Rapport de projet SGBD : Bandes dessinées

Auteurs : *David Bitonneau, Ludovic Hofer, Benoît Ruelle*
Encadrants : *Mme. Allyx Fontaine, M. Sylvain Lombardy, M. Mohamed Mosbah*

Deuxième année, filière informatique
Date : 17 décembre 2012

1 Introduction

L'objectif du projet est de mettre en œuvre, sur un cas pratique, les notions et les méthodes vues dans le module de SGBD. L'application réalisée ici est liée à la gestion de bandes dessinées.

Dans ce rapport, la base de données est décrite de sa conception à son utilisation, en passant par son implémentation.

2 Modélisation des données

2.1 Description du contexte de l'application

L'application doit permettre la gestion de bandes dessinées à partir des informations suivantes :

- Chaque volume de bande dessinée est soit un album, soit une revue.
- Tout volume a un éditeur, et une année d'édition.
- Un album peut éventuellement appartenir à une collection, et dans ce cas, il peut avoir un numéro dans cette collection. Deux albums de la même collection ont forcément le même éditeur.
- Un volume a un titre, qui est soit le titre de l'album, soit celui de la revue ; dans le cas de la revue, elle a aussi un numéro.
- Un volume peut contenir plusieurs histoires.
- Chaque histoire a un titre et une année de (première) parution ; elle a un ou plusieurs auteurs, chacun de ces auteurs s'occupant du dessin ou du scénario (ou des deux).
- Une même histoire peut apparaître dans différents volumes ; on veut pouvoir annoter la présence d'une histoire dans un volume ("Première publication", "Pages 15 à 22", "Version longue", etc.).
- Une histoire peut appartenir à une série et dans ce cas elle peut avoir un numéro de série.

La lecture de ces informations fait ressortir les entités suivantes :

- volume (regroupe les attributs communs des albums et des revues) ;
- album avec collection ;
- album sans collection ;
- revue ;
- éditeur ;
- collection ;
- histoire ;
- série d'histoires ;
- rôle d'un auteur ;
- auteur.

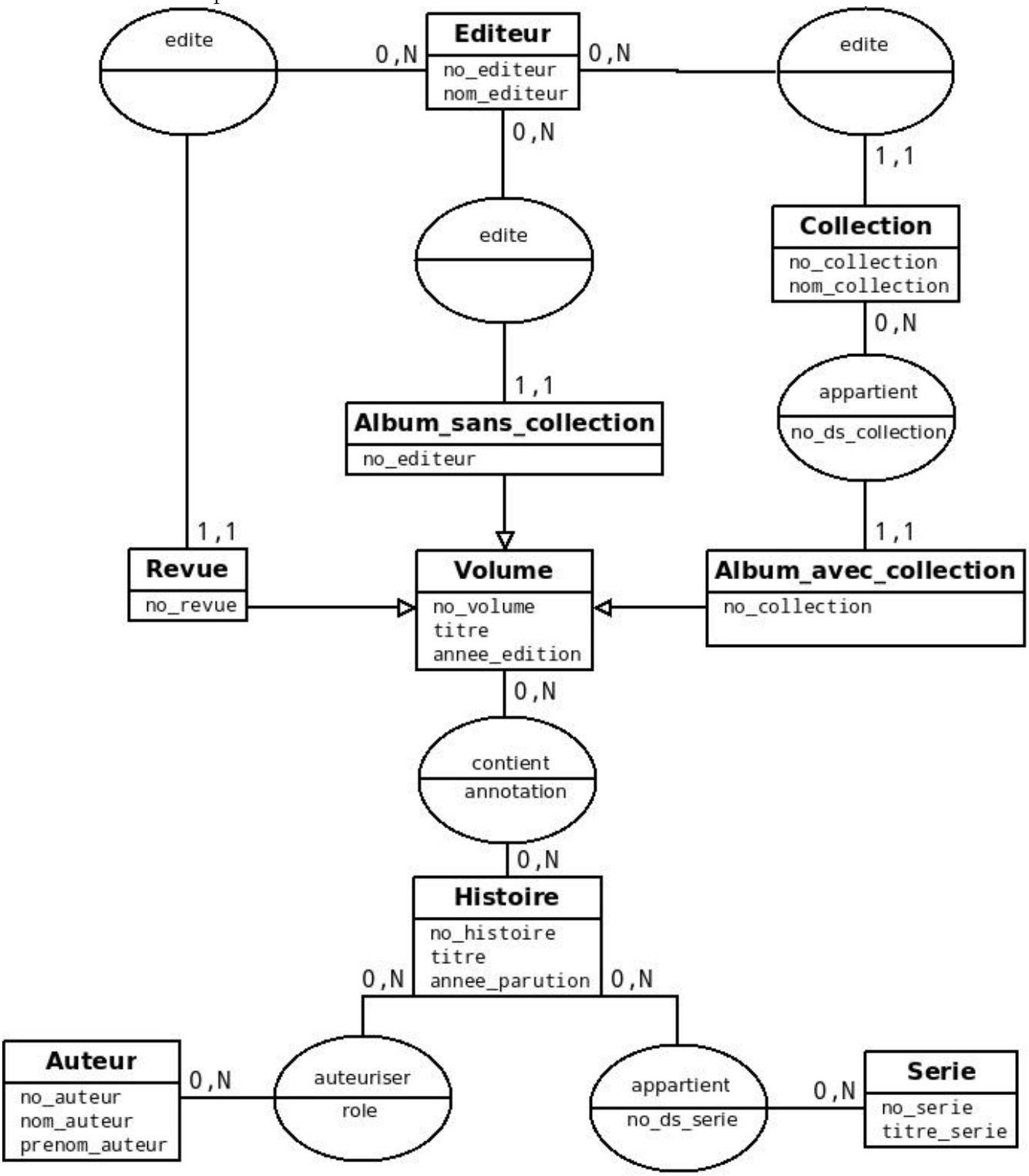
Les hypothèses suivantes sont effectuées :

