Bases de données
SQL

Dr. Rania OTHMAN

rania.othman@intervenants.efrei.net

01

Langage SQL

Base de données: aeroport

Schéma de la base de données aeroport:

pilotes(PLNUM, PLNOM , PLPRENOM, VILLE , SALAIRE)
avions (AVNUM, AVNOM , CAPACITE, LOCALISATION)
vols(VOLNUM, #*PLNUM*, #*AVNUM*, VILLEDEP, VILLEARR, HEUREDEP, HEUREARR)

Installation de MySQL Workbench

1- Installer :
https://dev.mysql.com/downloads/installer/
2- Workbench
https://dev.mysql.com/downloads/workbench/

SQL - Langage de description de données LDD

LDD : Présentation

- Sous-ensemble du langage SQL
- Manipuler les structures de données et non les données elles-mêmes
- Permet de :
  - définir le domaine des données : ensemble des valeurs que peut prendre une donnée
  - Regrouper les données ayant un lien conceptuel au sein d'une même entité
  - Définir les liens entre plusieurs entités
  - Ajouter des contraintes de valeur sur les données

Les types SQL (Norme ANSI)

Types numériques

- Types exactes : INTEGER (SMALLINT, BIGINT) et DECIMAL(NUMERIC) : spécifier la précision souhaitée pour un attribut numérique
- Types flottants : FLOAT, DOUBLE (REAL): présenter une valeur avec une précision limitée

Caractères et chaines de caractères

- CHAR et VARCHAR : stocker des chaines de caractères d'une taille maximale fixée
- La différence essentielle entre les deux types est qu'une valeur CHAR a une taille fixée, et se trouve donc complétée avec des blancs si sa taille est inférieure à **celle précisée**. En revanche une valeur VARCHAR a une taille variable et est tronquée après le dernier caractère non blanc.
- La norme SQL propose un type BIT VARYING qui correspond à de très longues chaînes de caractères. Souvent les systèmes proposent des variantes de ce type sous le nom BLOB (pour Binary Long Object) ou LONG .

Dates

- Un attribut DATE stocke les informations jour, mois et année (sur 4 chiffre)
- La représentation interne n'est pas spécifiée par la norme. Tous les systèmes proposent de nombreuses opérations de conversion (non normalisées) qui permettent d'obtenir un format d'affichage quelconque.
- Un attribut de type TIME stocke les informations heure, minute et seconde
- L'affichage se fait par défaut au format HH:MM:SS.
- Le type DATETIME permet de combiner une date et un horaire suivant la forme AAAA-MM-JJ HH:MM:SS

Les types SQL (Norme ANSI)

Type	Description	Taille
`INTEGER`	Type des entiers relatifs	4 octets
`SMALLINT`	Idem.	2 octets
`BIGINT`	Idem.	8 octets
`FLOAT`	Flottants simple précision	4 octets
`DOUBLE PRECISION`	Flottants double précision	8 octets
`REAL`	Synonyme de `FLOAT`	4 octets
`NUMERIC` (*M*, *D*)	Numérique avec précision fixe.	*M* octets
`DECIMAL` (*M*, *D*)	Idem.	*M* octets
`CHAR`(*M*)	Chaînes de longueur fixe	*M* octets
`VARCHAR`(*M*)	Chaînes de longueur variable	*L+1* avec $L \leqslant M$
`BIT VARYING`	Chaînes d'octets	Longueur de la chaîne.
`DATE`	Date (jour, mois, an)	env. 4 octets
`TIME`	Horaire (heure, minutes, secondes)	env. 4 octets
`DATETIME`	Date et heure	8 octets
`YEAR`	Année	2 octets

Création des tables : CREATE TABLE

```
create table table_1(
    att1 type_1 option_1,..., option_n,
    att2 type_2 option_1, ..., option_n,
    ...);
```

Exemple : Internaute

```
create table internaute(
    email varchar(50) not null,
    nom varchar(50) not null,
    prenom varchar(50) not null
    mdp varchar(50) not null,
    anne_naissance int);
```

- NOT NULL indique que l'attribut correspondant doit toujours avoir une valeur
  - On ne peut pas faire d'opération incluant NULL
  - On ne peut pas faire de comparaison avec un NULL
  - Le SGBD rejette toute tentative d'insérer une ligne dans la table sans donner une valeur pour un attribut NOT NULL
  - Si les valeurs à NULL sont autorisées, il faudra en tenir compte quand on interroge la base
  - Une autre manière de forcer l'attribut à toujours prendre une valeur est de spécifier une valeur par défaut avec l'option DEFAULT (adresse VARCHAR (50) DEFAULT 'Inconnue')

- Assurer l'intégrité de la base
  - Un attribut doit toujours avoir une valeur (NOT NULL)
  - Un attribut (ou un ensemble d'attributs) constitue(nt) la clé de la relation
  - Un attribut dans une table est liée à la clé primaire d'une autre table (intégrité référentielle)
  - La valeur d'un attribut doit être unique au sein de la relation

Contrainte d'intégrité : clé primaire

Une **clé primaire** est un attribut (ou un ensemble d'attributs) qui identifie(nt) de manière unique un t-uple d'une relation

```
create table internaute(
    email varchar(50) not null,
    nom varchar(50) not null,
    prenom varchar(50) not null,
    mdp varchar(50) not null,
    anne_naissance int,
    primary key(email));
```

