

# Bases de Données:

## Partie I : fondements du modèle relationnel

**Prof. Ladjel BELLATRECHE**  
**LISI/ISAE-ENSMA, Poitiers, France**

[bellatreche@ensma.fr](mailto:bellatreche@ensma.fr)

<http://www.lisi.ensma.fr/members/bellatreche>

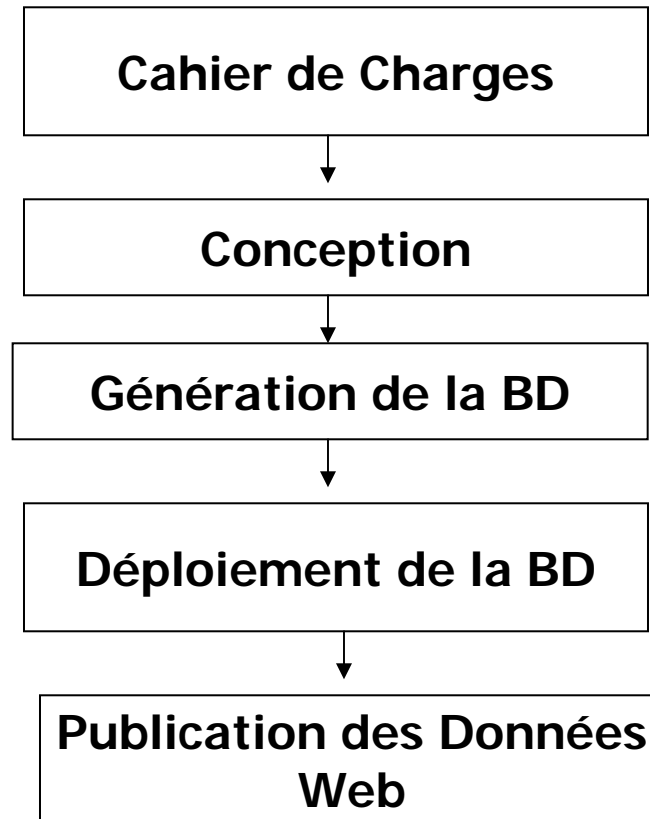
**Téléphone: 05 49 49 80 72**

# Objectifs & Plan



- Connaissances des modèles et outils concernant les bases de données (BD) relationnelles
- Exploitations d'une BD
  - o Algèbre relationnelle & arbres algébriques
  - o Langage SQL
- Conception d'une BD
  - o Modèle Entité Association

