Projet

Consignes générales

Projet à faire en groupe.

Combien ?
Minimum 2 personnes, maximum 3 personnes.
Veuillez indiquer la composition de vos groupes dans ces documents :
- A2S : A2S - Administration BDD - Liste des groupes.xlsx
- A2MSI : A2MSI - Administration BDD - Liste des groupes.xlsx

Quand ?
Le projet doit être rendu le lundi 11 décembre avant 23h59.
Ensuite, le mardi 12 décembre, vous allez présenter votre projet.

Quoi ?
Vous devez rendre :
- Les scripts SQL qui vous ont permis de faire le projet ;
- Autres fichiers (explications diverses, MCD, MLD etc)
- Facultatif : un support de présentation pour votre soutenance

Étape 0 : Préparation

1. Télécharger le fichier `project.csv`

2. Avec MySQL Workbench, créer une base de données nommée : `project`

3. Importer le fichier `project.csv` dans votre base de données à l'aide de la méthode de votre choix.

4. Explorer la base de données en faisant les requêtes pour afficher les informations mentionnées ci-dessous.

Remarque :

- Ces requêtes doivent être sauvegardées dans un script SQL nommé `preparation.sql`. Vous devez à chaque fois préciser l'information que vous souhaitez afficher en commentaire et ensuite la requête SQL.
  Exemple :
  ```
  -- Afficher la structure d'une table
  DESCRIBE nom_de_table;
  ```

- L'attribut `year` correspond à l'année du cursus de l'étudiant (1ère année, 2ème année etc.)

Les requêtes SQL à faire sont les suivantes :

a. Afficher l'ensemble des tables en SQL
b. Afficher les colonnes de la table "project"
c. Le nombre d'étudiants dans la base de données
d. Les différents cours dans la base de données
e. Les différentes maisons dans la base de données
f. Les différents préfets dans la base de données
g. Quel est le préfet pour chaque maison ?
h. Pour compter le nombre d'étudiants par année
i. Pour afficher les noms et les emails des étudiants qui suivent le cours "potion"
j. Pour trouver le nombre d'étudiants de chaque maison qui suivent le cours "potion"
k. Afficher les maisons des étudiants et le nombre d'étudiants dans chaque maison
l. Afficher les maisons des étudiants et le nombre d'étudiants dans chaque maison, triés par ordre décroissant
m. Afficher le nombre d'étudiants inscrits à chaque cours, triés par ordre décroissant
n. Afficher les préfets de chaque maison, triés par ordre alphabétique des maisons

Étape 1 - Normaliser le schéma

Remarque :
Les explications que vous donnez doivent être dans un fichier `normalisation.txt`.
Vos requêtes doivent être sauvegardées dans un script SQL nommé `normalisation.sql`.
Vous devez à chaque fois préciser l'information que vous souhaitez afficher en commentaire et ensuite la requête SQL.

Exemple :

```
-- Afficher la structure d'une table
DESCRIBE nom_de_table;
```

1. Expliquer pourquoi cette base de données n'est pas normalisée.
2. Identifier les dépendances fonctionnelles et les formes normales qui ne sont pas respectées.
3. Proposer une normalisation du schéma. Expliquer les modifications apportées pour normaliser le schéma.
4. Faire le MCD du schéma normalisé.
5. Faire le MLD du schéma normalisé.
6. Trouvez un moyen (ou plutôt une commande) pour sauvegarder votre base de données avant d'effectuer vos modifications. Garder précieusement votre sauvegarde. PS. NON, faire un copier-coller de votre fichier SQL n'est **PAS** un moyen de sauvegarde accepté.
7. Maintenant, passons aux choses sérieuses, vous devez modifier votre base de données pour la normaliser avec des requêtes SQL.

Vous devez faire des requêtes SQL pour chacune de ces étapes :

i. Regrouper les attributs qui dépendent fonctionnellement les uns des autres en des tables distinctes. Donc, créer des tables normalisées.
ii. Créer une clé primaire pour chaque table nouvellement créée.
iii. Ajouter des clés étrangères pour les tables qui ont des dépendances fonctionnelles avec d'autres tables.
iv. Supprimer des données si nécessaire.

Étape 2 – Les transactions

Remarque :
Les explications que vous donnez doivent être dans un fichier `transaction.txt`.
Vos requêtes doivent être sauvegardées dans un script SQL nommé `transaction.sql`.
Vous devez à chaque fois préciser l'information que vous souhaitez afficher en commentaire et ensuite la requête SQL.

