Bases de Données SQL Server et leur stockage

Introduction aux bases de donnés

Quelques définitions

Définitions courantes: Une base de données

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.

Les bases de données sont présentes dans la plupart des applications que nous utilisons régulièrement. Exemples:

- Les opérations bancaires (toutes les banques et les caisses)
- Les achats en ligne (Amazon)
- Les jeux vidéos pour garder les caractéristiques(force, habiletés etc..) et les items(armes, armures etc..) d'un personnage du jeu
- Etc….

Quelques définitions

Définitions courantes: Un SGBD

Un SGBD 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.

Exemples de SGBD:

SQL Server, MySQL, MS SQL Server, MongoDB, SQLite

Quelques notions théoriques

- Problèmes:
  - 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
- Solutions: Base de données: (la première fois en 1964)
  - 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 aux bases de données

- Quelques notions théoriques:

- 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.

Quelques notions théoriques

- Les SGBD qui gèrent les bases de données relationnelles sont appelés SGBD**R**.
- Les SGBDR sont les plus populaires de nos jours
- SQL Server est un SGBDR.

- Les SGBDR utilisent **SQL** (Structured Query Language) pour exploiter une base de données

Conclusion

Points clés:

- Les bases de données sont très utilisées dans le domaine de l'informatique.
- Les bases de données relationnelles utilisent des tables pour structurer et stocker les données. Il y a des **relations** entre les tables.
- SQL Server est le SGBDR (relationnel) qui sera utilisé dans ce cours.
- SQL est le langage que nous allons utiliser dans ce cours.
- La prochaine séance portera sur « introduction à SQL ».

Introduction aux bases de donnés

La commande SELECT

La commande SELECT

Plan de la séance

- Retour sur la dernière séance:
  - Point de vue de l'étudiant
  - Point de vue de l'enseignant.
- Rappel, bases de données relationnelles.
- SQL, Introduction
- La commande SELECT
- Exemples
- Laboratoire 1

Rappels

Bases de données relationnelles:

- Une base de données relationnelle et une base de dont les données sont stockées dans des objets appelés: Table.
- Les tables sont reliées entre elles.

Une table est un ensemble de lignes et de colonnes.

Les lignes sont aussi appelées : Enregistrements

Les colonnes sont aussi appelées: Attributs

SQL, Introduction

Définition: SQL

- SQL pour Structured Query Language ou langage structuré de requêtes, est un langage qui permet de définir, de manipuler et de contrôler une base de données relationnelle.
- Il permet également d'extraire les données d'une base de données.
- SQL est essentiellement un ensemble de commandes permettant d'exploiter une base de données relationnelles. Ces commandes peuvent-être regroupées comme suit:
    - La commande d'extraction des données: La commande SELECT
    - Les commandes qui permettent de définir la structure d'une table, qui permettent de créer ou de détruire des objets d'une base de données (comme une table). Ces commande s'appellent des commandes DDL (Data Definition Language). Parmi ces commandes nous avons CREATE, DROP, ALTER.
    - Les commandes qui permettent de modifier le contenu d'une table. Ces commandes sont appelées commandes DML (Data Manipulation Language). Parmi ces commandes nous avons: INSERT INTO, DELETE, UPDATE.

SQL, Introduction

Conseils Généraux

- 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, Introduction

Convention d’écriture

- Les <> indique une obligation
- Les () pourra être répété plusieurs fois, il faut juste les séparer par une virgule
- Les [] indique une option.

Attention ⚠ Les données à l’intérieur des tables sont sensibles à la case. YANICK est différent de Yanick

SQL, la commande SELECT

La commande SELECT

- La commande SELECT est la commande la 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>];
```

SQL, la commande SELECT

- La clause **WHERE** permet de cibler les lignes (enregistrements) à extraire
- 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 joueurs;

SELECT nom, prenom FROM joueurs;

SELECT nom, prenom FROM joueurs ORDER BY nom;

SELECT num AS Numéro, nom, prenom FROM joueurs ORDER BY nom,prenom;

SQL, la commande SELECT

La clause WHERE

Cette clause permet de restreindre, en utilisant des critères, les enregistrements qui seront affectés par la requête. En d'autres mots, la requête SELECT ramène des résultats si la **condition ou** les **conditions** sont vérifiées.

Exemples :

On affiche le nom et le prénom des joueurs, s'ils sont du Canadien de Montréal

On affiche le nom et le prénom des joueurs s'ils ont un salaire plus élevé que 1 000 000

On affiche on affiche le nom et le prénom des joueurs s'ils ont un salaire plus élevé que 1 000 000 ET qu'ils sont du canadien de Montréal.

SQL, la commande SELECT

Quelques opérateurs utilisés dans la clause WHERE

Opérateurs	Signification	Exemple
=	Égalité	SELECT nom, prenom FROM employes WHERE nom ='Patoche'; SELECT nom, prenom FROM employes WHERE salaire =50 000
<> ou != ou ^=	Différent	SELECT nom, prenom FROM employes WHERE nom ! ='Patoche';
>	Plus grand	SELECT nom, prenom FROM employes WHERE salaire >50 000;
>=	Plus grand ou égal	SELECT nom, prenom FROM employes WHERE salaire >=50 000;
LIKE	Si la valeur est comme une chaine de caractères. Le % est utilisé pour débuter ou compléter la chaine de caractère.	SELECT nom, prenom FROM employes WHERE nom LIKE 'Le%'; Va ramener tous les employés dont le nom commence par **Le**
IN	Égal à une valeur dans une liste. Ramène des résultats si la valeur de la condition est égale à au moins une des valeurs fournies par une liste.	SELECT * FROM EMPLOYES where nom in('Patoche','Leroy','Lejeune',LeBeau');
IS NULL	Si la valeur retournée est NULL (est vide). Attention NULL ne veut pas dire zéro.	SELECT nom, prenom FROM employes WHERE salaire is NULL; Ramène le nom et le prénom des employes qui N'ONT PAS de salaire.
BETWEEN x AND Y	Si la valeur est comprise entre x et y	SELECT * FROM employes WHERE salaire BETWEEN 9000 AND 16000 ;

SQL, la commande SELECT

Important:

- La liste complète des opérateurs est dans le document du cours.
- Nous avons également les opérateurs <, <=, NOT LIKE, NOT IN, IS NOT NULL, NOT BETWEEN x AND y
- Dans la clause WHERE, si la valeur de la condition est :
  - De type caractère (CHAR ou VARCHAR2), alors elle doit être mise entre apostrophes. Si la chaine de caractère contient des apostrophes, ceux-ci doivent être doublés.
  - De type numérique (NUMBER) alors la valeur est saisie en notation standard. La virgule décimale est remplacée par un point lors de la saisie
  - De type date alors elle doit être mise entre apostrophes. Cependant il faudra connaître le format DATE de votre système. Pour saisir une date dans n'importe quel format, il faut s'assurer de la convertir dans le format avec la fonction TO_DATE (à voir plus loin)
- Il est possible:
  - D'utiliser une expression arithmétique dans la clause WHERE à condition que la colonne sur laquelle porte la condition soit de type numérique. (Avec des sous-requêtes)
  - De combiner des opérateurs logiques (OR et AND) dans la clause WHERE.

Exemples

```
SELECT nom, prenom from joueurs where CODEEQUIPE ='MTL' AND salaire >=1000000;

SELECT nom, prenom from joueurs where codeequipe ='MTL' OR codeequipe ='OTT';
```

La commande SELECT

Conclusion

Questions

Introduction aux bases de donnés

La commande CREATE TABLE

Les commandes: CREATE TABLE INSERT INTO

Plan de la séance

- Rappel: définition d'une table
- La commande CREATE TABLE
  - Définitions
  - Types de données SQL Server
  - Les contraintes d'intégrités
  - Syntaxe
  - Exemples, de création de table

Rappels

- Une table est un ensemble de lignes et de colonnes.
- Une table est l'objet dans lequel les données sont stockées. C'est l'objet principal manipulé par un SGBDR
- Les lignes sont aussi appelées : Enregistrements
- Les colonnes sont aussi appelées: Attributs

La commande CREATE TABLE

Introduction:

- Tous les objets de la base de données sont créés avec la commande CREATE. Cette commande est une commande DDL. (Data Definition Laguage)

- La commande CREATE TABLE permet de créer une table dans une base de données.
- Pour créer une table nous avons besoin de connaître:
  - La liste des attributs de la table
  - Le type de données pour chaque attribut
  - Les contraintes d'intégrité, s'il y'en a, pour chaque attribut
  - Et l'ensemble des contraintes d'intégrité sur la table, s'il y'en a.
- Nous devons connaître aussi le nom de la table qui sera créée.

Les types de données SQL Server .

- Type de données: Dans tous les SGBDs (comme pour les langages de programmation) les données qui seront stockées ont un type. Pour SQL Server, voici les principaux types

Pour le reste des types de données, allez sur: https://docs.SQL Server.com/cd/B28359_01/server.111/b28318/datatype.htm#CNCPT183

Types	Significations , Exemples
VARCHAR2(**n**)	Chaîne de caractères de longueur variable. La taille maximale de cette chaîne est déterminée par la valeur**n** et peut atteindre 4000 caractères (bytes). La longueur minimale est 1. la précision du n est obligatoire. Exemple : NomPrograme VARCHAR2(20)
CHAR(n)	Chaine de caractères de longueur fixe allant de 1 à 2000 caractères (bytes). La chaîne est complétée par des espace si elle est plus petite que la taille déclarée Exemple CodeProgramme CHAR(3).
NUMBER(n,d)	Pour déclarer un nombre sur maximum **n** chiffres (positions) dont **d** chiffres (positions) sont réservés à la décimale. **n** peut aller de 1 à 38. Exemple salaire NUMBER(6,2). La valeur maximale du salaire dans ce cas est 9999,99
DATE	Donnée de type date située dans une plage comprise entre le 1er janvier 4712 av JC et le 31 décembre 9999 ap JC stockant l'année, mois, jour, heures, minutes et secondes
BLOB	Binary Large Object : Gros objet binaire pouvant aller jusqu'à 4 gigaoctet : Exemple photo BLOB

Définitions, les contraintes d'intégrité.

- Une contrainte d'intégrité est une règle sur la table qui permet d'assurer que les données stockées dans la base de données soient cohérentes par rapport à leur signification. Avec SQL Server, on peut implémenter plusieurs contraintes d'intégrité.

Contraintes	Significations , Exemples
PRIMARY KEY	La contrainte de PRIMARY KEY, permet de définir une clé primaire pour la table. Une clé primaire est un attribut qui permet d'identifier chaque occurrence(enregistrement) d'une table de marinière unique. Ou encore c'est un attribut permettant d'identifier chaque ligne d'une table d'une manière unique. **Exemples :** Le numéro d'admission d'un étudiant Un numéro de facture Le nom de l'étudiant ne peut être défini comme Clé primaire.
CHECK	Indique les valeurs permises qui peuvent être saisies pour la colonne (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. Les opérateurs arithmétiques (+,*,/,-), les opérateurs de comparaisons et les opérateurs logiques sont permis. Exemples: La note de l'étudiant est comprise entre 0 et 100. Le salaire minimum est plus grand que 15.

Les contraintes d'intégrité.

Contraintes d'intégrité (suite)

Contraintes	Significations , Exemples
NOT NULL	Indique que la valeur de la colonne ou de l'attribut est obligatoire. Si cette contrainte n'est pas précisée alors par défaut la valeur est NULL. Exemple: le nom de l'étudiant est obligatoire
UNIQUE	Indique que les valeurs saisies pour les colonnes (attributs) doivent être uniques, ce qui veut dire pas de doublons. Exemple: Le numéro d'assurance sociale, le numéro d'un permis de conduire, un courriel. Le nom de l'étudiant ne peut pas avoir cette contrainte.
DEFAULT	Indique la valeur par défaut que prendra l'attribut si aucune valeur n'est saisie.

La contrainte de PRIMARY KEY a les contraintes UNIQUE et NOT NULL

Syntaxe simplifiée

Syntaxe simplifiée : CREATE TABLE

```
CREATE TABLE nom_table
(
nom_colonne   type_donnee_colonne   [definition_contrainte_colonne],
nom_colonne   type_donnee_colonne   [definition_contrainte_colonne],
....
[definition_contrainte_table],
....
[definition_contrainte_table]
);
```

Exemples

Exemple1

```
CREATE TABLE Etudiants
(
numad NUMBER(10,0) CONSTRAINT pk_etudiant PRIMARY KEY,
nom VARCHAR2(20) NOT NULL,
prenom VARCHAR2 (20)
);
```

Lecture de l'exemple:

- Le nom de la table est Etudiants
- Chaque définition de colonne se termine par une virgule sauf la dernière
- Le numad a la contrainte de PRIMARY KEY, cette contrainte est sur la colonne numad
- Pour définir une contrainte, on utilise le mot réservé CONSTRAINT . Le nom pk_etudiant représente le nom de la contrainte dans le SGBD

Exemples

Exemple 2

