

GPA775

1

# Bases de données

LANGAGE SQL

GPA775

2

## Langage SQL Introduction

- ▶ Le langage de consultation le plus utilisé par les SGBDR modernes est le SQL (*Structured Query Langage*).
- ▶ SQL est fortement basé sur l'algèbre relationnelle et le calcul relationnel sur tuple. Néanmoins, il ne respecte pas entièrement les notions initiales présentées dans le modèle relationnel de Codd.
- ▶ Il existe plusieurs différences qui peuvent sembler mineures à première vue, en voici quelques unes:
  - ▶ le SQL permet la duplication des lignes d'une table alors que l'AR interdit la duplication de tuples;
  - ▶ l'AR stipule que chaque attribut d'une relation doit avoir un nom unique alors que le SQL permet la création de colonnes ayant le même nom et même des colonnes sans nom;
  - ▶ formellement, plusieurs sens différents peut être donné à une valeur nulle : toutefois, il existe des différences majeures entre ce que permet l'AR et le SQL;
  - ▶ ...

GPA775

3

## Langage SQL

### Introduction

- ▶ Le langage a été standardisé en 1986 (ANSI).
- ▶ Le langage a subi une révision majeure en 1992 et a été normalisé (ISO 9075).
- ▶ En 2011, SQL subit sa 7<sup>e</sup> révision majeure – technologie de l'information incluant principalement la notion de base de données temporelle.
- ▶ Malgré une norme bien établie, son implémentation varie de façon plus ou moins importante d'un SGBD à un autre.
- ▶ Ainsi, les SGBD existants tels que *Oracle*, *MySQL*, *MS Access*, *SQL Server* et tous les autres respectent généralement la norme mais présentent plusieurs particularités qui sont souvent ennuyeuses.

GPA775

4

## Langage SQL

### Introduction

- ▶ Le langage SQL permet une manipulation efficace de toutes les opérations liées à la base de données (n'est pas nécessairement lié directement aux opérations de gestion du SGBD).
- ▶ SQL est un langage non procédural qui spécifie ce qui doit être fait au lieu de comment le faire. Il fait partie des langages de 4<sup>e</sup> génération.
- ▶ En plus des manipulations conventionnelles, le langage SQL possède une multitude d'outils permettant de gérer les vues, les index, les fonctions, les déclencheurs, les transactions, les usagers, ...
- ▶ SQL est si répandu que tous les environnements de développement possèdent une implémentation d'outils supportant son usage (C, C++, Pascal, Python, Ruby, Excel, Matlab, web, ...).

GPA775

5

# Langage SQL

## Introduction

- ▶ Avantages :
  - ▶ langage standardisé indépendant du SGBD utilisé et de la complexité sous-jacente;
  - ▶ langage de 4<sup>e</sup> génération simplifiant l'apprentissage et réduisant le temps de développement;
- ▶ Impacts :
  - ▶ les interfaces communes du langage facilitent le transfert d'applications sur les différentes plateformes, les différents environnements de développement et opérationnels;
  - ▶ permet de réduire les coût de formation, de développement et de maintenance;
  - ▶ augmentation significative de la durée de vie des projets.

GPA775

6

# Langage SQL

## Introduction

- ▶ Inconvénients :
  - ▶ SQL comporte des limitations technologiques qui ne permettent pas de réaliser le plein potentiel du modèle relationnel tel que présenté par Codd;
  - ▶ les diverses implémentations du langage possèdent des variantes particulières qui apportent des irritants importants et réduits certains avantages annoncés;
  - ▶ l'optimisation des requêtes SQL n'est pas toujours claire et il est parfois difficile de connaître l'impact réelle d'écrire une requête d'une façon plutôt que d'une autre.

GPA775

7

## Langage SQL

### Introduction

- ▶ Malgré le fait que l'apprentissage des règles de base du langage SQL soit relativement aisé, la présentation stricte du langage peut être très ardu et surtout déroutante pour ceux qui débutent.
- ▶ En effet, pour l'ensemble des commandes, il existe plusieurs options qui font souvent référence à des notions plus avancées. Nous ne présenterons ici qu'une version simplifiée des principales commandes en omettant les options plus avancées.
- ▶ Pour plusieurs raisons, les différents SGBD commerciaux présentent des différences majeures quant aux objets disponibles et aux éléments syntaxiques pour les adresser. Nous verrons les grandes lignes des deux systèmes les plus utilisés dans le contexte du génie : Oracle et MySQL – noté **O** et **M**.

GPA775

8

## Langage SQL

### Nomenclature et objets d'un SGBD

- ▶ Le langage SQL utilise cette nomenclature spécifique :
  - ▶ objets : entités des SGBD telles que les tables, indexes, vues, usagers, ...
  - ▶ table : équivalent d'une relation du MR
  - ▶ colonne : équivalent à un attribut du MR (structure verticale d'une table)
  - ▶ ligne : équivalent à un tuple du MR (structure horizontale d'une table)
  - ▶ déclaration : une commande SQL (écrite sous forme de script)

GPA775

9

# Langage SQL

## Nomenclature et objets d'un SGBD

- ▶ Il existe un très grand nombre d'objets pour les SGBD, voici les principaux à connaître et que nous verrons (de façon plus ou moins exhaustive) :
  - ▶ base de données
  - ▶ schémas
  - ▶ tables
  - ▶ indexes
  - ▶ vues
  - ▶ synonymes
  - ▶ séquences
  - ▶ procédures
  - ▶ Fonctions
  - ▶ déclencheurs
  - ▶ usagers
  - ▶ ...

GPA775

10

# Langage SQL

## Nomenclature et objets d'un SGBD

- ▶ Base de données :
  - ▶ pour un SGBD, le concept de base de données correspond à un espace logique regroupant un ensemble d'objets nécessaires à la gestion relative à un ensemble de données jointes logiquement (comme pour un projet par exemple);
  - ▶ c'est une couche d'abstraction qui permet une gestion plus efficace de plusieurs projets utilisant une base de données;
  - ▶ ainsi, un SGBD peut faire la gestion de plusieurs bases de données à la fois.

GPA775

11

## Langage SQL

### Nomenclature et objets d'un SGBD

- ▶ Schéma :
  - ▶ la notion de schéma est une extension du concept de base de données appliquée aux utilisateurs;
  - ▶ plus précisément, un schéma est un regroupement logique des objets créés par un utilisateur à l'intérieur d'une base de données;
  - ▶ ainsi chaque utilisateur possède son propre schéma qu'il peut partager ou non;
  - ▶ la notion de schéma est très utilisée dans Oracle alors qu'elle n'existe pas dans MySQL (d'ailleurs, pour MySQL, le terme SCHEMA est un synonyme de DATABASE).

GPA775

12

## Langage SQL

### Nomenclature et objets d'un SGBD

- ▶ Table :
  - ▶ les tables sont au cœur des SGBDR, elles correspondent aux relations du modèle relationnel;
  - ▶ toutes les données manipulées par le SGBD sont stockées dans les tables;
  - ▶ chaque table est constituée d'un nom, de colonnes et de contraintes;
  - ▶ les contraintes définies par le concepteur sont des guides qui permettent au SGBD d'assurer l'intégrité des données.

GPA775

13

## Langage SQL

### Nomenclature et objets d'un SGBD

- ▶ **Indexe :**
  - ▶ un indexe est une structure de données complémentaire et optionnelle servant à l'optimisation de la recherche de ligne (de tuple ou d'enregistrement) dans une table;
  - ▶ il permet un accès beaucoup plus rapide aux données;
  - ▶ l'utilisation des indexs est très simple car leur création est facile et leur usage est implicite par le système lorsqu'ils sont définis (aucune modification aux requêtes écrites).

GPA775

14

## Langage SQL

### Nomenclature et objets d'un SGBD

- ▶ **Vue :**
  - ▶ une vue est une construction logique faites à partir de table(s) existante(s);
  - ▶ elle ne contient aucune données en soit, elle n'est qu'une représentation indirecte de données contenues dans d'autres tables;
  - ▶ les tables utilisées pour la construction d'une vue sont dites : tables de base;
  - ▶ une vue peut servir de base à une autre vue;
  - ▶ les vues sont très utilisées pour simplifier et optimiser l'usage de structures intermédiaires souvent sollicitées;
  - ▶ elles sont constamment mises à jour par le SGBD.

GPA775

15

## Langage SQL

### Nomenclature et objets d'un SGBD

- ▶ **Synonyme :**
  - ▶ un synonyme est un nom alternatif pour un objet existant;
  - ▶ ils sont très utilisés pour faciliter la gestion du système puisqu'ils présentent une indépendance face à la source des données (très utilisés pour une gestion efficace des usagers);
  - ▶ comme pour la notion de schéma, le concept de synonyme est un outil puissant que possède Oracle mais qui n'existe pas dans MySQL (il existe des méthodes indirectes pour simuler les synonymes sous MySQL – mais ces artifices sont des équivalents incomplets).

GPA775

16

## Langage SQL

### Nomenclature et objets d'un SGBD

- ▶ **Séquence :**
  - ▶ une séquence est un objet qui permet de générer une suite de nombres automatiquement incrémentés;
  - ▶ ces objets n'existent pas sous MySQL qui utilise un autre mécanisme pour permettre l'auto incrémentation d'une valeur (la contrainte AUTO\_INCREMENT);
  - ▶ les séquences sont disponibles sous Oracle et permettent une gestion plus flexible que la technique utilisée par MySQL.



GPA775

17

## Langage SQL

### Nomenclature et objets d'un SGBD

- ▶ Procédure et fonction :
  - ▶ ces objets sont des sous-routine écrites par les développeurs afin de consolider plusieurs instructions en un ensemble facilement réutilisable;
  - ▶ leur usage est avantageux et permettent une meilleur modularité du développement;
  - ▶ un des désavantages importants de leur usage est que les SGBD rendent disponible des outils de programmation différents et que lors du passage d'un SGBD à un autre, il faut souvent réécrire le code existant;
  - ▶ une **procédure** doit absolument être appelée avec l'utilisation de l'instruction CALL ou EXECUTE – de plus elle peut retourner plusieurs données et une table intermédiaire;
  - ▶ une **fonction** peut être appelée directement à même n'importe quelle instruction – par contre, elle ne peut que retourner un scalaire.

GPA775

18

## Langage SQL

### Nomenclature et objets d'un SGBD

- ▶ Déclencheur :
  - ▶ le concept de déclencheur (TRIGGER) est un mécanisme permettant l'appel automatique d'une procédure en réaction à des événements spécifiques (principalement lors de la manipulation des tables et des vues);
  - ▶ le rôle des déclencheurs est très importants pour les systèmes un peu plus complexes et permet d'assurer l'intégrité des données liées à un plus haut niveau – par exemple, en insérant une nouvelle entrée dans la table client on pourrait vouloir ajuster automatiquement le montant de la prime en fonction du sexe;
  - ▶ pour cet aspect, MySQL offre plusieurs limitations lorsqu'on le compare à Oracle.

