Travail final d'intégration[1]

MobiGame, pour vous divertir partout

- Ce travail sera réalisé **individuellement**
- À terminer pour le 12 décembre 2025 avant 23h59;
- Le travail compte pour 20 % de la note finale;

Important :

1- Tout plagiat entraine la note zéro pour ce travail. Si vous avez zéro pour ce travail, vous échouez le cours (double standard) .
2- Utiliser L'IA pour effectuer ce travail sera considéré comme du plagiat.
3- En tout temps, vous devez fournir la source de tout code utilisé qui ne provient pas des notes de cours ou de votre analyse, conception et réflexion.

Objectifs

Ce travail vise principalement à vous faire expérimenter les aspects suivants:

- Écrire de procédures stockées;
- Écrire de déclencheurs;
- Expérimenter l'aspect sécurité des données;
- Veiller à ce que les données soient cohérentes en tout temps (Transactions)

Mise en situation

La compagnie **MobiGame** est une petite PME qui œuvre dans le développement de jeux vidéo sur appareils mobiles. Sa base de données et une base de données MS SQL Sever

Cette compagnie compte deux départements distincts, « Ressources humaines **(RSH) »** et « Développement (**DEV**) ». Chaque département a sa propre base de données.

Le département RSH compte 3 employés : 1 technicienne en administration : Martine Labelle, un commis Hugo Lapointe et le responsable du département ***Saturne Lune***.

Le département DEV compte 6 employés : 5 développeurs (Guy Tares, Bien Thierry, Martin Lejeune, Lyna Pilon et Kevin Lafleur) et le responsable du département ***Alain Patoche***.

[1] Source de l'image : https://pixabay.com/fr/photos/ancienne-kent-ville-2708365/

Authentification au serveur MS SQL

Les tâches spécifiques à la gestion de « MS SQL Server » telles que créer et supprimer une base de données, gérer les rôles et autorisations, créer les connexions et les usagers des bases de données ainsi que de créer les tables dans la base de données **dbrsh** seront réservées uniquement à l'administrateur du serveur qui est **votre compte admin**

Chaque employé de l'entreprise aura sa propre **connexion** pour s'authentifier au SGBDR. Ces connexions n'auront d'autres rôles que **public**. (Serveur et BD)

Les bases de données de l'entreprise

Chacun des départements à sa propre base de données. Les données propres au département des « Ressources humaines **(RSH) »** sont conservées dans la base de données **dbrsh** et celles propres au département « Développement (**DEV**) » dans la base de données **dbdev**. À leur création, le nom de ces bases de données doit-être précédé de vos initiales.
(Exemple : **SYdbrsh**)

Base de données dbrsh

La base de données **dbrsh** contiendra les 3 tables décrites ci-dessous.

Employes	Departements	Emplois
IdEmploye :PK, IDENTITY	codeDep : PK, char(3)	typeEmploi : PK char(3)
Nom	nomDepartement	description
Prenom		salaireMax
Adresse		salaireMin
Salaire (money)		
CodeDep		
typeEmploi		
Echelon (int)		

L'augmentation des salaires des employés est contrôlée par un trigger ***CTRLAugSalaire.*** Lorsque l'échelon d'un employé est augmenté alors le salaire est augmenté du pourcentage de cette augmentation. Exemple si l'échelon est augmenté de 3 alors le salaire et augmenté de 3%

Le salaire des employés est contrôlé par un trigger ***CTRLSalaire*** : Lors de l'ajout d'un nouvel employé, ou lors de la modification du salaire, nous devons vérifier que le salaire de celui-ci se retrouve dans la fourchette des salaires. Les salaires sont en fonction des TypeEmploi et sont dans la table EMPLOIS.

Initialement, la table Employes contient tous les employés de l'entreprise.

