الأقسام التحضيرية للمدارس العليا
تازة

المملكة المغربية
ⵜⴰⴳⵍⴷⵉⵜ ⵏ ⵍⵎⵖⵔⵉⴱ

وزارة التربية الوطنية
والتعليم الأولي والرياضة
ⵜⴰⵎⴰⵡⴰⵙⵜ ⵏ ⵜⵣⵔⴼⵜ ⵏⵜⵎⵙⵡⵔⵜ
ⴷ ⵜⵎⵎⴰⵔⵜ ⴰⵎⴰⵡⴰⵙ ⴷ ⵜⵣⵍⵣⵓⵜ

Base de Données

Yussuf Daghuj

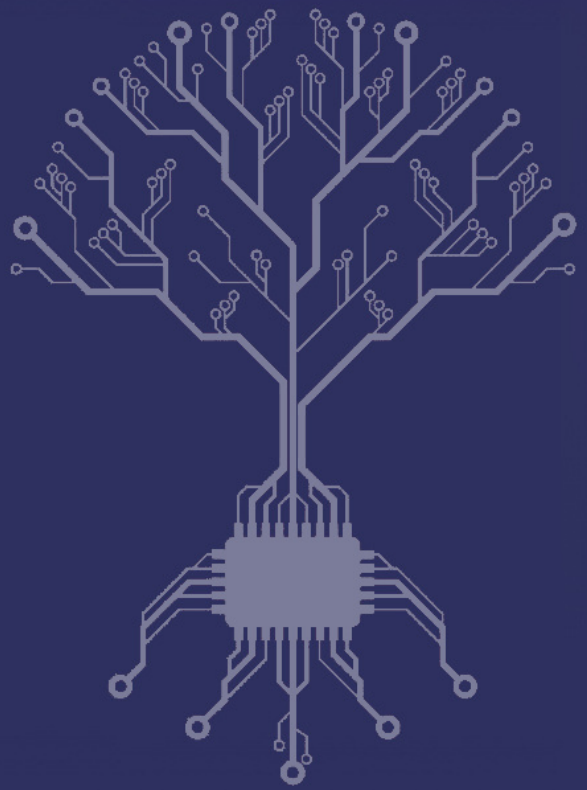
15 Novembre 2024

1 Introduction

2 Définitions

3 SQL
- Langage de Définition de Données (LDD)
- Langage de Manipulation de Données (LMD)

Introduction



Objectif

Regrouper, ordonner, et mettre à disposition un grand nombre de données, en vue d'une utilisation par des experts (études statistiques ...) ou des particuliers (renseignement ...).

Problème

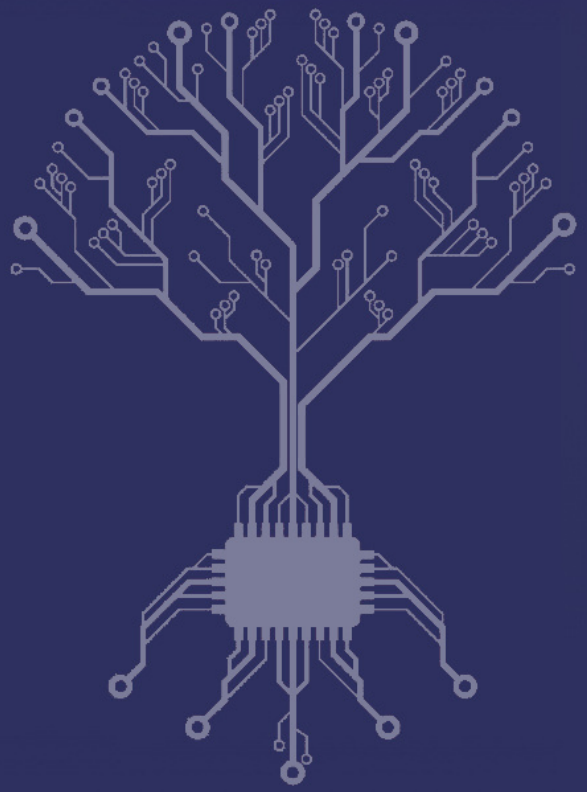
Les données peuvent être éparpillées sur différents supports : papiers, mémoires binaires, cerveau, ...

