Travail pratique 1, Automne 2025

WoW! des items pour la quête sombre : DarQuest[1]

- **Ce travail sera réalisé individuellement et compte pour 20 % de la note finale.**
- À remettre :
  - **Le modèle de données : Jeudi le 02 octobre avant minuit**
  - **Le groupe 2 et 3 vendredi 10 octobre avant minuit**
  - **Le groupe 4 le vendredi 24 octobre 2025**

Attention !!

Pour ce travail, vous devez utiliser uniquement les concepts vus en classe ou dans le cours de KB6 (Introduction aux bases de données)

Si vous avez besoin d'utiliser autre que les concepts présentés en classe alors vous devez :

1. **Expliquer le concept avant son utilisation**
2. Donner la source de l'information
3. Demander l'autorisation à l'enseignante ou à l'enseignant

Une évaluation orale pourrait se faire pour le travail remis pour n'importe quelle question et n'importe quel étudiant.

La débrouillardise et l'autonomie sont deux qualités recherchées chez un informaticien. **Le plagiat sous toutes ses formes c'est NON et entraine les sanctions prévues par la PIEA**

Objectifs

1. Produire un modèle de données normalisé
2. Écrire des procédures stockées (Procédures, fonctions et triggers)
3. Veiller à la cohérence des données par les transactions
4. Utiliser les curseurs.
5. Développer son esprit d'analyse et son autonomie.

Note importante :

Au moment de la distribution de ce TP (semaine du 29-09), le cours sur les curseurs n'a pas été abordé. Dans ce TP seules les questions 6, 7, 8 et 9 du groupe 4 utilisent les curseurs. Par conséquent vous pouvez faire toutes les autres questions du groupe 4 en particulier les questions 1 à 5 avant la présentation du cours sur les curseurs.

[1] Source de l'image : https://wall.alphacoders.com/big.php?i=159670

Mise en contexte :

Nous sommes dans un environnement de jeu vidéo de type **World of Warcraft** (WoW), et nous souhaitons développer une application qui permet l'achat d'un certain nombre d'items concernant le jeu WoW.

Les items :

1. Les items du jeu sont les armes, les armures, les potions et les éléments.
2. Tous les items ont en commun un numéro **unique** (IDENTITY), un nom, une quantité en inventaire, un prix unitaire (nombre de pièces d'or), et un flag de disponibilité. Ce flag est tout le temps égal à 1 aussi longtemps que l'item n'est pas supprimé. Sinon le flag est égal à 0.
3. Les armes ont en plus, une efficacité, un genre (une main, deux mains) une description. Exemples : Flèches, arc, marteau, épée, bâton de pierre, sabre etc...
4. Les potions ont en plus un effet attendu et la durée pour l'effet. Exemple : Potion de vitesse, potion de protection, potion de soin etc...
5. Les armures ont en plus une taille, le poids, et la matière qui la compose : Cuire, métal…
6. Un élément a en plus le type (plante, épice , sang etc..) rareté , la dangerosité. Les éléments sont moins chers que les potions, cette contrainte sera garantie par un trigger.
7. Les éléments servent à concocter des potions. Une potion est réalisée à partir de plusieurs éléments. Un élément peut faire partie de la composition de plusieurs potions.
8. Pour faire une potion, nous avons besoin de connaitre la quantité de chaque élément qui compose la potion.

Dans l'environnement du jeu WoW, il y a plusieurs joueurs, chaque joueur peut acheter plusieurs items. Des joueurs différents peuvent acheter des items identiques.

Les joueurs

1. Un joueur a un alias unique, un nom, un prénom et un montant initial en **pièces d'or**. Les joueurs sont identifiés par un numéro séquentiel. **(IDENTITY).** Certains joueurs sont des « **alchimistes »**
2. Tous les joueurs peuvent acheter n'importe quel item sauf les Elements.
3. La quantité d'un Item achetée peut-être supérieure à 1
4. Seuls les alchimistes peuvent acheter des Elements.
5. Lorsqu'un joueur alchimiste possède les composants d'une potion avec les quantités suffisantes, alors il peut concocter la potion.

