

Projet SpicyInvaders

Table des matières

7.3.2 Spécificités UX	1
1. Persona	1
a. Contexte.....	1
b. Données récoltés.....	1
c. Constat grâce à mes Personas	1
2. Palette graphique	1
3. Eco-conception	1
7.3.3 Spécificités POO.....	1
7.3.4 Spécificités DB	2
A.....	2
B. Gestion des utilisateurs	3
C. Réaliser et expliquer en détail les requêtes	4
Requête n°1 :	4
Requête n°2 :	5
Requête n°3 :	6
Requête n°4 :	7
Requête n°5 :	8
Requête n°6 (TODO):	9
Requête n°7 :	10
Requête n°8 :	11
Requête n°9 :	12
Requête n°10 (TODO):	13

7.3.2 Spécificités UX

1. Persona

a. Contexte

Je crée d'abord les persona afin de savoir comment créer mes maquettes par la suite. Je crée un petit nombre de persona, 2, afin de pouvoir en créer plus par la suite si j'ai des idées intéressantes.

b. Données récoltées

Je récupérerai l'âge, le genre, le domicile, le travail et le statut familial en premier.

Je créerai par la suite une biographie de mon persona à l'aide de ChatGPT.

À l'aide de graphique j'estime à quel point mon persona apprécie certains aspects du jeu tel que :

- Faire les meilleures scores que possibles
- Atteindre le plus haut niveau possible
- Explorer des modes de jeu alternatif
- Faire des défis (de la communauté)
- Passer le temps

Je cherche ce que souhaite et ne souhaite pas rencontrer mon persona sur un jeu de Space Invaders. J'établis un pourcentage des appareils utilisés pour jouer à Space Invaders afin de savoir comment adapter mon jeu.

c. Constat grâce à mes Personas

Mon premier persona, Robbie Heineman me fait prendre conscience que le premier jeu Space Invaders étant sorti en 1978 il peut s'adresser à des personnes actuellement âgé et qu'il vaut donc mieux ne pas créer trop d'effet spéciaux pour ne pas déranger l'utilisateur. De plus les jeux Space Invaders étant sorti sur arcade il pourrait être utile d'avoir des contrôles imitant les bornes d'arcade. D'anciens joueurs pourraient aussi souhaiter avoir une version original du jeu.

Mon second persona, Martine Dupont, me fait prendre conscience que mes utilisateurs pourraient jouer à Space Invaders depuis leur téléphone. Ils pourraient vouloir jouer à plusieurs en local comme en multijoueur. De nouveau joueur pourrait souhaiter avoir facilement accès à des modifications sur leur partie, des niveaux afin qu'une communauté puisse proposer des ajouts sur le jeu, il pourrait donc être utile que le jeu soit ouvert à la création de mod.

Mon troisième persona, TODO, me fait prendre conscience que mon jeu doit être accessible au daltonien

2. Palette graphique

Pour la palette de couleur de base je voulais évoquer l'espace en utilisant ces couleurs :« #324856 #4A746A #D18237 #D66C44 »

3. Eco-conception

7.3.3 Spécificités POO

7.3.4 Spécificités DB

A.Importer les données et le schéma de base de données

Création de la base de données :

```
docker exec -i db mysql -uroot -proot < db_space_invaders.sql
```

B. Gestion des utilisateurs

1. Administrateur du jeu

- *Peut créer, lire, mettre à jour et supprimer (CRUD) n'importe quelle table.*
- *Gérer les utilisateurs et leurs privilèges.*

```
CREATE ROLE 'Admin-jeu';
```

```
GRANT ALL PRIVILEGES ON *.* TO 'Admin-jeu' WITH GRANT OPTION;
```

2. Joueur

- *Lire les informations des armes (pour voir quelles armes il peut acheter).*
- *Créer une commande.*
- *Lire toutes les commandes.*

```
CREATE ROLE 'joueur';
```

```
GRANT SELECT ON db_space_invaders.t_arme TO 'joueur';
```

Créer une commande TODO (CREATE ?)

