



## Système d'inventaire → dans un RPG →

# Système d'Inventaire pour un RPG

HASSAN AMRI [595737]      David Poplawski [587317]  
MEHDI EL MARDAH [589290]      Bui The [546997]

2025 , June 1st

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Diagramme entité-association</b>	<b>4</b>
2.1	Diagramme . . . . .	4
2.2	Contraintes d'intégrité . . . . .	4
2.3	Hypothèses . . . . .	5
2.4	Justification des Choix de Cardinalité . . . . .	5
<b>3</b>	<b>Traduction en Modèle relationnel</b>	<b>7</b>
3.1	Schéma . . . . .	7
3.2	Contraintes d'intégrité . . . . .	8
3.2.1	Contraintes sur les personnages . . . . .	8
3.2.2	Contraintes sur les PNJ (NPC) . . . . .	8
3.2.3	Contraintes sur l'inventaire et les objets . . . . .	8
3.2.4	Contraintes sur les récompenses et le bestiaire . . . . .	8
<b>4</b>	<b>Requêtes</b>	<b>8</b>
4.1	Requête 1 : Les 10 joueurs ayant le plus d'or. . . . .	8
4.2	Requête 2 : Le joueur ayant le plus de personnages de la même classe. . . . .	9
4.3	Requête 3 : La quête ayant la plus grosse récompense en or par niveau de difficulté. . . . .	10
4.4	Requête 4 : Le PNJ possédant l'inventaire contenant les objets dont la valeur en or cumulée est la plus importante. . . . .	10
4.5	Requête 5 : Le type d'objet (arme, armure, potion ou relique) le plus souvent offert en récompense de quêtes de niveau 5. . . . .	11
4.6	Requête 6 : Les monstres avec les meilleures récompenses en valeur en or cumulée en fonction de leurs points de vie. . . . .	12
<b>5</b>	<b>Choix de modélisation</b>	<b>12</b>
5.1	Entités principales . . . . .	12
5.2	Relations principales . . . . .	13
5.3	Gestion des interactions et implémentation . . . . .	13
<b>6</b>	<b>Conclusion</b>	<b>14</b>

# 1 Introduction

Dans l'univers des jeux de rôle (RPG), la gestion de l'inventaire est un élément central de l'expérience du joueur. Que ce soit pour stocker des armes, des potions ou des artefacts rares, un inventaire bien conçu peut enrichir l'immersion et fluidifier la progression dans le jeu. À l'inverse, une mauvaise gestion des objets peut rapidement devenir une source de frustration et nuire à l'expérience de jeu.

Conscients de cette problématique, nous avons entrepris le développement d'un système d'inventaire optimisé permettant aux joueurs d'acquérir, d'organiser et d'utiliser leurs objets de manière intuitive et efficace. La clé de la réussite de ce projet repose sur la mise en place d'un modèle de base de données solide, capable de gérer les interactions complexes entre les personnages, les objets et les différentes mécaniques du jeu.

À travers ce rapport, nous détaillerons la conception de notre base de données, en mettant en avant la structure choisie, les entités clés et les relations essentielles. Nous explorerons également la manière dont les données sont manipulées et exploitées pour assurer une gestion fluide et cohérente des objets dans le jeu.

