BDR - Cahier des charges : Plateforme de jeux

Arthur Bécaud, Bruno Egremy et Stéphane Teixeira Carvalho

Janvier 2020

Table des matières

1 Introduction

Dans le cadre d'un laboratoire, il nous a été demandé de mettre en place un projet exploitant les fonctionnalités d'un serveur MySQL.

Pour cela nous avons décidés de créer un site de vente de jeux vidéo. Cette idée nous est venus car nous utilisons fréquemment un site de vente très similaire à notre idée se nommant Steam. Ce site permet d'acheter des jeux ainsi que d'avoir sa propre bibliothèque de jeu.

Dans un premier temps, nous avons dû mettre en place une description du projet montrant les différentes fonctionnalités que notre projet mettra à disposition ainsi qu'une liste des caractéristiques de chaque entité de notre application.

Ensuite, nous avons créer les schémas EA et MR de notre base de données. Ces schémas sont accompagnés d'explications détaillants nos différents choix de conception.

S'en suit le développement d'une application web permettant de réaliser les différentes fonctionnalités listées et décrites dans la suite de ce document. Les technologies utilisée sont :

- Un serveur web Apache
- PHP
- HTML CSS (Bootstrap)
- Ajax et JQuery (vient avec Bootstrap)
- MySQL
- Git

2 Description du projet

Comme expliqué auparavant le but de notre application est de proposer un site servant de plateforme de vente de jeu vidéo comme le fait actuellement Steam. Notre application contiendra des comptes qui pourront effectués des achats soit pour eux soit pour leur amis.

2.1 Fonctionnalités proposées

Notre application proposera donc les fonctionnalités suivantes :

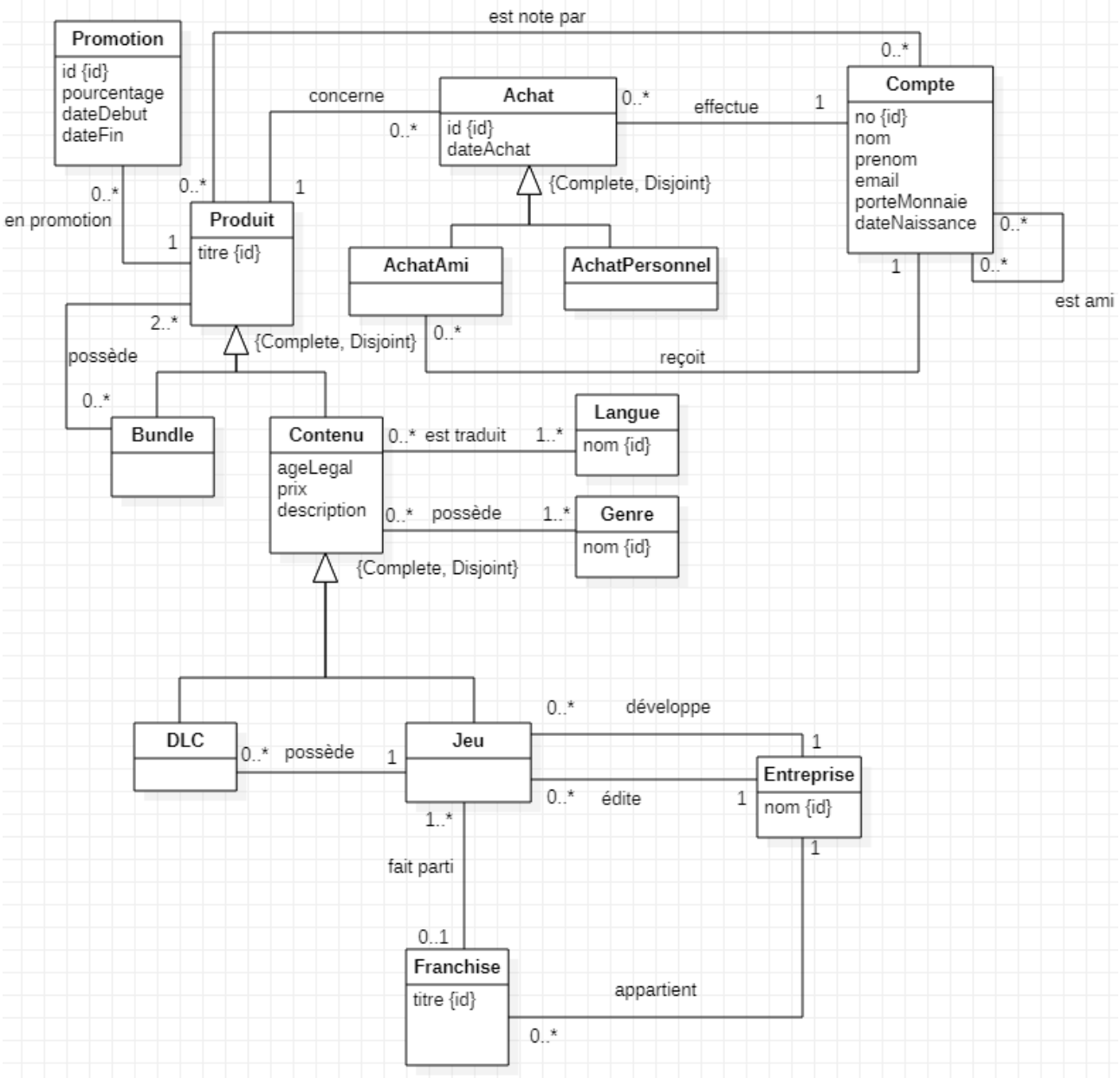
- Note site proposera des jeux, des DLCs ainsi que des Bundle. Un Bundle contient plusieurs jeux/DLCs/-Bundle disponibles sur notre site.
- Un utilisateur peut créer un compte. Il aura aussi la possibilité de supprimer son compte.
- Un client peut acheter un jeu grâce à son porte-monnaie lié au site. Il est aussi possible d'offrir un jeu à un ami.
- Un jeu peut changer de prix dans le temps (promotion)
- Historique des ventes des jeux, principalement pour les statistiques des ventes.
- Un compte possède une liste d'amis à laquelle il peut ajouter ou supprimer des amis.
- Un compte peut voir les jeux auxquels jouent ses amis.
- Un magasin sera disponible avec les différents jeux proposés avec leur prix avec promotion si celui-ci en a une.
- Un compte peut évaluer un jeu auquel il a joué, modifiant la note général du jeu.

2.2 Entité dans l'application

Voici la liste des caractéristiques de chacune de entités représenté dans notre service.

1. Compte :
   - un propriétaire
   - un numéro de compte
   - un email
   - un porte-monnaie
   - une bibliothèque de jeu
   - une liste d'amis
   - Propriétaire
     - un nom
     - un prénom
     - une adresse
     - une date de naissance
2. Jeu/DLC :
   - un titre
   - un prix
   - un âge légal
   - un développeur
   - un éditeur
   - des genres
   - une description
   - une note
   - une franchise
   - des langues
3. Bundle :
   - un titre
   - un prix
   - un âge légal
   - une note
4. Vente :
   - un jeu
   - une promotion
   - une date de début de vente
   - une date de fin de vente
5. Franchise :
   - un nom
   - un éditeur
6. Developpeur :
   - un nom
7. Editeur :
   - un nom

3 Schéma EA


Promotion
id {id}
pourcentage
dateDebut
dateFin
est note par
0..*
concerne
Achat
id {id}
dateAchat
0..*
effectue
1
Compte
no {id}
nom
prenom
email
porteMonnaie
dateNaissance
0..*
0..*
est ami
0..*
en promotion
1
Produit
titre {id}
{Complete, Disjoint}
AchatAmi
AchatPersonnel
2..*
possède
{Complete, Disjoint}
0..*
reçoit
Bundle
Contenu
ageLegal
prix
description
0..* est traduit 1..*
Langue
nom {id}
0..* possède 1..*
Genre
nom {id}
{Complete, Disjoint}
DLC
0..* possède 1
Jeu
0..* développe
1
Entreprise
nom {id}
0..* édite 1
1..*
fait parti
0..1
Franchise
titre {id}
appartient
0..*
1

3.1 Héritages

Dans notre schéma EA nous avons mis en place deux héritages :

3.1.1 Produit ⟵ *Bundle*, *Contenu*

Cet héritage permettra d'indiquer les produits mis en vente sur notre plateforme. Ces produits seront soit un bundle soit un contenu. Un produit est défini par un titre et une note. Le prix d'un contenu sera défini à l'ajout dans la base de donnée, mais pour le prix d'un bundle, il sera calculé en fonction de l'addition des prix de ses contenus. de plus, un bundle peut contenir des bundles également.

3.1.2 Contenu ⟵ *DLC*, *Jeu*

Cet héritage est mis en place car les entitées jeu et DLC ont des données en commun qui sont l'âge légal, une description, un prix, un éditeur et un développeur. Cependant, un jeu peut posséder des DLC. En différenciant jeu de DLC, nous empêchons un jeu de posséder d'autres jeux.

3.1.3 Achat ⟵ *AchatAmi*, *AchatPersonnel*

Cet héritage est mis en place pour permettre lors d'un achat d'effectuer un achat soit pour soi-même soit pour un ami. Grâce à cela nous pourrons alors différencer 2 types d'achat ceux pour soi-même et ceux pour un ami. Pour pouvoir avoir un compte ami il faut alors créer une association entre la table AchatAmi et Compte. Un compte peut alors recevoir de 0 à plusieurs achats mais un achat n'est lié qu'à un compte.

3.2 Association

Dans cette section sont expliquées les différentes associations misent en place sur notre schéma.

3.2.1 Compte achète Produit

Un compte client peut acheter de 0 à plusieurs produits. La cardinalité minimale de 0 est destinée à la création d'un compte ainsi, il n'est pas contraint à l'achat d'un produit directement.

Dans cette association, nous avons créé une classe d'association contenant la date d'achat d'un jeu. Cela pourra nous permettre de garder un historique des achats effectués par un compte et faire des statistiques des ventes. Un jeu peut être vendu de 0 (s'il vient d'être mis en vente) à plusieurs fois, pour des raisons évidentes.

La bibliothèque de jeu d'un compte sera alors calculée à l'aide des jeux achetés ou offert pour ledit compte.

