

# Langage SQL

## Introduction

ULCO

7 february 2024

- 1 Introduction
- 2 DDL
- 3 DML
- 4 DQL
- 5 Compléments

# Introduction

# Introduction

# Bases de données

## Base de données

Une base de données (en anglais : database) est un ensemble de données structurées, généralement stocké sur un support de persistance (disque dur).

## Base de données relationnels

Les données d'une base de données relationnelle (en anglais : relational database) sont organisées sous forme de tables mises en relation. Elles s'appuient sur la théorie des relations de Codd.

# SGBD

## SGBD

Un SGBD (en anglais : DBMS, Database Management System) est un ensemble de programmes qui gère le contenu et la structure d'une base de données de manière : sécurisée, partagée, cohérente, fiable, performante, indépendante

## SGBDR

Les SGBDR (R pour Relationnels) sont les SGBD des bases de données relationnelles (en anglais : RDBMS, Relational Database Management System). Un langage standard, SQL, permet le dialogue entre les programmes (des clients) et le SGBDR (le serveur de données).

# SGBD - types

## SGBDr fichier

Logiciel qui s'appuie sur des mécanismes de partage d'un fichier de base de données (MS Access, OpenOffice Base)

## SGBDR serveur

Logiciel qui gère les fichiers d'une base de données, qui est exécuté sous forme d'un processus, et qui répond aux requêtes des programmes.  
Fonctionnement de type client-serveur.

# Intégrité

## Intégrité

L'intégrité est définie par le respect de certaines règles comme :

- intégrité de domaine : vérification du type de données et des valeurs autorisées
- intégrité d'entité : vérification que chaque ligne est identifiable (pas de doublons de lignes)
- intégrité référentielle : vérification qu'une colonne d'une table référence une autre colonne (d'une autre table), et que cette dernière ne peut être supprimée si elle est référencée

## Contraintes

Les contraintes sont les règles qu'on assigne au SGBD et qu'il devra vérifier afin de garantir l'intégrité des données

# Indépendance et norme ANSI/SPARC

## Indépendance

La norme définit 3 niveaux d'indépendance devant être mis en oeuvre par un SGBD (digne de ce nom...)

- niveau externe : des vues du schémas de la base de données sont offertes aux clients-programmes (en fonction de leur rôle, de leur responsabilité, etc.
- vue conceptuelle (ou logique) : des tables sont définies avec leurs contraintes
- vue interne (ou physique) : des fichiers physiques contiennent les tables et sont répartis sur des supports de persistance (disque dur, généralement)



# Indépendance et norme ANSI/SPARC

## Indépendances

On doit pouvoir modifier

- le schéma physique (répartition sur différents disques)
- le schéma (ajout de tables)
- les vues (modification des vues)

sans incidence sur les autres niveaux

# Passage modèle relationnel vers le modèle physique

## Règles

- chaque relation devient une table
- chaque attribut devient une colonne associé à un type de données (perte du domaine en général, certains SGBDr définissent la notion de domaine de valeurs)
- un attribut (ou un ensemble d'attributs) clef primaire devient clef primaire de la table (application du controle d'intégrité d'entité par le SGBD)
- un attribut (ou un ensemble d'attributs) clef étrangère devient clef étrangère de la table et cible l'attribut (ou un ensemble d'attributs) formant la clef primaire d'une (autre) table (application du controle d'intégrité référentielle par le SGBDr)
- une clef candidate devient un index

# Choix du SGBD et types de données

Le domaine des attributs est généralement remplacé par le type d'une colonne.

## Types de données

Chaque SGBDR va définir son propre jeu de types de données : mais un certain nombre de types sont standardisés. Par exemple ici pour le SGBDR MySQL (libre et associé à Oracle)

- chaîne fixe : `char(longueur)` et chaîne variable : `varchar(longueur maximale)`
- entier : `int`
- numérique avec un nombre de décimales fixe : `numeric(longueur, dont décimales)`
- réel : `real`
- date : `date`
- heure : `time`

# Conventions

## nommage

caractères : [a..z] ou [A..Z], [0..9], [\_] pour coder les noms de tables et colonnes.

## écriture

SGBDR : généralement insensibles à la casse des caractères en ce qui concerne les composants (ordres, clauses) de leur syntaxe. Deux approches :

- les éléments SQL en lettres capitales et les éléments variables (tables et colonnes) en lettres minuscules
- ou l'inverse
- ou *tout en minuscule* : les éditeurs mettent en évidence la syntaxe

# Modèle Physique de Données

## représentation

vol		
<u>numvol</u>	int	PK
numpil	int	FK
numav	int	FK
departvol	date	
dureevol	decimal(6,2)	
vildepvol	varchar(20)	
vilarrvol	varchar(20)	

*numpil = numpil*

*numav = numav*

pilote		
<u>numpil</u>	int	PK
nompil	varchar(20)	
vilpil	varchar(20)	
salairepil	numeric(10,2)	

avion		
<u>numav</u>	int	PK
nomav	varchar(20)	
localisav	varchar(20)	
capamaxav	int	

Les domaines ont été remplacés par des types de données spécifiques au SGBDR cible.

# SQL

SQL (en anglais : *Structured Query Language*) : LE langage permettant de donner des ordres aux SGBDR afin d'accéder aux services qu'il offre

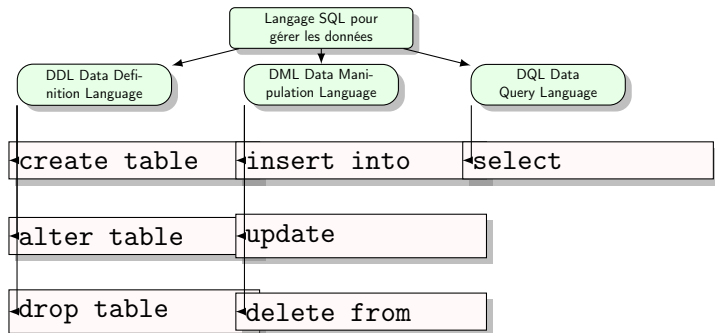
## Versions

- SQL1 (SQL89) : première version avec quelques défauts corrigés dans la suivante
- → *SQL2 (SQL92)* : la version la plus implantée actuellement (norme ANSI X3.135-1992, [www.ansi.org](http://www.ansi.org)); les versions suivantes vont étendre les possibilités
- SQL3 (SQL99) : Hybride objet-relationnel
- SQL4 (2003) : SQL Object Langage Binding, OLAP
- SQL5 (2006) : SQL/XML, XQuery
- SQL6 (2008) : SQL/JRT pour fonctions Java
- SQL7 (2011) : ISO/IEC 9075-2011

# SQL

## Famille d'ordres SQL pour gérer les données

- DDL : définir la structure des tables
- DML : gérer le contenu des tables
- DQL : interroger le contenu des tables (requêtes d'interrogation)



# SQL

Une *requête* est un ordre transmis au SGBD(R) afin de modifier la structure ou le contenu de la base de données, ou d'interroger les données qui y sont stockées.

Une requête retourne un résultat différent selon le type d'ordre : soit une information signalant que l'ordre a bien été exécuté, soit les informations qui ont été demandées.



# SQL

Les requêtes SQL d'interrogation (DQL) sont soumises au SGBDR qui effectue les traitements suivants :

- 1 *analyse lexicale* du texte de la requête : en cas d'erreur de syntaxe, une erreur est retournée
- 2 *traduction en opérateurs d'algèbre relationnelle*
- 3 *optimisation* à partir des statistiques de volumétrie des tables utilisées et construction du plan d'exécution
- 4 *exécution* et retour du résultat

# Terminologie

<i>algèbre relationnelle</i>	<i>bases de données relationnelles et SQL</i>
relation	table (en anglais : <i>table</i> )
attribut	colonne (en anglais : <i>column</i> ) ou champ (en anglais : <i>field</i> )
n-uplet ou tuple	ligne (en anglais : <i>row</i> ) ou enregistrement (en anglais : <i>record</i> )
domaine de valeur	type de donnée (en anglais : <i>data type</i> )
expression d'algèbre relationnelle	requête (en anglais : <i>query</i> )

# DDL - structure des tables

- create table : créer une table
- alter table : modifier une table
- drop table : supprimer une table

# Les contraintes d'intégrité

- `not null` : assure qu'une colonne ne peut avoir une valeur nulle
- `default` : assure une valeur par défaut au cas où la valeur n'est pas renseignée pour une colonne
- `unique` : assure que toutes les valeurs d'une colonne seront différentes (chacune sera unique...)
- `primary key` : identifie chaque ligne avec une valeur unique
- `foreign key` : identifie une ligne d'une autre table (généralement) avec une valeur unique
- `check` : détermine les conditions de validité d'une valeur, les valeurs autorisées pour une colonne

# Créer une table

```
create table nomTable (  
    nomColonne type [contrainte][auto_increment][default],  
    [nomColonne type [contrainte][default],etc.]  
);
```

où :

- *nomColonne* : nom d'une colonne de la table
- *type* : type de donnée associé à la colonne
- *contrainte* : contrainte concernant une seule colonne
- *default* : valeur par défaut lors de l'ajout d'une ligne
- *auto\_increment* : mot-clef précisant que la valeur de cette colonne sera, par défaut, incrémentée automatiquement

# Créer une table - exemple

```
create table personnel (
  numero      int      not null,
  nom         varchar(16) not null,
  prenom      varchar(16) not null,
  ville       varchar(16) null,
  salaire     decimal(8,2) not null default 100,
  dateentree  date     not null,
  sexe       char(1)   not null default 'h'
);
```

## Modifier une table : structure, contraintes

```
alter table  nomTable (  
    add constraint defContrainte  
| add [column] defColonne  
| rename [to|as] nouvNomTable  
| drop [constraint] nomContrainte  
| drop foreign key nomContrainte  
| drop primary key  
| drop [column] nomColonne  
);
```

où :

- *nomTable* : nom de la table concernée par la modification de structure
- *defContrainte* : contrainte concernant une colonne ou une table : type, nom, etc.
- *defColonne* : définition d'une colonne : nom, type, contrainte
- *nomColonne* : nom d'une colonne de la table
- *nouvColonne* : nouveau nom de colonne de la table
- *nomContrainte* : nom d'une contrainte associée à la table

