Projet Base de données avancées

Equipe : SENECHAL Morgan, SAVORY Edwin, PARMENTIER HUGO-L3-Groupe C

Professeur : BENNAI Soufia

Module : TI603

Table des matières

I. Introduction

Les bases de données sont des outils informatiques qui permettent de stocker, organiser et gérer des informations de manière structurée. Elles sont essentielles dans notre monde numérique moderne où les quantités de données générées chaque jour sont astronomiques. Les bases de données sont utilisées dans presque tous les secteurs de l'industrie, de la santé à la finance, en passant par la vente au détail et la logistique.

Le fonctionnement des bases de données est basé sur le principe de stockage d'informations dans des tables, où chaque table contient des enregistrements et des champs de données. Les enregistrements sont les éléments de données stockés dans chaque table, tandis que les champs décrivent les différentes caractéristiques de ces enregistrements. Les bases de données permettent de créer des relations entre différentes tables, ce qui facilite l'accès aux informations.

L'utilité des bases de données est multiple : elles permettent de stocker et de gérer des informations de manière sécurisée, de faciliter la recherche et l'accès aux données, d'automatiser les processus de traitement des données, de faciliter la prise de décision et de permettre l'analyse de données pour l'amélioration des performances. Elles sont essentielles dans notre monde numérique, où les quantités de données générées chaque jour augmentent de manière exponentielle.

Dans le cadre de ce projet, nous allons réaliser différentes tâches touchant à différentes thématiques des bases de données (Préparation des données, Normalisation du schéma, Utilisation d’index, Utilisation des vues, procédure stockée et utilisation des trigger) en utilisant la base de données project.csv, une base centrée sur le thème d’Harry Potter.

II. Partie 1

1. Étape 0 : Préparation

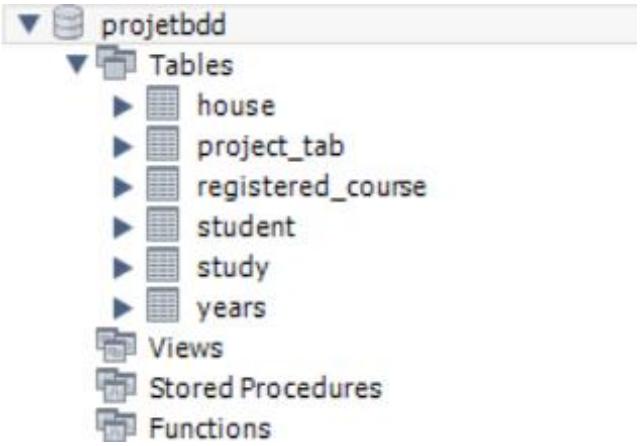
1. Récupérer le fichier project.csv sur Moodle et le télécharger

Le fichier récupéré sur Moodle « project.csv » correspond à la base de données que nous allons exploiter lors de ce projet.

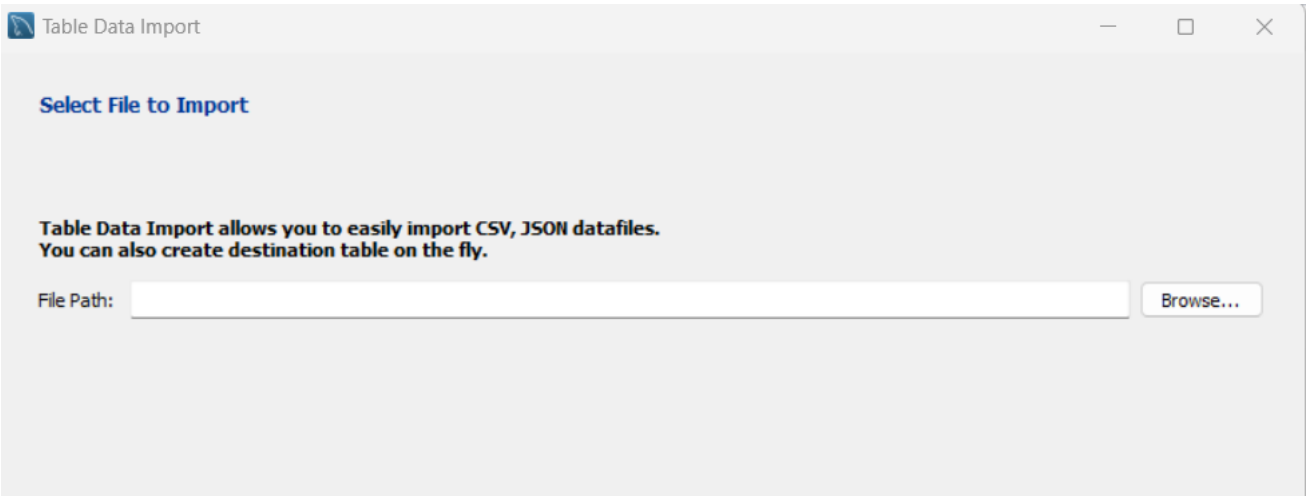
2. Avec MySQL Workbench, créer une base de données nommée : project

Nous avons réussi à créer notre base de données nommée : projetbdd

3. Importer le fichier project.csv dans votre base de données à l'aide de la méthode de votre choix.



Pour importer le fichier « project.csv » dans notre base de données : projetbdd, nous avons importé notre fichier csv à l'intérieur de la base de données crée projetbdd :



4. Explorer la base de données en faisant les requêtes pour afficher les informations mentionnées ci-dessous.

Les requêtes SQL à faire sont les suivantes :

a. Afficher l'ensemble des tables en SQL :

```
SHOW tables;
```

Tables_in_projetbdd
house
project_tab
registered_course
student
study
years

b. Afficher les colonnes de la table "project" :

```
SHOW COLUMNS FROM project_tab;
```

Field	Type	Null	Key	Default	Extra
student_name	text	YES		NULL	
email	text	YES		NULL	
registered_course	text	YES		NULL	
year	double	YES		NULL	
house	text	YES		NULL	
prefet	text	YES		NULL	

c. Le nombre d'étudiants dans la base de données :

```
SELECT Count(distinct student_name)from project_tab;
```

