Aperçu du projet : Conception et administration d'une base de données pour un site d'information sur les séries télévisées

Membres:

Bozu Esat Can
Miller Adrien

EPSI 12-12-2023

TABLE DES MATIÈRES :

Objectifs du projet :

L'objectif principal de ce projet est de concevoir et de mettre en œuvre un système de base de données robuste utilisant SQL Server. La base de données est destinée à soutenir un site web qui fournit des informations détaillées sur les séries télévisées. Les principaux objectifs sont les suivants :

1. **Conception de la base de données** : Créer un schéma de base de données efficace et évolutif qui représente logiquement les données et les relations relatives aux séries télévisées, y compris les détails sur les séries, les épisodes, les personnes (acteurs, réalisateurs, producteurs), les utilisateurs et les interactions des utilisateurs (évaluations, commentaires, messages).

2. **Gestion des données** : Assurer un traitement efficace des données par la mise en œuvre de procédures stockées, de fonctions et de déclencheurs qui facilitent l'insertion, la mise à jour, la récupération et l'intégrité des données.

3. **Sécurité et gestion des rôles** : Définir et appliquer des règles d'accès et de manipulation des données en créant des rôles d'utilisateur spécifiques, tels que Administrateur, Ingénieur de données, Analyste de données, Scientifique de données, Stagiaire et Utilisateur professionnel, chacun disposant d'autorisations distinctes.

4. **Optimisation des performances** : Appliquer les meilleures pratiques en matière de conception de bases de données afin de garantir l'efficacité du traitement des données et des requêtes, notamment en ce qui concerne le partitionnement des bases de données (vertical/horizontal).

5. **Traitement des erreurs et intégrité des données** : Intégrer des mécanismes robustes de gestion des erreurs et maintenir l'intégrité des données grâce à des contraintes, des déclencheurs et la gestion des transactions.

Exigences du projet :

Le projet comprend plusieurs produits et tâches clés :

1. **Développement du schéma de la base de données** :

   Modèle conceptuel (MCD) : Développer un diagramme entité-relation (ERD) qui représente conceptuellement toutes les entités, leurs attributs et leurs relations.
   Modèle logique (MLD) : Traduire l'ERD en un schéma logique, en définissant les tables, les clés primaires, les clés étrangères et d'autres contraintes.
   Modèle physique (MPD) : Implémenter le modèle logique dans le serveur SQL, en s'assurant que le schéma est normalisé (au moins à la troisième forme normale).

2. **Script SQL** :

   Script de création de schéma (creation.sql) : Script SQL pour créer toutes les tables et relations de la base de données.
   Script d'insertion de données (insertion.sql) : Script SQL pour remplir la base de données avec des ensembles de données initiaux et réalistes.
   Script de requête (queries.sql) : Script SQL permettant d'exécuter une série de requêtes prédéfinies démontrant les capacités de la base de données.

3. **Définition des rôles et sécurité** :

   Définir les rôles et les autorisations pour les différents types d'utilisateurs qui interagissent avec la base de données.
   Mettre en œuvre des mesures de sécurité, y compris des déclencheurs pour l'audit et des contraintes pour la validation des données.

4. **Fonctions et procédures de gestion de la base de données** :

   Développer des procédures et des fonctions stockées pour les opérations courantes de la base de données.
   Créer des déclencheurs pour maintenir l'intégrité des données et gérer la redondance des données.

5. **Tests et documentation** :

   Tester minutieusement tous les aspects du système de base de données afin d'en garantir les performances et la fiabilité.
   Fournir une documentation complète détaillant les décisions de conception, le schéma et les fonctionnalités du système de base de données.

6. **Présentation et démonstration** :

   Préparer une présentation décrivant la portée, la conception et les fonctionnalités du projet.

Effectuer une démonstration en direct des capacités de la base de données, y compris l'interrogation et la manipulation des données.

7. **Contrôle des versions et soumission** :

   Utiliser un système de contrôle de version (par exemple, Git) pour gérer et documenter le processus de développement.
   Soumettez le projet final sous la forme d'une combinaison de scripts SQL, de documentation et d'un fichier zippé de l'ensemble du référentiel.