Solution

Utiliser des formes spéciales qui permettent de structurer les données de façon à pouvoir en extraire facilement ce qui nous intéresse et d'avoir la mémoire nécessaire pour le stockage des données sans redondances.

La structure de base de données relationnelle répond à cette question.

Définitions



Base de Données

Une Base de Données (BD) est un ensemble structuré permettant le stockage de grandes quantités de données (représentant des informations du monde réel), avec le moins de redondance possible, afin d'en faciliter l'exploitation.

Système de Gestion de Base de Données

Un SGBD est un logiciel qui permet d'interagir avec une BD pour : stocker, décrire, et manipuler les données, tout en garantissant leur cohérence et leur sécurité.

Cycle de vie d'une base de données

- Conception : Déterminer le futur contenu de la BD
- Implantation : Tansmettre le contenu de la BD au logiciel du SGBD choisi comme (SQLite), ceci se fait via le LDD (Langage de Définition de Données).
- Utilisation : Interrogation de la BD, ajout, suppression, et modification au moyen du LMD (Langage de Manipulation de Données)

L'organisation des données au sein d'une base de données a une grande importance pour la facilité de leur manipulation. On s'intéresse à ce qu'on appelle bases de données relationnelles (modèle relationnel).

Selon le modèle relationnel, la base de données est composée d'un ensemble de relations (table 2D) dans lesquelles sont placées les données, ainsi que les liens entre ces différentes relations.
Exemples :

idstudent	firstname	lastname	field	average	idschool
24142356	ahmad	bachir	economy	15	12340
24167849	omar	alaoui	economy	14	12340
24167843	fatima	zahra	math	18	12340

Table – Student

idschool	name	city
12340	Charif Alidrissi	Taza
12347	Ibn Yassmine	Taza

Table – School

Les tables ne représentent qu'une abstraction de l'enregistrement physique des données en mémoire.

Définitions

- Base de données relationnelle : Ensemble de tables contenant des données reliées.
- Table (relation) : Ensemble de données relatives à un même objet. Elle est structurée sous forme d'un tableau qui contient un ensemble fini d'enregistrements.
- Ligne (enregistrement) : Une occurrence de l'objet représentée par la table.
- Attribut (colonne de la table) : Une propriété élémentaire de l'objet décrite par une table.

Définitions

- Schéma de relation : Précise le nom d'une relation et la liste de ses attributs avec leurs domaines.

Exemple

Student(Identifiant de l'étudiant : entier, nom : chaîne de caractères, prénom : chaîne de caractères, filière : chaîne de caractères, moyenne : réel)

- Schéma de la BD relationnelle : L'ensemble des schémas des relations composantes et un ensemble de contraintes d'intégrité.

Définitions

- Clé : Ensemble d'attributs qui permettent d'identifier un enregistrement de la relation.

Exemples

Le couple (Nom, Prénom) peut être une clé de la relation Étudiant, de même pour l'attribut numéro de l'étudiant.

Définitions

- Clé primaire (Identifiant) : Une clé choisie parmi les clés possibles de la relation, de sorte qu'elle soit la plus simple et qu'elle soit de valeur unique et non vide.

Exemple

L'attribut numéro de l'étudiant est la clé primaire de la relation Étudiant.

Grâce à cette clé primaire on pourra indexer les données de la relation, et aussi créer des liens entre plusieurs relations.

- Clés étrangères : Un ensemble d'attributs qui sont des clés primaires dans d'autre tables, et qui permettent de créer des liens entre ces tables toutes en garantissant la cohérence des données. Un lien entre deux tables se traduit donc par l'ajout dans une table d'un champ correspondant à la clé primaire de l'autre table.

Contraintes d'intégrité

Une contrainte d'intégrité est une règle appliquée à une colonne (attribut) ou une table (relation) et qui doit être toujours vérifiée, afin de garantir la cohérence du schéma relationnel de la base de données.

