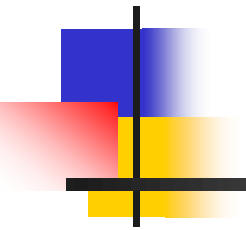
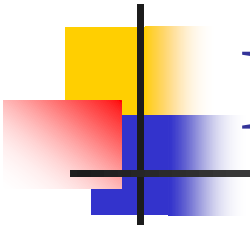


# Systèmes de Base de Données

Université de Lille2 – IUT C - STID

Dr Khalid GABER



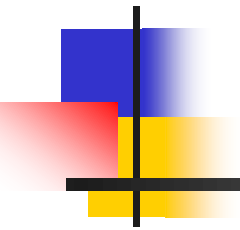
# Introduction

- Une base de données est une collection de données mises en relation .
- Un Système de Gestion de Base de Données (SGBD) est un logiciel qui permet la création et l'exploitation de la base de données, il assure l'interface entre la base de données et les requêtes de l'utilisateur qui l'interroge.
- Les requêtes sont exprimées dans un langage : langage d'interrogation ( SQL).

# SGBD - suite

On peut distinguer 3 générations de SGBD :

- Année 60-70 elle est conçue sur le modèle de données hiérarchique ceci impose au programmeur de spécifier le chemin d'accès au données
- 1970 avec l'apparition du **modèle de données** relationnel. Les programmeurs n'ont plus la charge de spécifier le meilleur chemin d'accès aux données mais c'est au système de le déterminer.
- Année 80 avec le modèle **objet**.



# Le modèle relationnel

# Modèle relationnel

- Dans le modèle relationnel les structures de données sont des relations au sens mathématique du terme (les valeurs sont des éléments d'un produit cartésien). On représente une relation par une table.

# Les concepts de base

- **DOMAINE** :

  Ensemble de valeurs caractérisé par un nom. Chaque valeur du domaine est atomique et donc indivisible

  COULEUR = { Blanc, Rouge, Vert }

- **Un schéma de relation** R :

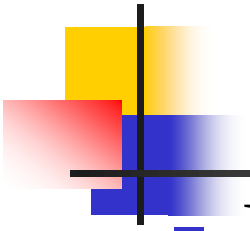
  R(A1:D1, A2:D2, ..., An:Dn) est un ensemble d'attributs. Chaque attribut Ai est le nom d'un rôle joué par son domaine Di dans le schéma de relation R.

# Schéma de relation : exemple

- *ABONNE(**NumAbo:entier**, Nom:Texte, Prénom: chaine, Rue: Texte, Ville:Texte, CodePostal: entier).*
- Cette relation regroupe les informations sur les abonnés de la médiathèque : *NumAbo* qui identifie tout abonné de manière individuelle et qui prendra pour valeur un entier représentant le numéro de l'abonné.

# Schéma de relation - suite

- Définir un schéma de relation revient à spécifier un nouveau type de données équivalent à un type struct en langage C.
- Le modèle relationnel n'autorise qu'un seul niveau de structure. Il n'est pas possible par exemple de définir un attribut Adresse qui se décompose en Rue, Ville, CodePostal.

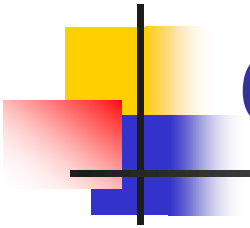


# Table

- Relations sont  stockées sous forme de tables.
- Chaque colonne ( ou propriété ou champ ou attribut) à un nom et un type.
- Chaque ligne ( ou enregistrement ou tuple) contient les données relatives à une occurrence de l'objet concerné par la table.
- Toutes les données d'une colonne sont de même type.

# Exemple de table : Pièce

| Etudiant | | | | |
|---|---|---|---|---|
| **CodeEtu** | **Nom** | **Prenom** | **DateNaissance** | **Ville** |
| 1 | DUCROCQ | Florent | 10/12/1990 | Lille |
| 2 | DUPRE | Julien | 01/02/1989 | Lille |
| 3 | Gaber | Ahmed | 14/02/1989 | Lyon |
| 4 | MAILLIET | René | 05/09/1992 | Lyon |
| 5 | MARIE | Pauline | 30/10/1990 | Paris |
| 6 | THUEUX | Audrey | 28/02/1989 | Lille |
| 7 | VASSEUR | Pascal | 10/08/1987 | Lille |
| 8 | Gaber | Ahmed | 10/12/1980 | Alexandrie |