Count(distinct student_name)
31

d. Les différents cours dans la base de données :

```
SELECT distinct registered_course from project_tab;
```

registered_course
potion
sortilege
botanique

e. Les différentes maisons dans la base de données :

```
SELECT distinct house from project_tab;
```

house
Gryffondor
Serpentard
Poufsouffle
Serdaigle

f. Les différents préfets dans la base de données :

```
SELECT distinct prefet from project_tab;
```

	prefet
▶	Godrick
	Hermione
	Vrouminette
	Help

g. Quel est le préfet pour chaque maison ? :

```
SELECT distinct prefet, house from project_tab;
```

	prefet	house
▶	Godrick	Gryffondor
	Hermione	Serpentard
	Vrouminette	Poufsouffle
	Help	Serdaigle

h. Pour compter le nombre d'étudiants par année :

```
SELECT count(distinct student_name) as nb_éudiant,year from project_tab group by year;
```

	nb_éudiant	year
▶	27	1
	2	2
	2	3

i. Pour afficher les noms et les emails des étudiants qui suivent le cours "potion" :

```
SELECT distinct student_name, email from project_tab where (SELECT registered_course='potion');
```

	student_name	email
▶	Aiden Ortiz	aiden.ortiz@poudlard.edu
	Levi Patel	levi.patel@poudlard.edu
	Olivia Plea	olivia.plea@poudlard.edu
	William Davis	william.davis@poudlard.edu
	Penelope Price	penelope.price@poudlard.edu
	Liam Zune	liam.zune@poudlard.edu
	Victoria Bailey	victoria.bailey@poudlard.edu
	Connor Gutierrez	connor.gutierrez@poudlard.edu
	Noah Misonal	noah.misonal@poudlard.edu

j. Pour afficher les étudiants qui ont une année supérieure à 2 :

```
SELECT distinct student_name,year from project_tab where year>2;
```

	student_name	year
▶	Ava Sarda	3
	Emma Smith	3

k. Pour trier les étudiants par ordre alphabétique de leur nom :

```
SELECT distinct student_name from project_tab order by student_name;
```

student_name
Aiden Ortiz
Aria Flores
Ava Jones
Ava Sarda
Caleb Howard
Chloe Reed
Connor Gutierrez
Emma Romalas

l. Pour trouver le nombre d'étudiants de chaque maison qui suivent le cours "potion" :

```
SELECT count(distinct student_name) as nb_étudiant_of_eache_house_that_follow_potion_course, house
from project_tab where registered_course='potion' group by house;
```

nb_étudiant_of_eache_house_that_follow_potion	house
8	Gryffondor
8	Poufsouffle
7	Serdaigle
8	Serpentard

m. Afficher les maisons des étudiants et le nombre d'étudiants dans chaque maison :

```
SELECT count(distinct student_name) as nb_étudiant,house from project_tab group by house
```

nb_étudiant	house
8	Gryffondor
8	Poufsouffle
7	Serdaigle
8	Serpentard

n. Afficher le nombre de cours pour chaque année :

```
SELECT count(distinct registered_course) as nb_course, year from project_tab group by year;
```

nb_course	year
3	1
3	2
3	3

o. Afficher le nombre d'étudiants inscrits à chaque cours :

```
SELECT count(distinct student_name) as nb_étudiant, registered_course from project_tab group by registered_course;
```

nb_étudiant	registered_course
31	botanique
31	potion
31	sortilege

p. Afficher les cours auxquels les étudiants de chaque maison sont inscrits :

```
SELECT distinct registered_course,house from project_tab order by house;
```

registered_course	house
botanique	Gryffondor
potion	Gryffondor
sortilege	Gryffondor
botanique	Poufsouffle
potion	Poufsouffle
sortilege	Poufsouffle
botanique	Serdaigle
potion	Serdaigle
sortilege	Serdaigle

q. Afficher le nombre d'étudiants dans chaque année pour chaque maison :

```
SELECT COUNT(distinct student_name) as student_count,house, year FROM project_tab GROUP BY house, year ORDER BY year;
```

student_count	house	year
7	Gryffondor	1
7	Poufsouffle	1
6	Serdaigle	1
7	Serpentard	1
1	Poufsouffle	2
1	Serdaigle	2
1	Gryffondor	3
1	Serpentard	3

r. Afficher les cours auxquels les étudiants de chaque année sont inscrits :

```
SELECT distinct registered_course, student_name, year from project_tab group by student_name order by year;
```

registered_course	student_name	year
potion	Aiden Ortiz	1
sortilege	Ava Jones	1
sortilege	Madison Wright	1
sortilege	Connor Gutierrez	1
sortilege	Victoria Bailey	1
sortilege	Luna Kim	1
sortilege	Emma Romalas	1
sortilege	Chloe Reed	1
sortilege	Grace Nelson	1

s. Afficher les maisons des étudiants et le nombre d'étudiants dans chaque maison, triés par ordre décroissant :

```
SELECT count(distinct student_name) as nb_étudiant, house from project_tab group by house order by house DESC;
```

nb_étudiant	house
8	Serpentard
7	Serdaigle
8	Poufsouffle
8	Gryffondor

t. Afficher le nombre d'étudiants inscrits à chaque cours, triés par ordre décroissant :

```
SELECT count(distinct student_name) as nb_étudiant, registered_course
from project_tab group by registered_course order by registered_course DESC;
```

nb_étudiant	registered_course
31	sortilege
31	potion
31	botanique

u. Afficher les préfets de chaque maison, triés par ordre alphabétique des maisons :

```
SELECT distinct prefet,house from project_tab group by house order by house;
```

prefet	house
Godrick	Gryffondor
Vrouminette	Poufsouffle
Help	Serdaigle
Hermione	Serpentard

2. Étape 1-Normaliser le schéma

1. Expliquer pourquoi cette base de données n'est pas normalisée :

La base de données n'est pas normalisée car il y a des redondances.

2. Identifier les dépendances fonctionnelles et les formes normales qui ne sont pas respectées :

Dépendance :

-Student_name avec email car dans l'email on a le nom et prénom.

-Prefet avec House car une maison a un préfet.

Forme normale non respecté :

-1NF : La forme normal 1NF ne contient pas de valeurs répétitives ou de groupes de valeurs répétitives. Notre table contient de la redondance par conséquent elle ne respecte pas la 1ère forme.

Comme la 1NF n'est pas respectée, la 2NF, 3NF et 4NF ne le sont pas aussi.