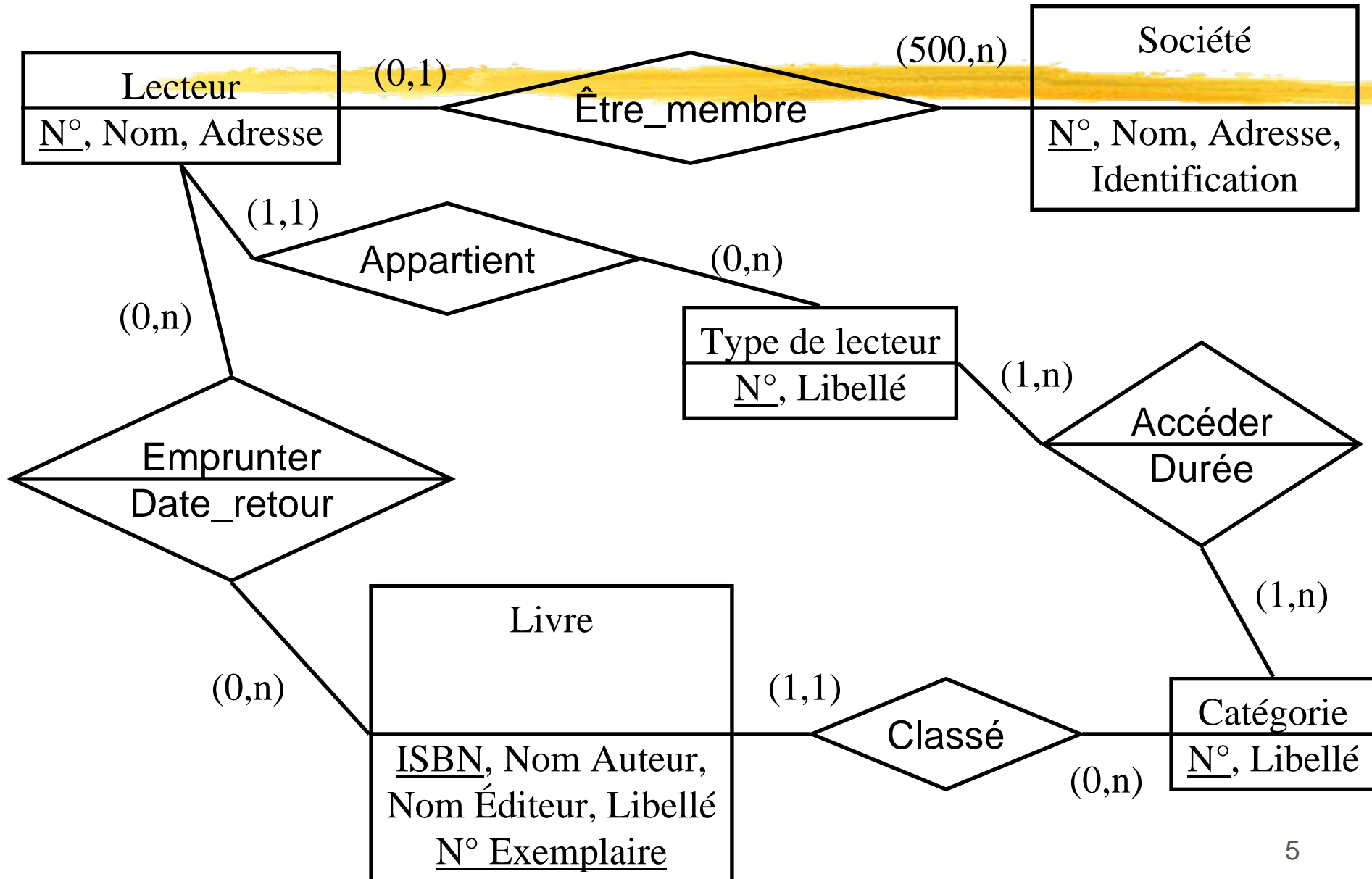
# Big Picture



# Cahier des Charges: Gestion Bibliothèque

- La bibliothèque enregistre chaque lecteur à qui elle donne un numéro de lecteur. Elle lui prend son nom et son adresse. Le lecteur peut éventuellement être membre d'une société adhérente. On enregistre alors l'identification de cette société.
- Un lecteur peut emprunter plusieurs livres chaque jour. A chaque prêt, on associe une « date de retour au plus tard ».
- Un lecteur appartient à un « type de lecteur ». Ce type lui permet d'avoir ou non accès à certaines catégories de livres.
- La durée du prêt dépend de la catégorie du livre et du type de lecteur. Elle est la même pour tous les livres d'une catégorie donnée empruntés par un quelconque lecteur d'un type donné.
- Un livre est caractérisé par son numéro d'inventaire. Il est nécessaire de connaître sa catégorie, le nom de son auteur, son éditeur, ainsi que le nombre de ses différents exemplaires disponibles. L'édition, lorsqu'elle existe, est également à connaître.
- La catégorie d'un livre se repère par un numéro et possède un libellé. Il en est de même pour le type de lecteur.
- Une société adhérente possède un nom et une adresse ; elle s'engage à envoyer un minimum de 500 lecteurs.

# Un exemple d'un Modèle Conceptuel



# Partie 1: fondements plan du cours



## **1 - GENERALITES**

Définitions des Bases de Données (BD)  
Systèmes de Gestion de BD (SGBD)  
Modèle architectural