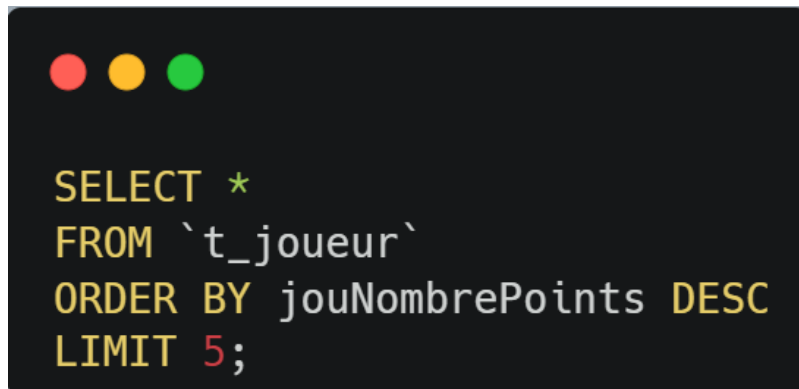
C. Réaliser et expliquer en détail les requêtes

Requête n°1 :

La première requête que l'on vous demande de réaliser est de sélectionner les 5 joueurs qui ont le meilleur score c'est-à-dire qui ont le nombre de points le plus élevé. Les joueurs doivent être classés dans l'ordre décroissant

Réaliser la requête :

```
SELECT * FROM `t_joueur` ORDER BY jouNombrePoints DESC LIMIT 5;
```

A screenshot of a terminal window with a dark background. At the top left, there are three colored circles: red, yellow, and green. The SQL query is displayed in a monospaced font with syntax highlighting: 'SELECT' is yellow, '*' is yellow, 'FROM' is yellow, '`t_joueur`' is white, 'ORDER BY' is yellow, 'jouNombrePoints' is white, 'DESC' is yellow, 'LIMIT' is yellow, '5' is red, and ';' is red.

Explication des nouveaux éléments de la requête :

`SELECT * FROM `t_joueur`` : permet de sélectionner tous les attributs de la table `t_joueur`.

`ORDER BY jouNombrePoints DESC` : le `DESC` permet d'afficher les valeurs dans l'ordre décroissant du `ORDER BY` selon les valeurs dans jouNombrePoints.

`LIMIT 5 ;` : me permet de n'afficher que les 5 premiers résultats et le `;` me permet de terminer la commande.

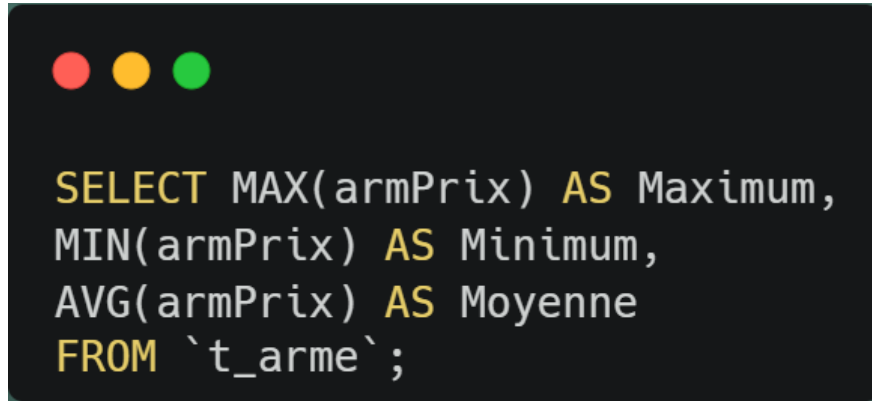
Requête n°2 :

Trouver le prix maximum, minimum et moyen des armes.

Les colonnes doivent avoir pour nom « Prix Maximum », « PrixMinimum » et « PrixMoyen »

Réaliser la requête :

```
SELECT MAX(armPrix) AS Maximum, MIN(armPrix) AS Minimum, AVG(armPrix) AS Moyenne FROM `t_arme`;
```



Explication des nouveaux éléments de la requête :

`SELECT Max(armPrix) AS Maximum` : permet d'afficher la plus grande valeur dans l'attribut armPrix. La virgule permet de sélectionner d'autres valeurs.

`MIN(armPrix) AS Minimum` : me permet d'afficher le plus petit résultat dans l'attribut armPrix. La virgule permet de sélectionner d'autres valeurs.

`AVG(armPrix) AS Moyenne FROM `t_arme` ;` : me permet d'afficher la moyenne des données dans l'attribut armPrix.

