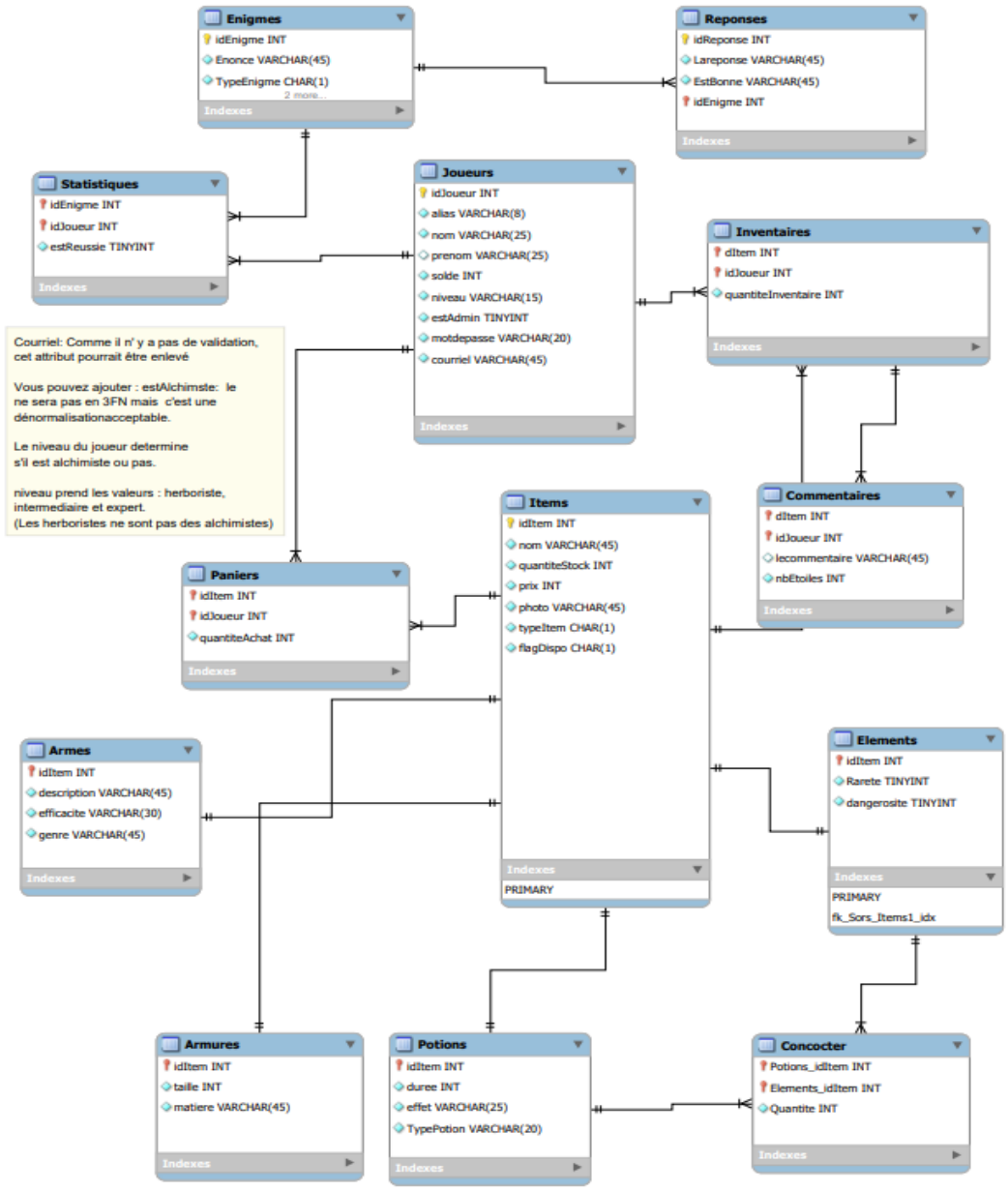
Modèle suggéré pour la base de de données Chevaleresk .

