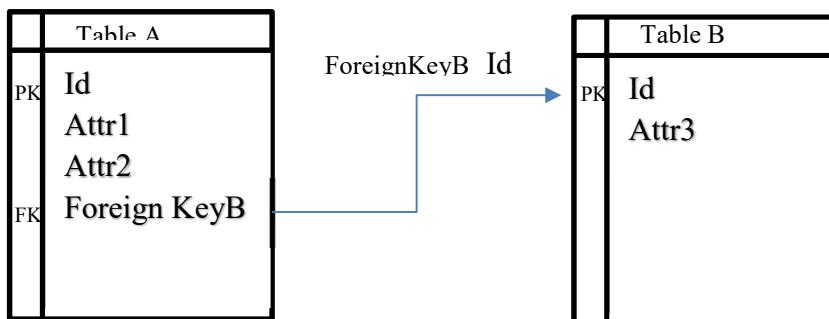
- 7.1) Qu'est-ce que JPA ?
- a) Une bibliothèque de traitement de fichiers JavaScript.
 - b) Une spécification de Java pour la gestion de la persistance des données.
 - c) Un framework de développement web en Java.
 - d) Une méthode d'authentification sécurisée pour les applications Java.
- 7.2) Quelle est la signification de l'annotation `@Entity` en JPA ?
- a) Indique à JPA que la classe est un bean de session.
 - b) Indique à JPA que la classe est une entité persistante.
 - c) Spécifie la configuration de l'unité de persistance.
 - d) Configure le contexte de persistance.
- 7.3) Qu'est-ce que JPQL (Java Persistence Query Language) ?
- a) Un langage de requête SQL spécifique à JPA.
 - b) Un langage de programmation pour les applications Java.
 - c) Un langage de requête orienté objet pour interagir avec les entités JPA.
 - d) Un langage de balisage pour la définition des entités JPA.
- 7.4) Comment JPA prend-il en charge la cartographie de l'héritage ?
- a) Stratégies de table unique, de jointure et de table par hiérarchie de classes.
 - b) Stratégies de table unique et de jointure uniquement.
 - c) Stratégies de table unique et de table par sous-classe uniquement.
 - d) Stratégies de table par hiérarchie de classes et de table par sous-classe uniquement.
- 7.5) Quel est le comportement par défaut du chargement paresseux (lazy loading) en JPA ?
- a) Une association est chargée dès que l'entité parente est chargée.
 - b) Une association est chargée uniquement lorsque l'attribut lié à cette association est explicitement accédé.
 - c) Les associations sont chargées de manière asynchrone.
 - d) Les associations ne sont jamais chargées.
- 7.6) Quelle option de fetch par défaut est utilisée pour charger les associations `@ManyToOne` et `@OneToOne` dans JPA ?
- a) FetchType.EAGER
 - b) FetchType.LAZY
- 7.7) Quel est le rôle des cascades dans les relations d'entités en JPA ?
- a) Elles déterminent la façon dont les requêtes SQL sont exécutées.
 - b) Elles permettent de spécifier le niveau de visibilité des entités.
 - c) Elles contrôlent le comportement de persistance des entités associées.
 - d) Elles facilitent la gestion des transactions dans les applications.
- 7.8) Quelle annotation JPA est utilisée pour spécifier le nom de la table associée à une entité ?
- a) `@Table`
 - b) `@EntityTable`
 - c) `@DatabaseTable`
 - d) `@EntityName`

- 7.9) Quelle est la fonction de l'annotation `@JoinColumn` dans une relation entre deux entités en JPA ?
- Elle définit le type de jointure à utiliser lors de la récupération des données.
 - Elle spécifie le nom de la colonne de jointure dans la table associée.
 - Elle indique le nom de la colonne à utiliser comme clé primaire dans la table principale.
 - Elle configure le comportement de chargement paresseux pour la relation.
- 7.10) Quel est le rôle de l'attribut `mappedBy` de l'annotation `@OneToMany` en JPA ?
- Il spécifie le nom de la colonne dans la table de la base de données utilisée pour mapper la relation.
 - Il indique le nom de la propriété dans l'entité associée qui maintient la relation inverse.
 - Il définit la stratégie de chargement de la relation, c'est-à-dire si elle doit être chargée de manière paresseuse ou immédiate.
 - Il précise le nom de la classe de l'entité propriétaire dans la relation many-to-one.

Q8) Le résultat d'une situation en paradigme *create* (7 points)

A moins de 3 mois des Jeux Olympiques, une société vous demande de créer un modèle applicatif permettant de simuler les Jeux Olympiques. Vous avez proposé un schéma, que vous trouverez dans l'annexe.

- 8.1) Donner le schéma relationnel créé par JPA après le démarrage de Spring. Veuillez préciser les noms des tables, les clés primaires, les clés étrangères et les relations entre les tables en suivant le modèle de représentation graphique ci-dessous.



- 8.2) Quelle est la fonction de l'annotation `@GeneratedValue` en JPA ?
- Elle définit le type de génération de la valeur de la clé primaire de l'entité.
 - Elle spécifie que la valeur de la clé primaire doit être générée automatiquement par la base de données.
 - Elle indique que la valeur de la clé primaire doit être calculée en fonction des autres attributs de l'entité.
 - Elle permet de définir la stratégie de génération de la valeur de la clé primaire, comme l'incrémentation automatique ou la génération par séquence.
- 8.3) Définir le repository de l'entité 'OlympiadEntity' et ajouter une fonction nommée qui compte le nombre d'événements qui existent dans l'olympiade entre deux années ('int year', voir annexe).

V. Annexe Spring JPA

AthleteEntity :

```

@Entity
public class AthleteEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;

    @ManyToOne
    private CountryEntity country;

    @ManyToMany(mappedBy = "athletes")
    private Set<EventEntity> events;
}

```

CompetitionEntity :

```

@Entity
public class CompetitionEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String location;

    @ManyToOne
    private EventEntity event;

    @OneToMany(mappedBy = "competition")
    private Set<ResultEntity> results;
}

```

CountryEntity :

```

@Entity
public class CountryEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;

    @OneToMany(mappedBy = "country")
    private Set<AthleteEntity> athletes;
}

```

EventEntity :

```

@Entity
public class EventEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;

    @ManyToOne
    private OlympiadEntity olympiadEntity;

    @OneToMany(mappedBy = "event")
    private Set<CompetitionEntity> competitionEntities;

    @ManyToMany
    private Set<AthleteEntity> athletes;
}

```

OlympiadEntity :

```
@Entity
public class OlympiadEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private int year;

    @OneToOne(mappedBy = "olympiadEntity")
    private Set<EventEntity> events;
}
```

ResultEntity :

```
@Entity
public class ResultEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private int rank;

    @ManyToOne
    private AthleteEntity athlete;

    @ManyToOne
    private CompetitionEntity competition;
}
```