- On considère qu'une revue est un ensemble de numéros de la revue et que l'éditeur d'une revue peut varier dans le temps.
- Plusieurs collections différentes peuvent avoir le même nom.
- On autorise que des histoires n'aient pas d'auteur renseigné ou que des volumes ou des collections ne soient pas associés à un éditeur présent dans la base. Inversement, des auteurs

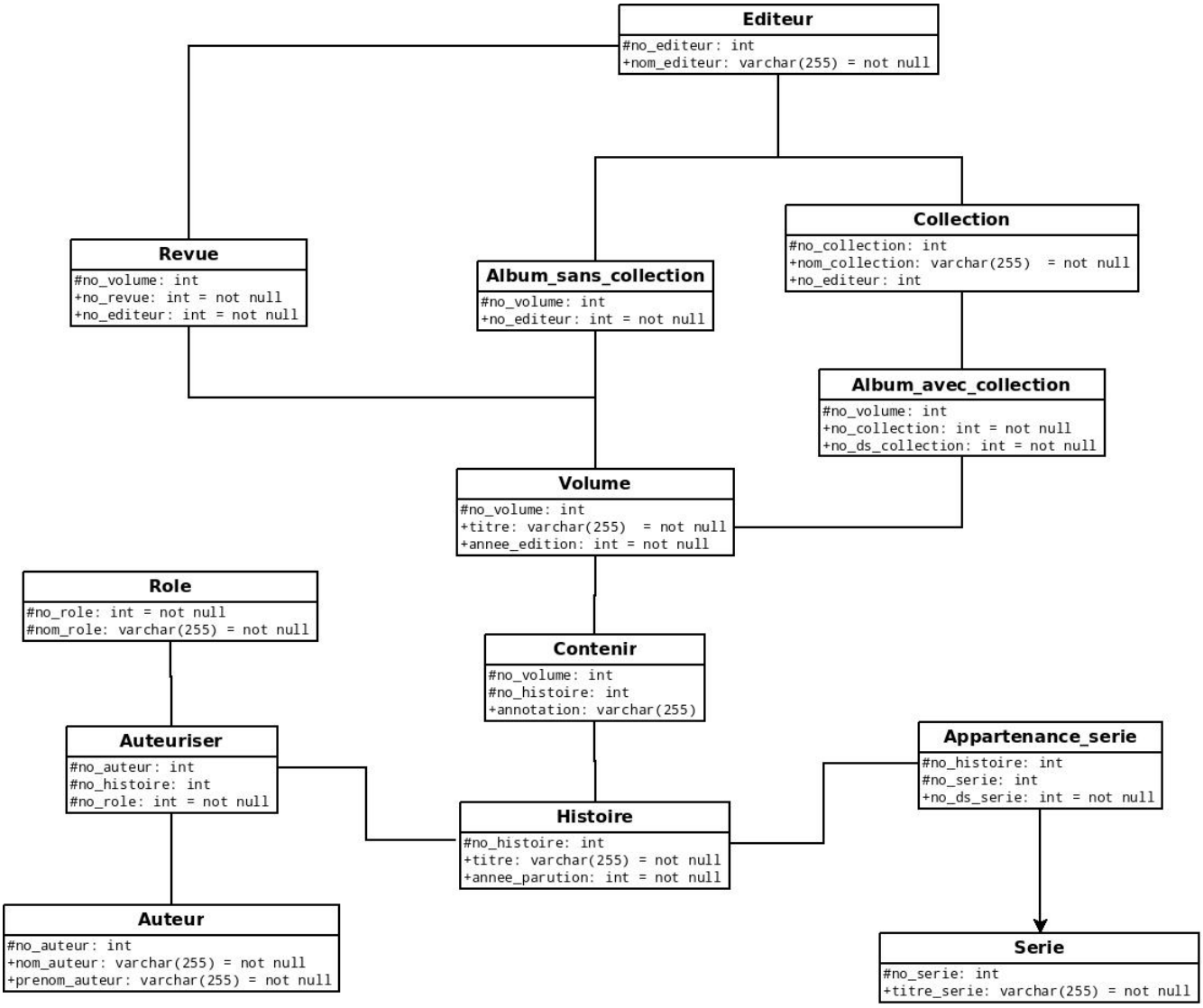
ou des éditeurs peuvent être présents dans la base sans être liés à une histoire, un volume ou une collection.