Ou encore

```
create table internaute(
email varchar(50) primary key,
nom varchar(50) not null,
prenom varchar(50) not null,
mdp varchar(50) not null,
anne_naissance int);
```

```
create table notation(
id_film int not null,
email varchar(50) not null,
note int default 0,
primary key(id_film, email));
```

Contrainte d'intégrité : unicité

- On peut également spécifier que la valeur d'un attribut est unique pour l'ensemble de la colonne. Cela permet d'indiquer des clés secondaires (UNIQUE) .

```
create table artiste(
    id_artiste int not null,
    nom varchar(50) not null,
    prenom varchar(50) not null,
    annee_naissance int,
    primary key(id_artiste),
    unique(nom, prenom));
```

⇒ Il est facile de supprimer cette contrainte de clé secondaire par la suite. Ce serait beaucoup plus difficile si on avait utilisé la paire **(NOM, PRENOM)** comme clé primaire puisqu'elle serait alors utilisée pour référencer un artiste dans d'autres tables.

```
create table cinema(
nom_cinema varchar(50) not null,
adresse varchar(50) unique,
primary key(nom_cinema));
```

La clause UNIQUE ne s'applique pas aux valeurs NULL : il peut y avoir plusieurs cinémas d'adresse inconnue. En revanche le nom du cinéma est obligatoire (clause NOT NULL) et il est unique (clause PRIMARY KEY).

Contrainte d'intégrité référentielle : Clé étrangère

Une **clé étrangère** est spécifiée avec l'option FOREIGN KEY

```
create table film(
    id_film int not null,
    titre varchar(50) not null,
    annee int not null,
    id_mes int,
    code_pays int,
    primary key(id_film),
    foreign key(id_mes) references artiste(id_artiste),
    foreign key (code_pays) references pays(code_pays));
```

Le SGBD vérifie pour toute modification pouvant affecter le lien entre les deux tables, que la valeur de ID_MES correspond bien à une ligne de ARTISTE

- ✓ L'insertion dans FILM avec une valeur inconnue pour ID_MES
- ✓ La destruction d'un artiste
- ✓ La modification de ID_ARTISTE dans ARTISTE ou de ID_MES dans FILM

Contraintes de validation : CHECK

Énumération des valeurs possibles avec CHECK (condition)

- Permet d'exprimer des contraintes portant soit sur un attribut, soit sur une ligne
- La condition elle-même peut être toute expression suivant la clause WHERE dans une requête SQL. Les contraintes les plus courantes sont celles consistant à restreindre un attribut à un ensemble de valeurs
- On peut trouver des contraintes arbitrairement complexes, faisant référence à d'autres relations.

```
create table film(
    id_film int not null,
    titre varchar(50) not null,
    annee int check (annee between 1890 and 2000) not null,
    id_mes int,
    code_pays int,
    primary key(id_film),
    foreign key(id_mes) references artiste(id_artiste)
    on delete set null,
    foreign key (code_pays) references pays(code_pays));
```

LDD : insertion des données (INSERT INTO)

Insertion d'un seule ligne (enregistrement) :

INSERT INTO nom_table VALUES(valeur_att1, valeur_att2,…);

INSERT INTO nom_table(email, nom, prenom) values (val1, val2, val3)

Bien respecter l'ordre des attributs

Insertion de plusieurs enregistrements en même temps :

INSERT INTO nom_table VALUES((valeur_att1, valeur_att2,…), (valeur_att1, valeur_att2,…), (valeur_att1,valeur_att2,…), …);

```sql
insert into cinema values('pathe', '15eme arrondissement');
insert into cinema values('rex', '10eme arrondissement');
-- ou encore
insert into cinema values(('pathe', '15eme arrondissement'), ('rex', '10eme arrondissement'));
```

LDD : modification du schéma (alter table)

ALTER TABLE nomTable ACTION description

- **ACTION** : ADD, MODIFY, DROP ou RENAME
- **Description** : commande de modification associée à ACTION

1- Ajouter des attributs

```
alter table internaute add region varchar(15);
```

2- S'il existe déjà des données dans la table, la valeur sera à NULL ou à la valeur par défaut. La taille de region étant certainement insuffisante, on peut l'agrandir avec MODIFY , et la déclarer NOT NULL

```
alter table internaute modify region varchar(30) not null;
```

4- Détruire un attribut :

```
alter table internaute drop region;
```

3- Ajouter une valeur par défaut

```
alter table internaute alter region set default 'PACA';
```

SQL - Langage de manipulation de données
LMD

LMD : Langage de Manipulation des Données

- Le sous-langage LMD de SQL permet de consulter le contenu des tables et de les modifier. Il comporte 4 verbes.
  - La requête select extrait des données des tables
  - La requête insert insère de nouvelles lignes dans une table
  - La requête delete supprime des lignes d'une table
  - La requête update modifie les valeurs de colonnes de lignes existantes

LMD : projection simple «select »

```
-- projection simple avec choix des colonnes
select AVNUM, AVNOM, LOCALISATION from avion;
```

AVNUM	AVNOM	LOCALISATION
1	A300	NICE
2	A310	NICE
3	B707	PARIS
4	A300	LYON
5	CONCORDE	NICE
6	B747	PARIS
7	B707	PARIS
8	A310	TOULOUSE
9	MERCURE	LYON
10	CONCORDE	PARIS

On peut préciser explicitement le nom de la table à partir de laquelle est issu l'attribut. Il faut alors préfixer le nom de l'attribut par le nom de la table.

```
select avion.AVNUM, avion.AVNOM, avion.LOCALISATION from avion;
```

```
select * from avion;
```

*** = liste des colonnes**

LMD : projection simple «select »