# Colonne

- Le nom complet d'une colonne comprend le nom de la table à laquelle elle appartient (obligatoire en cas d'ambiguïté) :

  Client.Nom, Pilote.Nom

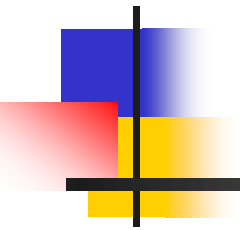
# Quelques définitions

- **ARITE d'une relation** : Nombre fixe de ses attributs (colonnes)
- **CARDINALITE d'une relation** : Nombre de lignes (tuples, enregistrements)
- **SCHEMA d'une relation** : Nom de la relation suivi de la liste de ses attributs et des domaines associés :

  R1( A1:D1, A2:D2, ……, An:Dn)

# Clé d’une relation

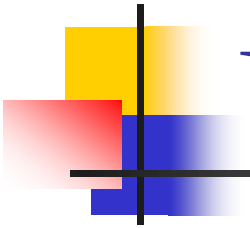
- Clé d’une relation (ou table) est un sous-ensemble d'attributs qui permet de caractériser tout enregistrement d'une relation.
- Une relation est un ensemble d’enregistrements, il ne peut donc pas y avoir deux enregistrements strictement identiques dans la même relation.



# Algèbre relationnelle

# Algèbre relationnelle

- C'est une collection d'opérations formelles sur les relations, et dont le résultat est également une relation.
- Le langage SQL appelée langage d'interrogation est basée sur l'utilisation des **opérateurs de l'algèbre relationnels**. Il existe **huit opérateurs**. Quatre sont spécifiques à la manipulation des données : **Projection**, **Sélection**, **Jointure**, **Division**.
- Quatre correspondent aux opérateurs de la théorie des ensembles : **Union**, **Intersection**, **Différence**, **Produit cartésien**.



# Union

- L'union de deux relations R et S de même schéma est une relation T de même schéma contenant l'ensemble des n-uplets (occurrences) appartenant à R ou à S ou au deux.
- Notation : $\mathbf{T = R \cup S}$

**Pièces1**

| Numéro | désignation |
|---|---|
| 1 | écrou |
| 2 | vis |
| 3 | clou |

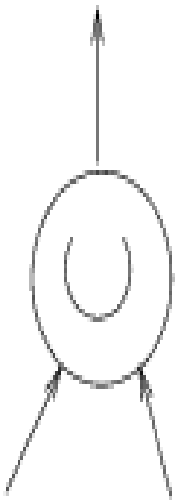
**Pièces2**

| Numéro | désignation |
|---|---|
| 4 | punaise |
| 1 | écrou |
| 5 | agrafe |

**Pièces3**

| Numéro | désignation |
|---|---|
| 1 | écrou |
| 2 | vis |
| 3 | clou |
| 4 | punaise |
| 5 | agrafe |

# Union



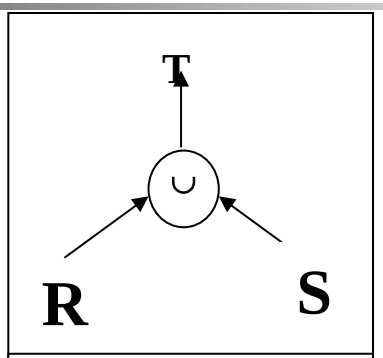
UNION

| A | B | C |
|---|---|---|
| a | 1 | a |
| b | 1 | b |
| a | 1 | d |
| b | 2 | f |
| a | 3 | f |

| A | B | C |
|---|---|---|
| a | 1 | a |
| b | 1 | b |
| a | 1 | d |
| b | 2 | f |

| A | B | C |
|---|---|---|
| a | 3 | f |

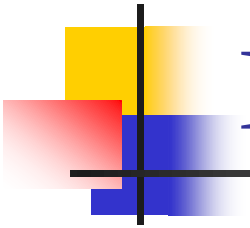
# Union



Représentation graphique

Propriétés de $T = R \cup S$ :

- $R \cup S = S \cup R$
- $Card(T) \leq Card(R) + Card(S)$
- Arité(T) = Arité(R) = Arité(S)



# Différence

- La différence de deux relations de même schéma (dans l'ordre R et S), est une relation T de même schéma contenant les n-uplets appartenant à R et n'appartenant pas à S.
- Notation : **T = R – S**

