Introduction

- Historique:
  - SGF (systèmes de gestion de fichiers) vers le début des années 60.
  - Chaque application a ses propres fichiers.
  - Si plusieurs applications utilisent le même fichier, alors la description de celui-ci doit se faire au niveau de chacune des applications.
  - Redondance accrue des données
  - Difficulté de mise à jours
- L'expression base de données apparaît pour la première fois en 1964. Fort intérêt pour le traitement d'un gros volume d'information
  - Les données sont indépendantes des programmes
  - La redondance est réduite
  - La mise à jour est facilitée
  - Les accès aux données sont contrôlés
  - Nous avons une intégrité des données.

Introduction

- Le modèle hiérarchique: (structure en arbre)
  - Dans le modèle hiérarchique, les données sont classées selon une arborescence descendante. Les enregistrements sont connectés les uns aux autres par des liens. On utilise des pointeurs pour le parcours de l'arbre.
    - Problème: les données n'ont pas nécessairement une structure hiérarchique dans le monde réel.
    - Pour trouver une information, parfois il faudra parcourir tout l'arbre
- Le modèle Réseau:
    - Vient palier aux problèmes du modèle hiérarchique
    - Système de pointeurs pour accéder aux données.
    - Les liens ne sont pas uniquement hiérarchiques

Introduction

- Le modèle relationnel: début des années 70
  - Définit par le Britannique Edgar Frank Codd et basé sur l'algèbre relationnelle.
  - Les données sont organisées dans des tables ou des relations.
  - Les liens qui existent entre les relations est exprimé sous forme de clé étrangère ou clé primaire.
  - C'est le modèle le plus répandu de nos jours.
  - Les SGBDs relationnels utilisent le langage SQL pour manipuler les données de la base de données.

Introduction

- Exemples de SGBDR: (dans l'ordre de popularité
  - Oracle, MYSQL
  - SQL server, (MSSQL)
  - PostgreSQL
  - DB 2
  - SQLite
- Autres SGBD
  - MangoDB (orienté documents)
  - Cassandra (orienté colonnes)

  Source: https://db-engines.com/en/ranking

Introduction

- Définitions :
  - Une base de données est un ensemble de données modélisant les objets d'une partie du monde réel et se servant de support à une application informatique.
  - Un SGBD (système de gestion de base de données) peut-être perçu comme un ensemble de logiciels système permettant aux utilisateurs d'insérer, de modifier et de rechercher efficacement des données spécifiques dans une grande masse d'information partagée par de multiples utilisateurs.

Introduction

- Rôles des SGBDs
  - Non redondance des données : permet de réduire le risque d'incohérence lors des mises à jour, de réduire les mises à jour et les saisies.
  - Partage des données : ce qui permet de partager les données d'une base de données entre différentes applications et différents usagers.
  - Cohérence des données : ce qui permet d'assurer que les règles auxquelles sont soumises les données sont contrôlées surtout lors de la modification des données.
  - Sécurité des données : ce qui permet de contrôler les accès non autorisés ou mal intentionnés. Il existe des mécanismes adéquat pour autoriser, contrôler ou d'enlever des droits à n'importe quel usager à tout ensemble de données de la base de données.

Introduction

Présentation du SGBDR Oracle

- Architecture:
  - Noyau: assure la gestion de l'intégrité des données, le stockage des données, les accélérateurs (index, clusters), gestion des connexions à la BD et l'exécution des requêtes.
  - Le dictionnaire: décrit de manière dynamique les objets et les utilisateurs de la base de données.
  - La couche SQL: interface d'accès aux objets. Tous les accès se font en utilisant le langage SQL
  - La couche PL/SQL: extension de la couche SQL.

Introduction

- Présentation du SGBDR oracle
  - Les objets:
    - Les tables
    - Les vues
    - Les synonymes
    - Les users
    - Les séquences
    - Les déclencheurs
    - Les procédures
    - Les fonctions
    - Les packages
    - …

Introduction

- Définitions:
  - Une table:
    - objets contenant les données des utilisateurs ou appartenant au système. Une table est composée de colonnes (Champs ou attributs) et de lignes (enregistrements ou occurrences).
    - Exemples: Table Joueurs

Nom	Prénom	Nb_Points
Patoche	Alain	44
Zouba	Alex	88
Lafleur	Serge	54
Patoche	Peter	102

- Définitions:
  - Dans la table joueurs:
    - **Une ligne** représente un enregistrement de la table. (Patoche, Alain, 44) est un enregistrement.
    - **Les colonnes** Nom, prénom et Nb_points sont des attributs de la table joueurs. Ils sont également appelés «champs»
    - Lafleur est une valeur de l'attribut NOM.
  - Remarque: dans la table joueurs, il existe deux enregistrements ayant la même valeur pour l'attribut nom. Afin de distinguer les enregistrements nous introduisons la notion de clé primaire ou PRIMARY KEY.