```
CREATE TABLE Employes
(
empno NUMBER(4,0) CONSTRAINT pk_employes PRIMARY KEY,
nom VARCHAR2(20) NOT NULL,
prenom VARCHAR2 (20),
salaire number(8,2)  CONSTRAINT ck_salaire CHECK(salaire> 50000)
);
```

Lecture de l'exemple:

- Le nom de la table est Employes
- Le empno a la contrainte de PRIMARY KEY.
- Pour définir une contrainte, on utilise le mot réservé CONSTRAINT . Le nom pk_employes représente le nom de la contrainte dans le SGBD
- Il y a une contrainte NOT NULL sur le nom
- Il y a une contrainte CHECK sur le salaire. Cette contrainte est définie avec le mot réservé **CONSTRAINT.** Le nom de la contrainte est ck_salaire

Exemples

Exemple 3, cette écriture est fortement déconseillée. (et à ne pas utiliser)

```
CREATE TABLE EmployesBidon
(
empno NUMBER(4,0) PRIMARY KEY,
nom VARCHAR2(20) NOT NULL,
prenom VARCHAR2 (20),
salaire number(8,2) CHECK(salaire> 50000)
);
```

Lecture de l'exemple:

- Le nom de la table est EmployesBidon
- Le empno a la contrainte de PRIMARY KEY. Nous n'avons pas donné de nom à cette contrainte
- Le salaire a une contrainte de CHECK. Nous n'avons pas donné de nom à cette contrainte
- Lorsque vous ne donnez pas de nom aux contraintes, le système se chargera de leur donner un nom.
- L'exemple est ici uniquement parce que vous allez le trouver dans stack overflow. Cette définition des contraintes est déconseillée.

Exemples

Exemple 4

```
CREATE TABLE personnes
(
numero NUMBER(4,0) CONSTRAINT pk_personne PRIMARY KEY,
nom VARCHAR2 (15) NOT NULL,
prenom VARCHAR2 (15),
courriel VARCHAR2(40) UNIQUE,
ville VARCHAR2 (20) DEFAULT 'Montréal' CONSTRAINT ck_ville CHECK(ville IN ('Montréal','Laval',’Québec'))
);
```

Nous avons:

- Une contrainte de PRIMARY KEY sur la colonne numero. Cette contrainte a un nom : pk_personne
- Une contrainte NOT NULL sur le nom. Cette contrainte a un nom donné par le système
- Une contrainte UNIQUE sur le courriel. Cette contrainte a un nom donné par le système
- Une contrainte CHECK sur la ville. Cette contrainte a un nom : ck_ville.
- Lorsque la ville n’est pas saisie, par défaut la valeur est Montréal.

Exemples

Exemple 5

```
CREATE TABLE EmployesClg
(
empno NUMBER(4,0) ,
nom VARCHAR2(20) NOT NULL,
prenom VARCHAR2 (20),
salaire number(8,2),
CONSTRAINT pk_employes PRIMARY KEY (empno),
CONSTRAINT ck_salaire CHECK(salaire> 50000)
);
```

Lecture de l'exemple:

- La contrainte de PRIMARY KEY est définie comme si c'était une colonne. Remarquez la virgule jaune. C'est ce que nous appelons une définition de contrainte au niveau TABLE.
- La contrainte de CHECK est définie comme si c'était une colonne. Elle est définie au niveau TABLE.

CREATE TABLE: L'option IDENTITY

À partir de la version 12c de la base de données, SQL Server a introduit l'option IDENTITY pour incrémenter automatiquement la clé primaire. L'avantage d'avoir une telle option est que la clé primaire ne sera jamais dupliquée. On diminue les erreurs . La clé est auto-générée

L'option IDENTITY convient surtout pour les tables ayant comme clé primaire un numéro séquentiel : Clients, fournisseurs, joueurs, commandes, factures etc.. Le type de données pour la clé primaire est **NUMBER.**

Lorsque la clé est auto-généré (IDENTITY) alors elle est soit générée **BY DEFAULT** (par défaut) ou **ALWAYS.**

Pour mieux comprendre le concept IDENITY, nous verrons plus d'exemples lors de la présentation de la commande INSERT INTO

Syntaxe: **GENERATED BY DEFAULT**

```
CREATE TABLE clients
(
id_client number(4,0) GENERATED BY DEFAULT AS IDENTITY,
nom varchar2(30) not null,
prenom varchar2(30),
CONSTRAINT pk_client PRIMARY KEY(id_client)
);
```

CREATE TABLE: L'option IDENTITY

Syntaxe: **GENERATED ALWAYS**

```
CREATE TABLE fournisseurs
(id_fournisseur number(4,0) GENERATED ALWAYS AS IDENTITY,
nom VARCHAR2(40) NOT NULL,
prenom VARCHAR2(30) ,
CONSTRAINT pk_fournisseur PRIMARY KEY (id_fournisseur));
```

Que ce soit BY DEFAULT ou ALWAYS, nous pouvons déterminer le début de la séquence (le numéro du premier enregistrement) et l'incrément de la séquence. (un peu comme un compteur en C#).

```
CREATE TABLE clientsClg
(
id_client number(4,0) GENERATED BY DEFAULT AS IDENTITY START WITH 10 INCREMENT BY 2,
nom varchar2(30) not null,
prenom varchar2(30),
CONSTRAINT pk_clientclg primary key(id_client)
);
```

CREATE TABLE: Important

Important:

- Les tables sont des objets de la base de données. Les noms doivent être uniques. Vous ne pouvez pas avoir deux tables Employes dans votre BD
- Les noms des tables doivent être significatifs.
- Les noms des tables ne doivent pas être des mots réservés du SGBD. Exemple de mot réservés: Table, sequence, sysdate,create,cycle, constraint, primary etc…
- Les colonnes des tables doivent avoir des noms significatifs et ne doivent pas être des mots réservés.
- Les noms de colonnes sont uniques dans une table, mais pas dans la base de données.
- Les contraintes de PRIMARY KEY et CHECK doivent être définies avec le mot réservé CONSTRAINT et doivent avoir un nom significatif.
- Les contraintes sont des objets de la base de données, le nom doit être unique.
- Les contraintes de NOT NULL et UNIQUE sont généralement définies sans nom.(Le système se charge de leur donner un nom)
- Pour tous les exemples précédents, nous avons défini les contraintes lors de la définition des colonnes. Ce sont des contraintes niveau colonnes. (ou attribut)

Introduction aux bases de donnés

Les commandes DML

La commande INSERT INTO

Définition et syntaxe

- Cette commande permet d'insérer des données dans une table, une ligne à la fois. C'est une commande du DML (Data Manipulation Language)
- Deux syntaxes sont possibles pour cette commande

Syntaxe 1, cette syntaxe indique que l'on doit fournir les valeurs valides pour toutes les colonnes de la table

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

Exemple: pour la table de l'exemple 5

```
INSERT INTO Employes values(1,'Leroy','Rémi', 60000);
```

La commande INSERT INTO

Définition et syntaxe

Syntaxe2 , cette syntaxe indique que l'on doit fournir les valeurs valides pour chaque colonne indiquée dans la commande INSERT INTO

```
INSERT INTO <nom_de_table> VALUES (<liste_de_valeurs>);
```

Exemple: pour la table de l'exemple 5

```
INSERT INTO Departements VALUES
('inf','Informatique'), ('rsh','Ressources humaines'),
('ges','Gestion'), ('rec','Recherche et developpement');
```

INSERT INTO, Important

Important:

- 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.

```
INSERT INTO employes VALUES (3,'O''Brian','Kévin',100000);
```

- Le type numérique (NUMBER) est saisi en notation standard. La virgule décimale est remplacée par un point lors de la saisie

```
INSERT INTO employes VALUES (4,'O''Neil','Carl',800000.88)
```

- Le type date doit être saisie selon la norme américaine (JJ-MMM-AA pour 12 jan 21) et entre apostrophes. Pour saisir une date dans n'importe quel format, il faut s'assurer de la convertir dans le format avec la fonction TO_DATE

Si vous ne connaissez pas le format de la date, exécutez: **SELECT SYSDATE FROM DUAL;**

- Lorsque la valeur d'une colonne n'est pas connue et que celle-ci possède une contrainte de NOT NULL, alors on peut saisir le NULL entre apostrophe comme valeur pour cette colonne.

INSERT INTO, Important

Important:

- Il est possible d'utiliser une expression arithmétique dans la commande INSERT INTO à condition que cette colonne soit de type numérique.

```
INSERT INTO employes VALUES(5,'Patoche','Alain',(50000 +(50000*0.1)/2));
```

- Si des valeurs dans certaines colonnes ne doivent pas être saisies (contiennent des valeurs par défaut) alors la précision des colonnes dans lesquelles la saisie doit s'effectuer est obligatoire. Noter que les valeurs à saisir doivent être dans le même ordre de la spécification des colonnes. Voir syntaxe 2

Table personnes de l'exemple 4 (dans la colonne Villes, Montréal sera inséré par défaut)

```
INSERT INTO personnes(numero,nom,prenom,courriel) VALUES(11,'Saturne', 'Lune','Saturne@gmail.com');
```

INSERT INTO, l'option IDENTITY BY DEFAULT

Voici la table Clients

```
CREATE TABLE clients
(
id_client number(4,0) GENERATED BY DEFAULT AS IDENTITY,
nom varchar2(30) not null,
prenom varchar2(30),
CONSTRAINT pk_client PRIMARY KEY(id_client)
);
```

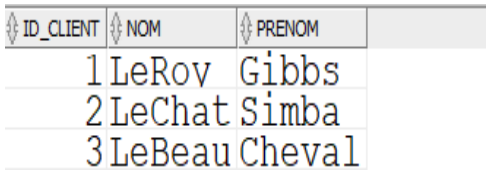
- Le id_client est une clé primaire qui s'insère automatiquement. Elle commence à 1 et s'incrémente de 1.
- Lorsque j'insère ces lignes, je n'ai pas d'erreurs puis que la clé s'insère automatiquement

Dans les insertions suivantes, il n' y a pas la colonne id_client. Le système insère un numéro automatiquement.

- insert into clients (nom, prenom) values('LeRoy','Gibbs');
- insert into clients (nom, prenom) values('LeChat','Simba');
- insert into clients (nom, prenom) values('LeBeau','Cheval');

INSERT INTO, l'option IDENTITY BY DEFAULT

Lorsque j'exécute un SELECT* FROM Clients, j'obtiens le résultat ci-après



ID_CLIENT	NOM	PRENOM
1	LeRov	Gibbs
2	LeChat	Simba
3	LeBeau	Cheval

A chaque insertion, la clé primaire augmente de 1.

Je peux briser la séquence en faisant:

insert into clients(id_Client, nom, prenom) values (**10**, 'Ce nouveau','Client');

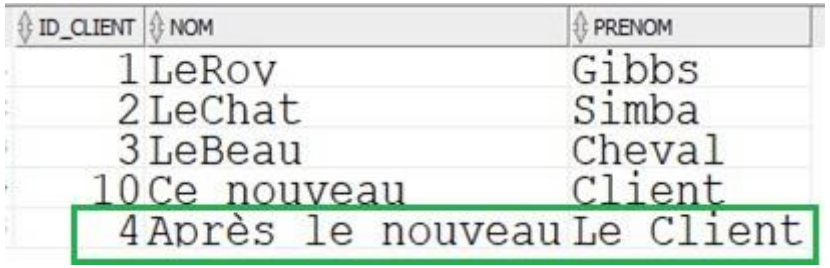
Après le SELECT* FROM Clients, j'obtiens le résultat ci-après.

ID_CLIENT	NOM	PRENOM
1	LeRov	Gibbs
2	LeChat	Simba
3	LeBeau	Cheval
10	Ce nouveau	Client

Et si je reviens à la séquence, est-ce le id_client est 4 ou 11 ?

insert into clients (nom, prenom) values('Après le nouveau','Le Client');

Remarque: lorsque la séquence arrive à 10, il y 'aura une erreur



ID_CLIENT	NOM	PRENOM
1	LeRov	Gibbs
2	LeChat	Simba
3	LeBeau	Cheval
10	Ce nouveau	Client
4	Après le nouveau	Le Client

INSERT INTO, l'option IDENTITY BY DEFAULT START WITH .. INCREMENT BY

Voici la table ClientsCLG

```
CREATE TABLE clientsClg
(
id_client number(4,0) GENERATED BY DEFAULT AS IDENTITY START WITH 10 INCREMENT BY 2,
nom varchar2(30) not null,
prenom varchar2(30),
CONSTRAINT pk_clientclg primary key(id_client)
);
```

- Le id_client est une clé primaire qui s'insère automatiquement. Elle commence à 10 et s'incrémente de 2.
- Lorsque j'insère ces lignes, je n'ai pas d'erreurs puis que la clé s'insère automatiquement

Dans cet exemple, la clé primaire commence à 10 et s'incrémente de 2

- insert into clientsclg (nom, prenom) values('LeRoy','Des Singes');
- insert into clientsclg (nom, prenom) values('Lefou','Du Village');
- insert into clientsclg (nom, prenom) values('Soleil','Vert');

ID_CLIENT	NOM	PRENOM
10	LeRoy	Des Singes
12	Lefou	Du Village
14	Soleil	Vert

INSERT INTO, l'option IDENTITY ALWAYS

Voici la table fournisseurs