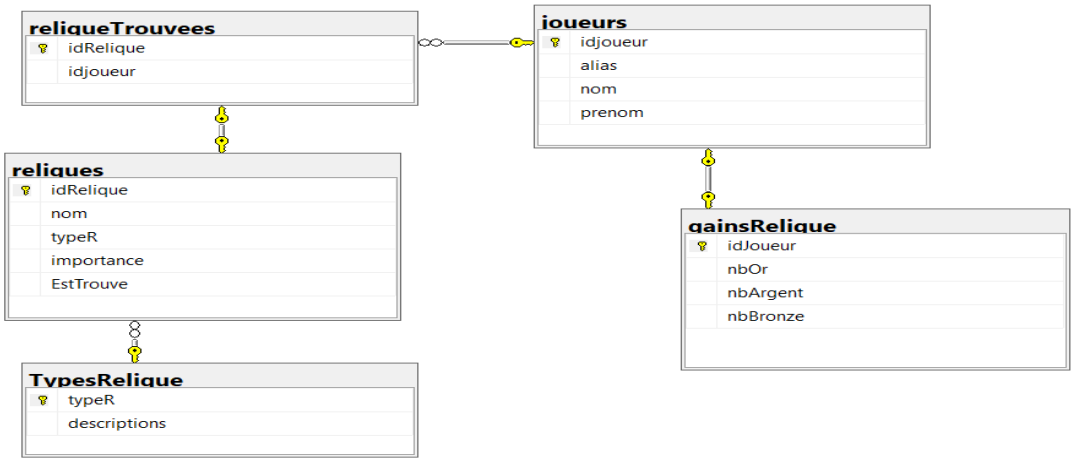
Initialement, la table Departements contient au moins les départements **RSH** pour ressources humaines et **DEV** pour Développement.

La table **emplois** contient les informations suivantes.

TypeEmploi	Description	salaireMax	salaireMin
DBA	Data Base Administration	300 000	120 000
RES	Responsable	180 000	80 000
DEV	Développeurs	150 000	40 000
COM	Commis	60 000	25 000
TEC	Techniciens	90 000	35 000

Base de données dbdev

La base de données **dbdev** contiendra les tables du jeu **TrouverRelique** donné par le diagramme ci-dessous, et le script TrouverRelique .sql .



Les joueurs sont les employés du département DEV.

Pour bien comprendre le déroulement du jeu, il faut lire la section « Développement et sécurité des données au département DEV »

Résumé de la structure de MobiGame

RSH, ressources humaines
BD: dbrsh
Responsable: **Saturne Lune**
Tables: Employes, Departements, Emplois

Données:
Employes: contient tous les employés des deux départements
Departements:
DEV
RSH
Emplois: Voir le contenu du document de la PFI

Les rôles et les privilèges:
Pour les employés du RSH sur dbrsh
Martine Labelle,
Hugo Lapointe,
Saturne Lune. (responsable)

DEV,développement
BD: dbDEV
Responsable: **Alain Patoche**
Tables: Celles du script « TrouverRelique .sql »

- gainsRelique
- reliqueTrouvees;
- joueurs;
- reliques;
- typesrelique;

Les rôles et les privilèges:
Pour les développeurs sur la BD dbdev

Guy Tares,
Bien Thierry,
Martin Lejeune,
Lyna Pilon,
Kevin Lafleur,
Alain Patoche (responsable)

A- Sécurité et intégrité des données.

Pour respecter les règles de sécurité, l'administrateur (DBA) en occurrence VOUS, décide de ne pas donner tous les droits à tous les employés. Il décide plutôt d'y aller par attribution de rôles. Pour chaque type d'emploi, un rôle est défini.

Voici un tableau qui résume un peu la structure de la PME concernant les droits et les privilèges.

Rôles	Privilèges	Membres
RespRSH	SELECT, UPDATE, INSERT sur toutes les table du Département RSH. Également le privilège EXECUTE.	Responsable des RSH
RoleTech	SELECT sur toutes les tables du département RSH UPDATE de la colonne Adresse de la table Employés, UPDATE de description de la table Emplois, INSERT et UPDATE de la table Départements	Techniciens en administrations
RoleCommis	SELECT sur toutes les table du département RSH	Les commis
db_owner	Propriétaire de la base de données du département DEV	Responsable du département Dev
db_datawriter	Peuvent écrire sur la BD du département DEV	Les développeurs
db_datareader	Peuvent lire sur la BD du département DEV	Les développeurs
db_ddladmin	Peuvent exécuter les commande DDL su la bd du département DEV	Les développeurs
En plus, Les développeurs doivent pouvoir exécuter les procédures stockées de la base de données de leur département		

Tableau 1:Tableau 1: Rôles et autorisations

Ce qu'on vous demande : (Partie 1)