GPA775

19

## Langage SQL

### Nomenclature et objets d'un SGBD

- ▶ Usager :
  - ▶ un usager est un principe lié à la sécurité d'accès aux objets d'une base de données, principalement les données des tables;
  - ▶ un usager peut être un administrateur du système ou un utilisateur (au sens large – un individu ou une application);
  - ▶ chaque objet « base de données » d'un SGBD possède une liste d'utilisateurs qui ont chacun un mot de passe et des droits spécifiques définis par l'administrateur;
  - ▶ il existe plusieurs types de droits possibles mais mentionnons ces deux plus importants : les droits liés aux actions et à l'accès aux données;
  - ▶ les différents SGBD offrent plusieurs stratégies de gestion qui peuvent devenir assez élaborées (Oracle).

GPA775

20

## Langage SQL

### Présentation générale

- ▶ Concepts de base de SQL :
  - ▶ Chaque déclaration peut retourner une erreur spécifique au contexte des instructions.
  - ▶ Le langage SQL n'est pas sensible à la casse, cependant l'usage des majuscules pour les mots réservés est recommandé.
  - ▶ Selon ANSI-SQL-92, chaque instruction devrait se terminer par un « ; ». Néanmoins, plusieurs SGBD ne l'exigent pas et fonctionnent sans. De plus, pratiquement tous les SGBD permettent l'omission des « ; » pour une seule ligne d'instruction. Toutefois, il est fortement recommandé de mettre les points virgules pour chaque ligne d'instruction.

GPA775

21

## Langage SQL Présentation générale

- ▶ Les chaînes de caractères sont insérées entre apostrophe : 'Voilà!'
- ▶ Un apostrophe dans une chaîne de caractères est précédé par \ : 'voilà l\'adresse'
- ▶ L'insertion de commentaires se fait par :
  - ▶ -- (tiret + tiret + espace) ou # pour une seule ligne de commentaires
  - ▶ /\* ... \*/ pour plusieurs lignes de commentaires

```
-- Voici un premier commentaire
#Voici un deuxième commentaire, la ligne qui suit est une ligne d'instruction SQL

SELECT * FROM Etudiant; -- ...suivi par un commentaire de fin de ligne

/* Voilà plusieurs commentaires présentés
sur plusieurs lignes */
```

GPA775

22

## Langage SQL Présentation générale

- ▶ On divise le langage SQL en 4 parties :
 

▶ Langage de définition des données LDD	<b>DDL</b>
▶ Langage de manipulation des données LMD	<b>DML</b>
▶ Langage de contrôle des données LCD	<b>DCL</b>
▶ Langage de contrôle des transactions LCT	<b>TCL</b>
- ▶ Cette présentation présente progressivement les concepts au lieu d'aborder en détail chaque fonctions et clause.

GPA775

23

## Langage SQL DDL | DATABASE

- Les objets de type base de données (DATABASE) se gèrent principalement par :

```

OM CREATE DATABASE nom_bd;

OM RENAME DATABASE nom_bd TO nouveau_nom_bd
    [, nom_bd2 TO nouveau_nom_bd2, ...];

OM DROP DATABASE nom_bd;
M USE nom_bd;
  
```

*attention à cette instruction*

*attention cette instruction ne fait pas partie du DDL mais plutôt du TCL*

GPA775

24

## Langage SQL DDL | TABLE

- Les objets de type table (TABLE) se gèrent principalement par :

```

OM CREATE TABLE ...
OM ALTER TABLE ...
OM RENAME TABLE nom_table TO nouveau_nom_table
    [, nom_table2 TO nouveau_nom_table2, ...];
O DROP TABLE nom_table
    [CASCADE [CONSTRAINTS]] [PURGE];
M DROP TABLE [IF EXISTS] nom_table;
  
```

*voir prochaines diapositives*

*voir prochaines diapositives*

*attention à cette instruction*

*attention à cette instruction*

GPA775

25

# Langage SQL

## DDL | TABLE

- ▶ L'instruction CREATE TABLE contient une très grande variété d'option et sa structure contient plusieurs éléments importants.
- ▶ Outre le nom de la table, les paramètres importants à spécifier sont :
  - ▶ le nom et le type de chaque colonne,
  - ▶ les contraintes liées aux colonnes.

```
OM CREATE TABLE nom_table (
    nom_coll      type_coll      [contrainte_coll]
    [, nom_coll2  type_coll2     [contrainte_coll2], ...]

    [CONSTRAINT nom_contrainte_sup NOM_CONTRAINT option_contrainte, ...]
)
```

GPA775

26

# Langage SQL

## DDL | TABLE | Types de données

```
-- Types numériques entiers à valeurs exactes
M TINYINT [(p)]                8 bits
M SMALLINT [(p)]              16 bits
M MEDIUMINT [(p)]            24 bits
M INT [EGER] [(p)]            32 bits
M BIGINT [(p)]                64 bits

-- Types numériques à points flottant à valeurs exactes
O NUMBER[(p[, s])]
M NUMERIC[(p[, s])] - DECIMAL[(p[, s])]    identiques

-- Types numériques à points flottant à valeurs approximatives
O FLOAT[(n)]                    n bits (de 1 à 126)
M FLOAT[(p[, s])]              32 bits
M DOUBLE[(p[, s])]             64 bits

-- p = nombre total de chiffres
-- s = nombre de chiffre après le point
-- n = nombre de bits de précision
```

GPA775

27

# Langage SQL

## DDL | TABLE | Types de données

```
-- Types caractères
OM CHAR[(n)]                               chaîne de caractères de longueur fixe

O  VARCHAR2[(n)]                           chaîne de caractères de longueur variable
M  VARCHAR[(n)]                             n = longueur maximum de la chaîne

M  TINYTEXT                                chaîne de caractères de longueur variable max. 255
M  MEDIUMTEXT                             chaîne de caractères de longueur variable max. 16 M
M  LONGTEXT                               chaîne de caractères de longueur variable max. 4 G

-- Types énumérés
OM ENUM(a, b, c, ...)                       valeur réservée exclusivement aux choix indiqués
```

GPA775

28

# Langage SQL

## DDL | TABLE | Types de données

```
-- Types liés au temps
M  TIME                                     heure de format : HH:MM:SS

M  DATE                                     date de format : YYYY-MM-JJ
                                         de 1000-01-01 à 9999-12-31

M  DATETIME                               date et heure de format : YYYY-MM-JJ HH:MM:SS
                                         de 1000-01-01 00:00:00 à 9999-12-31 23:59:59

M  TIMESTAMP                             nombre de seconde depuis: 1970-01-01 00:00:01

O  DATE                                     date et heure de format : YYYY-MM-JJ HH:MM:SS
                                         de 4712BC-01-01 00:00:00 à 9999-12-31 23:59:59
```

GPA775

29

## Langage SQL

### DDL | TABLE | Contraintes

- ▶ Il est possible d'ajouter 6 types de contraintes différentes :
  - ▶ valeur nulle permise ou interdite (NULL par défaut)      NULL ou NOT NULL
  - ▶ unicité      UNIQUE
  - ▶ clé primaire      PRIMARY KEY
  - ▶ clé étrangère      FOREIGN KEY
  - ▶ validation      CHECK
  - ▶ valeur par défaut      DEFAULT
- ▶ Toutes les contraintes qui seront spécifiées seront automatiquement gérées par le SGBD et toutes les manipulations violant ces contraintes seront interdites.

GPA775

30

## Langage SQL

### DDL | TABLE | Contraintes

- ▶ On peut ajouter les contraintes de deux façons :
  - ▶ à la fin de la déclaration d'une colonne  
avantage, écriture plus compacte
  - ▶ à la fin de la déclaration de toutes les colonnes  
avantage, on peut nommer une contrainte et y faire référence plus tard.
  - ▶ on privilégie généralement la deuxième approche lorsqu'elle est possible.

GPA775

31

## Langage SQL

### DDL | TABLE | Contraintes

- ▶ Contrainte de valeur nulle permise ou interdite
  - ▶ cette contrainte doit être mise sur la déclaration de la colonne;
  - ▶ si rien n'est indiqué, la valeur par défaut NULL est appliquée;
  - ▶ l'exemple suivant indique qu'un employé doit avoir un nom mais que son prénom et que sa date de naissance peuvent être absents (valeur nulle).

```
-- insertion à la fin d'une colonne
OM CREATE TABLE Employe (
  Id                INT,
  Nom               VARCHAR(32) NOT NULL,
  Prenom            VARCHAR(32) NULL,
  DateNaissance     DATE
);
```

GPA775

32

## Langage SQL

### DDL | TABLE | Contraintes

- ▶ Contrainte d'unicité
  - ▶ une déclaration à la fin permet de mettre une contrainte sur un ensemble de colonnes;
  - ▶ dans l'exemple suivant, la colonne **Id** doit être unique ainsi que le duo **Nom** et **Prenom**;
  - ▶ pris individuellement, **Nom** et **Prenom** peuvent être répétés mais pas pour la même ligne.

```
-- insertion à la fin d'une colonne et à la fin de toutes les colonnes
-- le nom de la contrainte (uc_NomPrenom) est optionnel
OM CREATE TABLE Employe (
  Id                INT                UNIQUE,
  Nom               VARCHAR(32)       NOT NULL,
  Prenom            VARCHAR(32),
  CONSTRAINT uc_NomPrenom UNIQUE (Nom, Prenom)
);
```



GPA775

33

# Langage SQL

## DDL | TABLE | Contraintes

### ► Contrainte de clé primaire

- une insertion à la fin permet aussi de préciser une clé primaire sur plusieurs attributs;
- implicitement, une clé primaire inclue les contraintes UNIQUE et NOT NULL.

```
-- insertion à la fin d'une colonne
OM CREATE TABLE Employe (
    Id      INT          PRIMARY KEY,
    Nom     VARCHAR(32)
);

-- insertion à la fin de toutes les colonnes, le nom pk_Id est optionnel
OM CREATE TABLE Employe (
    Id      INT,
    Nom     VARCHAR(32),
    CONSTRAINT pk_Id PRIMARY KEY (Id)
);
```

GPA775

34

# Langage SQL

## DDL | TABLE | Contraintes

### ► Contrainte de clé étrangère

```
-- insertion à la fin d'une colonne
OM CREATE TABLE Employe (
    Id      INT          PRIMARY KEY,
    Nom     VARCHAR(32),
    Depart  INT          REFERENCES Departement(Id)
);

-- insertion à la fin de toutes les colonnes, le nom pk_Id est optionnel
OM CREATE TABLE Employe (
    Id      INT,
    Nom     VARCHAR(32),
    Depart  INT,
    CONSTRAINT pk_Id PRIMARY KEY (Id),
    CONSTRAINT fk_IdDepart FOREIGN KEY (Depart) REFERENCES Departement(Id)
);
```