3.2.2 Un Bundle possède des Produits

Cette association permet de mettre des produits dans un bundle. Un bundle ne peut pas être créer sans avoir au minimum 2 produits, il n'aurait aucunes raisons d'être sinon. Comme mentionné précédemment, un bundle peut être constitué d'autre bundle pour autant qu'aucun produit ne soit dupliqué.

3.2.3 Un Produit peuvent avoir des Promotions

Cette association permet de faire des promotions sur le prix d'un bundle ou d'un contenu. De plus, pour un bundle ou un contenu, plusieurs promotions peuvent s'appliquer en même temps. Cela pour d'une part, cumuler des promotions et d'autre part, garder un historiques de celles-ci. Une promotion peut être appliqué qu'à un seul produit

Lorsqu'une promotion sera entrée et liée, soit à un contenu soit à un bundle, le prix de ces produits sera changé en conséquence.

3.2.4 Un Contenu est traduit en Langue

Un contenu peut être disponible dans 1 à plusieurs langues. L'entité Langue a été créée afin d'éviter la redondance de données avec les différentes langues disponibles. Le même principe a été utilisé pour l'association Contenu -> Genre.

3.2.5 Un Jeu possède des DLC

Cette association oblige à lier un DLC à un jeu car un DLC est une extension du jeu de base. Un jeu pourra alors contenir de 0 à plusieurs DLC mais un DLC ne peut être lié qu'à un seul jeu.

3.2.6 Une Franchise appartient à une Entreprise et un Jeu est édité ou développé par une entreprise

Pour l'association "Une franchise appartient à une entreprise" nous avons mis une cardinalité de 1 de franchise vers Entreprise car une franchise appartient qu'à une seule entreprise. Nous avons également mis une cardinalité de entreprise vers Franchise à 0..* car une entreprise peut avoir plusieurs franchise.

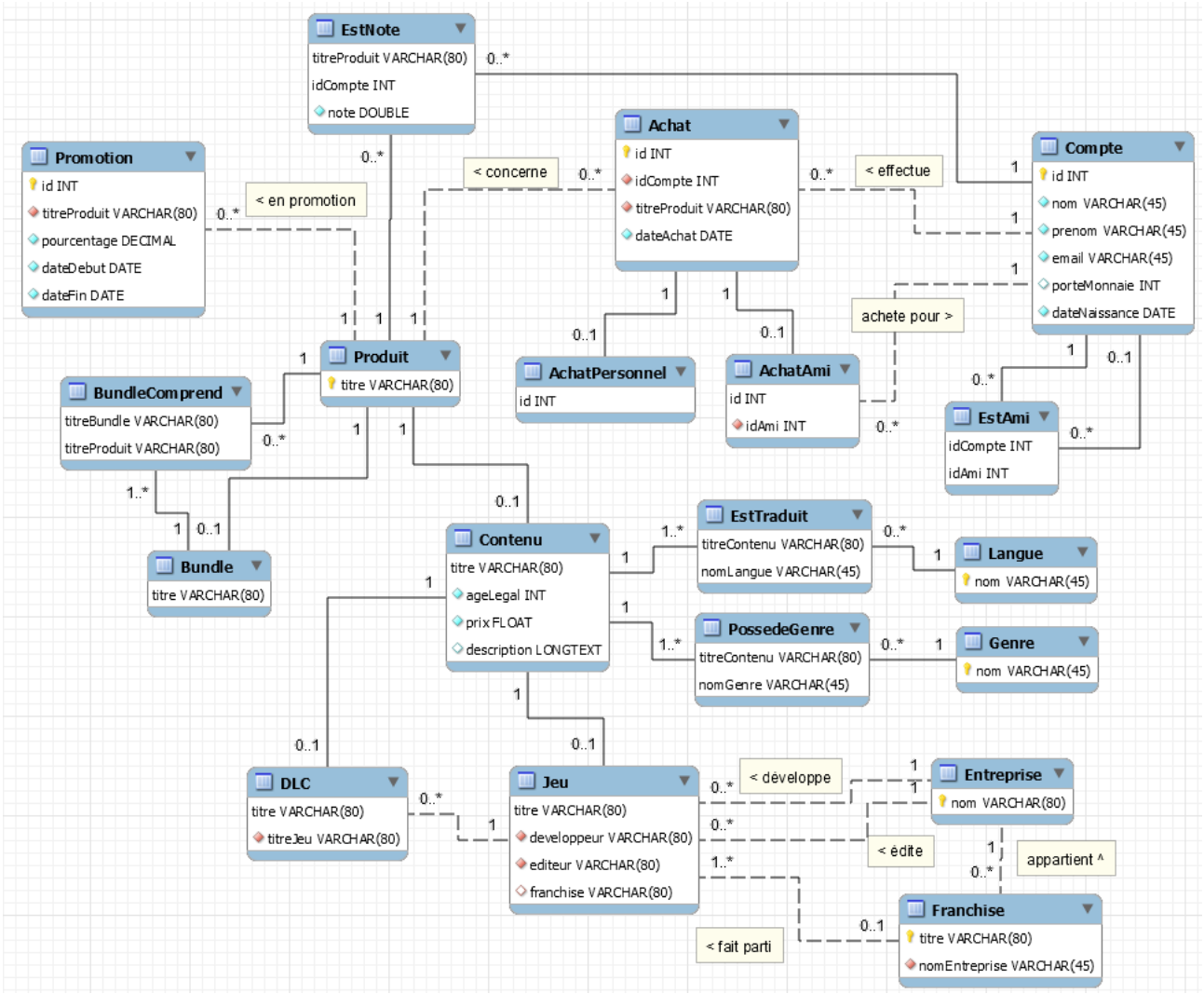
Ensuite nous avons effectués 2 associations entre jeu et entreprise la première association est pour définir un éditeur de jeu et la seconde est pour définir un développeur de jeu.

3.3 Contraintes d'intégrité

1. Un bundle ne peut pas s'inclure lui-même.
2. Un bundle ne peut contenir un produit plus d'une fois. Cela est a appliquer de manière récursive à tous les produits du bundle.
   Exemple : Un bundle contenant un bundle contenant un bundle contenant un produit A empêche le bundle racine de contenir une autre occurrence du produit A.
3. Un client ne peut pas acheter un jeu s'il n'a pas l'âge légal.
4. Un client ne peut pas s'offrir un jeu a lui-même.
5. L'âge légal d'un contenu est strictement positif.
6. Un client ne peut pas avoir une date de naissance dans le future.
7. Le solde d'un compte ne peut pas être négatif.
8. Une date d'achat ne peut pas être dans le future.
9. La note d'un jeu doit être doit être incluse dans l'intervalle [0 ;5].
10. Le prix d'un produit doit être strictement positif (>= 0).
11. La somme des promotions appliquées sur un produit doit être incluse dans l'intervalle [0 ;100]%.
12. La date de début d'une promotion est strictement plus petite que sa date de fin.
13. Un compte ne peut pas s'ajouter lui-même en ami.
14. Un produit spécifique ne peut être acheté plus d'une fois par un compte.
15. Un compte ne peut avoir plus d'une fois un même compte en ami.

4 Schema MR

Voici le schéma MR correspondant au schéma EA de la section précédente.



5 Installation

Cette procédure d'installation de base est à faire sur un système d'exploitation Windows 10, en utilisant les logiciels XAMPP, MySQL Server et Git.

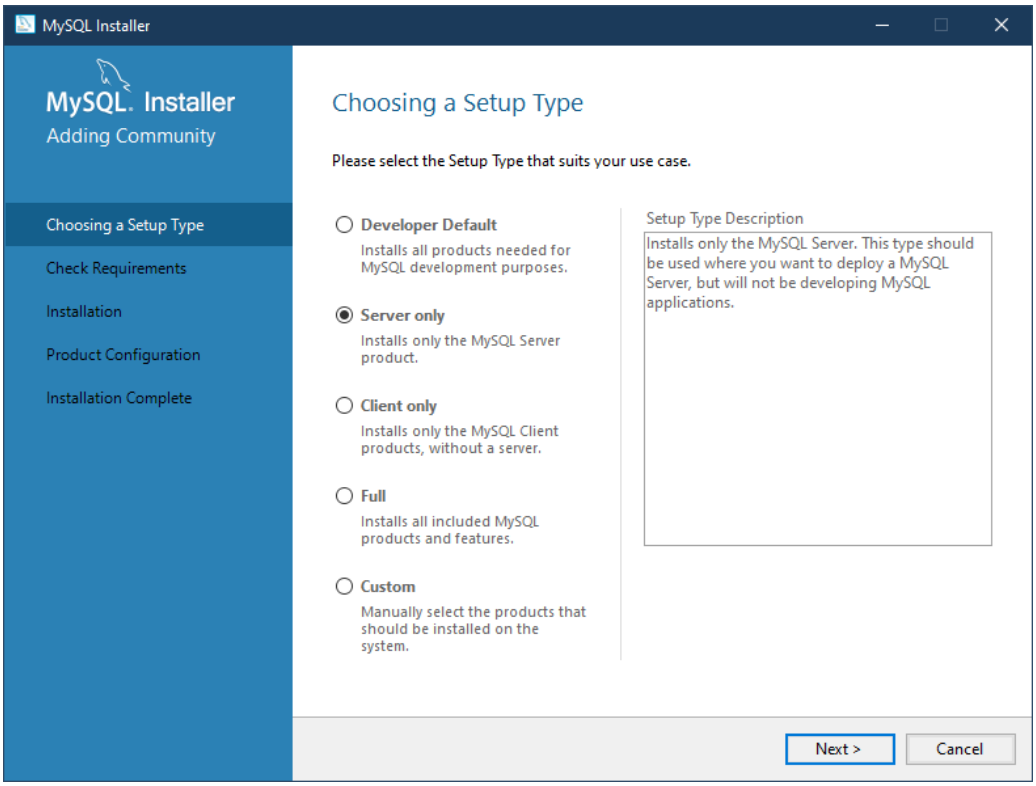
5.1 Installation de MySQL Server

Télécharger l'installeur MySQL (mysql-installer-web-community-x.msi avec x représentant la version) disponible au lien suivant :