Il est possible de renommer (alias) un attribut en utilisant le mot-clé **AS.**
Ce mot clé est facultatif entre l'attribut et son alias.

```
select
    PLNOM,
    PLPRENOM
from
    pilote ;
```

PLNOM	PLPRENOM
MIRANDA	SERGE
LETHANH	NAHN
TALADOOIRE	GILLES
BONFILS	ELIANE
LAKHAL	LOTFI
BONFILS	GERARD

```
select
    PLNOM as Nom,
    PLPRENOM as Prenom
from
    pilote ;
```

Nom	Prenom
MIRANDA	SERGE
LETHANH	NAHN
TALADOOIRE	GILLES
BONFILS	ELIANE
LAKHAL	LOTFI
BONFILS	GERARD
MORCENAC	PIERRE

LMD : projection simple conditionnée

```
-- projection simple conditionnée
select AVNOM
from avion
where LOCALISATIOn = 'Paris';
```

AVNOM
B707
B747
B707
CONCORDE

Il y a une différence de comportement entre SQL et l'algèbre relationnelle : ***Pour SQL, les doublons existent et sont affichés***.

```
-- projection simple conditionnée et sans doublons
select distinct AVNOM
from avion
where LOCALISATIOn = 'Paris';
```

AVNOM
B707
B747
CONCORDE

LMD : projection avec conditions plus complexes : null

```
-- projection avec conditions plus complexes : null
select PLNOM
from pilote
where VILLE = null;
```

null **ne peut être comparé à rien, même pas à lui-même !**

```
select PLNOM
from pilote
where VILLE is null;

select PLNOM
from pilote
where VILLE is not null;
```

LMD : projection avec conditions plus complexes - in et between)

```sql
select PLNOM NOM, PLPRENOM Prenom
from pilote
where VILLE in ('Paris', 'Nantes', 'Toulouse');

select PLNOM NOM, PLPRENOM Prenom
from pilote
where VILLE not in ('Paris', 'Nantes', 'Toulouse');

select PLNOM NOM, PLPRENOM Prenom
from pilote
where SALAIRE between 17000 and 25000;
```

LMD : projection avec conditions plus complexes – les masques)

```
select PLPRENOM
from   pilote
where   PLNOM like 'B_';
```

'_' = un caractère quelconque

masques

```
select PLPRENOM
from   pilote
where  PLNOM like '%AU%';
```

'%' = une chaîne quelconque

Un masque définit une famille de chaînes de caractères :

'B_'	⟶	'B1' 'Bd' 'B '
'B_'	↛	'xB' 'B' 'B12'

'%AU%'	⟶	'SEAU' 'Plein AU' 'AU LENT'
'%AU%'	↛	'BE A UPE A U' 'Achetez S A P A U !'

LMD : projection avec conditions plus complexes – Combinaisons logiques

Informations sur les pilotes habitant ‘Toulouse’ et ayant un salaire >10000

```
select PLNOM, PLPRENOM, VILLE, SALAIRE
from   pilote
where  VILLE = 'Toulouse' and SALAIRE > 10000;
```

Informations sur les pilotes ayant un salaire >10000 et qui habitent soit ‘Toulouse’ soit ‘Paris’

```
select PLNOM, PLPRENOM, VILLE, SALAIRE
from   pilote
where  SALAIRE > 1000
and    (VILLE = ‘TOULOUSE' or VILLE = 'Paris')
```

LMD : Données extraites et données dérivées - expressions de calcul

```
select PLNOM, 'SALAIRE NET', ' = ',SALAIRE - 0.25*SALAIRE
from   pilote
where  SALAIRE > 20000;
```

PLNOM	SALAIRE NET	=	SALAIRE - 0.25*SALAIRE
MIRANDA	SALAIRE NET	=	15750
LETHANH	SALAIRE NET	=	15750

Mieux encore, en utilisant des alias !

```
select PLNOM NOM, SALAIRE - 0.25*SALAIRE Salaire_NET
from   pilote
where  SALAIRE > 20000;
```

NOM	Salaire_NET
MIRANDA	15750
LETHANH	15750

LMD : les fonctions d'agrégation (statistiques)

```
select 'Paris',avg(CAPACITE) as CAPACITE_Moyenne,
         max(CAPACITE)-min(CAPACITE) as Ecart_max,
         count(*) as Nombre
from   avion
where  LOCALISATION = 'Paris';
```

	Paris	CAPACITE_Moyenne	Ecart_max	Nombre
▶	Paris	280.0000	300	4

le résultat ne comprend qu'une seule ligne

```
select sum(SALAIRE)
from   pilote
where  LOCALISATION like 'PARIS';
```

LMD : les fonctions d'agrégation (statistiques)

Attention aux valeurs dupliquées

```
select count(PLNUM)
from   pilote;
```

```
select distinct count(PLNUM)
from   vol;
```

count(PLNUM)
16

```
select count(distinct PLNUM)
from   vol;
```

count(distinct PLNUM)
9

LMD : les fonctions d'agrégation (statistiques)

```
select count(VOLNUM) as Vol,
       count(VILLEDEP) as Départ,
       count(VILLEARR) as Arrivée,
       count(AVNUM) as Avion
from   vol;
```

Vol	Départ	Arrivée	Avion
16	16	16	16

```
select count(distinct VOLNUM) as Vol,
       count(distinct  VILLEDEP) as Départ,
       count(distinct  VILLEARR) as Arrivée,
       count(distinct  AVNUM) as Avion
from   vol;
```