Numero	Nom	Prénom	Nb_Points
21	Patoche	Alain	44
55	Zouba	Alex	88
93	Lafleur	Serge	54
78	Patoche	Peter	102

Introduction

- Définitions:
  - Une clé primaire est un attribut d'une table permettant d'identifier chaque enregistrement de manière unique.
  - SQL pour Structured Query Language est un langage de définition de données (DDL, Data Definition Language), de manipulation de données (DML, Data Manipulation Language) et de contrôle de données (DCL, Data Control Language)
  - DDL: CREATE, ALTER , DROP
  - DML: INSERT INTO, UPDATE, DELETE
  - DCL: GRANT, REVOKE

Introduction

- **Conseils Généraux pour le langage SQL:**
  - SQL n'est pas sensible à la casse, cependant il est conseillé d'utiliser les mots réservés (commandes, le type de données …) en majuscules
  - Il ne faut pas oublier le point virgule à la fin de chaque ligne de commande.
  - Utiliser les deux traits -- pour mettre une ligne en commentaire
  - Utiliser /* et */ pour mettre plusieurs lignes en commentaire
  - Utiliser des noms significatifs pour les objets que vous créez
  - Ne pas utiliser de mots réservés comme noms d'objets (tables, vue, colonne..)
  - Mettre une clé primaire pour chacune des tables que vous créez
  - Si vous avez à contrôler l'intégrité référentielle, alors il faudra déterminer l'ordre dans lequel vous allez créer vos tables.

SQL

La commande SELECT:

La commande SELECT est la commande le plus simple à utiliser avec SQL. Cette commande n'affecte en rien la base de données et permet d'extraire des données d'une ou plusieurs tables. La syntaxe simplifiée n'utilise pas de jointure et elle se présente comme suit :

SELECT <nom_de colonne1,…nom_de_colonnen>

FROM <nom_de table>

WHERE <condition>

ORDER BY <nom_de_colonne>;

Remarque: **le résultat d'une requête SELECT est une table.**

La clause WHERE permet de fixer la condition de sélection (utilisée dans le UPDATE et le DELETE)

Quelques opérateurs de comparaison

=, !=, <, <=, >, >=, Like, IN, IS NULL, BETWEEN x AND Y …

(voir page 20 du document pour tous les opérateurs de comparaison)

Les opérateurs logiques OR et AND peuvent être utilisés pour combiner deux conditions.

SQL

La commande SELECT:

- La clause ORDER BY spécifie le tri des données après extraction. Si l'ordre de tri n'est pas précisé alors le tri est par défaut croissant.
- Pour avoir un tri décroissant il faut ajouter l'option DESC.
- Le tri peut se faire selon plusieurs colonnes, il faut les séparer par des virgules
- Dans la commande SELECT, le joker * indique que toutes les colonnes seront sélectionnées.
- L'option AS permet de changer le nom de colonnes pour l'affichage uniquement.

SQL

La commande SELECT: Exemples

```
SELECT * FROM employes;

---------------------------------

SELECT empno AS numero, ename, job
FROM EMPLOYES;

---------------------------------

SELECT empno, ename, job
FROM EMPLOYES ORDER BY ename, job;
```

SQL

```
SELECT empno, ename, job
FROM EMPLOYES ORDER BY ename DESC, job;
--------------
SELECT * FROM EMPLOYES ORDER BY 1,2;
```

Le tri se fait selon la colonne 1, puis la colonne 2

```
--------------------------------
SELECT empno, ename, job
FROM EMPLOYES
WHERE SAL >2500
ORDER BY ename ;
```

SQL

Exercice : Voici le contenu de la table Etudiants:

NUMAD	NOM	Prenom	Adresse	CodeProg	DateNaissance
123	Fafar	Chritiane	12 Victor Mirabel	SCH	1995-01-03
124	Roy	Stéphane	23 de Jupiter Montréal	INFO	1996-05-03
345	Fafar	Martin	234 Mars Montréal	SCN	1994-08-04
356	Pronovost	William	23 Cartier Laval	INFO	1996- 02-07
321	Patoche	Alain	2345 Labelle Laval	SIM	1997—09-01

SQL

- Pour cette table, donner un exemple de: enregistrement, attribut, d'occurrence de l'attribut NOM.
- Écrire les requêtes suivantes:
  - lister tous les enregistrements de votre table Etudiants;
  - lister tous les étudiants en Info
  - sélectionner le non, prénom et adresse ordonnés par NOM
  - sélectionner le non, prénom et adresse ordonnés par NOM puis par prénom
  - Afficher tous les étudiants dont le nom commence par P
  - Sélectionner tous les étudiants de Montréal
  - Lister tous les étudiants
  - Lister les étudiants nés entre 1994 et 1996

Semaine 2

- Retour sur la semaine dernière.
  - Point de vue de l'étudiant.
  - Point de vue de l'enseignant.
- Plan de séance:
  - Rappels
  - Les instructions DML (insert, update, delete)
  - La commande de création de table.
  - La commande ALTER TABLE.
- Conclusion.

INSERT INTO

- INSERT INTO <nom_de_table> VALUES (<liste de valeurs>);

INSERT INTO EmployesInfo VALUES (20,'Fafar','Patrice',40000);

- INSERT INTO <nom_de_table>(<nom_de_colonne>) VALUES (<liste_de_valeurs>);

INSERT INTO EmployesInfo (NumEmp, NOM) VALUES (12,'Lebeau');

INSERT INTO

- Cette commande permet de saisir des données dans une table **une rangée à la fois.**
- Le type numérique (NUMBER) est saisi en notation standard. La virgule décimale est remplacée par un point lors de la saisie
- Une valeur de type caractère (CHAR ou VARCHAR2) doit être mise entre apostrophes. Si la chaine de caractère contient des apostrophes, ceux-ci doivent être doublés.
- Il est possible d'utiliser des séquences pour l'insertion automatique d'un numéro séquentiel pour une colonne

La commande UPDATE

- La commande UPDATE permet d'effectuer des modifications des données sur une seule table. Cette modification peut porter sur une ou plusieurs lignes. (Enregistrement)
- Lors de la modification des données, les contraintes d'intégrité doivent être respectées.

UPDATE

UPDATE <nom_de _table> SET
<nom_de_colonne>=<nouvelle_valeur>;

UPDATE employesinfo SET salaire = salaire
+(salaire*0.5)
WHERE nom like 'Faf%';

La commande DELETE

La commande DELETE permet de supprimer de la base de données une ou plusieurs rangées d'une table.

DELETE FROM <nom_de _table>

WHERE <condition>;

DELETE FROM employesinfo WHERE NOM IN ('Blues','Simpson');

Avant toute opération **COMMIT**, faire un SELECT afin de vérifier que nous n'avons pas fait des suppressions par erreur. Utiliser un **ROLLBACK** dans le cas d'une suppression par erreur.

DELETE

- Faire un SELECT avant le DELETE
- Respecter les contrainte d'intégrité lors d'un DELETE
- Il est impossible de supprimer une valeur de la clé primaire si cette valeur est référée par une valeur de la clé étrangère sauf si l'option ON DELETE CASCADE est définie

La commande CREATE TABLE

La commande CREATE TABLE permet de créer une table avec toutes les contraintes d'intégrité.

Avant de créer une table, il est important de connaître:

- Le nom de la table
- Les noms des colonnes
- Les types (number, varchar2, date ….) des colonnes
- Les contraintes d'intégrité sur les colonnes
- Les contraintes sur les tables.
- L'ordre de création des tables, si vous en avez plusieurs

Exemple

```
CREATE TABLE Produit (
codeProduit NUMBER CONSTRAINT pk1 PRIMARY KEY,
NomProduit VARCHAR2(20) NOT NULL,
description VARCHAR2 (30)
)
;
```