## Modifier une table : exemple

Listing 1 – Ordre SQL d'ajout d'une colonne 'stockMini'

```
alter table produit
  add column stockMini real null default 0
;
```

Listing 2 – Ordre SQL d'ajout de clef primaire (intégrité d'entité)

```
alter table produit
  add constraint PK_Produit
  primary key (refProduit)
;
```

Listing 3 – Ordre SQL d'ajout de clef étrangère (contrainte d'intégrité référentielle)

```
alter table produit
  add constraint FK_Produit_Famille
  foreign key (codeFamille)
  references famille (codeFamille)
;
```



# Supprimer une table

```
drop table [if exists] nomTable;
```

où :

- *nomTable* : nom de la table concernée par la suppression

## Supprimer une table - exemple

### Listing 4 – Suppression de la table produit

```
drop table produit ;
```

Soient les tables 'produit' et 'famille', soit la clef étrangère  
Fk\_Produit\_Famille :

Listing 5 – Ordre SQL de suppression de la table 'famille', cible d'une contrainte  
d'intégrité référentielle

```
drop table famille  
;
```

L'ordre de suppression de la table 'famille' produit une erreur :



**ERROR 1217 (23000):**

| Cannot delete or update a parent row: a foreign key constraint fails

# Exercice

- 1 créer la structure des tables de la base de données 'concert'
- 2 ajouter les clef primaires et clefs étrangères

# SQL

## DML

- `insert into` : ajouter des lignes à une table
- `update` : modifier des valeurs de colonnes d'une table
- `delete from` : supprimer des ligne d'une table

## Ajouter des lignes

```
insert into nomTable
    (col1[, col2, col3, ...])
values ( val1[,val2, val3, ...])[,
    ( val1[,val2, val3, ...]),...]
;
```

où :

- *nomTable* : nom de la table concernée par l'ajout
- *col1, col2, col3, ...* : liste des noms de colonnes dont les valeurs vont être précisées dessous
- *val1, val2, val3, ...* : liste des valeurs associées chacune à une colonne; toutes les colonnes doivent avoir une valeur qui leur correspond (dans l'ordre défini juste au dessus)

Cette syntaxe est la plus sûre :

- elle est indépendante l'ordre des colonnes dans la table
- elle reste correcte même si des colonnes (non obligatoires) ont été ajoutées

# Ajouter une ligne - exemple

## Listing 6 – Ordre SQL d'ajout d'une personne

```
insert into personne
    (numero, nom, prenom, salaire, dateEntree)
values (5, 'Rigole', 'Jean', 1300, '2007-09-28')
```

# Modifier des valeurs : contenu

```
update nomTable
  set col1 = valeur1 [,col2 = valeur2, col3 = valeur3,...]
  [where condition]
;
```

où : où :

- *nomTable* : table mise à jour
- *col1, col2, col3, ...* : colonnes dont le contenu est modifié
- *valeur1, valeur2, valeur3, ...* : nouvelles valeurs
- *condition* : condition spécifiée peut être exprimée :
  - par rapport aux colonnes de la table mise à jour
  - par rapport à une sous-requête indépendante ou corrélée

## Modifier des valeurs : exemple

Listing 7 – Ordre SQL de modification de la colonne 'prenom' d'une personne sélectionné sur son numéro

```
update personne
  set prenom = 'john'
  where numero = 4
;
```

Listing 8 – Ordre SQL de modification de la colonne 'salaire', augmentation de 10% pour tous

```
update personne
  set salaire = salaire * 1.1
;
```

Listing 9 – Ordre SQL de modification de la colonne 'salpil', augmentation de 10% pour tous les pilotes qui ont assuré un vol

```
update pilote
  set salpil = salpil * 1.1
  where numpil in (select numpil from vol)
```



# Supprimer des lignes

```
delete from nomTable  
  [where condition]  
;
```

où :

- *nomTable* : nom de la table concernée par la suppression
- *condition* : la condition de sélection des lignes à supprimer, exprimée :
  - en lien avec les colonnes de la table
  - en utilisant une sous-requête (indépendante ou corrélée)

## Supprimer des lignes : exemple

Listing 10 – Ordre SQL de suppression des lignes de pilote dont le numero est 10

```
delete from pilote
  where numpil = 10
;
```

Listing 11 – Ordre SQL de suppression des lignes de pilote qui n'ont pas volé

```
delete from pilote
  where numpil not in (
    select distinct numpil
      from vol
  )
;
```

# Exercice

- 1 ajouter les lignes dans les tables
- 2 effectuer des modifications

# SQL

## DQL

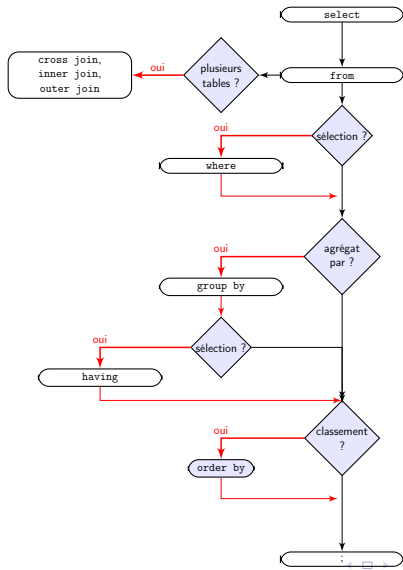
- `select` : interroger les valeurs de tables

# SQL

Une phrase pour interroger les lignes et colonnes des tables; cette phrase va "contenir" tous les opérateurs relationnels d'une requête algébrique équivalente

```
select [distinct | all] {* | listeDeColonnes}  
  from table(s)  
  [where criteresDeSelection]  
  [group by listeDeColonnesDeRegroupement]  
  [having criteresDeSelectionApresRegroupement]  
  [order by criteresDeClassement]  
;
```

## SQL



## SQL

 Projection SQL

$\pi_{xa,\dots,xz}(R)$  (suppression des doublons)

```
select distinct xa , . . . , xz
  from R
;
```

(pas de suppression des doublons : par défaut)

```
select [all] xa , . . . , xz
  from R
;
```

où :

- `distinct` : mot-clef SQL de suppression des doublons
- `all` : mot-clef SQL de non-suppression des doublons (action par défaut)
- `xa, ..., xz` : la liste des colonnes conservées

# Projection SQL - exemple 1

Listing 12 – lister les noms et prenom des personnes

```
select nom, prenom, ville, dateEntree  
from personne;
```

personne	nom	prenom	ville	dateEntree
	Dupont	Max	arras	2007-01-01
	Durand	Tim	Aix	2007-03-15
	Lambert	Betty	Pau	2007-04-20
	Bradford	Jean	Arras	2007-09-07
	Rigole	Jean	null	2007-09-28

nombre de lignes: 5



# Projection SQL - exemple 1

Listing 13 – lister les noms et prenom des personnes

```
select nom, prenom  
from personne;
```

personne	nom	prenom
	Dupont	Max
	Durand	Tim
	Lambert	Betty
	Bradford	Jean
	Rigole	Jean

nombre de lignes: 5

# Projection SQL - exemple 2

## Listing 14 – lister les personnes

```
select *  
  from personne;
```

personne	numero	nom	prenom	ville	salaire	dateEntree	sexe
	1	Dupont	Max	arras	1000.00	2007-01-01	h
	2	Durand	Tim	Aix	1500.00	2007-03-15	h
	3	Lambert	Betty	Pau	1350.00	2007-04-20	f
	4	Bradford	Jean	Arras	1250.00	2007-09-07	h
	5	Rigole	Jean	null	1300.00	2007-09-28	h

nombre de lignes: 5

## Projection SQL - exemple 3

Listing 15 – lister les prenom et sexe

```
select prenom, sexe
from personne;
```

personne	prenom	sexe
	Max	h
	Tim	h
	Betty	f
	Jean	h
	Jean	h

nombre de lignes: 5

On constate que 2 lignes sont identiques<sup>1</sup>

---

<sup>1</sup>parfois à la casse près : mais généralement les SGBDr sont paramétrés comme étant CI, Case Insensitive, c'est à dire insensibles à la casse des caractères

# Projection SQL - exemple 4

Listing 16 – lister les prenom et sexe (sans doublons)

```
select distinct prenom , sexe  
from personne ;
```

personne	prenom	sexe
	Max	h
	Tim	h
	Betty	f
	Jean	h

nombre de lignes: 4

# Exercice

- 1 lister les nom et prénom des musiciens
- 2 lister les différentes villes de résidence des musiciens
- 3 lister les instruments

## SQL

 Sélection SQL $\sigma_Q(R)$ 

```
select *  
  from R  
 where Q  
;
```

où :

- \* : facilité d'écriture = 'toutes les colonnes'
- $R$  : table d'origine
- $Q$  : condition de sélection des lignes


## SQL

 Opérateurs logiques et connecteurs

égal	=
différent	!= ou <>
inférieur	<
inférieur ou égal	<=
supérieur	>
supérieur ou égal	>=

et	and
ou	or
non	not

## SQL

 Opérateurs SQL étendus

est égal à une valeur au moins d'une liste	in (liste)
est différent de toutes les valeurs d'une liste	not in (liste)
est dans l'intervalle $[a, b]$	between a and b
n'est pas dans l'intervalle $[a, b]$	not between a and b
ressemble à un modèle	like "modele"
ne ressemble pas à un modèle	not like "modele"
a une valeur nulle	is null
n'a pas une valeur nulle	is not null



# Sélection SQL - exemple 1

Les personnes dont le numéro est 1, 3 ou 5 et qui sont entrées entre le 1er mars 2007 et le 30 avril 2007

```
select *
  from personne
 where (numero = 1 or numero = 3 or numero = 5)
       and (dateEntree >= "2007-03-01" and dateEntree <= "
           2007-04-30" )
;
```

resultat	numero	nom	prenom	ville	salaire	dateEntree	sexe
	3	Lambert	Betty	Pau	1350.00	2007-04-20	f

nombre de lignes: 1

## Sélection SQL - exemple 1

Les personnes dont le numéro est 1, 3 ou 5 et qui sont entrées entre le 1er mars 2007 et le 30 avril 2007

En utilisant les opérateurs SQL étendus :

```
select *
  from personne
  where numero in (1,3,5)
        and dateEntree between "2007-03-01" and "2007-04-30"
;
```

resultat	numero	nom	prenom	ville	salair	dateEntree	sexe
	3	Lambert	Betty	Pau	1350.00	2007-04-20	f

nombre de lignes: 1

## Sélection SQL - exemple 2

Les personnes dont le numéro n'est pas 1, 3 ou 5 et qui ne sont pas entrées entre le 1er mars 2007 et le 30 avril 2007

```
select *
  from personne
 where (numero <> 1 and numero <> 3 and numero <> 5)
       and (dateEntree < "2007-03-01" or dateEntree > "
           2007-04-30" )
;
```

resultat	numero	nom	prenom	ville	salaire	dateEntree	sexe
	4	Bradford	Jean	Arras	1250.00	2007-09-07	h

nombre de lignes: 1

Le mélange de connecteurs OR et AND nécessite la présence de parenthèses !