Vol	Départ	Arrivée	Avion
16	6	6	7

LMD : les fonctions d'agrégation (statistiques)

Attention aux ensembles vides

```
select count(*) as Nombre,
          sum(SALAIRE) as SALAIRE,
          max(SALAIRE) as Max
from   pilote
where  VILLE = 'Paris';
```

Nombre	SALAIRE	Max
4	76000	21000

```
select count(*) as Nombre,
          sum(SALAIRE) as SALAIRE,
          max(SALAIRE) as Max
from   pilote
where  VILLE = 'Lyon';
```

Nombre	SALAIRE	Max
0	NULL	NULL

LMD : les sous- requêtes

Les numéros des pilotes de Paris :

```
select PLNUM
from   pilote
where  VILLE = 'Paris';
```

Les numéros et villes de départ de vols qui ont utilisé des avions

Situés à Paris:

```
select VOLNUM, VILLEDEP
from   vol
where  AVNUM in(3, 6, 7, 10);
```

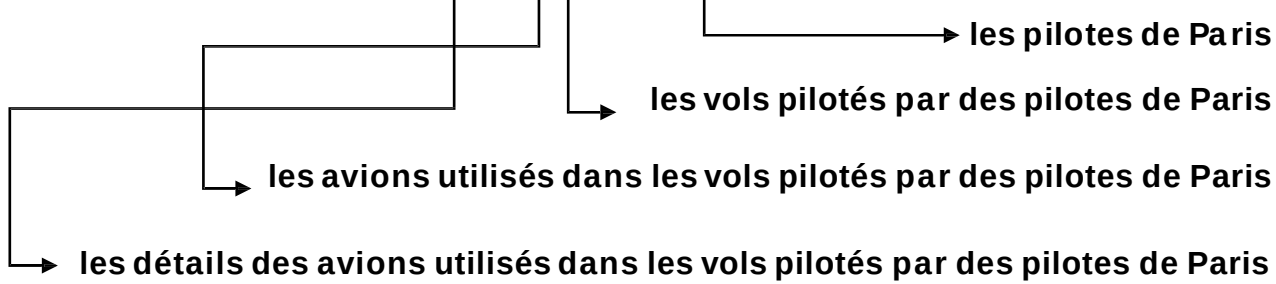
ne marche qu'une fois

```
select VOLNUM, VILLEDEP
from   vol
where  AVNUM in (select AVNUM
                 from   avion
                 where  LOCALISATION = 'Paris');
```

marchera toujours

LMD : Les sous-requêtes

```
select *
from   avion
where  AVNUM in
       (select AVNUM
        from   vol
        where  VOLNUM in
               (select VOLNUM
                from   vol
                where  PLNUM in
                       (select PLNUM
                        from   pilote
                        where  VILLE='Paris')));
```



LMD : les sous-requêtes – conditions d'association

Une condition *in (sous-requête)* correspond le plus souvent à une condition d'association = *qui sont associés à ...*

```
select *
from   T
where  CT in (select CS
              from   S
              where  <condition>);
```

"on recherche les T qui sont associés à des S qui ..."

LMD : les sous- requêtes – Références multiples

Condition d'association quantifiée : ***recherche des pilotes ayant effectué au moins 3 vols***

```
select PLNOM, PLPRENOM
from   pilote P
where  (select count(*)
        from vol
        where PLNUM = P.PLNUM ) >= 3;
```

LMD : les quantificateurs ensemblistes - exists, not exists

exists et not exists

le prédicat exists(E), où E est une sous-requête,
est *vrai* si l'ensemble désigné par E est *non vide*

Exemple : quels sont les avions pour lesquels il existe au moins un vol ?

```
select AVNUM, AVNOM
from   avion as A
where  exists (select *
               from   vol
               where  AVNUM = A.AVNUM);
```

le prédicat not exists(E),
est vrai si l'ensemble désigné par E est vide

LMD : les quantificateurs ensemblistes - all, any

Quelles sont les commandes qui utilisent la plus petite quantité de PA60 ?

```
select distinct NCOM
from   DETAIL
where  QCOM <= all (select QCOM
                    from   DETAIL
                    where  NPRO = 'PA60')
and    NPRO = 'PA60';
```

```
select distinct NCOM
from   DETAIL
where  QCOM = (select min(QCOM)
               from   DETAIL
               where  NPRO = 'PA60')
and    NPRO = 'PA60';
```

"distinct" est-il utile ?

LMD : les quantificateurs ensemblistes - all, any

Quels sont les commandes qui ne spécifient pas la plus petite quantité de PA60 ?

```
select *
from   DETAIL
where  QCOM > any (select QCOM
                   from   DETAIL
                   where  NPRO = 'PA60')
and    NPRO = 'PA60';
```

```
select distinct NCOM
from   DETAIL
where  QCOM > (select min(QCOM)
               from   DETAIL
               where  NPRO = 'PA60')
and    NPRO = 'PA60';
```

variante

LMD : les quantificateurs ensemblistes – pour tout

Curieusement, SQL ne permet pas
d'exprimer directement le quantificateur pour tout