## 2 Diagramme entité-association

### 2.1 Diagramme

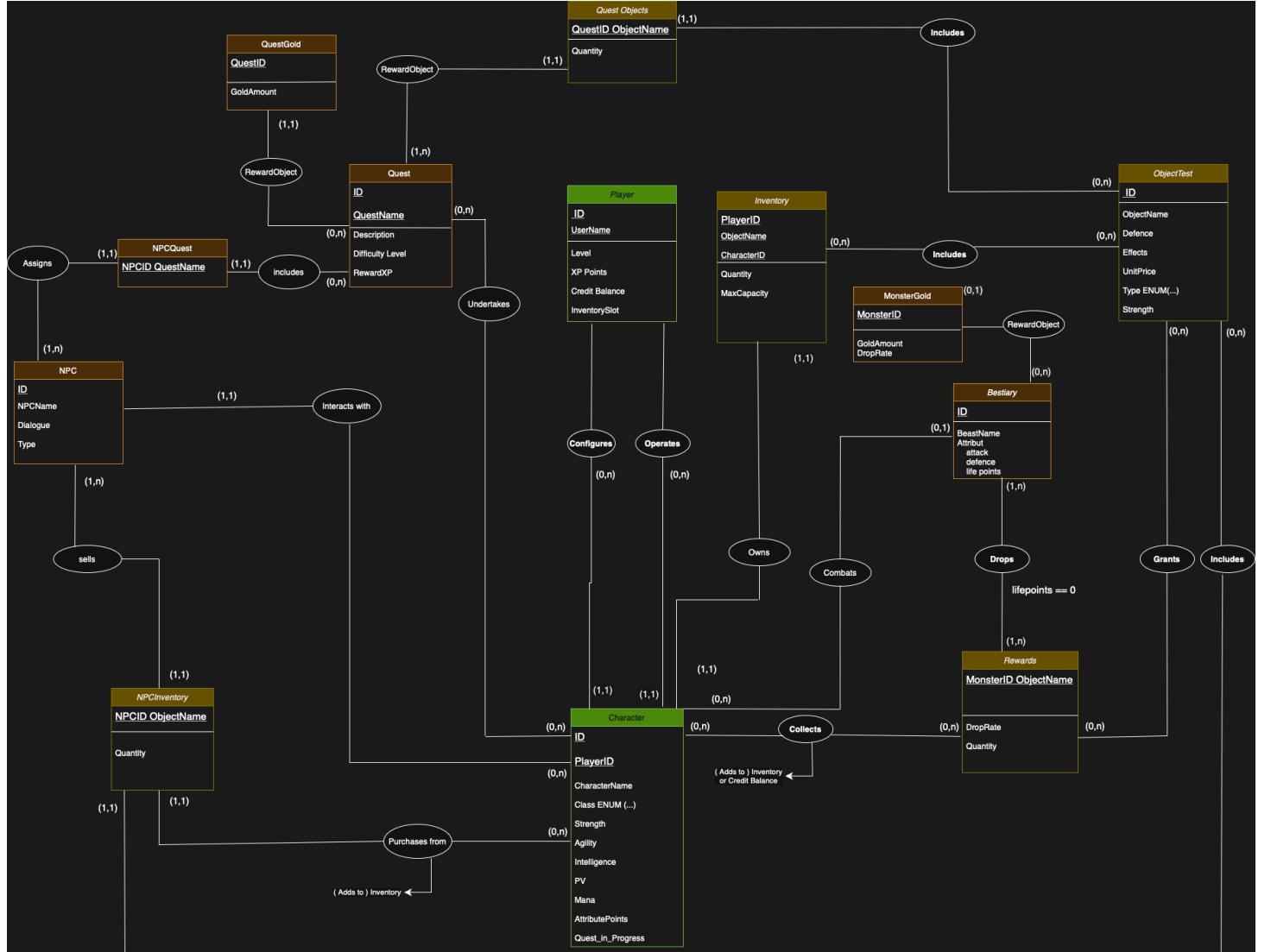


FIGURE 1 – Diagramme entité association

### 2.2 Contraintes d'intégrité

Notre système de gestion d'inventaire pour le jeu RPG repose sur plusieurs contraintes garantissant la cohérence et la validité des données :

- Un **joueur** (Player) peut créer plusieurs **personnages** (CharacterTable), mais chaque personnage appartient à un seul joueur.
- Un joueur ne peut pas avoir deux personnages avec le même nom.
- Un objet (ObjectTest) peut apparaître dans plusieurs inventaires, mais chaque entrée est liée à un seul personnage.
- Un objet récompensé par une quête ou un monstre doit exister dans la table ObjectTest.

- Les crédits d'un joueur (`WalletCredits`) doivent être supérieurs ou égaux à 0 :
- Le prix d'un objet (`ObjectTest.Price`) doit être supérieur à 0 :
- La capacité maximale d'un inventaire (`Inventory.MaxCapacity`) dépend du niveau du joueur (`Player.PlayerLevel`). Cette contrainte fonctionnelle est à gérer via la logique applicative.
- Les points d'expérience récompensés par une quête (`RewardXP`) doivent être strictement positifs :
- Un personnage choisit une seule classe, qui ne peut pas être modifiée.
- Un personnage ne peut collecter, équiper ou jeter qu'un seul type d'objet à la fois. Cette règle métier est à implémenter via la logique applicative ou des procédures.
- Un personnage ne peut récupérer la récompense d'un monstre qu'après l'avoir vaincu. Cette règle doit être assurée via des triggers ou la logique métier du jeu.
- Lorsqu'un personnage est supprimé, tous les objets qu'il possède doivent être supprimés automatiquement :
  - ON DELETE CASCADE sur `Inventory.CharacterID`