Descriptions détaillées du schéma de la base de données :

Modèle conceptuel de données (MCD) :

MCD montre les entités et les relations de haut niveau au sein de base de données pour un système d'information sur les séries télévisées :

- Genres : Représente les différents genres de séries télévisées. Chaque genre est identifié par un GenreID unique et possède un Name.
- Series : Contient toutes les séries télévisées. Chaque série possède un SeriesID unique, un Title, une Year of release, un CountryOfOrigin et un CreationDate.
- Seasons (Saisons) : Représente les saisons d'une série. Chaque saison est liée à une série et est identifiée par un SeasonID unique et possède un Number.
- Episodes : Représente les épisodes individuels d'une saison d'une série. Chaque épisode possède un numéro d'épisode unique, un titre, une durée, une date de diffusion et un résumé.
- People (Personnes) : Contient les personnes impliquées dans la série, telles que les acteurs, les réalisateurs et les producteurs. Chaque personne possède un PersonID, un FirstName et un LastName.
- Utilisateurs : Représente les utilisateurs du système, avec un UserID, un UserName, un RegistrationDate, un Age et un Gender.
- Messages : Représente les messages postés par les utilisateurs, avec un MessageID, un PostedDate, un Text, et une indication s'il s'agit d'un First_Message.
- Ratings : Contient les notes données par les utilisateurs aux séries, avec un RatingID, un Score, un Commentaire et un RatingDate.

Les relations entre ces entités indiquent comment elles interagissent les unes avec les autres. Par exemple, une série peut être associée à plusieurs genres, saisons et personnes. Les utilisateurs peuvent évaluer les séries et publier des messages.

le modèle logique de données (MLD) :

Le MLD fournit une représentation plus détaillée, y compris les attributs spécifiques de chaque entité et les relations entre elles :

- SérieGenres : Une table de liaison entre les entités Series et Genres, indiquant les genres associés à chaque série.
- SeriesPeople : Une table de correspondance qui associe des personnes à des séries, y compris un rôle par l'intermédiaire de l'entité Role.
- EpisodePeople : Associe les personnes à des épisodes spécifiques, y compris les rôles.
- Rôles : Une liste de rôles que les personnes peuvent avoir en relation avec une série ou un épisode, tels qu'acteur, réalisateur ou producteur.
- PersonalRating et History : Il semble qu'il s'agisse d'entités supplémentaires ou peut-être d'entités associatives capturant des types spécifiques d'interactions entre utilisateurs, telles que l'évaluation d'épisodes individuels et le suivi de l'historique des messages ou des fils de discussion.
- La cardinalité des relations est également spécifiée dans les diagrammes, montrant, par exemple, qu'une série peut avoir de nombreux épisodes, mais que chaque épisode est associé à une seule série.

Dictionnaire de données (exemplaire) :

Voici quelques exemples d'entrées pour un dictionnaire de données basé sur notre schéma :

- **Table des séries** :
    - **SeriesID** : INT, clé primaire, identifie de manière unique une série.
    - **Title** : NVARCHAR(255), titre de la série.
    - **Year** : INT, l'année de publication de la série.
    - **CountryOfOrigin** : NVARCHAR(100), le pays d'origine de la série.
    - **CreationDate** : DATE, la date à laquelle l'enregistrement de la série a été créé dans la base de données.

- **Table Episodes** :
    - **EpisodeID** : INT, clé primaire, identifie de manière unique un épisode.
    - **Series_SeriesID** : INT, clé étrangère, référence SeriesID dans la table Series.
    - **Title** (Titre) : NVARCHAR(255), titre de l'épisode.
    - **Durée** : INT, la durée de l'épisode en minutes.
    - **AirDate** : DATE, la date de diffusion de l'épisode.
    - **Summary** : NVARCHAR(MAX), un bref résumé de l'épisode.

- **Table d'évaluation** :
    - **RatingID** : INT, Primary Key, identifie de manière unique une évaluation.
    - **Series_SeriesID** : INT, Foreign Key, référence SeriesID dans la table Series.

- **Users_UserID** : INT, Foreign Key, fait référence à UserID dans la table Users.
- **Score** : INT, le score ou la note attribuée à une série.
- **Comment** : NVARCHAR(MAX), le commentaire de l'utilisateur sur la série.
- **RatingDate** : DATE, la date à laquelle la note a été attribuée.