```
CREATE TABLE fournisseurs
(id_fournisseur number(4,0) GENERATED ALWAYS AS IDENTITY,
nom varchar2(40) not null,
prenom varchar2(30) ,
constraint pk_fournisseur primary key (id_fournisseur));
```

Dans l'exemple suivant, la clé primaire est toujours IDENTITY, donc aucune insertion manuelle de la clé primaire n'est possible. Si je fais:

- insert into fournisseurs values (10,'Yacoub','Saliha');

J'obtiens l'erreur suivante:

INSERT INTO, Important

Important:

- Il est possible d'utiliser des insertions à partir d'une table existante (commande SELECT et une sous-requête -----à voir plus loin)
- Il est possible d'utiliser une séquence pour l'insertion automatique d'un numéro séquentiel pour une colonne (à voir plus loin)
- Après les insertions, il est recommandé d'exécuter la commande COMMIT pour enregistrer vos données (à voir plus loin)

INSERT INTO, Conclusion

Conclusion

- INSERT INTO est une commande DML qui permet d'insérer des données dans une table. Une ligne à la fois.
- La commande INSERT INTO ne pourra pas s'exécuter si les contraintes d'intégrités ne sont pas respectées. En tout temps, il faut vérifier les contraintes d'intégrité.
- Pour que les insertions (les données) sont définitivement enregistrées dans la BD, il suffit d'exécuter un COMMIT.

La commande UPDATE

- Tout comme la commande INSERT INTO, la commande UPDATE est une commande du DML (Data Manipulation Language)
- La commande UPDATE permet d'effectuer des modifications des **données sur une seule table.**
- Contrairement à la commande INSERT INTO, la commande UPDATE permet de la modification de plusieurs enregistrements (lignes) en même temps
- Lors de la modification des données, les contraintes d'intégrité doivent être respectées

Syntaxe:

```
UPDATE <nom_de _table> SET
(<nom_de_colonne>=<nouvelle_valeur>)
[WHERE <condition>];
```

Exemple

```
UPADTE employes SET salaire = salaire*1,01 where empno =10
```

La commande UPDATE

- La clause WHERE a le même rôle que pour le SELECT. Elle permet de cibler les lignes à mettre à jour. (à modifier)
- Il est possible de mettre à jour plusieurs colonnes en même temps comme le montre la syntaxe.

UPDATE employesinfo SET salaire = salaire +(salaire*0.5)
WHERE nom ='Fafar'; → ajoute 1% du salaire à l'employé dont le nom est Fafar.

UPDATE employesinfo SET salaire = salaire +(salaire*0.1) → ajoute 1% du salaire à tous les employés

UPDATE employesinfo SET salaire = salaire +(salaire*0.1), commission =200 ; → on met à jour le salaire et la commission pour tous les employés

La commande DELETE

- Tout comme les commande INSERT INTO et UPDATE la commande DELETE est une commande du DML (Data Manipulation Language)
- La commande DELETE permet de supprimer une ou plusieurs lignes d'une table
- Lors de la suppression des données, les contraintes d'intégrité référentielle doivent être respectées (voir plus loin)
- Syntaxe:

```
DELETE FROM <nom_de _table>
[WHERE <condition>];
```

Exemple

```
DELETE FROM employes WHERE empno =10
DELETE FROM employes WHERE adresse LIKE '%Montréal%'
```

Officialiser ses transactions

- Les opérations DML, une fois exécutées doivent être confirmées pour que la sauvegarde soit effective dans la base de données.
- COMMIT: elle permet d'officialiser une mise à jour (INSERT, UPDATE, DELETE) ou une transaction (série de commandes de manipulation de données effectuées depuis le dernier COMMIT) sur la base de données.
- ROLLBACK : permet d'annuler une transaction avant un COMMIT ; une fois le COMMIT exécuté aucun ROLLBACK n'a d'effet sur la BD

```
insert into clients (nom, prenom) values('LeRoy','Gibbs');
insert into clients (nom, prenom) values('LeChat','Simba');
insert into clients (nom, prenom) values('LeBeau','Cheval');
COMMIT
```

Conclusion, les commandes DML

Il existe 3 commandes du DML (Data Manipulation Language):

- La commande INSERT INTO, qui permet d'ajouter des données dans une table une ligne à la fois. Lors des insertions les contraintes d'intégrités doivent être respectées.
- la commande UPDATE, qui permet de modifier les données d'une table. La clause WHERE est utilisée pour cibler les lignes à modifier. Lors des modifications, les contraintes d'intégrité doivent être respectées.
- La commande DELETE, qui permet de supprimer des données dans une table. La clause WHERE est utilisée pour cibler les lignes à supprimer. Lors des modifications, les contraintes d'intégrité référentielles doivent être respectées

Après exécution des opérations DML, vous devez exécuter COMMIT pour officialiser les transactions.

Pour annuler une opération DML on exécute un ROLLBACK avant le COMMIT. Après un COMMIT, aucun ROLLBACK n'a d'effet.

Introduction aux bases de donnés

Intégrité référentielle, la contrainte de Foreign KEY
Clé primaire composée

La contrainte de FOREIGN KEY

Plan de la séance

- Retour sur la dernière séance:
  - Point de vue de l'étudiant
  - Point de vue de l'enseignant.
- Rappels:
  - CREATE TABLE, les contraintes d'intégrité
- Définition
- Syntaxe
- Exemples
- Laboratoire 3

Rappel: CREATE TABLE

- La commande CREATE TABLE permet de créer une table dans une base de données.
- Pour créer une table nous avons besoin de connaître:
  - La liste des attributs de la table
  - Le type de données pour chaque attribut (NUMBER, VARCHAR2(n), CHAR(n), DATE, etc…)
  - Les contraintes d'intégrité, s'il y'en a, pour chaque attribut
    - PRIMARY KEY,
    - CHECK,
    - NOT NULL,
    - UNIQUE
    - Default
  - **Les contraintes niveau table**

Contrainte d'intégrité référentielle, définition

- On parle d'intégrité référentielle lorsque les valeurs d'une ou plusieurs colonnes d'une table sont déterminées ou font référence à des valeurs d'une colonne d'une autre table.
- Dans notre exemple ci-après, les valeurs de la colonne codeequipe de la table **Joueurs** font référence aux valeurs de la colonne codeequipe de la table **Equipes**
- La table Joueurs ne contient aucun **codeequipe** qui n'est pas dans la table Equipes.

Table joueurs

NUMJOUEUR	NOM	PRENOM	CODEEQUIPE
1	PRICE	CAREY	MTL
2	MARKOV	ANDRÉ	MTL
3	SUBBAN	KARL	MTL
4	PATIORETTY	MAX	MTL
10	HAMOND	ANDREW	OTT
6	STONE	MARC	OTT
9	TURIS	KYLE	OTT
7	GALLAGHER	BRANDON	MTL
8	TANGUAY	ALEX	AVL
11	THOMAS	BIL	AVL

référence

Table Equipes

	CODEEQUIPE	NOMEQUIPE	VILLE	NBCOUPES
1	MTL	LES CANADIENS DE MONTRÉAL	MONTRÉAL	24
2	TOR	LES MAPLE LEAFS	TORONTO	22
3	OTT	LES SÉNATEURS	OTTAWA	4
4	AVL	LES AVALANCHES	COLORADO	2
5	VAN	LES CANUKS	VANCOUVER	1
6	BRU	LES BRUNS DE BOSTON	BOSTON	13

Contrainte d'intégrité référentielle, définition

- Dans la table Equipes, le **codeequipe** est une clé primaire.
- Dans la table Joueurs, le **codeequipe** est appelé **Clé étrangère** ou **FOREIGN KEY.**
- Pour parler de contrainte de **FOREIGN KEY** nous avons besoins de deux tables :
  - Une table A (exemple Equipes) qui contient un attribut (exemple Codeequipe) de clé primaire : PRIMARY KEY
  - Une table B (exemple Joueurs) qui va contenir un attribut de clé étrangère.
- La contrainte de FOREIGN KEY indique que la valeur de l'attribut de la table B(Joueurs) correspond à une valeur d'une clé primaire de la table spécifiée A. (Table Equipes)
- La clé primaire de l'autre table A (exemple table Equipes) doit être obligatoirement crée pour que cette contrainte soit acceptée. On ne peut pas faire référence à un attribut qui n'existe pas.
- La clé primaire la table A (Equipes) et l'attribut défini comme clé étrangère de la table B (Joueurs) doivent être de même type et de même longueur.
- Il n'est pas nécessaire que les attributs de clé primaire et de clé étrangère aient des noms identiques.

Contrainte d’intégrité référentielle

Exemple

La table EQUIPES doit être créée en premier

```
CREATE table EQUIPES
(
Codeequipe  CHAR(3) CONSTRAINT pk_equipe PRIMARY KEY,
nomequipe  VARCHAR2(50) NOT NULL,
Ville VARCHAR2(40),
nbcoupes NUMBER(2,0) CONSTRAINT ck_nbcoupe CHECK (nb_coupes > =0)
) ;
```

Contrainte d'intégrité référentielle, Exemples

On crée la table JOUEURS après

```
CREATE TABLE Joueurs
(
numjoueur NUMBER(3,0) CONSTRAINT pk_joueurs PRIMARY KEY,
nom VARCHAR2(30) NOT NULL,
prenomVARCHAR2(30),

codeequipe CHAR(3),
CONSTRAINT fk_Joueurs_equipes FOREIGN KEY (codeequipe) REFERENCES equipes(codeequipe)
);
```

Les attributs codeequipe des deux tables sont de même type et de même longueur: CHAR(3). C'est une obligation.

Il y a une virgule en rouge dans la table JOUEURS après le codeequipe ce qui indique que la contrainte de FOREIGN KEY est une contrainte sur la table. C'est une obligation

Le mot réservé REFERENCES indique la colonne (attribut) référencée de la table de la clé primaire. C'est pourquoi la table de la clé primaire doit être créée en premier. On ne peut pas référencer quelque chose qui n'existe pas.

Contrainte d’intégrité référentielle

La contrainte de FOREIGN KEY garantie l’intégrité référentielle ce qui veut dire :

- Vous ne pouvez pas modifier (UPDATE) la valeur de l’attribut de la clé primaire s’il a une valeur de clé étrangère.
    - Exemple dans la table EQUIPES vous ne pouvez pas modifier le codeequipe MTL pour BLA, car il existe des joueurs ayant le codeequipe MTL.
- Vous ne pouvez pas supprimer (DELETE) une ligne de la table référencée s’il existe des enregistrements ayant une valeur de la clé étrangère égale à la valeur de la clé primaire de la ligne à supprimer.
    - Exemple vous ne pouvez pas supprimer de la table equipes l’équipe dont le codeequipe est OTT, car il existe des joueurs ayant ce codeequipe.
- Aucune insertion (INSERT INTO) ni modification (UPDATE) n’est possible dans la table de la clé étrangère si la contrainte d’intégrité référentielle n’est pas respectée.
    - Exemple INSERT INTO Joueurs VALUES (1, 'LeRoy','Gibbs’, 'GOM’) ne marchera pas car GOM n’est pas dans la table Equipes
- Vous ne pouvez pas détruire la table EQUIPES par un simple DROP TABLE (à voir plus loin). Il faudra
    - Soit supprimer la table joueurs en premier (à condition qu’elle ne soit pas référencée elle aussi)
    - Soit utiliser CASCADE CONSTRAINTS.
    - Soit désactiver ou détruire la contrainte d’intégrité (la FOREIGN KEY)

Clé primaire composée

- Il arrive qu'une table ait besoin de deux (ou plus) attributs pour identifier de manière unique les enregistrements. Dans ce cas on parle de clé primaire composée.
- Dans la plupart des cas les attributs de la clé primaire sont des clés étrangères.
- La définition de la clé primaire se fait au niveau TABLE et non pas colonne ou attribut.
- Pour définir une clé primaire composée, il suffit de fournir la liste des attributs de la clé séparés par des virgules entre parenthèses.
- Si une table a une clé primaire composée dont les attributs sont des clés étrangères, la table est dite : Table de relation

Clé primaire composée

Exemple

```
create table Films
(
id_film number(5,0) constraint pk_film primary key,
titre varchar(100) not null,
annee number(4,0) not null
)
```

```
create table Acteurs
(
id_acteur number(5,0) constraint pk_acteur PRIMARY key,
nom_acteur varchar2(40) not null,
prenom_acteur varchar2(30) not null
);
```

Clé primaire composée

On crée la table Acteurs_films après avoir crée les table Films et Acteurs.