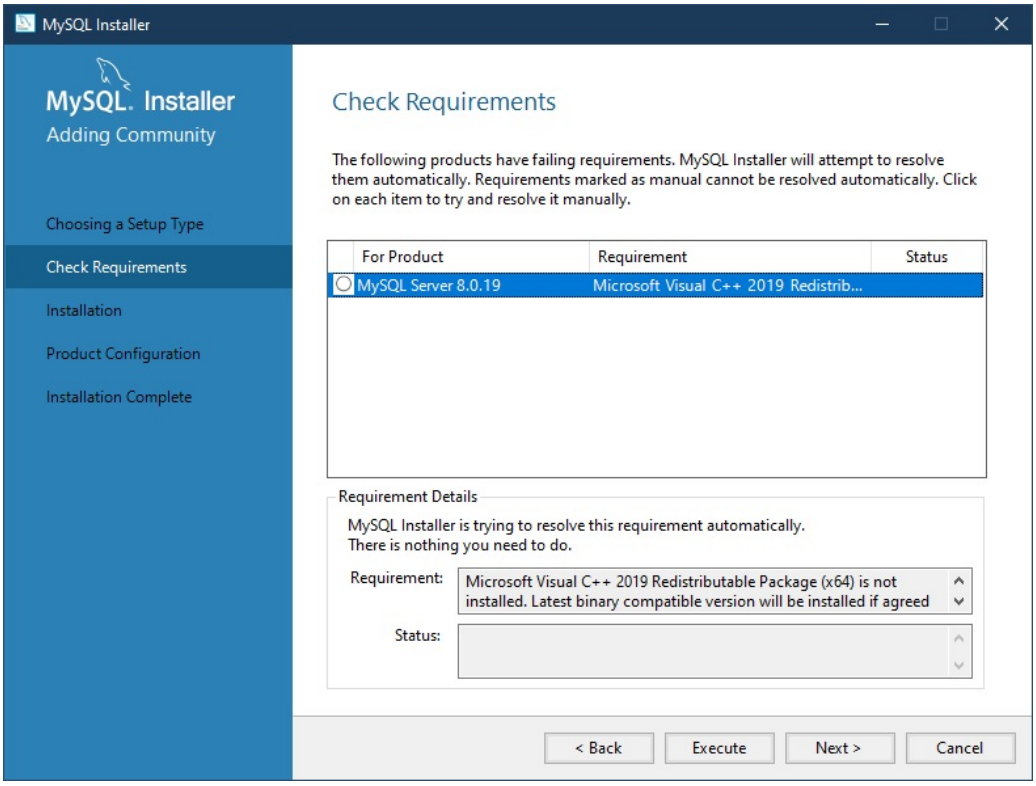
`https://dev.mysql.com/downloads/installer/`

Exécutez ensuite l'installer et suivez les captures d'écrans suivantes pour terminer l'installation.

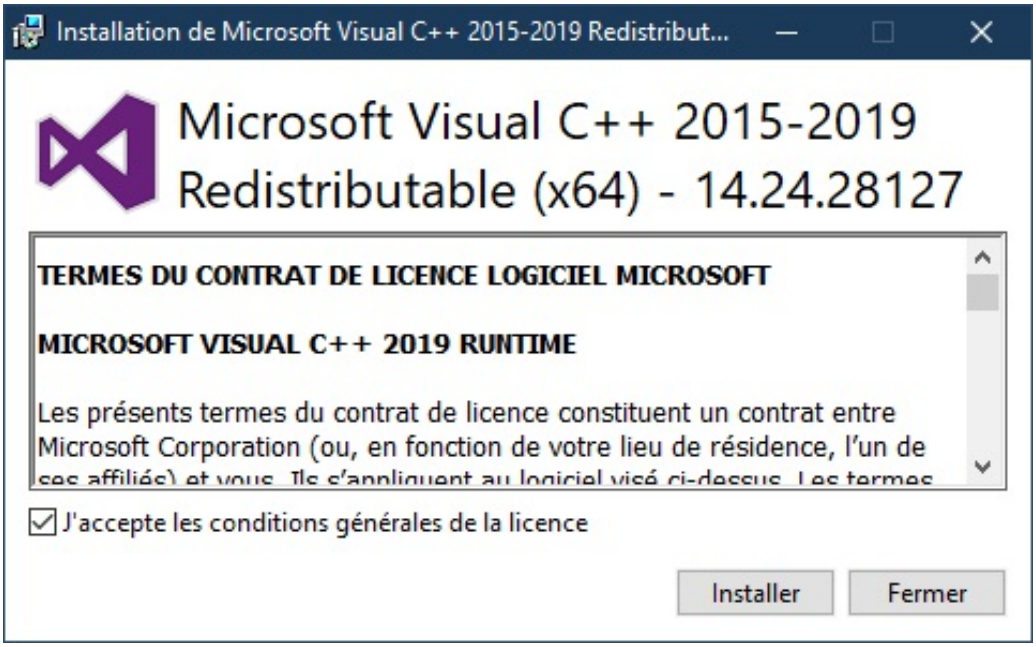
Sélectionnez "Server Only".



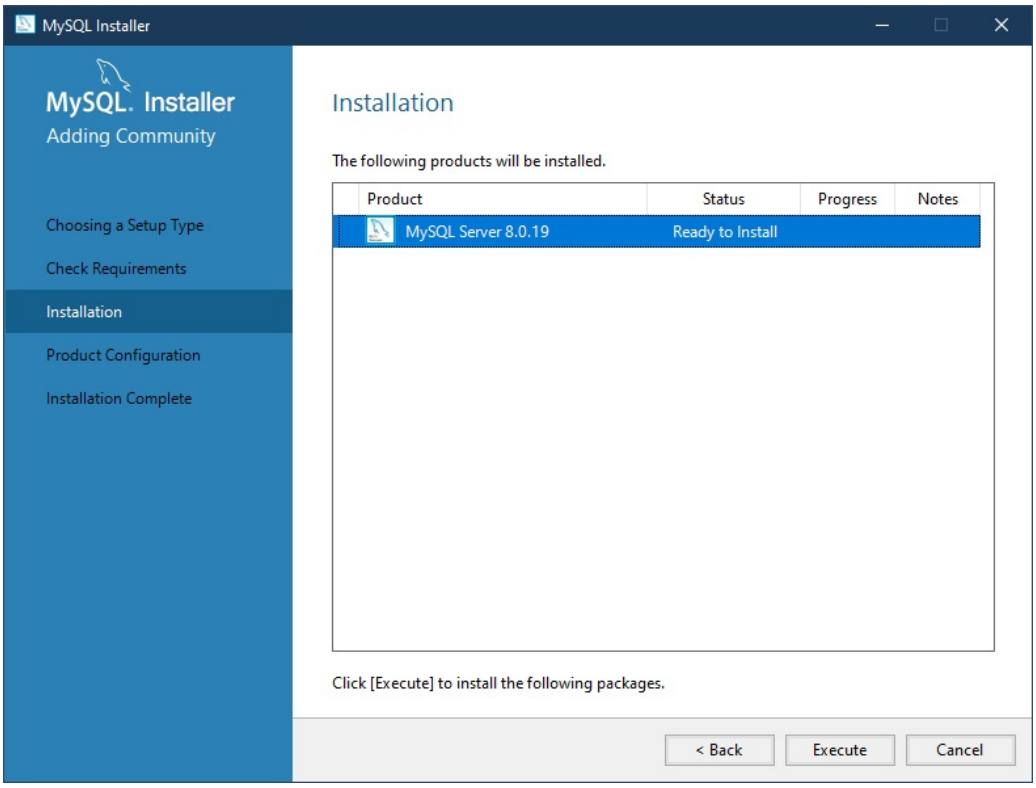
Sélectionnez "MySQL Server x" (x représentant la version) et cliquez sur "Execute".



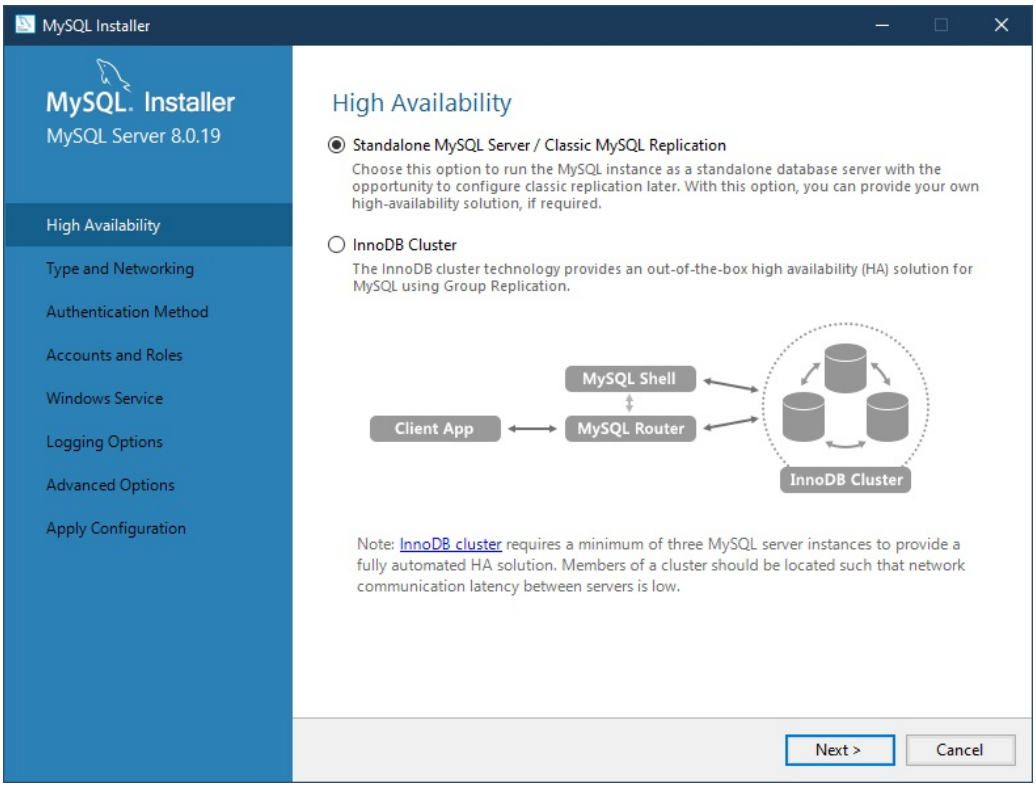
Installez la dépendance Microsoft Visual Studio.