## 2.3 Hypothèses

Certaines hypothèses ont été établies pour simplifier la conception du système et garantir la cohérence des interactions dans l'univers du jeu :

- Avant de commencer une partie, le joueur choisit un **niveau de difficulté**, qui influencera la progression. Cette difficulté ne peut être modifiée en cours de partie.
- Un **joueur** possède toujours **au moins un personnage** lorsqu'il commence à jouer.
- Un **personnage** peut interagir avec **un seul PNJ (NPC)** à la fois.
- L'**économie du jeu** est conçue de manière équilibrée : les crédits ne deviennent pas négatifs en conditions normales.
- Les **PNJ ont des dialogues fixes** ; ils ne varient pas dynamiquement selon le joueur ou le contexte.
- Les **récompenses de quêtes** ne sont attribuées qu'à la **fin de la mission**, sans exception.
- Les **objets collectés** par un personnage lui restent attribués tant qu'il ne les utilise pas ou ne les jette pas.
- Un joueur **ne peut pas transférer** ses crédits ni ses objets à un autre joueur.
- Plus le joueur avance dans le jeu, plus les **monstres deviennent puissants et difficiles à vaincre**.
- Certains objets spéciaux ne deviennent disponibles **qu'après une certaine progression** dans l'histoire ou le niveau.
- Le joueur doit être **connecté** pour accéder aux fonctionnalités en ligne.
- Chaque quête possède un **objectif défini**, que le joueur ne peut modifier.
- Un personnage de type `Warrior` ou `Rogue` ne peut utiliser **qu'une seule arme à la fois**.

## 2.4 Justification des Choix de Cardinalité

### 1. Relation entre Player et Character

- **Cardinalité** : Un Player peut jouer avec **0 à n Character(s)**.

- **Justification** : Dans un jeu multijoueur, chaque joueur peut posséder plusieurs personnages pour tester différentes classes (*Guerrier, Mage, Voleur, etc.*). La possibilité d'avoir **0 personnage** permet de prendre en compte le cas d'un nouveau joueur qui vient de s'inscrire sans encore en avoir créé.

## 2. Configuration des Personnages

- **Cardinalité** : Un Player peut configurer **0 à n Character(s)**.
- **Justification** : Chaque joueur peut choisir de ne configurer aucun personnage immédiatement ou, au contraire, de personnaliser plusieurs personnages en fonction de ses préférences stratégiques.

## 3. Interaction entre un Personnage et un NPC

- **Cardinalité** : Un Character peut interagir avec **0 à n NPC(s)**.
- **Justification** : Un personnage peut ne jamais interagir avec un PNJ ou, au contraire, discuter avec plusieurs PNJ au cours du jeu (pour des quêtes, du commerce, etc.).

## 4. Missions attribuées par un NPC

- **Cardinalité** : Un NPC peut attribuer **1 à n Mission(s)**.
- **Justification** : Un PNJ doit obligatoirement proposer au moins une mission (**1**) mais peut en proposer plusieurs (**n**).

## 5. Vente d'objets par un NPC

- **Cardinalité** : Un NPC peut vendre **0 à n Item(s)**.
- **Justification** : Certains PNJ n'ont rien à vendre (**0**), tandis que d'autres disposent d'un inventaire plus large (**n objets**).

## 6. Collecte d'objets par un Personnage

- **Cardinalité** : Un Character peut collecter **0 à n Item(s)**.
- **Justification** : Selon les choix du joueur, un personnage peut ne rien ramasser (**0**) ou accumuler plusieurs objets (**n**) au fil des quêtes et combats.

## 7. Utilisation des équipements par les Classes Warrior et Rogue

- **Cardinalité** : **0 à n Warrior/Rogue** peuvent utiliser (ou non) un **Equipment**.
- **Justification** : Dans le jeu, il peut ne pas y avoir de guerrier/voleur actif (**0**), ou bien plusieurs (**n**). Chaque personnage peut choisir d'utiliser un équipement ou non, selon son niveau et sa stratégie.

## 8. Utilisation des Armes par Warrior et Rogue

- **Cardinalité** :
  - Une **Weapon** peut être utilisée par **0 à 1 Warrior/Rogue**.
  - Un **Warrior/Rogue** peut utiliser **0 à n Weapon(s)**.
- **Justification** : Une arme, une fois équipée, ne peut être utilisée que par un seul guerrier ou voleur (**0 ou 1**). Un personnage peut cependant en équiper plusieurs au fil du jeu (**0 à n**).