```
create table Acteurs_films
(
id_film number(5,0),
id_acteur number(5,0),
salairePourleFilm number(10,0) not null constraint ck_salairefilm check(salairePourleFilm >500000),

constraint fk_films foreign key (id_film) references films(id_film),
constraint fk_acteurs foreign key (id_acteur) references acteurs(id_acteur),
constraint pk_acteurs_films primary key (id_film,id_acteur)
);
```

- La table Acteurs_films a une clé primaire composée. Cette clé est (id_film,id_acteur)
- La contrainte de clé primaire composée est une contrainte sur la TABLE. (tout comme la contrainte de Foreign key).
- La table Acteurs_films a deux contraintes de FOREIGN KEY.
- La table Acteurs_films est appelée table de RELATION.

Conclusion et bonnes pratiques pour la commande CREATE TABLE

- Comme mentionné déjà il faut que les noms des tables, des colonnes, et des contraintes soient significatifs.
- Pour les noms de contrainte, vous pouvez utiliser:
  - pk_nomdetable pour PRIMARY KEY
  - fk_nomdetable_nomtableOrigine pour la FOREIGN KEY ou fk_nomTableOrigine

    constraint fk_films foreign key (id_film) references films(id_film),
    constraint fk_films_categories foreign key (id_categorie) references Categories(id_categorie),

  - ck_Colonne pour CHECK
- Lorsque vous pouvez utiliser les contraintes niveau table, faîtes-le. Ça donne un code plus propre.
- Lorsque c'est recommandé, et possible, utilisez un numéro séquentiel pour la clé primaire.

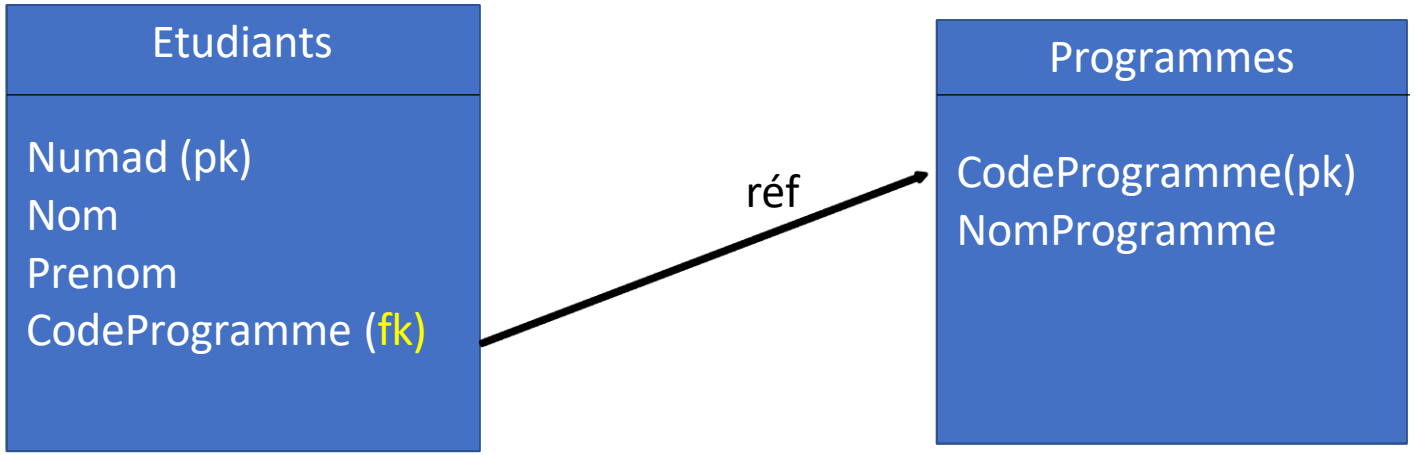
Conclusion et bonnes pratiques pour la commande CREATE TABLE

```
create table Films
(
id_film number(5,0),
titre varchar(100) not null,
annee number(4,0) not null,
classe char(3) not null,
id_categorie number(2,0),
id_langue char(2) not null,

---définition des contraintes
constraint pk_film primary key(id_film),
constraint  ck_classe check (classe in('gen','13+','08+')),
constraint fk_films_categories foreign key (id_categorie)references Categories(id_categorie),
constraint fk_langues foreign key(id_langue ) references langues(id_langue)
);
```

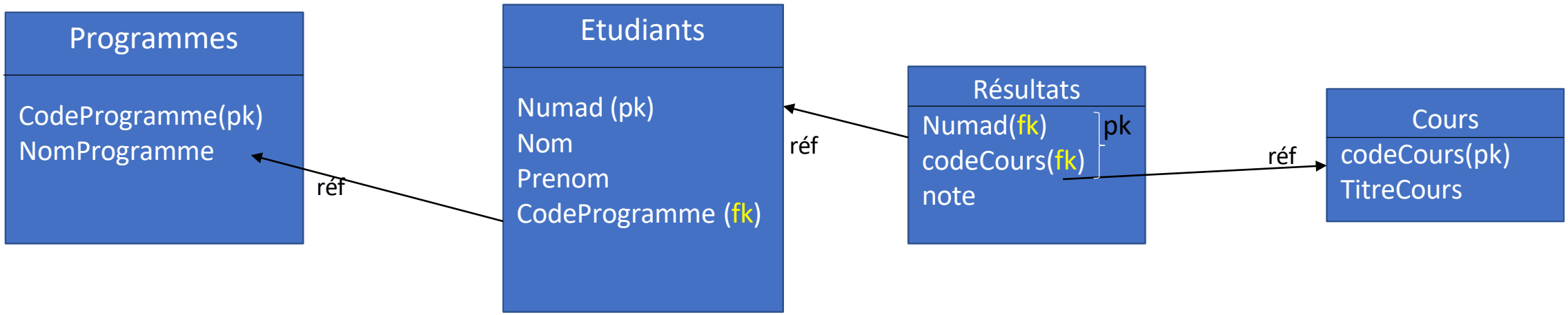
Représentation graphique, diagramme référentiel

- Dans le schéma suivant, on montre que la FK de la table étudiants fait référence à la PK de la table Programmes.
- Les tables sont représentées par des carrés. Le nom des tables est clairement indiqué en haut.
- Les attributs des tables, sont clairement indiqués aussi. Les PK et les FK sont clairement mises en évidence
- Le lien pointe de la clé étrangère vers la clé primaire. On dit que la clé étrangère fait référence à la clé primaire.
- Plus tard dans la session, le digramme référentiel dans SQL Server Data Modeler vous sera montré.



Représentation graphique.

- Autre exemple.



Introduction aux bases de donnés

La commande ALTER TABLE

La commande ALTER TABLE

Plan de la séance

- Retour sur la dernière séance:
  - Point de vue de l'étudiant
  - Point de vue de l'enseignant.
- Rappels:
  - CREATE TABLE
  - Rappel, intégrité référentielle
- La commande ALTER TABLE
  - L'option ADD
  - L'option DROP
  - L'option MODIFY
  - L'option DISABLE
  - L'option RENAME
- La commande DROP TABLE
- La commande RENAME
- Laboratoire 4

ALTER TABLE, définition

- La commande ALTER TABLE est une commande du DDL (Data Definition Language), tout comme la commande CREATE TABLE.
- Cette commande permet de modifier la structure d'une table et non son contenu.
- Après la création d'une table, il est parfois nécessaire de modifier sa structure, comme par exemple: Ajouter une colonne, modifier le type d'une colonne, supprimer une contrainte…la commande ALTER TABLE sert à cela.

Voici une liste d'opérations qu'on peut faire avec la commande ALTER TABLE.

ALTER TABLE, définition

- 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.
- Détruire une colonne
- Renommer une colonne.

ALTER TABLE, Les options:

La commande ALTER TABLE a les options suivantes:

- ADD
- DROP
- MODIFY
- RENAME.
- ENABLE/DISABLE

Exemple:

```
CREATE TABLE Employes
(
numemp number(4,0),
nom varchar2(15),
prenom varchar2(20),
ville varchar2(30)
);
```

ALTER TABLE, l'option ADD

L'option **ADD:** permet d'ajouter une colonne ou une contrainte à une table.

ALTER TABLE employes **ADD CONSTRAINT employes_pk PRIMARY KEY (numeemp);**

Permet d'ajouter une contrainte de Primary Key pour la table employes sur l'attribute numeemp.

ALTER TABLE employes **ADD (salaire NUMBER (8,2));**

Peremt d'ajouter un attribut Salaire de type NUMBER(8,2) pour la table Employes.

ALTER TABLE employes **ADD (echelon NUMBER(2,0) NOT NULL CHECK (echelon >10))**

Permet d'ajouter une colonne avec ses contraintes. Nous n'avons pas donné de nom à cette contrainte. Le système lui donnera un nom

ALTER TABLE employes **ADD** prime NUMBER(6,2) CONSTRAINT ck_prime CHECK(prime>500); On ajoute une colonne avec une contrainte CHECK. Nous avons donné un nom à cette contrainte

ALTER TABLE employes **ADD** CONSTRAINT ck_salaire CHECK(salaire>20000); Nous avons ajouté une contrainte sur une colonne existante.

ALTER TABLE, l'option ADD

Pour ajouter une colonne avec une contrainte de FOREIGN KEY, il faudra le faire en 2 étapes.

1. Ajouter la colonne
2. Ajouter la contrainte.

La raison ? Car la contrainte de FOREIGN KEY est une contrainte niveau table

En 1---) ALTER TABLE etudiants ADD code NUMBER(3);

En 2---) ALTER TABLE ETUDIANTS ADD CONSTRAINT fk_etudiants_programme FOREIGN KEY(code) REFERENCES programmes(codeprogramme);

ALTER TABLE, l'option DROP

L'option **DROP:** permet de supprimer une colonne ou une contrainte à une table.

ALTER TABLE employes DROP PRIMARY KEY; -- C'est possible car une table a une seule clé primaire.

ALTER TABLE employes DROP CONSTRAINT ck_salaire.

ALTER TABLE employes DROP **COLUMN** prime

Important:

Pour détruire une colonne c'est DROP COLUMN

Le mot réservé COLUMN n'est pas là pour ADD

ALTER TABLE, l'option MODIFY

L'option MODIFY: Cette option permet de modifier le type de données, la valeur par défaut et la contrainte de NOT NULL sur une colonne déjà existante.

ALTER TABLE employes **MODIFY (nom NOT NULL);**
ALTER TABLE employes **MODIFY (nom varchar2(40));**
ALTER TABLE employes **MODIFY (ville DEFAULT 'Montréal');**
ALTER TABLE employes **MODIFY** prime check (prime >200);

Lorsque vous modifiez une contrainte, si la table contient des données qui ne respectent la nouvelle contrainte, la modification ne sera pas acceptée.

Exemple : **ALTER TABLE** employes **MODIFY** prime check (prime >400); ne marchera pas si la table contient des prime <400.

Il est de même losque vous ajoutez des contraintes.

ALTER TABLE, l'option RENAME

L'option RENAME, permet de renommer une contrainte ou une colonne.

ALTER TABLE joueurs RENAME CONSTRAINT SYS_C00126194 TO pk_joueurs;

ALTER TABLE employes RENAME COLUMN salaire TO SalaireEmp;

Important: Pour renommer une colonne c'est RENAME COLUMN

ALTER TABLE, l'option DISABLE /ENABLE

L'option DISABLE ou ENABLE, permet de désactiver ou activer une contrainte. (importance de connaitre le nom de la contrainte.)

ALTER TABLE employes **DISABLE CONSTRAINT ck_salaire;**

ALTER TABLE employes **ENABLE CONSTRAINT ck_salaire;**

ALTER TABLE employes **DISABLE PRIMARY KEY; (**possible car nous avons une contrainte PK par table)

DROP TABLE

DROP TABLE , permet de supprimer une table – Détruire une table

Exemple : DROP TABLE Employes.

A cause de la FOREIGN KEY, les tables référencées ne peuvent être détruites.

Pour pouvoir supprimer les tables référencées:

1. Soit, vous supprimez les contraintes de FOREIGN KEY d'abord ou les désactiver. (ce n'est pas conseillé du tout.)
2. Soit supprimez les table dans un certain ordre. (les tables non référencées en premier)
3. Soit par un CASCADE CONSTRAINT, qui supprime les contraintes de FOREIGN KEY en cascade.

DROP TABLE nomdeTable CASCADE CONSTRAINTS.

À l'avenir, cette commande doit être placée au début de tous vos scripts.

RENAME

Permet de renommer une table ou un objet de la base de données.

RENAME Employes TO EmployesInfo;

Introduction aux bases de donnés

Requêtes avec jointures

Requêtes avec jointures

Plan de la séance

- Retour sur la dernière séance:
  - Point de vue de l'étudiant
  - Point de vue de l'enseignant.
- Rappels:
  - La commande SELECT
  - La contrainte de FK
- Le produit cartésien
- Jointure interne: INNER JOIN
- Jointure externe: LEFT/RIGHT OUTER JOIN
- Début du laboratoire 3
- Importance des alias. (exemple)

Rappel, la commande SELECT

La commande SELECT

- La commande SELECT est la commande la 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>];
```

Jointures: Introduction

Pour les explications nous utiliserons les deux tables suivante

Table films

ID_FILM	TITRE	ANNEE	CLASSE	ID_CATEGORIE
1	Pirates des Caraïbes,la Malédiction du Black Pearl	2003	08+	2
2	Pirates des Caraïbes,le Secret du coffre maudit	2006	08+	2
3	Pirates des Caraïbes,Jusqu'au bout du monde	2007	13+	2
4	Élémentaire	2018	gen	5
5	Holmes à la rescousse	2017	gen	5
6	l'incroyable Holmes	2007	gen	5
7	Mission Mars	2019	08+	4
8	Film d'horreur	2006	13+	3
9	Dîner de cons	2000	gen	1
10	Volver	2007	08+	2
11	Le temps va	2020	gen	(null)
12	La covid est finie	2021	gen	(null)
13	La vie est belle	1999	gen	(null)

Table **categories**

ID_CATEGORIE	NOM_CATEGORIE
1	Comédie
2	Fantastique
3	Horreur
4	Science fiction
5	Action
6	Drame
7	Comédie musicale

Produit cartésien, Définition

- Produit cartésien:

Le produit cartésien est une requête de sélection qui met en jeux plusieurs tables. Pour deux tables, la sélection consisteà afficher la première ligne de la première table avec toutes les lignes de la deuxième table, puis la deuxième ligne de la première table avec toutes les lignes de la deuxième table et ainsi de suite. Ce type de sélection implique beaucoup de redondances.

Exemple :

```
SELECT  titre,nom_categorie
FROM films, categories;
```

Va nous donner comme résultat

91 lignes. (13*7)

il y a une suite au résultat de la figure)

Il y **a beaucoup de redondance**.