3 Schéma conceptuel

Le schéma conceptuel suivant montre les relations entre ces différentes entités :



4 Schéma relationnel



Nous avons utilisé les méthodes vues en cours pour le passage du schéma conceptuel au schéma relationnel. Nous avons été forcé de rajouter des tables de manière à assurer la troisième forme normale de la base. Nous avons préféré utiliser des relations 0,N plutôt que 1,N pour faciliter l'insertion d'entrées dans la base, ainsi il est possible d'ajouter des histoires sans volume les contenant par exemple.

5 Implémentation

La base a été implémentée en utilisant MySQL. Ce SGBD a été choisi pour sa simplicité d'installation et de configuration.

5.1 Contraintes d'intégrité, dépendances fonctionnelles

Des contraintes d'intégrité sont réalisées au niveau de la plupart des clés étrangères pour trois situations :

– Réaliser des suppressions en cascade d'entrées. C'est le cas par exemple de la suppression automatique de relations 'contenir' lorsqu'un album ou une histoire est supprimée ou de relations 'auteuriser' dans le cas de la suppression d'auteurs ou d'histoires. /bin/bash : e : command not found ou d'un album pour retirer automatiquement de l'entrée volume associée.
– Mettre des valeurs par défaut (null) dans les attributs concernant les éditeurs (album sans collection, collection, revue) lorsqu'un éditeur est supprimé.
– Bloquer la suppression d'une collection s'il y a encore des albums dans cette collection.

Pour alléger le code de l'interface et limiter la duplication, des vues ont été créées. Elles permettent par exemple d'associer des histoires à leurs auteurs en une requête simple sur laquelle il suffit d'appliquer une condition supplémentaire pour isoler un auteur particulier. La même chose est faite pour associer des albums à leur collection ou leur éditeur, des volumes à leurs histoires, etc.

En termes de performances, cette approche n'a pas d'effet négatif car les vues dans MySQL sont par défaut utilisées comme des alias et ne sont pas précompilées (algorithme 'merge'). Cela signifie que l'optimiseur aura l'occasion de réaliser la sélection dès le début de la requête et n'aura pas à construire la table regroupant toutes les associations avant de faire la sélection.