## 9. Lancement des Sorts par un Wizard

- **Cardinalité** :
  - Un **Spell** peut être lancé par **0 à 1 Wizard**.
  - Un **Wizard** peut lancer **0 à n Spell(s)**.
- **Justification** : Un sort, lorsqu'il est utilisé, l'est par un seul sorcier (**0 ou 1**), mais un sorcier peut maîtriser et lancer plusieurs sorts durant le jeu (**0 à n**).

## 10. Attribution des Récompenses aux Personnages

- **Cardinalité** :
  - Une **Reward** peut être collectée par **0 à 1 Character**.
  - Un **Character** peut collecter **0 à n Reward(s)**.

- **Justification** : Une récompense, une fois attribuée, est collectée par un seul personnage (**0 ou 1**). Un personnage peut cependant en gagner plusieurs au fil du jeu (**0 à n**).

## 11. Or acquis en tuant un monstre

- **Cardinalité** :

- Un monstre peut drop 0 à plusieurs pièces d'or.
- **Justification** : Certains monstres ne rapportent pas d'or à leur mort, tandis que d'autres peuvent en offrir plusieurs.

## 12. Or acquis en validant une quête

- **Cardinalité** :

- les récompenses d'une quête peuvent contenir 0 ou plusieurs pièces d'or.
- **Justification** : Certaines quêtes ne procurent pas d'or tandis que d'autres en offre en récompenses après avoir été validées.

## 3 Traduction en Modèle relationnel

### 3.1 Schéma

- Player(**ID**, **UserName**, Level, Experience points, Wallet Credits, InventorySlot)
- Character(**ID**, **PlayerID**, Strength, Agility, Intelligence, Pv, Mana, AttributePoints, QuestInProgress)
  - **PlayerID** fait référence à Player.id
- ObjectTest(**UniqueID**, ObjectName, Unitprice, Defence, Effects, Type, Strength )
- Inventory(**PlayerID**, **ObjectName**, **PlayerID**, Quantity, Max capacity)
  - **CharacterID** fait référence à Character.id
  - **ObjectName** fait référence à ObjectTest.ObjectName
  - **PlayerID** fait référence à Player.id
- Bestiary(**BeastId**, BeastName, AttributAttack, AttributDefence, AttributLifePoints)
- MonsterGold(**MonsterID**, GoldAmount, DropRate)
  - **MonsterID** fait référence à Bestiary.ID
- Rewards(**MonsterID**, **ObjectName**, Quantity, Drop Rate)
  - Double clé primaire** (**MonsterID**, **ObjectName**)
    - **MonsterID** fait référence à Bestiary.ID
    - **ObjectName** fait référence à ObjectTest.ObjectName
- NPC(**ID**, NPCName, Dialogue, type)
- Quest(**QuestID**, **QuestName**, Description, Difficulty Level, RewardXP)
- QuestGold(**QuestID**, GoldAmount)
  - **QuestID** fait référence à Quest.ID
- Quest\_Objects(**QuestID**, **ObjectName**, Quantity)
  - Double clé primaire** (**QuestID**, **ObjectName**)
    - **QuestID** fait référence à Quest.ID
    - **ObjectName** fait référence à ObjectTest.ObjectName
- NpcQuest(**NpcID**, **QuestName**)
  - Double clé primaire** (**NpcID**, **QuestName**)
    - **NpcID** fait référence à NPC.id
    - **QuestName** fait référence à Quest.QuestName

- NpcInventory(NpcID, ObjectID, Quantity)
  - Double clé primaire (NpcID, ObjectID)**
  - NpcID fait référence à NPC.id
  - ObjectName fait référence à ObjectTest.ObjectName

## 3.2 Contraintes d'intégrité

Pour garantir la cohérence et le bon fonctionnement du système, plusieurs contraintes d'intégrité ont été mises en place :

### 3.2.1 Contraintes sur les personnages

- Pour tout Character.ID, il existe un et un seul Player.ID associé.
- Pour tout Character.ID, il existe un inventaire qui lui est associé.
- Pour tout Character.ID, une classe doit être choisie.
- Pour tout Character.ID, un personnage peut seulement combattre un Monstre (Bestiary.ID) à la fois.
- Pour tout Character.ID, il ne peut interagir qu'avec un seul NPC.ID à la fois.

### 3.2.2 Contraintes sur les PNJ (NPC)

- Pour tout NPC.ID, il peut exister zéro ou au moins une Quest associée.
- Pour tout NPC.ID, il peut exister zéro ou au moins un Object en vente.
- Pour tout NPC.ID, il doit avoir un type(sorcier, pirate,...).