Le SGBD contrôle les contraintes d'intégrité de chaque table.

Contraintes d'intégrité

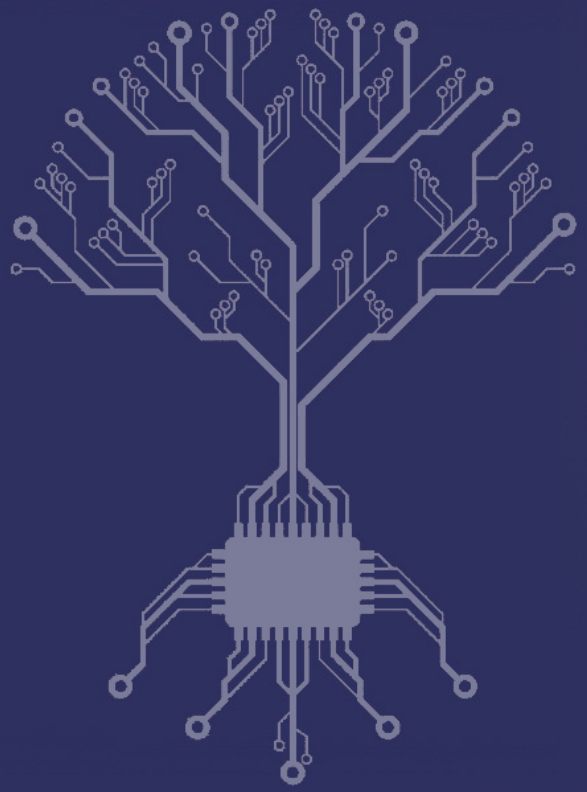
- Les contraintes de domaine : Les valeurs d'un attribut restent dans son domaine.

Exemple

Si l'attribut identifiant d'étudiant est défini sur un domaine de valeurs numériques, il ne peut pas contenir de lettres.

- Les contraintes d'intégrité des tables permettent de s'assurer que chaque table possède obligatoirement une clé primaire qui doit être unique (pas de doublons) et non vide (toujours spécifiées).
- Les contraintes d'intégrité référentielle permettent de s'assurer que les valeurs introduites dans l'attribut clé étrangère d'une table fille figure déjà dans la table mère.

SQL

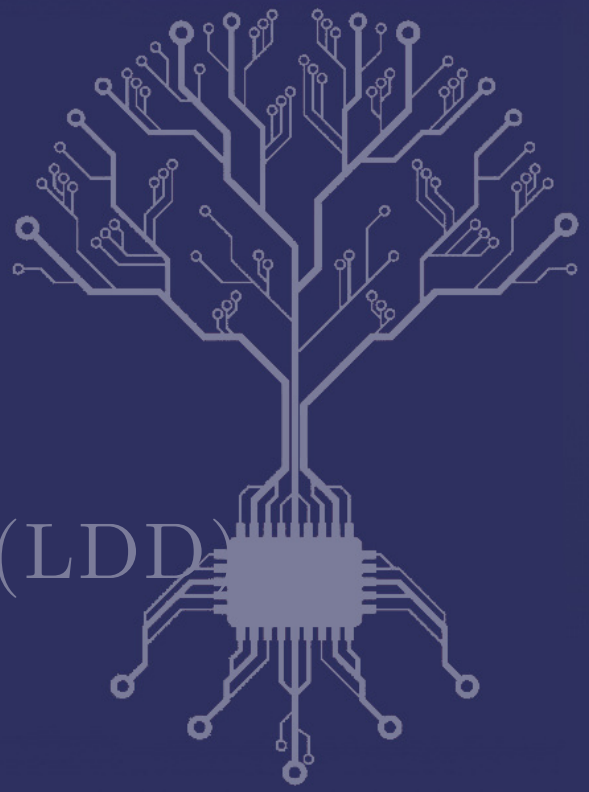


Une fois nous avons défini le schéma relationnel de notre base de données relationnelle, cette dernière peut être implémentée sous un SGBD à l'aide d'un langage appelé SQL (Structured Query Language).