6 Opérations possibles sur la base

Différentes opérations peuvent être effectuées sur la base de données. Ces opérations peuvent être classées en 3 différentes catégories :

– les opérations de consultation ;
– l'extraction de statistiques ;
– les mises à jour de la base.

Consultation Ces opérations permettent de fournir des informations basiques concernant diverses entités de la base de données à partir des informations qui y sont stockées. Les informations suivantes sont notamment fournies :

– la bibliographie d'un auteur, soit par ordre chronologique, soit classé par séries, en tant que scénariste, dessinateur ou auteur complet, en indiquant ses co-auteurs et leurs rôles.
– la liste des auteurs collaborant à une revue durant une certaine période, avec le nombre de numéros auxquels ils ont participé.
– l'historique de la publication d'une histoire.
– les histoires différentes ayant le même titre.

L'implémentation des requêtes permettant d'obtenir ces informations a été effectué en deux phases :

– Écriture de requête non paramétrées dans un fichier sql et test du résultat.
– Inclusion des requêtes dans des fonctions php, en utilisant des paramètres.

Cette démarche a permis de concevoir et d'affiner les requêtes dans un cas assez simple pour ensuite pouvoir les intégrer de manière efficace à notre interface. Certaines requêtes étant compliquées, elles ont été conçues étape après étape, en s'approchant peu à peu du résultat. Il est plus simple, pour ce genre de procédure, d'affiner le résultat en exécutant un fichier sql plutôt qu'en passant directement par du php.

Statistiques Les statistiques suivantes peuvent être prélevées grâce aux requêtes implémentées :

– le nombre d'histoires auxquelles un auteur a participé ;

– la série ayant le plus grand nombre d'auteurs ;
– les histoires classées selon le nombre de fois où elles apparaissent en album ;
– le nombre moyen d'auteurs participant à une revue pendant une période donnée.

Les méthodes utilisées pour le développement des requêtes de statistiques sont les mêmes que celles utilisées pour les requêtes de consultation. La requête permettant de classer les auteurs selon la décade durant laquelle ils ont créé le plus d'histoires n'a pas été développée, notre compréhension du résultat attendu n'étant pas précise. Cependant, nous avons tout de même testé la possibilité d'afficher le nombre d'histoires ayant été écrites pour chaque décade, il ne nous serait donc pas particulièrement difficile d'afficher la décade durant laquelle un auteur spécifique a créé le plus d'histoires. En revanche, il nous semble bien plus complexe d'effectuer cette opération pour tous les auteurs simultanément.

Mise à jour Il est possible d'appliquer les opérations de modification de la tables suivantes :
– Ajout, suppression, modification d'une histoire, d'un volume, d'un auteur, d'une série.
– Création de collections rendant par la suite possible la création d'albums avec collection.
– Association d'histoires à des séries et à des auteurs.

6.1 Explication détaillée des requêtes

La bibliographie d'un auteur par ordre chronologique

```
SELECT DISTINCT *
 FROM auteuriser ai, histoire h
 WHERE ai.no_auteur = 2
   AND ai.no_histoire = h.no_histoire;
```

Cette requête consiste simplement à conserver les différentes histoires pour lesquels une entrée contient le numéro d'auteur approprié dans la table auteuriser. Dans ce cas, le numéro d'auteur est le 2, mais il est possible de le paramétrer simplement en php. Le "DISTINCT" permet d'éviter les répétitions dans le cas où un auteur a rempli plusieurs rôles dans la création d'une histoire.

La bibliographie d'un auteur triée par année de parution comprenant ses collaborateurs

```
SELECT DISTINCT tmp.no_histoire, titre, annee_parution,
                nom_auteur, prenom_auteur, nom_role
  FROM auteur a, auteuriser_et_role ai,
       (SELECT DISTINCT h.no_histoire, titre, annee_parution
          FROM auteuriser ai, histoire h
          WHERE ai.no_auteur = 2
            AND ai.no_histoire = h.no_histoire) tmp
  WHERE tmp.no_histoire = ai.no_histoire
    AND ai.no_auteur = a.no_auteur
  ORDER BY annee_parution;
```

À partir des numéros d'histoire auquels l'auteur spécifié a participé, il suffit de faire le lien entre les auteurs et les histoires à l'aide de la vue auteuriser_et_role[1]. Le tri se fait simplement par "ORDER BY annee_parution". À nouveau, dans ce cas le numéro d'auteur choisi est 2, mais il est facile d'adapter la requête pour qu'elle soit paramétrable en php.