## Sélection SQL - exemple 2

Les personnes dont le numéro n'est pas 1, 3 ou 5 et qui ne sont pas entrées entre le 1er mars 2007 et le 30 avril 2007

En utilisant les opérateurs SQL étendus :

```
select *
  from personne
 where numero not in (1,3,5)
       and dateEntree not between "2007-03-01" and "2007-04-30"
;
```

resultat	numero	nom	prenom	ville	salaire	dateEntree	sexe
	4	Bradford	Jean	Arras	1250.00	2007-09-07	h

nombre de lignes: 1

## Sélection SQL - exemple 3

Les personnes dont le nom commence par b ou se termine par e

```
select numero, nom, prenom, ville
  from personne
 where nom like "b%"
       or nom like "%e"
;
```

resultat	numero	nom	prenom	ville
	4	Bradford	Jean	Arras
	5	Rigole	Jean	null

nombre de lignes: 2

## Sélection SQL - exemple 3

Les personnes dont le nom contient ad

```
select numero, nom, prenom, ville
  from personne
 where nom like "%ad%"
;
```

resultat	numero	nom	prenom	ville
	4	Bradford	Jean	Arras

nombre de lignes: 1

## Sélection SQL - exemple 4

Les personnes dont le nom contient un m au 3eme caractère

```
select numero, nom, prenom, ville
  from personne
 where nom like "__m%"
;
```

resultat	numero	nom	prenom	ville
	3	Lambert	Betty	Pau

nombre de lignes: 1

## Sélection SQL - exemple 4

Les personnes dont la ville n'est pas renseignée (est nulle)

```
select numero, nom, prenom, ville
  from personne
 where ville is null
;
```

resultat	numero	nom	prenom	ville
	5	Rigole	Jean	null

nombre de lignes: 1



## Sélection SQL - exemple 4

Les personnes dont la ville est renseignée (n'est pas nulle)

```
select numero, nom, prenom, ville
  from personne
 where ville is not null
;
```

resultat	numero	nom	prenom	ville
	1	Dupont	Max	arras
	2	Durand	Tim	Aix
	3	Lambert	Betty	Pau
	4	Bradford	Jean	Arras

nombre de lignes: 4

# Exercice

- 1 lister les musiciens dont le numéro est soit 1, soit 2 soit 6
- 2 lister les musiciens qui ont pour prenom joe et qui résident à Lille
- 3 lister les musiciens qui sont nés en mai 1962
- 4 lister les musiciens dont le prenom contient la lettre e
- 5 lister les musiciens dont la ville de résidence n'est pas renseignée

## SQL

 Renommage de table en SQL - alias de table $\rho_S(R)$ 

```
select *  
  from R S  
;
```

où :

- $R$  : table d'origine
- $S$  : alias de la table

Après qu'un alias a été défini dans une requête, on ne peut plus faire référence au nom originel de la table dans la requête.

# Renommage SQL - exemple 1

## Listing 17 – lister les personnes

```
select *  
  from personne A;
```

personne	numero	nom	prenom	ville	salaire	dateEntree	sexe
	1	Dupont	Max	arras	1000.00	2007-01-01	h
	2	Durand	Tim	Aix	1500.00	2007-03-15	h
	3	Lambert	Betty	Pau	1350.00	2007-04-20	f
	4	Bradford	Jean	Arras	1250.00	2007-09-07	h
	5	Rigole	Jean	null	1300.00	2007-09-28	h

nombre de lignes: 5

# SQL

L'alias de colonne permet de donner un nom plus lisible à une colonne dans le résultat.

## Renommage de colonne en SQL - alias de colonne

$\rho_{x1 \rightarrow nx1, x2 \rightarrow nx2, \dots, xn \rightarrow nxn}(R)$

```
select x1 AS nx1 , x2 AS nx2 , ... , xn AS nxn
  from R
;
```

où :

- $x1, x2, \dots, xn$  : noms des attributs
- $nx1, nx2, \dots, nxn$  : nouveaux noms donnés aux attributs
- $R$  : table d'origine

L'alias de colonne est donnée dans la clause `select` et ne peut être utilisé qu'après la clause `where`.

## Renommage SQL - exemple 2

Listing 18 – Exemple SQL d'alias de colonne dans une projection

```
select nom, prenom, salaire as salaireMensuel
from personne;
```

resultat	nom	prenom	salaireMensuel
	Dupont	Max	1000.00
	Durand	Tim	1500.00
	Lambert	Betty	1350.00
	Bradford	Jean	1250.00
	Rigole	Jean	1300.00

nombre de lignes: 5

# Exercice

- 1 lister les musiciens en renommant idMusicien en numero, et villeResidence en ville

## SQL

 Classement des lignes en SQL

```
select *  
  from table(s)  
  [where ...]  
  [group by ...]  
  [having ...]  
  order by critere1[, criteres2, ...]  
;
```

où :

- *critere1*, *critere2*, etc. sont des critères de classement comportant chacun un nom de colonne suivi de :
  - soit *asc*, ordre croissant (en anglais : *ascending*), par défaut
  - soit *desc*, ordre décroissant (en anglais : *descending*)



# Classement SQL - exemple 1

Listing 19 – les personnes classées par prenom puis date d'entree

```
select *
  from personne
 order by prenom asc, dateEntree asc;
```

resultat	numero	nom	prenom	ville	salaire	dateEntree	sexe
	3	Lambert	Betty	Pau	1350.00	2007-04-20	f
	4	Bradford	Jean	Arras	1250.00	2007-09-07	h
	5	Rigole	Jean	null	1300.00	2007-09-28	h
	1	Dupont	Max	arras	1000.00	2007-01-01	h
	2	Durand	Tim	Aix	1500.00	2007-03-15	h

nombre de lignes: 5

# Classement SQL - exemple 1

Listing 20 – les personnes classées par salaire décroissant puis par nom croissant

```
select *
  from personne
 order by salaire desc, nom asc;
```

resultat	numero	nom	prenom	ville	salaire	dateEntree	sexe
	2	Durand	Tim	Aix	1500.00	2007-03-15	h
	3	Lambert	Betty	Pau	1350.00	2007-04-20	f
	5	Rigole	Jean	null	1300.00	2007-09-28	h
	4	Bradford	Jean	Arras	1250.00	2007-09-07	h
	1	Dupont	Max	arras	1000.00	2007-01-01	h

nombre de lignes: 5

# Exercice

- 1 lister les concerts classés par ville puis par date
- 2 lister les programmation classées par cachet décroissant puis numéro de concert croissant

# SQL

## Union en SQL - sans ou avec doublons (R U S)

```
requete1  
union [all]  
requete2  
;
```

où :

- *requete1* et *requete2* : toute requete SQL select valide, les 2 requêtes devant être union-compatibles
- `union` : mot-clef de l'union de 2 requêtes
- `all` : optionnel, permet de conserver les doublons de lignes

# Union SQL - exemple 1

Listing 21 – les personnes entrées en mars 2007 et celles habitant Arras

```
select *
  from personne
  where dateEntree between '2007-03-01'
                        and '2007-03-31'

union

select *
  from personne
  where ville = 'arras'
;
```

resultat	numero	nom	prenom	ville	salaire	dateEntree	sexe
	2	Durand	Tim	Aix	1500.00	2007-03-15	h
	1	Dupont	Max	arras	1000.00	2007-01-01	h
	4	Bradford	Jean	Arras	1250.00	2007-09-07	h

nombre de lignes: 3 Suppression des doublons (sauf si union all)

# Exercice

- 1 lister les villes des concerts et les villes de résidence
- 2 lister les titres des concerts et les noms des musiciens : possible ou pas ? cohérent ou pas ?

## SQL

## Produit cartésien en SQL - SQL92

$(R \times S)$  SQL92

```
select *  
  from R  
     cross join S  
;
```

SQL89 :

```
select *  
  from R, S  
;
```

où :

- $R$  et  $S$  : tables objets du produit cartésien
- `cross join` : le mot-clef SQL92 du produit cartésien