GPA775

35

# Langage SQL

## DDL | TABLE | Contraintes

- ▶ Contrainte de clé étrangère - plusieurs options supplémentaires sont possibles:
  - ▶ la valeur par défaut est RESTRICT.

```
-- option supplémentaire sur une clé étrangère
OM ... contrainte_clé_étrangère ...
    ON { UPDATE | DELETE } { RESTRICT | CASCADE | SET NULL }

-- un exemple simple
OM CREATE TABLE Employe (
    Id            INT,
    Nom           VARCHAR(32),
    Depart        INT,
    CONSTRAINT pk_Id      PRIMARY KEY (Id),
    CONSTRAINT fk_IdDepart FOREIGN KEY (Depart) REFERENCES Departement(Id)
    ON DELETE CASCADE
);
```

GPA775

36

# Langage SQL

## DDL | TABLE | Contraintes

- ▶ Contrainte de validation
  - ▶ il est possible d'insérer une expression validant si la valeur insérée respecte une condition.

```
-- insertion à la fin d'une colonne
OM CREATE TABLE Employe (
    Id            INT      PRIMARY KEY,
    Nom           VARCHAR(32),
    Age           INT      CHECK (Age >= 18)
);

-- insertion à la fin de toutes les colonnes
OM CREATE TABLE Employe (
    Id            INT      PRIMARY KEY,
    Nom           VARCHAR(32),
    Age           INT,
    CONSTRAINT cc_ValidAge CHECK (Age >= 18 AND Age < 65)
);
```

GPA775

37

# Langage SQL

## DDL | TABLE | Contraintes

- ▶ Contrainte de valeur par défaut
  - ▶ si aucune valeur n'est spécifiée pour une colonne lors de l'insertion, la valeur définie par défaut sera insérée;
  - ▶ cette contrainte s'ajoute à la fin de la déclaration de la colonne.

```
-- insertion à la fin d'une colonne
OM CREATE TABLE Employe (
  Id          INT          PRIMARY KEY,
  Nom         VARCHAR(32)  DEFAULT NULL,
  Adresse     VARCHAR(32)  DEFAULT 'aucune adresse',
  DateEmbauche DATE       DEFAULT GETDATE()
);
```

GPA775

38

# Langage SQL

## DDL | TABLE | ALTER

- ▶ Il est possible de modifier la structure d'une table avec la commande ALTER TABLE :

```
OM ALTER TABLE nom_table
| ADD COLUMN nom_col type_col [contrainte_col] [, ...]
| ADD CONSTRAINT [nom_contrainte] {PRIMARY KEY | UNIQUE} paramètres [, ...]
| CHANGE [COLUMN] nom_col nouveau_nom_col type_col [contrainte_col] [, ...]
| MODIFY [COLUMN] nom_col type_col [contrainte_col] [, ...]
| DROP [COLUMN] nom_col [, ...]
| DROP PRIMARY KEY
| DROP FOREIGN KEY nom_contrainte [, ...]
| RENAME [TO | AS] nouv_nom_table
... ;
```

GPA775

39

# Langage SQL

## DML

- ▶ Le DML possède 4 instructions principales :

<b>OM</b> SELECT ...	recherche
<b>OM</b> INSERT ...	insertion
<b>OM</b> DELETE ...	suppression
<b>OM</b> UPDATE ...	mise à jour

GPA775

40

# Langage SQL

## DML | SELECT

- ▶ L'instruction SELECT est l'instruction la plus complexe du langage SQL.
- ▶ Néanmoins, elle offre une grande flexibilité pour la recherche d'information dans les tables.
- ▶ Elle est constituée de 2 clauses obligatoires et de 4 clauses optionnelles :

<b>OM</b> SELECT colonne [, ...]	les colonnes à afficher
FROM table [, ...]	les tables impliquées dans la requête
[WHERE expression]	les lignes sélectionnées (condition sur les tuples)
[GROUP BY colonne [, ...]]	les colonnes utilisées pour l'agrégation
[HAVING expression]	les agrégats sélectionnés (condition sur les agrégats)
[ORDER BY colonne [{ASC   DESC}] [, ...]];	les colonnes utilisées pour le tri

GPA775

41

## Langage SQL

### DML | SELECT

- ▶ Précisions générales sur la clause **SELECT** :
  - ▶ cette clause permet de choisir quelle(s) colonne(s) est retournée(s);
  - ▶ on sépare les colonnes à retourner par des virgules;
  - ▶ il est possible de préfixer une colonne par : « *nom de la table.* » (permet de lever l'ambiguïté);
  - ▶ l'usage de **\*** indique que toutes les colonnes sont sélectionnées;
  - ▶ malgré sa ressemblance, l'instruction **SELECT** n'est pas lié l'opération de sélection de l'AR.

<p><b>OM</b> <b>SELECT</b> *</p> <p><b>FROM</b> Employe</p>	<p><i>selection de toutes les colonnes</i></p>
<p><b>OM</b> <b>SELECT</b> Nom, Prenom</p> <p><b>FROM</b> Employe</p>	<p><i>selection de deux colonne</i></p>

GPA775

42

## Langage SQL

### DML | SELECT

- ▶ Précisions sur la clause **FROM** :
  - ▶ cette clause permet de choisir quelle(s) table(s) est utilisée(s) pour la requête;
  - ▶ on sépare les tables à utiliser dans la requête par des virgules.

<p><b>OM</b> <b>SELECT</b> Employe.Id, Departement.Id</p> <p><b>FROM</b> Employe, Departement</p>	<p><i>usage de préfixes</i></p>
<p><b>OM</b> <b>SELECT</b> *</p> <p><b>FROM</b> Employe, Departement</p>	<p><i>intéressant =&gt; produit cartésien</i></p>

GPA775

43

## Langage SQL

### DML | SELECT

- ▶ Précisions générales sur la clause **WHERE** :
  - ▶ la clause WHERE permet d'ajouter une restriction sur les lignes retournées;
  - ▶ elle prend la forme d'une expression conditionnelle;
  - ▶ lorsque la clause WHERE est omise, toutes les lignes sont affichés (équivalent à WHERE TRUE).

```

OM SELECT Nom
FROM Employe
WHERE Salaire > 50000;

OM SELECT *
FROM Employe, Departement
WHERE Employe.Depart = Departement.Id

```

*intéressant => équi-jointure*

GPA775

44

## Langage SQL

### DML | SELECT

- ▶ Clause **SELECT TOP** :
  - ▶ cette clause de la norme SQL permet de spécifier le nombre de ligne(s) retournée(s);
  - ▶ son usage est toutefois différent pour les deux SGBD utilisés.

```

O SELECT ...
FROM ...
WHERE ROWNUM < nombre_rang [...];

```

*Oracle utilise la pseudo colonne ROWNUM*

```

M SELECT ...
FROM ...
LIMIT {[décalage,] nombre_rang | nombre_rang OFFSET décalage}];

```

GPA775

45

## Langage SQL

### DML | SELECT

#### ► Clause **SELECT DISTINCT** :

- SQL n'élimine pas les doublons des lignes retournées;
- la clause **DISTINCT** permet de retirer les doublons (un doublon est une ligne entièrement identique);
- on verra plus loin que cette clause peut être utilisée avec les fonctions d'agrégation.

```
OM SELECT Ville
FROM Employe;                si on desire connaître dans quelles villes habitent les employés
                              ici, les mêmes villes peuvent être présentes sur plusieurs lignes
```

```
OM SELECT DISTINCT Ville
FROM Employe;                une seule instance des villes est retournée
```

GPA775

46

## Langage SQL

### DML | SELECT

#### ► Concept d'alias :

- il est possible de créer un alias pour une colonne et/ou une table;
- permet d'alléger la syntaxe des requêtes (l'usage du mot clé **AS** est optionnel mais recommandé);
- les alias de colonnes ont l'avantage de renommer la colonne pour la sortie, souvent pertinent pour apporter des précisions (l'usage d'apostrophes permet des mots avec espaces);
- les alias de table sont parfois essentielle lors de requêtes récursives afin de lever l'ambiguïté.

```
OM SELECT nom_colonne [[AS] alias_colonne] [, ...]
FROM nom_table [[AS] alias_table] [, ...];
```

```
OM SELECT EMP.Nom AS Employé, SUP.Nom AS Superviseur
FROM Employe AS EMP, Employe AS SUP
WHERE EMP.Superviseur = SUP.NAS;                requête récursive
```

GPA775

47

## Langage SQL

### DML | SELECT

► Opérateurs de la clause **WHERE** – opérateurs conditionnels :

► égalité :	=		
► différent:	<> ou !=,		
► plus grand et plus grand ou égal :	> et >=		
► plus petit et plus petit ou égal :	< et <=		
► égalité totale sur une valeur nulle :	<=>		
SELECT 1 = 1	---> 1	SELECT 1 <=> 1	---> 1
SELECT NULL = NULL	---> NULL	SELECT NULL <=> NULL	---> 1
SELECT NULL = 1	---> NULL	SELECT NULL <=> 1	---> NULL

GPA775

48

## Langage SQL

### DML | SELECT

► Opérateurs de la clause **WHERE** – opérateurs logiques :

► et logique :	<b>AND</b> ou <b>&amp;&amp;</b>
► ou logique :	<b>OR</b> ou <b>  </b>
► négation logique :	<b>NOT</b> ou <b>!</b>
► ou exclusif logique :	<b>XOR</b>
► attention, la valeur nulle indique une indétermination et non une valeur fausse, ainsi :	
SELECT 1    NULL	---> 1
SELECT 0    NULL	---> NULL
SELECT 1 && NULL	---> NULL
SELECT 0 && NULL	---> 0



GPA775

49

## Langage SQL

### DML | SELECT

- ▶ Opérateurs de la clause **WHERE** – opérateur de recherche textuelle :
  - ▶ recherche d'un patron dans une chaîne de caractère : **LIKE**
- ▶ Les éléments génériques de chaîne de caractères sont :
  - ▶ chaîne de caractères : `'xyz'` **LIKE** `'abc'`
  - ▶ substitue pour  $n$  caractère(s) ( $n \geq 0$ ) : `%` **LIKE** `'%abc%'`
  - ▶ substitue pour 1 caractère : `_` (soulignement) **LIKE** `'_abc_'`
  - ▶ substitue pour un choix de caractères présents : `[...]`
    - `[abc]` (un choix parmi a, b et c) **LIKE** `'[aA]%'`
    - `[a-e]` (un choix parmi a, b, c, d et e) **LIKE** `'[s-u]%'`
  - ▶ substitue pour un choix de caractères absents : `[^...]` ou `[!...]`
    - `[!abc]` (aucun parmi a, b et c) **LIKE** `'[^aA]%'`
    - `[^a-e]` (aucun parmi a, b, c, d et e) **LIKE** `'[!s-u]%'`