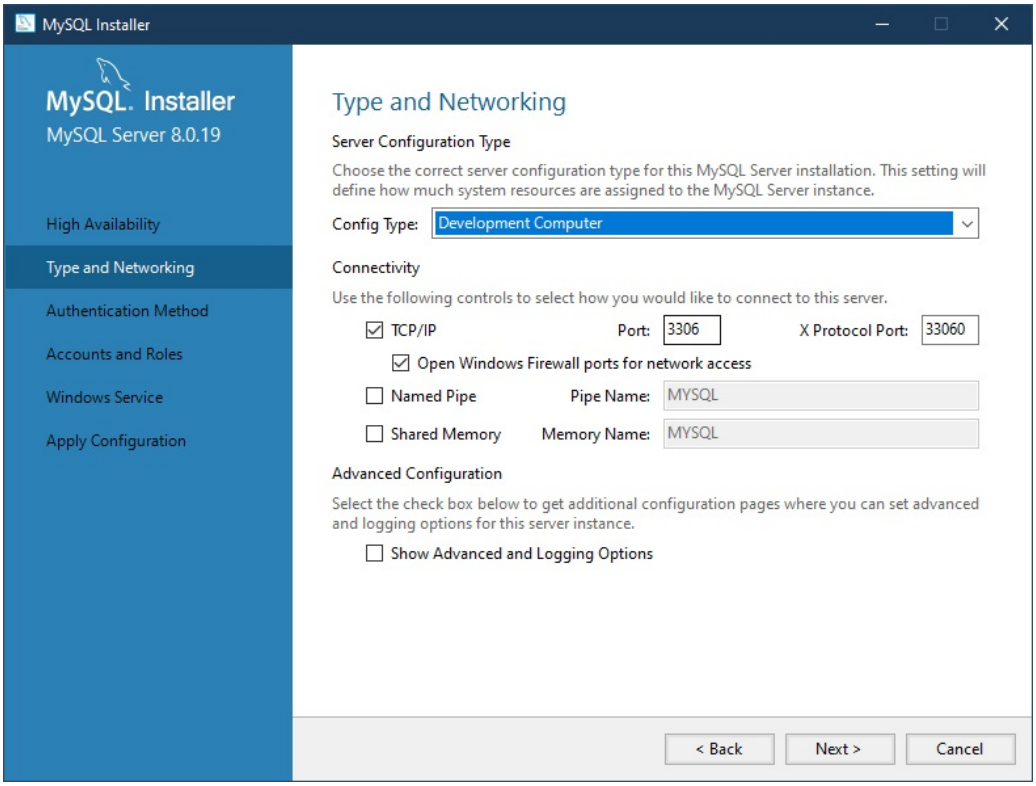
Exécutez l'installation.



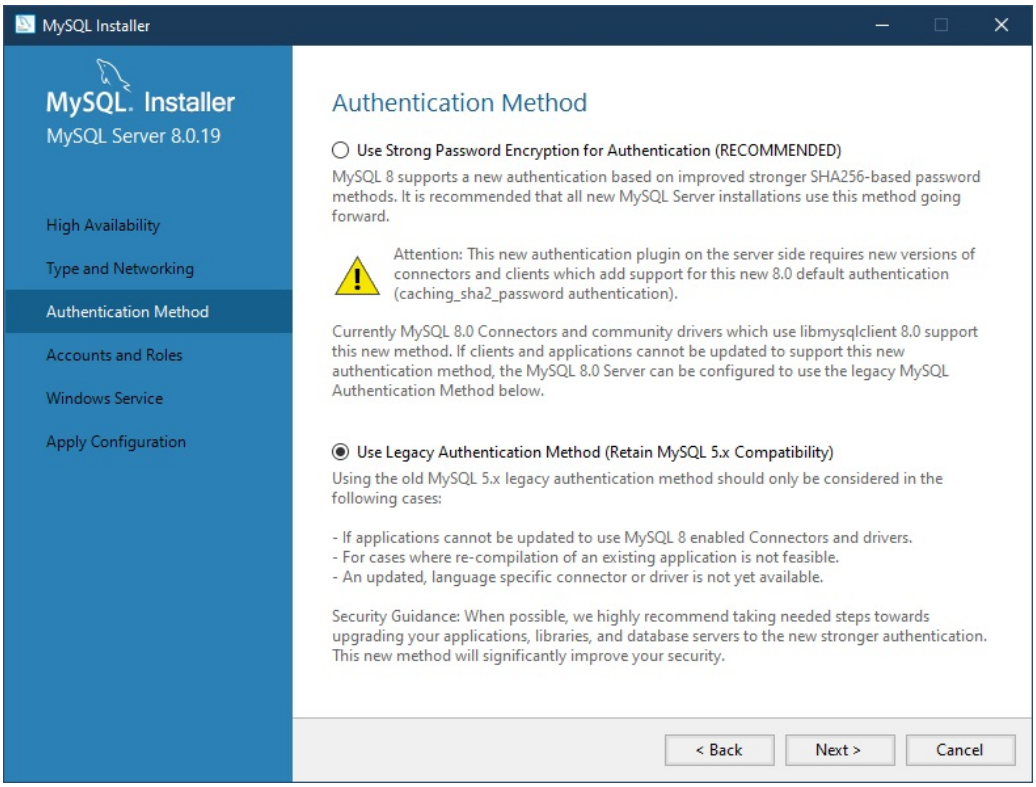
Cochez la version Standalone de MySQL Server.



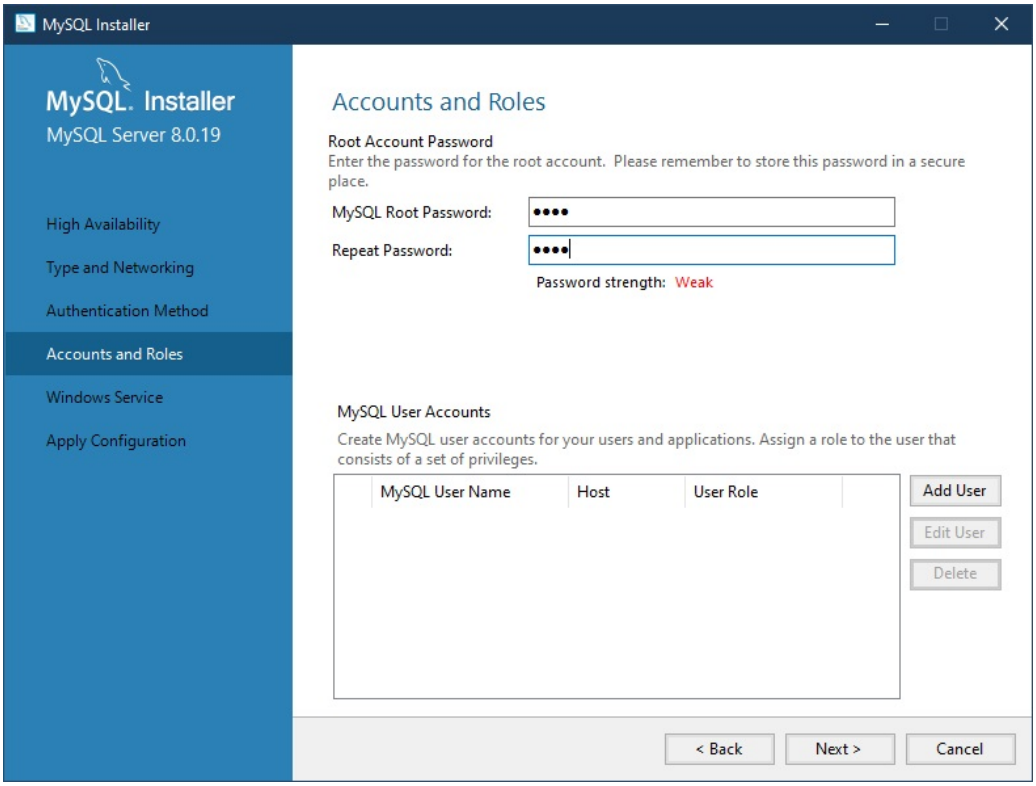
Laissez les paramètres par défauts.



Cochez "Use Legacy Authentification Method (...)". Très important car le site ne pourra pas se connecter à la base de données sans cela.



Entrez un mot de passe pour l'utilisateur root.



Terminez ensuite la configuration avec les paramètres par défauts.

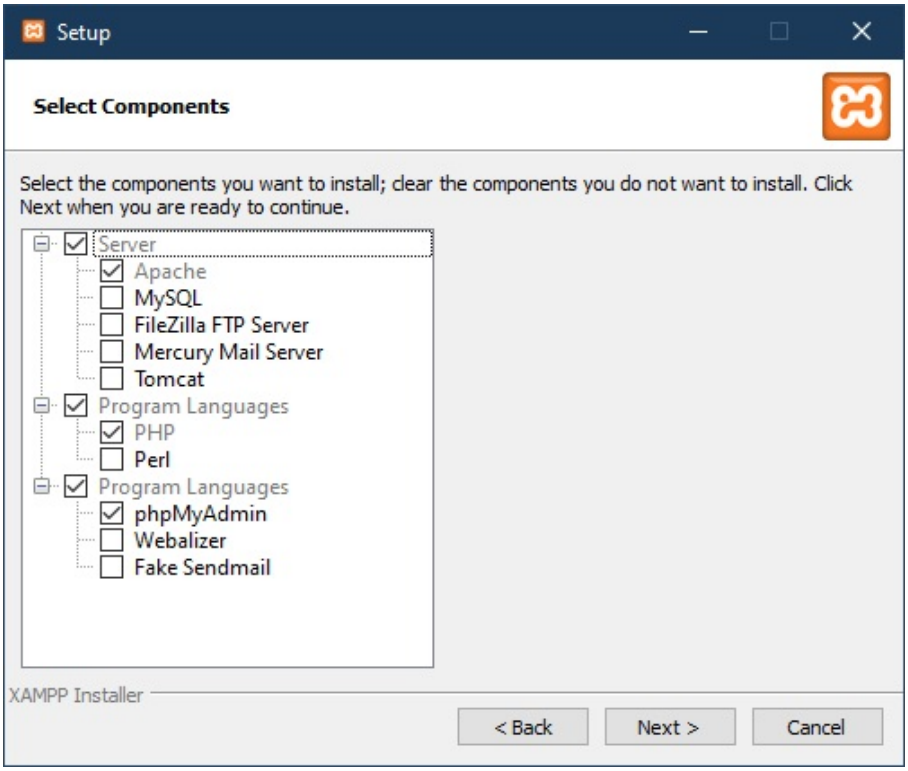
5.2 Installation de XAMPP

Commencez par télécharger l'installeur XAMPP avec la version la plus récente disponible au lien suivant :