# Produit cartésien SQL - exemple 1

## Listing 22 – les pilotes et les vols

```
select *
  from pilote
      cross join vol
;
```

resultat	numpil	nompil	vilpil	datnaispil	salairpil	numvol	numpil	numav	departvol	dureevol	vildepvol	vilarrvol
1	William	Nice	1980-06-29	16000.00	1002	3	104	2015-09-01 09:00:00.0	20	Marseille	Nice	
2	Peter	Nice	1970-01-10	18000.00	1002	3	104	2015-09-01 09:00:00.0	20	Marseille	Nice	
3	Max	Paris	1975-04-03	12000.00	1002	3	104	2015-09-01 09:00:00.0	20	Marseille	Nice	
4	Scott	Marseille	1981-08-29	12000.00	1002	3	104	2015-09-01 09:00:00.0	20	Marseille	Nice	
5	Mandy	Marseille	1985-05-16	16000.00	1002	3	104	2015-09-01 09:00:00.0	20	Marseille	Nice	
6	John	Nice	1982-06-11	20000.00	1002	3	104	2015-09-01 09:00:00.0	20	Marseille	Nice	
7	Bill	Marseille	1980-06-29	18000.00	1002	3	104	2015-09-01 09:00:00.0	20	Marseille	Nice	
8	Camille	Paris	1989-10-06	16000.00	1002	3	104	2015-09-01 09:00:00.0	20	Marseille	Nice	
9	boule	Paris	1980-06-29	17000.00	1002	3	104	2015-09-01 09:00:00.0	20	Marseille	Nice	
1	William	Nice	1980-06-29	16000.00	1003	4	105	2015-09-01 15:30:00.0	65	Paris	Nice	
2	Peter	Nice	1970-01-10	18000.00	1003	4	105	2015-09-01 15:30:00.0	65	Paris	Nice	
3	Max	Paris	1975-04-03	12000.00	1003	4	105	2015-09-01 15:30:00.0	65	Paris	Nice	
4	Scott	Marseille	1981-08-29	12000.00	1003	4	105	2015-09-01 15:30:00.0	65	Paris	Nice	
5	Mandy	Marseille	1985-05-16	16000.00	1003	4	105	2015-09-01 15:30:00.0	65	Paris	Nice	
6	John	Nice	1982-06-11	20000.00	1003	4	105	2015-09-01 15:30:00.0	65	Paris	Nice	
7	Bill	Marseille	1980-06-29	18000.00	1003	4	105	2015-09-01 15:30:00.0	65	Paris	Nice	
8	Camille	Paris	1989-10-06	16000.00	1003	4	105	2015-09-01 15:30:00.0	65	Paris	Nice	
9	boule	Paris	1980-06-29	17000.00	1003	4	105	2015-09-01 15:30:00.0	65	Paris	Nice	
1	William	Nice	1980-06-29	16000.00	1004	2	101	2015-09-01 16:10:00.0	60	Paris	Marseille	
2	Peter	Nice	1970-01-10	18000.00	1004	2	101	2015-09-01 16:10:00.0	60	Paris	Marseille	
3	Max	Paris	1975-04-03	12000.00	1004	2	101	2015-09-01 16:10:00.0	60	Paris	Marseille	



# Produit cartésien SQL - exemple 1

Listing 23 – les pilotes et les vols et selection pour rétablir la cohérence

```
select *
  from pilote
      cross join vol
 where pilote.numpil = vol.numpil
;
```

resultat	numpil	nompil	vilpil	datnaispil	salairpil	numvol	numpil	numav	departvol	dureevol	vildepvol	vilarrvol
	3	Max	Paris	1975-04-03	12000.00	1002	3	104	2015-09-01 09:00:00.0	20	Marseille	Nice
	4	Scott	Marseille	1981-08-29	12000.00	1003	4	105	2015-09-01 15:30:00.0	65	Paris	Nice
	2	Peter	Nice	1970-01-10	18000.00	1004	2	101	2015-09-01 16:10:00.0	60	Paris	Marseille
	3	Max	Paris	1975-04-03	12000.00	1005	3	101	2015-09-02 10:50:00.0	45	Marseille	Lyon
	4	Scott	Marseille	1981-08-29	12000.00	1006	4	107	2015-09-03 11:10:00.0	20	Nice	Marseille
	5	Mandy	Marseille	1985-05-16	16000.00	1007	5	101	2015-09-03 15:00:00.0	45	Paris	Lyon
	6	John	Nice	1982-06-11	20000.00	1008	6	101	2015-09-04 09:30:00.0	45	Lyon	Marseille
	7	Bill	Marseille	1980-06-29	18000.00	1009	7	101	2015-09-05 08:00:00.0	60	Paris	Marseille
	2	Peter	Nice	1970-01-10	18000.00	1010	2	108	2015-09-05 17:00:00.0	60	Nice	Paris

nombre de lignes: 9

# Produit cartésien SQL - exemple 1

Listing 24 – les pilotes et les vols et selection pour rétablir la cohérence et projection

```
select pilote.numpil, nompil, numvol
  from pilote
      cross join vol
 where pilote.numpil = vol.numpil
;
```

resultat	numpil	nompil	numvol
	3	Max	1002
	4	Scott	1003
	2	Peter	1004
	3	Max	1005
	4	Scott	1006
	5	Mandy	1007
	6	John	1008
	7	Bill	1009
	0	D	1010

# Produit cartésien SQL - exemple 1

## Listing 25 – les pilotes et les vols et les avions

```
select *
  from pilote
      cross join vol
      cross join avion
 where pilote.numpil = vol.numpil
      and vol.numav = avion.numav
;
```

resultat	numpil	nompil	vilpil	datnaispil	salairespil	numvol	numpil	numav	departvol	dureevol	vildepvol	vilarrvol	numav	nomav	localisav	capamaxav
	2	Peter	Nice	1970-01-10	18000.00	1004	2	101	2015-09-01 16:10:00.0	60	Paris	Marseille	101	A300	Marseille	300
	3	Max	Paris	1975-04-03	12000.00	1005	3	101	2015-09-02 10:50:00.0	45	Marseille	Lyon	101	A300	Marseille	300
	5	Mandy	Marseille	1985-05-16	16000.00	1007	5	101	2015-09-03 15:00:00.0	45	Paris	Lyon	101	A300	Marseille	300
	6	John	Nice	1982-06-11	20000.00	1008	6	101	2015-09-04 09:30:00.0	45	Lyon	Marseille	101	A300	Marseille	300
	7	Bill	Marseille	1980-06-29	18000.00	1009	7	101	2015-09-05 08:00:00.0	60	Paris	Marseille	101	A300	Marseille	300
	3	Max	Paris	1975-04-03	12000.00	1002	3	104	2015-09-01 09:00:00.0	20	Marseille	Nice	104	A330	Marseille	208
	4	Scott	Marseille	1981-08-29	12000.00	1003	4	105	2015-09-01 15:30:00.0	65	Paris	Nice	105	A330	Marseille	208
	4	Scott	Marseille	1981-08-29	12000.00	1006	4	107	2015-09-03 11:10:00.0	20	Nice	Marseille	107	A320	Lille	178
	2	Peter	Nice	1970-01-10	18000.00	1010	2	108	2015-09-05 17:00:00.0	60	Nice	Paris	108	A330	Paris	208

nombre de lignes: 9

# Produit cartésien SQL - exemple 1

Listing 26 – les pilotes et les vols et les avions avec selection et projection

```
select A.numpil, nompil, numvol, C.numav, nomav
from pilote A
     cross join vol B
     cross join avion C
where A.numpil = B.numpil
     and B.numav = C.numav
;
```

resultat	numpil	nompil	numvol	numav	nomav
	2	Peter	1004	101	A300
	3	Max	1005	101	A300
	5	Mandy	1007	101	A300
	6	John	1008	101	A300
	7	Bill	1009	101	A300
	3	Max	1002	104	A330
	4	Scott	1003	105	A330

# Exercice

- 1 lister musicien X instrument
- 2 lister musicien X jouer; compléter par une sélection pour obtenir un résultat cohérent
- 3 lister (musicien X jouer)Xinstrument; compléter par une sélection pour obtenir un résultat cohérent et une projection pour avoir le nom du musicien et le nom de l'instrument joué

# Jointure interne SQL

On définit les jointures et critères associés dans la clause 'from' (SQL92) grâce au mot-clef `inner join`

```
select  [distinct | all] {* | listeDeColonnes}
        from table1 inner join table2
                critereDeJointure
        [inner join table3
                critereDeJointure]
        ...
[where  criteresDeSelection]
[group by listeDeColonnesDeRegroupement]
[having criteresDeSelectionApresRegroupement]
[order by criteresDeClassement]
;
```

# Jointure SQL - exemple 1

les pilotes et leurs vols

```
select *
  from pilote
      inner join vol
      on pilote.numpil = vol.numpil
;
```

resultat	numpil	nompil	vilpil	datnaispil	salairpil	numvol	numpil	numav	departvol	dureevol	vildepvol	vilarrvol
	3	Max	Paris	1975-04-03	12000.00	1002	3	104	2015-09-01 09:00:00.0	20	Marseille	Nice
	4	Scott	Marseille	1981-08-29	12000.00	1003	4	105	2015-09-01 15:30:00.0	65	Paris	Nice
	2	Peter	Nice	1970-01-10	18000.00	1004	2	101	2015-09-01 16:10:00.0	60	Paris	Marseille
	3	Max	Paris	1975-04-03	12000.00	1005	3	101	2015-09-02 10:50:00.0	45	Marseille	Lyon
	4	Scott	Marseille	1981-08-29	12000.00	1006	4	107	2015-09-03 11:10:00.0	20	Nice	Marseille
	5	Mandy	Marseille	1985-05-16	16000.00	1007	5	101	2015-09-03 15:00:00.0	45	Paris	Lyon
	6	John	Nice	1982-06-11	20000.00	1008	6	101	2015-09-04 09:30:00.0	45	Lyon	Marseille
	7	Bill	Marseille	1980-06-29	18000.00	1009	7	101	2015-09-05 08:00:00.0	60	Paris	Marseille
	2	Peter	Nice	1970-01-10	18000.00	1010	2	108	2015-09-05 17:00:00.0	60	Nice	Paris

nombre de lignes: 9

# Jointure SQL - exemple 1 avec using

les pilotes et leurs vols

```
select *
  from pilote
      inner join vol
            using (numpil)
;
```

resultat	numpil	nompil	vilpil	datnaispil	sairepil	numvol	numav	departvol	dureevol	vildepvol	vilarrvol
	3	Max	Paris	1975-04-03	12000.00	1002	104	2015-09-01 09:00:00.0	20	Marseille	Nice
	4	Scott	Marseille	1981-08-29	12000.00	1003	105	2015-09-01 15:30:00.0	65	Paris	Nice
	2	Peter	Nice	1970-01-10	18000.00	1004	101	2015-09-01 16:10:00.0	60	Paris	Marseille
	3	Max	Paris	1975-04-03	12000.00	1005	101	2015-09-02 10:50:00.0	45	Marseille	Lyon
	4	Scott	Marseille	1981-08-29	12000.00	1006	107	2015-09-03 11:10:00.0	20	Nice	Marseille
	5	Mandy	Marseille	1985-05-16	16000.00	1007	101	2015-09-03 15:00:00.0	45	Paris	Lyon
	6	John	Nice	1982-06-11	20000.00	1008	101	2015-09-04 09:30:00.0	45	Lyon	Marseille
	7	Bill	Marseille	1980-06-29	18000.00	1009	101	2015-09-05 08:00:00.0	60	Paris	Marseille
	2	Peter	Nice	1970-01-10	18000.00	1010	108	2015-09-05 17:00:00.0	60	Nice	Paris