# Pièces4 = Pièces1 – Pièces2
# Pièces5 = Pièces2 – Pièces1

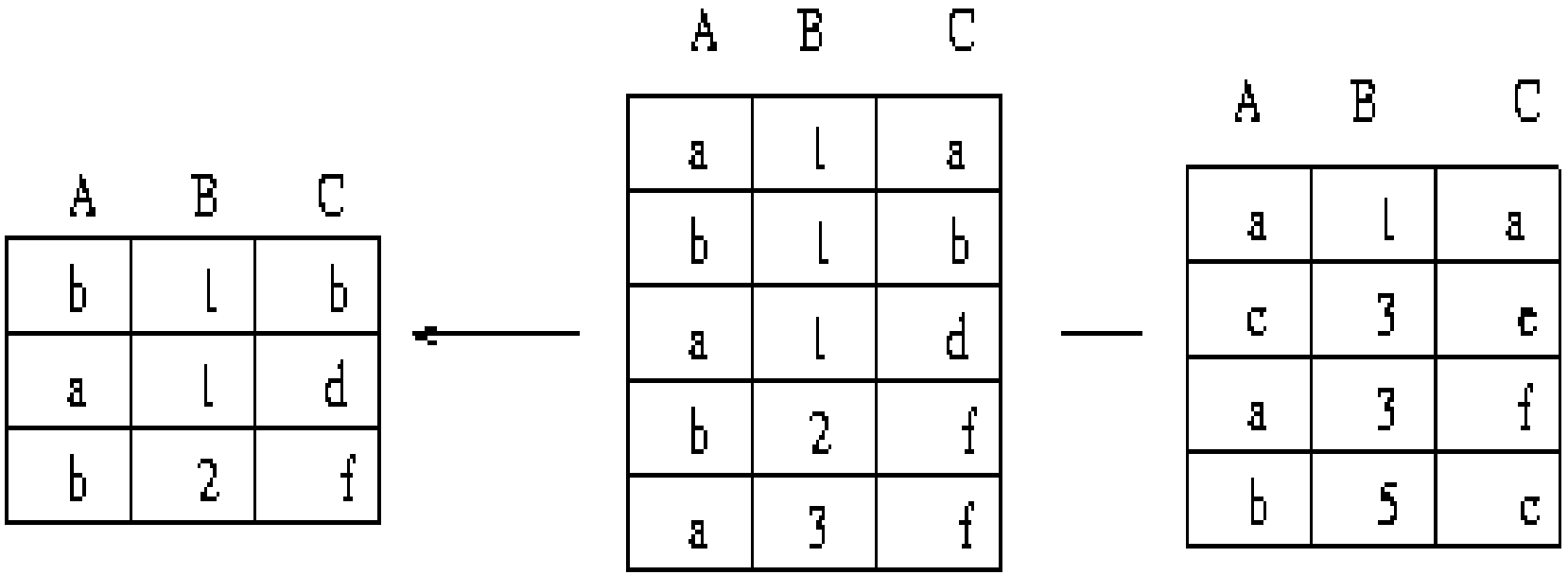
**Pièces4**

| **Numéro** | **désignation** |
|---|---|
| 2 | vis |
| 3 | clou |

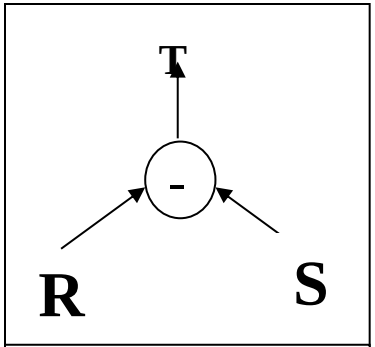
**Pièces5**

| **Numéro** | **désignation** |
|---|---|
| 4 | punaise |
| 5 | agrafe |

# Différence



# Différence



Représentation graphique

Propriétés de **T = R – S :**

- R - S ≠ S - R
- Card(T) = Card(R) - Card(R ∩ S)
- Arité(T) = Arité(R) = Arité(S)

# Produit cartésien

- Le produit cartésien de deux relations quelconques (de schéma quelconque) R et S est une relation ayant pour attributs la concaténation de ceux de R et de S et dont les n-uplets sont tous les concaténations d'un n-uplet de R à un n-uplet de S.
- Notation : $T \subseteq R \times S$