Le modèle :

Le modèle suivant est une suggestion pour votre BD Chevaleresk. Vous n'êtes pas obligé de prendre le modèle suivant si votre base de données **est normalisée et validée.**

Le **modèle ne montre pas** les potions concoctées par le joueur pour monter de niveau. Il faut le rajouter



Quelques détails :

1. Aucune contrainte CKECK n'est représentée dans le modèle. Il faut les ajouter manuellement : ALTER TABLE
    a. estAdmin prend la valeur 0 ou 1
    b. niveau prend les valeurs : **herboriste**, debutant, intermediaire et expert. (Les herboristes ne sont pas des alchimistes). Vous pouvez utiliser une autre valeur pour herboriste.
    c. le montant initial (DEFAULT) il est égal à 1000.
    d. le typeItem : est CHAR (1). R pour Armures, A pour Armes, P pour Potions et E pour les élément.
    e. Pour la table Enigmes, le type Enigme est P pour Potion, E pour élément et Z pour autre. estPigee prend la valeur o (pour oui) ou n (pour non)
    f. difficulte prend les valeurs : d pour difficile, m pour moyen et f pour facile
    g. Dans Potions, typePotion prend les valeurs : A pour attaque, D pour défense.
    h. Dans Statistiques, estReussi prend la valeur 0 ou 1.
    i. Le prix d'une potion est toujours plus élevé que leprix d'un élément sera granti par les deux triggers plus bas. Lisez les carrées bleus et oranges pour comprendre.
    j. flagDispo est DEFAULT 1, a comme valeur 1 si l'item est disponible 0 s'il n'est pas disponible (on ne le vend plus). Ce qui n'a rien à voir avec la quantité. Un item peut avoir sa quantié 0 mais il reste disponible. Vous pouvez utiliser le flagDispo si vous voulez supprimer un item alors que celui-ci est dans l'inventaire du joueur.
    k. Si vous ajoutez estAlchimiste alors cet attribut prend la valeur 0 ou 1. La présence de cet attribut dénormalise le modèle mais c'est accepatble.
2. Pour autres les contraintes :
    a. La clé primaire est clairement visible en jaune
    b. La clé primaire qui est en même temps clé étrangère est une clé rouge
    c. Le losange bleu plein : NOT NULL (obligatoire)
    d. Le losange vide : optionnel.
3. Dans le modèle ci-dessus, nous ne conservons pas l'historique des achats. Ce n'est pas demandé
4. Pour la table Paniers, il faudra la vider après chaque achat. Après la confirmation de l'achat d'un joueur, vous devez vider la table. Nous n'avons pas besoin de garder les données du panier. La table panier, est comme une table temporaire. D'ailleurs il est possible de gérer les achats sans avoir la table : **Lepanier**.

Les tables, ce que le modèle ne montre pas :

Tables	Les attributs PK	Les attribut FK
Items	idItem	Aucune
Armes	idItem	idItem (vient de la table Items)
Armures	idItem	idItem (vient de la table Items)
Potions	idItem	idItem (vient de la table Items)
Elements	idItem	idItem (vient de la table Items)

Joueurs	idJoueur	aucune
Inventaires	idJoueur et idItem	idJoueur et idItem
Commentaires	idJoueur et idItem	idJoueur vient de Inventaires idItem vient de la table Inventaires
Paniers	idJoueur et idItem	idJoueur et idItem

Pour la table Commentaires le lien est avec Inventaire de cette façon on garantit qu'un joueur ne peut pas commenter un Items qu'il ne possède pas dans son inventaire.

On voit bien que la table Items est la table qui contient les informations communes à un Item. Les détails d'un Item sont stockés dans les tables qui correspondent au typeItem. Exemple les détails d'une Arme sont dans la table Armes.