nombre de lignes: 9 Remarquer la suppression du doublon de colonne



# Jointure naturelle SQL - exemple 1

les pilotes vols et avions

```
select *
  from pilote
      natural join vol
      natural join avion
;
```

resultat	numav	numpil	nompil	vilpil	datnaispil	salairpil	numvol	departvol	dureevol	vildepvol	vilarrvol	nomav	localisav	capamaxav
	101	2	Peter	Nice	1970-01-10	18000.00	1004	2015-09-01 16:10:00.0	60	Paris	Marseille	A300	Marseille	300
	101	3	Max	Paris	1975-04-03	12000.00	1005	2015-09-02 10:50:00.0	45	Marseille	Lyon	A300	Marseille	300
	101	5	Mandy	Marseille	1985-05-16	16000.00	1007	2015-09-03 15:00:00.0	45	Paris	Lyon	A300	Marseille	300
	101	6	John	Nice	1982-06-11	20000.00	1008	2015-09-04 09:30:00.0	45	Lyon	Marseille	A300	Marseille	300
	101	7	Bill	Marseille	1980-06-29	18000.00	1009	2015-09-05 08:00:00.0	60	Paris	Marseille	A300	Marseille	300
	104	3	Max	Paris	1975-04-03	12000.00	1002	2015-09-01 09:00:00.0	20	Marseille	Nice	A330	Marseille	208
	105	4	Scott	Marseille	1981-08-29	12000.00	1003	2015-09-01 15:30:00.0	65	Paris	Nice	A330	Marseille	208
	107	4	Scott	Marseille	1981-08-29	12000.00	1006	2015-09-03 11:10:00.0	20	Nice	Marseille	A320	Lille	178
	108	2	Peter	Nice	1970-01-10	18000.00	1010	2015-09-05 17:00:00.0	60	Nice	Paris	A330	Paris	208

nombre de lignes: 9

Très simple, mais éviter cette notation qui s'appuie sur le fait que seules les colonnes de jointure portent le même nom dans les tables...

# Exercice

- 1 lister les noms des musiciens et les titres des concerts auxquels ils ont participé
- 2 lister les noms des musiciens qui ont participé à un concert
- 3 lister les noms des musiciens et les noms des instruments qu'ils jouent
- 4 lister les noms des musiciens qui jouent d'un instrument
- 5 lister les titres des concerts dans lesquels un guitariste à joué

## Jointure externe SQL

La jointure externe conserve toutes les lignes d'une des tables jointes. On définit les jointures et critères associés dans la clause 'from' (SQL92) grâce au mot-clef (left|right|full) outer join

```
select [distinct | all] {* | listeDeColonnes}
  from (table1 (left|right|full) outer join table2
        critereDeJointure)
       [inner (left|right|full) outer join table2
        critereDeJointure]
[where criteresDeSelection]
[group by listeDeColonnesDeRegroupement]
[having criteresDeSelectionApresRegroupement]
[order by criteresDeClassement]
;
```

# Jointure SQL - exemple 1

tous les pilotes et éventuellement leurs vols

```
select *
  from pilote
      left outer join vol
        on pilote.numpil = vol.numpil
;
```

resultat	numpil	nompil	vilpil	datnaispil	salairepil	numvol	numpil	numav	departvol	dureevol	vildepvol	vilarrvol
1	William	Nice	1980-06-29	16000.00	null	null	null	null	null	null	null	null
2	Peter	Nice	1970-01-10	18000.00	1004	2	101	2015-09-01 16:10:00.0	60	Paris	Marseille	
2	Peter	Nice	1970-01-10	18000.00	1010	2	108	2015-09-05 17:00:00.0	60	Nice	Paris	
3	Max	Paris	1975-04-03	12000.00	1002	3	104	2015-09-01 09:00:00.0	20	Marseille	Nice	
3	Max	Paris	1975-04-03	12000.00	1005	3	101	2015-09-02 10:50:00.0	45	Marseille	Lyon	
4	Scott	Marseille	1981-08-29	12000.00	1003	4	105	2015-09-01 15:30:00.0	65	Paris	Nice	
4	Scott	Marseille	1981-08-29	12000.00	1006	4	107	2015-09-03 11:10:00.0	20	Nice	Marseille	
5	Mandy	Marseille	1985-05-16	16000.00	1007	5	101	2015-09-03 15:00:00.0	45	Paris	Lyon	
6	John	Nice	1982-06-11	20000.00	1008	6	101	2015-09-04 09:30:00.0	45	Lyon	Marseille	
7	Bill	Marseille	1980-06-29	18000.00	1009	7	101	2015-09-05 08:00:00.0	60	Paris	Marseille	
8	Camille	Paris	1989-10-06	16000.00	null	null	null	null	null	null	null	null
9	boule	Paris	1980-06-29	17000.00	null	null	null	null	null	null	null	null

nombre de lignes: 12

# Jointure SQL - exemple 1 avec using

tous les pilotes et éventuellement leurs vols

```
select *
  from pilote
      left outer join vol
        using (numpil)
;
```

resultat	numpil	nompil	vilpil	datnaispil	salairepil	numvol	numav	departvol	dureevol	vildepvol	vilarrvol
	1	William	Nice	1980-06-29	16000.00	null	null	null	null	null	null
	2	Peter	Nice	1970-01-10	18000.00	1004	101	2015-09-01 16:10:00.0	60	Paris	Marseille
	2	Peter	Nice	1970-01-10	18000.00	1010	108	2015-09-05 17:00:00.0	60	Nice	Paris
	3	Max	Paris	1975-04-03	12000.00	1002	104	2015-09-01 09:00:00.0	20	Marseille	Nice
	3	Max	Paris	1975-04-03	12000.00	1005	101	2015-09-02 10:50:00.0	45	Marseille	Lyon
	4	Scott	Marseille	1981-08-29	12000.00	1003	105	2015-09-01 15:30:00.0	65	Paris	Nice
	4	Scott	Marseille	1981-08-29	12000.00	1006	107	2015-09-03 11:10:00.0	20	Nice	Marseille
	5	Mandy	Marseille	1985-05-16	16000.00	1007	101	2015-09-03 15:00:00.0	45	Paris	Lyon
	6	John	Nice	1982-06-11	20000.00	1008	101	2015-09-04 09:30:00.0	45	Lyon	Marseille
	7	Bill	Marseille	1980-06-29	18000.00	1009	101	2015-09-05 08:00:00.0	60	Paris	Marseille
	8	Camille	Paris	1989-10-06	16000.00	null	null	null	null	null	null
	9	boule	Paris	1980-06-29	17000.00	null	null	null	null	null	null

nombre de lignes: 12

# Jointure SQL - exemple 1 avec using

tous les pilotes et éventuellement leurs vols

```
select *
  from vol
      right outer join pilote
        using (numpil)
;
```

resultat	numpil	nompil	vilpil	datnaispil	salairepil	numvol	numav	departvol	dureevol	vildepvol	vilarrvol
	1	William	Nice	1980-06-29	16000.00	null	null	null	null	null	null
	2	Peter	Nice	1970-01-10	18000.00	1004	101	2015-09-01 16:10:00.0	60	Paris	Marseille
	2	Peter	Nice	1970-01-10	18000.00	1010	108	2015-09-05 17:00:00.0	60	Nice	Paris
	3	Max	Paris	1975-04-03	12000.00	1002	104	2015-09-01 09:00:00.0	20	Marseille	Nice
	3	Max	Paris	1975-04-03	12000.00	1005	101	2015-09-02 10:50:00.0	45	Marseille	Lyon
	4	Scott	Marseille	1981-08-29	12000.00	1003	105	2015-09-01 15:30:00.0	65	Paris	Nice
	4	Scott	Marseille	1981-08-29	12000.00	1006	107	2015-09-03 11:10:00.0	20	Nice	Marseille
	5	Mandy	Marseille	1985-05-16	16000.00	1007	101	2015-09-03 15:00:00.0	45	Paris	Lyon
	6	John	Nice	1982-06-11	20000.00	1008	101	2015-09-04 09:30:00.0	45	Lyon	Marseille
	7	Bill	Marseille	1980-06-29	18000.00	1009	101	2015-09-05 08:00:00.0	60	Paris	Marseille
	8	Camille	Paris	1989-10-06	16000.00	null	null	null	null	null	null
	9	boule	Paris	1980-06-29	17000.00	null	null	null	null	null	null

nombre de lignes: 12

# Exercice

- 1 lister les noms de tous les musiciens et éventuellement les titres des concerts auxquels ils ont participé
- 2 lister les noms de tous instruments et éventuellement les noms des musiciens qui en jouent

# Agrégats SQL

```
select  [a1, a2, ..., an]
        fn1(x1)[, fn2(x2), ..., fnN(xn)]
from    table(s)
[where  Q1]
[group by a1, a2, ..., an]
[having Q2]
[order by ...]
;
```

où :

- $fn1, fn2, \dots, fnN$  sont des fonctions d'agrégation appliquées aux colonnes  $x1, x2, \dots, xn$
- `group by a1, a2, ..., an` : mot-clef SQL précisant les colonnes de regroupement
- `having Q2` : mot-clef SQL précisant la condition de sélection s'appliquant généralement à des valeurs agrégées



# Agrégats SQL : fonctions

count	compter les valeurs d'une colonne
sum	calculer la somme des valeurs d'une colonne
avg	calculer la moyenne des valeurs d'une colonne
min	déterminer la plus petite des valeurs d'une colonne
max	déterminer la plus grande des valeurs d'une colonne

Ces

fonctions ne comptabilisent pas les valeurs nulles.