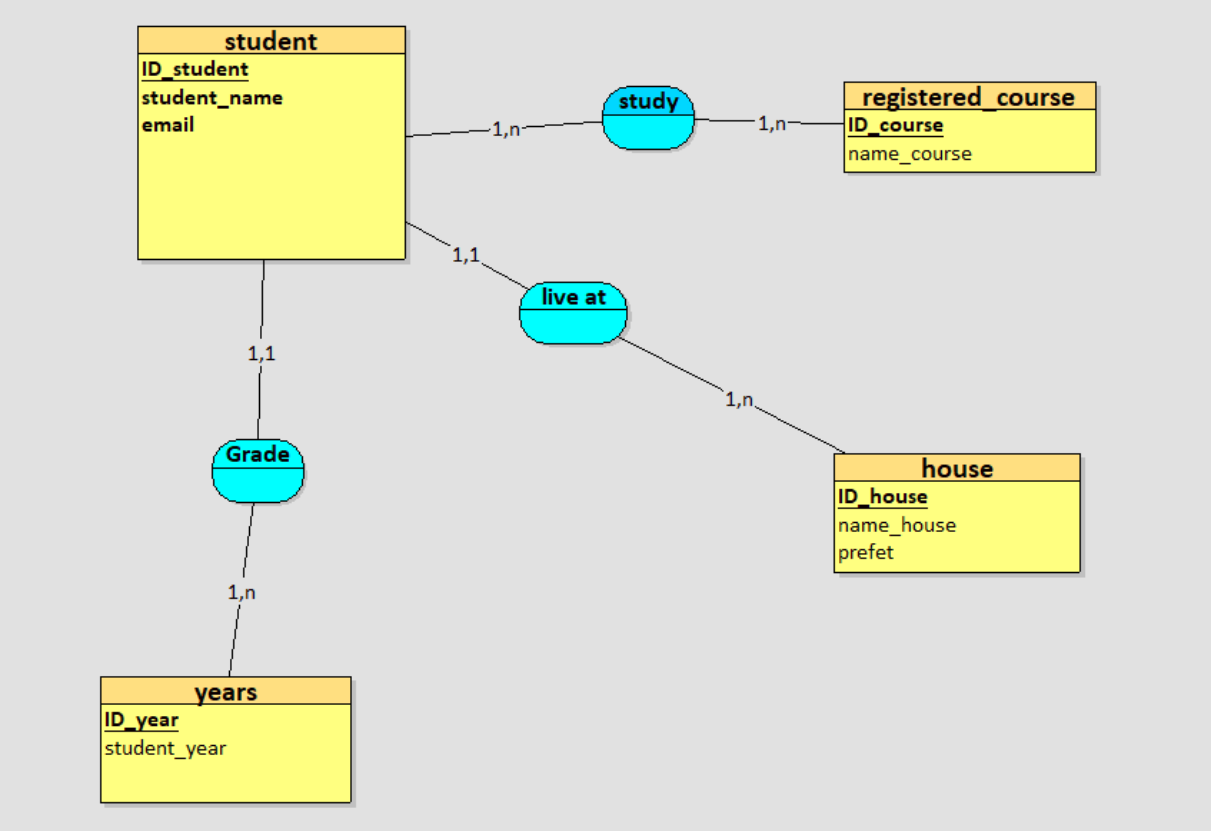
3. Proposer une normalisation du schéma. Expliquer les modifications apportées pour normaliser le schéma :

1-Nous devons créer une entité student_name pour y mettre : ID_student, student_name et email.

2-Nous devons créer une entité house pour y mettre : ID_house, nom_house et prefet.

3-Pour les autres entités (registered_course et years) nous devons créer 2 entités à part pour y placer les attributs de celle-ci.

4. Faire le MCD du schéma normalisé :



5. Faire le MLD du schéma normalisé :

MLD

house = (ID_house *BYTE*, name_house *VARCHAR(50)*, prefet *VARCHAR(50)*);
registered_course = (ID_course *BYTE*, name_course *VARCHAR(50)*);
years = (ID_year *BYTE*, student_year *BYTE*);
student = (ID_student *INT*, **student_name** *VARCHAR(255)*, **email** *VARCHAR(255)*, *#ID_house, #ID_year*);
study = (*#ID_student, #ID_course*);

6. Trouvez un moyen (ou plutôt une commande) pour sauvegarder votre base de données avant d'effectuer vos modifications. Garder précieusement votre sauvegarde. PS. NON, faire un copier-coller de votre fichier SQL n'est PAS un moyen de sauvegarde accepté :

Pour sauvegarder sur MySQL : mysqldump -u username -p dbname > backup.sql

7. Maintenant, passons aux choses sérieuses, vous devez modifier votre base de données pour la normaliser avec des requêtes SQL :

- Vous devez faire des requêtes SQL pour chacune de ces étapes :
- Regrouper les attributs qui dépendent fonctionnellement les uns des autres en des tables distinctes. Donc, créer des tables normalisées.
- Créer une clé primaire pour chaque table nouvellement créée.
- Ajouter des clés étrangères pour les tables qui ont des dépendances fonctionnelles avec d'autres tables. iv. Supprimer des données si nécessaire.