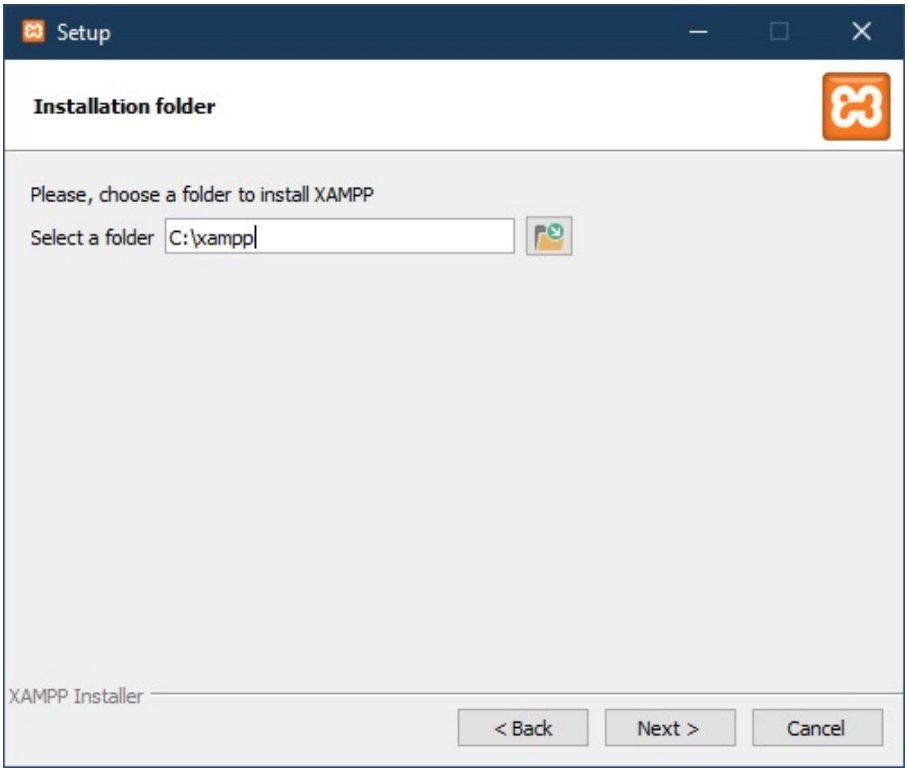
`https://www.apachefriends.org/fr/download.html`

Exécutez l'installeur et passez le message d'information informant d'éventuelles possibles restrictions sur l'application.

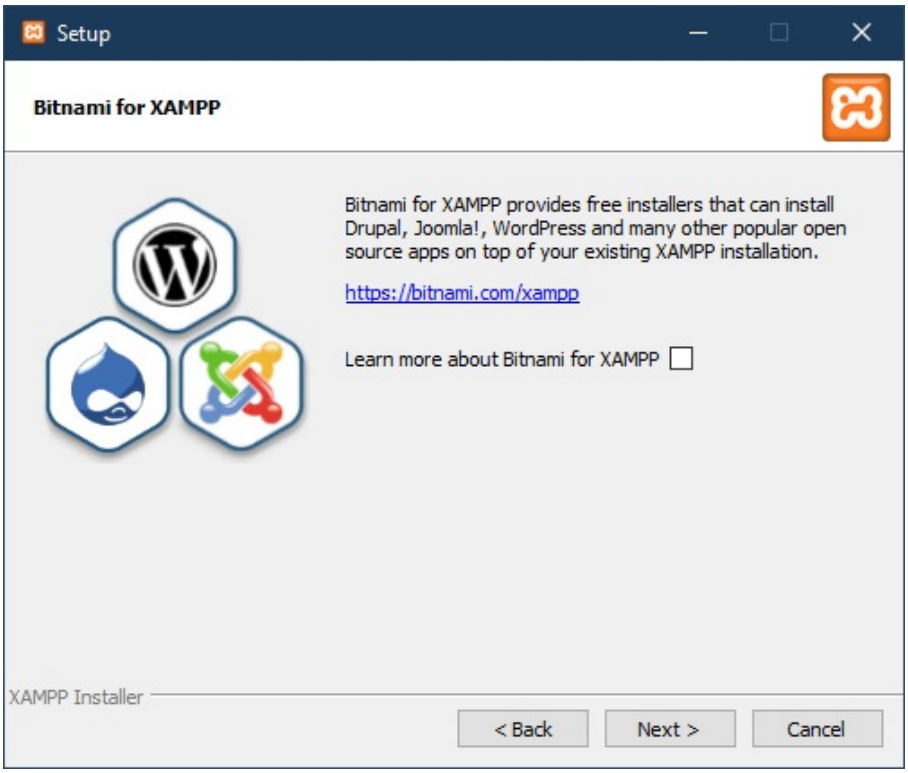
Sélectionnez les options suivantes.



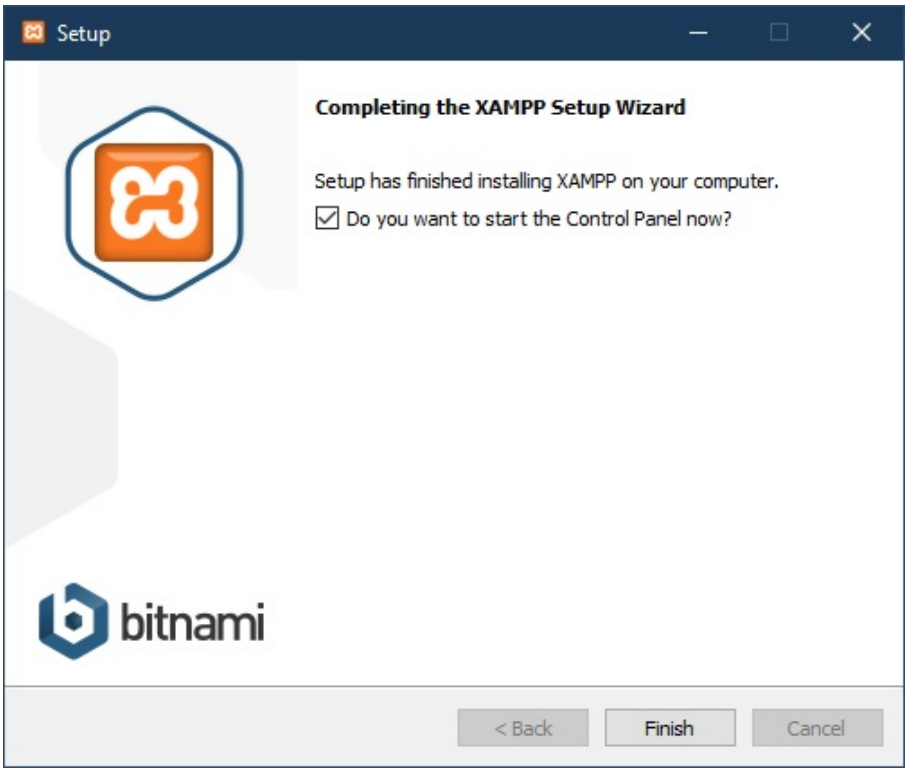
Laissez le chemin d'installation de XAMPP par défaut.



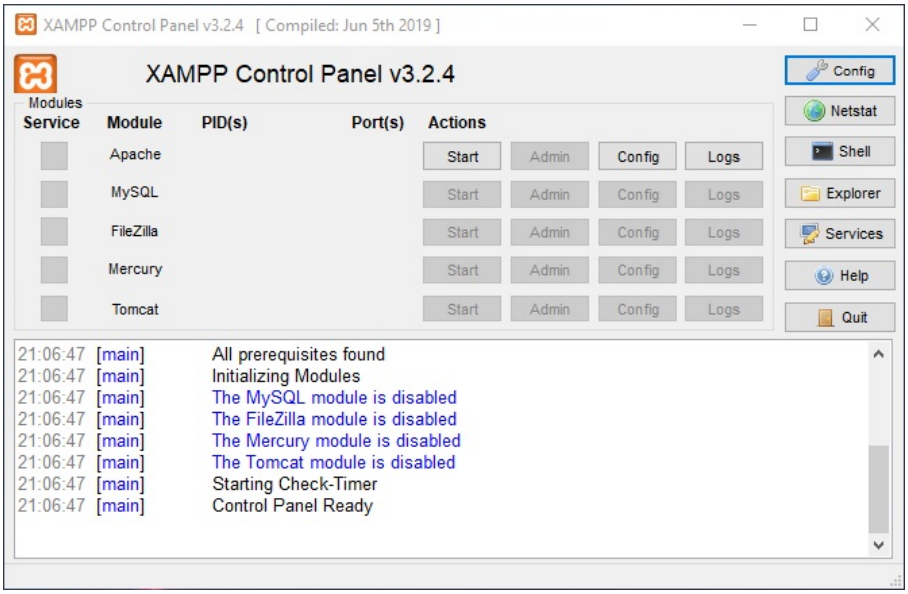
Décochez l'option de Bitnami.



Cochez "... start the Control Panel ...".



Vous pouvez à présent démarrer et éteindre le serveur apache depuis ce panneau de contrôles.