# Agrégats SQL : fonctions

Le mot-clef SQL `distinct` peut être utilisé en complément pour appliquer la fonction sur des valeurs distinctes des colonnes :

<code>count(distinct c1)</code>	compter les valeurs distinctes de c1
<code>sum(distinct c1)</code>	calculer la somme des valeurs distinctes de c1
<code>avg(distinct c1)</code>	calculer la moyenne des valeurs distinctes de c1

# Agrégat global SQL

```
select fn1(x1)[, fn2(x2), ..., fnN(xn)]  
  from table(s)  
  [where Q1]  
;
```

où :

- $fn1, fn2, \dots, fnN$  sont des fonctions d'agrégation appliquées aux colonnes  $x1, x2, \dots, xn$

# Agrégat global SQL

Le nombre total de pilotes et la somme de leurs salaires

```
select count(*) as NbPilotes, sum(salairepil) as  
    SommeSalaires  
from pilote  
;
```

resultat	NbPilotes	SommeSalaires
	9	145000.00

nombre de lignes: 1 Détail :

```
select numpil, salairepil  
from pilote  
;
```

resultat	numpil	salairepil
	1	16000.00
	2	18000.00
	3	12000.00
	4	12000.00

# Agrégat SQL : attention aux valeurs nulles

Le nombre de personnes : \* = comptage des lignes

```
select count(*) as Nb1, count(numero) as Nb2, count(ville)
  as Nb3
  from personne
;
```

resultat	Nb1	Nb2	Nb3
	5	5	4

nombre de lignes: 1

## Agrégat global SQL : attention

Le nombre total de pilotes qui ont volé et la somme de leurs salaires et des durées de vol : *résultat erroné* : les lignes et salaire sont comptés plusieurs fois...

```
select count(*) as NbPilotes, sum(salairepil) as
    SommeSalaires, sum(dureevol) as totalDureeVol
from pilote
    inner join vol using (numpil)
;
```

resultat	NbPilotes	SommeSalaires	totalDureeVol
	9	138000.00	420

nombre de lignes: 1

## Agrégat global SQL : attention

Détail : (l'agrégat est calculé sur les lignes récupérées par la clause from)

```
select numpil, salairepil, dureeVol
  from pilote
       inner join vol using (numpil)
 order by numpil
;
```

resultat	numpil	salairepil	dureeVol
	2	18000.00	60
	2	18000.00	60
	3	12000.00	20
	3	12000.00	45
	4	12000.00	65
	4	12000.00	20
	5	16000.00	45
	6	20000.00	45
	7	18000.00	60

# Exercice

- 1 lister le nombre de concerts
- 2 lister le nombre de concerts ayant lieu à Arras
- 3 lister le nombre de villes de résidence
- 4 lister le nombre de villes de résidence différentes
- 5 lister la moyenne des cachets
- 6 lister la moyenne des cachets différents



## Agrégats par regroupement de valeurs SQL

```
select  [a1, a2, ..., an]
        fn1(x1), fn2(x2), ..., fnN(xn)
from    table(s)
[where  Q1]
group  by a1, a2, ..., an
[having Q2]
[order by ...]
;
```

où :

- $fn1, fn2, \dots, fnN$  sont des fonctions d'agrégation appliquées aux colonnes  $x1, x2, \dots, xn$
- `group by a1, a2, ..., an` : mot-clef SQL précisant les colonnes de regroupement
- `having Q2` : mot-clef SQL précisant la condition de sélection s'appliquant généralement à des valeurs agrégées

# Agrégat par valeur de regroupement SQL

## Listing 27 – par ville

```
select ville, sum(salaire) as totalDesSalaires,
       count(numero) as nombrePersonnes,
       count(ville) as nombreVilles,
       count(distinct ville) as villesDiff
from personne
group by ville
;
```

resultat	ville	totalDesSalaires	nombrePersonnes	nombreVilles	villesDiff
	null	1300.00	1	0	0
	Aix	1500.00	1	1	1
	arras	2250.00	2	2	1
	Pau	1350.00	1	1	1

nombre de lignes: 4

On peut voir ci-dessus que les comptages sont différents : en effet une ligne a sa colonne vide à la valeur null (non renseignée).

# Exercice

- 1 lister le nombre de concerts par ville
- 2 lister la moyenne des cachets par numéro de musicien
- 3 lister la moyenne des cachets par nom de musicien

## Sélection après agrégat

```
select  [a1, a2, ..., an]
        fn1(x1)[, fn2(x2), ..., fnN(xn)]
from table(s)
[where Q1]
group by a1, a2, ..., an
having Q2
[order by ...]
;
```

où :

- **having Q2** : mot-clef SQL introduisant la condition de sélection s'appliquant à des valeurs agrégées

## Sélection après agrégat

par ville, somme des salaires, en ne conservant que les villes qui ont plus de 2000 en somme de salaire

```
select ville, sum(salaire) as totalDesSalaires
  from personne
  group by ville
  having totalDesSalaires > 2000
;
```

resultat	ville	totalDesSalaires
	arras	2250.00

nombre de lignes: 1

# Exercice

- 1 lister le nombre de concerts par ville en ne conservant que les villes ayant plus de 1 concert
- 2 lister la moyenne des cachets par numéro de musicien en ne conservant que les musiciens ayant plus de 2000 en cachet
- 3 lister la moyenne des cachets par nom de musicien en ne conservant que les musiciens ayant moins de 2000 en cachet

# SQL

Les valeurs retournées par une requête correspondent généralement à celles des colonnes des tables. Elles peuvent être également provenir

- de calculs arithmétiques classiques
- ou de fonctions spécifiques

pour construire des réponses adaptées aux requêtes complexes.

Les valeurs ainsi calculées peuvent être utilisées

- comme valeurs renvoyées dans la liste des colonnes retournées (`select`)
- ou dans des expressions logiques pour limiter le nombre de lignes retournées (`where`)

# Opérateurs SQL

<i>opérateur</i>	<i>signification</i>
+	somme
-	différence
*	produit
/	rapport
%	modulo (reste de la division euclidienne, ou entière)



# Opérateurs SQL

Listing 28 – liste des commandes avec calcul du montant et de l'écart de valeur

```
select idComm, (qteComm * prixComm) as montant,
        (qteComm * (prixComm - prix)) as ecart
from commande
        inner join produit
        using (refProduit)
;
```

resultat	idComm	montant	ecart
	1	15.00	5.00
	2	19.00	6.50
	3	105.60	93.60
	4	100.00	80.00

nombre de lignes: 4

# Fonctions intégrées SQL

- fonctions mathématiques : abs, power, sqrt, rand, pi, cos, sin, acos, log, etc.
- fonctions de date : day, month, year, date\_diff, date\_add, now, etc.
- fonctions de chaînes : concat, lower, upper, right, left, substr, trim, char\_length, reverse, soundex, etc.
- fonctions de chaînes : cas, coalesce, etc.

# Opérateurs SQL

Listing 29 – calcul de l'aire et de la circonférence d'un disque

```
select 5 as rayon, (pi()*pow(5,2)) as aireDisque ,  
       (2*pi()*5) as circonference  
from dual  
;
```

resultat	rayon	aireDisque	circonference
	5	78.53981633974483	31.415927

nombre de lignes: 1

'dual' (MySQL) est une pseudo-table utilisable pour effectuer des calculs sans pour autant avoir de données

# Opérateurs SQL

Listing 30 – les membres en mieux bien présentés...

```
select upper(nom_memb) as NOM,  
       concat(upper(left(prenom_memb,1)),lower(right(  
           prenom_memb,(length(prenom_memb)-1)))) as Prenom  
       ,  
       upper(concat(left(nom_memb,1), left(prenom_memb,1)))  
       as initiales  
from membre  
;
```

resultat	NOM	Prenom	initiales
	DUPONT	Pierre	DP
	LAJOIE	Caroline	LC
	DURANT	Jacques	DJ
	DURAND	Fred	DF
	LAMBERT	Annie	LA
	DURANT	Paul	DP

nombre de lignes: 6

# Opérateurs SQL

Listing 31 – les personne et leur ville

```
select nom, prenom, coalesce(ville, "** absent **") as ville
from personne
;
```

resultat	nom	prenom	ville
	Dupont	Max	arras
	Durand	Tim	Aix
	Lambert	Betty	Pau
	Bradford	Jean	Arras
	Rigole	Jean	** absent **

nombre de lignes: 5

Listing 32 – date et heure du jour

```
select now(), current_date, current_time
from dual
;
```

# Expression conditionnelle SQL : choix multiple

## Syntaxe case : choix de valeur

```
case valeur
  when valeurComparaison THEN valeurResultat
  [when valeurComparaison THEN valeurResultat] ...
  [else valeurResultat]
end
```

ou

```
case
  when condition THEN valeurResultat
  [when condition THEN valeurResultat] ...
  [else valeurResultat]
end
```

# Opérateurs SQL

```
select nom, prenom,  
       case sexe  
         when 'f' then 'femme'  
         when 'h' then 'homme'  
         else 'inconnu'  
       end as sexe  
from personne  
;
```

resultat	nom	prenom	sexe
	Dupont	Max	homme
	Durand	Tim	homme
	Lambert	Betty	femme
	Bradford	Jean	homme
	Rigole	Jean	homme

nombre de lignes: 5

# Exercice

- 1 lister les titre, jour de la semaine, jour, mois et année des concerts
- 2 lister les concerts ayant lieu en mai (de n'importe quelle année)
- 3 lister les concerts et le nombre de jours restant avant leur date
- 4 lister, par numéro de concert, la somme des cachets convertis en USD (1EUR = 1.2 USD) , arrondir le calcul (pas de décimale)



## Sous-requêtes SQL

Une *sous-requête* (en anglais : *subquery*), ou *requête interne* (en anglais : *inner subquery*), ou encore requête imbriquée (en anglais : *nested subquery*), est une requête utilisée pour constituer un résultat qui va être utilisé dans une *requête principale*, ou *requête externe* (en anglais : *outer query*), de la manière suivante :

- en général comme élément de comparaison dans les clauses `where` ou `having` (dans les ordres DML et DQL),
- parfois comme valeur d'une colonne dans la clause `select`.
- exceptionnellement en remplacement d'une table dans la clause `from` (SQL92).

## Sous-requêtes indépendantes ou corrélées

Deux formes de sous-requêtes sont définies, en fonction de leur lien avec la requête principale :

- sous-requêtes *indépendantes*, sans aucun lien avec la requête appelante
  - elle peut être lancée seule, elle est totalement indépendante de la requête appelante; son évaluation est effectuée une seule fois et le jeu de résultat qu'elle renvoie va servir à la requête appelante comme valeur unique, ou liste de valeurs .
- sous-requêtes *corrélées*, ou dépendantes ou liées, qui ont un lien avec la requête appelante
  - son exécution va dépendre, cette fois, de valeurs de colonnes de la requête principale, elle est liée à la requête principale, elle en est dépendante, corrélée (en relation avec); elle est exécutée pour chaque ligne de la requête appelante (coût très élevé !).