Chaque entrée de notre dictionnaire de données doit fournir des détails similaires pour toutes les tables et leurs colonnes respectives, ainsi que pour toutes les tables de liaison qui résolvent les relations de plusieurs à plusieurs dans notre base de données.

Explications des choix de conception

Normalisation

La conception de la base de données suit des règles de normalisation jusqu'à la troisième forme normale (3NF) afin de garantir que la base de données est exempte de redondances indésirables et d'assurer l'intégrité des données :

Première forme normale (1NF) :

Chaque table possède une clé primaire qui identifie ses lignes de manière unique, et tous les attributs contiennent des valeurs atomiques, sans groupes ou tableaux répétitifs.

Deuxième forme normale (2NF) :

Toutes les tables sont en 1NF et tous les attributs non clés sont entièrement fonctionnels et dépendent de la clé primaire. Cela signifie qu'aucune colonne ne dépend partiellement de la clé primaire.

Troisième forme normale (3NF) :

Toutes les tables sont en 2NF et tous les attributs ne dépendent que de la clé primaire, ce qui garantit l'absence de dépendances transitives.

En respectant ces formes, la conception réduit les doublons et favorise la cohérence des données. Par exemple, la table Personnes est séparée des tables Séries et Épisodes, les relations étant établies par le biais d'entités associatives telles que SériesPeople et ÉpisodesPeople, ce qui permet d'éviter le stockage redondant des données relatives aux personnes.

Types de données

Les types de données ont été sélectionnés pour optimiser le stockage et refléter la nature des données :

INT :
utilisé pour les identifiants tels que SeriesID, PersonID, EpisodeID, etc., car ils sont numériques et séquentiels.

NVARCHAR :
choisi pour les attributs basés sur des chaînes de caractères tels que Title et Name afin de prendre en compte les caractères Unicode, ce qui permet l'internationalisation et l'utilisation de caractères spéciaux.

DATE :
utilisé pour les champs de date tels que CreationDate et AirDate afin de stocker des informations de date sans heure, ce qui est suffisant pour la granularité requise.

DATETIME :
Appliqué à des attributs tels que PostedDate dans la table Messages pour capturer l'heure exacte à laquelle les messages sont postés.
Contraintes
Les contraintes sont utilisées pour renforcer l'intégrité des données et la logique commerciale :

Contrainte de clé primaire : Assure l'unicité et une valeur non nulle pour l'identifiant de chaque entité (par exemple, SeriesID, PersonID).
Contrainte de clé étrangère : Établit des relations entre les tables et assure l'intégrité référentielle. Par exemple, Series_SeriesID dans la table Episodes fait référence à SeriesID dans la table Series.
Contrainte CHECK : Appliquée à des attributs tels que Score dans la table Ratings pour restreindre la plage des valeurs valides (par exemple, entre 0 et 10).
Contrainte d'unicité : Utilisée lorsqu'un attribut ou une combinaison d'attributs doit être unique dans la base de données, comme le nom d'utilisateur UserName.
Autres décisions de conception
Liens entre les tables : Pour gérer les relations de plusieurs à plusieurs, telles qu'une série ayant plusieurs genres et un genre étant associé à plusieurs séries, des tables de liaison telles que SeriesGenres sont utilisées.

Indexation : Des index peuvent être appliqués aux champs fréquemment recherchés afin d'améliorer les performances des requêtes. Toutefois, les décisions en matière d'indexation doivent être fondées sur les schémas d'interrogation et la taille des données.
Vues : Elles sont créées pour les requêtes complexes utilisées fréquemment, ce qui simplifie les opérations côté client et maintient la cohérence de la présentation des données.
Procédures stockées et déclencheurs : Utilisées pour encapsuler la logique d'entreprise dans la base de données, automatiser les tâches récurrentes, renforcer la sécurité et maintenir l'intégrité des données.

Modèle de sécurité basé sur les rôles

Création de rôles : Le script définit des rôles distincts dans le modèle de sécurité de la base de données. Chaque rôle correspond à un ensemble d'autorisations qui reflète les responsabilités et les besoins des différents types d'utilisateurs qui interagissent avec la base de données. Cette approche suit le principe du moindre privilège, garantissant que les utilisateurs ne se voient accorder que les autorisations nécessaires à l'exécution de leurs tâches.