5.3 Configuration Apache et PHP

Ajoutez les lignes suivantes en fin de fichier "c :/xampp/apache/conf/extra/httpd-vhosts.conf".

```
<VirtualHost *>
 DocumentRoot "C:\XAMPP\htdocs"
 ServerName localhost
</VirtualHost>

<VirtualHost *>
 DocumentRoot "C:\XAMPP\htdocs\stome"
 ServerName localhost/stome
 <Directory "C:\xampp\htdocs\stome">
 Order allow,deny
 Allow from all
 </Directory>
</VirtualHost>
```

Modifiez la valeur de la variable "error_reporting" par la ligne suivante dans le fichier "C :/xampp/php/php.ini".

```
error_reporting=E_ALL & ~E_NOTICE & ~E_DEPRECATED & ~E_WARNING
```

Redémarrez le service apache dans le panneau de contrôles XAMPP pour appliquer les modifications.

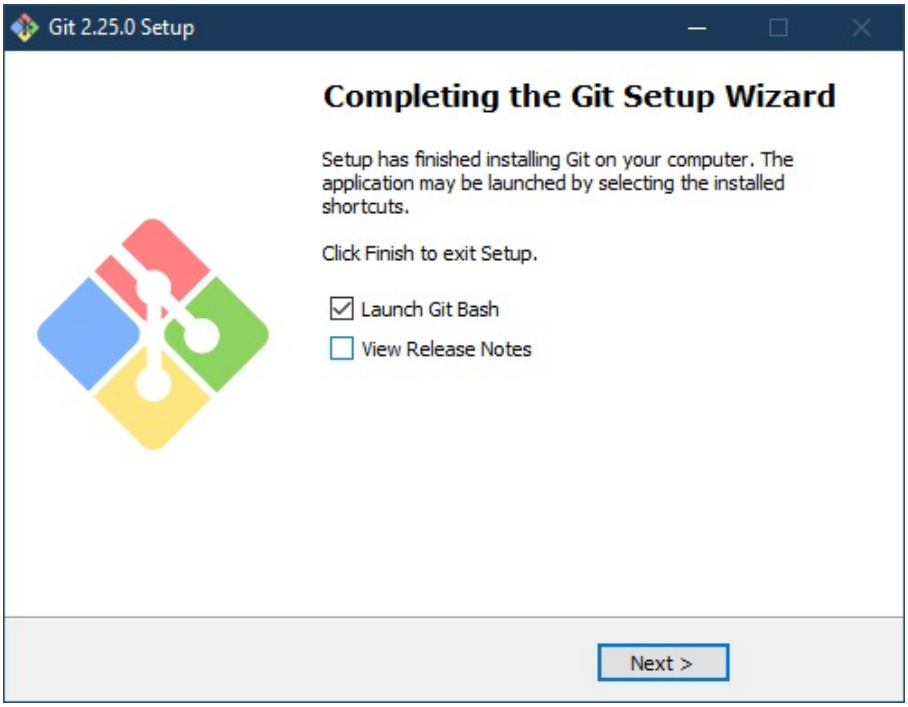
5.4 Installation de Git

Téléchargez l'installer git disponible au lien suivant :

urlhttps ://git-scm.com/downloads

Exécutez l'installation avec les paramètres par défauts

Cochez "Launch Git Bash".



5.5 Téléchargement du projet GitHub

Depuis le terminal Git, exécutez les commandes suivantes :

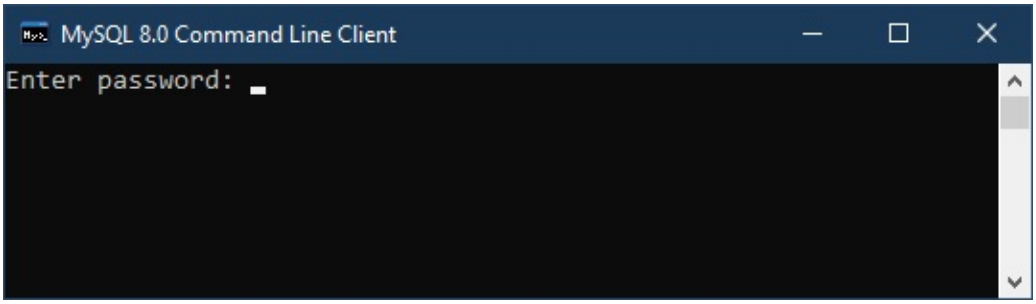
```
cd c:/xampp/htdocs
git clone https://github.com/BecaudA/BDR_mini_projet.git ./stome/
```

Vous devriez à présent avoir le dossier suivant "c :/xampp/htdocs/stome/" contenant toute l'arborescence du site web.

5.6 Créer le schéma dans MySQL Server

Exécutez le terminal de commande MySQL installé avec MySQL Server.

Entrez votre mot de passe root.



Exécutez le script SQL de création du schéma à l'aide de la commande suivante :

```
mysql> source C:\xampp\htdocs\stome\Database\Creation.sql
```

5.7 Connexion à la base de données depuis le site

Pour que le site puisse se connecter à la base de données, il faut modifier le fichier "c :/xampp/htdocs/stome/dbConfig.ini" avec le mot de passe root que vous avez utilisé.

Exemple de fichier "dbConfig.ini".

```
[database]
db_host = localhost
db_name = stome
db_user = root
db_password = <password>
```

5.8 Accès au site

`localhost/stome/`

6 Manuel Utilisateur

L'application web se décompose selon l'arborescence suivante :

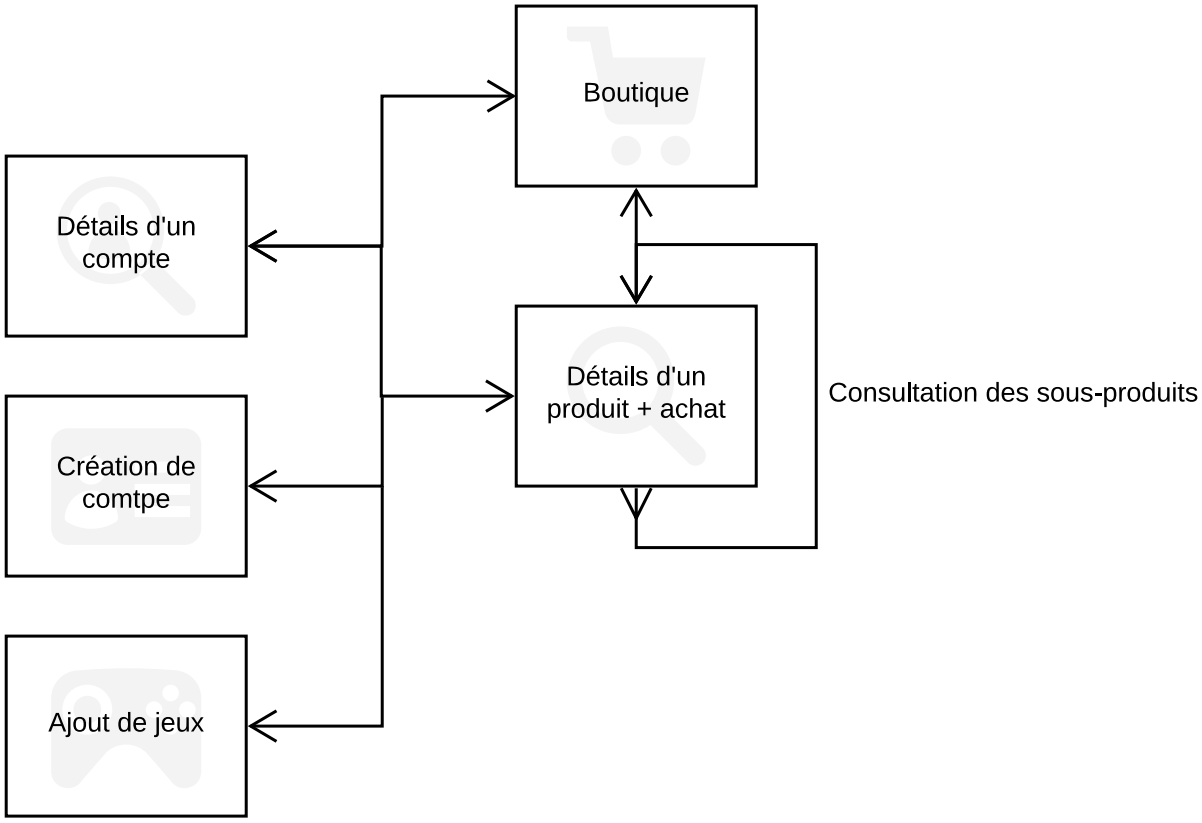


FIGURE 1 – Plan de l'application web

Il est important de noté que les pages non-fonctionnelles ne sont pas représentées sur le schéma ci-dessus.

— La boutique fait office de page d'accueil. Celle-ci permet de voir tous les produits ainsi que leur prix et promotion si il y en a. En cliquant notamment, sur le prix et le titre évidemment.
— La page de détails d'un produit permet de visualiser les informations du produit considéré. Si c'est un jeu ou DLC, les informations consultables sont les suivantes :

1. Description
2. Genres
3. Langues
4. Âge légal
5. Franchise
6. Éditeur
7. Développeur

Si le produit considéré est un bundle, alors l'ensemble des produits contenus dans celui-ci seront affichés de la même manière que sur la page Boutique. L'âge légal global du bundle est également affiché. Cette page permet aussi d'acheter le produit présenté. Pour ce faire, il suffit de cliquer sur le bouton Acheter qui fera apparaître deux listes déroulantes ainsi qu'un bouton de confirmation d'achat. Les listes permettent de choisir qui est l'acheteur et qui est le receveur du produit. Il est donc possible d'acheter et recevoir avec un même compte.

— La page de détails de compte est accessible par la barre de navigation. Celle-ci permet de consulter la bibliothèque de jeux du compte. C'est à dire l'ensemble des jeux/DLC qu'il a acheté ou reçu. Les bundles ne seront pas affiché, mais leur contenu oui (de manière récursive). L'historique d'achat est affiché et quelques statistiques liés également, notamment le total dépensé ainsi que le total économisé par les promotions. Les informations du compte suivantes sont présentes sur cette page :

1. Identifiant (numéro ID de la base de donnée)
2. Adresse email
3. Date de naissance
4. Liste des comptes amis
5. Le solde du porte-monnaie

— la page Création de compte et création de jeu permettent d'ajouter d'insérer respectivement de nouveaux comptes et jeux à la base de données.