De plus, remarquez que “Le temps va”

A la catégorie Action alors qu’il n’en a pas

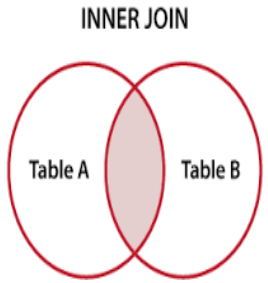
TITRE	NOM_CATEGORIE
Volver	Action
Dîner de cons	Action
Film d'horreur	Action
Mission Mars	Action
l'incroyable Holmes	Action
Holmes à la rescousse	Action
Le temps va	Action
Pirates des Caraïbes,Jusqu'au bout du monde	Action
Pirates des Caraïbes,le Secret du coffre maudit	Action
Pirates des Caraïbes,la Malédiction du Black Pearl	Action
La vie est belle	Action
La covid est finie	Action
Élémentaire	Action
Dîner de cons	Comédie
Volver	Comédie

Jointures: Définitions

- Une jointure

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 valeurs communes, en général les clés primaires.

Il existe plusieurs types de jointure que nous allons aborder au fur et à mesure. Parmi ces jointures, nous avons la jointure interne appelée aussi jointure simple



Une jointure simple consiste en un produit cartésien avec un **INNER JOIN** faisant ainsi une restriction sur les lignes. La restriction est faîte sur l'égalité de la valeur de deux attributs (cas de deux tables) qui sont la valeur **d'une clé primaire est égale à la valeur d'une clé étrangère.**

Les jointures se font au niveau de la clause FROM

Jointures: Définitions

Syntaxe:

```
SELECT colonne1, colonne2, …..
FROM  [(]nom_table1  INNER JOIN nomTable2
ON nom_table1.cleEtrangere = nomTable2.clePrimaire [)];
[WHERE <condition>]
[ORDER BY <nom_de_colonne>];
```

Exemple 1

```
SELECT titre, nom_categorie
   FROM (films INNER JOIN  categories
   ON films.id_categorie = categories.id_categorie);
```

Lorsque nous avons INNER JOIN, les parenthèses ne sont pas obligatoires

Jointures

- Explications

Une requête avec jointure interne INNER JOIN, va ramener des résultats des deux tables Films et Categories uniquement s'il y a égalité entre la **clé étrangère code_categorie de la table films** et la **clé primaire code_categorie de la table categories.**

Selon le contenu initial des tables, la requête va ramener exactement 10 enregistrements (lignes)

Les films qui n'ont pas de catégories ne seront pas ramenés. Les catégories qui n'ont pas de films ne seront pas ramenées

TITRE	NOM_CATEGORIE
Dîner de cons	Comédie
Pirates des Caraïbes,la Malédiction du Black Pearl	Fantastique
Pirates des Caraïbes,le Secret du coffre maudit	Fantastique
Pirates des Caraïbes,Jusqu'au bout du monde	Fantastique
Volver	Fantastique
Film d'horreur	Horreur
Mission Mars	Science fiction
l'incroyable Holmes	Action
Holmes à la rescousse	Action
Élémentaire	Action

Les jointures

- 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. Si le nom de la table n'est pas précisé cela va renvoyer une erreur: « nom de colonne ambiguë »

```
SELECT titre, nom_categorie,categories.id_categorie
   FROM (films INNER JOIN  categories
   ON films.id_categorie = categories.id_categorie);
```

- Toutes les tables dont les attributs apparaissent dans la clause **SELECT** ou dans la clause **WHERE** doivent apparaître dans la clause FROM. (dans la jointure)

Dans l'exemple suivant, dans le FROM, nous avons une jointure entre films et categories à cause du WHERE.

```
SELECT titre
   FROM films INNER JOIN  categories
   ON films.id_categorie = categories.id_categorie
   WHERE nom_categorie ='Action';
```

Les jointures

- 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.
- Un alias est un nom , généralement une lettre, qu'on donne aux tables dans le FROM.

```
SELECT titre, nom_categorie,c.id_categorie
   FROM fiLms f INNER JOIN  categories c
   ON f.id_categorie = c.id_categorie;
```

L'exemple suivant va renvoyer une erreur: « **categorie.id_categorie** » identificateur invalide

```
SELECT titre, nom_categorie,categorie.id_categorie
   FROM fiLms f INNER JOIN  categories c
   ON f.id_categorie = c.id_categorie;
```

Donner des Alias n'est pas obligatoire sauf dans un cas que nous verrons plus loin

Jointures: Autres exemples

Pour la suite utiliserons les deux tables suivante

Table films

ID_FILM	TITRE	ANNEE	CLASSE	ID_CATEGORIE
1	Pirates des Caraïbes,la Malédiction du Black Pearl	2003	08+	2
2	Pirates des Caraïbes,le Secret du coffre maudit	2006	08+	2
3	Pirates des Caraïbes,Jusqu'au bout du monde	2007	13+	2
4	Élémentaire	2018	gen	5
5	Holmes à la rescousse	2017	gen	5
6	l'incroyable Holmes	2007	gen	5
7	Mission Mars	2019	08+	4
8	Film d'horreur	2006	13+	3
9	Dîner de cons	2000	gen	1
10	Volver	2007	08+	2
11	Le temps va	2020	gen	(null)
12	La covid est finie	2021	gen	(null)
13	La vie est belle	1999	gen	(null)

Table acteurs

ID_ACTEUR	NOM_ACTEUR	PRENOM_ACTEUR	SALAIREMINIMUM	NATIONALITE
1	Depp	Johnny	1000000	Américain
2	Radcliffe	Daniel	2000000	Britanique
3	L'acteur	Georges	800000	Américain
4	Actrice	Alice	1500000	Américaine
5	Saturne	Kévin	1200000	Canadien
6	Lechat	Simba	3000000	Canadien
7	Fafar	Ruby	900000	Canadienne
8	Watson	Emma	1500000	Britanique
9	Monsieur	Spok	2500000	Américain
10	Holmes	Sherlock	1000000	Américain

Table syActeurs_film

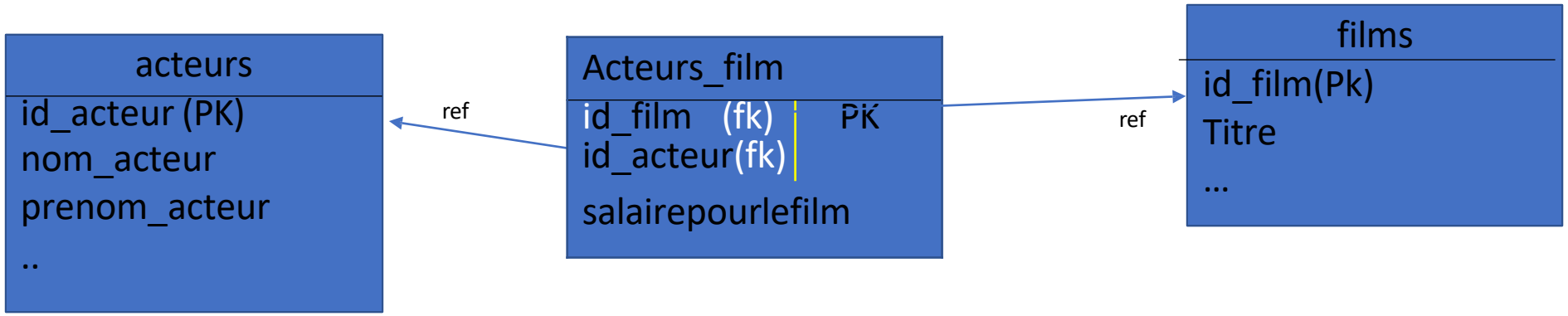
ID_FILM	ID_ACTEUR	SALAIREPOURLEFILM
1	1	1000000
2	1	5000000
3	1	8000000
2	7	1000000
3	5	1500000
1	5	600000

Jointures, exemples

Nous voulons savoir avoir le nom des acteurs, leurs prénoms ,le titre du film et le salaire qu'ils ont gagné pour le film. La requête serait la suivante:

```
SELECT nom_acteur, prenom_acteur,titre,salairepourlefilm
FROM  acteurs
INNER JOIN Acteurs_film ON acteurs.id_acteur = Acteurs_film.id_acteur
INNER JOIN films on films.id_film = Acteurs_film .id_film;
```

Avant de se lancer dans l'écriture de la requête, il est très importants de visualiser le lien entre les 3 tables pour comprendre les jointures. Car n'oublions pas, une jointure c'est juste une valeur de clé étrangère qui est égale à une valeur de clé primaire.



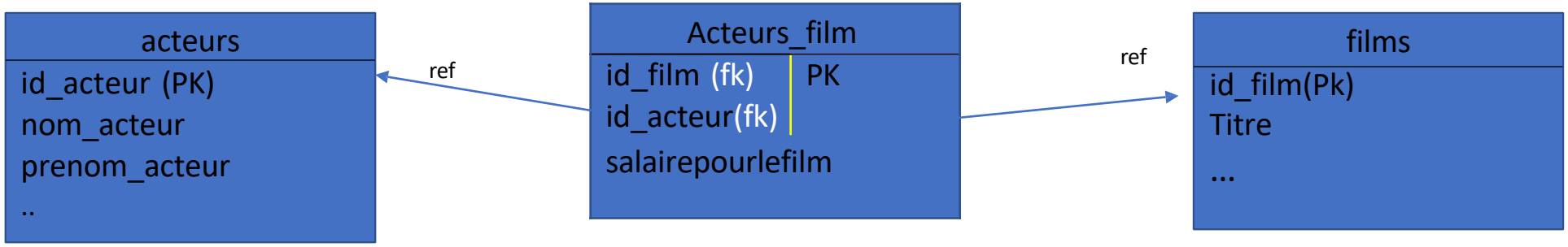
Jointures, Table de lien

Nous voulons savoir le nom des acteurs, leurs prénoms qui on joué dans le film: Pirates des Caraïbes,Jusqu'au bout du monde

```
SELECT nom_acteur, prenom_acteur
FROM  syacteurs
INNER JOIN Acteurs_film ON syacteurs.id_acteur = Acteurs_film .id_acteur
INNER JOIN films ON films.id_film = Acteurs_film.id_film
WHERE titre = 'Pirates des Caraïbes,Jusqu''au bout du monde';
```

Quand on regarde la clause SELECT, nous avons le nom et le prenom de l'acteur donc nous avons besoin de la table **syacteurs**. La clause WHERE indique que nous avons besoin de la table **films**. (à cause du titre du film).

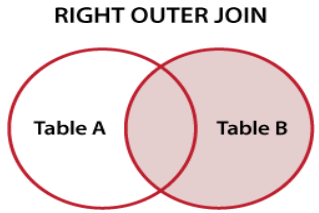
Les deux tables **syacteurs et Films** ne sont aps directement liées ensemble. Elles sont liées par la table **syActeurs_film.** La table syActeurs_film **est appelée table de lien**.



Les jointures externes

Définitions :

Jointure externe droite (RIGHT OUTER JOIN)



Dans la jointure externe droite, des enregistrements de table à droite de la jointure seront ramenés même si ceux-ci n'ont pas d'occurrences dans l'autre table.

Dans l'exemple suivant, la requête va renvoyer tous les films(titre) ayant une catégorie et toutes les catégorie y compris celles qui n'ont pas de films (voir acétate suivante.

La raison est que **categories** est à droite de la jointure (JOIN), donc le système va ramener TOUTES les catégories

```
SELECT titre, nom_categorie
   FROM (fiLms RIGHT OUTER JOIN  categories
   ON films.id_categorie = categories.id_categorie);
```

Les jointures externes

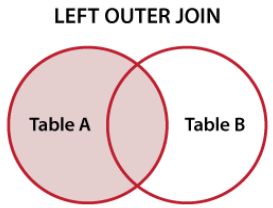
TITRE	NOM_CATEGORIE
Dîner de cons	Comédie
Pirates des Caraïbes,la Malédiction du Black Pearl	Fantastique
Pirates des Caraïbes,le Secret du coffre maudit	Fantastique
Pirates des Caraïbes,Jusqu'au bout du monde	Fantastique
Volver	Fantastique
Film d'horreur	Horreur
Mission Mars	Science fiction
l'incroyable Holmes	Action
Holmes à la rescousse	Action
Élémentaire	Action
(null)	Drame
(null)	Comédie musicale

```
SELECT titre, nom_categorie
   FROM (fiLms RIGHT OUTER JOIN  categories
   ON films.id_categorie = categories.id_categorie);
```

Les jointures externes

Définitions :

Jointure externe Gauche (LEFT OUTER JOIN)



Dans la jointure externe gauche, des enregistrements de table à gauche de la jointure seront ramenés même si ceux-ci n'ont pas d'occurrences dans l'autre table.