## **2 – LE MODELE RELATIONNEL**

Fondements  
Algèbre des relations

## **3 - LE MODELE RELATIONNEL : le langage SQL**

Le langage d'interrogation  
Création de schémas relationnels  
Moteur SQL

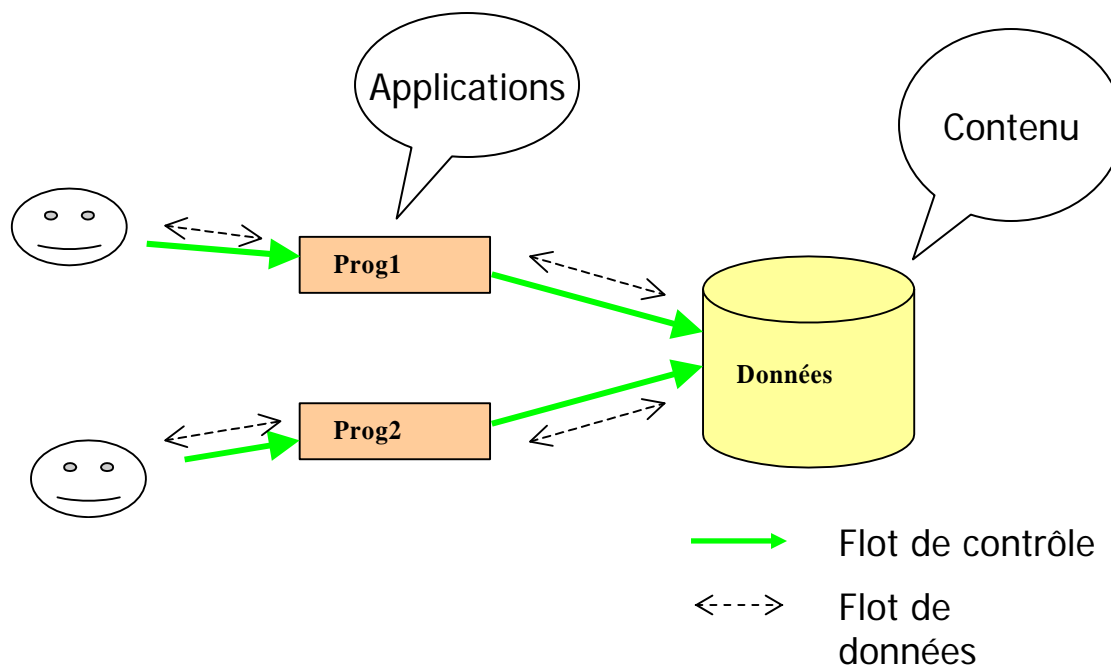
## **4 – ETUDE DE CAS**

Revue des notions abordées sur  
un exemple

# BD: généralités

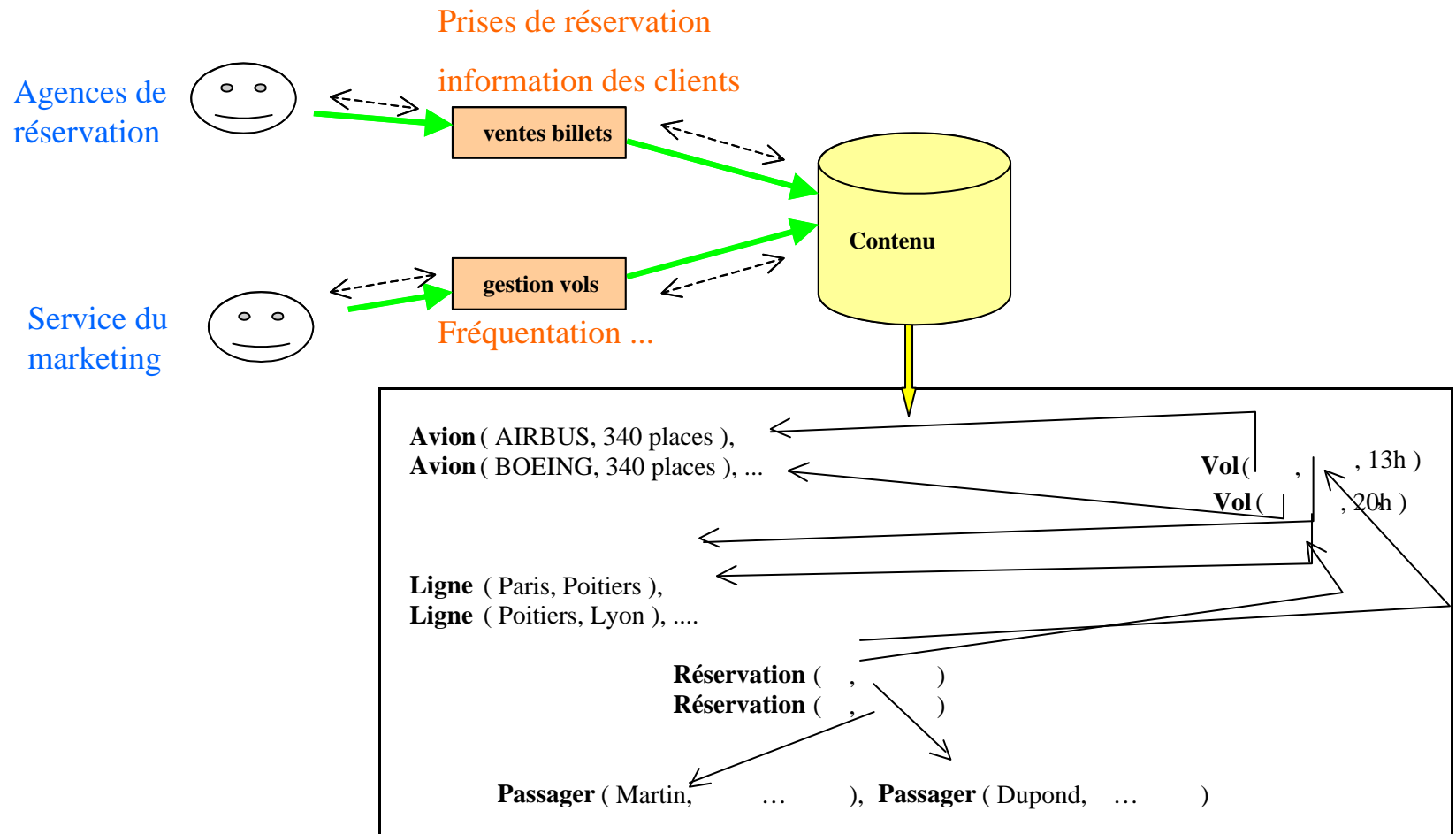
## *Une Définition*

Une BD est un ensemble structuré de données (→ organisation et description de données)  
enregistrées sur des supports accessibles par l'ordinateur (→ stockage sur disque)  
pour satisfaire simultanément plusieurs utilisateurs de manière sélective (→ partage des données & confidentialité)  
en un temps opportun (→ performance : passage à l'échelle)



# bd: généralités

## Exemple: compagnie aérienne





# Système de fichiers=base de données?



o Quelques problèmes ...

## 1- concurrence:

- Vous et un camarade éditez le même fichier,
- vous sauvez ensemble ce fichier en même temps,