**Pièces1**

| Numéro | désignation |
|---|---|
| 1 | écrou |
| 2 | vis |
| 3 | clou |

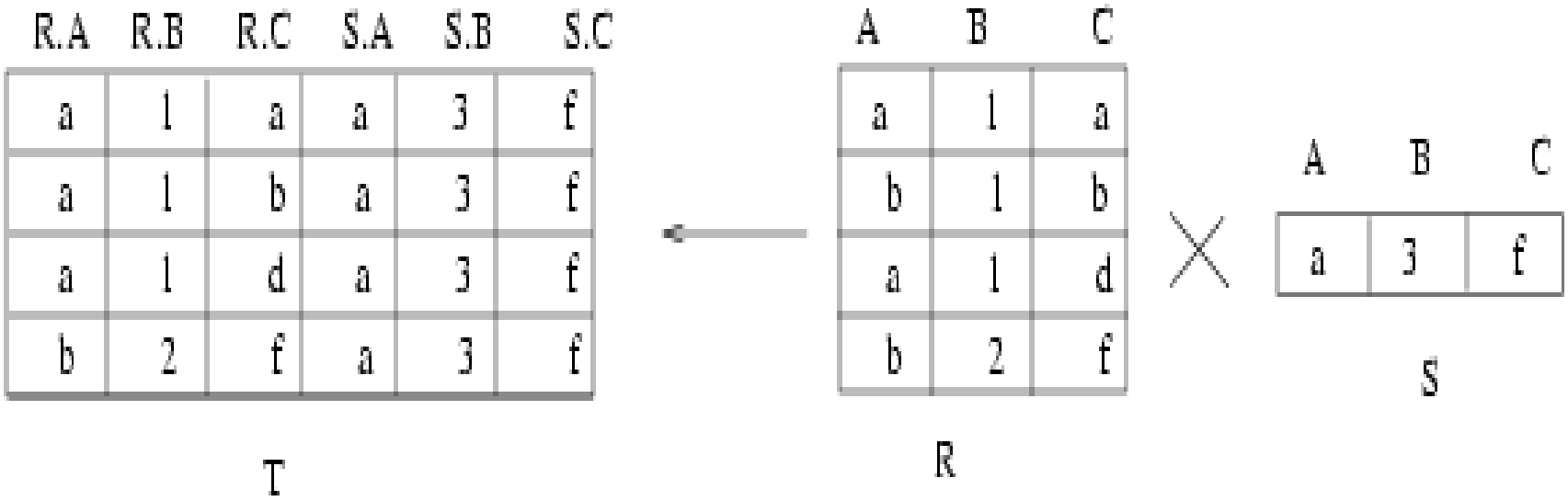
**Machine**

| Numéro | désignation | prix |
|---|---|---|
| M10 | perceuse | 50 |
| M30 | pompe | 69 |

**Pièces1 × Machine**

| P.Numéro | P.désignation | M.Numéro | M.désignation | M.prix |
|---|---|---|---|---|
| 1 | écrou | M10 | perceuse | 50 |
| 1 | écrou | M30 | pompe | 69 |
| 2 | vis | M10 | perceuse | 50 |
| 2 | vis | M30 | pompe | 69 |
| 3 | clou | M10 | perceuse | 50 |
| 3 | clou | M30 | pompe | 69 |
| | | | | |

# Produit cartésien



| R.A | R.B | R.C | S.A | S.B | S.C |
|---|---|---|---|---|---|
| a | 1 | a | a | 3 | f |
| a | 1 | b | a | 3 | f |
| a | 1 | d | a | 3 | f |
| b | 2 | f | a | 3 | f |

T

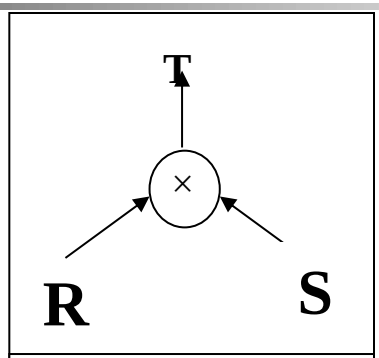
| A | B | C |
|---|---|---|
| a | 1 | a |
| b | 1 | b |
| a | 1 | d |
| b | 2 | f |