GPA775

50

## Langage SQL

### DML | SELECT

- ▶ Opérateurs de la clause **WHERE** :
  - ▶ l'opérateur **IN** permet de spécifier une liste de choix possible;
  - ▶ très utilisé pour les requêtes imbriquées.

```

OM SELECT nom_colonne [, ...]
   FROM nom_table [, ...]
   WHERE colonne IN (valeur1, valeur2, ...);

OM SELECT Nom, Prenom
   FROM Employe
   WHERE Ville IN ('Montréal', 'Québec');
```

GPA775

51

## Langage SQL

### DML | SELECT

- ▶ Opérateurs de la clause **WHERE** :
  - ▶ l'opérateur **BETWEEN** permet de spécifier un intervalle de valeurs possibles;
  - ▶ s'applique autant pour les types numériques qu'alpha numériques.

```

OM SELECT nom_colonne [, ...]
   FROM nom_table [, ...]
   WHERE colonne BETWEEN valeurMin AND valeurMax;

OM SELECT Nom, Prenom
   FROM Employe
   WHERE Age BETWEEN 35 AND 45;

```

GPA775

52

## Langage SQL

### DML | SELECT

- ▶ Opérateurs de la clause **WHERE** – il existe plusieurs autres éléments syntaxiques dont voici les plus significatifs :
 

▶ test un résultat logique (vrai/faux) :	<b>IS</b>
▶ inversion logique d'un résultat booléen :	<b>IS NOT</b>
▶ si une valeur est nulle :	<b>IS NULL</b>
▶ si une valeur n'est pas nulle :	<b>IS NOT NULL</b>
▶ autres variantes :	<b>NOT LIKE, NOT IN(...), ...</b>

```

OM SELECT Nom, Prenom
   FROM Employe
   WHERE Courriel IS NULL AND Ville NOT IN ('Montréal', 'Québec');

```

GPA775

53

## Langage SQL

### DML | SELECT

- ▶ Usage de requêtes imbriquées :
  - ▶ il est possible d'utiliser le résultat d'une requête comme entrée dans une autre;
  - ▶ ainsi, on peut imbriquer plusieurs requêtes les unes dans les autres;
  - ▶ ce mécanisme offre des solutions élégantes et puissantes à plusieurs situations;
  - ▶ comme nous le verrons plus loin, la notion d'imbrication fait partie du langage SQL et s'applique sur tous les éléments du langage (n'est pas réservé à la clause SELECT).

```
OM SELECT Nom, Prenom
FROM (SELECT * FROM Employe WHERE Sexe = 'f') AS FemmeEmploye
WHERE Nom LIKE 'Mont%';
```

*pour cet exemple, l'utilisation de l'alias est obligatoire puisque la table intermédiaire doit être nommée afin d'être utilisée dans la clause FROM*

GPA775

54

## Langage SQL

### DML | SELECT

- ▶ Usage de requêtes imbriquées :
  - ▶ puisque les résultats sont toujours des tables, on peut utiliser ces tables comme données d'entrée;
  - ▶ par contre, si la requête utilisée retourne une table n'ayant qu'une seule colonne et une seule valeur, il est alors possible d'utiliser ce résultat comme un scalaire;
  - ▶ si la requête utilisée retourne une table ayant une colonne de plusieurs valeurs, il est possible d'utiliser ce résultat comme une liste.

```
OM SELECT Nom, Prenom
FROM Employe
WHERE IdDepart = (SELECT Id FROM Departement WHERE Nom = 'Ventes') AND
Ville IN (SELECT NomVille FROM Geographie WHERE Province = 'Ontario');
```

*1<sup>er</sup> SELECT imbriqué est utilisé comme étant un scalaire  
2<sup>e</sup> SELECT imbriqué est utilisé comme étant une liste*

GPA775

55

## Langage SQL

### DML | SELECT

- ▶ SQL propose plusieurs fonctions d'agrégation agissant comme des fonctions de calcul statistique sur les données :
  - ▶ valeur minimum : **MIN**
  - ▶ valeur maximum : **MAX**
  - ▶ nombre de valeurs : **COUNT**
  - ▶ somme des valeurs : **SUM**
  - ▶ moyenne des valeurs : **AVG**
- ▶ Selon les SGBD, plusieurs autres fonctions d'agrégation sont disponibles : valeur médiane, écart type, variance et plusieurs autres.

GPA775

56

## Langage SQL

### DML | SELECT

- ▶ Les fonctions d'agrégation ne sont possibles que dans la clause SELECT et HAVING.
- ▶ Sans la clause GROUP BY, l'usage des fonctions d'agrégation se fait en considérant toutes les lignes de la table.

```

OM SELECT MIN(Salaire), MAX(Salaire), SUM(Salaire), AVG(Salaire)
FROM Employe;

OM SELECT COUNT(*)
FROM Employe;

OM SELECT COUNT(DISTINCT Salaire)
FROM Employe;

```

GPA775

57

## Langage SQL

### DML | SELECT

#### ► Clause **GROUP BY** :

- cette clause permet d'appliquer les fonctions d'agrégation aux sous groupes de ligne qui ont une valeur commune pour une colonne donnée;
- il est très important de savoir que les fonctions de regroupement et d'agrégation ne s'appliquent qu'après les jointures de la requête (s'il y a lieu);
- l'usage de plusieurs colonnes pour cette clause implique que le regroupement se fera sur l'ensemble des valeurs provenant de ces colonnes (comme un tout).

```
OM SELECT IdDepart, COUNT(*), AVG(Salaire)
FROM Employe
GROUP BY IdDepart;
```

*cette requête retourne la liste des n départements avec chacun le Id, le nombre d'employé et le salaire moyen du département*

GPA775

58

## Langage SQL

### DML | SELECT

#### ► Clause **HAVING** :

- applique une restriction sur les regroupements ne respectant pas l'expression donnée;
- ne s'utilise qu'avec la clause GROUP BY;

```
OM SELECT dep.Nom, COUNT(*)
FROM Employe AS emp, Departement AS dep
WHERE emp.IdDepart = dep.Id
GROUP BY dep.Id
HAVING COUNT(*) > 2;
```

*cette requête retourne le nom des départements ainsi que le nombre d'employés mais uniquement pour les départements ayant plus de 2 employés*

GPA775

59

## Langage SQL

### DML | SELECT

- ▶ Clause **HAVING** :
  - ▶ l'usage simultané des clauses **WHERE** et **HAVING** requiert une certaine attention;
  - ▶ la clause **WHERE** limite les lignes alors que la clause **HAVING** limite les regroupements;
  - ▶ pour résoudre l'ambiguïté, il faut savoir que la clause **WHERE** est appliquée avant la clause **HAVING**.
  - ▶ cette situation requiert une grande attention car il est facile d'écrire une requête erronée.

GPA775

60

## Langage SQL

### DML | SELECT

- ▶ Mise en situation :
  - ▶ on désire compter le nombre d'employés ayant un salaire supérieur ou égal à 65 000\$ pour chaque département;
  - ▶ par contre, nous ne sommes intéressé que par les département ayant plus de 5 employés.
- ▶ On comprend que :
  - ▶ il y aura un regroupement sur la colonne département;
  - ▶ il y aura une restriction sur les employés (selon le salaire);
  - ▶ il y aura une restriction sur les regroupements (selon le nombre d'employés par département).

GPA775

61

## Langage SQL

### DML | SELECT

- ▶ On pourrait être tenté d'écrire cette requête erronée :

```
OM SELECT dep.Nom, COUNT(*)
FROM Employe AS emp, Departement AS dep
WHERE emp.IdDepart = dep.Id AND Salaire > 65000
GROUP BY dep.Id
HAVING COUNT(*) > 5;
```

- ▶ Cette requête ne retourne que les départements qui ont plus de 5 employés gagnant plus de 65 000\$.

GPA775

62

## Langage SQL

### DML | SELECT

- ▶ Puisqu'on désire d'abord un nombre minimum d'employés par département (peut importe leur salaire), il faudra faire une requête imbriquée (HAVING) pour ensuite éliminer les lignes non pertinentes (WHERE).
- ▶ Voici une solution correcte même si elle est moins intuitive.

```
OM SELECT dep.Nom, COUNT(*)
FROM Employe AS emp, Departement AS dep
WHERE emp.IdDepart = dep.Id AND Salaire > 65000 AND
      emp.IdDepart IN ( SELECT IdDepart
                        FROM   Employe
                        GROUP BY IdDepart
                        HAVING  COUNT(*) > 5)
GROUP BY dep.Id
```

GPA775

63

## Langage SQL

### DML | SELECT

► Clause **ORDER BY** :

- cette clause permet d'ordonner les lignes selon un critère spécifique;
- on peut utiliser un tri ascendant ou descendant (ascendant par défaut);
- on peut utiliser une ou plusieurs colonnes.

```
OM SELECT nom_colonne [, ...]
FROM nom_table [, ...]
ORDER BY colonne [ASC | DESC] [, ...];

OM SELECT Nom, Prenom
FROM Employe
ORDER BY IdDepart ASC, Nom ASC, Prenom DESC;
```

GPA775

64

## Langage SQL

### DML | INSERT

- L'instruction INSERT permet l'insertion de ligne (de données) dans une table.

```
OM INSERT INTO nom_table[nom_colonne1 [, nom_colonne2...]]
VALUES ( {valeur_colonne1 | DEFAULT} [, valeur_colonne2 | DEFAULT ...] );

OM INSERT INTO Employe(Nom, Prenom, NAS, Telephone)
VALUES ('Lapierre', 'Pierre', 987654321, '(514) 456-7890');

OM INSERT INTO Employe
VALUES ('Lapierre', 'Pierre', 987654321, '(514) 456-7890', 'plp@Hotmail.com');
cet exemple suppose qu'il y a 5 colonnes dans la table : + le courriel
```



GPA775

65

## Langage SQL

### DML | INSERT

- ▶ Même s'il est possible d'omettre le nom des colonnes, il est recommandé de les spécifier afin de simplifier les cas éventuels où la structure de la table serait modifiée.
- ▶ Il est fréquent d'utiliser une requête imbriquée afin d'aller chercher la valeur d'une clé primaire pour l'utiliser à titre de clé étrangère.

```
OM INSERT INTO Employe(Nom, Prenom, NAS, Telephone, IdDepartement)
VALUES ( 'Lapierre',
        'Pierre',
        987654321,
        '(514) 456-7890',
        (SELECT Id FROM Departement WHERE Nom = 'Ventes'));
```

GPA775

66

## Langage SQL

### DML | INSERT

- ▶ On peut insérer plusieurs lignes grâce à une requête imbriquée. Il faut cependant que les schémas soient les mêmes.

```
OM INSERT INTO nom_table[nom_colonne1 [, nom_colonne2...]]
SELECT expression;

OM INSERT INTO DepartementInfo(NomDepartement, NombreEmploye, SalaireTotal)
SELECT d.Nom, COUNT(d.Id), SUM(e.Salaire)
FROM Enseignant AS e, Departement AS d
WHERE Depart = Id
GROUP BY d.Id;
```