Pour ajouter un Item, il faut insérer toutes les informations d'un Items, **dans ce cas une procédure stockées est recommandée :** (Exemple)

Quelques procédures :

```
use dbsaliha;
drop procedure if exists ajouterArme;
delimiter |
create procedure ajouterArme(
    in pNom varchar(50),
    in pQuantite int,
    in pPrix int,
    in pPhoto varchar(100),
    in pDescription varchar(50),
    in pEfficacite varchar(30),
    in pGenreArme varchar(45))

begin
    declare pTypeItem char(1) default 'A';
       declare       pidItem int;
       start transaction;
              insert into Items (nom, quantiteStock, prix, photo,typeItem)
              values ( pNom, pQuantite, pPrix, pPhoto, ptypeItem);
               select LAST_INSERT_ID() into pidItem;
              insert into Armes (idItem, description,efficacite, genre)
               values (pidItem, pdescription,pEfficacite,  pGenreArme);
       commit;

end |
```

```
-- Appel de la procédure
```

```
call ajouterArme('la nouvelle hache',21,60,'hache.jpg','hache hache ','super
efficace','deuxmain');
```

```
delimiter |
create procedure ajouterPotion(
 in pNom varchar(45),
 in pQuantite int,
 in pPrix int,
 in pPhoto varchar(100),
 in pEffet varchar(45),
 in pDuree int,
 in ptypePotion char(1))

begin
declare pTypeItem char(1) default 'P';
declare pidItem int;
start transaction;
       insert into Items (nom, quantiteStock, prix, photo, typeItem)
       values ( pNom, pQuantite, pPrix, pPhoto, ptypeItem);
       select LAST_INSERT_ID() into pidItem;
       insert into Potions (idItem, effet,duree, TypePotion)
       values (pidItem, pEffet, pDuree, ptypePotion);
commit;

end |
```

```
delimiter |
create procedure ajouterElement(
 in pNom varchar(45),
 in pQuantite int,
 in pPrix int,
 in pPhoto varchar(100),
 in prarete tinyint,
 in pdangerosite tinyint )

begin
declare pTypeItem char(1) default 'E';
declare pidItem int;
start transaction;
       insert into Items (nom, quantiteStock, prix, photo,typeItem)
       values ( pNom, pQuantite, pPrix, pPhoto, ptypeItem);
       select LAST_INSERT_ID() into pidItem;
       insert into Elements (idItem, rarete, dangerosite)
       values (pidItem, prarete, pdangerosite);
commit;

end |
```

Les triggers :

Une fois les procédures `ajouterElement et ajouterPotion` sont écrites il faudra écrire les triggers qui garantissent que le prix d'une potion est toujours plus élevé que le prix d'un élément.

Deux solutions s'offrent à nous pour l'écriture de ces deux triggers.

1. En fixant le prix minimum d'une potion et le prix maximum d'un élément. Exemple le prix minimum pour une potion est 100 et le prix maximum pour un élément est 99. C'est la solution la plus simple.

Ici le prix minimum d'une potion est 100

```
drop trigger checkprixPotion;

DELIMITER |
CREATE TRIGGER checkprixPotion
before insert  ON Items
for each row

begin
declare
minPrix int;
set minPrix=100;
if(new.TypeItem='P') Then
       if(new.prix < minPrix) then
       SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Le prix est tros bas';
       end if;
end if;
END |;
```

Ici le prix max d'un élément et 99 (ne peut pas être 100)

```
drop trigger checkprixElement
DELIMITER |;
CREATE TRIGGER checkprixElement
before insert  ON Items
for each row

begin
declare
maxPrix int;
set maxPrix=100;
if(new.TypeItem='E') Then
       if(new.prix >= maxPrix) then
       SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Le prix est tros élevé';
       end if;
end if;
END |;
```

Pour verifier:

```
CALL ajouterPotion('Potion1', 50, 100,'swiftne.jpg','vitesse accrue',5,'A');
CALL ajouterElement('Safran',10,99, 'saf.jpg',1,3);
```

Attention! Les triggers MYSQL ne prennent pas plusieurs opérations DML. Si vous voulez contrôler les modifications des prix→UPDATe alors il faut créer deux autres TRIGGERS BEFORE UPDATE.