La bibliographie d'un auteur triée par série comprenant ses collaborateurs

```
SELECT b.no_histoire, titre, annee_parution, nom_auteur, prenom_auteur,
       nom_role, titre_serie, s.no_serie, no_ds_serie
  FROM (SELECT b.no_histoire, titre, annee_parution, nom_auteur,
               prenom_auteur, nom_role, no_serie, no_ds_serie
          FROM (SELECT tmp.no_histoire, titre, annee_parution,
                       nom_auteur, prenom_auteur, nom_role
                  FROM auteur a, auteuriser_et_role ai,
                       (SELECT h.no_histoire, h.titre, annee_parution
                          FROM auteuriser ai, histoire h
                          WHERE ai.no_auteur = 2
                            AND ai.no_histoire = h.no_histoire) tmp
                  WHERE tmp.no_histoire = ai.no_histoire
                    AND ai.no_auteur = a.no_auteur) b
          LEFT OUTER JOIN appartenance_serie a
          ON a.no_histoire = b.no_histoire) b
  LEFT OUTER JOIN serie s
  ON s.no_serie = b.no_serie
  ORDER BY no_serie, no_ds_serie;
```

Cette requête est très proche de la précédente, la principale différence réside dans le fait qu'il est nécessaire de récupérer les données des séries afin d'afficher leur nom et de pouvoir les trier. Il est très important ici de bien utiliser une jointure externe gauche, car des histoires peuvent ne pas être présentes dans des séries, or dans ce cas, on désire tout de même les conserver[2]. À nouveau, le numéro de l'auteur est facilement paramétrable en php. Et cette fois l'ordre est imposé par le numéro de série ainsi que par le numéro interne à la série.

Les auteurs participants à une revue durant une période donnée

```
SELECT nom_auteur, prenom_auteur, nb_revues
  FROM auteur a,
       (SELECT no_auteur, count(*) as nb_revues
          FROM (SELECT DISTINCT no_auteur, no_revue
                  FROM auteuriser a,
                       (SELECT no_histoire, no_revue
                          FROM contenir c,
                               (SELECT v.no_volume, no_revue
```

1. Il est nécessaire d'avoir accès au rôle pour les afficher, ce qui est préférable si l'on souhaite afficher les collaborateurs, il est intéressant de savoir quel rôle ils ont tenu dans la création de l'histoire.

2. Avec une jointure naturelle, les histoires qui ne sont pas présentes dans la table appartenance_serie ne sont pas conservées.

```
                                    FROM volume v, revue r
                                    WHERE v.no_volume = r.no_volume
                                      AND titre = 'Revue_1'
                                      AND annee_edition >= 1996
                                      AND annee_edition <= 1998) r
                            WHERE c.no_volume = r.no_volume) h
                    WHERE a.no_histoire = h.no_histoire) t
            GROUP BY no_auteur) t
  WHERE t.no_auteur = a.no_auteur;
```

Pour cette requête, nous avons d'abord cherché à obtenir les numéros de volume et les numéros de revue des volumes correspondant aux conditions spécifiées, cette partie est celle présente dans le *select* le plus imbriqué. Il faut ensuite obtenir les numéros des histoires écrites dans ces revues et combiner cette information avec la table auteuriser pour obtenir une table comportant les auteurs associés aux numéros de revue auxquels ils ont participé. Ayant assuré qu'il n'y a pas de doublons grâce au "SELECT DISTINCT no_auteur, no_revue", il est possible d'associer au numéro d'auteur le nombre de revues auquelles il a participé à l'aide du "GROUP BY no_auteur". Il ne reste plus qu'à ajouter les noms et prénoms des auteurs en se servant de la table auteur. Cette requête possède trois paramètres, le titre de la revue choisie, l'année d'édition minimale et l'année d'édition maximale, ils sont tous les trois faciles à paramétrer en php.