Les achats :

Les achats sont conservés dans un panier. Le panier d'achats contient la quantité achetée de chaque item pour un joueur donné. Au fur et à mesure que les items sont ajoutés dans le panier, l'inventaire de l'item est mis à jour. Les items s'accumulent dans le panier jusqu'au moment où le joueur passe à la caisse pour payer le panier.

Au moment de payer le panier , le solde en pièces d'or du joueur est déduit du montant total des achats. Après que le joueur ai payé le contenu du panier, le panier est supprimé.

Votre base de données doit-être conçue de sorte que l'on puisse **conserver l'inventaire des joueurs**. Cet inventaire contient le contenu des paniers d'achats payés.

C'est au moment de payer le panier que l'inventaire des achats est mis à jour avec le contenu du panier.

Questions :

Partie1, Conception de la base de données :

1. Donner le modèle relationnel de la base de données **DbDarquest**. La remise est pour au plus tard **jeudi 02 octobre 23h59** au format pdf ou image et rien d'autre. Cette date est pour tous les groupes

Partie 2, Exploitation de la base de données : Procédures stockées et triggers.

Groupe 1 : Initialisation de la base de données :

1. Créer la base de données : **DbDarquest**;
2. Créer toutes les tables de la base de données en exécutant le script : **createTables.sql.** Vous êtes obligés d'utiliser le script pour créer les tables afin de faciliter le débogage et la correction
3. Exécuter le script **insertJoueur.sql** pour insérer les joueurs dans votre base de données.

Groupe 2, Les triggers pour l'intégrité de la base de données

1. Écrire le trigger ***CTRLPrixItem qui*** garantit que le prix d'une potion est toujours plus élevé que le prix d'un élément.

2. Le trigger ***CTRLInsertArme*** qui permet de garantir qu'un numéro inséré dans la table Armes ne sera pas utilisé (inséré) dans la table Armure ni la table potions, ni la table Element

3. Un trigger ***CTRLInsertPotion*** qui permet de garantir qu'un numéro inséré dans la table Potions ne sera pas utilisé (inséré) dans la table Armes ni la tables Armures, ni la table Element.

4. Tester TOUS VOS triggers par les requêtes DML appropriées.

Groupe 3, Gestion des Items :

Écrire les procédures stockées suivantes pour la gestion des Items.

Toutes les procédures et fonctions doivent-être exécutées et testées. Pour tester vos procédures il faut vous référer aux 5 tableaux suivants.

1. Une procédure ***ajouterPotion***, cette procédure permet d'ajouter une Potion dans la base de données. Insérer toutes les informations d'une potion.

2. Une procédure ***ajouterElement*** cette procédure permet d'ajouter un élément dans la base de données. Insérer toutes les informations de l'élément

3. Une procédure ***ajouterArmure*** cette procédure permet d'ajouter un sort dans la base de données. Insérer toutes les données d'une armure.

4. Une procédure ***ajouterArme***, cette procédure permet d'ajouter une arme dans la base de données. Insérer toutes les informations d'une arme.

5. Donner le script d'exécution des procédures d'insertion pour que le contenu de vos tables soit au minimum le suivant : (voir exemple d'exécution plus bas). Commencez par

donner les cas qui marchent pour toutes les procédures afin de ne pas briser la séquence de idItem.

Table Items :

idItem	nom	qtStock	prix	typeItem	flagDispo
1	Potion du hibou	100	45	P	1
2	Potion violet	70	100	P	1
3	Potion rouge	80	55	P	1
4	Sang de loup	10	10	E	1
5	Sang cobra	40	5	E	1
6	Pétale noire	25	10	E	1
7	Potion Mortel	7	500	P	1
8	Poivre Jaune	10	5	E	1
9	Veste Elfique	50	300	R	1
10	Armure de Chevalier	20	60	R	1
11	Armure de Dragonnier	5	100	R	1
12	Épée longue	80	50	A	1
13	Hallebarde chargée	10	250	A	1