Q: *quelles modifications restent?*

## 2- panne:

- vous éditez un fichier
- le courant s'éteint

Q: *que deviennent vos modifications?*

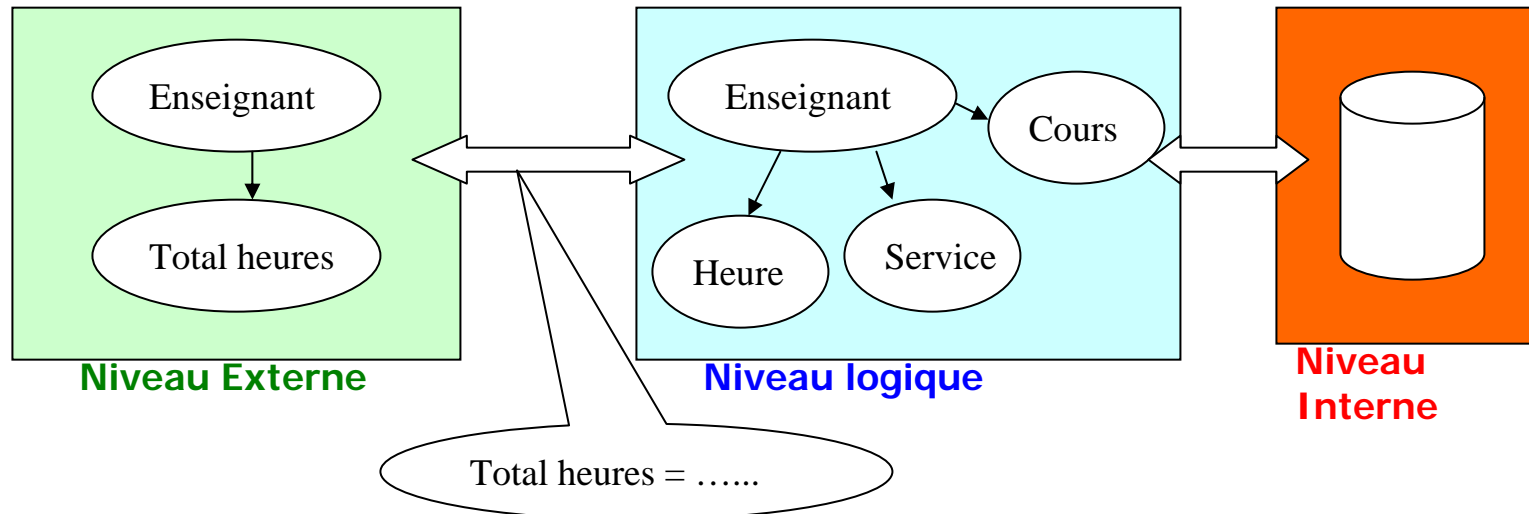
# bd: modèle architectural

## L'ARCHITECTURE ANSI-SPARC

**Objectif:** indépendance entre la structure de stockage des données et les programmes d'application

**Résultat:** architecture à 3 niveaux (ANSI 75)

- ✍ **Niveau Interne** *aspect physique et concret* de la base, constitué des fichiers et de leur organisation interne;
- ✍ **Niveau logique** : désigne la structure logique du contenu
- ✍ **Niveau Externe** Ensemble des vues extraites ou déduites du niveau logique