7 Requêtes SQL, vues, triggers

Dans cette partie nous détaillerons les différentes fonctionnalité SQL que nous avons mis en place dans notre Base de donné.

7.1 Requêtes et Fonctions

Indications : Lorsque des informations sont demandées sur l'une des entités, cela fait référence aux attributs suivants :

- Compte : id, nom, prenom, email, porteMonnaie
- Achat : titreProduit, idCompte, prixProduitInitial, prixProduitFinal, date, idAmi
- Produit : titre, prix initial, prix final, pourcentagePromotion, age
- Contenu : titre, prix inital, age, prix final, franchise, developpeur, editeur, description
- Jeu : titre, langues, genres, description, developpeur, editeur, franchise
- DLC : titre, developpeur, editeur, franchise
- Bundle : titre, prix inital, prix final, age
-

Requêtes ayant été effectuées ces requêtes se trouve dans le fichier requests.php se trouvant à la racine de notre application.

1. Afficher tous les Produits.
2. Afficher un Produit.
3. Afficher les produits d'un Bundle.
4. Afficher un Contenu(Jeu/DLC)
5. Afficher les langues d'un contenu.
6. Afficher les genres d'un contenu.
7. Afficher un Bundle.
8. Afficher un Produit.
9. Afficher un Jeu.
10. Afficher tous les Comptes.
11. Afficher tous les Achats d'un Compte.
12. Afficher tous les Produits possédé par un Compte.
13. Afficher tous les Amis d'un Compte.
14. Afficher toutes les Entreprise.
15. Afficher toutes les Franchise
16. Afficher tous les Genres.
17. Afficher toutes les Langues.
18. Afficher tous les Contenus

7.1.1 Afficher tous les Produits possédés par un Compte

```
01 |  WITH RECURSIVE cte_COB(titre) AS (
02 |                  SELECT DISTINCT vPC.titreProduit FROM stome.vueProduitsComptes AS vPC
03 |                      WHERE vPC.idProprietaire = ". $idCompte . "
04 |                  UNION ALL
05 |                  SELECT BC.titreProduit
06 |                      FROM cte_COB JOIN stome.BundleComprend as BC
07 |                              ON cte_COB.titre = BC.titreBundle
08 |              ) SELECT cte_COB.titre FROM cte_COB
09 |                  INNER JOIN stome.Contenu
10 |                          ON Contenu.titre = cte_COB.titre
11 |                  GROUP BY cte_COB.titre ORDER BY cte_COB.titre ASC;
```

Cette requête est la requête permettant de trouver tous les contenus que possèdent un Compte.

Elle a été faite de manière récursive. Le premier SELECT permet de constituer notre pool de titre de produit de départ. C'est à dire que tout les produits appartenant au compte ayant l'id idCompte seront mis dans cette liste. Puis, après le mot clef UNION ALL, vient la partie récursive qui sera exécutée sur chaque titre du pool de départ. Ainsi, Seront ajouter à la liste, tous les produits inclus dans un produit déjà présent dans le pool. Cela aura pour effet de "déballer" tous les bundles de la liste de manière récursive et donc de donner l'ensemble des jeux/DLC appartenant à un compte spécifique.

7.1.2 Fonction calculPrixInitialContenusBundle

```
01 |  DELIMITER $$
02 |  CREATE FUNCTION calculPrixInitialBundle(titreB VARCHAR(80))
03 |      RETURNS INT
04 |      READS SQL DATA
05 |      DETERMINISTIC
06 |  BEGIN
07 |      DECLARE prixInitialTotalContenu INT;
08 |
09 |     WITH RECURSIVE cte_COB(titre) AS (
10 |                  SELECT DISTINCT vPC.titreProduit FROM stome.bundlecomprend AS vPC
11 |                      WHERE vPC.titreBundle = titreB
12 |                  UNION ALL
13 |                  SELECT BC.titreProduit
14 |                      FROM cte_COB JOIN stome.BundleComprend as BC
15 |                              ON cte_COB.titre = BC.titreBundle
16 |              ) SELECT SUM(vueContenu.prixInitial) INTO prixInitialTotalContenu FROM
       cte_COB
17 |                  INNER JOIN stome.vueContenu
18 |                          ON vueContenu.titre = cte_COB.titre;
19 |
20 |      RETURN prixInitialTotalContenu;
21 |  END
22 |  $$
```

La fonction présentée ci-dessus permet de trouver le prixInitial par rapport au titre d'un Bundle. Cette fonction utilise la fonction récursive présenté au-dessus mais cette fois-ci il faudra lister tout les produits d'un Bundle. De cette manière si un bundle contient un bundle la requête ira chercher le prix des jeux qui sont à l'intérieur du deuxième Bundle.

Une fonction ayant un code similaire a également était implémenté elle se nomme calculPrixReelBundle. Elle permet de trouver le prix d'un bundle si un des jeux est en promotion dans ce cas là le prix diffère du prix initial.

7.2 Triggers

Nous avons définis les triggers suivants en fonction des contraintes d'intégrité du schéma EA :

1. promotionPourcentage : permet de définir que le pourcentage actuel d'un produit ne dépasse pas 100%
2. noteProduit : permet qu'un utilisateur n'entre pas une note au-dessus de 5 dans la notation d'un produit.
3. bundleLuiMeme : Permet, lors de l'insertion d'un produit dans un bundle, que celui-ci ne puisse pas s'ajouter lui-même.
4. offreLuiMeme : interdit à un utilisateur de s'offrir un jeu à lui-même
5. verifAchat : permet de vérifier si le compte qui achète le jeu a le solde suffisant pour cela.
6. verifAchatAmi : permet de vérifier si le compte pour qui le jeu est acheté est un ami et qu'il a l'âge légal pour y jouer.
7. verifAchatPerso : permet de vérifier si le compte qui s'achète le jeu a bien l'âge pour y jouer.
8. doubleAchatProduitAmi : permet de vérifier si le compte a qui le jeu est offert n'a pas déjà acheté le jeu.

Voici l'explication des codes des triggers nous semblant les plus compliqués à comprendre :

7.2.1 verifAchat

```
01 |  DELIMITER $$
02 |  CREATE TRIGGER verifAchat
03 |      BEFORE INSERT
04 |      ON Achat
05 |      FOR EACH ROW
06 |  BEGIN
07 |      DECLARE ageCompte TINYINT;
08 |      DECLARE ageProduit TINYINT;
09 |      DECLARE porteMonnaieUser INT;
10 |
11 |      SELECT porteMonnaie INTO porteMonnaieUser
12 |      FROM Compte
13 |      WHERE id = NEW.idCompte;
14 |
15 |      IF (porteMonnaieUser - (SELECT prixFinal FROM vueProduit WHERE NEW.titreProduit =
       vueProduit.titre) < 0) THEN
16 |                  SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Porte Monnaie < 0';
17 |          ELSE
18 |                  UPDATE Compte
19 |                  SET Compte.porteMonnaie = Compte.PorteMonnaie - (SELECT prixFinal FROM
       vueProduit WHERE NEW.titreProduit = vueProduit.titre)
20 |                  WHERE NEW.idCompte = id;
21 |          END IF;
22 |  END
23 |  $$
```

Dans ce trigger, effectué sur la table Achat, nous allons tout d'abord chercher la valeur du porteMonnaie de l'utilisateur qui a effectué l'achat. Puis, si lors de la soustraction du porteMonnaie avec le prixFinal (prix avec Promotion) du produit, le porte-monnaie descends en dessous de 0 alors un message d'erreur est affiché et l'achat n'est donc pas insérer dans la table.

7.2.2 verifAchatAmi

```
01 | DELIMITER $$
02 | CREATE TRIGGER verifAchatAmi
03 |     BEFORE INSERT
04 |     ON AchatAmi
05 |     FOR EACH ROW
06 | BEGIN
07 |     DECLARE ageCompte TINYINT;
08 |     DECLARE ageProduit TINYINT;
09 |
10 |         /*Permet de trouver l'age du compte de l'ami entre*/
11 |     SELECT DISTINCT TIMESTAMPDIFF(YEAR ,Compte.dateNaissance, CURRENT_DATE()) INTO
       ageCompte
12 |     FROM Compte
13 |     INNER JOIN EstAmi
14 |                 ON EstAmi.idAmi = NEW.idAmi
15 |     WHERE Compte.id = NEW.idAmi;
16 |
17 |     /*Si l'age est nul cela veut dire qu'aucun ami n'a ete trouve donc le compte entre
       ne fait pas parti des amis*/
18 |     IF(ageCompte IS NULL) THEN
19 |      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Pas ami';
20 |         END IF;
21 |
22 |     /*Trouve l'age du produit grace    la vue sur les Produits*/
23 |     SELECT vueProduit.age INTO ageProduit
24 |     FROM vueProduit
25 |         INNER JOIN Achat
26 |               ON vueProduit.titre = Achat.titreProduit
27 |     WHERE Achat.id = NEW.id;
28 |
29 |     /*Verification que l'age du compte Ami est inferieur    l'age du produit achete*/
30 |     IF (ageCompte < ageProduit) THEN
31 |         SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Compte trop jeune';
32 |         END IF;
33 | END
34 | $$
```

Dans un premier temps, nous allons chercher l'âge du compte de l'ami à qui le produit est offert pour ensuite le stocker dans une variable. Si cette requête ne renvoie aucun résultat cela veut dire que la personne a choisi d'offrir un produit à un compte qui n'était pas son ami dans ce cas le tuple concernant cet achat pour un ami est alors refusé et il n'est donc pas ajouté à la table achatAmi.

Nous avions pensé à effectuer une procédure avec à l'intérieur une transaction mais malheureusement nous n'avons pas réussi à faire fonctionner cela et la meilleure solution que nous avons trouvé est celle proposée ci-dessus.

Nous avons également défini des triggers lors de l'insertion d'élément dans une table fille. Ces triggers vérifient que l'élément ajouté dans celle-ci soit présent dans la table parent et qu'il ne soit pas déjà ajouté dans une autre table fille.