Table Potions

idItem	effet	durée
1	Vision nocturne	50
2	Rend invisible	5
3	Courir vite vite	7
7	Rend invincible	8

Table Element :

idItem	typeElement	dangerosite	rarete
4	sang	très dangeureux	3
5	sang	mortel	2
6	Plante	moyen	1
8	Épice	faible	2

Table Armures :

idItem	matiere	poids	taille
9	Mithril	2	4
10	Fer	50	8
11	Peau de dragon	30	12

Table Armes :

idItem	efficacité	genre	descriptionArme
12	15	Une main	Épée très tranchante
13	15	Deux mains	Pointue et tranchant

Exemple d'exécution pour *ajouterArme*,

```
execute ajouterArme
@nom = 'Hallebarde chargée',
@qtstock = 10,
@prix = 250,
@efficacite = 15,
@genre = 'Deux mains',
@description = 'Pointue et tranchant';
```

6- Une procédure ***afficherItems*** (@typeItem) cette procédure permet d'afficher les items selon le **type de l'item**. Afficher toutes les informations des items. Exemple pour une arme, nous allons afficher les informations (nom, quantité en stock, le prix unitaire, le nombre de points de dommage et le genre et la description). Pour cette question, il est conseillé de créer des **vues** que vous allez utiliser dans votre procédure. Le code sera plus propre. **Les vues sont créées à l'extérieur des procédures**

7- Une fonction ***prixMoyenItems*** qui retourne sous forme de table le prix moyen de chaque type d'item. Afficher le nom du type (voir image suivante)

typeItem	moyennePrix
Potion	175
Armure	153
Arme	150
Element	7

8- Une procédure ***supprimerItem*** *(*@idItem*),* qui permet de supprimer un item étant donné un numéro d'item. La suppression d'un item implique la mise à jour du flag de disponibilité à 0. Un item supprimé qui est contenu dans des paniers doit aussi être supprimé. **Exécutez cette procédure pour l'item 12, et un item que vous aurez inséré.**

Groupe 4, Gestion du panier d'achat

Écrire les procédures stockées suivantes pour la gestion des Achats.

Toutes les procédures et fonctions doivent-être exécutées et testées. Pour cette partie, les données de test vous seront fournies.

1. Une fonction **montantPanier** (@alias) qui retourne le montant total du panier d'achats d'un joueur étant donné son **alias**. S'il n'y a pas de panier pour le joueur, la fonction retourne 0;

   Pour tester faire les insertions suivantes dans la table Paniers.

```
insert into Paniers (idJoueur,idItem,qtItem)  values(3,1,2);
insert into Paniers (idJoueur,idItem,qtItem)  values(3,4,4);
insert into Paniers (idJoueur,idItem,qtItem)  values(6,6,1);
```

Tester vos fonctions, puis vider le panier en exécutant : (il faut vider le panier c'est une obligation)

```
delete from Paniers;
```

2. Une procédure ***ajouterItemPanier*** (@alias, @idItem, @qtItem) qui permet d'ajouter un item au panier avec une quantité donnée pour un joueur étant donné son **alias**.

   Les paramètres de la procédure sont donc : L'alias du joueur, le numéro de l'item et la quantité à acheter.

   - Si la quantité en inventaire de l'item est insuffisante, l'ajout est annulé.
   - Si le solde du joueur est insuffisant pour couvrir le montant total du panier plus l'ajout de l'item, l'ajout est annulé.
   - La procédure ajoute l'item au panier et prend soin de mettre à jour l'inventaire de l'item.
   - Si l'item existe déjà dans le panier du joueur, la quantité est mise à jour uniquement.

3. Une procédure ***modifierItemPanier*** (@idItem, @nouvelleQtItem, @alias) qui permet de modifier la quantité achetée d'un item dans le panier d'achats d'un joueur étant donné son **alias**.
   - Si la quantité en inventaire est insuffisante pour couvrir la nouvelle quantité, la modification est annulée;
   - La procédure met à jour la quantité et prend soin de mettre à jour l'inventaire de l'item.
   - Si le solde du joueur est insuffisant pour couvrir le montant total du panier plus l'ajout de l'item, l'ajout est annulé

4. Une procédure ***supprimerItemPanier*** (@idItem, @alias) qui permet de supprimer un item du panier d'achats d'un joueur étant donné son **alias**.
   - La procédure supprime l'item du panier et prend soin de mettre à jour l'inventaire de l'item.