R

| A | B | C |
|---|---|---|
| a | 3 | f |

S

# Produit cartésien




Représentation graphique

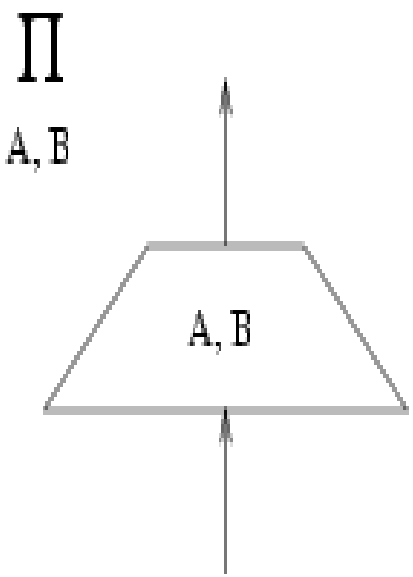
Propriétés de $\mathbf{T \subseteq R \times S}$ :

- $Card(T) \subseteq Card(R) \times Card(R \cap S)$
- $Arité(T) = Arité(R) + Arité(S)$

# Projection

- La projection d'une relation R de schéma $R(A_1,A_2,\ldots A_n)$ sur les attributs $A_{i1},A_{i2},\ldots,A_{ip}$, avec $i_j \neq i_k$ et $p < n$, est une relation T de schéma $T(A_{i1},A_{i2},\ldots,A_{ip})$ dont les n-uplets sont obtenus par élimination des valeurs des attributs de R n'appartenant pas à T et par suppression des n-uplets en double.

# Projection



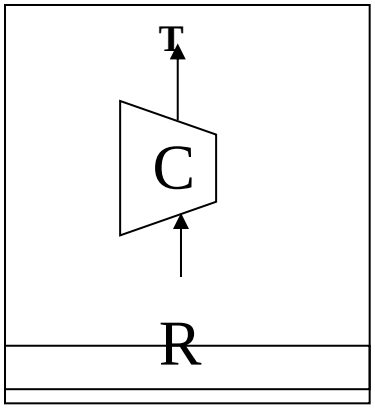

| A | B |
|---|---|
| a | 1 |
| b | 2 |

Π
A, B

| A | B | C |
|---|---|---|
| a | 1 | a |
| a | 1 | b |
| a | 1 | d |
| b | 2 | f |

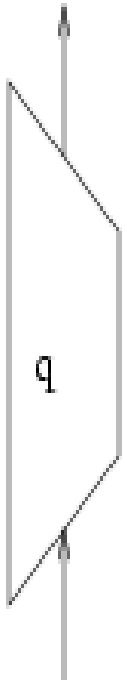
# Sélection

- La sélection d'une relation R par une (ou plusieurs) condition(s) C sur ses attributs est une relation R' de même schéma et dont les n-uplets sont ceux satisfaisant la condition C.
- Card(T) ≤ Card(R)
- Arité(T) = Arité(R)



# Sélection

$\sigma_q$



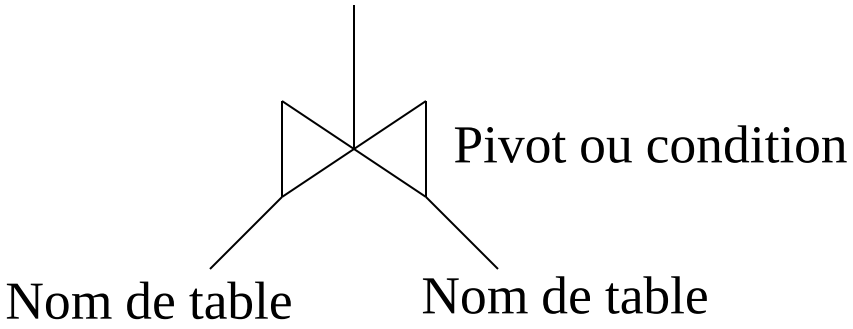

| A | B | C |
|---|---|---|
| a | 1 | a |
| a | 1 | b |
| a | 1 | d |

$\sigma_{A=a}$

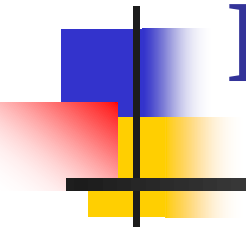
| A | B | C |
|---|---|---|
| a | 1 | a |
| a | 1 | b |
| a | 1 | d |
| b | 2 | f |

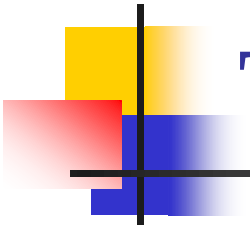
# Jointure

- La jointure entre deux relations R et S en fonction d'une (ou de plusieurs) condition(s) C sur les attributs est l'ensemble des n-uplets du produit cartésien R×S satisfaisant la (ou les) condition(s).