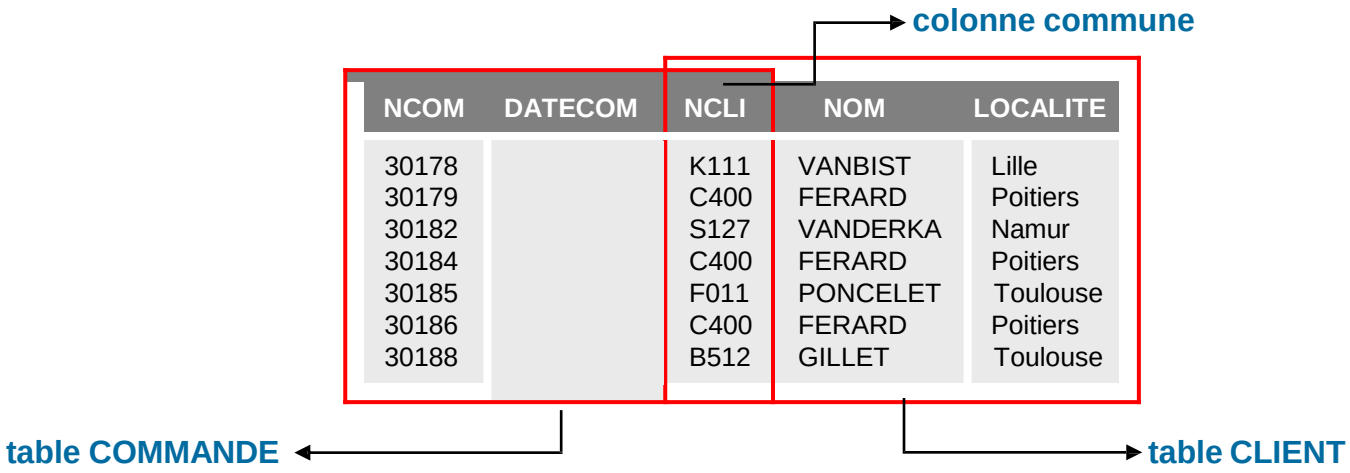
la logique nous apprend que : $(\forall\ x, p(x)) \equiv \neg(\exists\ x, \neg p(x))$

Application : ***quelles sont les commandes qui spécifient tous les produits*** **?**

la COMMANDE M est retenue	→	`select NCOM from COMMANDE M`
si,	→	`where`
il n'existe pas	→	`  not exists`
de PRODUIT P,	→	`    (select * from PRODUIT P`
tel que	→	`     where`
P n'est pas dans	→	`       P.NPRO not in`
l'ensemble des PRODUITs	→	`           (select NPRO from DETAIL`
commandés par M	→	`            where  NCOM = M.NCOM));`

LMD : les jointures

La jointure permet de produire une table constituée de données extraites de plusieurs tables :



EQUI-JOINTURE SUR ATTRIBUTS DE MEME NOM

```
select NCOM, DATECOM, CLIENT.NCLI, NOM, LOCALITE
from   COMMANDE, CLIENT
where  COMMANDE.NCLI = CLIENT.NCLI;
```

Ou encore

```
select NCOM, DATECOM, CLIENT.NCLI, NOM, LOCALITE
from   COMMANDE Join CLIENT
using(NCLI);
```

LMD : les jointures

NCOM	DATECOM	NCLI	NOM	LOCALITE
30178		K111	VANBIST	Lille
30179		C400	FERARD	Poitiers
30182		S127	VANDERKA	Namur
30184		C400	FERARD	Poitiers
30185		F011	PONCELET	Toulouse
30186		C400	FERARD	Poitiers
30188		B512	GILLET	Toulouse

préfixe nécessaire car ambiguité

```
select NCOM , DATECOM, CLIENT.NCLI, NOM, LOCALITE
from   COMMANDE, CLIENT
where  COMMANDE.NCLI = CLIENT.NCLI
```

plusieurs tables

condition de jointure

LMD : les jointures

```
select avion.AVNOM
from   pilote, avion, vol
where  pilote.PLNUM = vol.PLNUM
and    vol.AVNUM = avion.AVNUM;;
```

jointure de 3 tables

```
select avion.AVNOM
from pilote
join vol using(PLNUM)
join avion using(AVNUM);
```

```
select avion.AVNOM
from pilote
join vol on pilote.PLNUM=vol.PLNUM
join avion on vol.AVNUM=avion.AVNUM
where avion.CAPACITE >200;
```

condition de jointure
+
conditions de sélection

LMD : les jointures - Produit relationnel

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

pas de condition
de jointure !

Produit relationnel :
chaque ligne de avion est couplée avec chaque ligne de pilote

requête valide mais d'utilité réduite

LMD : les jointures - Produit relationnel

La requête :

```
select *
from   vol, pilote
where  vol.PLNUM = pilote.PLNUM;
```

. . . ignore les lignes de pilote
qui n'ont pas de lignes correspondantes dans vol

Extraire ces lignes de pilotes:

```
select '--', PLNUM, '--', PLNOM, VILLE
from   pilote
where  not exists (select * from vol
                   where PLNUM = pilote.PLNUM);
```

LMD : Sous- requête ou jointure ?