5. Une fonction **table *afficherPanier*** (@alias) qui retourne le contenu du panier d'un joueur. La fonction retourne sous forme d'une table le nom de l'item et son type (**'Arme', 'Armuret', 'Potion', 'Element**), la quantité et le prix total par Item.

6. Une procédure ***payerPanier*** (@alias) qui permet de payer le panier d'achats d'un joueur étant donné son **alias**.

    - Mettre à jour le solde du joueur avec le montant du panier.
    - Mettre à jour l'inventaire des achats du joueur. (Précision : Si l'item est déjà dans l'inventaire du joueur, il faut mettre à jour la quantité, sinon il faut insérer l'item dans l'inventaire du joueur)
    - Supprimer le panier d'achats du joueur;

7. Une procédure ***supprimerPanier*** (@alias) qui permet de supprimer le panier d'achats (abandonner un achat) d'un joueur étant donné son **alias**;

    - Remettre en inventaire la quantité achetée de chaque item contenue dans le panier du joueur;
    - Une fois l'inventaire mis à jour, on supprime le panier d'achats du joueur;

8. Une fonction ***potionFaisable*** (@alias,@idPotion) qui retourne 1 lorsque la potion est réalisable pour le joueur 0 sinon. Une potion est réalisable si le joueur possède la liste des éléments avec les quantités suffisantes pour faire la potion.

9. Écrire une procédure ***realiserPotion*** (@alias,@idPotion) qui permet à un joueur de réaliser une potion. Cette procédure doit permettre :

    - On ajoute la potion à l'inventaire du joueur. Si la potion est déjà dans l'inventaire, il faut augmenter la quantité de 1, sinon il faudra l'insérer
    - On met à jour la quantité des éléments utilisés pour la potion dans l'inventaire du joueur.
    - Si la quantité est nulle pour un élément après avoir concocter une potion alors cet élément est supprimé de l'inventaire du joueur.

10. Une fonction **table *afficherInventaire*** (@alias) qui retourne la liste de tous les achats payés (inventaire) étant donné **l'alias** d'un joueur. La table retournée contient l'alias du joueur, le nom de l'item, son type ('Arme', 'Armure', 'Potion', 'Element') et la quantité.

11. Le trigger ***cascadeDeleteItem*** qui fait la suppression en cascade d'un item qui n'est pas dans l'inventaire du joueur. Les items supprimés qui sont contenus dans des paniers doivent aussi être supprimés. Si l'item est dans l'inventaire du joueur alors le flag doit être à zéro dans la table Items.

12. Écrire le trigger ***afterInsertPanier*** qui garantit que les joueurs qui ne sont pas alchimiste ne puissent ajouter un item de type Element au panier.

À chaque remise, remettre dans un dossier Zippé portant votre nom :

1. Un script SQL contenant les procédures, les fonctions stockées et les triggers;
2. Un script SQL contenant les requêtes d'exécution des procédures, des fonctions et de la vérification du déclenchement des triggers;

Évaluation

Élément évalués	Pondération
Modèle de données	10
Groupe 2 : Q1, Q2 →3points chacun. Q3 →1pts.(7 points incluant le script d'exécution)	7
Groupe 3 : Q1, Q2, Q3,Q4 → deux points chacune. (8) Q6,Q7,Q8 →3 points chacune (9) Q5, 2 points	19
Groupe 4 : Q1,Q,4,Q5 et 10 →4 points chacune (16 points) Q2, Q3, Q6, Q7 ,Q8 et Q9 →5 points chacune (30) Q11 et Q12 →3 points chacune (6 points)	52
Script d'exécution groupe 4	12
Total	100