La Commande CREATE TABLE

- Les contraintes:
  - NOT NULL: indique que la valeur de la colonne ou de l'attribut est obligatoire (par défaut NULL)
  - UNIQUE:valeur unique pour l'attribut (champs) . Pas de doublons
  - DEFAULT: indique la valeur par défaut que prendra l'attribut si aucune valeur n'est saisie.
  - CHECK: Indique les valeurs permises qui peuvent être saisies pour la colonne (champ ou attribut) lors de l'entrée des données ou une condition à laquelle doit répondre une valeur insérée. La condition doit impliquer le nom d'au moins une colonne.
  - PRIMARY KEY
  - FOREIGN KEY
- Exemple:

La contrainte PRIMARY KEY

- Permet de définir une clé primaire sur la table.
- La contrainte de PRIMARY KEY assure également les contraintes de NOT NULL et UNIQUE
- Lorsque la clé primaire est composée, il faut la définir au niveau de la table (voir plus loin)

ALTER TABLE

- Ajout d'une nouvelle colonne à la table avec ses contraintes
- Augmente ou diminuer la largeur d'une colonne existante
- Changer la catégorie d'une colonne, d'obligation à optionnelle ou vice versa (NOT NULL à NULL ou vice versa)
- Spécification d'une valeur par défaut pour une colonne existante
- Changer le type de données d'une une colonne existante
- Spécification d'autres contraintes pour une colonne existante
- Activer ou désactiver une contrainte
- Détruire une contrainte.

ADD , MODIFY et DROP

ADD

ALTER TABLE Employes ADD Salaire NUMBER (8,2);

ALTER TABLE Questions ADD CONSTRAINT Questions_PK
PRIMARY KEY (NumQuestion) ;

MODIFY

ALTER TABLE Employes MODIFY (nom NOT NULL);

DROP

ALTER TABLE Employes DROP Primary Key;

ALTER TABLE Employes DROP CONSTRAINT emppk**;**

DROP et RENAME

DROP TABLE personne;

DROP TABLE personne CASCADE CONSTRAINTS;

RENAME <Ancien_nom> TO <Nouveau_nom>;

RENAME Employes TO EmployesInfo;

Semaine 3

- Retour sur la semaine 2
  - Point de vue de l'étudiant
  - Point de vue de l'enseignant.
- Plan de séance
  - Définition d'une clé étrangère et intégrité référentielle.
  - Clé primaire composée.
  - Requêtes avec jointures.

La contrainte de **FOREIGN KEY**

- Définition: c'est une contrainte indique que la valeur de l'attribut correspond à une valeur d'une clé primaire de la table spécifiée.
- La clé primaire de l'autre table (Table A, exemple EQUIPES) doit être obligatoirement crée pour que cette contrainte soit acceptée.
- La clé primaire de l'autre table (Table A, exemple EQUIPES) et l'attribut défini comme clé étrangère (dans la Table B, exemple JOUEUR) doivent être de même type et de même longueur

La contrainte de **FOREIGN KEY** exemple

```
CREATE TABLE programme (codePrg VARCHAR2(3)
CONSTRAINT pk3 PRIMARY KEY,
nomProg VARCHAR2(20));

CREATE TABLE etudiants
(
NumAd NUMBER CONSTRAINT pk4 PRIMARY KEY,
codePrg VARCHAR2(3),
Nom VARCHAR2(20),
Prenom VARCHAR2(20),
CONSTRAINT fk1 FOREIGN KEY(codePrg) REFERENCES
programme (codePrg)
)
;
```

Autre exemple

```
CREATE TABLE EMPLACEMENT
(
CODE_EMPLACEMENT CHAR(4) CONSTRAINT CEPK PRIMARY KEY,
LIEU VARCHAR2(15)
)
;

CREATE TABLE CATEGORIE (CODE_CATEGORIE CHAR(5) CONSTRAINT
    CCPK PRIMARY KEY,
DESCRIPTION VARCHAR2(15)
)
;
```

Autre exemple (suite)

```
CREATE TABLE PRODUITS
(
CODE_PRODUIT CHAR(4) CONSTRAINT CDPK PRIMARY KEY,
NOM_PRODUIT VARCHAR2(20), QUANTITE NUMBER,STOCKLIMITE
    NUMBER,
CODE_EMPLACEMENT CHAR(4),
CODE_CATEGORIE CHAR(5),
CONSTRAINT CEFK FOREIGN KEY (CODE_EMPLACEMENT) REFERENCES
    EMPLACEMENT (CODE_EMPLACEMENT),
CONSTRAINT CCFK FOREIGN KEY (CODE_CATEGORIE) REFERENCES
    CATEGORIE (CODE_CATEGORIE)
)
;
```

Clé primaire composée.