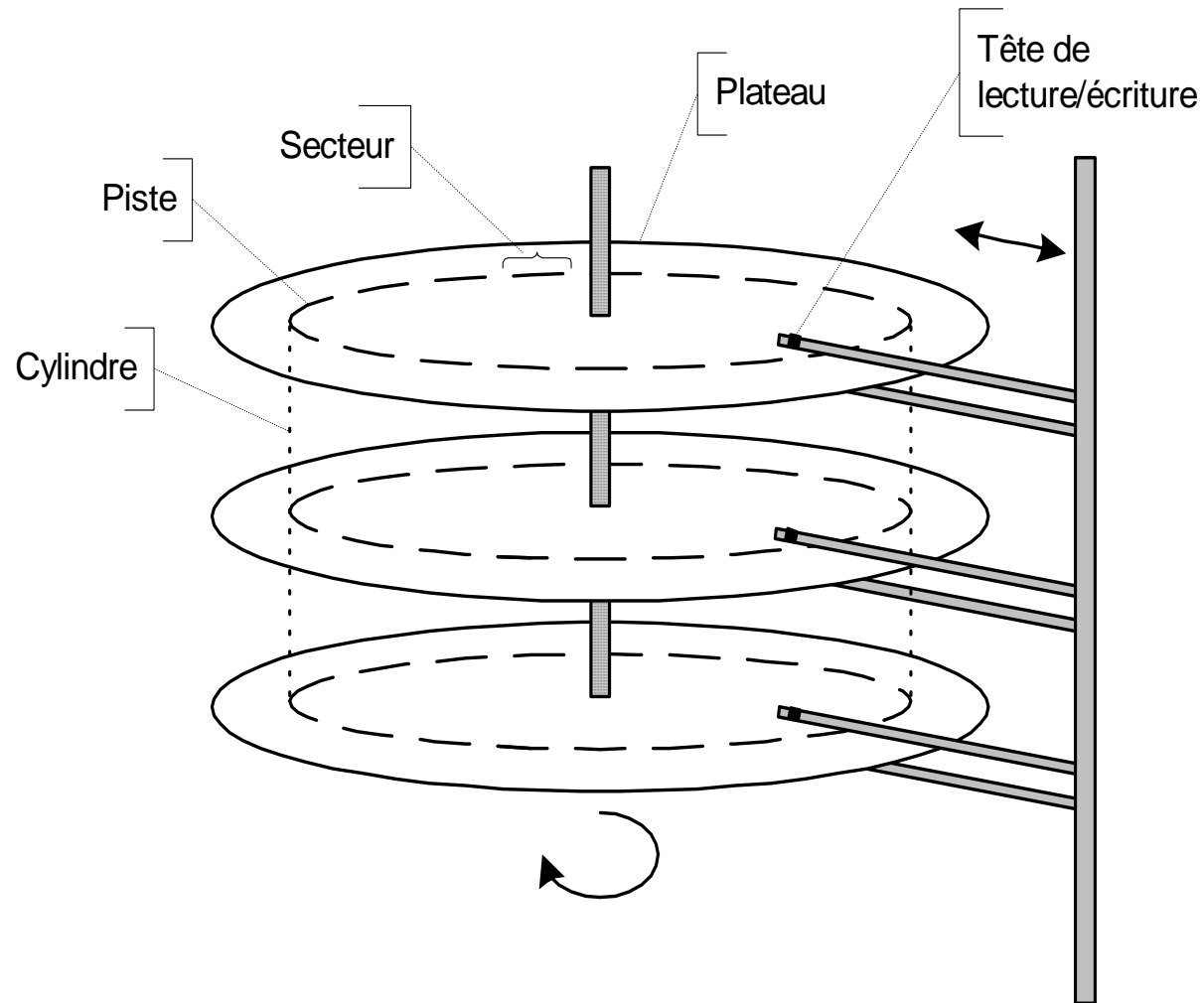


# Schéma interne



- Définit les chemins d'accès aux données
- Hypothèses générales :
  - BD est grande (e.g., General Motors >2 terabytes)
  - Stockée sur le disque
- Problèmes
  - Allocation d'espace disque
  - Accès direct aux enregistrement
  - Tampon Mémoire

# Structure d'un disque

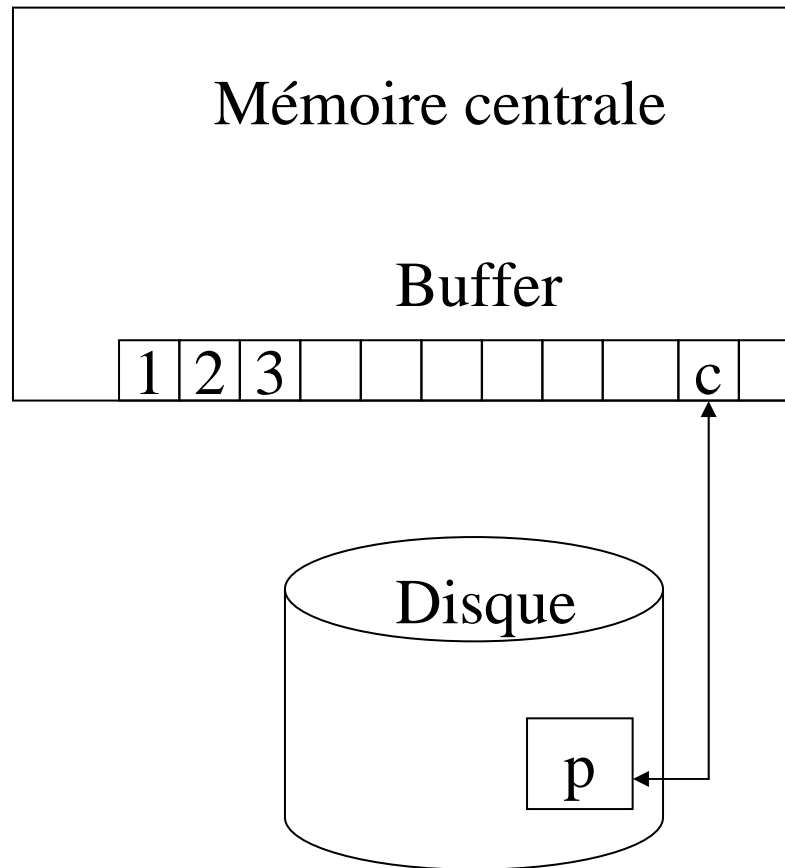


# Stockage physique des données



- Physiquement les instances d'une BD sont stockées dans un ou plusieurs **fichiers** (qui peuvent être répartis)
- Un fichier est stocké sur un **disque**
- Caractérisé par un nom et une suite de blocs (**pages**)
- Les instances sont rangées dans des pages
- ⌘ Echanges entre disque et mémoire centrale se font à travers d'une zone mémoire appelée **tampon** (buffer)
- ⌘ Prise en charge de ces échanges est gérée par le gestionnaire de tampon

# Structure d'un tampon



Un tampon est formé d'une suite de **cases** et chacune peut contenir une **page**