GPA775

67

## Langage SQL

### DML | DELETE

- ▶ L'instruction DELETE permet de supprimer des lignes dans une table (une table à la fois).
- ▶ Cette instruction ne permet pas de supprimer une table mais son contenu. Même si le contenu de la table est vidé, elle existe toujours avec le même schéma.
- ▶ Si la clause WHERE n'est pas spécifiée, la table est entièrement vidée de son contenu.
- ▶ Attention aux clés étrangères.

```

OM DELETE FROM nom_table [WHERE expression];

OM DELETE FROM Employe WHERE Ville = 'Laval';
OM DELETE FROM Employe WHERE IdDep IN (SELECT Id FROM Departement WHERE Nom = 'R&D');
OM DELETE FROM Employe;

```

GPA775

68

## Langage SQL

### DML | UPDATE

- ▶ L'instruction UPDATE permet de modifier les valeurs d'attributs d'une ou plusieurs lignes.
- ▶ Si la clause WHERE n'est pas spécifiée, toutes les lignes de la table sont modifiées.

```

OM UPDATE nom_table
SET nom_colonne1 = valeur1 [, nom_colonne2 = valeur2 ...]
[WHERE expression];

OM UPDATE Employe
SET Ville = 'Montréal', Province = 'Québec';
      change la ville et la province aux valeurs de Montréal et Québec pour tous les employés

```

GPA775

69

## Langage SQL

### DML | UPDATE

- ▶ L'instruction UPDATE permet de modifier les valeurs d'attributs d'une ou plusieurs lignes.
- ▶ Si la clause WHERE n'est pas spécifiée, toutes les lignes de la table sont modifiées.

```

OM UPDATE Employe
SET IdDepartement = (SELECT Id FROM Departement WHERE Nom = 'Ventes')
WHERE IdDepartement = (SELECT Id FROM Departement WHERE Nom = 'Achats');
applique le département des Ventes à tous les employés de département des Achats

OM UPDATE Employe
SET Salaire = Salaire * 1,15
WHERE IdDepartement IN (SELECT Id FROM Departement WHERE Nom IN ('Ventes', 'R&D'));
donne une augmentation de 15% aux employés des départements des Ventes et de la R&D

```

GPA775

70

## Langage SQL

### DML | Requêtes corrélées

- ▶ Le SQL permet des requêtes plus complexes qu'on appelle des requêtes imbriquées corrélées (ou sous-requêtes corrélées ou requêtes synchronisées).
- ▶ En contre partie, les requêtes imbriquées que nous avons vues sont nommées requêtes imbriquées non corrélées.
- ▶ Ces requêtes ont la forme suivante :
  - ▶ une requête est imbriquée à l'intérieure d'une autre;
  - ▶ contrairement aux requêtes imbriquées non corrélées, la requête interne utilise des valeurs de la requête externe;
  - ▶ la résolution de telles requêtes implique que la requête interne est évaluée pour chaque ligne de la requête externe.

GPA775

71

## Langage SQL

### DML | Requêtes corrélées

```

OM SELECT Nom
FROM Employe AS chaqueEmploye
WHERE Salaire > ( SELECT AVG(Salaire)
                  FROM Employe AS sousGroupeEmploye
                  WHERE sousGroupeEmploye.IdDepartement = chaqueEmploye.IdDepartement );
recherche les employés qui gagnent plus que la moyenne de leur propre département

OM SELECT NAS, Nom, Prenom
FROM Etudiant AS etu
WHERE ( SELECT COUNT(*)
        FROM EtudiantCours AS ec
        WHERE etu.NAS = ec.NASEtudiant ) > 4;
recherche les étudiants qui suivent plus de 4 cours

```

GPA775

72

## Langage SQL

### DML | Jointure

- ▶ Nous avons vu qu'il existait plusieurs types de jointure pour l'AR. Or, le langage SQL possède différents opérateurs permettant de réaliser ces opérations.
- ▶ Voici les 4 opérateurs SQL importants et leur équivalent de l'AR :
 

▶ INNER JOIN	jointure naturelle	⋈
▶ FULL JOIN	jointure externe	⋈
▶ LEFT JOIN	jointure externe de gauche	⋈
▶ RIGHT JOIN	jointure externe de droite	⋈

```

OM SELECT nom_colonne1 [, nom_colonne2, ...]
FROM nom_table1
{ INNER | { FULL | LEFT | RIGHT } [OUTER] } JOIN nom_table2
ON condition_sur_colonnes;

```

GPA775

73

## Langage SQL DML | Jointure

- ▶ Rappel sur l'interprétation de ces jointures :
  - ▶ les jointures sont basées sur un produit cartésien et à une sélection liée à l'égalité de valeur commune d'au moins 2 colonnes provenant de chacune des tables;
  - ▶ les jointures éliminent tous les doublons;
  - ▶ INNER JOIN *retourne les lignes respectant l'égalité sur les deux tables*
  - ▶ FULL JOIN *retourne les lignes respectant l'égalité sur les deux tables + ajoute les lignes n'ayant pas d'équivalent en remplissant les valeurs manquantes des tables opposées à nulle*
  - ▶ LEFT JOIN *retourne les lignes respectant l'égalité sur les deux tables + ajoute les lignes de la 1<sup>re</sup> table n'ayant pas d'équivalent en remplissant les valeurs manquantes de la 2<sup>e</sup> table à nulle*
  - ▶ RIGHT JOIN *retourne les lignes respectant l'égalité sur les deux tables + ajoute les lignes de la 2<sup>e</sup> table n'ayant pas d'équivalent en remplissant les valeurs manquantes de la 1<sup>re</sup> table à nulle*

GPA775

74

## Langage SQL DML | Jointure

- ▶ MySQL ne supporte pas l'opérateur FULL JOIN.
- ▶ Nous verrons plus loin comment obtenir le même résultat.

```

O SELECT nom_colonne1 [, nom_colonne2, ...]
  FROM nom_table1
  { INNER | { FULL | LEFT | RIGHT } [OUTER] } JOIN nom_table2
  ON condition_sur_colonnes;

M SELECT nom_colonne1 [, nom_colonne2, ...]
  FROM nom_table1
  { INNER | { LEFT | RIGHT } [OUTER] } JOIN nom_table2
  ON condition_sur_colonnes;

```

GPA775

75

## Langage SQL DML | Jointure

- ▶ MySQL ne supporte pas l'opérateur FULL JOIN.
- ▶ Nous verrons plus loin comment obtenir le même résultat.

```

O SELECT nom_colonne1 [, nom_colonne2, ...]
  FROM nom_table1
 { INNER | { FULL | LEFT | RIGHT } [OUTER] } JOIN nom_table2
 ON condition_sur_colonnes;

M SELECT nom_colonne1 [, nom_colonne2, ...]
  FROM nom_table1
 { INNER | { LEFT | RIGHT } [OUTER] } JOIN nom_table2
 ON condition_sur_colonnes;

```

GPA775

76

## Langage SQL DML | Jointure

- ▶ Supposons ces deux tables : Departement et Enseignant

```

(1) SELECT * FROM Departement;
(2) SELECT * FROM Enseignant;

```

*contenu de la table Departement*  
*contenu de la table Enseignant*

1

Id	Nom	Sigle
1	Génie électrique	ELE
2	Génie mécanique	MEC
3	Génie logiciel	LOG

2

NAS	Nom	Prenom	IdDepartement
111222333	Tremblay	Mohamed	5
123456789	Miron	Gaston	2
555555555	Perrier	Caroline	1

GPA775

77

## Langage SQL DML | Jointure

(3) **SELECT** \* **FROM** Enseignant, Departement;

*produit cartésien entre les deux tables*

3

NAS	Nom	Prenom	IdDepartement	Id	Nom	Sigle
111222333	Tremblay	Mohamed	5	1	Génie électrique	ELE
123456789	Miron	Gaston	2	1	Génie électrique	ELE
555555555	Perrier	Caroline	1	1	Génie électrique	ELE
111222333	Tremblay	Mohamed	5	2	Génie mécanique	MEC
123456789	Miron	Gaston	2	2	Génie mécanique	MEC
555555555	Perrier	Caroline	1	2	Génie mécanique	MEC
111222333	Tremblay	Mohamed	5	3	Génie logiciel	LOG
123456789	Miron	Gaston	2	3	Génie logiciel	LOG
555555555	Perrier	Caroline	1	3	Génie logiciel	LOG

GPA775

78

## Langage SQL DML | Jointure

(4) **SELECT** \* **FROM** Departement, Enseignant;

*produit cartésien entre les deux tables*

4

Id	Nom	Sigle	NAS	Nom	Prenom	IdDepartement
1	Génie électrique	ELE	111222333	Tremblay	Mohamed	5
2	Génie mécanique	MEC	111222333	Tremblay	Mohamed	5
3	Génie logiciel	LOG	111222333	Tremblay	Mohamed	5
1	Génie électrique	ELE	123456789	Miron	Gaston	2
2	Génie mécanique	MEC	123456789	Miron	Gaston	2
3	Génie logiciel	LOG	123456789	Miron	Gaston	2
1	Génie électrique	ELE	555555555	Perrier	Caroline	1
2	Génie mécanique	MEC	555555555	Perrier	Caroline	1
3	Génie logiciel	LOG	555555555	Perrier	Caroline	1

GPA775

79

## Langage SQL

### DML | Jointure

- Les deux requêtes suivantes sont équivalentes en terme de résultat. Néanmoins, la requête 5a correspond davantage au modèle relationnel alors que la requête 5b respecte la norme ANSI. On privilégiera l'usage du INNER JOIN pour plusieurs raisons (performance interne, flexibilité, ...).

(5a) **SELECT** \* **FROM** Enseignant, Departement **WHERE** IdDepartement = Id;  
 (5b) **SELECT** \* **FROM** Enseignant **INNER JOIN** Departement **ON** IdDepartement = Id;

5

NAS	Nom	Prenom	IdDepartement	Id	Nom	Sigle
555555555	Perrier	Caroline	1	1	Génie électrique	ELE
123456789	Miron	Gaston	2	2	Génie mécanique	MEC

GPA775

80

## Langage SQL

### DML | Jointure

- Attention, ce type de jointure n'est pas supporté par MySQL.