- Il arrive qu'une table ait besoin d'une clé primaire composée pour pouvoir identifier les enregistrements. Dans la plus part des cas, ces deux attributs qui composent la clé sont issus deux autres tables. La table qui contient la clé primaire composée est appelée «table de relation».
- Exemple:

Exemple

La table Étudiants

NumAd (PK)	Nom	prenom
10215	Pigeon	Sébastien
12356	Fafar	Chantale
32332	Leblanc	Éric
22222	Desjardins	Christian

La table cours

Code_Cours (PK)	Description
420-KED	Conception de bases de données
420-KEG	Gestion de réseaux.

Exemple suite

La table Resultats

NumAd	Code_Cours	Note
10215	420-KED	75
10215	420-KEG	70
22222	420-KEG	75
32332	420-KEG	67

Dans la table Resultats, pour identifier clairement et de manière unique les enrergistrements (les notes des étudiants), nous avons besoin du code cours et du numéro d'admission. Ce couple d'attributs ne peut être dupliqués et ne peut être nul. Il a la caractéristique d'un clé primaire.

Création d'une table avec la clé primaire composée.

- (***)On suppose que les deux tables Etudiants et cours sont déjà créées.

```
CREAT E TABLE resultat
(
Numad number(8) ,
Code_cours CHAR (10),
Note number (5,2),
Constraint fknumad foreign key (numad) references etudiants(numad),
Constraint fkcours foreign key (code_cours) references
    cours(code_cours),
CONSTRAINT pkcompose PRIMARY KEY (Numad, code_cours)
)
;
```

Semaine 4

- Retour sur la semaine 3
  - Point de vue de l'étudiant.
  - Point de vue de l'enseignant.
- Plan de séance
  - Requêtes avec Jointures:
  - Suite atelier 3.

Requête avec jointures(1)

Une jointure est une opération relationnelle qui sert à chercher des lignes ou des enregistrements à partir de deux ou plusieurs tables disposant d'un ensemble de valeur communes, en général les clés primaires.

La jointure simple ou INNER JOIN : c’est une jointure entre deux ou plusieurs tables qui retourne des résultats uniquement si la condition (égalité des clés primaires) est vérifiée.

Requête avec jointures(2)

- Lorsqu'un attribut sélectionné est présent dans plus d'une table alors il faut le précéder du nom de la table à partir de laquelle on désire l'extraire.
- Vous pouvez donner un alias aux noms de tables afin de faciliter la référence aux tables. Cependant si un alias est donné alors, il faudra utiliser l'alias à la place du nom de la table.
- Les attributs qui apparaissent dans la jointure ne sont pas nécessairement dans le SELECT.
- Toutes les tables dont les attributs apparaissent dans la clause SELECT ou dans la clause WHERE doivent apparaître dans la clause FROM.

Requête avec jointures(3)

Exemple1:

```
SELECT  nomEtudiant,
   PrenomEtudiant,NomProgramme
FROM  Etudiants E C INNER JOIN Programmes P
ON  P.codeprogramme = E.codeProgramme;
```

Requête avec jointures(3)

Exemple2:

```
select ename, job, sal, dname
from (syemp inner join sydept
on syemp.deptno = sydept.deptno);
```

Exemple 3:, cette instruction va renvoyer une erreur car deptno est dans les deux tables syemp et sydept

```
select ename, job, sal, loc
from (syemp inner join sydept on syemp.deptno = sydept.deptno)
    where deptno = 10;
```

Il faut écrire where sydept.deptno = 10;

Requête avec jointures(4)

- Exemple4:

```
SELECT nom,prenom,  description, note
FROM  ((etudiant E  INNER JOIN  NOTE  ON
   E.numad = R.numad)
INNER JOIN cours C ON C.code_cours =
   R.code_cours)
```

Requête avec jointures(5)

La jointure externe: Dans le cas d'une jointure externe, il faut faire suivre la colonne pour laquelle il n'est pas obligatoire d'avoir des lignes correspondant à l'égalité par l'opérateur LEFT OUTER JOIN ou RIGHT OUTER JOIN