L'historique des publications d'une histoire

```
SELECT v.no_volume, titre, annee_edition, annotation
  FROM contenir c, volume v
  WHERE c.no_histoire = 1
    AND c.no_volume = v.no_volume
  ORDER BY annee_edition;
```

Cette requête nécessite simplement d'obtenir les volumes présent dans la relation contenir avec le numéro d'histoire approprié. Inclure les annotations permet d'avoir une meilleure vision de la différence entre deux publications dans la même revue la même année par exemple.

Les histoires différentes portant le même titre

```
SELECT h1.no_histoire, h1.titre
  FROM histoire h1, histoire h2
  WHERE h1.no_histoire <> h2.no_histoire
    AND h1.titre = h2.titre
  ORDER BY titre , h1.no_histoire;
```

Cette requête nécessite d'avoir deux alias définis pour la même table, si elles n'ont pas le même numéro d'histoire mais le même titre, c'est qu'elles sont différentes tout en portant le même titre. Cette requêtes n'est pas paramétrable étant donné qu'elle est de type général.

Le nombre d'histoires écrites par un auteur

```
SELECT count(*) as nb_histoires
  FROM (SELECT DISTINCT no_histoire
          FROM auteuriser
                WHERE no_auteur = 2) t;
```

Pour obtenir le nombre d'histoires écrites par un auteur il suffit de compter le nombre d'histoires différentes auxquelles il a participé. Cette requête peut facilement être paramétrée en fonction du numéro d'auteur souhaité.

La série avec le plus d'auteurs

```
SELECT s.no_serie, titre_serie, nb_auteur
  FROM serie s,
       (SELECT no_serie, count(*) as nb_auteur
          FROM (SELECT DISTINCT no_auteur, no_serie
                  FROM auteuriser ai, appartenance_serie a
                  WHERE ai.no_histoire = a.no_histoire) t
          GROUP BY no_serie
          ORDER BY count(*) desc
          LIMIT 1) t
  WHERE s.no_serie = t. no_serie;
```

Pour cette requête, il a été nécessaire d'obtenir en premier lieu une liste des auteurs ayant participé à chaque série. Dans ce cas, le "DISTINCT" est indispensable, car il est fortement probable que certains auteurs aient participé à plusieurs histoire appartenant à une même série, on évite ainsi les doublons. À l'aide du "GROUP BY no_serie", on obtient le nombre d'auteurs ayant participé à chaque série, le "ORDER BY count(*) desc" garantit que la première ligne sera la série avec le plus d'auteurs et le "LIMIT 1" permet de ne prendre que la première ligne, donc la série avec le plus d'auteurs. Le reste de la requête sert uniquement à récupérer le titre de la série en question.

Les histoires triées par nombre d'albums

```
SELECT h.*, nb_albums
  FROM histoire h,
       (SELECT no_histoire, count(*) as nb_albums
          FROM contenir c,
               (SELECT no_volume
                  FROM album_sans_collection
                UNION
                SELECT no_volume
                  FROM album_avec_collection) v
          WHERE v.no_volume = c.no_volume
          GROUP BY no_histoire) t
  WHERE h.no_histoire = t.no_histoire
  ORDER BY nb_albums desc;
```

Pour cette requête, le premier pas est d'obtenir les numéros de volumes présent dans la table album_sans_collection ou album_avec_collection, ce qui se fait naturellement avec une union. Pour ce genre de cas, l'utilisation d'une vue serait parfaitement pertinente. En combinant les numéros obtenus avec la relation c et en utilisant le bon "GROUP BY", on obtient le nombre d'albums correspondant à chaque histoire. Avec le premier SELECT, on obtient les autres champs d'histoires (tel que le titre) et grâce au "ORDER BY" on assure que les entrées seront classées de manière décroissantes par rapport au nombre d'albums.