### 3.2.3 Contraintes sur l'inventaire et les objets

- Pour un même Character.ID, un Object.UniqueID ne peut apparaître qu'une seule fois dans l'inventaire.
- On ne peut pas ajouter d'item que ce soit en achetant dans le marché ou en tuant des monstres, si le poids actuel de l'inventaire est égal à Player.InventorySlot.
- Pour tout Objet que vend les NPC si l'object est hors stock, alors il ne peut pas être ajouter à l'inventaire.

### 3.2.4 Contraintes sur les récompenses et le bestiaire

- Pour toute Bestiary.ID, chaque créature appartient à un seul Bestiary.
- Pour tout Rewards.ObjectName, chaque objet est drop par un seul Monstre (Bestiary.ID).

## 4 Requêtes

### 4.1 Requête 1 : Les 10 joueurs ayant le plus d'or.

Algèbre Relationnelle

$$\lambda_{10} \left( \tau_{\text{desc}(\text{WalletCredits})}(\text{Player}) \right)$$

## SQL

```
SELECT
    ID ,
    UserName ,
    WalletCredits AS Gold
FROM Player
ORDER BY WalletCredits DESC
LIMIT 10;
```

## Calcul Tuple

$$\{ p \in \text{Player} \mid |\{q \in \text{Player} \mid q.\text{WalletCredits} > p.\text{WalletCredits}\}| < 10 \}$$

$$\{ \langle p.\text{ID}, p.\text{UserName}, p.\text{WalletCredits} \rangle \mid \text{Player}(p) \wedge |\{q \mid \text{Player}(q) \wedge q.\text{WalletCredits} > p.\text{WalletCredits}\}| < 10 \}$$

## 4.2 Requête 2 : Le joueur ayant le plus de personnages de la même classe.

### Algèbre Relationnelle

$$PC := \pi_{\text{PlayerID}, \text{Class}}(\text{CharacterTable})$$

$$PC\_Count := \gamma_{\text{PlayerID}, \text{Class}; \text{COUNT}(*)}(\text{CharacterTable})$$

$$MaxCount := \gamma_{\text{NbPers}}(\text{PC\_Count})$$

$$TopPC := \sigma_{\text{NbPers} = \text{MaxNb}}(PC\_Count \bowtie MaxCount)$$

$$JoueurMax := TopPC \bowtie_{\text{PlayerID} = \text{ID}} \text{Player}$$

$$Rsultat := \pi_{\text{UserName}}(\text{JoueurMax})$$

## SQL

```
SELECT p.\text{UserName} , c.\text{Class} , COUNT(*) AS NumberofCharacters
FROM Player p
JOIN CharacterTable c ON p.ID = c.PlayerID
GROUP BY p.\text{UserName} , c.\text{Class}
ORDER BY NumberofCharacters DESC
LIMIT 1;
```

## Calcul Tuple

$$\left\{ p.\text{UserName} \mid \begin{array}{l} \text{Player}(p) \wedge \exists c (\text{CharacterTable}(c) \wedge c.\text{PlayerID} = p.\text{ID} \wedge \\ \forall p_2, cl_2 (\exists c_2 (\text{CharacterTable}(c_2) \wedge c_2.\text{PlayerID} = p_2.\text{ID} \wedge c_2.\text{Class} = cl_2) \\ \Rightarrow \#\{x \mid \text{CharacterTable}(x) \wedge x.\text{PlayerID} = p.\text{ID} \wedge x.\text{Class} = c.\text{Class}\} \\ \geq \#\{y \mid \text{CharacterTable}(y) \wedge y.\text{PlayerID} = p_2.\text{ID} \wedge y.\text{Class} = cl_2\} \end{array} \right\}$$

### 4.3 Requête 3 : La quête ayant la plus grosse récompense en or par niveau de difficulté.

#### Algèbre Relationnelle

```

QuestGoldQuest ←  $\sigma_{\text{Quest.ID} = \text{QuestGold.QuestID}}(\text{Quest} \times \text{QuestGold})$ 
MaxGoldPerLevel ←  $\gamma_{\text{DifficultyLevel}, \max(\text{GoldAmount}) \rightarrow \text{MaxGold}}(\text{QuestGoldQuest})$ 
    Result ←  $\sigma_{\text{QuestGoldQuest.DifficultyLevel} = \text{MaxGoldPerLevel.DifficultyLevel} \wedge}$ 
         $\text{QuestGoldQuest.GoldAmount} = \text{MaxGoldPerLevel.MaxGold}(\text{QuestGoldQuest})$ 
    Résultat final ←  $\pi_{\text{QuestName}, \text{DifficultyLevel}, \text{GoldAmount}}(\text{Result})$ 