Définition des utilisateurs : Des identifiants de connexion distincts sont créés pour les différents utilisateurs. Chaque utilisateur est ensuite associé à un rôle spécifique. Cela permet de responsabiliser chaque utilisateur et de personnaliser l'accès en fonction des besoins.

Gestion des mots de passe : Dans le script, des caractères génériques tels que "Password123 !" sont utilisés à des fins de démonstration.

Adhésion à un rôle : Les utilisateurs sont ajoutés aux rôles à l'aide de la syntaxe ALTER ROLE...ADD MEMBER. Cette appartenance détermine les actions que chaque utilisateur peut effectuer dans la base de données, en fonction des autorisations accordées à leurs rôles respectifs.

Attribution des autorisations

Autorisations de l'administrateur : Le rôle d'administrateur (Role_Administrator) permet de contrôler la base de données, ce qui constitue le niveau d'autorisation le plus élevé et offre effectivement les mêmes autorisations que le propriétaire de la base de données. Ce rôle doit être réservé aux utilisateurs qui doivent gérer la configuration, la sécurité et l'existence de la base de données, tels que les administrateurs de bases de données.

Permissions de l'ingénieur des données : Le rôle d'ingénieur des données est assorti d'une série d'autorisations (CREATE TABLE, ALTER, DELETE, INSERT, UPDATE, CREATE VIEW) qui permettent de créer et de modifier les structures de la base de données et de manipuler les données. Ce rôle peut être attribué aux utilisateurs chargés de maintenir et de faire évoluer le schéma et les structures de la base de données.

Permissions de l'analyste de données : Le rôle d'analyste de données se voit accorder des autorisations SELECT sur le schéma dbo, ce qui permet aux utilisateurs de récupérer des données sans pouvoir les modifier. Il s'agit d'une autorisation typique pour les rôles qui impliquent l'interrogation et la création de rapports sur les données.

Permissions de scientifique des données : Le Rôle_DataScientist dispose d'autorisations similaires à celles du Data Engineer, mais il peut également être amené à créer des tables

temporaires ou à effectuer des opérations complexes impliquant la création de nouvelles vues ou tables dans le cadre de l'analyse des données.

Permissions de l'utilisateur professionnel : Le rôle d'utilisateur professionnel est limité aux autorisations SELECT, ce qui convient aux utilisateurs qui ont besoin de visualiser et d'analyser des données, mais pas de les modifier.

Meilleures pratiques en matière de sécurité
Principe du moindre privilège : Chaque rôle est soigneusement conçu pour disposer des autorisations minimales nécessaires à l'exécution de ses fonctions, ce qui réduit le risque d'accès non autorisé aux données ou de modification de celles-ci.

Auditabilité : En ayant des identifiants et des rôles individuels, il est plus facile de vérifier les actions effectuées sur la base de données, car les activités de chaque utilisateur peuvent être suivies.

Flexibilité : Cette approche basée sur les rôles permet d'ajouter ou de retirer des utilisateurs des rôles lorsque les fonctions changent, sans qu'il soit nécessaire de réattribuer des autorisations individuelles à chaque fois.

Évolutivité : Au fur et à mesure que la base de données s'enrichit et que la base d'utilisateurs évolue, de nouveaux rôles peuvent être créés et les autorisations peuvent être ajustées pour répondre aux nouvelles exigences.

Éléments à prendre en considération et autres mesures
Chiffrement : Pour les environnements de production, il convient d'envisager la mise en œuvre d'un système de cryptage pour les données sensibles.
Révisions régulières : Examinez périodiquement les niveaux d'accès des utilisateurs et ajustez-les si nécessaire.
Surveillance : Mettre en place une surveillance pour détecter et alerter sur toute tentative d'accès non autorisé.

Défis rencontrés et solutions mises en œuvre :

Défis rencontrés et solutions :
Défi 1 : Exécution partielle des scripts
Le modèle d'exécution de SQL Server, qui peut exécuter des parties d'un script qui ne contiennent pas d'erreurs tout en ignorant les parties contenant des erreurs, a entraîné des problèmes avec des scripts partiellement exécutés et susceptibles de mal fonctionner.

