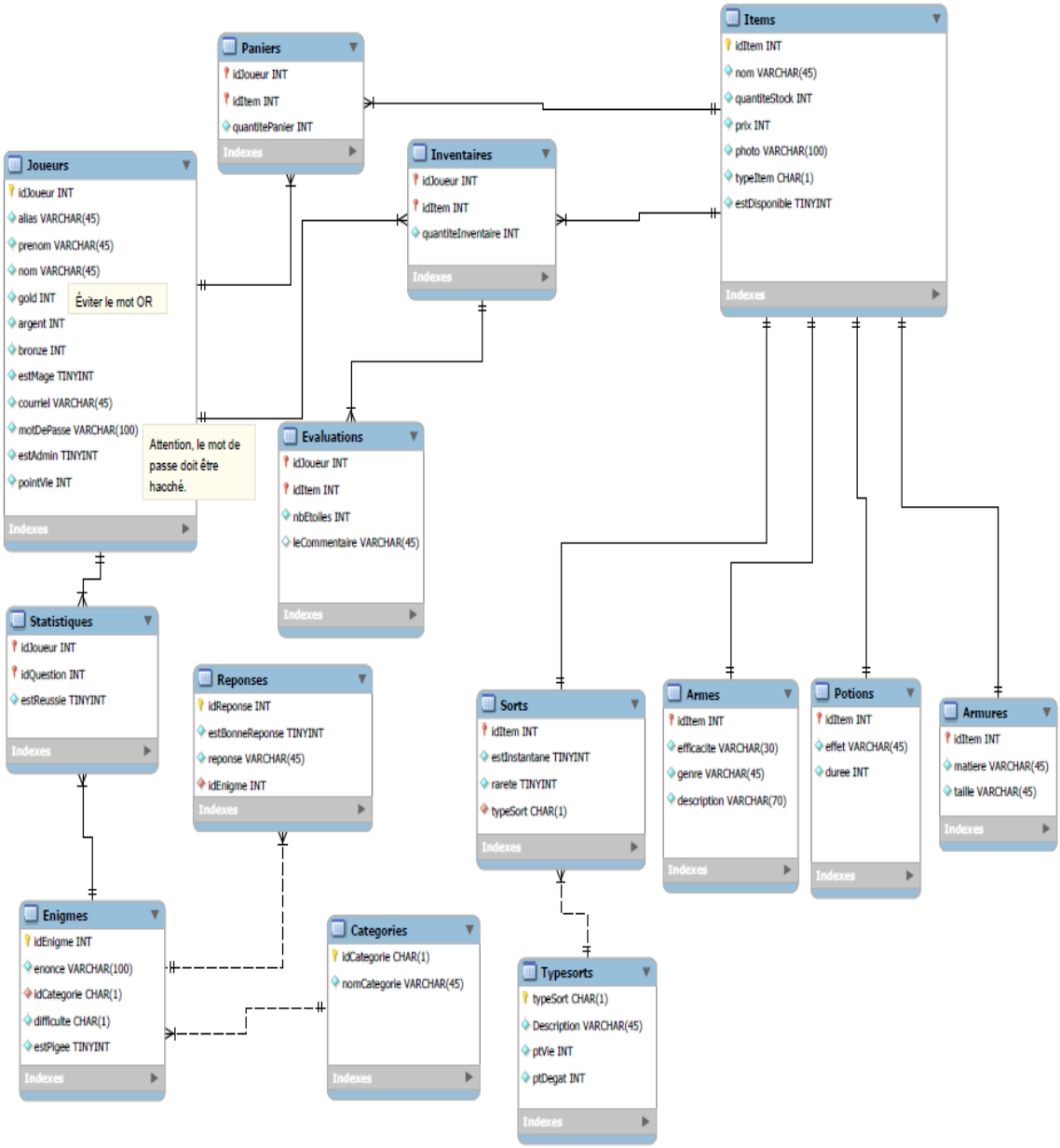
Modèle suggéré pour la base de de données **Darquest**

Le modèle :

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



Quelques détails concernant les contraintes:

1. Aucune contrainte CKECK n'est représenté dans le modèle. Il faut les ajouter manuellement : ALTER TABLE
    a. estAdmin prend la valeur 0 ou 1
    b. estMage prend la valeur 0 ou 1
    c. le montant initial (DEFAULT) des chacune des pièces est connu
        i. 1000 pièces d'or
        ii. 1000 pièces d'Argent
        iii. 1000 pièces de bronze
    d. Le nombre de points de vie initial d'un joueur est 50. Un maximum pourrait être fixé.--> Ceci est un ajout de la part du client
    e. À l'insertion le mot de passe doit être haché
    f. le typeItem : est CHAR (1).
        i. R→ pour Armures
        ii. A→ pour Armes
        iii. →P pour Potions
        iv. →S pour Sorts.
    g. Les sorts sont classés par type et peuvent donner des points de vie (ceci est un ajout du client).
    h. Les sorts ont une rareté : 1→ très rare, 2→moyenement rare, 3 →fréquent (ajout du client)
    i. Pour la table Items, vous pouvez ajouter un attribut : estDisponible. estDisponible 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 estDisponible si vous voulez supprimer un item alors que celui-ci est dans l'inventaire du joueur du joueur.
    j. Pour la table Enigmes, estPigee prend la valeur 0 ou 1 (1 veut dire que l,énigme est déjà pigée)
    k. difficulte prend les valeurs : d pour difficile, m pour moyen et f pour facile
    l. Dans Statistiques, estReussi prend la valeur 0 ou 1.
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**.
5. Pour l'instant, le modèle ne représente pas les dons de pièces de la part de l'admin

Les tables, ce que le modèle ne montre pas :

Dans la conception, on considère la règle de gestion suivantes :

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)
Sors	idItem	idItem (vient de la table Items)
Joueurs	idJoueur	aucune
Inventaires	idJoueur idItem	idJoueur idItem
Commentaires	idJoueur idItem	idJoueur vient de Inventaires idItem vient de Inventaire
Paniers	idJoueur 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)

Exemple de code SQL pour ajouter des items

```
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(500),
    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 |
```

Remarquez :

- Le **in** devant la déclaration des paramètres. Le in n'est pas par défaut.
- Le SELECT ---- INTO

Appel de la procédure :

```
call ajouterArme('lahache2',21,60,'hache.jpg','hache','très fficace','deuxmain');
```

Pour ajouter un Sort, il faut d'abord faire des insertions dans la table Typesorts : Voici un exemple d'insertion dans cette table :

```
insert into Typesorts values ('A', 'Attaque',0, 20);
insert into Typesorts values ('D', 'Defense',20, 0);
insert into Typesorts  values ('Y', 'Attaque Plus', -10, 50);
insert into Typesorts  values ('Z', 'Défense Plus', 60, 10);
```

```
delimiter |
CREATE  PROCEDURE ajouterSort(
 in pNom varchar(45),
 in pQuantite int,
 in pPrix int,
 in pPhoto varchar(100),
 in pInstantane tinyint,
 in prarete tinyint,
 in ptype char(1))
begin
declare pTypeItem char(1) default 'S';
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 Sorts (idItem, estInstantane, rarete,TypeSort)
       values (pidItem, pInstantane,prarete, ptype);
commit;
end |
```