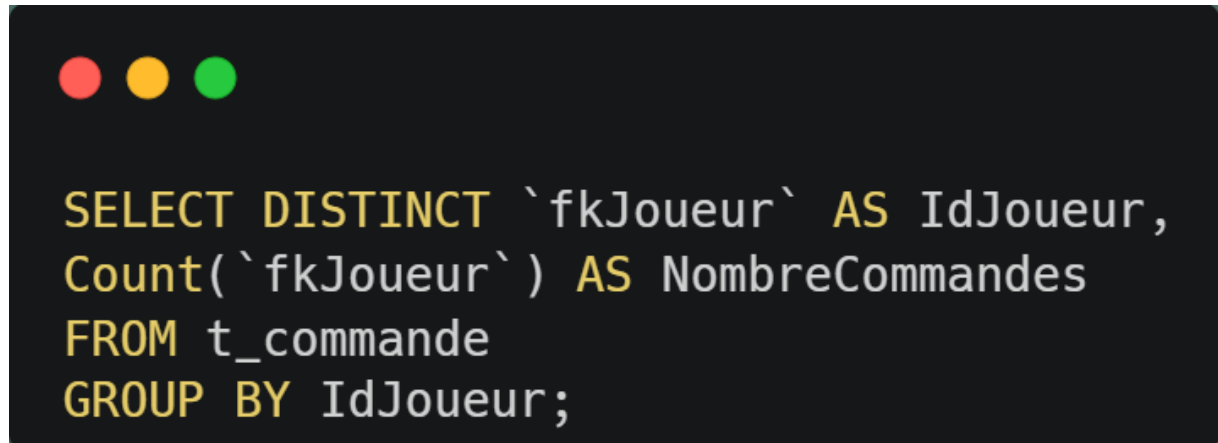
Requête n°3 :

Trouver le nombre total de commandes par joueur et trier du plus grand nombre au plus petit.

La 1^{ère} colonne aura pour nom "IdJoueur", la 2^{ème} colonne aura pour nom "NombreCommandes".

Réaliser la requête :

```
SELECT DISTINCT `fkJoueur` AS IdJoueur, Count(`fkJoueur`) AS NombreCommandes  
FROM t_commande GROUP BY IdJoueur;
```

A screenshot of a terminal window with a dark background. At the top left, there are three colored circles: red, yellow, and green. The SQL query is displayed in a monospaced font with syntax highlighting: 'SELECT DISTINCT `fkJoueur` AS IdJoueur, Count(`fkJoueur`) AS NombreCommandes FROM t_commande GROUP BY IdJoueur;'. The keywords 'SELECT', 'DISTINCT', 'FROM', and 'GROUP BY' are in yellow, while the identifiers and function names are in white or light blue.

```
SELECT DISTINCT `fkJoueur` AS IdJoueur,  
Count(`fkJoueur`) AS NombreCommandes  
FROM t_commande  
GROUP BY IdJoueur;
```

Explication des nouveaux éléments de la requête :

`SELECT DISTINCT `fkJoueur` AS IdJoueur,` Permet d'afficher les valeurs de fkJoueur une seule fois par valeur.

`Count(`fkJoueur`) AS NombreCommandes` Permet de compter le nombre de valeur dans fkJoueur.

`FROM t_commande GROUP BY IdJoueur;` Permet de sélectionner la table t_commande puis grouper par l'identifiant du joueur afin de calculer le nombre de commande par joueur et non pas au total.

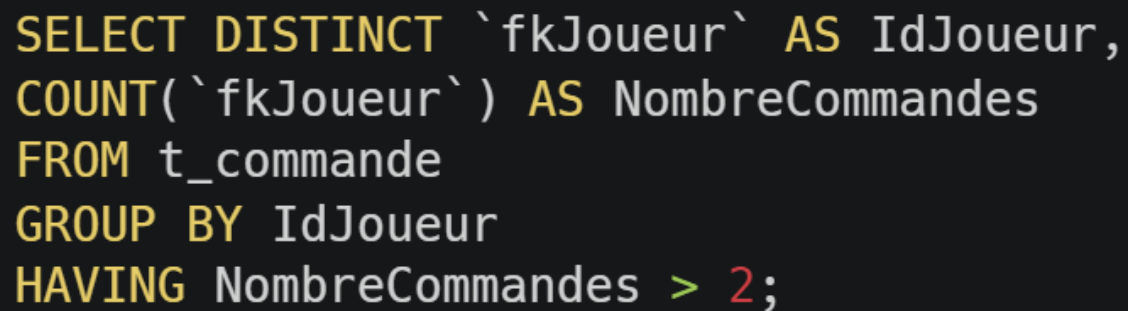
Requête n°4 :

Trouver les joueurs qui ont passé plus de 2 commandes.