## Sous-requêtes appliquée à la différence

$(R - S)$

```
select  col1, col2, col3, ...colN
  from  R
 where  (col1, col2, col3, ...colN) not in
 (
   select colS1, colS2, colS3, ...colSN
     from  S
 )
;
```

où :

- $R$  et  $S$  : tables objets de la différence
- $col1, col2, col3, ...colN$  : liste des attributs format une ligne de  $R$  qu'on ne doit pas trouver dans  $S$  (ligne de  $S$  formée par  $colS1, colS2, colS3, ...colSN$ )

On pourra souvent simplifier en utilisant seulement des valeurs de clef primaire au lieu de tous les attributs.

## Sous-requêtes appliquée à la différence

Exemple : les pilotes qui n'ont pas volé

- 1 Sous-requête : les numéros des pilotes qui ont volé

```
select distinct numpil
  from vol
;
```

resultat	numpil
	3
	4
	2
	5
	6
	7

nombre de lignes: 6

Cette sous-requête, indépendante (qu'on peut lancer seule), retourne comme résultat une liste de numéro de pilotes qui ont volé.

# Sous-requêtes appliquée à la différence

Exemple : les pilotes qui n'ont pas volé

## 1 Requête complète

```
select  numpil
  from  pilote
 where  numpil not in
        (select  numpil
         from  vol)
;
```

resultat	numpil
	1
	8
	9

nombre de lignes: 3

soit : les pilotes dont le numéro n'est pas dans la liste de ceux qui ont volé

# Exercice

- 1 lister les noms des musiciens qui n'ont pas été programmés dans un concert
- 2 lister les noms des instruments qui ne sont pas joués par les musiciens
- 3 lister les noms des instruments qui n'ont pas été joué dans un concert (on peut considérer que si un musicien joue d'un instrument, il en joue en concert)

## Sous-requêtes appliquée à l'intersection

$(R \cap S)$

```
select col1, col2, col3, ...colN
  from R
 where (col1, col2, col3, ...colN) in
 (
   select colS1, colS2, colS3, ...colSN
     from S
 )
;
```

où :

- $R$  et  $S$  : tables objets de l'intersection
- $col1, col2, col3, \dots colN$  : liste des attributs format une ligne de  $R$  qu'on doit trouver dans  $S$  (ligne de  $S$  formée par  $colS1, colS2, colS3, \dots colSN$ )

On pourra souvent simplifier en utilisant seulement des valeurs de clef primaire au lieu de tous les attributs.

# Sous-requêtes appliquée à l'intersection

Exemple : les pilotes qui ont volé

- 1 Sous-requête : les numéros des pilotes qui ont volé

```
select distinct numpil  
  from vol  
;
```

resultat	numpil
	3
	4
	2
	5
	6
	7

nombre de lignes: 6



# Sous-requêtes appliquée à l'intersection

Exemple : les pilotes qui ont volé

## 1 Requête complète

```
select  numpil
  from  pilote
 where  numpil in
        (select  numpil
         from  vol)
;
```

resultat	numpil
	2
	3
	4
	5
	6
	7

nombre de lignes: 6

# Exercice

- 1 lister les noms des musiciens qui ont été programmés dans un concert
- 2 lister les noms des instruments qui sont joués par les musiciens
- 3 lister les noms des instruments qui ont été joué dans un concert (on peut considérer que si un musicien joue d'un instrument, il en joue en concert)

## Sous-requêtes appliquée à la division

Exemple : les membres qui se sont inscrits à toutes les activités  
Quels sont les membres tels qu'il n'existe pas d'activités pour lesquelles ces membres ne s'y soient pas inscrits ?

Listing 33 – Division avec double négation

```
select id_memb, nom_memb, prenom_memb
  from membre
 where not exists
    (select * from activite
     where not exists
        (select *
         from inscrire
         where id_memb = membre.id_memb
              and id_activ = activite.id_activ
        )
     )
;
```

resultat	id_memb	nom_memb	prenom_memb
----------	---------	----------	-------------

## Sous-requêtes appliquée à la division

Exemple : les membres qui se sont inscrits à toutes les activités  
Quels sont les membres pour lesquelles le nombre d'activités différentes auxquels ils ont participé est égal au nombre total d'activités différentes ?  
(attention : on ne pourra pas toujours utiliser les comptages car il est possible de compter des choses différentes !)

### Listing 34 – Division avec un agrégat

```
select m.id_memb, nom_memb, prenom_memb
  from membre m
       inner join inscrire p
            on m.id_memb = p.id_memb
 group by id_memb, nom_memb, prenom_memb
 having count(distinct id_activ) =
        (select count(distinct id_activ)
         from activite)
;
```

resultat	id_memb	nom_memb	prenom_memb
	1	dumont	nierrF

## Sous-requêtes appliquée à la division

Exemple : les membres qui se sont inscrits à toutes les activités  
Quels sont les membres pour lesquelles le nombre d'activités différentes auxquels ils ont participé est égal au nombre total d'activités différentes ?  
(idem. précédent : on ne pourra pas toujours utiliser les comptages...)

Listing 35 – Division avec plusieurs sous-requêtes

```
select m.id_memb, nom_memb, prenom_memb
  from membre m
  where (select count(distinct id_activ)
         from inscrire
         where inscrire.id_memb = m.id_memb
        )
        = (select count(distinct id_activ)
           from activite)
;
```

resultat	id_memb	nom_memb	prenom_memb
	1	dupont	pierrE

nombre de lignes: 1

# Exercice

- 1 lister les noms des musiciens qui jouent de tous les instruments
- 2 lister les noms des musiciens qui sont programmés à tous les concerts

# Exercice

- 1 opérateur de comparaison + `all` (listeDeValeurs) : vrai si l'opérateur s'applique à toutes les valeurs de la liste
- 2 opérateur de comparaison + `any` (listeDeValeurs) : vrai si l'opérateur s'applique à au moins une valeur de la liste

# Compléments

- 1 performance : organisation des index
- 2 sécurité : transactions
- 3 confidentialité : utilisateurs et privilèges
- 4 intégrité : déclencheurs
- 5 bases de données et législation
- 6 grammaire BNF (Backus-Naur form)



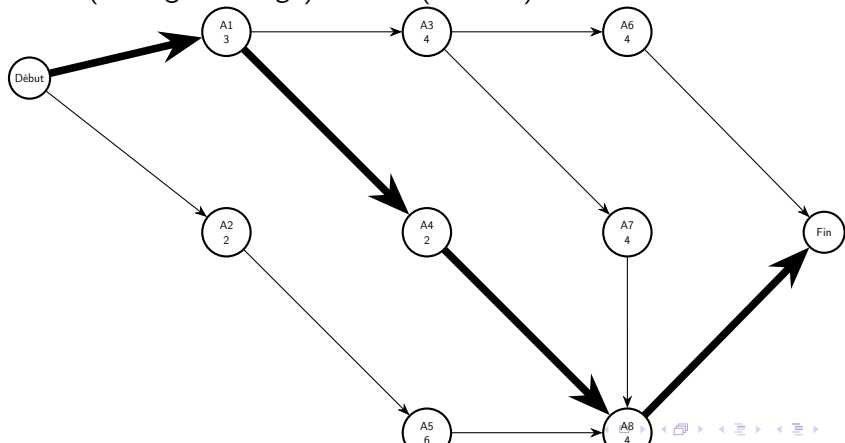
# Index

Les *index* sont des tables "système" gérées par le SGBDR pour ses besoins propres : intégrité (clefs primaires, colonnes à valeur unique), performance (recherche par nom).

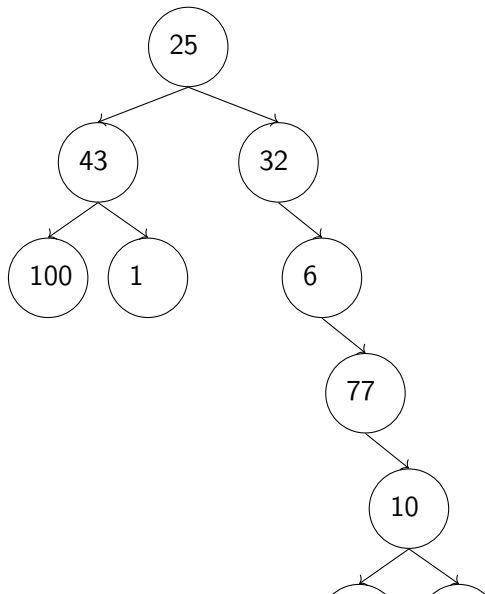
Ils sont implantés sous forme d'arbres, graphes de structure hiérarchique.

# Graphes

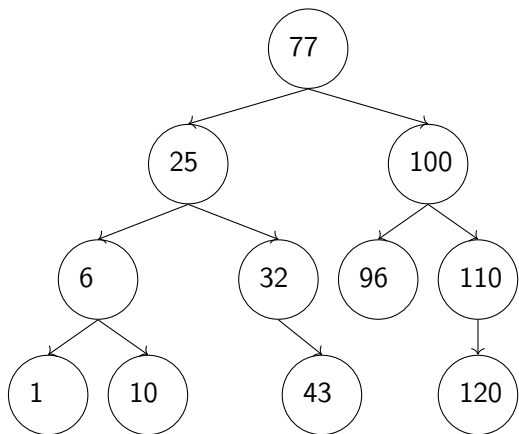
ensemble de points dont certains sont reliés 2 à 2. Les liaisons entre ces points peuvent être orientées ou non. Les points sont appelés sommets (en anglais : *vertice*) ou noeuds (en anglais : *nodes*), les liens sont appelés arêtes (en anglais : *edge*) ou arcs (orientés).