Nous utilisons ces requêtes SQL pour créer et remplir nos tables suivant le MCD/MLD proposé pour normaliser notre base de données :

```sql
CREATE TABLE house(
   ID_house INT,
   name_house VARCHAR(50),
   prefet VARCHAR(50),
   PRIMARY KEY(ID_house) #Clés primaire
);
CREATE TABLE prefet (
  ID_house INT,
  prefet_name VARCHAR(50),
  FOREIGN KEY (ID_house) REFERENCES house(ID_house)
);


CREATE TABLE registered_course(
   ID_course INT,
   name_course VARCHAR(50),
   PRIMARY KEY(ID_course) #Clés primaire
);

CREATE TABLE years(
   ID_year INT,
   student_year INT,
   PRIMARY KEY(ID_year) #Clés primaire
);


CREATE TABLE student(
    ID_student INT,
    student_name VARCHAR(255),
    email VARCHAR(255),
    ID_house INT NOT NULL,
    ID_year INT NOT NULL,
    PRIMARY KEY(ID_student), #Clés primaire
    UNIQUE(student_name),
    UNIQUE(email),
    FOREIGN KEY(ID_house) REFERENCES house(ID_house), #Clés étrangére
    FOREIGN KEY(ID_year) REFERENCES years(ID_year) #Clés étrangére
);

CREATE TABLE study(
    ID_student INT,
    ID_course INT,
    PRIMARY KEY(ID_student, ID_course), #Clés primaire
    FOREIGN KEY(ID_student) REFERENCES student(ID_student), #Clés étrangére
    FOREIGN KEY(ID_course) REFERENCES registered_course(ID_course) #Clés étrangére
);
```

```
ALTER TABLE study DROP FOREIGN KEY study_ibfk_1;
ALTER TABLE student MODIFY ID_student INT AUTO_INCREMENT;
ALTER TABLE study ADD CONSTRAINT study_ibfk_1 FOREIGN KEY (ID_student) REFERENCES student(ID_student);

ALTER TABLE student DROP FOREIGN KEY student_ibfk_1;
ALTER TABLE house MODIFY ID_house INT AUTO_INCREMENT;
ALTER TABLE student ADD CONSTRAINT student_ibfk_1 FOREIGN KEY (ID_house) REFERENCES house(ID_house);


ALTER TABLE study DROP FOREIGN KEY study_ibfk_2;
ALTER TABLE registered_course MODIFY ID_course INT AUTO_INCREMENT;
ALTER TABLE study ADD CONSTRAINT study_ibfk_2 FOREIGN KEY (ID_course) REFERENCES registered_course(ID_course);


ALTER TABLE student DROP FOREIGN KEY student_ibfk_2;
ALTER TABLE years MODIFY ID_year INT AUTO_INCREMENT;
ALTER TABLE student ADD CONSTRAINT student_ibfk_2 FOREIGN KEY (ID_year) REFERENCES years(ID_year);


#Remplir les tables:
INSERT INTO house (ID_house, name_house)
SELECT DISTINCT NULL, house
FROM project;

INSERT INTO registered_course (ID_course, name_course)
SELECT DISTINCT NULL, registered_course
FROM project;

INSERT INTO years (ID_year, student_year)
SELECT DISTINCT NULL, year
FROM project;


INSERT INTO student (ID_student, student_name, email, ID_house, ID_year)
SELECT DISTINCT NULL, student_name, email, house.ID_house, years.ID_year
FROM project
INNER JOIN house ON project.house = house.name_house
INNER JOIN years ON project.year = years.student_year;


INSERT INTO study (ID_student, ID_course)
SELECT DISTINCT student.ID_student, registered_course.ID_course
FROM project
INNER JOIN student ON project.email = student.email
INNER JOIN registered_course ON project.registered_course = registered_course.name_course;


select distinct * from student
```

Cette requête SQL créer les tables nécessaires pour stocker des informations sur les maisons, les préfets, les cours inscrits, les années scolaires, les étudiants et les études. Elle définit également les clés primaires et les clés étrangères appropriées pour assurer l'intégrité des données.

Les premières requêtes créent les tables house, registered_course, years, student et study, chacune avec ses colonnes appropriées et des clés primaires définies pour les colonnes ID_xxx.

Les commandes ALTER TABLE qui suivent ajustent certaines des tables, notamment en ajoutant l'option AUTO_INCREMENT pour les colonnes ID_xxx et en définissant des contraintes de clé étrangère pour assurer l'intégrité des données.

Les dernières requêtes insèrent des données dans les tables à partir d'une autre table appelée "project". La dernière requête SELECT DISTINCT * FROM student affiche tous les enregistrements de la table des étudiants.

En résumé, ce code SQL crée et peuple des tables pour stocker des informations sur les maisons, les préfets, les cours inscrits, les années scolaires, les étudiants et les études.

<u>Nous obtenons nos tables normalisées remplit :</u>

Table house :

ID_house	name_house
1	Gryffondor
2	Serpentard
3	Poufsouffle
4	Serdaigle
NULL	NULL

Table prefet:

ID_house	prefet
1	Godrick
2	Hermione
3	Vrouminette
4	Help

Table registered_course :

ID_course	name_course
1	potion
2	sortilege
3	botanique
NULL	NULL

Table student :

ID_student	student_name	email	ID_house	ID_year
1	Aiden Ortiz	aiden.ortiz@poudlard.edu	1	1
2	Ava Jones	ava.jones@poudlard.edu	2	1
3	Madison Wright	madison.wright@poudlard.edu	2	1
4	Connor Gutierrez	connor.gutierrez@poudlard.edu	3	1
5	Victoria Bailey	victoria.bailey@poudlard.edu	2	1
6	Luna Kim	luna.kim@poudlard.edu	4	1
7	Emma Romalas	emma.romalas@poudlard.edu	1	1
8	Chloe Reed	chloe.reed@poudlard.edu	4	1
9	Grace Nelson	grace.nelson@poudlard.edu	4	1
10	Scarlett Torres	scarlett.torres@poudlard.edu	4	1
11	Olivia Williams	olivia.williams@poudlard.edu	4	1
12	Levi Patel	levi.patel@poudlard.edu	1	1
13	Olivia Plea	olivia.plea@poudlard.edu	3	2
14	William Davis	william.davis@poudlard.edu	3	1
15	Owen Wood	owen.wood@poudlard.edu	3	1
16	Penelope Price	penelope.price@poudlard.edu	2	1
17	Liam Zune	liam.zune@poudlard.edu	4	2
18	Noah Misonal	noah.misonal@poudlard.edu	2	1
19	Aria Flores	aria.flores@poudlard.edu	2	1

Table study :

ID_student	ID_course
1	1
2	1
3	1
4	1
5	1
6	1
7	1
8	1
9	1
10	1
11	1
12	1
13	1
14	1
15	1
16	1
17	1

Table years :

	ID_year	student_year
▶	1	1
	2	2
	3	3
*	NULL	NULL

<u>On peut aussi voir la création de nos tables en faisant un SHOW Table :</u>

Tables_in_projetbdd
house
project
registered_course
student
study
years

III. Partie 2

1. Influence d'un index

1. Expliquer à quoi sert un index en base de données :

Un index en base de données est un mécanisme permettant d'optimiser la recherche et la récupération des données. Il s'agit d'une structure de données qui est créée sur une ou plusieurs colonnes d'une table de base de données. L'index contient une copie triée de la valeur de la colonne et un pointeur vers la ligne correspondante dans la table.

L'utilisation d'index permet d'accélérer les requêtes de recherche et de récupération de données dans une table. En effet, sans index, une recherche implique de parcourir l'ensemble des enregistrements de la table, ce qui peut prendre beaucoup de temps si la table contient un grand nombre de données. Avec un index, la recherche se fait beaucoup plus rapidement car le système de gestion de base de données peut accéder directement aux enregistrements qui correspondent à la recherche, sans avoir besoin de parcourir l'ensemble de la table.