(6) **SELECT** \* **FROM** Enseignant **FULL JOIN** Departement **ON** IdDepartement = Id;

6

NAS	Nom	Prenom	IdDepartement	Id	Nom	Sigle
555555555	Perrier	Caroline	1	1	Génie électrique	ELE
123456789	Miron	Gaston	2	2	Génie mécanique	MEC
111222333	Tremblay	Mohamed	5	NULL	NULL	NULL
NULL	NULL	NULL	NULL	3	Génie logiciel	LOG



GPA775

81

## Langage SQL

### DML | Jointure

- (7) **SELECT** \* **FROM** Enseignant **LEFT JOIN** Departement **ON** IdDepartement = Id;  
 (8) **SELECT** \* **FROM** Enseignant **RIGHT JOIN** Departement **ON** IdDepartement = Id;

7

NAS	Nom	Prenom	IdDepartement	Id	Nom	Sigle
555555555	Perrier	Caroline	1	1	Génie électrique	ELE
123456789	Miron	Gaston	2	2	Génie mécanique	MEC
111222333	Tremblay	Mohamed	5	NULL	NULL	NULL

8

NAS	Nom	Prenom	IdDepartement	Id	Nom	Sigle
123456789	Miron	Gaston	2	2	Génie mécanique	MEC
555555555	Perrier	Caroline	1	1	Génie électrique	ELE
NULL	NULL	NULL	NULL	3	Génie logiciel	LOG

GPA775

82

## Langage SQL

### DML | Fonctions

- ▶ La fonction **EXISTS** peut être pratique pour certaines requêtes imbriquées :
  - ▶ valide s'il y a au moins une ligne dans la sous-requête : **EXISTS**
  - ▶ valide s'il n'y a aucune ligne dans la sous-requête : **NOT EXISTS**

```
OM SELECT Nom, Prenom
FROM Employe AS e
WHERE EXISTS ( SELECT Id
               FROM Departement AS d
               WHERE d.SuperviseurNAS = e.NAS );
```

*sélectionne tous les employés qui sont superviseur d'un département*

GPA775

83

## Langage SQL

### DML | Fonctions

- ▶ La fonction **CONTAINS** permet de comparer deux ensembles de valeur et de retourner vrai si le premier contient tous les éléments du deuxième.
- ▶ attention, les schémas doivent être les mêmes.

```

OM SELECT Nom, Prenom
FROM Employe AS emp
WHERE ( ( SELECT empsub.NAS
          FROM Employe AS empsub
          WHERE empsub.NAS = emp.NAS )
        CONTAINS
        ( SELECT empdep.NAS
          FROM Employe AS empdep
          WHERE empdep.IdDepartement = emp.IdDepartement ) );

```

*sélectionne tous les employés qui supervisent tous les employés de leur propre département*

GPA775

84

## Langage SQL

### DML | Opérateurs ensemblistes

- ▶ Opérateurs ensembliste sur les tables :
  - ▶ mise en commun du contenu de deux tables : **UNION**
  - ▶ sélection des lignes communes de deux tables : **INTERSECT**
  - ▶ extraction des lignes d'une première table se trouvant dans une deuxième table : **MINUS**
  - ▶ les opérateurs UNION, INTERSECT et MINUS permettent de manipuler deux tables pour en produire une seule;
  - ▶ ces opérations ne s'appliquent pas sur un SELECT mais plutôt à l'extérieur pour mettre en commun deux résultats intermédiaire;
  - ▶ attention, les schémas doivent être identiques.

GPA775

85

## Langage SQL

### DML | Opérateurs ensemblistes

- Les opérateurs INTERSECT et MINUS ne sont pas disponibles avec MySQL. Les pages suivantes présentent comment les simuler.

```
OM déclaration1 UNION déclaration2
O  déclaration1 INTERSECT déclaration2
O  déclaration1 MINUS  déclaration2

M  -- a INTERSECT b
   SELECT a.col1 [, a.col2, ...]
   FROM a INNER JOIN b
   ON a.col1 = b.col1 [AND a.col2 = b.col2, ...];

M  -- a MINUS b
   SELECT DISTINCT a.col1 [, a.col2, ...]
   FROM a LEFT JOIN b
   ON a.col1 = b.col1 [AND a.col2 = b.col2, ...]
   WHERE b.col1 IS NULL;
```

GPA775

86

## Langage SQL

### DML | Opérateurs ensemblistes

```
SELECT Nom, Prenom
FROM ( SELECT NAS, Nom, Prenom FROM Etudiant
      UNION
      SELECT NAS, Nom, Prenom FROM Enseignant ) AS res;
                                     le nom de tous les étudiants et enseignants

SELECT Etudiant.Nom
FROM Etudiant INNER JOIN Enseignant
ON Etudiant.Nom = Enseignant.Nom;
                                     les noms qu'ont en commun les étudiants et les enseignants

SELECT DINSTINCT Etudiant.Nom
FROM Etudiant LEFT JOIN Enseignant
ON Etudiant.Nom = Enseignant.Nom
WHERE Enseignant.Nom IS NULL;
                                     les noms que possèdent les étudiants et qu'aucun enseignant ne possède
```

GPA775

87

## Langage SQL

### DML | Jointure externe et MySQL

- Nous avons vu que MySQL ne permet pas de jointure externe. Il est très simple de reproduire ce résultat ainsi :

```
...a FULL JOIN b... = ...a LEFT JOIN b...
                     UNION
                     ...a RIGHT JOIN b...
```

GPA775

88

## Langage SQL

### DDL/DML | Les colonnes à valeur automatique

- Il est très fréquent d'avoir des tables dont une colonne sert d'identifiant unique et où la valeur n'a aucune importance (id).
- Le SGBD peut générer automatiquement ces valeurs.
- MySQL utilise un mécanisme très simple qui consiste à ajouter la contrainte `AUTO_INCREMENT` à la colonne lors du `CREATE TABLE`. Ensuite, lors de chaque insertion, MySQL s'occupe d'attribuer une valeur unique.

```
M CREATE TABLE Departement (
    Id      INT      PRIMARY KEY      AUTO_INCREMENT,
    Nom     VARCHAR(32)
);

INSERT INTO Departement (Nom) VALUES ('Ventes');
```

GPA775

89

## Langage SQL

### DDL/DML | Les colonnes à valeur automatique

- ▶ Oracle utilise un concept légèrement différent qui est un peu plus complexe mais plus flexible : l'utilisation d'objet SEQUENCE.
- ▶ L'exemple plus bas montre un exemple où l'objet SEQUENCE est utilisé manuellement pour chaque insertion. Il est possible d'automatiser cette tâche avec les TRIGGER.

```

o CREATE TABLE Departement (
    Id      INT      PRIMARY KEY,
    Nom     VARCHAR(32)
);
CREATE SEQUENCE seqDepId START WITH 1 INCREMENT BY 1;

INSERT INTO Departement (Id, Nom) VALUES (seqDepId.nextval, 'Ventes');
```

GPA775

90

## Langage SQL

### DML | Les fonctions de manipulation

- ▶ Il existe un très grand nombre de fonctions disponibles pour le langage SQL. Voici les plus importantes de MySQL.
- ▶ Fonctions mathématiques :
  - ▶ ABS, SIGN,
  - ▶ DIV, MOD,
  - ▶ POW, SQRT, EXP, LOG, LOG2, LOG10, LN,
  - ▶ PI, COS, SIN, TAN, ACOS, ASIN, ATAN, ATAN2, RADIANS, DEGREES
  - ▶ RAND,
  - ▶ ROUND, FLOOR, CEIL

GPA775

91

## Langage SQL

### DML | Les fonctions de manipulation

- ▶ Fonctions de manipulation de chaîne de caractères :
  - ▶ ASCII, CHAR, LENGTH, LOWER, UPPER
  - ▶ CONCAT, TRIM, LEFT, RIGHT,
  - ▶ MID, SUBSTR, REPLACE, LOCATE,
  - ▶ FORMAT,
- ▶ Fonctions de manipulation de la date et de l'heure :
  - ▶ NOW, SYSDATE, DATEDIFF, TIMEDIFF
  - ▶ YEAR, MONTH, WEEK, DAY, HOUR, MINUTE, SECOND, MICROSECOND
  - ▶ STR\_TO\_DATE, DATE\_FORMAT,

GPA775

92

## Langage SQL

### DML | Les fonctions de manipulation

```
M  INSERT INTO ResultatRecherche(DateMesure, Resultat1, Resultat2, Commentaires)
      VALUES ( STR_TO_DATE('28-11-2001', '%d-%m-%Y'),
                POW(5, 2),
                DEGREES(SIN(PI()/2)),
                UPPER('Calibration réussie!'));
```

GPA775

93

## Langage SQL

### DDL | VIEW

- ▶ Les vues sont des tables virtuelles dérivées à partir d'autres tables (appelées tables de base). Elles ne sont jamais stockées sur disque mais plutôt maintenue en mémoire vive.
- ▶ Une vue permet d'éviter l'exécution de requêtes coûteuses à répétition.
- ▶ Lorsqu'une vue est créée, son contenu est constamment maintenu à jour lors des opérations du DML sur les tables de bases concernées.
- ▶ Une vue peut servir de base à une autre vue.

```
OM CREATE VIEW nom_vue [( colonne1 [, colonne2 ...])]
AS declaration;

OM DROP VIEW nom_vue1 [, nom_vue2 ...];
```

GPA775

94

## Langage SQL

### DDL | VIEW

- ▶ Si les colonnes résultantes sont issues de calculs ou de fonctions d'agrégation, il faut nommer les colonnes de la vue.

```
OM CREATE VIEW EmployeDepartement
AS SELECT emp.Nom AS Nom, emp.Prenom AS Prenom, dep.Nom AS Departement
FROM Employe AS emp INNER JOIN Departement AS dep
ON emp.IdDepartement = dep.Id
WHERE DATEDIFF(NOW(), emp.DateNaissance)/365 > 35;
    une vue ayant le nom, le prénom et le département des employés de plus de 35 ans

OM CREATE VIEW DepartementInfo(Nom, NbrEmploye, SalaireTotal)
AS SELECT dep.Nom, COUNT(*), SUM(emp.Salaire)
FROM Employe AS emp, Departement AS dep
WHERE emp.IdDepartement = dep.Id
ORDER BY dep.Id;
    une vue sur les statistiques des départements (nom, nombre d'employés et salaire total)

OM DROP VIEW EmployeDepartement, DepartementInfo;
```

GPA775

95

## Langage SQL

### DML | VIEW

- ▶ Toutes les opérations du DML s'appliquent sur les vues (consultation, insertion, modification ou suppression).
- ▶ Attention car certaines opérations du DML sur une vue peuvent être très complexes et ambiguës. Si la vue est créée à partir d'une seule table, l'ambiguïté est levée. Si par contre la vue est créée à partir de plusieurs tables, plusieurs interprétations peuvent être faites.
- ▶ Selon leur définition, certaines vues ne peuvent être mise à jour car de telles requêtes ne font aucun sens (voir le dernier exemple).

...

GPA775

96

## Langage SQL

### DDL | INDEX

- ▶ Les indexes permettent une amélioration notable des performances des SGBD.
- ▶ Ils ne sont pas prescrits par la norme SQL mais font partie intégrante de toutes les implantations des SGBDR.
- ▶ Leur usage est très simple car le DDL permet de définir ou supprimer les indexes. Toutes les opérations du DML gèrent automatiquement l'utilisation adéquate des indexes sans intervention du concepteur.
- ▶ Néanmoins, une optimisation de la base de données nécessite une analyse permettant de faire une définition adéquate des indexes.
- ▶ Chaque SGBD étant différent, nous verrons ici l'utilisation avec MySQL. Oracle présente des concepts similaires mais offre une plus grande panoplie d'outils.