Dans l'exemple suivant, la requête va renvoyer tous les films(titre) ayant une catégorie et les films qui ne sont dans aucune catégorie. La requête ne va ramener que les catégories ayant des films

La raison est que **films** est à gauche de la jointure (JOIN), donc le système va ramener TOUS les films

```
SELECT titre, nom_categorie
   FROM (films LEFT OUTER JOIN  Categories
   ON Films.id_categorie = Categories.id_categorie);
```

Les jointures externes

TITRE	NOM_CATEGORIE
Élémentaire	Action
Holmes à la rescousse	Action
l'incroyable Holmes	Action
Dîner de cons	Comédie
Pirates des Caraïbes,la Malédiction du Black Pearl	Fantastique
Pirates des Caraïbes,le Secret du coffre maudit	Fantastique
Pirates des Caraïbes,Jusqu'au bout du monde	Fantastique
Volver	Fantastique
Film d'horreur	Horreur
Mission Mars	Science fiction
Le temps va	(null)
La covid est finie	(null)
La vie est belle	(null)

```
SELECT titre, nom_categorie
   FROM (Films LEFT OUTER JOIN  Categories
   ON films.id_categorie = categories.id_categorie);
```

Les jointures externes

Si nous avons bien compris, ces deux requêtes donnent le même résultats.

C'est la façon dont vous placez vos tables par rapport à la jointure qui est importante

```
SELECT titre, nom_categorie
   FROM (films LEFT OUTER JOIN  categories
   ON films.id_categorie = categories.id_categorie);
```

```
SELECT titre, nom_categorie
   FROM (categories  RIGHT OUTER JOIN  films
   ON films.id_categorie = categories.id_categorie);
```

Jointures: importance des Alias

Pour la suite utiliserons les deux tables suivante

Table films

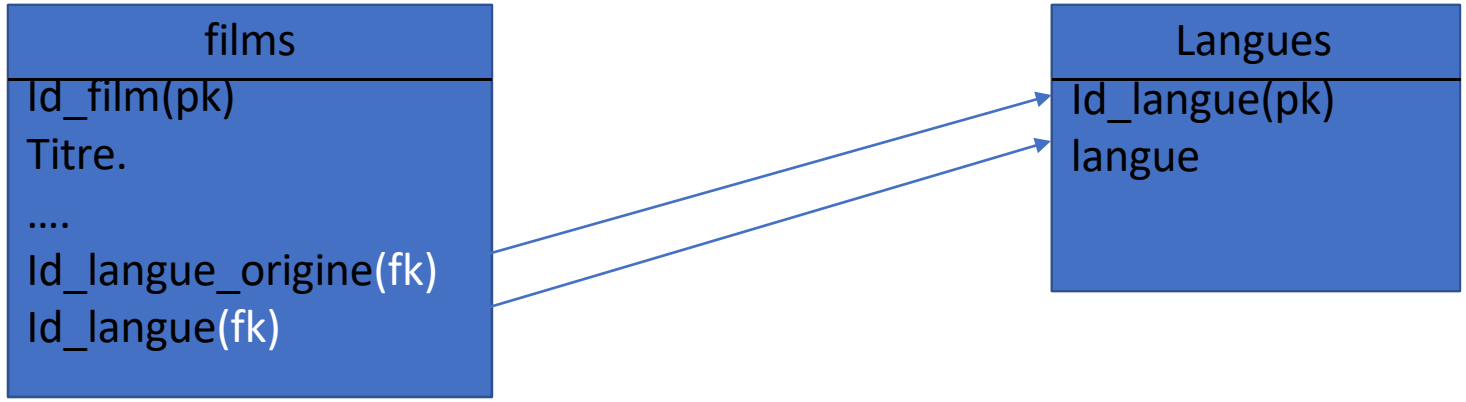
ID_FILM	TITRE	ANNEE	CLASSE	ID_CATEGORIE	ID_LANGUE_ORIGINE	ID_LANGUE
1	Pirates des Caraïbes,la Malédiction du Black Pearl	2003	08+	2	en	fr
2	Pirates des Caraïbes,le Secret du coffre maudit	2006	08+	2	en	fr
3	Pirates des Caraïbes,Jusqu'au bout du monde	2007	13+	2	en	fr
4	Élémentaire	2018	qen	5	en	fr
5	Holmes à la rescousse	2017	qen	5	en	fr
6	l'incroyable Holmes	2007	qen	5	en	fr
7	Mission Mars	2019	08+	4	fr	en
8	Film d'horreur	2006	13+	3	fr	en
9	Dîner de cons	2000	qen	1	fr	es
10	Volver	2007	08+	2	es	de
11	Le temps va	2020	qen	(null)	es	fr
12	La covid est finie	2021	qen	(null)	fr	es
13	La vie est belle	1999	qen	(null)	en	fr

Table langues

ID_LANGUE	LANGUE
fr	Français
en	Anglais
es	Espagnol
de	Allemand

Importance des Alias

Dans la table Films, nous avons id_langue_dorigine et id_langues qui sont deux FK qui font références à la même table Langues



La question est comment faire pour aller chercher le titre du film, la langue d'origine du film et la langue du film.

On doit faire deux jointures sur la même table langues. La seule façons de la faire est d'utiliser des Alias. C'est comme si la table langues existe deux fois avec des noms différents, Une fois pour les langues d'origine, une autre fois pour la langue du film

Importance des Alias

Pour la requête suivante, nous avons donné :

l'Alias o pour langue d'origine

l'Alias t pour langue de traduction

Nous avons donné des alias la colonne langue. Une fois avec Langue_origine et une autre fois avec Langue_traduction

Quand on donne un alias aux colonnes, on utilise le mot réservé **AS** ce qui n'est pas le cas pour une table

Le résultat de la requête est sur l'acétate suivante.

```
SELECT titre, o.langue as Langue_origine, t.langue as langue_traduction
FROM films f
INNER JOIN langues o on o.id_langue =f.id_langue_origine
INNER JOIN langues t on t.id_langue =f.id_langue;
```

Jointures: importance des Alias

TITRE	LANGUE_ORIGINE	LANGUE_TRADUCTIO
Élémentaire	Anglais	Français
l'incroyable Holmes	Anglais	Français
Holmes à la rescousse	Anglais	Français
Pirates des Caraïbes,Jusqu'au bout du monde	Anglais	Français
Pirates des Caraïbes,le Secret du coffre maudit	Anglais	Français
Pirates des Caraïbes,la Malédiction du Black Pearl	Anglais	Français
La vie est belle	Anglais	Français
Le temps va	Espagnol	Français
Mission Mars	Français	Anglais
Film d'horreur	Français	Anglais
Dîner de cons	Français	Espagnol
La covid est finie	Français	Espagnol
Volver	Espagnol	Allemand

Introduction aux bases de donnés

Les fonctions de groupement.

Fonctions de groupements

Plan de la séance

- Retour sur la dernière séance:
  - Point de vue de l'étudiant
  - Point de vue de l'enseignant.
- Rappels:
  - La commande SELECT et jointures
- Les fonctions:
  - MAX, MIN
  - AVG,SUM
  - COUNT
- La clause GROUP BY
- La clause Having

Définition et exemples

Définition

- Les fonctions de groupement sont des fonctions utilisées pour traiter des groupes de rangées(lignes) et d'afficher un seul résultat.
- Exemples:
  - Nous souhaitons connaître le salaire moyen des employés du département d'informatique
  - Nous souhaitons connaître la masse salariale (somme des salaires de tous les joueurs) du canadien de Montréal
  - Le nombre total d'étudiants au collège Lionel Groulx
  - Le nombre d'étudiants dans chaque programme au collège Lionel Groulx.

Les fonction MIN et MAX

La fonction MIN

Cette fonction permet de chercher le **MINIMUM** parmi l'ensemble des valeurs de la colonne indiquée.

Syntaxe simplifiée

```
SELECT MIN(colonne) FROM nomTable [WHERE expression]
```

Exemple

```
SELECT MIN(salaire) FROM Employes
```

La fonction MAX

Cette fonction permet de chercher le **MAXIMUM** parmi l'ensemble des valeurs de la colonne indiquée.

Syntaxe simplifiée

```
SELECT MAX(colonne) FROM nomTable [WHERE expression]
```

Exemple

```
SELECT MAX(salaire) FROM Employes
```

Les fonctions AVG et SUM

La fonction AVG

Cette fonction permet de chercher la MOYENNE de l'ensemble des valeurs de la colonne indiquée.

Syntaxe simplifiée

```
SELECT AVG(colonne) FROM nomTable [WHERE expression]
```

Exemple

```
SELECT AVG(salaire) FROM Employes WHERE deptno =10
```

La fonction SUM

Cette fonction permet de chercher La SOMME de toutes des valeurs de la colonne indiquée.

Syntaxe simplifiée

```
SELECT SUM(colonne) FROM nomTable [WHERE expression]
```

Exemple

```
SELECT SUM(salaire) FROM Employes WHERE deptno =10
```

La fonction COUNT

La fonction COUNT

Cette fonction permet de compter le nombre de lignes (rangées) qui répondent à un critère.

Syntaxe simplifiée

```
SELECT COUNT ( {*| [DISTINCT |ALL] nomcolonne})  FROM nomTable [WHERE expression]
```

- count(*) On ramène le nombre total, même ceux ayant des colonnes avec des valeurs nulles
- count(colonne) :Dans ce cas on calcule le nombre lignes dans la table selon la colonne indiquée. Seules les lignes avec des valeurs de la colonne indiquée qui sont NOT NULL seront comptées.
- count(ALL colonne) :Dans ce cas on calcule le nombre lignes dans la table selon la colonne indiquée. Les lignes ayant des valeurs nulles pour la colonnes ne seront pas comptées
- count(DISTINCT colonne) :Dans ce cas on calcule le nombre lignes dans la table selon la colonne indiquée. Seules les lignes avec des valeurs de la colonne indiquée qui sont NOT NULL seront comptées. De plus les lignes avec des valeurs identiques de la colonnes indiquées seront comptées une seule fois.

Voir l'exemple de l'acétate après pour mieux comprendre

La fonction COUNT

La fonction COUNT

- **SELECT COUNT (*) FROM joueurs** va retourner 12(on compte tout le monde)
- **SELECT COUNT (numjoueur) FROM joueurs** va retourner 12. car numjoueur est la PK il est donc NOT NULL. Il y a 12 numjoueur
- **SELECT COUNT(nom) FROM Joueurs** va retourner 10 car il y a 10 joueurs avec un nom. (2 ont NULL à nom)
- **SELECT COUNT(ALL nom) FROM Joueurs** va retourner 10 on compte tous les noms. Il y a 10 nom
- **SELECT COUNT (DISTINCT nom) FROM Joueurs** va retourner 9 car on compte que les noms DISTINCT (différents)

Il est recommandé d'utiliser le * lorsque vous voulez compter TOUTES les lignes.

	NUMJOUEUR	NOM	PRENOM	CODEEQUIPE	SALAIRE
1	1	PRICE	CAREY	MTL	1999999
2	2	MARKOV	ANDRÉ	MTL	1546357
3	3	SUBBAN	KARL	MTL	1654657
4	4	PATIORETTY	MAX	MTL	500000
5	10	HAMOND	ANDREW	OTT	1234565
6	6	STONE	MARC	OTT	1234567
7	9	TURIS	KYLE	OTT	870697
8	7	GALLAGHER	BRANDON	MTL	534543
9	8	TANGUAY	ALEX	AVL	1543456
10	11	PRICE	BIL	AVL	798098
11	50	(null)	Leprenom	(null)	(null)
12	52	(null)	Leprenom2	(null)	(null)

La Clause GROUP BY

La clause GROUP BY

- Parfois, il est nécessaire de grouper les enregistrements avant de les compter. Par exemple dans la table joueurs ci-après, comment déterminer le nombre de joueurs dans chaque équipe. ?

Pour répondre à la question, il suffira de grouper les joueurs selon leurs codeequipe, puis les compter. Rien de plus simple.

- Pour grouper des enregistrements on utilise la clause GROUP BY.

	NUMJOUEUR	NOM	PRENOM	CODEEQUIPE	SALAIRE
1	1	PRICE	CAREY	MTL	1999999
2	2	MARKOV	ANDRÉ	MTL	1546357
3	3	SUBBAN	KARL	MTL	1654657
4	4	PATIORETTY	MAX	MTL	500000
5	10	HAMOND	ANDREW	OTT	1234565
6	6	STONE	MARC	OTT	1234567
7	9	TURIS	KYLE	OTT	870697
8	7	GALLAGHER	BRANDON	MTL	534543
9	8	TANGUAY	ALEX	AVL	1543456
10	11	PRICE	BIL	AVL	798098
11	50	(null)	Leprenom	(null)	(null)
12	52	(null)	Leprenom2	(null)	(null)

La Clause GROUP BY

La clause GROUP BY, exemple de la table joueurs précédentes

```
SELECT COUNT(*), codeequipe
FROM joueurs
GROUP BY codeequipe ;
```

a comme résultat

	COUNT(*)	CODEE...
1	2	AVL
2	2	(null)
3	3	OTT
4	5	MTL

Très important pour la clause GROUP BY.

Toutes les colonnes qui apparaissent dans le SELECT doivent apparaitre dans la clause GROUP BY

Exemple, la requête suivante va renvoyer une erreur (*ORA-00979: n'est pas une expression GROUP BY*) car salaire n'est pas dans le GROUP BY alors qu'il est dans SELECT

```
SELECT COUNT(*), codeequipe, salaire FROM joueurs
GROUP BY codeequipe ;
```

La Clause GROUP BY

Autre Exemple:

Pour raffiner la requête, on pourrait utiliser un ALIAS et ordonner le résultat de la requête

```
SELECT COUNT(*),  AS nbJoueurs , codeequipe
FROM joueurs
GROUP BY codeequipe
ORDER BY nbJoueurs DESC;
```

	NBJOUEURS	CODEEQUIPE
1	5	MTL
2	3	OTT
3	2	AVL
4	2	(null)

La Clause HAVING

La clause HAVING:

Parfois, il nécessaire de cibler ou de restreindre les enregistrements spécifiés. Comme par exemple, ne sortir que les codeequipe ayant le nombre de joueurs supérieurs ou égal à 3. Dans ce cas on utilise la clause HAVING.