Peut-on remplacer une sous-requête par une jointure ?

```
select VOLNUM, VILLEDEP
from   vol
where  PLNUM in (select PLNUM
                 from   pilote
                 where  VILLE = 'Paris');
```

=

```
select VOLNUM, VILLEDEP
from   vol
join pilote
on  vol.PLNUM = pilote.PLNUM
and    VILLE = 'Paris';
```

LMD : Sous-requête ou jointure

Mais ...

```
select NCOM, DATECOM, NCLI
from   COMMANDE
where  NCOM not in ( select NCOM
                     from   DETAIL
                     where NPRO = 'PA60');
```

≠

```
select distinct COMMANDE.NCOM, DATECOM, NCLI
from   COMMANDE, DETAIL
where  COMMANDE.NCOM = DETAIL.NCOM
and    NPRO <> 'PA60';
```

≠

```
select distinct COMMANDE.NCOM, DATECOM, NCLI
from   COMMANDE, DETAIL
where  COMMANDE.NCOM <> DETAIL.NCOM
and    NPRO = 'PA60';
```

Sous requête ou jointure

La sous-requête permet de formuler

- une condition d'association (in)
- une condition de non-association (not in)

La jointure permet de formuler

- une condition d'association

LMD : Les jointures - Valeurs dérivées dans une jointure

```
select NCOM, D.NPRO, QCOM*PRIX
from   DETAIL D join PRODUIT P
on  D.NPRO = P.NPRO;
```

```
select 'Montant commande 30184 = ', sum(QCOM*PRIX)
from   DETAIL D, PRODUIT P
where  D.NCOM = '30184'
and    D.NPRO = P.NPRO;
```

LMD : Les jointures - Interprétation du résultat d'une jointure

- Une ligne de pilote représente un pilote.
- Une ligne de vol représente un vol.

Que représente chaque ligne de la jointure vol * pilote :

```
select P.PLNUM, P.PLNOM, P.VILLE
 from   pilote P join vol V
 on P.PLNUM = V.PLNUM ;
```

- un pilote ?
- un pilote qui a effectué un vol?
- Un vol?

Autre formulation : il y autant de lignes dans le résultat qu'il y a

- de pilotes ?
- de pilotes qui ont effectués des vols ?
- de vols ?

LMD : opérations ensemblistes

union (∪)

intersection (∩)

différence (-)

Pas de problèmes pour deux *ensembles*.

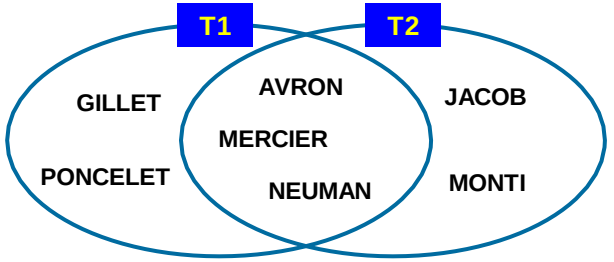
Mais qu'en est-il pour deux *tables* ?

LMD : Opérations ensemblistes

Opérateurs ensemblistes entre 2 tables sans doublons

T1
NOM
GILLET
AVRON
MERCIER
PONCELET
NEUMAN

T2
NOM
MONTI
NEUMAN
JACOB
MERCIER
AVRON



```
select NOM
from   T1
union
select NOM
from   T2
```

NOM
GILLET
AVRON
MERCIER
PONCELET
NEUMAN
MONTI
JACOB

```
select NOM
from   T1
intersect
select NOM
from   T2
```

NOM
AVRON
MERCIER
NEUMAN

```
select NOM
from   T1
except
select NOM
from   T2
```

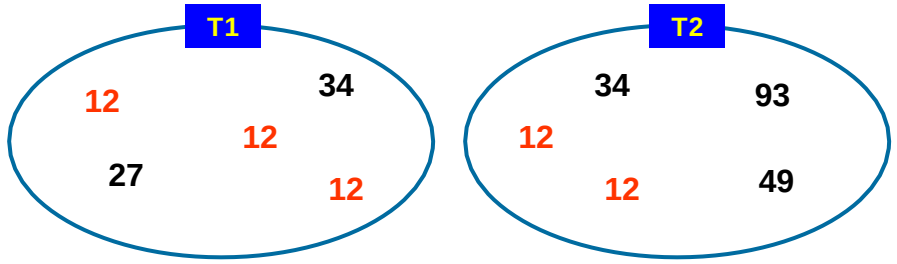
NOM
GILLET
PONCELET

LMD : Opérations ensemblistes

Opérateurs ensemblistes entre 2 tables avec doublons

T1
NUM
12
34
27
12
12

T2
NUM
93
12
34
12
49



```
select NUM
from   T1
union
select NUM
from   T2
```

NUM
12
34
27
93
49

```
select NUM
from   T1
intersect
select NUM
from   T2
```

NUM
34
12

```
select NUM
from   T1
except
select NUM
from   T2
```

NUM
27

```
select NUM
from   T2
except
select NUM
from   T1
```

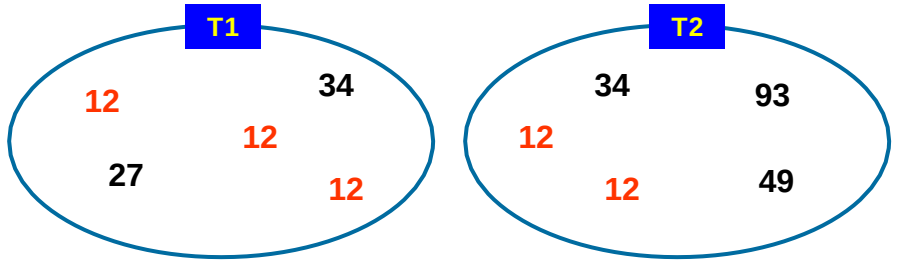
NUM
93
49

LMD : Opérations ensemblistes – les multi-ensembles

Opérateurs multi-ensemblistes entre 2 tables avec doublons

T1
NUM
12
34
27
12
12

T2
NUM
93
12
34
12
49




```
select NUM
from   T1
union all
select NUM
from   T2
```

NUM
12
12
12
34
27
12
12
34
93
49

```
select NUM
from   T1
intersect all
select NUM
from   T2
```