La solution : Nous avons mis en place des blocs TRY...CATCH complets dans nos scripts SQL. Ce mécanisme de traitement des erreurs nous a permis de gérer plus efficacement les exceptions en annulant les transactions en cas d'erreur, en garantissant que nos scripts de base de données réussissent ou échouent complètement, en maintenant l'atomicité des transactions.

Défi 2 : Gestion efficace des erreurs
Au départ, l'identification et le débogage des erreurs prenaient beaucoup de temps, et le manque d'informations détaillées sur les erreurs rendait le dépannage difficile.

Solution : Nous avons affiné notre utilisation de TRY...CATCH en incluant la journalisation des erreurs dans les blocs CATCH. Cela a permis d'obtenir des informations détaillées sur les erreurs, notamment leur nature et leur emplacement, ce qui a simplifié le processus de débogage. Nous avons également utilisé les fonctions intégrées de SQL Server telles que ERROR_NUMBER() et ERROR_MESSAGE() pour obtenir des informations plus détaillées sur les erreurs.

Défi 3 : Courbe d'apprentissage des fonctionnalités de SQL Server
L'équipe s'est heurtée à une courbe d'apprentissage des fonctionnalités et des meilleures pratiques spécifiques à SQL Server, ce qui a eu un impact sur l'efficacité du développement initial.

La solution : Nous avons consacré du temps à l'étude de la documentation de Microsoft, des forums en ligne et des ressources communautaires. Cette autodidaxie nous a permis de mieux comprendre les capacités de SQL Server et la manière de les exploiter efficacement. Nous avons également participé à des forums de codage et demandé conseil à des administrateurs de bases de données plus expérimentés lorsque cela s'avérait nécessaire.

Procédures de test et résultats :
Procédure de test 1 : tests unitaires
Nous avons effectué des tests unitaires sur des composants individuels de la base de données, tels que les procédures stockées, les déclencheurs et les fonctions, afin de nous assurer qu'ils exécutaient correctement les opérations prévues.

Résultats : Grâce à des tests itératifs, nous avons affiné la logique de notre base de données, en veillant à ce que chaque composant se comporte comme prévu. Les anomalies ont été enregistrées et traitées dans les cycles de développement ultérieurs.

Procédure de test 2 : tests d'intégration
Après les tests unitaires, nous avons effectué des tests d'intégration pour nous assurer que les différentes parties de la base de données fonctionnaient ensemble de manière transparente. Il s'agissait notamment de tester les relations entre les tables, l'intégrité de nos données après avoir simulé des opérations CRUD (création, lecture, mise à jour, suppression) et l'efficacité de nos contraintes et déclencheurs.

Résultats : Ces tests ont mis en évidence des problèmes de non-concordance des clés étrangères et de cascade, que nous avons résolus en affinant notre schéma et nos définitions de relations.

Procédure de test 3 : tests de performance et de charge
Nous avons simulé un environnement multi-utilisateurs pour tester la base de données sous charge, en nous concentrant sur les performances des requêtes, les stratégies d'indexation et l'efficacité de nos vues et procédures stockées.

Résultats : Les tests de charge ont révélé quelques goulets d'étranglement au niveau des performances. Nous avons optimisé nos requêtes, ajusté notre stratégie d'indexation et révisé certaines de nos vues pour améliorer les performances. En outre, nous avons introduit la pagination des requêtes pour gérer plus efficacement les grands ensembles de données.

Procédure de test 4 : tests de sécurité et de contrôle d'accès basé sur les rôles
Nous avons testé le modèle de sécurité en simulant différents rôles d'utilisateurs essayant d'effectuer des actions en dehors de leurs permissions.

Résultats : Ces tests ont confirmé l'efficacité de notre modèle de sécurité basé sur les rôles. Cependant, nous avons découvert quelques privilèges involontaires, que nous avons corrigés en affinant la définition des rôles et l'octroi des autorisations.

Réflexion : Les défis et les procédures de test ont permis de mettre au point un système de base de données robuste et performant. Le processus a souligné l'importance de tests approfondis et d'une gestion proactive des erreurs. Il a également mis en évidence la valeur d'un modèle de sécurité bien défini et la nécessité de régler les performances pour garantir l'évolutivité et la fiabilité.