La clause HAVING permet de mieux cibler les enregistrements spécifiés. Cette clause s'utilise à la place du WHERE une fois que le GROUP BY est réalisé.

```
SELECT COUNT(*) AS nbJoueurs , codeequipe
FROM joueurs
GROUP BY codeequipe
HAVING COUNT(*)>=3
ORDER BY nbJoueurs DESC;
```

Après la clause GROUP BY, un WHERE n'est plus possible. Il faut utiliser HAVING

Dans la clause HAVING , nous ne pouvons pas utiliser l'ALIAS

La Clause HAVING

La clause HAVING:

La requête suivante va renvoyer une erreur *ORA-00904: « nbJoueurs" : identificateur non valide* car nbJoueurs ne doit pas s'utiliser dans la clause HAVING

```
SELECT COUNT(*) AS nbJoueurs , codeequipe
FROM joueurs
GROUP BY codeequipe
HAVING nbJoueurs >=3
ORDER BY nbJoueurs DESC;
```

Les Clauses HAVING et WHERE

La clause WHERE peut-être utilisée dans une requête de groupement, il faudra l'utiliser avant le GROUP BY

La clause HAVING peut-être utilisée dans une requête de groupement, il faudra l'utiliser après le GROUP BY

Lorsque c'est possible, il vaut mieux utiliser un WHERE à la place du HAVING . La requête dans le carrée vert est mieux (Car on réduit les lignes à grouper) que la requête dans le carré gris. Les deux requêtes donnent le même résultats.

```
SELECT COUNT(*) AS nbJoueur,CODEEQUIPE
FROM JOUEURS  WHERE codeequipe ='MTL'
GROUP BY codeequipe;
```

```
SELECT COUNT(*) AS nbJoueur,CODEEQUIPE
FROM JOUEURS
GROUP BY codeequipe
HAVING  codeequipe ='MTL';
```

Les Clauses GROUP BY et HAVING

- Les clauses GROUP BY ET HAVING s'utilisent également sur les fonctions MIN, MAX, AVG et SUM
- Les fonctions de groupements s'utilisent aussi avec des jointures

```
SELECT COUNT(*) AS nbJoueurs,nomEquipe
FROM ( Joueurs INNER JOIN  equipes ON joueurs.codeequipe =equipes.codeequipe)
GROUP BY equipes.NOMEQUIPE
ORDER BY nbJoueurs DESC;
```

```
SELECT AVG(Salaire) AS MoyenneSalaire,nomEquipe
FROM (Joueurs inner join equipes on joueurs.codeequipe =equipes.codeequipe)
GROUP BY equipes.NOMEQUIPE
HAVING AVG(salaire)>1200000;
```

Introduction aux bases de donnés

Les sous-requêtes

Requêtes imbriquées ou sous-requêtes

Plan de la séance

- Retour sur la dernière séance:
  - Point de vue de l'étudiant
  - Point de vue de l'enseignant.
- Rappels:
  - La commande SELECT
- Requêtes imbriquées: Définition
  - Dans la clause WHERE (SELECT, DELETE, UPDATE)
  - Dans la clause FROM
  - Dans la clause SET de la commande UPDATE,
  - La clause VALUES de la commande INSERT
  - Dans la commande CREATE.

Rappel, la commande SELECT

La commande SELECT

- La commande SELECT est la commande la 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.
  - On peut utiliser un SELECT sur une seule TABLE
  - On peur écrire des requêtes SELECT sur plusieurs tables → On utilise des jointures
  - On peut utiliser des requêtes SELECT avec des fonction de groupement
  - On peut écrire des requêtes SELECT utilisant des jointures et des fonctions de groupement

Requêtes imbriquées (sous-Requête), Définition

- Une sous requête est une requête avec la commande **SELECT** imbriquée avec les autres commandes (SELECT, 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 ou CREATE VIEW
- On utilise les sous-requête lorsque les jointures ne sont pas possibles

Sous-requêtes : Dans la clause WHERE
Cas, d'une Requête SELECT

- 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 suivant :

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

- L'écriture d'une telle requête pourrait se présenter sous la syntaxe suivante

```
SELECT colonne1, colonne2, colonnex .....
FROM  nom_tableA
WHERE colonnex  =
                  (
                    SELECT colonnex from nom_tableB....
                  .. Suite requête
                  )
Suite requête
```

Voir explications acétate suivante

Sous-requêtes : Dans la clause WHERE
Cas, d'une Requête SELECT

1. La requête en vert est appelée sous-requête
2. Une sous-requête est obligatoirement entre parenthèses.
3. La colonne du WHERE de la première requête est obligatoirement la colonne du SELECT de la sous-requête. Une fonction de groupement pourrait être appliquée à la colonnex de la sous-requête : voir Exemple 2
4. La première requête et la sous-requête pourraient (et généralement c'est le cas) porter sur la même table. Dans ce cas nom_tableA est identique à nom_tableB: voir exemple 1
5. L'opérateur = pourrait être remplacé par un des opérateurs de l'acétate précédente : voir exemple 4
6. Dans le premier SELECT, colonnex n'est pas obligée d'être dans le SELECT

```
SELECT colonne1, colonne2, colonnex .....
FROM  nom_tableA
WHERE colonnex  =
                  (
                    SELECT colonnex from nom_tableB....
                  .. Suite requête
                  )
Suite requête
```

Sous-requêtes : Dans la clause WHERE
Cas, d'une Requête SELECT

Exemple 1

Voici le contenu de la table syemp de votre labo1.

La question: On cherche le nom des employés qui travaillent dans le même département que l'employé FORD

Étape 1: on cherche le département (deptno) de l' employé FORD

Étape 2: on cherche le ename des employés ayant le même deptno que FORD

```
SELECT ename    FROM syemp
WHERE deptno =
                        (
                          SELECT deptno FROM syemp
                          WHERE ename ='FORD'
                        );
```

syemp

EMPNO	ENAME	SAL	DEPTNO
7839	KING	5000	10
7698	BLAKE	2850	30
7782	CLARK	2450	10
7566	JONES	2975	20
7902	FORD	3000	20
7369	SMITH	800	20
7499	ALLEN	1600	30
7521	WARD	1250	30
7654	MARTIN	1250	30
7844	TURNER	1500	30
7900	JAMES	950	30
7934	MILLER	1300	10

Remarque

Les deux requêtes portent sur la même table: syemp.

Sous-requêtes : Dans la clause WHERE
Cas, d'une Requête SELECT

Exemple 2

Voici le contenu de la table syemp de votre labo1.

La question: On cherche le nom de l'employé qui a le plus haut salaire. :**MAX(sal)**

L'idée est d'aller chercher le plus haut salaire (MAX), puis chercher le nom de l'employé à qui ça correspond.

Étape 1: on cherche le salaire le plus élevé:

Étape 2: on cherche le ename avec le plus haut salaire.

```
SELECT ename   FROM syemp
WHERE sal =
            (
           SELECT max(sal) FROM syemp
            );
```

syemp

EMPNO	ENAME	SAL	DEPTNO
7839	KING	5000	10
7698	BLAKE	2850	30
7782	CLARK	2450	10
7566	JONES	2975	20
7902	FORD	3000	20
7369	SMITH	800	20
7499	ALLEN	1600	30
7521	WARD	1250	30
7654	MARTIN	1250	30
7844	TURNER	1500	30
7900	JAMES	950	30
7934	MILLER	1300	10

Attention:

Les deux requêtes portent sur la même table: syemp.

Il y a une fonction de groupement sur la colonne sal de la sous-requête.

Sous-requêtes : Dans la clause WHERE
Cas, d'une Requête SELECT

Exemple 3

Voici le contenu des tables syemp et sydept de votre labo1.

La question: On cherche le nom, ename des employés du département SALES.

Étape 1: On cherche le deptno du département SALES

Étape 2, On cherche les employés ayant le deptno

```
SELECT ename SELECT syemp
WHERE deptno =
            (
            SELECT deptno FROM sydept
            WHERE dname ='SALES'
            );
```

Remarque: la requête précédente aurait pu s'écrire avec une jointure.

Attention: Il faut toujours utiliser des jointures à la place de sous requête. C'est une obligation

syemp

EMPNO	ENAME	SAL	DEPTNO
7839	KING	5000	10
7698	BLAKE	2850	30
7782	CLARK	2450	10
7566	JONES	2975	20
7902	FORD	3000	20
7369	SMITH	800	20
7499	ALLEN	1600	30
7521	WARD	1250	30
7654	MARTIN	1250	30
7844	TURNER	1500	30
7900	JAMES	950	30
7934	MILLER	1300	10

sydept

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

Sous-requêtes : Dans la clause WHERE
Cas, d'une Requête SELECT

Exemple 4

Voici le contenu de la table syemp de votre labo1.

La question: On cherche le nom des employés qui travaillent dans le même département que l'employé FORD ou l'employé MILLER

La sous-requête renvoie plus qu'une valeur, elle renvoie le deptno 20 et le deptno 10. Dans ce cas il faut utiliser IN à la place de =

```
SELECT ename    FROM syemp
WHERE deptno IN
                        (
                        SELECT deptno FROM syemp
                        WHERE ename ='FORD' OR ename ='MILLER'
                         );
```

syemp

EMPNO	ENAME	SAL	DEPTNO
7839	KING	5000	10
7698	BLAKE	2850	30
7782	CLARK	2450	10
7566	JONES	2975	20
7902	FORD	3000	20
7369	SMITH	800	20
7499	ALLEN	1600	30
7521	WARD	1250	30
7654	MARTIN	1250	30
7844	TURNER	1500	30
7900	JAMES	950	30
7934	MILLER	1300	10

Attention La requête interne renvoie plus qu'un résultat. Il faut utiliser IN dans ce cas.

Si = est utilisé dans la requête alors vous aurez cette erreur:

```
ORA-01427: sous-requête ramenant un enregistrement de plus d'une ligne
01427. 00000 -  "single-row subquery returns more than one row"
```

Sous-requêtes : Dans la clause WHERE
Cas, d'une Requête SELECT

Les opérateurs ANY et ALL

- Ces deux opérateurs s'utilisent lorsque la sous requête retourne plus qu'un enregistrement.
- On utilise l'opérateur ANY pour que la comparaison se fasse pour toutes les valeurs retournées. Le résultat est vrai si **au moins une des valeurs** répond à la comparaison
- On utilise l'opérateur ALL pour que la comparaison se fasse pour toutes les valeurs retournées. Le résultat est vrai si **toutes les valeurs** répondent à la comparaison

Sous-requêtes : Dans la clause WHERE
Cas, d'une Requête SELECT

Exemple 5

La question: Nous souhaitons chercher le nom des employés du département numéro 10 dont le salaire est plus grand que TOUS les employés du département 30

```
SELECT ename FROM syemp
WHERE deptno =10 AND sal > ALL
                        (
                         SELECT sal FROM syemp
                         WHERE deptno=30
                         );
```

Le résultat de cette requête sera KING, car la sous-requête (les salaires du département 30) est donné ci-après: ----------→

La requête principale va chercher les employés du département 10 dont le salaire est supérieur à tous les salaires de la liste précédente. **Seul KING** répond à ce critère.

syemp

EMPNO	ENAME	SAL	DEPTNO
7839	KING	5000	10
7698	BLAKE	2850	30
7782	CLARK	2450	10
7566	JONES	2975	20
7902	FORD	3000	20
7369	SMITH	800	20
7499	ALLEN	1600	30
7521	WARD	1250	30
7654	MARTIN	1250	30
7844	TURNER	1500	30
7900	JAMES	950	30
7934	MILLER	1300	10

Salaires du département 30

SAL
2850
1600
1250
1250
1500
950

Sous-requêtes : Dans la clause WHERE
Cas, d'une Requête SELECT

Exemple 6

La question: Nous souhaitons chercher le nom des employés du département numéro 10 dont le salaire est plus grand que n'importe lequel des salaires du département 30

```
SELECT ename FROM syemp
WHERE deptno =10 AND sal > ANY
                        (
                         SELECT sal FROM syemp
                         WHERE deptno=30
                         );
```

Le résultat de cette requête sera KING, car la sous-requête (les salaires du département 30) est donné ci-après: ----------→

La requête principale va chercher les employés du département 10 dont le salaire est supérieur à n'importe lequel des salaires de la liste précédente. Le résultat sera:

ENAME
KING
CLARK
MILLER

syemp

EMPNO	ENAME	SAL	DEPTNO
7839	KING	5000	10
7698	BLAKE	2850	30
7782	CLARK	2450	10
7566	JONES	2975	20
7902	FORD	3000	20
7369	SMITH	800	20
7499	ALLEN	1600	30
7521	WARD	1250	30
7654	MARTIN	1250	30
7844	TURNER	1500	30
7900	JAMES	950	30
7934	MILLER	1300	10

Salaires du département 30

SAL
2850
1600
1250
1250
1500
950

Sous-requêtes : Dans la clause FROM de la commande SELECT

- Parfois, nous avons besoin d'extraire des informations à partir de résultats calculés, ces résultats sont ramenés par une requête SELECT. Dans ce cas nous parlons de requêtes imbriquées dans le FROM.