Le nombre moyen d'auteurs participant à une revue durant une période donnée

```
SELECT avg(nb_auteurs) as nb_moyen_auteurs
  FROM (SELECT no_volume, count(*) as nb_auteurs
          FROM (SELECT DISTINCT no_auteur, c.no_volume
                  FROM auteuriser a,
                       contenir c,
                       revue r,
                       volume v
                  WHERE v.titre = 'Revue_1'
                    AND v.annee_edition >= 1996
                    AND v.annee_edition <= 1998
                    AND r.no_volume = v.no_volume
                    AND c.no_volume = v.no_volume
                    AND c.no_histoire = a.no_histoire) t
          GROUP BY no_volume) t;
```

Pour obtenir le nombre moyen d'auteurs participant à une revue durant une période donnée, il est tout d'abord nécessaire d'obtenir tous les auteurs ayant participé à chaque volume correspondant aux conditions spécifiées. En groupant les données par volume on peut ensuite obtenir le nombre d'auteurs ayant collaboré à chaque volume. Il suffit ensuite de faire la moyenne entre ces différents nombre pour obtenir la donnée souhaitée. Cette requête peut aisément être paramétrée en php afin de répondre à la question posée pour n'importe quelle revue, n'importe quelle année minimale d'édition et n'importe quelle année maximale d'édition.

7 Utilisation

7.1 Notice d'utilisation

L'interface peut être utilisée en plaçant les fichiers php, css et png dans un répertoire connu d'un serveur http équipé d'un module php. Les requêtes sont envoyées à MySQL via les fonctions du types `mysql_*`. Il peut être nécessaire d'installer les bibliothèques nécessaires et/ou activer ce jeu de fonctions dans la configuration de php (sous linux, cette configuration se trouve dans /etc/php/php.ini la plupart du temps).

La base est destinée à MySQL et peut être initialisée avec :

```
mysql> create database nom_de_la_bdd;
mysql> use nom_de_la_bdd;
mysql> source base.mysql;
```

```
mysql> source insertion.mysql;
```

Il est alors nécessaire d'éditer le fichier include.php pour mettre le contenu de la variable `$bdd` avec `nom_de_la_bdd` dans la fonction `connectdb()`. Si nécessaire, le contenu des variables `$user` et `$passwd` utilisés pour la connexion à la base peut être changé au même endroit.

Par défaut, la base de données "test" (disponible sans mot de passe et nom d'utilisateur après une installation fraîche de MySQL) est utilisée.

7.2 Description de l'interface

Le langage PHP a été utilisé pour réaliser l'interface utilisateur avec la base. Le menu à gauche permet d'accéder aux pages des différentes entités principales en cliquant sur les liens correspondants. Il est à chaque fois possible de revenir à la page d'accueil en cliquant sur le titre. En cliquant sur le lien "Auteurs", par exemple, l'utilisateur est dirigé vers une page listant le contenu de la table des auteurs, c'est-à-dire leur nom, leur prénom et un numéro les identifiant. Il est possible d'effectuer des insertions dans la table au moyen d'un formulaire d'ajout dans lequel le nom et le prénom des auteurs sont à remplir :

FIGURE 1 – Formulaire d'ajout d'un auteur

Le numéro d'identification de l'auteur est automatiquement affecté de manière à être unique : c'est un numéro incrémenté en interne à chaque ajout d'un auteur.

Pour chaque auteur, deux actions sont proposées : leur suppression de la table et leur édition. L'appui sur les boutons associés, situés sur chaque ligne d'auteur, permet d'effectuer ces actions. L'appui sur le bouton "Supprimer" va lancera l'exécution de la requête supprimant l'auteur de la table.

L'appui sur le bouton "Editer" fait apparaître un formulaire en haut de la page ; les champs "Nom" et "Prenom" sont pré-remplis avec le nom de le prénom de l'auteur édité et peuvent être modifié.

Par la suite, l'appui sur le bouton "Valider" applique les modifications mais ne fait pas disparaître le formulaire apparu pour des raisons de convenance mais il est néanmoins mis à jour pour rendre compte des modifications apportées à la base. Il est en effet pratique de pouvoir appliquer un ensemble de modifications sans avoir à appuyer sur le bouton "Editer" à chaque fois. Par exemple, pour la table histoire, de nombreuses modifications différentes peuvent être effectuées suite à l'appui sur le bouton "Editer" d'une histoire :

- modification du titre et de l'année de parution ;
- ajouts d'auteurs ;
- suppression d'auteurs.

Devoir appuyer sur éditer à chaque ajout d'un auteur, par exemple, aurait été contraignant.

