# Travail pratique no_1

# *JeuRelic*[1]

*Automne 2021*

- Ce travail sera réalisé individuellement.
- À remettre le 22 octobre 2021
- Le travail compte pour 20 % de la note finale.

## Objectifs

Ce travail vise l'atteinte des objectifs suivants :

1. Produire un modèle de données normalisé
2. Écrire des procédures stockées
3. Écrire des triggers.

[1] Source de l'image : https://photoshop-kopona.com/fr/56755-45-png-armes-et-armures-medievales-sur-fond-transparent.html

## Mise en contexte :

Nous sommes dans un environnement du jeu vidéo de type médiéval, et nous souhaitons développer une application ***jeuRelic*** où chaque joueur pourrait acheter des items appelés également Reliques afin équiper son personnage.

- Les items peuvent être des armures, des bijoux, des parchemins
- Tous les Items sont identifiés par un **numéro unique (IDENTITY)**, ont un nom, une brève description, un prix unitaire, un flag de disponibilité , une quantité en stock et une quantité stock limite. Le flag est tout le temps à 1 aussi longtemps que l'item n'est pas supprimé. Sinon le flag est égal à 0.
- Les armures ont en plus une taille, le poids et la matière qui la compose : Cuire, métal etc…
- Les bijoux ont en plus la matière dont ils sont faits (le métal qui compose le bijou : ivoire, or, argent, bronze etc), le type (bague, bracelet anneau, chaîne etc) et un pouvoir magique.
- Les parchemins ont en plus un sort, la durée du sort et l'antidote pour annuler l'effet du sort

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

Un joueur a un numéro **unique**, un alias **unique**, un nom, un prénom , un montant initial en écus et un niveau. Les niveaux sont : 1,2 , 3 , 4 ou 5. Les joueurs de niveau 1 ne peuvent pas acheter des parchemins .

Tous les joueurs peuvent acheter n'importe quel item, à condition qu'il soit disponible en inventaire. La quantité d'un Item achetée peut-être supérieure à 1.

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 écu du joueur est déduit du montant total des achats. Après que le joueur ait payé le contenu du panier, le panier est supprimé.

Votre base de données doit-être conçue de sorte que l'on puisse conserver l'historique des achats des joueurs. Cet historique contient le contenu des paniers **d'achats payés.** On a pour chaque historique d'achats un numéro unique, la date à laquelle les achats ont été payés ainsi que la liste des Item achetés .

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

# Questions :

## Conception de la base de données :

1. Donner le modèle relationnel de votre BD en 3FN. La remise du modèle a lieu le 05 octobre 2021 avant 08 heures du matin. Seuls les formats PDF ou image sont acceptés.
2. Créer la base de données : **BD*jeuRelic***
3. Créer toutes les tables de la base de données. Voir le corrigé du modèle de données sur le site Web. **Le nom des tables et des colonnes doit être respecté**.
4. Peupler votre base de données avec des données initiales. Vous devez avoir au moins :
    a. 5 joueurs (Obligatoire)
    b. 5 armures, 5 bijoux, 5 parchemins (Obligatoire) . Ici, vous pouvez utiliser vos procédures stockées
    c. 5 enregistrements dans historique d'achats (si vous n'avez pas pu faire la question 6 du groupe 2), ce qui vous permettras de faire la question 8 du groupe 2 si vous n'avez pas réussi à faire la 6

## Exploitation de la base de données : Procédures stockées et triggers.

### Groupe 1 :

Écrire les procédures stockées suivantes pour la gestion des Items.
(Toutes les procédures et fonctions doivent-être exécutées et testées. Vos données doivent être cohérentes en tout temps→**Transactions**)

1. Une procédure ***ajouterArmure***, cette procédure permet d'ajouter une armure dans la base de données. Insérer toutes les informations d'une armure
2. Une procédure ***ajouterBijoux***, cette procédure permet d'ajouter un bijou dans la base de données. Insérer toutes les informations d'un bijou.
3. Une procédure ***ajouterParchemin***, cette procédure permet d'ajouter un parchemin dans la base de données. Insérer toutes les informations du parchemin
4. Une procédure ***afficherItem*** cette procédure permet d'afficher les Items selon le **type de l'Item**. Afficher toutes les informations des Item. Exemple pour une armure, nous allons afficher les informations (nom, quantité en stock, le prix unitaire, le poids, la taille, matériel). Pour cette question, vous devez créer des **vues** que vous allez utiliser dans votre procédure. Le code sera plus propre.
5. Une fonction ***prixMoyenItem*** qui retourne sous forme **de table** le prix moyen de chaque type de Item.
6. Une procédure ***supprimerItem***, qui permet de supprimer un Item étant donné un numéro de l'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.

**Groupe 2**

Écrire les procédures stockées suivantes pour la gestion des Achats.
(Toutes les procédures et fonctions doivent-être exécutées et testées. Les transactions doivent être cohérentes)

1. Une fonction **montantPanier** 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;

2. Une procédure ***ajouterItemPanier*** qui permet d'ajouter un Item au panier avec une quantité donnée pour un joueur étant donné son **alias**.

    - 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.
    - Si le joueur est de niveau 1 et que l'item est un parchemin, l'ajout est annulé.

3. Une procédure ***modifierItemPanier*** 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.

4. Une procédure ***supprimerItemPanier*** 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*** 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 ('Armure', 'Bijou', 'Parchemin'), la quantité et le prix total.

6. Une procédure **p*ayerPanier*** 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'historique des achats du joueur.
    - Supprimer le panier d'achats du joueur;

7. Une procédure ***SupprimerPanier*** qui permet de supprimer le panier d'achats d'un joueur étant donné son **alias**;

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

8. Une fonction **table *afficherAchats*** qui retourne la liste de tous **les achats payés** étant donné **l'alias** d'un joueur. La table retournée contient l'alias du joueur, le nom de l'Item, son type ('Armure', 'Bijou', 'Parchemin') et la quantité.

**Groupe 3,** écrire les triggers suivants :

1. Le trigger ***beforeInsertParchemin*** qui permet de garantir qu'un numéro inséré dans la table Parchemins ne sera pas utilisé (inséré) dans la table Armures ni dans la table Bijoux
2. Un trigger ***beforeInsertArmure*** qui permet de garantir qu'un numéro inséré dans la table Armures ne sera pas utilisé (inséré) dans la table Bijoux ni la table Parchemins .
3. Un trigger ***updatestockItem*** qui permet d'augmenter la quantité en stock d'un Item lorsque la quantité limite est atteinte. Ce qui veut dire que la quantité en stock ne doit pas être plus petite que la quantité limite. On doit augmenter la quantité du triple de la quantité limite.
4. Écrire le trigger ***afterInsertPanier*** qui garantit que les joueurs de niveau 1 ne peuvent pas ajouter dans le panier des Items qui sont des parchemins.
5. Le trigger ***cascadeDeleteItem*** qui fait la suppression en cascade d'un Item qui n'est pas dans l'historique des achats. Les Items supprimés qui sont contenus dans des paniers doivent aussi être supprimés.

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

## Ce qu'il faut remettre dans un dossier Zippé portant votre nom :

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

Note : Le modèle de données est déjà remis.

## Évaluation

| Élément évalués | Pondération |
|---|---|
| Groupe 1 (3*6) | 18 |
| Groupe 2, toutes les questions sur 4 | 32 |
| Groupe 3, chaque trigger sur 3 points | 15 |
| Modèle de données | 10 |
| Script d'Exécution | 10 |
| Création des tables avec les contraintes | 10 |
| La base de données est bien peuplée | 5 |
| Total | 100 |