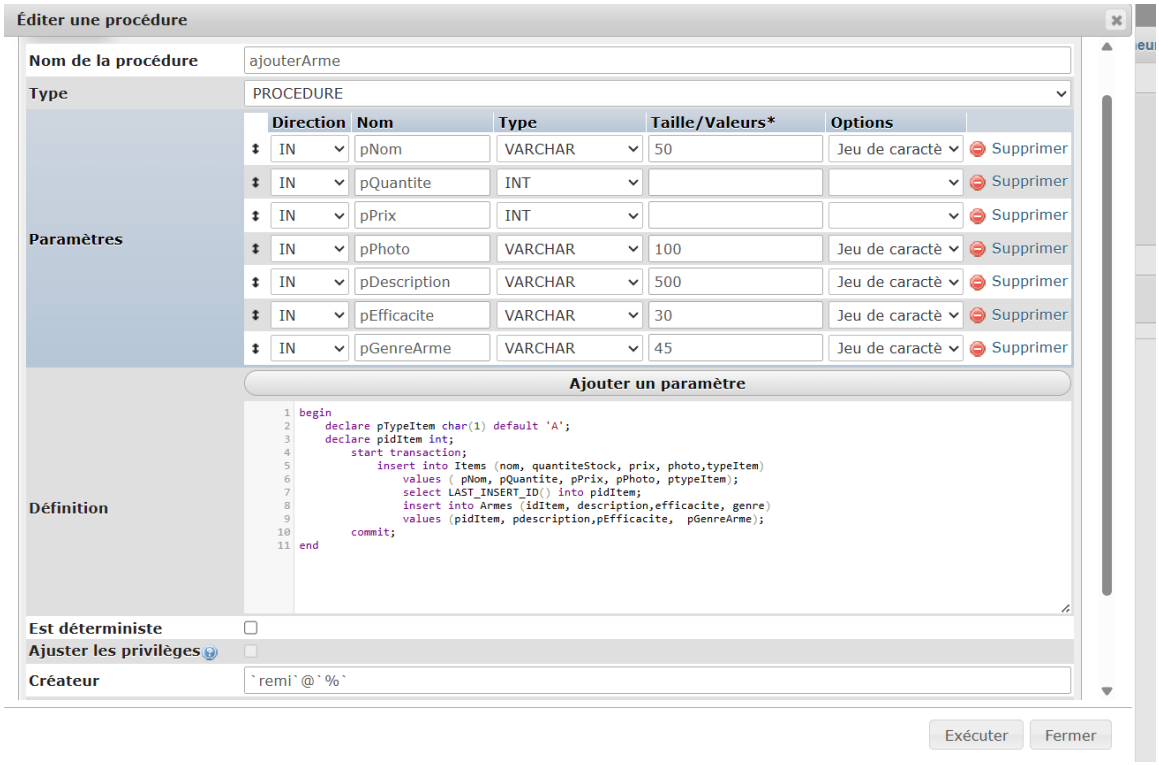
Appel de la procédure :

```
call ajouterSort('unSort',21,60,'sort.jpg',1,1,'A');
```

Dans MySQL Workbench :

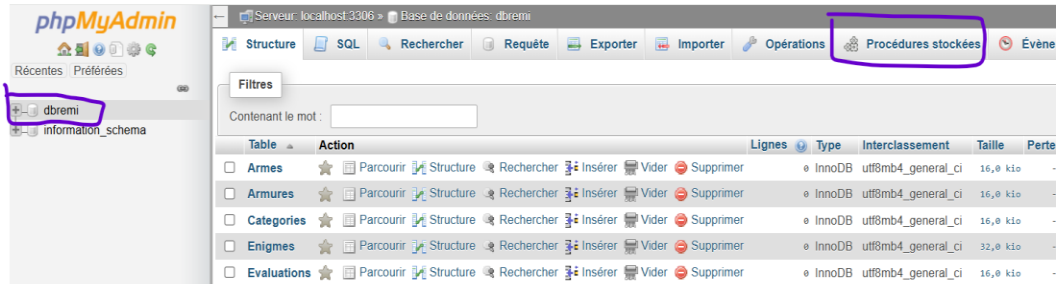
```
use dbsaliha;
drop procedure if exists ajouterArme;
delimiter |
create procedure ajouterArme(
    in pNom varchar(50),
    in pQuantite int,
    in pPrix int,
    in pPhoto varchar(100),
    in pDescription varchar(500),
    in pEfficacite varchar(30),
    in pGenreArme varchar(45))

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 |

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

Dans phpmyadmin



Pour écrire une procédure dans phpmyadmin,

1- Cliquer sur le nom de votre BD, puis sur procédure stockées (ou routine, selon la langue)



2- Nouvelle procédure, puis vous avez juste à écrire la procédure comme le montre l'exemple plus haut. Très facile mais meilleur gestion d'erreurs dans MySQL Workbench

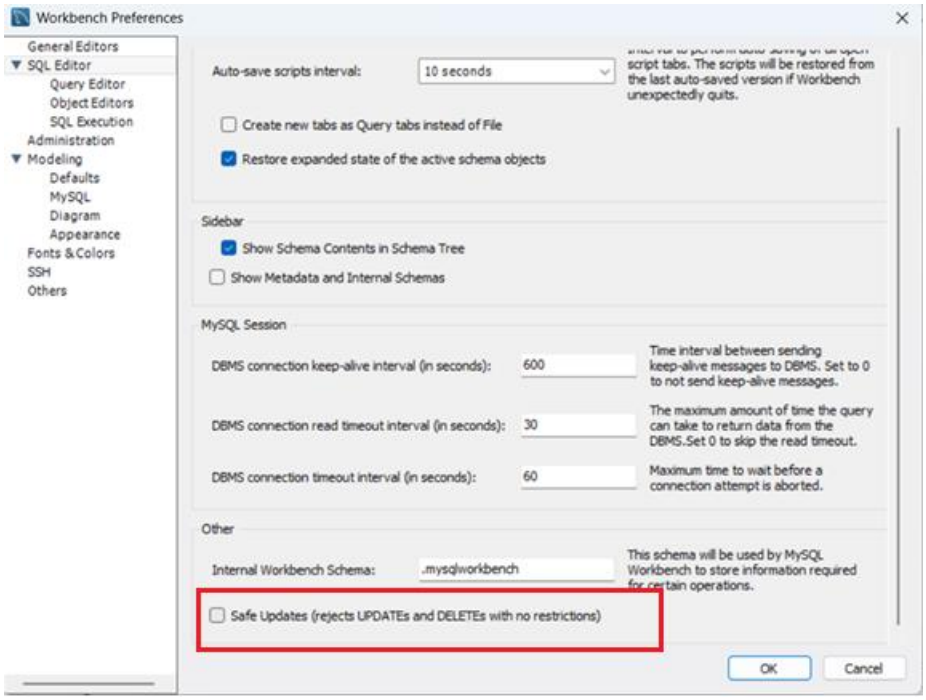
Exemple de trigger pour contrôler le prix des sorts :