GPA775

97

## Langage SQL

### DDL | INDEX

- ▶ Un index peut être créé sur plusieurs colonnes à la fois.
- ▶ Chaque clé primaire crée automatiquement un index de type unique.

```
M CREATE [UNIQUE | FULLTEXT | SPATIAL] INDEX nom_index
ON nom_table( nom_colonne1 [(longueur_champ)] [ASC | DESC], ...)
[ USING {BTREE | HASH} ];
```

```
M DROP INDEX nom_index ON nom_table;
```

GPA775

98

## Langage SQL

### Les requêtes préparées

- ▶ Les requêtes préparées (RP) (ou requêtes paramétrisées ou « *prepared statement* » en anglais) sont des requêtes stockées et précompilées par le SGBD. Ces requêtes ne sont pas exécutées lors de leur déclaration.
- ▶ Via un mécanisme de liaison (« *binding* »), elles offrent un outil puissant permettant de définir les valeurs des paramètres de la requête lors de la demande de leur exécution.
- ▶ En effet, il est fréquent de faire la même requête en ne faisant varier que certains paramètres:

```
SELECT Nom, Prenom FROM Employe WHERE Departement = 'Ventes';
SELECT Nom, Prenom FROM Employe WHERE Departement = 'Achats';
```

- ▶ Il faut comprendre que pour un SGBD, les deux requêtes précédentes sont différentes et demandent une interprétation indépendante par le SGBD lors de leur exécution. En prenant cet exemple, si cette requête est fréquente, il sera préférable d'utiliser le mécanisme des requêtes préparées. Ainsi, une seule analyse est faite lors de la déclaration et tous les appels subséquents pourront avoir leur propre spécificité.

GPA775

99

## Langage SQL

### Les requêtes préparées

- ▶ L'usage de requête préparée se fait en trois étapes :
- ▶ **1 – Préparation :**
  - ▶ consiste à définir la requête préparée dans le SGBD;
  - ▶ ce dernier analyse la requête et initialise les ressources internes nécessaires à son exécution.
- ▶ **2 – Liaison et exécution :**
  - ▶ le client envoie les paramètres de la requête;
  - ▶ le SGBD lie les paramètres et exécute la requête avec les ressources préalablement allouées.
- ▶ **3 – libération des ressources :**
  - ▶ lorsque la requête préparée n'est plus nécessaire, il est important de libérer les ressources.

GPA775

100

## Langage SQL

### Les requêtes préparées

- ▶ Les requêtes préparées ne font pas partie de la norme SQL mais plutôt des outils supplémentaires offerts par les SGBD.
- ▶ Voici les éléments syntaxiques pour MySQL.

```

M PREPARE nom_requete_prep FROM requete_txt;
M EXECUTE nom_requete_prep [USING @variable1, @variable2, ...];
M [DEALLOCATE | DROP] PREPARE nom_requete_prep ;

```

- ▶ Le paramètre *requete\_txt* doit être une requête mise sous forme de chaîne de caractères. Tous les paramètres devant être liés plus tard doivent être identifiés par le caractère « ? ».
- ▶ Les variables passées à l'appel de EXECUTE doivent être présentées dans le même ordre que leur emplacement dans la requête (caractère « ? »).

GPA775

101

## Langage SQL

### Les requêtes préparées

- ▶ Il est possible de déclarer et d'assigner des variables sessions avec la clause SET ainsi.

```
M  PREPARE EmpParDepartement FROM
    'SELECT Nom, Prenom
    FROM Employe
    WHERE Departement = ? AND Salaire > ?';

M  SET @dep = 'Ventes', @sal = 75000;
M  EXECUTE EmpParDepartement USING @dep, @sal;

M  SET @dep = 'Achats', @sal = 85000;
M  EXECUTE EmpParDepartement USING @dep, @sal;

M  DEALLOCATE PREPARE EmpParDepartement;
```

GPA775

102

## Langage SQL

### Les requêtes préparées

- ▶ Attention, il existe plusieurs restrictions aux requêtes préparées. On ne peut mettre à titre de paramètre les éléments suivants :
  - ▶ les mots réservés du langage SQL : SELECT, FROM, CREATE, ...
  - ▶ les opérateurs : =, <, >=, AND, ...
  - ▶ Les fonctions : NOW, UPPER, ...
  - ▶ Les objets de la base de données (aucune table, vue, index ou autre).

GPA775

103

## Langage SQL

### Les procédures stockées

- ▶ Une procédure stockée (PS) est un ensemble d'instructions précompilées.
- ▶ Comme pour les requêtes préparées, les PS sont interprétées, analysées, planifiées et optimisées lors de leur déclaration et rendues disponibles pour une exécution rapide et flexible via des paramètres définies en entrée et en sortie.
- ▶ Les PS ne font pas partie de la norme SQL mais ils sont plutôt des outils supplémentaires offerts par les SGBD. D'ailleurs, tous les SGBD offrant les PS offrent un langage procédurale complémentaire permettant une plus vaste possibilité que le simple SQL.
- ▶ Oracle offre le langage PL/SQL et Java tandis que MySQL offre un langage propriétaire très similaire au PL/SQL. Nous verrons celui de MySQL.

GPA775

104

## Langage SQL

### Les procédures stockées

- ▶ Les requêtes préparées et procédures stockées partagent beaucoup de points en commun au niveau des avantages. Néanmoins, les mécanismes sous jacents sont très différents.
- ▶ Parmi les avantages qu'elles offrent :
  - ▶ simplification pour les développeurs;
  - ▶ rapidité par la réduction de la bande passante;
  - ▶ gain de performance puisque déjà préparées lors de leur déclaration,
  - ▶ gain de performance puisqu'exécutées du côté serveur;
  - ▶ accroissement de la sécurité en limitant l'accès aux objets définies uniquement dans les PS.

GPA775

105

# Langage SQL

## Les procédures stockées

- ▶ Inconvénients :
  - ▶ développement et maintenance plus difficiles;
  - ▶ peu d'outil de débogage pour aider au développement (MySQL n'offre rien);
  - ▶ peu engendrer une surcharge de calcul du côté serveur.
- ▶ La présentation qui suit prend pour acquis que le lecteur a déjà des connaissances de base avec un langage de programmation procédurale tel que le langage C.

GPA775

106

# Langage SQL

## Les procédures stockées

### Syntaxes | Déclaration

- ▶ Présentation des principaux éléments syntaxiques utilisés par MySQL.
- ▶ Attention à l'instruction **DELIMITER**.
- ▶ **Déclaration, appel et suppression** d'une procédure stockée.

```

M DELIMITER //

CREATE PROCEDURE nom_procedure[(paramètres)]
  BEGIN
    contenu_procedure
  END //

DELIMITER ;

CALL nom_procedure;

DROP PROCEDURE nom_procedure;
```

GPA775

107

# Langage SQL

## Les procédures stockées

### Syntaxes | Variables

- **Variables locales** : déclaration et affectation. Les variables sont créées et libérées avec la fonction.

```
M DECLARE nom_variable TYPE(option) [DEFAULT valeur];

SET nom_variable = valeur;
SELECT colonne INTO nom_variable FROM nom_table ...;
```

- Les **variables de session** sont disponibles pendant toute la session de l'utilisateur et ne sont pas liées à une procédure en particulier. Elles sont précédées par le caractère @.

GPA775

108

# Langage SQL

## Les procédures stockées

### Syntaxes | Paramètres

- Il existe trois types de **paramètres** pouvant être utilisés avec les PS : IN, OUT et INOUT.
  - IN : ce paramètre sert à passer une valeur à la procédure (paramètres par défaut). La modification de ce paramètre à l'intérieur de la procédure n'a aucun effet sur la valeur externe.
  - OUT : ce paramètre permet à la procédure de retourner une valeur au programme appelant. La valeur initiale de ce paramètre est inaccessible.
  - INOUT : combinaison des deux types précédents.

```
M MODE nom_paramètre TYPE(option)

CREATE PROCEDURE NomDepartement(IN noDep NUMERIC, OUT nomDep VARCHAR(32))
BEGIN
    SELECT Nom INTO nomDep FROM Departement WHERE Id = noDep;
END
```

GPA775

109

# Langage SQL

## Les procédures stockées

### Syntaxes | Structures conditionnelles

- Structure conditionnelle **IF** et **CASE** :

```
M  IF condition THEN instructions
    [ELSEIF condition THEN instructions]
    [ELSE instructions]
END IF

CASE expression
  WHEN expression_case_1 THEN instructions_1
  WHEN expression_case_2 THEN instructions_2
  ...
  ELSE instructions
END CASE
```

GPA775

110

# Langage SQL

## Les procédures stockées

### Syntaxes | Boucles

- Boucles **WHILE**, **REPEAT** et **LOOP** :

```
M  WHILE condition DO
    instructions
END WHILE

REPEAT
  instructions
UNTIL expression
END REPEAT

identifiant_loop: LOOP
  instructions
  [LEAVE identifiant_loop;]
  [ITERATE identifiant_loop;]
END LOOP
```

équivalent au break en langage C  
équivalent au continue en langage C

GPA775

111

## Langage SQL

### Les procédures stockées

#### Syntaxes | Curseur

- ▶ Un curseur est un outil important des SGBDR. Ils permettent le parcours ligne par ligne du contenu d'une table. On peut ainsi faire un traitement spécifique pour chacune des lignes.
- ▶ Les curseurs ont une syntaxe plus exigeante et une lecture plus approfondie de ce sujet est nécessaire pour une compréhension complète. Néanmoins, voici les éléments importants.
- ▶ Les curseurs s'utilisent à l'intérieur d'une procédure stockée

GPA775

112

## Langage SQL

### Les procédures stockées

#### Syntaxes | Curseur

- ▶ Pour MySQL, les curseurs ont ces propriétés :
  - ▶ ils sont en lecture seule (on ne peut donc modifier directement une entrée dans la table);
  - ▶ le parcours se fait uniquement dans l'ordre de la requête utilisée;
  - ▶ ils travaillent directement sur les données sources (plus rapide mais sensible aux tâches parallèles qui peuvent modifier les données).
- ▶ L'usage d'un curseur se fait en quatre grandes étapes :
  - ▶ 1- Déclaration du curseur et de la condition d'arrêt.
  - ▶ 2- Ouvrir le curseur.
  - ▶ 3- Accéder aux lignes une à la fois dans une boucle de contrôle.
  - ▶ 4- Fermer le curseur.