Si votre MySQL ne permet pas de créer les deux triggers séparément alors vous pouvez créer un seul trigger. Voir le carrée vert

```
drop trigger checkprixItem;
DELIMITER |
CREATE TRIGGER checkprixItem
before insert  ON Items
for each row

begin
declare  minPrix int;
declare maxPrix int;
set maxPrix=100;
set minPrix=100;
-- on garantit le prix potion>=100

  if(new.TypeItem='P') Then
       if(new.prix < minPrix) then
       SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Le prix est trop bas';
       end if;
  end if;

-- on garantit que le prix element <100

  if(new.TypeItem='E') Then
       if(new.prix >= maxPrix) then
       SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Le prix est trop élevé';
       end if;
  end if;
END |;
```

2. On ne fixe pas le prix maximum d'un élément et le prix minimum d'une potion. On ira les chercher à chaque insertion avec les fonctions min/max. Un peu long mais c'est une bonne solution aussi.

```
drop trigger CTRLInsertionElement;

DELIMITER |
CREATE TRIGGER CTRLInsertionElement
before insert  ON Items
for each row

begin
declare
minPrix int;
if(new.typeItem='E') Then

       select  min(prix)  into minPrix from Items where typeItem ='P' ;
       if(new.Prix >=minPrix) then
       SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Le prix dépasse celui des potions';
       end if;

end if;
END |;

-- le trigger pour les Potions

drop trigger CTRLInsertPotion;

DELIMITER |;
CREATE TRIGGER CTRLInsertPotion
before insert  ON Items
for each row

begin
declare
maxPrix int;
if(new.TypeItem='P') Then

       select  max(prix)  into maxPrix from Items where typeItem ='E' ;
       if(new.prix <=maxPrix) then
       SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Le prix est plus bas les éléments';
       end if;

end if;
END |
```

Pour tester il suffit d'appeler les procédures

Attention! Les triggers MYSQL ne prennent pas plusieurs opérations DML. Si vous voulez contrôler les modifications des prix→UPDATe alors il faut créer deux autres TRIGGERS BEFORE UPDATE.

Si votre MySQL ne permet pas de créer les deux triggers séparément alors vous pouvez créer un seul trigger. Voir le carrée vert

```
DELIMITER |
CREATE TRIGGER CTRLInsertItems
before insert  ON Items
for each row

begin
declare
maxPrix int;
declare minPrix int;
-- on garantit que le prix de la potion doit être plus grand que celui de --l'élément
  if(new.TypeItem='P') Then

       select  max(prix)  into maxPrix from Items where typeItem ='E' ;
       if(new.prix <=maxPrix) then
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Le prix est plus bas que éléments';
       end if;

  end if;

-- on garantit que le prix de l'élément doit être plus petit que celui de la potion
  if(new.typeItem='E') Then

       select  min(prix)  into minPrix from Items where typeItem ='P' ;
       if(new.Prix >=minPrix) then
       SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Le prix dépasse celui des potions';
       end if;

  end if;
END |;
```

Stories techniques pour la base de données

- Le prix d'un item est de l'ordre de 1 à 200 Ecus . Vous n'avez pas besoin d'une contrainte CHECK. C'est à titre indicatif seulement.
- Le montant initial d'un joueur est : 1000 Ecus
- Le prix d'une potion est plus élevé que le prix d'un élément : Voir les triggers.
- Pour faire un DELETE ou un UPDATE sans que le WHERE porte sur la PRIMARY KEY utilisez : SET SQL_SAFE_UPDATES = 0;

Stories techniques	Détails
Créer et peupler la base de données	Vous devez vérifier que votre modèle de données répond, puis faire les opérations suivantes dans l'ordre. 1. Créer les tables de la base de données à partir de votre modèle 2. Ajouter toutes les contraintes check **3.** Créer les deux triggers pour contrôler les prix des potions et des éléments. Soit le carré bleu soit le carré orange **PAS les DEUX** 4. Créer les procédures stockées pour insérer des items. Ici les procédures stockées sont obligatoires pour garantir la cohérence des données.

	5. Appel des procédures pour faire les insertions. Au moins 5 items de chaque type. 6. Vérifier que vos insertions sont OK Pour le point 1, vous devez garder une copie de votre modèle au cas où vous aurez à recréer la BD. Pour les points 2 à 6 vous devez garder un fichier SQL.