Exemple :

```
-- Afficher la structure d'une table
DESCRIBE nom_de_table;
```

1. Expliquer ce qu'est une transaction en base de données.

2. Présenter les propriétés ACID des transactions.

3. Ajout d'un étudiant et rollback :

   a) Démarrez une transaction.
   b) Ajoutez un nouvel étudiant dans la base de données avec des détails fictifs.
   c) Vérifiez les détails du nouvel étudiant ajouté mais avant de valider la transaction.
   d) Utilisez ROLLBACK pour annuler l'ajout.
   e) Vérifiez à nouveau pour confirmer que l'étudiant n'a pas été ajouté de manière permanente.

4. Modification multiple et commit :

   a) Démarrez une transaction.
   b) Effectuez plusieurs opérations : par exemple, changez la maison d'un étudiant et mettez à jour le cours d'un autre étudiant.
   c) Vérifiez les modifications dans la base de données.
   d) Utilisez COMMIT pour valider les modifications.
   e) Vérifiez les informations des étudiants pour confirmer que les modifications sont permanentes.

5. Transaction avec erreur et rollback

   a) Démarrez une transaction.
   b) Effectuez une série d'opérations, dont au moins une doit échouer (par exemple, tenter d'ajouter un étudiant avec un email déjà existant).
   c) Observez le comportement de la transaction en cas d'erreur.
   d) Utilisez ROLLBACK pour annuler toutes les opérations de la transaction, y compris celles effectuées avant l'erreur.

Partie 3 – Les vues

1. Expliquer à quoi sert une vue en base de données. Expliquer quelle est la différence entre une vue logique et une vue matérialisée.

2. Trouver les requêtes SQL afin de :

   a. Créer une vue logique qui affiche le nom, l'email et la maison de chaque étudiant qui suit un cours de potions.

   b. Afficher le résultat de la vue.

   c. Rajouter 2 étudiants qui suivent un cours de potion.

   d. Afficher (encore) le résultat de la vue.

3. Modification interdite d'une vue :

Créer une vue `house_student_count` qui regroupe les étudiants par maison et compte le nombre d'étudiants dans chaque maison.

Il y aura donc 2 colonnes : une colonne `house_name` avec le nom des maisons et une colonne `student_count` avec le nombre d'étudiants par maison.

Cette vue va utiliser des fonctions d'agrégation (`COUNT` et `GROUP BY`).

Essayer de modifier la colonne contenant le nombre d'étudiants dans une maison. Par exemple, pour la maison Gryffondor, définir le nombre d'étudiants à 10.

Est-ce que cette requête génère une erreur ? Et si la vue `house_student_count` avait été une table normale, est-ce que cette requête aurait fonctionné ?

Partie 4 - Influence d'un index

1. Expliquer à quoi sert un index en base de données.

2. Trouver les requêtes SQL afin de :

- compter le nombre d'étudiants qui sont dans la maison "Gryffindor" ;

- mesurer le temps de la requête avec la commande SHOW PROFILE (https://dev.mysql.com/doc/refman/8.0/en/show-profile.html) ;

- ajouter un index sur la colonne "house_id" de la table "students" ;

- mesurer à nouveau le temps de la requête après l'ajout de l'index ;

- mesurer à nouveau le temps de la requête mais sans index.

Attention : dans MySQL, un index est utilisé par défaut par le SGBD pour optimiser les requêtes. Si vous souhaitez mesurer les performances d'une requête sans utiliser l'index que vous venez de créer, vous pouvez utiliser `IGNORE INDEX`.

3. Pour les requêtes suivantes, vous devez dire à quoi correspond chaque requête. Ensuite, vous devez mesurer le temps de la requête, rajouter un index, mesurer encore une fois le temps de la requête.

Requête a

```
SELECT houses.house_name, courses.course_name, COUNT(*) AS
num_students
FROM students
JOIN houses ON students.house_id = houses.house_id
JOIN courses ON students.course_id = courses.course_id
GROUP BY houses.house_name, courses.course_name
ORDER BY num_students DESC;
```

Requête b

```
SELECT student_name, email
FROM students
WHERE course_id IS NULL;
```

Requête c

```
SELECT houses.house_name, COUNT(*) AS num_students
FROM students
JOIN houses ON students.house_id = houses.house_id
WHERE EXISTS (
  SELECT *
```

```
  FROM courses
  WHERE course_name IN ('Potions', 'Sortilèges', 'Botanique')
    AND course_id = students.course_id
)
GROUP BY houses.house_name;
```

Requête d

```
SELECT s.student_name, s.email
FROM students s
JOIN (
  SELECT student_id, year_id, COUNT(DISTINCT course_id) AS
num_courses
  FROM students
  GROUP BY student_id, year_id
) AS sub
ON s.student_id = sub.student_id AND s.year_id = sub.year_id
JOIN (
  SELECT year_id, COUNT(DISTINCT course_id) AS num_courses
  FROM students
  GROUP BY year_id
) AS total
ON s.year_id = total.year_id AND sub.num_courses =
total.num_courses
WHERE sub.num_courses = total.num_courses;
```