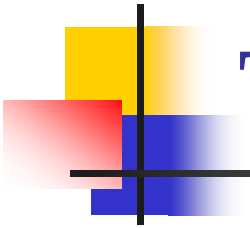
# SQL : Structured Query Language





# Types de données

- Chaînes de caractère ;
- Numérique ;
- Abstrait.



# Types

- **CHAR(n)**
- **VARCHAR(n)**
- Numérique : NUMBER, INTEGER, INT, REAL, DECIMAL, FLOAT, …
- Abstrait : Date,Time et Money

# Création d'une table

**CREATE TABLE <Nom_Table>**

**(Attribut1 type1 [DEFAULT valeur1] containte1,**

**…);**

**CREATE TABLE Pilote**

**(Code INT PRIMARY KEY,**

**Nom VARCHAR(20) NOT NULL,**

**Salaire FLOAT DEFAULT 2200,**

**Sexe CHAR ) ;**

# Destruction d'une table

- **DROP TABLE <Nom_Table> ;**

· La destruction des informations contenues dans la table ;

· La destruction du schéma de la table.

- Exemple :

  DROP TABLE Pilote ;

# Insertion de lignes (tuples) : INSERT

- **INSERT INTO {Nom_Table | Nom_Vue}**
  **[ ( colonne1 , colonne2, … ) ]**
  **VALUES ( val1 , val2 …);**
- **INSERT INTO {Nom_Table | Nom_Vue}**
  **SELECT … ;**
- INSERT INTO Pilote
  (Code, Nom, Sexe)
  VALUES (20, ‘GABER’, ‘M’) ;

# Modification des données : UPDATE

- **UPDATE Nom_Table**

  **SET colonne1 = {Exp1 | (SELECT …)}, colonne2 = {Exp2 | (SELECT …)}, …**

  **[WHERE Condition] ;**

***Exemple*** **: Augmenter de 10% le salaire mensuel des pilotes qui ne reçoivent pas de commission :**

**UPDATE pilote**

**SET sal = sal * 1.1**

**WHERE commission is NULL ;**

# Création et insertion simultanées

- **CREATE TABLE Nom_Table**

  **AS SELECT …**

  **Les colonnes de la table ainsi crée héritent du nom et du type des attributs projetés par l'ordre SELECT.**

# *Suppression de ligne : DELETE*

- **DELETE FROM {Nom_Table | Nom_Vue}**
  **[WHERE condition] ;**

  **DELETE FROM Pilote**
  **WHERE Sexe =’H’;**

# *Sélection de tuples (lignes)*

**SELECT** **<clause d'unicité>**
**<liste de colonnes>**
**FROM** **<liste de tables>**
**WHERE** **<critères de sélection>**
**GROUP BY** **<liste de colonnes>**
**HAVING** **<critère de sélection>**
**ORDER BY** **<critère d'ordre>;**

# *Sélection de lignes*

- Chaque requête de sélection contient au minimum la forme.

  SELECT <liste de colonnes>

  FROM <liste de tables> ;

**clause d'unicité** : **ALL** ou **DISTINCT**.

**liste des colonnes** : composée d'une * indique que toute les colonnes sont retenues.

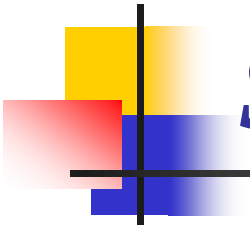
# *Sélection de lignes*

Le nom d'une colonne est soit :

- Le nom d'une colonne d'une table ou d'une vue (exemple : sexe) ;
- Le nom d'une colonne préfixée du nom de la table (ou d'une vue) dans laquelle elle se trouve (exemple : **pilote.nom**) ;
- Le nom d'une colonne d'un alias d'un nom de table ou de vue donné dans la <liste des tables> ;
- Une valeur calculée

# *Sélection de lignes*

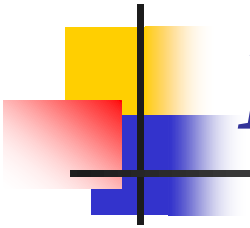
- **liste des tables** : est composée de noms des tables ou de vues séparées par des virgule.
- **ORDER BY d'ordre** est constitué :
  - d'une liste de colonnes séparée par des virgules ;
  - ou d'une liste de numéro séparée par des virgules.