La 1^{ère} colonne aura pour nom "IdJoueur", la 2^{ème} colonne aura pour nom "NombreCommandes "

Réaliser la requête :

```
SELECT DISTINCT `fkJoueur` AS IdJoueur, COUNT(`fkJoueur`) AS NombreCommandes  
FROM t_commande GROUP BY IdJoueur HAVING NombreCommandes > 2;
```

A screenshot of a terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top left corner. The terminal displays a SQL query in a monospaced font with syntax highlighting: 'SELECT DISTINCT `fkJoueur` AS IdJoueur, COUNT(`fkJoueur`) AS NombreCommandes FROM t_commande GROUP BY IdJoueur HAVING NombreCommandes > 2;'. The keywords 'SELECT', 'DISTINCT', 'FROM', 'GROUP BY', and 'HAVING' are in yellow, the table name 't_commande' is in white, and the column names and values are in white. The comparison operator '>' and the number '2' are in red.

```
SELECT DISTINCT `fkJoueur` AS IdJoueur,  
COUNT(`fkJoueur`) AS NombreCommandes  
FROM t_commande  
GROUP BY IdJoueur  
HAVING NombreCommandes > 2;
```

Explication des nouveaux éléments de la requête :

Ceci est la même commande que celle de la requête 3 hormis l'utilisation de

```
HAVING NombreCommandes > 2
```

Avec ceci on ne sélectionne que les lignes qui ont un nombre de commande supérieur à 2.

Requête n°5 :

Trouver le pseudo du joueur et le nom de l'arme pour chaque commande.

Réaliser la requête :

```
SELECT t_joueur.jouPseudo, t_arme.armNom FROM t_joueur INNER JOIN t_commande ON t_joueur.idJoueur = t_commande.fkJoueur INNER JOIN t_detail_commande ON t_commande.idCommande = t_detail_commande.fkCommande INNER JOIN t_arme ON t_detail_commande.fkArme = t_arme.idArme;
```



```
SELECT t_joueur.jouPseudo, t_arme.armNom
FROM t_joueur
INNER JOIN t_commande ON t_joueur.idJoueur = t_commande.fkJoueur
INNER JOIN t_detail_commande ON t_commande.idCommande = t_detail_commande.fkCommande
INNER JOIN t_arme ON t_detail_commande.fkArme = t_arme.idArme;
```

Explication des nouveaux éléments de la requête :

`INNER JOIN t_commande ON t_joueur.idJoueur = t_commande.fkJoueur` Le `INNER JOIN` permet de créer une jointure entre la table `t_commande` et la table `t_joueur`. Cela permet de sélectionner des éléments sur plusieurs tables. Le `ON` par la suite permet de ne prendre **que** les valeurs sélectionnées auparavant ou la clé primaire de `t_joueur`(`idJoueur`)et la clé étrangère de `t_commande` sont équivalentes. La clé étrangère étant le référencement entre les 2 tables.

Dans cet exercice j'utilise un `INNER JOIN` pour ne récupérer que les valeurs reliés à d'autres tables.

Requête n°6 (TODO):

Trouver le total dépensé par chaque joueur en ordonnant par le montant le plus élevé en premier, et limiter aux 10 premiers joueurs.

La 1^{ère} colonne doit avoir pour nom "IdJoueur" et la 2^{ème} colonne "TotalDepense".