```
use dbsaliha;
drop trigger CTRLInserItem;

DELIMITER |;
CREATE TRIGGER CTRLInserItem BEFORE INSERT  ON Items
FOR EACH ROW


begin
declare minPrix int;
declare maxPrix int;

-- si l'item inséré n'est pas un sort, vérifie le prix
-- minimum d'un sort
if(new.typeItem <>'S') Then
       select  min(prix)  into minPrix from Items where typeItem ='S' ;
       if(new.Prix >=minPrix) then
              SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'prix doit être bas';
       end if;
end if;

-- si l'item inséré est un sort, vérifie le prix
-- maximum des autres items
if(new.typeItem='S') Then
       select  max(prix)  into maxPrix from Items where typeItem <>'S' ;
       if(new.Prix <=maxPrix) then
              SIGNAL SQLSTATE '45001' SET MESSAGE_TEXT = 'prix doit être haut';
       end if;
end if;

END |;
```

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.
Le trigger est un Trigger BEFORE
Il n'y a pas de ROLLBACK dans le trigger. La fonction `SIGNAL SQLSTATE permet de faire le ROLLBACK.`

Pour faire un DELETE ou un UPDATE sans que le WHERE porte sur la PRIMARY KEY utilisez : SET SQL_SAFE_UPDATES = 0;

Vous pouvez le faire également par votre interface MySQL Workbench comme suit :
Menu :Edit→Preferences→ Une boite s'ouvre. Sur SQL Editor, puis tout à fait en bas, **décocher** la case « Safe Updates » (voir la figure plus bas)


Workbench Preferences
General Editors
SQL Editor
Query Editor
Object Editors
SQL Execution
Administration
Modeling
Defaults
MySQL
Diagram
Appearance
Fonts & Colors
SSH
Others
Auto-save scripts interval:
10 seconds
script tabs. The scripts will be restored from the last auto-saved version if Workbench unexpectedly quits.
Create new tabs as Query tabs instead of File
Restore expanded state of the active schema objects
Sidebar
Show Schema Contents in Schema Tree
Show Metadata and Internal Schemas
MySQL Session
DBMS connection keep-alive interval (in seconds):
600
Time interval between sending keep-alive messages to DBMS. Set to 0 to not send keep-alive messages.
DBMS connection read timeout interval (in seconds):
30
The maximum amount of time the query can take to return data from the DBMS.Set 0 to skip the read timeout.
DBMS connection timeout interval (in seconds):
60
Maximum time to wait before a connection attempt is aborted.
Other
Internal Workbench Schema:
mysqlworkbench
This schema will be used by MySQL Workbench to store information required for certain operations.
Safe Updates (rejects UPDATEs and DELETEs with no restrictions)
OK
Cancel