# *Sélection de lignes*

- Chaque numéro correspond au rang d'une colonne dans la liste de la clause SELECT.

- Faire une projection consiste à définir un sous-ensemble des colonnes de la liste des noms des tables.



# *Restriction*

- Une restriction correspond à un choix de ligne à sélectionner. Le résultat peut comporter 0, une ou plusieurs lignes. Chacune des lignes vérifient le critère de sélection.
- Exemple : sélection des hommes dont la taille est supérieur à 1.72 :

```
SELECT Nom
FROM personne
WHERE sexe = 'M' AND AND taille >1.72;
```

# *Sélection ordonnée*

- SQL permet de trier les enregistrements d'une requête.
- Il est possible de spécifier un ordre croissant « ASC » ou décroissant DESC.

Exemple : on souhaite afficher tous les noms des personnes dans l'ordre alphabétique :

```
SELECT Nom
FROM personne
ORDER BY Nom ;
```

# *Fonctions*

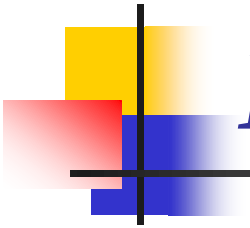
**COUNT ({ [DISTINCT | ALL] nom_colonne | * } )**

→ Compte le nombre d'occurrences d'une table.

Le paramètre DISTINCT permet de compter les occurrences uniques.

SELECT COUNT (DISTINCT Nom )

FROM personne ;

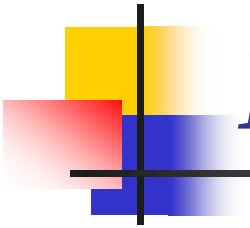


# *Fonctions*

**SUM ( [ DISTINCT | ALL] nom-colonne)**

→ Calcul la somme d'une colonne numérique.

Le paramètre DISTINCT permet d'effectuer la somme des occurrences uniques d'une colonne



# *Fonctions*

**AVG ( [ DISTINCT | ALL ] Nom_colonne)**

→ Calcul la moyenne d'une colonne numérique.

**MAX | MIN (Nom_colonne)**

→ Calcul le minimum / maximum d'une colonne numérique

# la clause GROUP BY

- La clause de regroupement GROUP BY permet de partitionner une table. Chaque partition possède une même valeur pour une liste d'attributs spécifiés après le terme « GROUP BY ». Ceci est souvent utilisé avec des fonctions de calcul.

# Syntaxe de GROUP BY

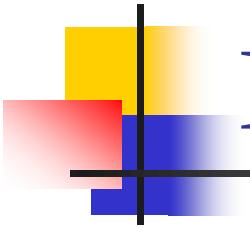
- SELECT colonne1, colonne2, ... colonne_n,
  fonction_de_calule (expression)
  FROM tables
  WHERE conditions
  GROUP BY colonne1, ..., colonne_n;
- *Fonction_de_calcule :* SUM, AVG, COUNT, MIN, MAX.

# Exemple

Requête :

*Nombre de personnes de nationalité différente :*

```
SELECT nationalité, COUNT(*) as totale
FROM personne
GROUP BY nationalité;
```



# Exemple

*Pour chaque Nationalité afficher le nombre de personnes qui ont un salaire supérieur à 1200 €.*

```
SELECT nationalité, COUNT(*) as Nbr
 FROM personne
 WHERE salaire > 1200
 GROUP BY nationalité;
```

# *la clause HAVING*

- La clause HAVING sélectionne un sous-ensemble de groupe.

- HAVING est toujours utilisée avec la clause GROUP BY.

# Syntaxe HAVING

SELECT colonne1, colonne2, ... colonne_n,
fonction_de_calule (expression)
FROM tables
WHERE conditions
GROUP BY colonne1, ...colonne_n
HAVING conditions;

# Exemple HAVING

*Afficher les nationalités dont la moyenne des salaires est supérieur à 1000 €.*

```
SELECT nationalité, AVG(salaire) as moyenne
 FROM personne
 GROUP BY nationalité
 HAVING AVG(salaire) >1000;
```