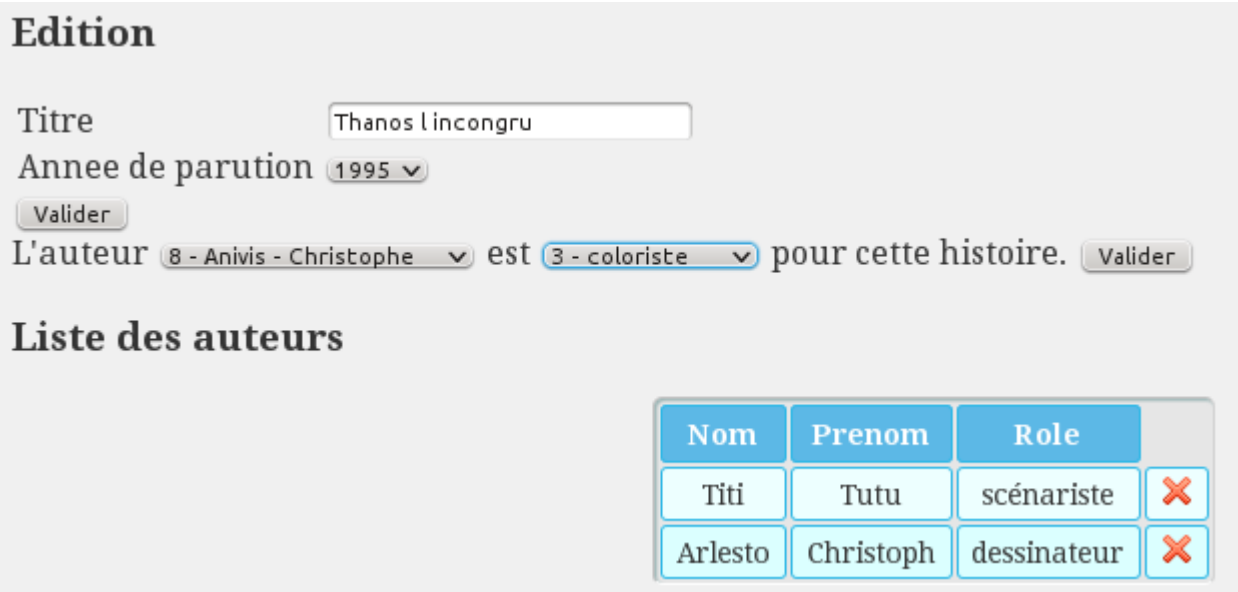


FIGURE 2 – Formulaire d'édition d'une histoire

Depuis la page d'accueil il est également possible d'accéder aux pages permettant d'effectuer les différentes requêtes. La page "Consultation" permet d'accéder aux requêtes de consultation et la page "Statistiques" permet d'effectuer les requêtes de statistiques. Dans ces deux pages, le principe est identique pour effectuer une requête :

- un menu déroulant permet de choisir la requête voulu ;
- l'appui sur le bouton "Valider" fait apparaître le choix des paramètres de cette requête ;
- une fois les paramètres choisis, l'appui sur le nouveau bouton "Valider" donne le résultat sous la forme d'une table. Si la table de résultat est vide alors le message "Ressource vide" est affiché.

FIGURE 3 – Menu de choix des requêtes

Remarque Un effort a été fait pour essayer empêcher les injections SQL en formatant les entrée de l'utilisateur. Pour cela, l'utilisation de la fonction "sprintf" permet de forcer l'entrée de l'utilisateur à être traitée comme un nombre lorsque cela doit être le cas. Lorsque l'utilisateur a la possibilité d'entrer des chaînes de caractères, la fonction "mysql_real_escape_string" a été employée afin de protéger les requêtes SQL de caractères spéciaux.

8 Conclusion

Ce projet nous a permis de comprendre plus précisément le déroulement de la création de bases de données dans son intégralité et nous a fait nous poser des questions sur l'optimisation des requêtes (notamment sur l'utilisation des vues). Des questions sur le bon rapport entre le recours au php pour réaliser certaines tâches contre une approche purement SQL ont été levées et nous avons préféré recourir le moins possible aux manipulations faites par l'interface et privilégier les contraintes d'intégrité.

L'utilisation de MySQL nous a permis de découvrir une alternative au SGBD proposé en TD.