Cependant, la création d'index peut également avoir des inconvénients. Tout d'abord, les index prennent de la place sur le disque dur, ce qui peut être problématique si la base de données est très grande et que l'espace de stockage est limité. De plus, les index doivent être maintenus à jour à chaque modification des données de la table, ce qui peut entraîner des ralentissements lors de l'insertion, la mise à jour ou la suppression de données.

2. Trouver les requêtes SQL afin de :

- compter le nombre d'étudiants qui sont dans la maison "Gryffindor" ;

```
SELECT COUNT(*) FROM student WHERE ID_house = (SELECT ID_house FROM house WHERE name_house = 'Gryffondor');
```

	COUNT(*)
▶	11

- mesurer le temps de la requête avec la commande SHOW PROFILE

```
SET profiling = 1;
SELECT * FROM student WHERE ID_house = (SELECT ID_house FROM house WHERE name_house = 'Gryffindor');
SHOW PROFILE;
```

	Status	Duration
▶	starting	0.000081
	Executing hook on transaction	0.000005
	starting	0.000006
	checking permissions	0.000003
	checking permissions	0.000003
	Opening tables	0.000059
	init	0.000004
	System lock	0.000009
	optimizing	0.000009
	statistics	0.000020
	optimizing	0.000006
	statistics	0.000016
	preparing	0.000019
	executing	0.000048
	executing	0.000086
	end	0.000006
	query end	0.000003
	waiting for handler commit	0.000015
	closing tables	0.000012
	freeing items	0.000084
	cleaning up	0.000019

- ajouter un index sur la colonne "house_id" de la table "students" ;

```
ALTER TABLE student ADD INDEX house_id_idx (ID_house);
```

- mesurer à nouveau le temps de la requête après l'ajout de l'index ;

	Status	Duration
▶	starting	0.000089
	Executing hook on transaction	0.000002
	starting	0.000007
	checking permissions	0.000002
	checking permissions	0.000003
	Opening tables	0.000075
	init	0.000004
	System lock	0.000008
	optimizing	0.000015
	statistics	0.000027
	optimizing	0.000005
	statistics	0.000025
	preparing	0.000029
	executing	0.000047
	executing	0.000011
	end	0.000003
	query end	0.000002
	waiting for handler commit	0.000009
	closing tables	0.000007
	freeing items	0.000067
	cleaning up	0.000007

- mesurer à nouveau le temps de la requête mais sans index.

```
SET profiling = 1;
SELECT * FROM student WHERE ID_house = (SELECT ID_house FROM house WHERE name_house = 'Gryffindor') USE INDEX ();
SHOW PROFILE;
```

	Status	Duration
▶	starting	0.000046
	freeing items	0.000058
	cleaning up	0.000004

3. Pour les requêtes suivantes, vous devez dire à quoi correspond chaque requête. Ensuite, vous devez mesurer le temps de la requête, rajouter un index, mesurer encore une fois le temps de la requête. Si besoin, adapter ces requêtes à votre base de données.

<u>Requête a :</u>

Cette requête est censée permettre de compter le nombre d'étudiants pour chaque maison et chaque cours, et d'afficher les résultats en ordre décroissant en fonction du nombre d'étudiants.

L'architecture de la base de données ne correspond pas du tout à la requête demandée.

```
SET profiling = 1;
SELECT house.name_house, registered_course.name_course, COUNT(*) AS num_students
FROM student
JOIN house ON student.ID_house = house.ID_house
JOIN study ON student.ID_student = study.ID_student
JOIN registered_course ON study.ID_course = registered_course.ID_course
GROUP BY house.name_house, registered_course.name_course
ORDER BY num_students DESC;
SHOW PROFILE;
```

	Status	Duration
▶	starting	0.000112
	Executing hook on transaction	0.000006
	starting	0.000008
	checking permissions	0.000004
	checking permissions	0.000002
	checking permissions	0.000002
	checking permissions	0.000004
	Opening tables	0.000079
	init	0.000006
	System lock	0.000025
	optimizing	0.000025
	statistics	0.000111
	preparing	0.000029
	Creating tmp table	0.000095
	executing	0.000649
	end	0.000005
	query end	0.000003
	waiting for handler commit	0.000035
	closing tables	0.000012
	freeing items	0.000093
	cleaning up	0.000011

Création de l'index :

```
ALTER TABLE student ADD INDEX idx_student_ID_house (ID_house);
```

Grâce à la création de notre index, l'exécution des requêtes est plus rapide de manière général.

Status	Duration
starting	0.000126
Executing hook on transaction	0.000003
starting	0.000007
checking permissions	0.000004
checking permissions	0.000002
checking permissions	0.000002
checking permissions	0.000003
Opening tables	0.001108
init	0.000008
System lock	0.000009
optimizing	0.000014
statistics	0.000081
preparing	0.000020
Creating tmp table	0.000118
executing	0.000672
end	0.000007
query end	0.000003
waiting for handler commit	0.000037
closing tables	0.000012
freeing items	0.000096
cleaning up	0.000015

Requête b :

```
SET profiling = 1;
SELECT student_name, email
FROM student
WHERE ID_student NOT IN (SELECT ID_student FROM study);
SHOW PROFILE ;
```

Cette requête utilise une sous-requête pour sélectionner les ID des étudiants qui sont inscrits à au moins un cours, et sélectionne ensuite les noms et emails des étudiants dont l'ID n'est pas dans cette liste.

Status	Duration
starting	0.000118
Executing hook on transaction	0.000004
starting	0.000008
checking permissions	0.000005
checking permissions	0.000004
Opening tables	0.000112
init	0.000005
System lock	0.000012
optimizing	0.000017
statistics	0.000064
preparing	0.000044
executing	0.000175
end	0.000003
query end	0.000002
waiting for handler commit	0.000015
closing tables	0.000008
freeing items	0.000067
cleaning up	0.000015

```
ALTER TABLE study ADD INDEX idx_study_ID_student (ID_student);
```

La création de l'index nous permet de réduire de manière général le temps d'exécution des requêtes.

Status	Duration
starting	0.000060
Executing hook on transaction	0.000003
starting	0.000005
checking permissions	0.000003
checking permissions	0.000003
Opening tables	0.000050
init	0.000003
System lock	0.000008
optimizing	0.000010
statistics	0.000030
preparing	0.000026
executing	0.000132
end	0.000002
query end	0.000002
waiting for handler commit	0.000009
closing tables	0.000006
freeing items	0.000071
cleaning up	0.000011

Requête c :

Cette requête joint les tables student et house pour récupérer les noms de maisons des étudiants, et utilise une sous-requête pour vérifier si l'étudiant est inscrit à l'un des cours de potions, sortilèges ou botanique.