GPA775

113

# Langage SQL

## Les procédures stockées

### Syntaxes | Curseur

```

M -- Étape 1a : déclaration du curseur
  DECLARE nom_curseur CURSOR FOR requete_standard;
-- Étape 1b : déclaration de la var. de contrôle et de la condition d'arrêt
  DECLARE variable_controle INTEGER DEFAULT valeur_depart;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET variable_controle = valeur_arrêt;

-- Étape 2 : ouverture du curseur
  OPEN nom_curseur;

-- Étape 3 : parcourir les données
label_loop: LOOP;
  FETCH nom_curseur INTO variable_reception1 [,variable_reception2 ...]
  IF variable_controle = valeur_arrêt THEN
    LEAVE label_loop;
  END IF;
  -- faire quelque chose
END LOOP label_loop;
-- Étape 4 : fermer curseur
CLOSE label_loop;

```

GPA775

114

# Langage SQL

## Les procédures stockées

### Syntaxes | Curseur

```

M -- Procedure retournant une liste de courriel pour l'envoi à toutes les
  femmes de l'entreprise

  DELIMITER //
  DECLARE PROCEDURE ListeCourriel(OUT liste TEXT)
  BEGIN
    DECLARE curseurEmploye CURSOR FOR
      SELECT Genre, Courriel FROM Employe;
    DECLARE finParcours INTEGER DEFAULT 0;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET finParcours = 1;
    DECLARE empGenre AS CHAR(1);
    DECLARE empCourriel AS VARCHAR(64);
    DECLARE listeTemp AS TEXT DEFAULT "";

    OPEN curseurEmploye;

```

GPA775

115

# Langage SQL

## Les procédures stockées

### Syntaxes | Curseur

M

```
-- ...
parcoursEmploye: LOOP;
    FETCH curseurEmploye INTO empGenre, empCourriel
    IF finParcours = 1 THEN
        LEAVE parcoursEmploye;
    END IF;

    IF empGenre = 'F';
        SET listeTemp = CONCAT(listeTemp, empCourriel, ";");
    END IF;
END LOOP parcoursEmploye;

SET list = listeTemp;
CLOSE curseurEmploye ;
END //
DELIMITER ;
```

GPA775

116

# Langage SQL

## Les procédures stockées

- ▶ Plusieurs autres éléments pourraient être couverts concernant les procédures stockées.
- ▶ Par exemple, la gestion d'erreur et l'émission de condition d'erreur sont des outils puissants permettant une gestion efficace et serrée du SGBD.
- ▶ En lien avec les procédures stockées, il est possible d'écrire des fonctions stockées qui sont pratiquement identiques aux procédures.
- ▶ La principale différence est qu'une fonction retourne une et une seule valeur. Par conséquent, tous les paramètres de la fonction sont implicitement identifiés comme étant des paramètres de type **IN**.
- ▶ De plus, les fonctions peuvent être appelées à n'importe quel endroit sans nécessité de l'instruction **CALL**.

GPA775

117

## Langage SQL

### Les procédures stockées

```

M CREATE FUNCTION nom_function(param1 [, param2 ...]) RETURNS type_de_retour
  BEGIN
    -- ...
    RETURN valeur_retournée;
  END

DELIMITER //
CREATE FUNCTION initialCompact(Nom VARCHAR(32), Prenom VARCHAR(32)) RETURNS CHAR(2)
  BEGIN
    RETURN UPPER(CONCAT(LEFT(Prenom, 1), LEFT(Nom, 1)));
  END //
DELIMITER ;

SELECT initialCompact('Cheriet', 'Mohamed'); -- retourne : MC

```

GPA775

118

## Langage SQL

### Les déclencheurs

- ▶ Les déclencheurs (TRIGGERS en anglais) sont des objets de la base de données permettant d'exécuter des tâches spécifiques à des moments spécifiques.
- ▶ Les événements pouvant être associés à un TRIGGER sont principalement liés aux opérations de manipulation des données : INSERT, UPDATE, DELETE.
- ▶ Pour MySQL, il est aussi possible de créer des EVENT basé sur le temps.
- ▶ La différence majeure entre les TRIGGER et les EVENT est que les TRIGGER sont synchrones (BEFORE INSERT, AFTER INSERT, ...) alors que les EVENT sont asynchrones.

GPA775

119

## Langage SQL Les déclencheurs

- ▶ Avantages et inconvénients :
  - ▶ permet une solution avancée d'assurer l'intégrité référentielle;
  - ▶ permet d'identifier des erreurs logiques lors des transactions liées au DML;
  - ▶ les événements permettent d'exécuter des tâches planifiées (maintenance, gestion, ...);
  - ▶ outils très puissants pour gérer les historiques de transaction liées au DML;
  - ▶ limitation de certains SGBD sur les réelles possibilités apportées par le concept;
  - ▶ les déclencheurs sont invisibles aux applications clientes et ainsi il peut être difficile de tracer la ligne entre ce qui doit être traité côté serveur / côté client;
  - ▶ les déclencheurs exigent des ressources significatives;
  - ▶ les déclencheurs complexifient le schéma et les interrelations de la BD. Son développement et sa maintenance peuvent être très coûteuses.

GPA775

120

## Langage SQL Les déclencheurs

- ▶ Pour le langage SQL, voici les éléments syntaxiques importants :

```
M CREATE TRIGGER nom_trigger
    { BEFORE | AFTER } { INSERT | UPDATE | DELETE }
    ON nom_table
    FOR EACH ROW
    BEGIN
        ... liste d'instructions
    END
```

- ▶ **BEFORE** et **AFTER** indique que l'évènement est appelé avant ou après l'instruction qui peut être ces éléments du DML **INSERT**, **UPDATE** et **DELETE**.
- ▶ À l'intérieur du TRIGGER, les instructions **OLD** et **NEW** font référence à la ligne antérieurement inscrite dans la table et à la nouvelle données. Un seul type de référence est possible à la fois.

GPA775

121

## Langage SQL Les déclencheurs

```
M CREATE TABLE Vehicule (
    Id          INTEGER PRIMARY KEY,
    Marque      VARCHAR(32),
    Modele      VARCHAR(32),
    Annee       DECIMAL(4,0),
    Etat        ENUM('Disponible', 'Réparation', 'Loué', 'Indisponible'
);
CREATE TABLE HistoriqueLocationVehicule (
    Id          SERIAL PRIMARY KEY,
    IdVehicule  INTEGER,
    Etat        ENUM('Disponible', 'Réparation', 'Loué', 'Indisponible'),
    DateAction  TIMESTAMP,
    FOREIGN KEY (IdVehicule) REFERENCES Vehicule(Id)
);
```

GPA775

122

## Langage SQL Les déclencheurs

```
M DELIMITER //
CREATE TRIGGER mise_a_jour_historique
    AFTER UPDATE ON Vehicule FOR EACH ROW
BEGIN
    IF OLD.Etat <> NEW.Etat THEN
        INSERT INTO HistoriqueLocationVehicule(IdVehicule, Etat, DateAction)
            VALUES (NEW.Id, NEW.Etat, CURTIME());
    END IF;
END //
DELIMITER ;

-- il faudrait créer le même trigger pour le AFTER INSERT

INSERT INTO Vehicule(Id, Marque, Modele, Annee)
    VALUES(77, 'Ford', 'Mustang', 2013);
UPDATE Vehicule SET Etat = 'Réparation' WHERE Vehicule.Id = 77;
UPDATE Vehicule SET Etat = 'Disponible' WHERE Vehicule.Id = 77;
UPDATE Vehicule SET Etat = 'Loué' WHERE Vehicule.Id = 77;
UPDATE Vehicule SET Etat = 'Disponible' WHERE Vehicule.Id = 77;
```

GPA775

123

# Langage SQL

## Les déclencheurs

- ▶ Un exemple simple d'évènement :

```
M      CREATE EVENT exemple_event
        ON SCHEDULE AT '2006-02-10 23:59:00' DO
        INSERT INTO table_du_temps VALUES (NOW());
```

GPA775

124

# Langage SQL

## TCL

- ▶ Le TCL est un concept important de tous les SGBD lié au langage SQL.
- ▶ Les propriétés d'une transaction :
  - ▶ transaction atomique : toutes les opérations unitaires nécessaire à une transaction ont réalisées (sinon, on annule les opérations intermédiaires);
  - ▶ consistance : la base de données reste consistante après chaque transaction validée;
  - ▶ isolation : chaque transaction est transparentes pour les autres et assure un fonctionnement indépendant de ces dernières;
  - ▶ durabilité : assure la persistance des transactions validées.

GPA775

125

## Langage SQL TCL

- ▶ Pour MySQL, on retrouve principalement 5 instructions liées au TCL :
  - ▶ **SAVEPOINT** : crée un point de sauvegarde intermédiaire dans la transaction;
  - ▶ **COMMIT** : valide la transaction en cours et sauvegarde les modifications faites;
  - ▶ **ROLLBACK** ou **ROLLBACK TO savepoint**: annule la transaction en cours et annule les modifications faites;
  - ▶ **SET autocommit** = { 0 | 1 };  
active ou désactive la validation automatique (par défaut, autocommit = 1);
  - ▶ **START TRANSACTION** : permet la déclaration du début d'une transaction et certaines options .
- ▶ Dès qu'une instruction COMMIT ou ROLLBACK a été exécuté, une nouvelle transaction débute et terminera au prochain appel du COMMIT ou ROLLBACK.

GPA775

126

## Langage SQL TCL

- ▶ Quelques précisions sur le déroulement du **START TRANSACTION** :
  - ▶ si il y a une transaction en cours, un COMMIT est automatiquement appliqué pour terminer la transaction antérieure;
  - ▶ si la validation automatique est activée, elle est temporairement désactivée;
  - ▶ définit le point de départ de la transaction et applique les options;
  - ▶ la transaction a cours jusqu'au prochain COMMIT ou ROLLBACK;
  - ▶ si la validation automatique avait été suspendue, elle est réactivée;
- ▶ Le standard SQL recommande son usage.

GPA775

127

# Langage SQL TCL

```

M  INSERT ...           -- operation 1 : sauvegardée autocommit par défaut

SET autocommit = 0;
INSERT ...             -- operation 2 : début d'une transaction
INSERT ...             -- operation 3 : poursuite de la transaction
COMMIT;                -- fin de la transaction et les opérations 2 et 3
                        -- sont sauvegardées
UPDATE ...             -- operation 4 : début d'une nouvelle transaction
DELETE ...             -- operation 5 : poursuite de la transaction
ROLLBACK;              -- fin de la transaction et les opérations 4 et 5
                        -- sont annulées

```

GPA775

128

# Langage SQL TCL

```

M  ...

START TRANSACTION;     -- début d'une transaction, toutes les operations
                        -- antérieures sont sauvegardées (COMMIT)
INSERT ...             -- operation 1 : poursuite de la transaction
DELETE ...             -- operation 2 : poursuite de la transaction

SAVEPOINT okTemp;      -- définit un point de reprise

INSERT ...             -- operation 3 : poursuite de la transaction
DELETE ...             -- operation 4 : poursuite de la transaction

ROLLBACK TO okTemp;    -- Annule les operations 3 et 4.

INSERT ...             -- operation 5 : poursuite de la transaction
DELETE ...             -- operation 6 : poursuite de la transaction

IF ... THEN COMMIT     -- fin de la trans. et sauvegarde op. 1, 2, 5 & 6
ELSE ROLLBACK END IF;  -- fin de la trans. et annule les op. 1, 2, 5 & 6

```