1. Créer les bases de données des deux départements, RSH (dbrsh) et DEV (dbdev).
2. Utiliser la BD dbrsh pour créer les tables pour le département RSH. Puis insérer les données.
3. Utiliser la BD dbdev puis exécuter le script TrouverRelique .sql pour créer les tables du département DEV et y insérer les données.
4. Créer les logins avec les utilisateurs mappés sur les logins et la base de données correspondante. Ici les logins sont au format suivant : vos initiales, suivis de la première lettre du prénom suivie du nom : Exemples : **YSGtares** pour Guy Tares, **YSHlapointe** pour Hugo Lapointe.
    - Tous les login ont leur bases de données par défaut.
    - La stratégie des mots de passe est appliquée pour tous les logins
5. Attribuer les rôles bases de données selon le tableau 1
6. Créer les RÔLES non prédéfinis du tableau 1.
7. Ajouter les membres aux rôles en tenant compte des autorisations.
8. Faites-en sorte que les développeurs puissent exécuter les procédures stockées.
9. Écrire le trigger SYCTRLAugSalaire. (le nom du trigger doit être précédé de vos initiales)
10. Écrire le trigger SYCTRLSalaire. (le nom du trigger doit être précédé de vos initiales)

Authentification des joueurs et information de crédit (dbdev). (Partie 2)

Vos initiales doivent précédées le nom des procédures et fonctions.

Dans un premier temps, vous devez implémenter un mécanisme d'authentification des joueurs à l'aide d'un mot de passe. Le mot de passe des joueurs sera conservé dans la table joueurs et sera **chiffré à l'aide d'une fonction de hachage : HASHBYTES.**

De plus les joueurs auront la possibilité d'associer un numéro de carte de crédit à leur profil dans la table **joueurs**. Votre travail est d'ajouter la fonctionnalité pour pouvoir conserver en toute sécurité le numéro de carte de crédit des joueurs.

Le numéro de carte de crédit sera crypté à l'aide d'un algorithme de chiffrement symétrique (**ENCRYPTBYPASSPHRASE**). La clé utilisée pour crypter le numéro sera **le mot de passe** du joueur. Cette façon de procéder garantit qu'uniquement le joueur en possession du mot de passe aura accès à l'information de crédit.

Vous aurez donc à ajouter deux nouvelles colonnes à la table **joueurs** pour conserver ce mot de passe et les informations de crédit.

Pour mettre en place cette nouvelle fonctionnalité, vous devez programmer les procédures et fonctions suivantes.

1. Pour la table joueurs ajouter les colonnes **mpasse** et **noCredit** de type VARBINARY(128)
2. Programmez la procédure **AjouterJoueur** qui ajoute un joueur à la table joueurs avec toutes ses informations, incluant son mot de passe hashé.
3. Programmez la fonction scalaire **ValiderIdentité** pour valider l'identité d'un joueur basé sur son alias et son mot de passe. La fonction retourne 0 (succès) ou 1 (échec).
4. Programmez la procédure **AjouterInfoCrédit** qui permet d'ajouter un numéro de carte de crédit à un joueur donné. Le mot de passe du joueur doit être fourni.
5. Programmez la procédure **ModifierMotPasse** qui permet à un joueur de modifier son mot de passe. L'ancien mot de passe doit être fourni pour valider son identité. Ici, vous devez tenir compte du changement du mot de passe pour décrypter et encrypter la carte de crédit avec le nouveau mot de passe.
6. Programmez la fonction table **ObtenirInfoCrédit** qui retourne toute l'information d'un joueur, incluant son numéro de carte de crédit, mais pas son mot de passe.

B- Le jeu : TrouverRelique :

Les employés du département DEV, ayant développé le jeu TrouverRelique veulent effectuer les tests nécessaires en jouant au jeu**.**

Description du jeu TrouverRelique :

Les joueurs ont pour but de trouver des reliques cachées sur le site de la ville de Kent. Les **reliques sont uniques**. (deux joueurs ne peuvent pas trouver la même reliques).

Les reliques ont une propriété : **estTrouve** qui indique si la relique est trouvée ou non (1→ la relique est trouvées, 0-→ la relique n'est pas trouvées). Initialement toutes les reliques ne sont pas trouvées.

Les reliques ont une importance qui est 1, 2 ou 3.

- Lorsqu'un joueur trouve une relique d'importance 1, on lui donne 5 pièces de Bronze
- Lorsqu'un joueur trouve une relique d'importance 2, on lui donne 10 pièces d'Argent
- Lorsqu'un joueur trouve une relique d'importance 3, on lui donne 15 pièces de d'Or.