La clause EXISTS retourne true si la sous-requête renvoie des résultats, ce qui signifie que l'étudiant est inscrit à l'un de ces cours.

Ensuite, la requête regroupe les résultats par maison et compte le nombre d'étudiants dans chaque maison.

```
SELECT house.name_house, COUNT(*) AS num_students
FROM student
JOIN house ON student.ID_house = house.ID_house
WHERE EXISTS (
SELECT *
FROM study
JOIN registered_course ON study.ID_course = registered_course.ID_course
WHERE registered_course.name_course IN ('Potions', 'Sortilèges', 'Botanique')
AND study.ID_student = student.ID_student
)
GROUP BY house.name_house;
```

Status	Duration
starting	0.000101
Executing hook on transaction	0.000002
starting	0.000005
checking permissions	0.000003
checking permissions	0.000001
checking permissions	0.000001
checking permissions	0.000002
Opening tables	0.000102
init	0.000005
System lock	0.000010
optimizing	0.000014
statistics	0.000086
preparing	0.000072
Creating tmp table	0.000125
executing	0.000256
end	0.000004
query end	0.000003
waiting for handler commit	0.000020
closing tables	0.000010
freeing items	0.000091
cleaning up	0.000013

```
ALTER TABLE study ADD INDEX idx_study_ID_course (ID_course);
```

L'index créé nous permet de réduire le temps d'exécution des requêtes :

Status	Duration
starting	0.000114
Executing hook on transaction	0.000002
starting	0.000007
checking permissions	0.000003
checking permissions	0.000001
checking permissions	0.000001
checking permissions	0.000003
Opening tables	0.000112
init	0.000004
System lock	0.000010
optimizing	0.000021
statistics	0.000090
preparing	0.000074
Creating tmp table	0.000179
executing	0.000297
end	0.000006
query end	0.000003
waiting for handler commit	0.000030
closing tables	0.000011
freeing items	0.000089
cleaning up	0.000017

Requête d :

Cette requête sélectionne les noms et les adresses email des étudiants qui ont le même nombre de cours que tous les autres étudiants de leur année. Les résultats sont obtenus en rejoignant trois sous-requêtes.

La première sous-requête sélectionne l'ID de l'étudiant, l'ID de l'année et le nombre de cours distincts suivis par chaque étudiant.

La deuxième sous-requête sélectionne l'ID de l'année et le nombre de cours distincts suivis par tous les étudiants de chaque année.

La troisième sous-requête relie les résultats de la première et de la deuxième sous-requête pour obtenir les noms et les adresses e-mail des étudiants qui ont le même nombre de cours que tous les autres étudiants de leur année.

```
SET profiling = 1;
SELECT s.student_name, s.email
FROM student s
JOIN (
SELECT ID_student, ID_year, COUNT(DISTINCT ID_course) AS
num_courses
FROM study
GROUP BY ID_student, ID_year
) AS sub
ON s.ID_student = sub.ID_student AND s.ID_year = sub.ID_year
JOIN (
SELECT ID_year, COUNT(DISTINCT ID_course) AS num_courses
FROM study
GROUP BY ID_year
) AS total
ON s.ID_year = total.ID_year AND sub.num_courses =
total.num_courses
WHERE sub.num_courses = total.num_courses;
SHOW PROFILE;
```

	Status	Duration
▶	starting	0.000140
	Executing hook on transaction	0.000004
	starting	0.000008
	checking permissions	0.000004
	checking permissions	0.000002
	checking permissions	0.000004
	Opening tables	0.000233
	end	0.000005
	query end	0.000006
	closing tables	0.000008
	freeing items	0.000097
	cleaning up	0.000013

```
ALTER TABLE study ADD INDEX idx_study_ID_student_year_course (ID_student, ID_year, ID_course);
```

L'index créé va permettre de réduire le temps d'exécutions général des requêtes.

	Status	Duration
▶	starting	0.000126
	Executing hook on transaction	0.000003
	starting	0.000007
	checking permissions	0.000003
	checking permissions	0.000001
	checking permissions	0.000004
	Opening tables	0.000058
	end	0.000002
	query end	0.000005
	closing tables	0.000008
	freeing items	0.000086
	cleaning up	0.000011

3. Partie 2 -Les vues

1. Expliquer à quoi sert une vue en base de données. Expliquer quelle est la différence entre une vue logique et une vue matérialisée.

Une vue en base de données est une représentation virtuelle d'une partie ou de l'ensemble des données stockées dans une base de données. Elle permet de présenter des informations spécifiques aux utilisateurs tout en masquant des données sensibles ou confidentielles. En d'autres termes, elle permet d'accéder aux données d'une manière simplifiée et structurée, en limitant l'accès à certaines informations selon les permissions accordées aux utilisateurs.

Il existe deux types de vues en base de données : **la vue logique et la vue matérialisée.**

Une vue logique est une vue virtuelle qui est créée au moment où elle est demandée, à partir d'une requête qui interroge les données stockées dans la base de données. Elle permet de simplifier l'accès aux données en présentant une vue personnalisée qui répond aux besoins des utilisateurs. La vue logique est mise à jour automatiquement lorsque les données sous-jacentes sont modifiées.

Une vue matérialisée, quant à elle, est une vue physique qui stocke les résultats d'une requête dans une table distincte. Contrairement à la vue logique, la vue matérialisée nécessite un traitement préalable pour sa création et son actualisation. Elle est utile lorsque la requête de base est complexe ou lorsque la vue est souvent utilisée, car elle permet d'améliorer les performances d'accès aux données. Cependant, elle nécessite une mise à jour manuelle ou automatisée pour refléter les changements apportés aux données sous-jacentes.