SQL est un langage informatique normalisé servant à exploiter des bases de données relationnelles. Il comporte plusieurs langages comme le :

- Langage de Définition de Données (LDD)
- Langage de Manipulation de Données (LMD)

SQL :
Langage de Définition de Données (LDD)



LDD

Le LDD permet de créer, modifier, ou supprimer une table.

Création

La création d'une table consiste à définir :

- Le nom de table
- Les colonnes de la table (leurs noms et types)
- Les contraintes d'intégrité de la table

Création

Syntaxe

```
CREATE TABLE [IF NOT EXISTS] nomTable (

nomColonne1 type contrainteColonne1

nomColonne2 type contrainteColonne2

...

contrainteTable1,

contrainteTable2,

...

);
```

Création : Types de données

- TEXT : Chaîne de caractères
- INTEGER : Entier signé
- REAL : Nombre décimal à virgule flottante

Création : Contraintes de colonne

- UNIQUE : Toutes les valeurs de la colonne doivent être différentes ou NULL.
- NOT NULL ou NULL : Interdit (NOT NULL) ou autorise (NULL) l'insertion de valeur NULL pour cet attribut.
- PRIMARY KEY : Désigne l'attribut comme clé primaire de la table (équivalente à la contrainte UNIQUE NOT NULL)
- DEFAULT valeur : Permet de spécifier la valeur par défaut de la colonne

Création : Contraintes de colonne

Exemple

Soit la table school avec l'attribut nom de type chaîne de caractère qui représente la clé primaire de la table et l'attribut adresse de type chaîne de caractères. Créer la table en indiquant qu'on ne peut pas trouver deux écoles à la même adresse :

```
CREATE TABLE school (
name TEXT PRIMARY KEY,
address TEXT UNIQUE
);
```

Création : Contraintes de colonne

- CHECK (condition) : Vérifie lors de l'insertion des enregistrements (lignes) que l'attribut vérifie la condition.

  Les opérateurs de comparaison :

  - $=, !=, <, >, <=, >=$
  - BETWEEN value1 AND value2
  - IN ou NOT IN (value1, value2, ...)
  - LIKE : permet d'utiliser des jokers (% pour une chaîne de caractères de longueur quelconque et _ pour un seul caractère)

Exemple

```
CREATE TABLE student (
idstudent INTEGER PRIMARY KEY,
firstname TEXT NOT NULL ,
lastname TEXT NOT NULL ,
average REAL NOT NULL CHECK (average>0 AND average<20),
mail TEXT
);
```

Création : Contraintes de table

Les contraintes de tables portent sur plusieurs attributs de la table sur laquelle elles sont définies :

- PRIMARY KEY (colonne1, colonne2, ...) : Désigne la concaténation des attributs cités comme clé primaire de la table.
- FOREIGN KEY (colonne) REFERENCES table (colonne) : Identifie une ou plusieurs colonnes comme étant clé étrangère.

Création : Contraintes de colonne

Exemple

Soit les schémas :

- professor(idprof, firstname, lastname, mail)
- course(idcourse, name, field)
- teaches(#idprof, #idcourse, date)

Création : Contraintes de colonne

Exemple

```
CREATE TABLE professor(
idprof INTEGER PRIMARY KEY,
firstname TEXT NOT NULL ,
lastname TEXT NOT NULL ,
mail TEXT NOT NULL
);
```

Création : Contraintes de colonne

Exemple

```
CREATE TABLE course (
idcourse INTEGER PRIMARY KEY,
name TEXT NOT NULL ,
field TEXT NOT NULL
);
```

Création : Contraintes de colonne

Exemple

```
CREATE TABLE teaches(
idprof INTEGER,
idcourse INTEGER,
date TEXT,
PRIMARY KEY (idprof, idcourse),
FOREIGN KEY (idprof) REFERENCES professor (idprof),
FOREIGN KEY (idcourse) REFERENCES course (idcourse)
);
```