```

#### SQL

```

SELECT q. DifficultyLevel , q. QuestName , qg. GoldAmount
FROM Quest q , QuestGold qg
WHERE q. ID = qg. QuestID
AND (q. DifficultyLevel , qg. GoldAmount) IN (
    SELECT q2. DifficultyLevel , MAX(qg2. GoldAmount)
    FROM Quest q2 , QuestGold qg2
    WHERE q2. ID = qg2. QuestID
    GROUP BY q2. DifficultyLevel
);

```

#### Calcul Tuple

$$\{q \mid \text{Quest}(q) \wedge \text{QuestGold}(g) \wedge q.\text{ID} = g.\text{QuestID} \wedge$$

$$\neg \exists q_2 \exists g_2 (\text{Quest}(q_2) \wedge \text{QuestGold}(g_2) \wedge q_2.\text{ID} = g_2.\text{QuestID} \wedge$$

$$q_2.\text{DifficultyLevel} = q.\text{DifficultyLevel} \wedge g_2.\text{GoldAmount} > g.\text{GoldAmount})\}$$

### 4.4 Requête 4 : Le PNJ possédant l'inventaire contenant les objets dont la valeur en or cumulée est la plus importante.

#### Algèbre Relationnelle

$$\lambda_1 \left( \tau_{\text{desc}(\text{ValeurTotale})} \left( \gamma_{n.\text{ID}, n.\text{NpcName}; \sum(i.\text{Quantity} \cdot o.\text{Price}) \rightarrow \text{ValeurTotale}}(\text{NPC} \bowtie \text{NPCInventory} \bowtie \text{ObjectTest}) \right) \right)$$

#### SQL

```

SELECT
    n. ID ,
    n. NpcName ,
    SUM(i. Quantity * o. Price) AS ValeurTotale

```

```

FROM NPC AS n
JOIN NPCInventory AS i ON i.NPCID = n.ID
JOIN ObjectTest AS o ON o.ObjectName = i.ObjectName
GROUP BY n.ID, n.NpcName
ORDER BY ValeurTotale DESC
LIMIT 1;

```

### Calcul Tuple

Le calcul relationnel par tuples classique (TRC) défini par Codd ne permet pas d'exprimer les agrégations (comme la somme). Il est donc impossible d'écrire cette requête purement en TRC sans extensions spécifiques au langage.

*Remarque :* Les opérations telles que SUM, AVG, MAX, etc., nécessitent des extensions du modèle relationnel classique, comme SQL ou TRC avec agrégats.

## 4.5 Requête 5 : Le type d'objet (arme, armure, potion ou relique) le plus souvent offert en récompense de quêtes de niveau 5.

### Algèbre Relationnelle

$$\begin{aligned}
Q5 &:= \sigma_{\text{DifficultyLevel}=5}(\text{Quest}) \\
R &:= Q5 \bowtie_{\text{ID}=\text{QuestID}} \text{Quest\_Objects} \\
RO &:= R \bowtie_{\text{ObjectName}=\text{ObjectName}} \text{ObjectTest} \\
\text{FiltresTypes} &:= \sigma_{\text{Type} \in \{\text{Arme, Armure, Potion, Relique}\}}(RO) \\
\text{Type\_Count} &:= \gamma_{\text{Type}}; \text{COUNT}(\text{*}) \rightarrow \text{NbOfferts}(\text{FiltresTypes}) \\
\text{MaxType} &:= \gamma; \max(\text{NbOfferts}) \rightarrow \text{MaxNb}(\text{Type\_Count}) \\
\text{Resultat} &:= \pi_{\text{Type}}(\sigma_{\text{NbOfferts}=\text{MaxNb}}(\text{Type\_Count} \bowtie \text{MaxType}))
\end{aligned}$$

### SQL

```

SELECT o.Type, COUNT(*) AS Frequency
FROM Quest q
JOIN Quest_Objects qo ON q.ID = qo.QuestID
JOIN ObjectTest o ON qo.ObjectName = o.ObjectName
WHERE q.DifficultyLevel = 5
GROUP BY o.Type
ORDER BY Frequency DESC
LIMIT 1;

