Examen de Bases de Données - M1 Mathématiques Appliquées

Durée : 2h - Documents autorisés (Support de cours uniquement)

Les documents sont interdits, sauf ceux qui sont présents sur la page https://irma.math.unistra.fr/~helluy/sql/sql.html

Avant de composer, vous devez télécharger et lancer le logiciel recme2. Ce logiciel doit rester actif pendant toute la durée de l'examen.

Les copies seront corrigées par un humain et pourront être vérifées par une IA il est donc recommandé d'écrire *très* lisiblement. Si vous barrez du texte, il doit être complètement masqué.

Partie 1 : Questions de cours (QCM)

Recopier **intégralement et exactement** l'énoncé et les réponses de la question 0 (entraînement de l'IA à la reconnaissance de votre écriture).

0. **Question gratuite** Que veut dire SQL ?
   ☒ Structured Query Language
   ☐ Simple Query Language
   ☐ Standard Query Language

Pour les questions suivantes, choisir la bonne réponse (une seule réponse possible par question). Expliquer brièvement votre réponse. Faites des phrases complètes: par exemple: "L'objectif principal ... est de ...".

1. **Normalisation (SQL)** : Quel est l'objectif principal de la Troisième Forme Normale (3FN) ?
   ☐ S'assurer que chaque attribut contient une valeur atomique.
   ☐ S'assurer que tous les attributs non-clés dépendent uniquement de la clé primaire (pas de dépendances transitives).
   ☐ Créer des relations M:N entre toutes les tables.
   ☐ Optimiser l'espace disque en compressant les données.
2. **NoSQL vs SQL** : Selon le cours, quel est un avantage majeur des bases NoSQL (comme MongoDB) par rapport aux bases relationnelles classiques ?
   ☐ Elles garantissent toujours des transactions ACID strictes sur plusieurs documents.
   ☐ Elles permettent de gérer des données semi-structurées avec un schéma flexible (ajout d'attributs à la volée).
   ☐ Elles utilisent le langage SQL standard pour toutes les requêtes.
   ☐ Elles sont toujours plus rapides que le SQL pour les jointures complexes.

3. **Modélisation MongoDB** : Dans le cours, pour modéliser une relation "Un Client possède plusieurs Animaux", quelle approche est recommandée pour optimiser la lecture d'un client avec ses animaux ?
   - ☐ Créer deux collections distinctes et effectuer plusieurs requêtes successives.
   - ☐ Imbriquer (Embed) le tableau des animaux directement dans le document du client.
   - ☐ Utiliser une table de liaison "Maitre" comme en SQL.
   - ☐ Dupliquer les informations du client dans chaque document animal.
4. **Neo4j / Cypher** : Quelle est la différence entre `CREATE` et `MERGE` ?
   - ☐ `CREATE` crée un noeud s'il n'existe pas, `MERGE` le crée toujours (doublons possibles).
   - ☐ `CREATE` crée toujours un nouveau noeud, `MERGE` le crée seulement s'il n'existe pas déjà (correspondance sur le pattern).
   - ☐ `MERGE` est utilisé pour supprimer des noeuds, `CREATE` pour en ajouter.
   - ☐ Il n'y a aucune différence, ce sont des synonymes.
5. **Patterns Cypher** : Que recherche le motif `MATCH (p)-[:AMI]->()-[:AMI]->(f)` vu en cours ?
   - ☐ Les amis directs de `p`.
   - ☐ Les amis des amis de `p` (niveau 2).
   - ☐ Une relation circulaire où `p` est son propre ami.
   - ☐ Deux relations indépendantes.
6. **Pandas & SQL** : Quelle méthode Pandas permet de charger directement le résultat d'une requête SQL dans un DataFrame ?
   - ☐ `pd.read_csv()`
   - ☐ `pd.from_sql()`
   - ☐ `pd.read_sql_query()`
   - ☐ `pd.sql_to_frame()`

Partie 2 : Modélisation et Conception

Contexte : Nous souhaitons modéliser les interactions entre des **Lieux** et des **Personnages** extraits d'un roman.

1. **Modèle Relationnel (SQL)** : Un *Personnage* (id, nom) peut visiter plusieurs *Lieux* (id, nom_lieu, description). Un *Lieu* peut être visité par plusieurs *Personnages*. Chaque visite se fait à une *date* précise.
   - — Proposez le schéma relationnel (tables, clés primaires, clés étrangères) pour gérer cette relation M:N.
2. **Modèle Document (MongoDB - Approche "Vue par Lieu")** : Proposez une structure de document JSON pour la collection `lieux`, où l'on souhaite accéder rapidement à la liste des personnages ayant visité ce lieu, ainsi qu'à la date de leur visite.