1. TRG_BundleProduit : Permet de vérifier l'héritage entre Bundle et Produit
2. TRG_ContenuProduit : Permet de vérifier l'héritage entre Contenu et Produit
3. TRG_JeuContenu : Permet de vérifier l'héritage entre Jeu et Contenu
4. TRG_DLCContenu : Permet de vérifier l'héritage entre DLC et Contenu
5. TRG_AchatPerso : Permet de vérifier l'héritage entre AchatPersonnel et Achat
6. TRG_AchatAmi : Permet de vérifier l'héritage entre AchatAmi et Achat

Voici le code d'un trigger : Le code est le même pour tous les triggers à l'exception des tables qui sont modifiées.

7.2.3 TRG_BundleProduit

```
01 | /*Trigger pour l'heritage Produit->Bundle,Contenu*/
02 | DELIMITER $$
03 | /*  Trigger sur bundle pour controler l'insertion  */
04 | CREATE TRIGGER TRG_BundleProduit
05 |     BEFORE INSERT
06 |     ON Bundle
07 |     FOR EACH ROW
08 | BEGIN
09 |     DECLARE erreur TINYINT;
10 |
11 |     SET @erreur = 0;
12 |
13 |     /*  La clef de "Produit" doit exister pour la creation de "Bundle"  */
14 |     IF NEW.titre NOT IN (SELECT *
15 |                          FROM  Produit P1
16 |                          WHERE P1.titre = NEW.titre)
17 |     THEN
18 |         SET @erreur = 1;
19 |     END IF;
20 |
21 |     /* l'identifiant de bundle ne doit pas etre utilise par les autres tables filles */
22 |     IF NEW.titre IN (SELECT Contenu.titre
23 |                      FROM Contenu
24 |                      WHERE NEW.titre = Contenu.titre)
25 |     THEN
26 |         SET @erreur = 1;
27 |     END IF;
28 |
29 |     IF @erreur = 1 THEN
30 |         SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Insertion de Bundle Impossible';
31 |     END IF;
32 |
33 | END
34 | $$
```

Par manque de temps, nous n'avons pas pu mettre en place la vérification dans la table parent pour que l'objet créé soit redéfini dans la table fille, mais si nous avions eu plus de temps nous aurions créé un événement qui se lancerai toutes les heures pour "nettoyer" les tables parentes d'élément qui ne serait pas défini.

Nous n'avons également pas pu respecter toutes les contraintes d'intégrités prévus initialement par manque de temps.

Pour voir la structure des autres triggers, se référer au fichier creation.sql.

7.3 Vues

Une partie importante de notre application est basée sur l'utilisation de vue pour regrouper un grand nombre d'informations. En regroupant des informations, nous avons pu gérer de manière plus aisé les données à afficher sur les pages de notre application.

Les vues que nous avons créées sont les suivantes :

1. vueBundle : permet de connaître le titre, le prix initial, le prix final(avec Promotion) et l'âge d'un bundle. Les champs cités ci-dessus, sauf pour le titre, ne sont pas insérés directement dans la base, nous avons donc dû les calculer.
2. vueDLC : permet de manipuler de manière plus simple les informations de la table DLC.
3. vueContenu : permet de connaître tous les Contenus de la base de données. Pour qu'un contenu soit affiché dans cette vue, il faut qu'il soit défini comme un Jeu soit comme un DLC. Les informations fournies par la vue sont les suivants : titre, prixInitial, age, prixFinal, promotion, franchise, developpeur, editeur et la description. Grâce à cette vue il sera alors faciles d'afficher les champs désirés dans la page de présentation d'un Contenu.
4. vueProduit : permet de définir les informations que nous désirons afficher dans notre Boutique.
5. vueAchats : permet de voir tous les achats liés à un compte ainsi que de savoir si celui-ci a offert un jeu à un ami. Cette vue sera utilisée pour l'historique des achats d'un compte.
6. vueProduitsComptes : Permet de voir tous les produits que des comptes possède soit grâce à un achat soit parce qu'il leur a été offert.

Voici quelques vues d'exemples avec des explications pour les éclaircir. Pour voir toutes les vues créer vous pouvez accéder au fichier Creation.sql

7.3.1 vueBundle

```
01 |  DELIMITER $$
02 |  CREATE VIEW vueBundle(titre, prixInitial, prixReel, age) AS
03 |  SELECT titreBundle, calculPrixInitialContenusBundle(titreBundle) AS prixInitial,
04 |         calculPrixReelBundle(titreBundle) AS prixReel,
05 |         MAX(Contenu.ageLegal) AS age
06 |  FROM BundleComprend
07 |           INNER JOIN Produit
08 |                      ON Produit.titre = titreProduit
09 |           INNER JOIN Contenu
10 |                      ON Contenu.titre = Produit.titre
11 |  GROUP BY BundleComprend.titreBundle;
12 |  $$
```

La vue ci-dessus représente la création de la vue pour les bundles. Nous utilisons 2 fonctions calculPrixInitial-ContenusBundle et calculPrixReelBundle qui permettent de connaître le prix sans ou avec promotion. L'âge est obtenu en faisant un MAX de l'âge entre les produits d'un Bundle. Un bundle prendra alors comme âge le produit ayant l'âge légal le plus élevé.

7.3.2 vueProduit

```
01 | DELIMITER $$
02 | CREATE VIEW vueProduit(titre, prixInitial, age, prixFinal, promotion) AS
03 |
04 | SELECT DISTINCT Produit.titre, vueBundle.prixInitial AS prixInitial, vueBundle.age AS
       age, calculPrixPromo(Produit.titre, vueBundle.prixReel) AS prixFinal,
05 |                 COALESCE((SELECT SUM(Promotion.pourcentage)
06 |                           FROM Promotion
07 |                           WHERE Promotion.titreProduit = Produit.titre AND
08 |                               CURRENT_TIMESTAMP() BETWEEN Promotion.dateDebut AND
       Promotion.dateFin), 0) AS pourcentagePromo
09 | FROM Produit
10 |          INNER JOIN vueBundle
11 |                     ON Produit.titre = vueBundle.titre
12 |          LEFT JOIN Promotion
13 |                     ON Promotion.titreProduit = Produit.titre
14 | UNION
15 | SELECT DISTINCT Produit.titre, vueContenu.prixInitial AS prixInitial, vueContenu.age AS
        age, vueContenu.prixFinal AS prixFinal,
16 |                 COALESCE((SELECT SUM(Promotion.pourcentage)
17 |                           FROM Promotion
18 |                           WHERE Promotion.titreProduit = Produit.titre AND
19 |                               CURRENT_TIMESTAMP() BETWEEN Promotion.dateDebut AND
       Promotion.dateFin), 0) AS pourcentagePromo
20 | FROM Produit
21 |          INNER JOIN vueContenu
22 |                     ON Produit.titre = vueContenu.titre
23 |          LEFT JOIN Promotion
24 |                     ON Promotion.titreProduit = Produit.titre;
25 | $$
```

Cette vue utilise un UNION pour pouvoir afficher d'un côté tous les Contenu et d'un autre côté tous les Bundles. Le fait d'utiliser cette approche nous permet alors de ne pas inclure les produits qui ne sont définis que dans Produit. Car les tuples qui ne sont que dans Produits ne sont pas encore considérés comme des produits pouvant être vendus. De plus, si nous avions mis en place l'évènement permettant de "nettoyer" la table ces tuples seraient effacés au bout d'un certain temps.

Une fonction se nommant calculPrixPromo est également utilisée pour calculer le prix final d'un produit. En passant le titre d'un produit ainsi que son prix initial, la fonction pourra alors effectuer la promotion sur le prix. Elle effectuera une promotion sur le prix par rapport à la table des Promotions.

Pour la promotion sur un bundle, le prix réel du bundle est utilisé car un bundle pourrait avoir un produit qui a déjà une promotion et dans ce cas le prix de base (somme des prix de tous les produits) du Bundle ne correspond pas au prix final.

L'UNION a également été utilisé pour les vues vueAchat et vueContenu car comme expliqué auparavant, seuls les tuples étant dans une des tables enfants et la table parent nous intéressent. Tout tuples ne se situant que dans la table parente ne doit pas être affiché dans la vue car cela voudrait dire que le tuple n'est pas complètement inséré dans la base de donnée.

8 Bugs connus

L'application n'étant pas entièrement terminée, ces fonctionnalités ne peuvent correspondre au cahier des charges dans sa globalité.

Les bugs suivants ont été relevé sous certaines configurations des serveur Apache et MySQL.

1. La page d'ajout de nouveaux jeux n'est pas fonctionnelle.
2. La page de création de nouveaux comptes n'est pas fonctionnelle.
3. L'achat des produits pour soi ou pour un ami n'est pas fonctionnel.

Seul le bug 3. est présent avec la configuration fournit dans le chapitre d'installation lorsque certaines conditions sont présentes.

Conditions :

— L'achat d'un jeu ne fonctionne pas si le compte acheteur à déjà acheté le jeu pour soi ou pour un ami.
— L'achat d'un jeu ne fonctionne pas si le receveur possède un porte monnaie à un solde inférieur au prix du produit. Lors de l'affichage des achats le prix affiché est le prix actuel du jeu sur le site au lieu que se soit le prix auquel le jeu a été acheté.

9 Conclusions

Ce projet nous a permis de mettre en pratique la création d'une base de donnée du début à la fin. Ainsi, nous avons pu réaliser l'ampleur d'un tel travail. En effet, il est relativement aisé de créer une base de donnée fonctionnelle mais il est bien plus ardu de la finaliser, c'est-à-dire concrétiser les moindre détails de la conception de départ, notamment mettre en oeuvre toutes les contraintes d'intégrité.

De plus, il faut ensuite choisir la manière dont les données vont être utilisées, montrées, modifiées ce qui représente une nouvelle problématique. Nous avons donc pris cette occasion pour apprendre de nouveaux langages (HTML, CSS et PHP) ainsi que l'architecture logicielle MVC [1]. Ces technologies furent difficiles à mettre en place et à utiliser pour nous pendant le temps mis à disposition cependant, l'ensemble de ces difficultés nous ont permis d'améliorer notre gestion d'un projet d'envergure plus élevé que nous n'en avons l'habitude.

Nous sommes content du résultat obtenu mais tout de même frustrés de ne pas pouvoir le rendre entièrement achevés. Nous voyons encore tout le potentiel d'amélioration de notre application et espérons faire encore mieux lors de nos projets futurs.

1. Modèle, Vue, Contrôleur