```

### Calcul Tuple

$$\left\{ o.Type \mid \begin{array}{l} \text{ObjectTest}(o) \wedge \exists q (\text{Quest}(q) \wedge q.DifficultyLevel = 5 \wedge \\ \forall o_2 (\text{ObjectTest}(o_2) \wedge o_2.Type \neq o.Type \Rightarrow \\ \#\{x \mid \text{Quest}(q_x) \wedge q_x.DifficultyLevel = 5 \wedge \text{Quest\_Objects}(qo) \wedge qo.QuestID = q_x.ID \wedge qo.ObjectName = o.ObjectName\} \\ \geq \#\{y \mid \text{Quest}(q_y) \wedge q_y.DifficultyLevel = 5 \wedge \text{Quest\_Objects}(qo_2) \wedge qo_2.QuestID = q_y.ID \wedge qo_2.ObjectName = o_2.ObjectName\})) \end{array} \right\}$$

## 4.6 Requête 6 : Les monstres avec les meilleures récompenses en valeur en or cumulée en fonction de leurs points de vie.

Algèbre Relationnelle en prenant les 10 premiers

$$\begin{aligned}
 R_1 &:= \text{Rewards} \bowtie \text{ObjectTest} \\
 R_2 &:= \pi_{\text{MonsterID}, (\text{Price} \times \text{Quantity}) \rightarrow \text{ObjGold}}(R_1) \\
 R_3 &:= \gamma_{\text{MonsterID}; \sum(\text{ObjGold}) \rightarrow \text{TotalObjGold}}(R_2) \\
 R_4 &:= \text{MonsterGold} \ltimes R_3 \\
 R_5 &:= \pi_{\text{MonsterID}, \text{GoldAmount}, \text{IFNULL}(\text{TotalObjGold}, 0) \rightarrow \text{TotalObjGold}}(R_4) \\
 R_6 &:= \pi_{\text{MonsterID}, (\text{GoldAmount} + \text{TotalObjGold}) \rightarrow \text{TotalGold}}(R_5) \\
 R_7 &:= \text{Bestiary} \ltimes R_6 \\
 R_8 &:= \pi_{\text{BeastName}, \text{LifePoints}, \text{IFNULL}(\text{TotalGold}, 0) \rightarrow \text{TotalGold}, (\text{TotalGold}/\text{LifePoints}) \rightarrow \text{GoldPerHP}}(R_7) \\
 \text{RésultatFinal} &:= \tau_{\text{GoldPerHP} \text{ DESC}}(R_8)
 \end{aligned}$$

SQL en prenant les 10 premiers

```

SELECT bi.BeastName, bi.LifePoints,
       IFNULL(mg.GoldAmount, 0) + IFNULL(SUM(ot.Price * r.Quantity), 0) AS TotalGoldValue,
       ROUND((IFNULL(mg.GoldAmount, 0) + IFNULL(SUM(ot.Price * r.Quantity), 0)) / bi.LifePoints, 2) AS GoldPerHP
FROM Bestiary bi
LEFT JOIN MonsterGold mg ON bi.ID = mg.MonsterID
LEFT JOIN Rewards r ON bi.ID = r.MonsterID
LEFT JOIN ObjectTest ot ON r.ObjectName = ot.ObjectName
GROUP BY bi.ID, bi.BeastName, bi.LifePoints
ORDER BY GoldPerHP DESC
LIMIT 10;
  
```

Calcul Tuple en prenant les 10 premiers

$$\left\{ t.\text{BeastName}, t.\text{LifePoints}, t.\text{TotalGoldValue}, t.\text{GoldPerHP} \mid \begin{array}{l} \exists b \in \text{Bestiary}, \exists ot \in \text{ObjectTest} \\ t.\text{BeastName} = b.\text{BeastName} \wedge t.\text{LifePoints} = b.\text{LifePoints} \wedge \\ t.\text{TotalGoldValue} = \text{IFNULL}(mg.\text{GoldAmount}, 0) + \text{IFNULL}(\text{SUM}(ot.\text{Price} \times r.\text{Quantity}), 0) \wedge \\ t.\text{GoldPerHP} = \text{ROUND}(t.\text{TotalGoldValue}/b.\text{LifePoints}, 2) \\ \text{avec } mg \in \text{MonsterGold}^?, r \in \text{Rewards}^? \text{ tels que :} \\ mg.\text{MonsterID} = b.\text{ID} \wedge r.\text{MonsterID} = b.\text{ID} \wedge r.\text{ObjectName} = ot.\text{ObjectName} \end{array} \right\}$$

## 5 Choix de modélisation

### 5.1 Entités principales

Dans notre projet de **Système d'inventaire pour un RPG**, plusieurs entités clés sont nécessaires pour assurer une gestion efficace des objets et des personnages.