Cette requête ramène tous les étudiants y compris ceux qui ne sont pas inscrits dans un programme

```
SELECT NOM, PRENOM, NOMPROG
FROM ETUDIANTS E LEFT OUTER JOIN PROGRAMME P ON
    E.CODEPRG=P.CODEPRG;
```

Requête avec jointures(6)

Autre exemple

Cette requête ramène tous les programmes y compris ceux qui n'ont pas d'étudiants inscrits

```
SELECT NOM, PRENOM, NOMPROG
FROM ETUDIANTS E RIGHT OUTER JOIN
    PROGRAMME P ON E.CODEPRG=P.CODEPRG;
```

Quelques fonctions SQL utilisées avec la commande SELECT

- **Les fonctions MIN et MAX**

SELECT MAX (NOTE) FROM RESULTATS WHERE CODE_COURS ='KED';

- **Les fonctions AVG et SUM**

SELECT AVG (NOTE) FROM RESULTATS WHERE CODE_COURS ='KED';

- **La fonction COUNT**

SELECT COUNT(*)

FROM ETUDIANTS;

Quelques fonctions SQL (suite)

- **Les clauses GROUP BY et HAVING**

cette clause permet d'indiquer au système de regrouper des enregistrements selon des valeurs distincts qui existent pour les colonnes spécifiées.

La clause HAVING permet de mieux cibler les enregistrements spécifiés

EXEMPLE

```
SELECT CODEPRG, COUNT(CODEPRG)
FROM ETUDIANTS
GROUP BY CODEPRG;


SELECT CODEPRG, COUNT(CODEPRG)
FROM ETUDIANTS
GROUP BY CODEPRG
HAVING CODEPRG ='420';
```

Exercice

La table employés contient les colonnes suivantes:

Numemp, nom, prenom, job.

Écrire une requête SQL qui permet d'afficher le nombre d'employés par JOB.

JOB	Total
Analyste	3
Commis	5
Designer	2

Semaine 5

Les sous requêtes

Atelier 5, exercice 2

Début du Tp no1

Les sous-requêtes(1)

Une sous requête est une requête avec la commande SELECT imbriquée avec les autres commandes (UPDATE, INSERT, DELETE et CREATE)

Une sous-requête peut être utilisée dans les clauses suivantes :

- La clause WHERE d'une instruction UPDATE,DELETE et SELECT
- La clause FROM de l'instruction SELECT
- La clause VALUES de l'instruction INSERT INTO
- La clause SET de l'instruction UPDATE
- L'instruction CREATE TABLE.

Les sous-requêtes(2)

- Dans la clause WHERE:

Ce type de sous-requête permet de comparer une valeur de la clause WHERE avec le résultat retourné par une sous-requête. Dans ce cas on utilise les opérateurs de comparaison suivants :

=, != ,< ,<= ,> ,>= ,et IN.

Les sous-requêtes(3)

- Exemple :

```
SELECT NUMAD FROM RESULTATS
WHERE  CODE_COURS='KED' AND  NOTE <
                (SELECT NOTE FROM RESULTATS
                WHERE NUMAD=100 AND
                CODE_COURS='KED');
```

Cette requête ramène les numéro d'admission (NUMAD)des étudiants dont la note en KED est plus petite que celle de l'étudiant dont le numéro d'admission est 100 pour le même cours (CODE_COURS=KED)

Les sous-requêtes(4)

- Quelle est-la requête qui ramène les noms des étudiants et leurs numéros et qui fait la même chose que la requête précédente ?

- Réponse:

Les sous-requêtes(5)

```
SELECT E.NUMAD, E.NOM,R.NOTE
FROM RESULTATS R INNER JOIN ETUDIANTS E
ON  E.NUMAD= R.NUMAD
WHERE CODE_COURS='KED'
AND  NOTE <
                (SELECT NOTE FROM RESULTATS
                WHERE NUMAD=100 AND
                CODE_COURS='KED');
```

Les sous-requêtes(6)

L'opérateur IN

IN est utilisée lorsque la sous requête retourne plus qu'une rangée :

Exemple

```
SELECT NUMAD,NOM, PRENOM
FROM ETUDIANTS
WHERE NUMAD NOT IN
                (SELECT NUMAD FROM RESULTATS);
```

Les sous-requêtes(7)

Question :

Comment avoir tous les étudiants qui sont dans le même programme que l'étudiant dont le nom est PATOCHE

Les sous-requêtes(8)