La question: nous souhaitons extraire le nom des employés ayant les 3 meilleurs salaires, donc les trois premières lignes (KING, FORD, JONES)

Dans SQL Server, la colonne qui indique le numéro de ligne est appelée **ROWNUM**

```
SELECT ename
FROM
        (
           SELECT ename , sal, deptno FROM syemp
           ORDER BY sal DESC
        )
WHERE ROWNUM <=3;
```

Syemp, ordonnée par SAL DESC

	ENAME	SAL	DEPTNO
1	KING	5000	10
2	FORD	3000	20
3	JONES	2975	20
4	BLAKE	2850	30
5	CLARK	2450	10
6	ALLEN	1600	30
7	TURNER	1500	30
8	MILLER	1300	10
9	MARTIN	1250	30
10	WARD	1250	30
11	JAMES	950	30
12	SMITH	800	20

Sous-requêtes : Dans la clause SET d'une requête UPDATE

- On peut chercher des valeurs existantes dans la base de données afin de mettre à jour des données d'une table.
- La sous-requête est une requête SELECT qui ne doit ramener qu'un seul résultat.

La question: Nous souhaitons mettre à jour les commissions null, par la moyenne de toutes les commissions. La requête serait:

```
UPDATE emp SET comm =
                  (
                   SELECT AVG(comm) FROM emp
                  )
WHERE comm IS NULL ;
```

Les commissions (comm) a NULL seront mises à jour par la valeur: 550 qui est la moyennes de toutes les comm

emp

	ENAME	SAL	COMM
1	KING	5000	(null)
2	BLAKE	2850	(null)
3	CLARK	2450	(null)
4	JONES	2975	(null)
5	FORD	3000	(null)
6	SMITH	800	(null)
7	ALLEN	1600	300
8	WARD	1250	500
9	MARTIN	1250	1400
0	TURNER	1500	0
1	JAMES	950	(null)
2	MILLER	1300	(null)

Sous-requêtes : Dans la clause VALUES d'une requête 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 ETUDIANTS (numad,nom)
                                    (
                                     SELECT empno,ename FROM syemp
                                    WHERE deptno =10
                                    );
```

Attention !

Lors de l'insertion, les contraintes d'intégrité doivent être respectées

Sous-requêtes : Dans la commande CREATE TABLE

- Ce type de requête est utilisé pour créer une table à partir d'une ou plusieurs table existantes. La table créée contient les données issues d'une ou de plusieurs table.
- Pour créer une table avec une sous requête, on utilise la syntaxe:

CREATE TABLE …. AS (SELECT …)

Exemple 1, La table emp contient exactement les mêmes colonnes et les mêmes données que la table syemp.

```
CREATE TABLE emp AS
                    (
                     SELECT * FROM syemp
                    );
```

Quel est l'intérêt de dupliquer ainsi une table ?

Dans ce cas précis, nous avons plus de privilèges sur la table emp. (nous pouvons faire des UPDATE, INSERT, DELETE sur emp mais pas sur syemp)

Sous-requêtes : Dans la commande CREATE TABLE

Exemple 2, regrouper les données éparpillée.

```
CREATE TABLE notesKB6 (nomEtudiant, PrenomEtudiant,titreCours,noteKB6) AS
        SELECT nom, prenom,titrecours,note
        FROM ((etudiants E
           INNER JOIN resultats R  ON E.numad=R.numad)
           INNER JOIN cours C ON C.codecours=R.codecours)
        WHERE titrecours ='Introduction aux bases de données'
        ORDER BY note DESC;
```

Nous avons utilisé les tables: Etudiants, Cours et Resultats pour créer les la tables notesKB6

Les colonnes de la table notesKB6 sont différentes des colonnes des tables sources .

Attention ! La modification des données des tables Etudiants, Cours et Resultats n'implique pas la modifications de la table noteKB6.

Les sous-requêtes

Conclusion

Questions

Introduction aux bases de donnés

Les vues, pour simplifier les requêtes

Les vues

Plan de la séance

- Rappels:
  - Les sous-requêtes
- Les vues:
  - Définition
  - Avantages
  - Syntaxe
  - Exemples
  - Supprimer/renommer une vue
  - Conclusion

Les vues

Définition

- Une vue est objet de la base de donnée. Nous pouvons la voir comme une table dont les données ne sont pas physiquement stockées mais se réfèrent à des données stockées dans d'autres tables. C'est une fenêtre sur la base de données permettant à chacun de voir les données comme il le souhaite.

- On peut ainsi définir plusieurs vues à partir d'une seule table ou créer une vue à partir de plusieurs tables. Une vue est interprétée dynamiquement à chaque exécution d'une requête qui y fait référence.

Les vues

Avantages

- Les vues permettent de simplifier la commande SELECT avec les sous requêtes complexes. On peut créer une vue pour chaque sous-requête complexe, ce qui facilite sa compréhension

- Il est possible de rassembler dans un seul objet (vue) les données éparpillées

- Les vues permettent de protéger l'accès aux tables en fonction de chacun des utilisateurs. On utilise une vue sur une table et on interdit l'accès aux tables. C'est donc un moyen efficace de protéger les données

- Une vue se comporte dans la plupart des cas comme une table. On peut utiliser une vue comme source de données dans les commandes SELECT, INSERT, UPDATE ou DELETE.

Les vues

Syntaxe:

```
CREATE [OR REPLACE ][FORCE] VIEW
<nom_de_la_vue> AS <sous_requête>
[WITH CHECK OPTION]
```

OR REPLACE : commande optionnelle qui indique lors de la création de la vue de modifier la définition de celle-ci ou de la créer si elle n'existe pas

FORCE : permet de créer la vue même si les sources de données n'existent pas.

WITH CHECK OPTION : cette option permet de contrôler l'accès à la vue et par conséquent à la table dont elle est issue. (Sécurité des données)

Les vues

Exemple 1, regrouper des données éparpillées.

La vue suivante regroupe des données issues de trois tables: etudiants, resultats et cours.

- *numad, nom , prenom* sont dans la table etudiants
- *Titrecours* est dans la table cours.
- *Note* est dans la table resultats

```
CREATE VIEW vResultats as
SELECT etudiants.numad,nom, prenom, titrecours,note
FROM ((etudiants
   INNER JOIN resultats  ON etudiants.numad=resultats.numad)
   INNER JOIN cours  ON cours.codecours =resultats.codecours);
```

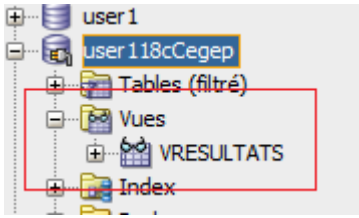
Les vues

Exemple 1 (suite), regrouper des données éparpillées.

Lorsque je fais SELECT * FROM vResultats;
on obtient le résultats de la figure ci-après.

NUMAD	NOM	PRENOM	TITRECOURS	NOTE
10	Patoche	Alain	Programmation orientée objet	85
10	Patoche	Alain	Introduction aux bases de données	65
10	Patoche	Alain	Projet dirigé	60
11	Poirier	Juteux	Programmation orientée objet	70
11	Poirier	Juteux	Programmation Web	75
11	Poirier	Juteux	Introduction aux bases de données	85
12	Fafar	Anick	Programmation orientée objet	60
12	Fafar	Anick	Programmation Web	75
12	Fafar	Anick	Introduction aux bases de données	77
16	Saturne	Francis	Programmation Web	66
16	Saturne	Francis	Projet dirigé	88

Lorsque vous allez dérouler l'onglet Vues,
vous remarquerez que votre vue est créée



Les vues

Exemple 2, la mise à jour est dynamique

Lorsque je fais SELECT * FROM vResultats; on obtient le résultats de la figure suivante.

On remarque qu'il y a 11 enregistrements. Et remarquez la note de l'étudiant numéro 10 dans le cours de Projet dirigé

NUMAD	NOM	PRENOM	TITRECOURS	NOTE
10	Patoche	Alain	Programmation orientée objet	85
10	Patoche	Alain	Introduction aux bases de données	65
10	Patoche	Alain	Projet dirigé	60
11	Poirier	Juteux	Programmation orientée objet	70
11	Poirier	Juteux	Programmation Web	75
11	Poirier	Juteux	Introduction aux bases de données	85
12	Fafar	Anick	Programmation orientée objet	60
12	Fafar	Anick	Programmation Web	75
12	Fafar	Anick	Introduction aux bases de données	77
16	Saturne	Francis	Programmation Web	66
16	Saturne	Francis	Projet dirigé	88

Les vues

Exemple 2, (suite)

Si on fait les instructions suivantes:

insert into resultats values (16,'KB6',85);

insert into resultats values (12,'KBE',90);

update resultats set note =62

where numad =10 and codecours ='KBE';

Puis on fait un SELECT * FROM vResultats on obtient la figure ci-après

On remarque que la vue s'est mise jour automatiquement: Les deux enregistrements ont été ajoutés et la note de l'étudiant 10 dans le cours de Projet dirigé est passée à 62.

NUMAD	NOM	PRENOM	TITRECOURS	NOTE
10	Patoche	Alain	Programmation orientée objet	85
10	Patoche	Alain	Introduction aux bases de données	65
10	Patoche	Alain	Projet dirigé	62
11	Poirier	Juteux	Programmation orientée objet	70
11	Poirier	Juteux	Programmation Web	75
11	Poirier	Juteux	Introduction aux bases de données	85
12	Fafar	Anick	Programmation orientée objet	60
12	Fafar	Anick	Programmation Web	75
12	Fafar	Anick	Introduction aux bases de données	77
12	Fafar	Anick	Projet dirigé	90
16	Saturne	Francis	Programmation Web	66
16	Saturne	Francis	Introduction aux bases de données	85
16	Saturne	Francis	Projet dirigé	88

Conclusion: Les vues sont dynamiques. Si les données des tables dont la vue est issues sont modifiées alors les données de la vue sont modifiées en conséquence.

Les vues

Exemple 3, simplification de requêtes.

On cherche:

Q1 - Les 3 meilleurs étudiants (les 3 meilleurs notes) dans le cours de Introduction aux bases de données.

Q2- Les étudiants avec la meilleurs note (la plus haute note)

Pour répondre à la question Q1 on utilise une sous requête dans la clause FROM.

```
SELECT * FROM (
                        SELECT etudiants.numad,nom, prenom, titrecours,note
                        FROM ((etudiants
                                INNER JOIN resultats  ON etudiants.numad=resultats.numad)
                                INNER JOIN cours  ON cours.codecours =resultats.codecours)
                       WHERE titrecours='Introduction aux bases de données'
                       ORDER BY note desc
                    )
WHERE ROWNUM<=3;
```

Les vues

Exemple 3, (suite)

On voit bien que la requête précédente est très longue.

Le résultat de la sous-requête est le suivant:

	NUMAD	NOM	PRENOM	TITRECOURS	NOTE
1	11	Poirier	Juteux	Introduction aux bases de données	85
2	16	Saturne	Francis	Introduction aux bases de données	85
3	12	Fafar	Anick	Introduction aux bases de données	77
4	10	Patoche	Alain	Introduction aux bases de données	65

La question: Peut-on mettre la requête en rouge (la sous-requête) dans une vue afin de simplifier la requête et de pouvoir réutiliser le résultat pour d'autres requêtes? Comme par exemple la question Q2.

La réponse est oui.

Les vues

Exemple 3, (suite)

Pour simplifier la requête, on va d'abord créer la vue des résultats des étudiants dans le cours d'introduction aux bases de données: Cette vue sera ordonnée par ordre décroissant des notes

```
CREATE VIEW VresultatsKB6 AS
                    SELECT etudiants.numad,nom, prenom, titrecours,note
                     FROM ((etudiants
                              INNER JOIN resultats  ON etudiants.numad=resultats.numad)
                              INNER JOIN cours  ON cours.codecours =resultats.codecours)
                    WHERE titrecours='Introduction aux bases de données'
                    ORDER BY note desc;
```

Puis, on répond aux questions Q1 et Q2 (voir la page suivante)

Les vues

Exemple 3, (suite)

Pour répondre à Q1 - les 3 meilleurs étudiants (les 3 meilleurs notes) dans le cours de Introduction aux bases de données

```
SELECT  * FROM  VresultatsKB6 WHERE ROWNUM <=3;
```

Pour répondre à Q2: Les étudiants avec la meilleurs note (la plus haute note)

```
SELECT  * FROM VresultatsKB6 WHERE note =
                                    (
                                    SELECT MAX(note) FROM VresultatsKB6
                                     );
```

Les vues

Supprimer et renommer une vue.

Pour détruire une vue: DROP VIEW nom_de_la_Vue

Exemple : **DROP VIEW VresultatsKB6;**

Pour renommer une vue: RENAME ancien_nom TO nouveau

Exemple:

RENAME VresultatsKB6 to ResultatsIntroBD;

Les vues

Conclusion

- Les vues sont des objets de la base de données. On utilise la commande CREATE pour créer une vue. On utilise la commande DROP pour détruire une vue.
- Une vue se crée à l'aide d'une sous-requête.
- Les vues sont dynamiques : ce qui veut dire que la vue se met à jour automatiquement lorsque la table (les tables) dont elle issue est mise à jour.
- Les vues servent à simplifier l'écriture de requêtes.
- Les vues servent à mettre ensemble des données éparpillées.
- Les vues servent à protéger les données de la BD (cours de session 3).

Les vues

Conclusion

Questions