2. Trouver les requêtes SQL afin de :

a. Créer une vue logique qui affiche le nom, l'email et la maison de chaque étudiant qui suit un cours de potions.

```sql
CREATE VIEW view_potions_students AS
SELECT student_name, email, name_house
FROM student
INNER JOIN study ON student.ID_student = study.ID_student
INNER JOIN registered_course ON study.ID_course = registered_course.ID_course
INNER JOIN house ON student.ID_house = house.ID_house
WHERE name_course = 'potion';
```

b. Afficher le résultat de la vue.

```sql
SELECT * FROM view_potions_students;
```

student_name	email	name_house
Aiden Ortiz	aiden.ortiz@poudlard.edu	Gryffondor
Emma Romalas	emma.romalas@poudlard.edu	Gryffondor
Levi Patel	levi.patel@poudlard.edu	Gryffondor
Samuel Adams	samuel.adams@poudlard.edu	Gryffondor
Mason Wilson	mason.wilson@poudlard.edu	Gryffondor
Ava Sarda	ava.sarda@poudlard.edu	Gryffondor
Liam Brown	liam.brown@poudlard.edu	Gryffondor
Sebastian Cook	sebastian.cook@poudlard.edu	Gryffondor
Ava Jones	ava.jones@poudlard.edu	Serpentard
Madison Wright	madison.wright@poudlard.edu	Serpentard
Victoria Bailey	victoria.bailey@poudlard.edu	Serpentard
Penelope Price	penelope.price@poudlard.edu	Serpentard
Noah Misonal	noah.misonal@poudlard.edu	Serpentard
Aria Flores	aria.flores@poudlard.edu	Serpentard
Isabella Moore	isabella.moore@poudlard.edu	Serpentard
Emma Smith	emma.smith@poudlard.edu	Serpentard
John Doe	john.doe@example.com	Serpentard
Connor Gutierrez	connor.gutierrez@poudlard.edu	Poufsouffle
Olivia Plea	olivia.plea@poudlard.edu	Poufsouffle
William Davis	william.davis@poudlard.edu	Poufsouffle
Owen Wood	owen.wood@poudlard.edu	Poufsouffle
Nicholas Ross	nicholas.ross@poudlard.edu	Poufsouffle
Caleb Howard	caleb.howard@poudlard.edu	Poufsouffle
Henry Rogers	henry.rogers@poudlard.edu	Poufsouffle
Noah Johnson	noah.johnson@poudlard.edu	Poufsouffle
Jane Smith	jane.smith@example.com	Poufsouffle
Luna Kim	luna.kim@poudlard.edu	Serdaigle
Chloe Reed	chloe.reed@poudlard.edu	Serdaigle

c. Rajouter 2 étudiants qui suivent un cours de potion.

```
INSERT INTO student (ID_student, student_name, email, ID_house, ID_year) VALUES
('80','Hugo p', 'hugo.doe@example.com', 2, 1),
('81','Morgan s', 'morgan.smith@example.com', 3, 2);

INSERT INTO study (ID_student, ID_course) VALUES
(80 , 1),
(81 , 1);
```

d. Afficher (encore) le résultat de la vue.

	Hugo p	hugo.doe@example.com	Serpentard
	Isabella Moore	isabella.moore@poudlard.edu	Serpentard
	Jane Smith	jane.smith@example.com	Poufsouffle
	John Doe	john.doe@example.com	Serpentard
	Levi Patel	levi.patel@poudlard.edu	Gryffondor
	Liam Brown	liam.brown@poudlard.edu	Gryffondor
	Liam Zune	liam.zune@poudlard.edu	Serdaigle
	Luna Kim	luna.kim@poudlard.edu	Serdaigle
	Madison Wright	madison.wright@poudlard.edu	Serpentard
	Mason Wilson	mason.wilson@poudlard.edu	Gryffondor
	Morgan s	morgan.smith@example.com	Poufsouffle

3. Modification interdite d'une vue :

Créer une vue house_student_count qui regroupe les étudiants par maison et compte le nombre d'étudiants dans chaque maison.

Il y aura donc 2 colonnes : une colonne house_name avec le nom des maisons et une colonne student_count avec le nombre d'étudiants par maison.

Cette vue va utiliser des fonctions d'agrégation (COUNT et GROUP BY).

Essayer de modifier la colonne contenant le nombre d'étudiants dans une maison. Par exemple, pour la maison Gryffondor, définir le nombre d'étudiants à 10.

Est-ce que cette requête génère une erreur ? Et si la vue house_student_count avait été une table normale, est-ce que cette requête aurait fonctionné ?

Une vue est une représentation virtuelle des données stockées dans les tables sous-jacentes, et ne peut pas être modifiée directement.

Si la vue house_student_count avait été une table normale, la requête UPDATE aurait fonctionné normalement.

```
CREATE VIEW house_student_count AS
SELECT house.name_house AS house_name, COUNT(*) AS student_count
FROM student
INNER JOIN house ON student.ID_house = house.ID_house
GROUP BY house.name_house;

UPDATE house_student_count SET student_count = 10 WHERE house_name = 'Gryffondor';
-- Error Code: 1288. The target table house_student_count of the UPDATE is not updatable
```

3. Partie 3 – Procédure stockée et trigger

1. Expliquer à quoi sert les procédures stockées et les triggers.

Les procédures stockées sont des programmes SQL précompilés et stockés dans la base de données pour être appelés par les applications ou par d'autres procédures.

Elles permettent d'exécuter des instructions SQL complexes de manière répétée et efficace.

Les triggers sont des instructions SQL qui s'exécutent automatiquement en réponse à des événements spécifiques, tels que l'insertion, la mise à jour ou la suppression de données dans une table.

Ils peuvent être utilisés pour appliquer des contraintes de données, effectuer des validations ou mettre à jour des tables de manière automatique.

2. Création d'une vue matérialisée pour house_student_count à l'aide d'une procédure stockée sur MySQL

a. Créez une table house_student_count_materialized qui sera notre vue matérialisée

```sql
CREATE TABLE house_student_count_materialized (
  house_name VARCHAR(255) NOT NULL,
  student_count INT NOT NULL,
  PRIMARY KEY (house_name)
);
```

b. Créez une procédure stockée pour rafraîchir la vue matérialisée house_student_count_materialized

```sql
-- b
DELIMITER //
CREATE PROCEDURE refresh_house_student_count_materialized()
BEGIN
    -- Vide la table house_student_count_materialized
    TRUNCATE TABLE house_student_count_materialized;

    -- Insère les données recalculées
    INSERT INTO house_student_count_materialized (house_name, student_count)
    SELECT house.name_house AS house_name, COUNT(student.ID_student) AS student_count
    FROM house
    LEFT JOIN student ON house.ID_house = student.ID_house
    GROUP BY house.name_house;

END //

DELIMITER ;
```

c. Exécutez la procédure stockée pour rafraîchir la vue matérialisée

```
-- c
CALL refresh_house_student_count_materialized();

-- Maintenant, la table house_student_count_materialized contient les mêmes informations que la vue house_student_count.
-- On peut exécuter des requêtes sur cette table comme on le ferait sur une vue matérialisée.
-- Exemple :
SELECT * FROM house_student_count_materialized;
```

house_name	student_count
Gryffondor	11
Poufsouffle	10
Serdaigle	7
Serpentard	10
NULL	NULL