Réponse

```
SELECT * FROM ETUDIANTS
WHERE CODEPRG = (SELECT CODEPRG FROM
ETUDIANTS WHERE NOM='PATOCHE');
```

Les sous-requêtes(9)

Question:

Comment peut-on avoir les étudiants (nom et prénoms) qui ont la même note que l'étudiant numéro 100 et dans le cours KED

Les sous-requêtes(10)

Réponse

```
SELECT nom, prenom  FROM ETUDIANTS
WHERE NUMAD IN
        (SELECT NUMAD FROM RESULTATS
      WHERE CODE_COURS ='KED' AND NOTE =
            (SELECT NOTE FROM RESULTATS
            WHERE CODE_COURS='KED' AND
            NUMAD =100));
```

Les sous-requêtes(11)

ANY et ALL

IN est utilisée lorsque la sous requête retourne plus qu'une rangée :

Exemple

```
SELECT NUMAD,NOM, PRENOM
FROM ETUDIANTS
WHERE NUMAD NOT IN
                (SELECT NUMAD FROM RESULTATS);
```

Les sous-requêtes(12)

ALL On utilise l'operateur ALL pour que la comparaison se fasse pour toutes les valeurs retournée. Le résultat est vrai si toutes les valeurs répondent à la comparaison

Les sous-requêtes(11)

L'opérateur EXISTS

Cet opérateur est utilisé dans une sous-requête pour vérifier l'existence ou non d'enregistrements dans la sous-requête.

Exemple:

```
SELECT * FROM clients WHERE  numclient =10
AND EXISTS (SELECT numclient FROM commande WHERE
    numclient=10);
```

Si le numéro client 10 a commandé des articles(est dans la table commande) alors on ramène les informations qui lui corresponde.

Les sous-requêtes: La création de table

Ce type de sous requête permet de créer une table à partir d'une autre table. La nouvelle table contient les valeurs de la sous-requête

```
CREATE TABLE NOTEKED
    (NUMADMISSION,NOTEKED)
AS SELECT NUMAD,NOTE FROM RESULTATS
    WHERE CODE_COURS='KED';
```

Question :Quelle-est la requête qui permet de créer une table NOTESKED qui va avoir comme attributs NOMETUDIANT, PRENOMETUDIANT,TITRECOURS,NOTEKED

Les sous-requêtes: La création de table

Réponse

```
CREATE TABLE NOTESKED (NOMETUDIANT,
    PRENOMETUDIANT,TITRECOURS,NOTEKED)AS
SELECT NOM, PRENOM,TITRE,NOTE
FROM ((ETUDIANTS E INNER JOIN RESULTATS R  ON
    E.NUMAD=R.NUMAD) INNER JOIN COURS CR ON
    R.CODE_COURS=CR.CODE_COURS AND ))
WHERE
R.CODE_COURS='KED' ;
```

Les sous-requêtes: SET de UPDATE

Dans ce cas le résultat retourné par la sous-requête doit contenir une seule rangée.

```
UPDATE RESULTATS SET NOTE =
  (SELECT AVG(NOTE) FROM RESULTATS WHERE
    CODE_COURS ='KEG')
WHERE NOTE <60 AND CODE_COURS ='KEG';
```

Les sous-requêtes: INSERT INTO

Ce type de sous-requête permet d'insérer des données dans une table à partir d'une autre table. La sous requête est utilisée à la place de la clause VALUES de la requête principale et peut retourner plusieurs résultats

```
INSERT INTO COURS_DU_SOIR
(SELECT * FROM COURS);
```

Les sous-requêtes: Dans la clause FROM

Exemple.

```
SELECT NUMSTAGE, DESCRIPTION, ADRESSEENT
FROM infostage ,
(SELECT nument, ADRESSEENT FROM
   entreprises WHERE ADRESSEENT IS NOT
   NULL)E
WHERE infostage.nument = E.nument;
```

Les sous-requêtes: Dans la clause FROM

```
SELECT EMPNO,E.JOB, ENAME, SAL/SOMME
FROM SYEMP E, (SELECT JOB, SUM(SAL) AS SOMME
FROM SYEMP GROUP BY JOB) V
WHERE V.JOB= E.JOB;
```

Cette requête ramène les noms des employés ainsi que la proportion de leur salaire par rapport à leur JOB