NUM
34
12
12

```
select NUM
from   T1
except all
select NUM
from   T2
```

NUM
12
27

```
select NUM
from   T2
except all
select NUM
from   T1
```

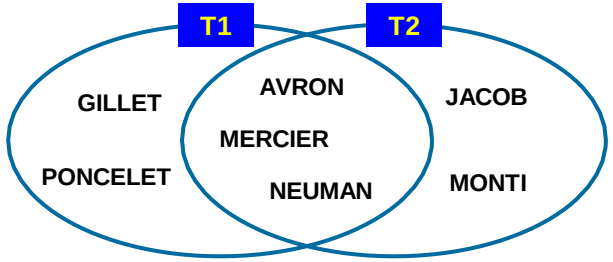
NUM
93
49

LMD : Opérations ensemblistes – Expressions complexes

Les opérateurs peuvent être combinés (ex. *différence symétrique*)

T1
NOM
GILLET
AVRON
MERCIER
PONCELET
NEUMAN

T2
NOM
MONTI
NEUMAN
JACOB
MERCIER
AVRON



```
(select NOM
from T1
except
select NOM
from T2)
union
(select NOM
from T2
except
select NOM
from  T1)
```

⇒

NOM
GILLET
PONCELET
JACOB
MONTI

LMD : Opérations ensemblistes – jointure externes

Jointure externe à gauche permet d'obtenir au moins une fois tous les tuples de la relation à gauche de la jointure. Sans correspondant dans relation de droite, des valeurs nulles sont insérées.

« Afficher les noms et prénoms de tous les pilotes, avec le nom des villes de départ des vols qu'ils assurent, le cas échéant »

```
SELECT
    PLNOM,
    PLPRENOM,
    VILLEDEP
FROM
    pilotes LEFT OUTER JOIN vols USING(PLNUM);
```

résultat : Gérard BONFILS apparaît au moins une fois, associé à une valeur **`null`** (puisqu'il n'y a aucun vol auquel le relier)

LMD : Opérations ensemblistes – jointure externes

Jointure externe à droite permet d'obtenir au moins une fois tous les tuples de la relation à droite de la jointure. Sans correspondant dans la relation de gauche, des valeurs nulles sont insérées

« Afficher les noms et prénoms de tous les pilotes, avec le nom des villes de départ des vols qu'ils assurent, le cas échéant »

```
SELECT
    PLNOM,
    PLPRENOM,
    VILLEDEP
FROM
    vols RIGHT OUTER JOIN pilotes USING(PLNUM);
```

LMD : Opérations ensemblistes – jointure externes

Jointure externe complète
(**`full outer join`**)

permet d'obtenir au moins une fois tous les tuples de la relation à droite de la jointure et au moins une fois tous les tuples de la relation à gauche de la jointure.

LMD : les jointures - récapitulatif

A	att1	att2
	1	2
	3	3
	5	3
	10	4

B	att3	att4
	2	2
	3	25
	5	20

```
SELECT *
FROM A JOIN B
ON A.att2 =
B.att3 ;
```

Résultat	att1	att2	att3	att4
	1	2	2	2
	3	3	3	25
	5	3	3	25

```
SELECT *
FROM A FULL OUTER JOIN B
ON A.att2 = B.att3 ;
```

Résultat	att1	att2	att3	att4
	1	2	2	2
	3	3	3	25
	5	3	3	25
	10	4	*null*	*null*
	null	*null*	5	20

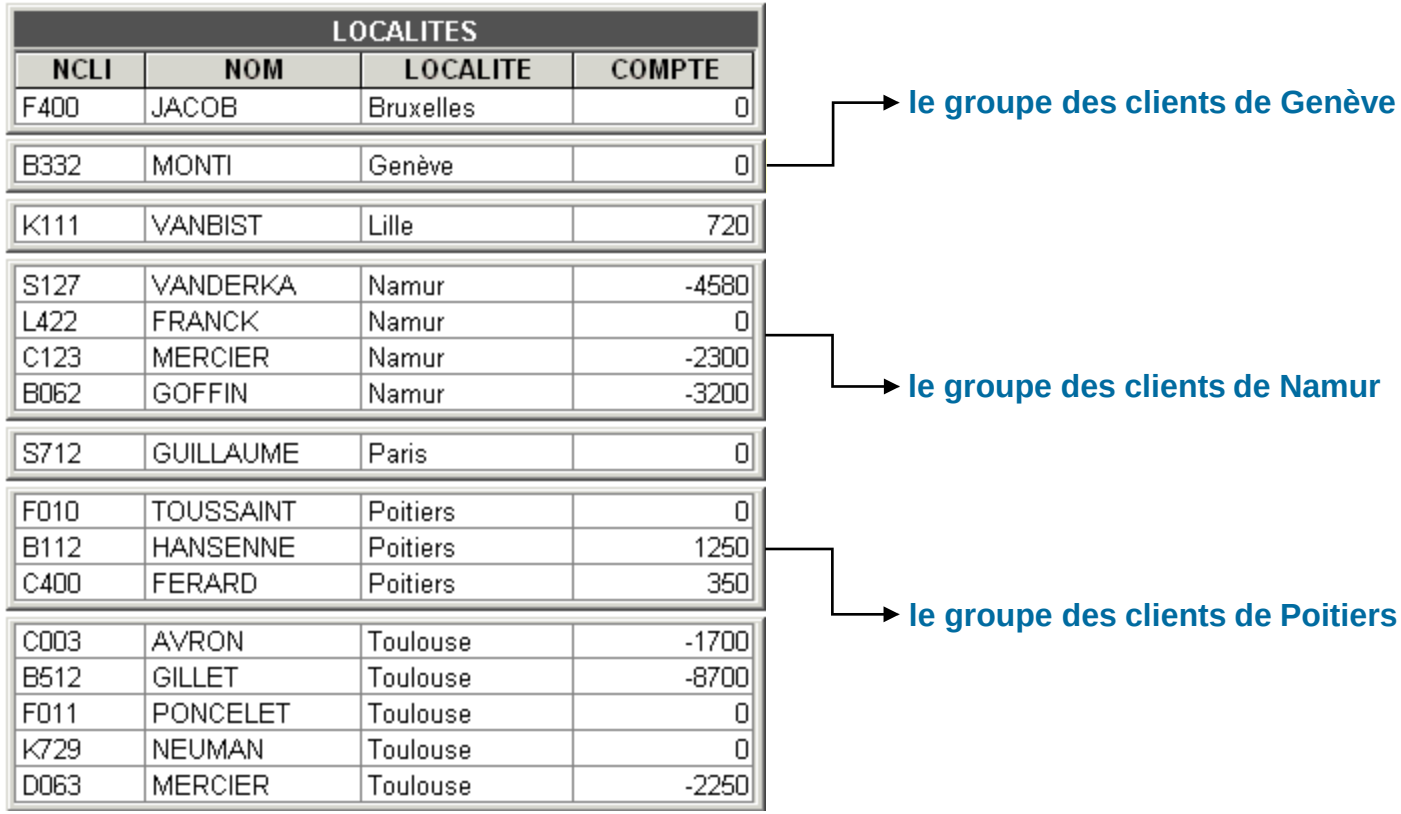
```
SELECT *
FROM A RIGHT JOIN B
ON A.att2 = B.att3 ;
```