Réaliser la requête :

```
SELECT t_joueur.idJoueur AS IdJoueur, SUM(armPrix) AS TotalDepense FROM t_a  
rme INNER JOIN t_arsenal ON t_arme.idArme = t_arsenal.fkArme INNER JOIN t_j  
oueur ON t_arsenal.fkJoueur = t_joueur.idJoueur GROUP BY IdJoueur LIMIT 10;
```



```
SELECT t_joueur.idJoueur AS IdJoueur,  
SUM(armPrix) AS TotalDepense  
FROM t_arme  
INNER JOIN t_arsenal ON t_arme.idArme = t_arsenal.fkArme  
INNER JOIN t_joueur ON t_arsenal.fkJoueur = t_joueur.idJoueur  
GROUP BY IdJoueur LIMIT 10;
```

Explication de la commande :

Utilisation de 2 `INNER JOIN` afin de relier 2 tables qui sont séparées par une table. Dans ce cas la table arsenal est entre la table joueur et arme, elle contient les clefs privés de la table joueur et de la table arme.

Requête n°7 :

Récupérez tous les joueurs et leurs commandes, même s'ils n'ont pas passé de commande.

Dans cet exemple, même si un joueur n'a jamais passé de commande, il sera quand même listé, avec des valeurs 'NULL' pour les champs de la table 't_commande',

Réaliser la requête :

```
SELECT idJoueur, t_commande.idCommande, t_commande.comDate, t_commande.comNumeroCommande FROM t_joueur LEFT JOIN t_commande ON t_joueur.idJoueur = t_commande.fkJoueur;
```



```
SELECT idJoueur, t_commande.idCommande,  
t_commande.comDate, t_commande.comNumeroCommande  
FROM t_joueur  
LEFT JOIN t_commande ON t_joueur.idJoueur = t_commande.fkJoueur;
```

Explication de la commande :

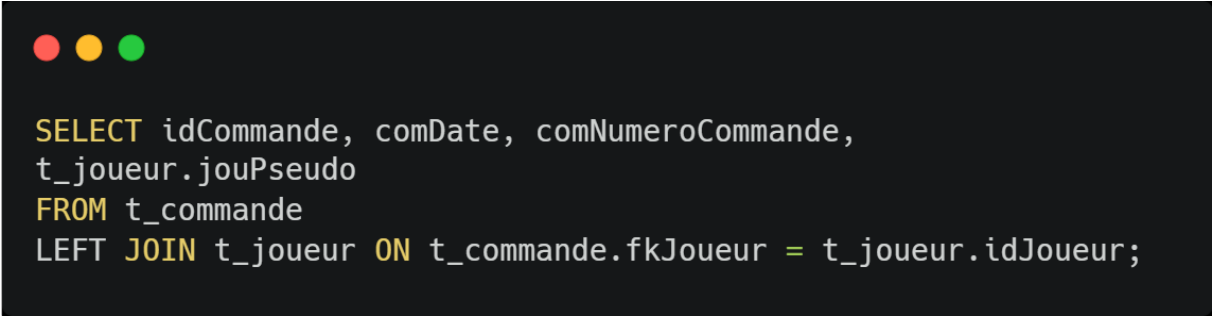
Utilisation de `LEFT JOIN` pour faire une jointure au lieu d'un `INNER JOIN` afin de récupérer tout les joueurs même s'ils n'ont aucun lien avec la table commande.

Requête n°8 :

Récupérer toutes les commandes et afficher le pseudo du joueur si elle existe, sinon montrer 'NULL' pour le pseudo.

Réaliser la requête :

```
SELECT idCommande, comDate, comNumeroCommande, t_joueur.jouPseudo FROM t_commande LEFT JOIN t_joueur ON t_commande.fkJoueur = t_joueur.idJoueur;
```



```
SELECT idCommande, comDate, comNumeroCommande,  
t_joueur.jouPseudo  
FROM t_commande  
LEFT JOIN t_joueur ON t_commande.fkJoueur = t_joueur.idJoueur;
```

Explication de la commande :

Utilisation d'un `LEFT JOIN` afin de récupérer toutes les valeurs de la table commande même si elles ne sont pas liés à la table joueur. De cette façon si la commande n'est relié à aucun joueur elle sera quand même affiché et le nom du joueur restera 'NULL'.

Requête n°9 :

Trouver le nombre total d'armes achetées par chaque joueur (même si ce joueur n'a acheté aucune Arme).