3. Mise à jour de la vue matérialisée house_student_count_materialized

a. Ajoutez un nouvel étudiant à la table students

```
-- a Voici une requête pour ajouter un nouvel étudiant à la table students :

SELECT * FROM house_student_count_materialized; -- 8 à gryffondor
INSERT INTO student (student_name, email, ID_house, ID_year)
VALUES ('Ron', 'rone@hogwarts.edu', 1, 1);
```

b. Affichez le contenu de la table house_student_count_materialized pour vérifier si le nouvel étudiant a été pris en compte

```
SELECT * FROM house_student_count_materialized;-- TOUJOURS 8 à gryffondor
```

house_name	student_count
Gryffondor	11
Poufsouffle	10
Serdaigle	7
Serpentard	10
NULL	NULL

c. Exécutez la procédure stockée refresh_house_student_count_materialized() pour mettre à jour les données de la vue matérialisée house_student_count_materialized

```
-- c Exécutez la procédure stockée
CALL refresh_house_student_count_materialized();
```

d. Affichez à nouveau le contenu de la vue matérialisée house_student_count_materialized pour vérifier si le nouvel étudiant a été pris en compte après l'exécution de la procédure stockée

```
-- d Affichez à nouveau le contenu de la vue matérialisée
SELECT * FROM house_student_count_materialized;-- 9 maintenant à gryffondor
```

house_name	student_count
Gryffondor	12
Poufsouffle	10
Serdaigle	7
Serpentard	10
NULL	NULL

4. Création de triggers sur la vue matérialisée house_student_count_materialized

a. Créez un trigger AFTER INSERT pour mettre à jour automatiquement la vue matérialisée house_student_count_materialized chaque fois qu'un étudiant est ajouté dans la base de données

```
DELIMITER //
CREATE TRIGGER tr_student_insert
AFTER INSERT ON student
FOR EACH ROW
BEGIN
  CALL refresh_house_student_count_materialized();
END//
DELIMITER ;
```

Il est déclenché après chaque ajout d'un enregistrement dans la table student grâce à l'instruction AFTER INSERT ON student FOR EACH ROW.

b. Créez un trigger AFTER DELETE pour mettre à jour automatiquement la vue matérialisée house_student_count_materialized chaque fois qu'un étudiant est supprimé dans la base de données

```
DELIMITER //
CREATE TRIGGER tr_student_delete
AFTER DELETE ON student
FOR EACH ROW
BEGIN
    CALL refresh_house_student_count_materialized();
END//
DELIMITER ;
```

Il est déclenché après chaque suppression d'un enregistrement dans la table student grâce à l'instruction AFTER DELETE ON student FOR EACH ROW.

5. Tester les triggers de la vue matérialisée house_student_count_materialized

a.b.c Affichez le contenu de la table house_student_count_materialized avant d'effectuer des modifications. Insérez un nouvel étudiant dans la table students. Affichez le contenu de la table house_student_count_materialized après l'insertion pour vérifier si le trigger AFTER INSERT a fonctionné

```
SELECT * FROM house_student_count_materialized;
INSERT INTO student (student_name,email, ID_house,ID_year) VALUES ('Ginny Weasley','ginny.w@hogward.edu', 1, 3);
SELECT * FROM house_student_count_materialized; -- Error Code: 1422. Explicit or implicit commit is not allowed in stored function or trigger.
```

d. Supprimez l'étudiant précédemment inséré de la table students e. Affichez le contenu de la table house_student_count_materialized après la suppression pour vérifier si le trigger AFTER DELETE a fonctionné

Cela n'a pas fonctionné avec les triggers. Nous avons donc rajouté une procédure avec les étudiants qui permet de créer et supprimer les étudiants.

```
DROP TABLE house_student_count_materialized;
DROP PROCEDURE refresh_house_student_count_materialized;
DROP TRIGGER tr_student_insert;
DROP TRIGGER tr_student_delete;
DROP PROCEDURE add_student;
```

Ajout de la procédure :

```
DELIMITER //
CREATE PROCEDURE add_student (
  IN student_name VARCHAR(255),
  IN email VARCHAR(255),
  IN ID_house INT,
  IN ID_year INT
)
BEGIN
  INSERT INTO student (student_name, email, ID_house, ID_year)
  VALUES (student_name, email, ID_house, ID_year);
  CALL refresh_house_student_count_materialized();
END;//
DELIMITER ;

DELIMITER //
CREATE PROCEDURE supp_student (
  IN ID INT
)
BEGIN
  DELETE FROM student where ID_student = ID;
END;//
DELIMITER ;

CALL supp_student(58);
CALL refresh_house_student_count_materialized();

SELECT * FROM house_student_count_materialized;

CALL add_student('Ginny Weasley', 'ginny.w@hogward.edu', 1, 3);
```

house_name	student_count
Gryffondor	12
Poufsouffle	10
Serdaigle	7
Serpentard	10
NULL	NULL

IV. Conclusion

La réalisation de ce projet en base de données sur MySQL Workbench nous aura permis d'appliquer les différentes notions vues en cours de TD et CM.

Grâce à la préparation minutieuse des données, la normalisation du schéma, l'utilisation d'index, de vues, de procédures stockées et de triggers, nous avons pu concevoir une base de données efficace et fonctionnelle.

Ces différentes techniques ont permis d'optimiser les performances de la base de données, de garantir la cohérence et l'intégrité des données stockées, ainsi que de faciliter la gestion et l'exploitation des informations.

Malgré quelques difficultés rencontrées, notamment sur le transfert des données dans notre base de données normalisée, nous avons pu répondre aux besoins spécifiques du projet, en proposant une solution fonctionnelle.

Ce projet a été une belle opportunité pour mettre en pratique et renforcer nos connaissances en base de données et ainsi nous préparer pour l'examen final qui nous attend.

Enfin, nous avons pris plaisir à relever les différents défis proposés par ce projet et nous sentons prêt à en relever de nouveau.