Résultat	att1	att2	att3	att4
	1	2	2	2
	3	3	3	25
	5	3	3	25
	Null	*Null*	5	20

```
SELECT *
FROM A LEFT JOIN B
ON      A.att2      =
B.att3 ;
```

Résultat	att1	att2	att3	att4
	1	2	2	2
	3	3	3	25
	5	3	3	25
	10	4	*Null*	*null*

LMD : Les données groupées - Principe



LOCALITES			
NCLI	NOM	LOCALITE	COMPTE
F400	JACOB	Bruxelles	0
B332	MONTI	Genève	0
K111	VANBIST	Lille	720
S127	VANDERKA	Namur	-4580
L422	FRANCK	Namur	0
C123	MERCIER	Namur	-2300
B062	GOFFIN	Namur	-3200
S712	GUILLAUME	Paris	0
F010	TOUSSAINT	Poitiers	0
B112	HANSENNE	Poitiers	1250
C400	FERARD	Poitiers	350
C003	AVRON	Toulouse	-1700
B512	GILLET	Toulouse	-8700
F011	PONCELET	Toulouse	0
K729	NEUMAN	Toulouse	0
D063	MERCIER	Toulouse	-2250

LMD : Les données groupées - Principe

```
select VILLE,
     count(*) as NOMBRE_PILOTES,
     avg(SALAIRE) as MOYENNE_SALAIRE
from   pilote
group by VILLE;
```

VILLE	NOMBRE_PILOTE	MOYENNE_SALAIRE
PARIS	4	19000
TOULOUSE	2	20000
NICE	3	17666.666666666668
LYON	1	15000

→ le groupe des pilotes de Paris

→ le groupe des pilotes de TOULOUSE

→ le groupe des pilotes de NICE

On s'intéresse aux VILLES et non plus aux pilotes

LMD : Les données groupées – Sélection des groupes

```
select VILLE, count(*), avg(SALAIRE)
from   pilote
group by VILLE
having count(*) >= 3;
```

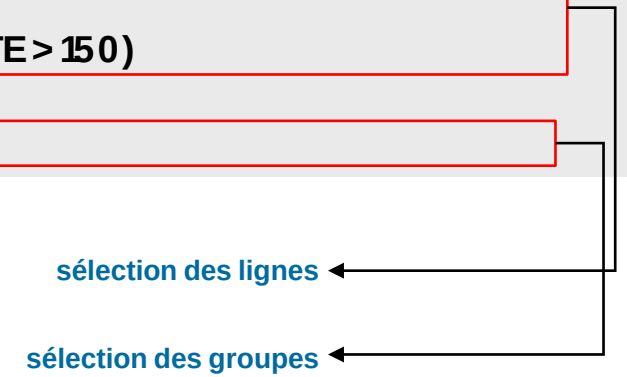
VILLE	count(*)	avg(SALAIRE)
PARIS	4	19000
NICE	3	17666.666666666668

```
select PLNUM, count(*)
from   vol
group by PLNUM
having count(*) >= 2;
```

PLNUM	count(*)
1	5
8	3
9	2

LMD : Les données groupées – Sélection de lignes et des groupes

```
select PLNUM, count(*)
from  vol
where AVNUM in ( select AVNUM
        from  avion
        where CAPACITE>150)
group by AVNUM
having count(*) >= 2;
```



LMD : Ordre et interprétation - Comment lire (écrire) une requête ?

```
7 : select NCLI, count(*), sum(QCOM)
1: from   COMMANDE M, DETAIL D
2 : where  M.NCOM = D.NCOM
3 : and    NPRO = 'PA60'
4 : group by NCLI
5 : having count(*) >= 2
6 : order by NCLI
```