Ces données sont consignées dans la table **`gainsRelique`**

Lorsqu'un joueur trouve une relique, les opérations suivantes doivent être effectuées et la BD doit rester dans un état cohérent:

- On doit mettre à jour que la relique est trouvée
- On attribue la relique au joueur en question. Ces informations sont dans la table : **`reliqueTrouvees`**
- On donne le nombre de pièces correspondant (Or, Argent, Bronze) au joueur correspondant. La table affectée est **`gainsRelique.`**
- Si le joueur existe dans la table **`gainsRelique,`** on met à jour ses gains sinon on fait une insertion.

Exemple, voici le contenu de la table : **`reliqueTrouvees`**

idRelique	idjoueur
100	2
101	2
103	2
104	6

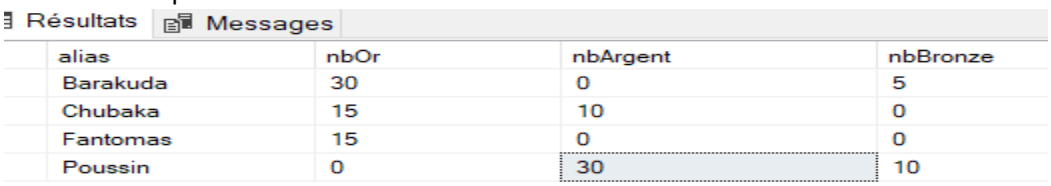
Et voici le contenu de la table : **`gainsRelique`**

idJoueur	nbOr	nbArgent	nbBronze
2	30	0	5
6	0	10	0

Ce qu'on vous demande : (partie 3) Pour les questions suivantes, vos initiales doivent précédées le nom des procédures.

1. Écrire une procédure ***objetTrouve(@alias varchar(10), @idRelique int)*** qui permet de trouver une relique. Cette procédure permet de faire les opérations décrites dans l'encadré vert de la page 6. De plus cette procédure doit vérifier que l'alias existe, et que la relique n'est pas encore trouvée.
2. Programmer **une procédure stockée** ***classementJoueurs*** affiche le classement des joueurs. On affiche l'alias, et le nombre de pièces associées à leurs trouvailles. Les joueurs ayant le nombre total de pièces d'Or en premiers, puis, Argent, enfin Bronze.

Voici un exemple d'exécution :



Résultats | Messages

	alias	nbOr	nbArgent	nbBronze
	Barakuda	30	0	5
	Chubaka	15	10	0
	Fantomas	15	0	0
	Poussin	0	30	10

3. Écrire une procédure ***updateRelique*** qui met le flag estTrouve à 0 lorsque toutes les reliques sont trouvées.
4. L'information de la table **`gainsRelique`** est confidentielle. Il est donc d'une importance capitale qu'un usager de la base de données du département DEV ait accès uniquement aux données qui le concernent. Ici on suppose qu'un joueur correspond à un usager. Vous devez mettre en place un mécanisme qui garantira:
    a. Qu'un joueur puisse consulter ses gains et uniquement les siens
    b. Qu'un joueur puisse mettre à jour (INSERT et UPDATE) ses gains et uniquement les siens

Pour la question précédente, **la seule solution acceptable** est celle qui utilise des vues avec l'option WITH CHECK OPTION. C'est une exigence du cours.

Vous pouvez donner la solution uniquement pour trois joueurs.

Ce que vous devez remettre :

Fichier SQL	Pondération
1. Un script (pfi_secure.sql) contenant les commandes sql de la partie «Sécurité et intégrité des données»	30
2. Un script (pfi_authentification.sql), spécifique à l'authentification et contenant la partie « authentification des joueurs » au département DEV	25
3. Un script (pfi_reliques.sql), spécifique au jeu TrouverRelique du département DEV. Q1/10, Q2 /5, Q3/5, Q4/10. (inclue les tests des procédures stockées)	30
Évaluation de la progression du travail en classe (semaine du 01 et 08 décembre)	10
Respect des consignes de remise. Ici, vous avez 0 5	5

Le barème ci-dessus inclue le script d'exécution de l'ensemble des procédures stockées et des triggers.

Modalité de remise : Boite de remise dans Colnet.