Modification

- Ajout d'une colonne
  - Syntaxe : ALTER TABLE nomTable ADD (nomColonne type contraintes) ;
  - Exemple : ALTER TABLE student ADD (age INTEGER) ;
- Renommage d'une colonne
  - Syntaxe : ALTER TABLE nomTable RENAME ancienNom TO nouveauNom
- Suppression d'une colonne
  - Syntaxe : ALTER TABLE nomTable DROP nomColonne
  - Exemple : ALTER TABLE student DROP age

Suppression

Supprimer une table revient à éliminer sa structure et toutes les données qu'elle contient avec la commande DROP TABLE nomTable.

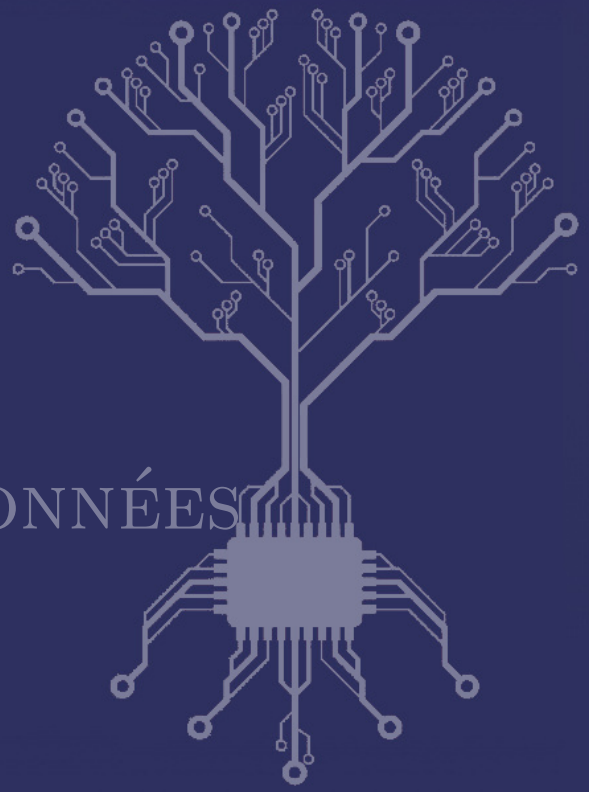
Modification et Suppression

La modification ou suppression d'une ligne dans une table pourra être impossible s'il existe des lignes dans d'autres tables référençant la clé primaire de cette ligne. Pour éviter ce genre de problème, on peut ajouter à la contrainte FOREIGN KEY des options :

Modification et Suppression

- En cas de suppression :
  - ON DELETE CASCADE : supprimer les lignes concernées
  - ON DELETE SET NULL : mettre à NULL des valeurs des clés étrangères
- en cas de modification :
  - ON UPDATE CASCADE : modifier les valeurs des clés étrangères
  - ON UPDATE SET NULL : mettre à NULL des valeurs des clés étrangères

SQL : Langage de Manipulation de Données (LMD)



C'est l'ensemble des commandes qui permettent l'ajout, la modification et la suppression de lignes.

Insertion de lignes

La commande INSERT permet d'insérer une ligne dans une table en spécifiant les valeurs à insérer.

Syntaxe

```
INSERT INTO nomTable VALUES
(valeur11, valeur12, . . . ),
(valeur21, valeur22, . . . ),
... ;
```

Insertion de lignes

Exemple

```
INSERT INTO student VALUES
(24142356, "Ahmad", "Bachir", 15, "ahmad@mail.com", 12340),
(24167849, "Omar", "Alaoui", 16, "omar@mail.com", 12340),
(241677834, "Fatima", "Zahra", 19, "fatima@mail.com", 12340) ;
```

Modification de lignes

La commande UPDATE permet de modifier les valeurs d'une ou plusieurs colonnes, dans une ou plusieurs lignes, selon une condition.

Syntaxe

UPDATE nomTable SET nomColonne = expression, ... WHERE condition ;

En l'absence d'une clause WHERE, toutes les lignes sont mises à jour.