Requêtes SELECT avec les opérateurs d'ensembles

- Trois opérateurs: INTERSECT, UNION, MINUS.
- Principe:
  - Le nombre de colonnes renvoyées par SELECT 1 doit être le même que celui renvoyé par SELECT 2
  - Le type de données SELECT 1 doit être le même que celui de SELECT 2
  - La clause optionnelle ORDER BY doit se faire selon un numéro de colonne et non le nom de colonne
  - SELECT1 et SELECT2 ne peuvent contenir le clause ORDER BY
- exemples

Requêtes SELECT avec les opérateurs d'ensembles

Table Fournisseurs

	NUMFOURNISSEURS	NOMFOURNISSEUR	ADRESSE
1	13	ALAIN PATOCHE	444 RUE DE LA LUNE MONTREAL, QC
2	10	LE MAGNIFIQUE	23 BOULEVARD ST LAURENT BLAIVILLE,QC
3	11	QUEBECINC	444 RUE DE LA LUNE QUEBEC, QC
4	12	LAVALINC	123 RUE SAINT-GEORGES LAVAL, QC

Table Clients

	NUMCLIENT	NOMCLIENT	ADRESSE
1	11	LE ROI DES SINGES	123 RUE SAINT-GEORGES MONTREAL, QC
2	12	LE RIGOLOT	34 RUE PILON LAVAL, QUEBEC
3	13	ALAIN PATOCHE	444 RUE DE LA LUNE MONTREAL, QC
4	10	LE MAGNIFIQUE	23 BOULEVARD ST LAURENT BLAIVILLE,QC
5	20	MR BEAN	10 AVENUE DE L'HUMOUR LONDRES
6	21	SIMPSON	AVENUE DE LA BÉTISE, SPRINGFIELD

Requêtes SELECT avec les opérateurs d'ensembles

- Opérateur INTERSECT:

```
SELECT NOMCLIENT, ADRESSE
FROM CLIENTS
INTERSECT
SELECT NOMFOURNISSEUR, ADRESSE
FROM FOURNISSEURS
ORDER BY 1;
```

A pour résultat les fournisseurs 10 et 13: les fournisseurs qui sont aussi des clients.

Requêtes SELECT avec les opérateurs d'ensembles

- Opérateur UNION:

```
SELECT NOMCLIENT, ADRESSE
FROM CLIENTS
UNION
SELECT NOMFOURNISSEUR, ADRESSE
FROM FOURNISSEURS
ORDER BY 1;
```

A pour résultat TOUS les fournisseurs et tous les clients. Si des doublons sont trouvés, ils sont ramenés une seule fois

Pour ramener toutes les lignes, il faut utiliser l'opérateurs ALL

Requêtes SELECT avec les opérateurs d'ensembles

	NOMCLIENT	ADRESSE
1	ALAIN PATOCHE	444 RUE DE LA LUNE MONTREAL, QC
2	LAVALINC	123 RUE SAINT-GEORGES LAVAL, QC
3	LE MAGNIFIQUE	23 BOULEVARD ST LAURENT BLAIVILLE,QC
4	LE RIGOLOT	34 RUE PILON LAVAL, QUEBEC
5	LE ROI DES SINGES	123 RUE SAINT-GEORGES MONTREAL, QC
6	MR BEAN	10 AVENUE DE L'HUMOUR LONDRES
7	QUEBECINC	444 RUE DE LA LUNE QUEBEC, QC
8	SIMPSON	AVENUE DE LA BÉTISE, SPRINGFIELD

Requêtes SELECT avec les opérateurs d'ensembles

- Opérateur MINUS:

```
SELECT NOMCLIENT, ADRESSE
FROM CLIENTS
MINUS
SELECT NOMFOURNISSEUR, ADRESSE
FROM FOURNISSEURS
ORDER BY 1;
```

A pour résultat TOUS Les CLIENTS (résultat du premier SELECT) MOINS tous les fournisseurs (résultat du deuxième SELECT)

Requêtes SELECT avec les opérateurs d'ensembles

	NOMCLIENT	ADRESSE
1	LE RIGOLOT	34 RUE PILON LAVAL, QUEBEC
2	LE ROI DES SINGES	123 RUE SAINT-GEORGES MONTREAL, QC
3	MR BEAN	10 AVENUE DE L'HUMOUR LONDRES
4	SIMPSON	AVENUE DE LA BÉTISE, SPRINGFIELD