- **Character** : Représente un personnage du jeu. Chaque personnage possède un inventaire et peut être équipé d'objets spécifiques.
- **ObjectTest** : Désigne un objet qui peut être stocké dans l'inventaire d'un personnage. Un item peut être une arme, une potion, une armure, etc.
- **Quest** : Désigne les quêtes qui sont assignées par les PNJ, avec les **RewardsXP** et la **Description**
- **Inventory** : Représente l'ensemble des objets qu'un character possède. Il est limité en capacité par rapport à son niveau d'avancement dans le jeu et peut être géré via diverses interactions (ajout, suppression, échange, etc.).

- **NPC** : Certains objets peuvent être achetés ou vendus via une interaction avec les NPCs, ce qui implique une gestion des transactions entre les personnages et les NPCs du jeu.
- **CharacterQuest** : Permet de suivre la progression d'un personnage dans une quête, notamment le nombre d'ennemis tués ('BeastKilled') par rapport au nombre requis ('killNumber').
- **Quest\_Objects** : Représente les objets donnés en récompense lorsqu'une quête est complétée. Ce lien explicite permet de définir précisément les gains matériels d'une mission.
- **Bestiary** : Contient les informations sur les créatures du jeu. Chaque monstre a des attributs ('Attack', 'Defence', 'LifePoints') et peut offrir des récompenses.
- **Rewards** : Associe les monstres ('Bestiary') aux objets qu'ils peuvent laisser tomber avec une certaine probabilité ('DropRate') et une quantité ('Quantity').
- **MonsterGold** : Définit la quantité d'or qu'un monstre peut donner après avoir été vaincu, ainsi que la probabilité que cette récompense soit donnée.
- **QuestGold** : Définit la quantité d'or reçue à la fin d'une quête, permettant une gestion différenciée entre objets gagnés ('QuestObjects') et ('QuestGold').
- **NPCQuest** : Permet de lier les PNJ ('NPC') aux quêtes qu'ils peuvent proposer, ce qui reflète leur rôle actif dans la narration.
- **NPCInventory** : Liste les objets que les PNJ peuvent vendre ou échanger avec les joueurs.

## 5.2 Relations principales

- Un **Character** possède un **Inventory**, qui contient plusieurs **Objects**.
- Un **Character** peut équiper certains **Objects**, en fonction des règles du jeu

## 5.3 Gestion des interactions et implémentation

L'implémentation du système d'inventaire repose sur un code source en **C++**, avec une interaction avec **MySQL** pour stocker les informations liées aux objets et personnages.

- Lorsqu'un personnage est créé, un inventaire lui est automatiquement assigné.
- Un personnage peut interagir avec son inventaire via des commandes spécifiques, lui permettant d'ajouter, supprimer, ou équiper des objets.
- Une interface textuelle (terminal) permet aux joueurs de gérer leur inventaire en temps réel. Une version **graphique (GUI)** a été développée pour améliorer l'expérience utilisateur.
- Les transactions avec un **Shop** sont gérées via des requêtes SQL pour assurer la cohérence des données entre la base et l'interface utilisateur.
- Une gestion des **restrictions d'équipement** est mise en place : un guerrier ne peut pas équiper une baguette magique, une armure lourde nécessite un certain niveau, etc.

L'architecture est pensée pour être évolutive et permettre l'ajout de nouvelles fonctionnalités, comme un système de **crafting** ou de **loot** aléatoire.

## 6 Conclusion

En conclusion, ce rapport présente une vue d'ensemble du projet de développement de notre système d'inventaire pour un RPG. Grâce à l'identification et à la modélisation des entités et relations essentielles, nous avons conçu un diagramme entité-relation (E-R) permettant de représenter clairement la structure et le fonctionnement du système.

L'inventaire, élément central de notre jeu, est étroitement lié aux personnages et aux objets. Les joueurs ont accès à diverses fonctionnalités, telles que la collecte, l'équipement et la gestion de leurs ressources. Les mécanismes intégrés garantissent le respect des contraintes définies, telles que la capacité limitée de l'inventaire et les restrictions d'usage des objets en fonction de la progression du joueur.

La base de données, enrichie de données structurées et cohérentes, est opérationnelle et assure une gestion efficace des interactions entre les joueurs, les objets et les quêtes. Les requêtes nécessaires ont également été implémentées et permettent un accès rapide aux informations via les différentes fonctionnalités du jeu.