Modification de lignes

Exemple

Soit la table employee dont les colonnes sont : idemployee, name, salary, departement. Augmenter de 10% le salaire des employés du département commercial :

```
UPDATE employee
SET salary = salary * 1.1
WHERE departement = "commerce" ;
```

Suppression de lignes

Syntaxe

DELETE FROM nomTable WHERE condition ;

Exemple

DELETE FROM employee WHERE departement = ”commerce” ;

Interrogation

Pour interroger une base de données on utilise la commande SELECT. Elle permet d'effectuer, sur les tables, des opérations de l'algèbre relationnel :

- Récupérer certaines colonnes d'une table (projection)
- Récupérer certaines lignes d'une table en fonction de leur contenu (sélection)
- Combiner des informations venant de plusieurs tables (jointure)

- La commande SELECT permet de :
  - récupérer certaines colonnes (SELECT) de certaines tables (FROM)
  - récupérer certaines lignes (WHERE)
  - regrouper certaines lignes (GROUP BY)
  - filtrer après le regroupement (HAVING)
  - trier les résultats (ORDER BY)
  - limiter le nombre d'enregistrements retournés (LIMIT)

Syntaxe

SELECT colonne1, colonne2, ... FROM table

ou bien

SELECT DISTINCT colonne1, colonne2, ... FROM table

Le mot clé DISTINCT permet de supprimer les doublons.

Exemple

Soit le modèle relationnel : student (id, firstname, lastname, field, average, school, city, phone).

Exemples

- Pour afficher toutes les informations des élèves :

  SELECT * FROM student ;

- Pour afficher les noms et prénoms des élèves :

  SELECT firstname, lastname FROM student ;

- Pour afficher les villes des élèves :

  SELECT DISTINCT city FROM student ;

Syntaxe

SELECT colonne1, colonne2, ... FROM table WHERE conditions

Exemples

- Afficher les noms et prénoms des élèves qui habitent à 'Taza' :
  - SELECT firstname, lastname FROM student WHERE city= 'Taza' ;
- Afficher les noms et les numéros des élèves qui habitent à 'Meknes' ou à 'Fes' :
  - SELECT firstname, lastname, phone FROM student WHERE city= 'Meknes' OR city='Fes' ;
  - SELECT firstname, lastname, phone FROM student WHERE city IN ('Meknes' ,'Fes') ;

Exemples

- Afficher les noms et les prénoms des élèves dont la moyenne est supérieure à 14 :
  - SELECT firstname, lastname FROM student WHERE average >= 14 ;

Jointure

Requête SQL

SELECT * FROM Table1 JOIN Table2 ON Table1.colonne1= Table2.colonne2 ;

Ou bien

SELECT * FROM Table1, Table2 WHERE Table1.colonne1= Table2.colonne2 ;

Exemples

- Soient les tables :
  - prof (idprof, firstname, lastname, email)
  - student (idstudent, firstname, lastname, city, email, #idprof)
- Afficher les noms des élèves dont le professeur est monsieur 'X' :
  - SELECT prof.lastname FROM student JOIN prof ON student.idprof=prof.idprof WHERE prof.lastname='X' ;
  - SELECT prof.lastname FROM student, prof WHERE student.idprof=prof.idprof AND prof.lastname='X' ;

- SQL dispose aussi d'un ensemble de fonctions d'agrégation comme :
  - AVG : Calcule la moyenne
  - COUNT : Calcule le nombre de lignes
  - MAX : Calcule la valeur maximale
  - MIN : Calcule la valeur minimale
  - SUM : Calcule la somme des valeurs

On peut faire une sélection après l'agrégation, en ajoutant une condition à laquelle chaque groupe doit répondre.