## Arbres

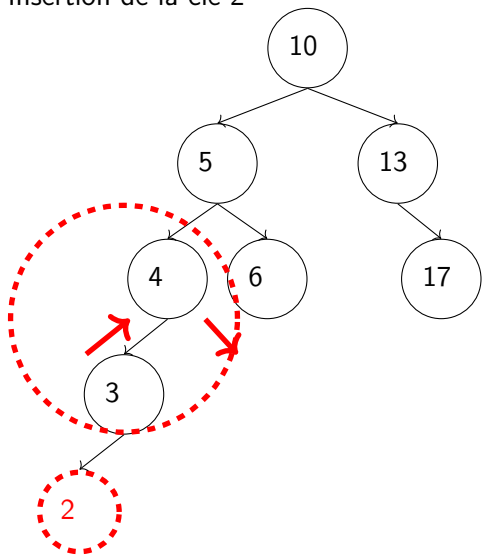


# Arbres équilibrés



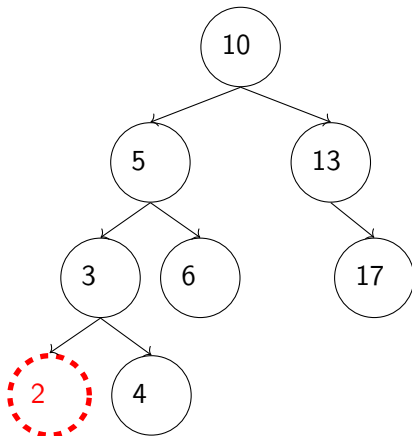
# Arbres équilibrés

Insertion de la clé 2



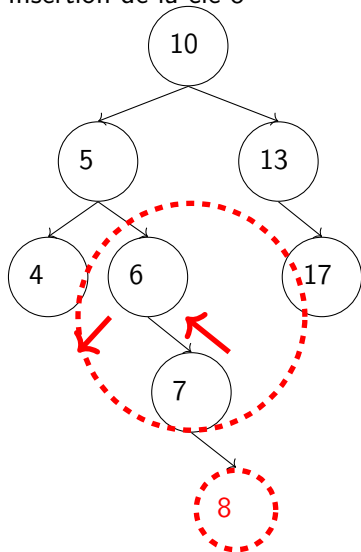
# Arbres équilibrés

Après insertion de la clé 2



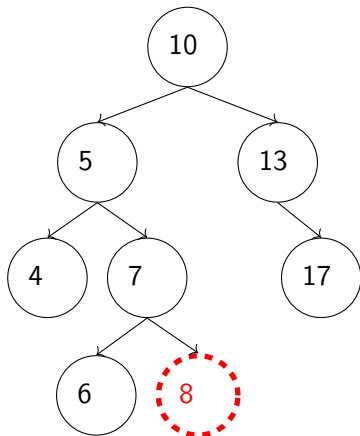
# Arbres équilibrés

Insertion de la clé 8



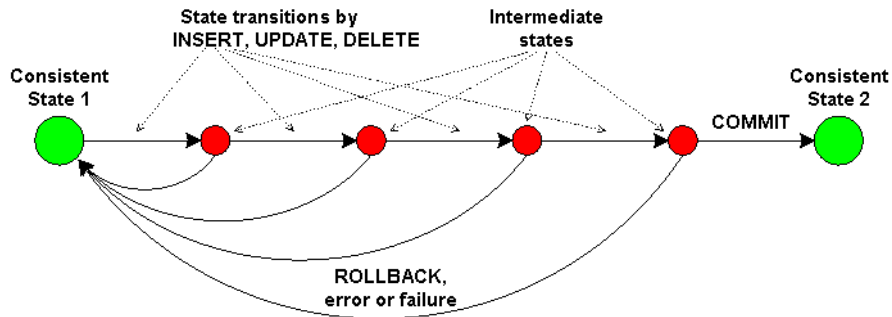
# Arbres équilibrés

Après insertion de la clé 8





# Transaction et journaux



# Transaction et journaux avec MySQL

Démarrer une transaction

```
start transaction;
```

Valider les mises à jour effectuées :

```
commit [transaction];
```

Annuler les mises à jour effectuées :

```
rollback [transaction];
```

# Utilisateurs et privilèges

L'accès aux services d'un SGBD nécessite un compte d'utilisateur.

```
CREATE USER 'toto'@'localhost' IDENTIFIED BY 'password';
```

'localhost' car l'accès ne peut être réalisé que par une application locale. Celui-ci est associé à des privilèges liées à certaines tables (et même certaines colonnes) et à certains ordres SQL, etc.

```
GRANT SELECT ON maBase.maTable to 'toto'@'localhost';  
GRANT ALL ON maBase.* to 'toto'@'localhost';
```

# Vues

Une vue (en anglais : *view*) est une requête pré-enregistrée dont le résultat peut être utilisé comme une table dans une requête. Le contenu de la vue est actualisé à chaque utilisation dans une requête.

L'utilisation d'une vue présente 2 avantages :

- elle est plus performante que la même requête soumise au SGBD car elle a déjà été vérifiée et optimisée
- elle permet la gestion plus fine des droits d'accès aux données en donnant des droits sur une vue (certaines colonnes) plutôt que sur une table complète (toutes ses colonnes)

# Vues

Pour créer une vue (ou la remplacer si elle existe déjà) :

```
CREATE OR REPLACE VIEW nomVue
AS
requeteSQL;
```

où :

- *nomVue* : le nom donné à la vue
- *requeteSQL* : toute requête SQL valide (il existe certaines restrictions dépendant des versions du SGBD)

Exemple :

```
CREATE OR REPLACE VIEW lesSalaries
AS
SELECT matricule, nom, prenom
FROM salarie;
```

# Vues

Utilisation de la vue 'lesSalaries' :

```
SELECT *  
  FROM lesSalaries  
 WHERE matricule < 1000  
 ORDER BY nom, prenom;
```

En MySQL, il est possible d'utiliser les ordres DML sur des vues à condition que le résultat soit en concordance avec le contenu de la table

## Intégrité : vérifications spécifiques

Les SGBD offrent des possibilités d'extension au déjà puissant langage SQL. Ils mettent à disposition un langage de programmation et différents objets qui permettent de l'utiliser.

Les procédures stockées sont des jeux de commandes stockés en tant qu'objets dans une base de données :

- Les procédures et fonctions : de la même manière que tout langage programmation, on peut écrire des sous-programmes pour effectuer des actions trop complexes pour être réalisées en SQL,
- Les déclencheurs : mécanisme qui permet l'exécution d'un bloc d'instructions lorsqu'un évènement de modification du contenu d'une table se produit.

Le code de ces procédures est pré-compilé.

# Langage procédural

Comme tout langage procédural, le langage intégré au SGBD utilise :

- des déclarations de variables,
- des types de données : types de base du SGBD, types spécifiques pour manipuler les lignes des tables, etc.,
- des opérateurs d'affectation de valeurs,
- des opérateurs de calcul d'expressions ,
- des instructions de parcours du résultat d'une requête (curseur)
- des structures de contrôles conditionnelles et répétitives,
- une gestion d'exception avec des instructions permettant de gérer les erreurs.



## Procédure stockée et curseur

Exemple : parcourir 'membre' et recopier les numéros dans 'listeMembre'

```
DELIMITER $$
CREATE PROCEDURE recopier()
CONTAINS SQL
BEGIN
  DECLARE done INT DEFAULT 0;
  DECLARE n INT;
  DECLARE curs1 CURSOR FOR SELECT id_memb FROM assocArt.
    membre;
  -- gestion de la fin du jeu de lignes
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

  OPEN curs1;

  lecture: LOOP
    FETCH curs1 INTO n;
    IF done THEN
      LEAVE lecture; -- sortie en fin de 'set'
    END IF;
    INSERT INTO listeMembre VALUES (n);
  END LOOP;
END;
```

# Procédure stockée et curseur

Exécution par :

```
call recopier;
```

Pour lister les procédures :

```
select name, db from mysql.proc;
```

## Procédure stockée et curseur

```
CREATE FUNCTION aireDisque (rayon REAL)
  RETURNS REAL
  DETERMINISTIC
  RETURN rayon*rayon*3.1459;
```

Exécution dans une requête SQL :

```
SELECT aireDisque(1);
```

Résultat :

```
+-----+
| aireDisque(1) |
+-----+
|           3.1459 |
+-----+
```

## Intégrité : vérifications spécifiques

Les contrôles d'intégrité sont assurés grâce :

- aux clefs primaires : intégrité d'entité (unicité)
- aux clefs étrangères : intégrité référentielle
- aux index : unicité de colonnes non clefs primaires

Mais des contraintes ne sont pas prises en compte :

- vérifications de valeurs comme la clé d'un RIB
- association 1,n avec n fixé à une valeur constante (1 étudiant peut s'inscrire à de 1 à 3 options)
- contraintes sur association (un étudiant qui passe un examen doit d'abord avoir validé son inscription)

Le mécanisme des triggers (déclencheur) permet la prise en compte de ces cas.

Un trigger est un bloc de code déclenché automatiquement suite l'insertion, la modification ou la suppression d'une ligne dans une table.

## Intégrité : vérifications spécifiques

Sur un ordre 'insert' appliqué à la table 'inscrire' :

```
DROP TRIGGER IF EXISTS surInsertionInscrire;
delimiter $$
CREATE TRIGGER surInsertionInscrire
  BEFORE INSERT ON inscrire
  FOR EACH ROW
  BEGIN
    DECLARE ct INT;
    SET ct = 0;

    if NEW.acompte NOT BETWEEN 50 AND 100 then
      SIGNAL SQLSTATE '45001' SET MESSAGE_TEXT = "acompte
        obligatoire entre 50 et 100";
    end if;

    SELECT COUNT(idForm) INTO ct FROM inscrire WHERE
      inscrire.idEtu = NEW.idEtu;
    if ct >= 3 then
      SIGNAL SQLSTATE '45001' SET MESSAGE_TEXT = "maximum d'
```

## Le droit lié aux bases de données

Les bases de données comme support d'informations stratégiques sont protégées par le droit pour les producteurs de ces bases (droit d'auteur). Ces droits leur permettent d'amortir les investissements nécessaires à leur réalisation.

Les bases de données peuvent contenir des informations susceptibles d'être protégées, par exemple, les données personnelles. Elles sont donc soumises, en autres, à la loi *informatique, fichiers et libertés de 1978*. En France, la CNIL, Commission Nationale Informatique et Liberté, est l'organisme chargé de la protection des données personnelle.