Réaliser la requête :

```
SELECT SUM(`detQuantiteCommande`) AS nbr_commandes, t_joueur.idJoueur FROM  
t_detail_commande INNER JOIN t_commande ON t_detail_commande.fkCommande = t  
_commande.idCommande INNER JOIN t_joueur ON t_commande.fkJoueur = t_joueur.  
idJoueur GROUP BY t_joueur.idJoueur;
```

A screenshot of a terminal window with a dark background. At the top left, there are three colored circles: red, yellow, and green. Below them, a SQL query is displayed in a light-colored monospace font. The query is: `SELECT SUM(`detQuantiteCommande`) AS nbr_commandes, t_joueur.idJoueur FROM t_detail_commande INNER JOIN t_commande ON t_detail_commande.fkCommande = t_commande.idCommande INNER JOIN t_joueur ON t_commande.fkJoueur = t_joueur.idJoueur GROUP BY t_joueur.idJoueur;`

```
SELECT SUM(`detQuantiteCommande`) AS nbr_commandes, t_joueur.idJoueur  
FROM t_detail_commande  
INNER JOIN t_commande ON t_detail_commande.fkCommande = t_commande.idCommande  
INNER JOIN t_joueur ON t_commande.fkJoueur = t_joueur.idJoueur  
GROUP BY t_joueur.idJoueur;
```

Explication de la commande :

Afin de sélectionner le nombre total d'arme acheté je fais un `SUM` de la quantité commandé car une commande peut comporter plusieurs armes. Par la suite je rejoin les tables nécessaires pour pouvoir accéder de la table `t_detail_commande` dans laquelle il y a le nombre de commande et la table `t_joueur` ou je peux obtenir l'identifiant et le nom du joueur.

Requête n°10 (TODO):

Trouver les joueurs qui ont achetés plus de 3 types d'armes différentes.

Réaliser la requête :

```
SELECT t_joueur.jouPseudo, Count(t_arme.idArme) AS NBRarmedifferente FROM t_arsenal INNER JOIN t_joueur ON t_arsenal.fkJoueur = t_joueur.idJoueur INNER JOIN t_arme ON t_arsenal.fkArme = t_arme.idArme GROUP BY t_joueur.jouPseudo HAVING NBRarmedifferente > 2;
```

OU

```
SELECT t_joueur.idJoueur, SUM(t_detail_commande.detQuantiteCommande) AS nbr FROM t_joueur INNER JOIN t_commande ON t_joueur.idJoueur = t_commande.fkJoueur INNER JOIN t_detail_commande ON t_commande.idCommande = t_detail_commande.fkCommande GROUP BY t_joueur.idJoueur HAVING SUM(t_detail_commande.detQuantiteCommande) > 3;
```

Explication de la commande :

D. Création des index

1. Pourtant certains index existent déjà. Pourquoi ?

MySQL crée automatiquement des indexes sur les clefs primaires et sur les clefs étrangères.

Ces indexes sont créés automatiquement car les clefs primaires et les clefs étrangères sont fréquemment utilisés lors de requêtes, spécifiquement lorsque l'on souhaite créer des jointures

2. Quels sont les avantages et les inconvénients des index ?

Avantages :

Les indexes permettent de structurer les données afin d'effectuer des recherches dans les données beaucoup plus rapides

Inconvénients :

Les indexes prennent de la place en mémoire

Ils ralentissent les requêtes car l'index doit se remettre à jour à chaque changement :

- d'insertion
- de modification
- de suppression

3. Sur quel champ (de quelle table), cela pourrait être pertinent d'ajouter un index ?

Sur les clefs primaires et les clefs étrangères (dans le cas où elles n'ont pas été automatiquement créées par le SGBDR utilisé). Cela permettrait d'effectuer des requêtes plus rapidement lorsque l'on souhaite effectuer des jointures. De plus des indexes peuvent être rajoutés.

E. Backup / Restore

Backup de la base de donnée :

```
Mysqldump -uroot -proot nom_db > nom.sql
```

```
Mysqldump -u root -p nom_db > nom.sql
```

Explication de la Backup

`Mysqldump` permet de spécifier que c'est l'utilitaire « Mysqldump » qui est utilisé.

`-uroot` spécifie que l'utilisateur est `ROOT`

`-p` spécifie que l'on va rentrer le password. Rien n'est mis à la suite afin de ne pas le divulguer.

Restore la base de donnée :

```
Mysql -uroot -p < nom.sql
```

Explication du Restore :