Partie 3 : Requêtes

Exercice 1 : SQL

Tables : - `Personnage(id, nom)` - `Lieu(id, nom, region)` - `Visite(id_perso, id_lieu, date_visite)`

1. **(Simple)** Écrire une requête pour sélectionner toutes les colonnes de la table `Personnage`.
2. Écrire une requête pour lister les noms des personnages ayant visité le lieu "La Comté".
3. Écrire une requête pour compter le nombre de visites pour chaque lieu (`nom` du lieu et nombre de visites), trié par nombre décroissant.
4. Écrire une requête pour trouver les lieux qui ont été visités plus de 10 fois (utilisation de `HAVING`).

Exercice 2 : Neo4j / Cypher

Graphe : Noeuds `:Personnage`, `:Lieu`. Relation `:A_VISITÉ {date: '...'}`.

1. **(Simple)** Écrire une requête Cypher pour trouver et retourner tous les noeuds de type `:Personnage`.
2. Écrire une requête Cypher pour créer le personnage "Frodon" et le lieu "Mordor", puis une relation `A_VISITÉ` entre eux.
3. Écrire une requête pour trouver tous les personnages qui ont visité le "Mordor".
4. **Chemin (Relation transitive)** : Écrire une requête pour trouver les "compagnons de voyage" potentiels de Frodon, c'est-à-dire les personnages qui ont visité un même lieu que "Frodon". *Indication : Utilisez un pattern du type (Frodon)-[:A_VISITÉ]->(Lieu)<-[:A_VISITÉ]-(AutrePerso).*

Exercice 3 : MongoDB

Collection `lieux` :

```
{
  "nom": "Rivendell",
  "visiteurs": [
    {"nom": "Gandalf", "date": "2023-01-01"},
    {"nom": "Frodon", "date": "2023-01-10"}
  ]
}
```

1. **(Simple)** Écrire une commande pour afficher tous les documents de la collection `lieux`.
2. Écrire une commande pour ajouter un nouveau visiteur "Aragorn" (date: "2023-10-20") au lieu "Rivendell". *Indication : Utilisez `$push`.*

3. Écrire une requête (`find`) pour trouver les lieux visités par "Gandalf" (recherche dans un tableau d'objets).
4. **Agrégation** : Écrire un code d'agrégation pour compter le nombre total de visiteurs (un visiteur unique par lieu compte pour 1) pour chaque lieu. *Indication : `$unwind` peut être utile.*

Partie 4 : Vérification du Projet

Contexte : Cette partie porte sur le code de construction de base de données à partir de texte réalisé pendant le projet.

Documents et Matériel autorisés : Vos propres codes de projet, terminal Python.

L'usage de l'IA est autorisé, si vous utilisez l'IA vous devez décrire votre démarche (réflexion préalable, prompting, tests, etc.)

Communications avec les autres étudiants interdites. Le travail doit être personnel.

Les réponses aux questions doivent être intégrées dans un *unique* document pdf ou markdown que vous devez m'envoyer par courriel à la fin de l'examen. Le sujet du courriel doit *impérativement* être "Examen Projet SQL".

Si vous avez rédigé un rapport de projet, vous pouvez l'ajouter à la fin du document envoyé par courriel.

1. Questions de compréhension et Méthodologie

1. **Objectifs** : décrire en quelques phrases quels étaient les objectifs de ce projet. Qu'avez-vous appris ?
2. **Nettoyage du graphe** : Dans le projet, quel était l'objectif d'utiliser un modèle de similarité (Cross-Encoder ou Embedding type MiniLM) ?
3. **Parsing des réponses LLM** : Les LLM ne répondent pas toujours avec le format JSON ou CSV exact demandé. Comment avez-vous géré ce problème dans votre projet ?
4. **Choix de LLM** : avez-vous testé divers choix de LLM ? Si oui lesquels ? Commentaires ?

2. Test Pratique : Extension de la liste des clients

Objectif : Vérifier que votre code fonctionne sur de nouvelles données sans (trop) modifier le code (robustesse). Indiquer les corrections éventuelles de code.

Consigne : 1. Reprenez le fichier `story.md` original. 2. Ajoutez-y le paragraphe suivant à la fin :

> *"Alors que le calme revenait à peine, le Maire, Pierre Richard, fait une entrée remarquée. C'est un homme important. Il est accompagné*

de sa tortue Caroline, âgée de 50 ans, qu'il tient prudemment dans ses mains. Juste derrière lui, Alice Merveille, une peintre excentrique, tente de maîtriser ses deux furets, Tic et Tac. Ils ont 2 ans et sont intenables."

3. Relancez votre code complet (Chunking -> Extraction -> Graphe -> SQL).
4. **Vérification** : Écrivez la requête SQL qui permet de retrouver **tous les animaux appartenant à "Alice Merveille"** et reportez le résultat obtenu.

Votre Requête SQL : > . . . . . > . . . . . .

Résultat retourné par votre base : (Copiez-collez ici les lignes de la table `Animaux` concernées) > . . . . . > . . . . . .