Requête SQL

SELECT colonne1, colonne2, ..., fonction1(colonne11), fonction2(colonne21), ... FROM table GROUP BY colonne1, colonne2, ... HAVING condition ;

Remarque

Chaque colonne figurant dans SELECT doit figurer dans GROUP BY Sur le résultat, on trouve qu'une seule ligne pour chaque groupe. Toutes les fonctions d'agrégation excluent par défaut les valeurs NULL avant de travailler sur les données Lorsqu'on a le choix, il vaut mieux sélectionner en amont (WHERE) qu'en aval (HAVING), pour n'effectuer l'agrégation que sur un nombre minimal de lignes.

Exemple

Soit la table : student (idstudent, firstname, lastname, city, phone, noteBac, classe) Afficher la note de bac maximale des élèves de chaque classe
SELECT MAX(noteBac) FROM student GROUP BY classe ;

Exemple

Afficher pour chaque classe le nombre d'élèves, mais seulement pour les classes dont la moyenne de la note du bac est supérieure à 14 SELECT classe, COUNT(*) AS numberStudent FROM student GROUP BY classe HAVING AVG(noteBac)¿14 ;

Il est possible de trier les données sur une ou plusieurs colonnes par ordre ascendant ou descendant (Ordre ascendant par défaut).

Requête SQL

```
ORDER BY colonne1 [ASC — DESC], colonne2 [ASC — DESC], ... ;
```

Exemple

Soit la table : student (idstudent, firstname, lastname, city, phone, noteBac, classe) Afficher la note de bac moyenne de chaque classe, et trier le résultat par ordre décroissant de la note SELECT classe, AVG(NoteBac) AS moyenne FROM student GROUP BY classe ORDER BY moyenne DESC ;

Pour limiter le nombre de lignes retournées par une requête, on utilise la commande :

Requête SQL

```
SELECT ... FROM ... LIMIT nombreLigne OFFSET nombreSaut ;
```

qui permet de limiter le nombre de résultats à nombreLigne, et de sauter nombreSaut lignes avant de renvoyer le résultat.

Exemple

object (idobjet, label, price) Afficher les noms des 3 plus chers articles :
SELECT label FROM object ORDER BY price DESC LIMIT 3 ;

Une requête imbriquée est une sous requête exécutée à l'intérieur d'une autre requête SQL. Elle remplace souvent une constante au sein d'une clause WHERE ou HAVING. Remarque : Les sous-requêtes sont souvent utilisées avec l'instruction SELECT. Toutefois, vous pouvez également les utiliser dans une instruction INSERT, UPDATE ou DELETE ou dans une autre sous-requête

Requête SQL

DELETE FROM ... WHERE [expression] opérateur (SELECT. . . FROM . . .) ;

Requête SQL

INSERT INTO nomTable(colonne1,. . . .., colonneN) SELECT . . . FROM . . . WHERE . . . ;

Requête SQL

UPDATE . . . SET colonne1 = (SELECT . . . FROM . . .) . . . WHERE [expression] opérateur (SELECT . . . FROM . . .) ;

Exemple

Soit la table : student (idstudent, firstname, lastname, city, phone, noteBac, classe) Afficher les élèves dont la note du bac est supérieur ou égale à la moyenne des notes de bac de tous les élèves SELECT * FROM student WHERE noteBac ¿= (SELECT AVG(noteBac) FROM student) ;

Exemple

Soit la table : student (idstudent, firstname, lastname, city, phone, noteBac, classe) Afficher l'élève qui a la meilleure note de bac SELECT * FROM student WHERE noteBac ¿= ALL (SELECT noteBac FROM student) ;

Exemple

Afficher l'élève qui a la meilleure note de bac SELECT * FROM student WHERE noteBac = (SELECT MAX (noteBac) FROM student) ;

Exemple

book (idbook, title, autor) borrow (#idstudent, #idbook) Afficher les élèves qui ont empruntés le livre dont idlivre est 50 SELECT * FROM student WHERE idstudent IN (SELECT idstudent FROM borrow WHERE idbook=50) ;

Exemple

Afficher les élèves qui ont emprunté le livre dont idbook est 50 SELECT * FROM student WHERE EXISTS (SELECT * FROM borrow WHERE idbook=50 AND student.idstudent=borrow.idstudent) ;