# Recherche d'une page

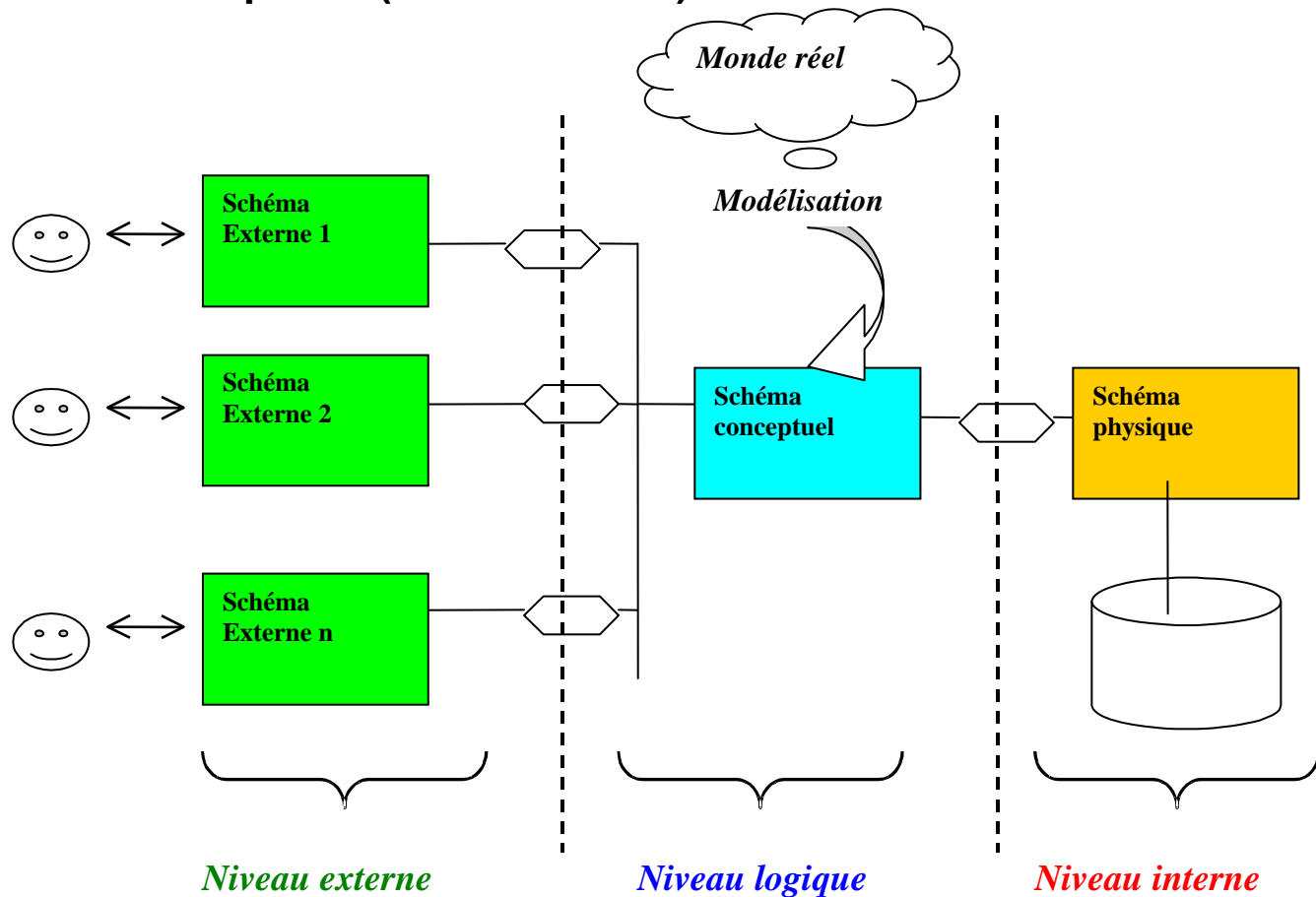


1. Si la page  $p$  est dans la case  $c$  du tampon, retourner  $c$  (économie un accès disque)
2. Si la page  $p$  n'est dans la tampon, il faut la lire sur le disque. On teste s'il existe une case libre pour la recevoir. Si oui, soit  $c$  cette case
3. Sinon, il faut libérer une case et donc renvoyer une page du tampon sur le disque  
→ LRU (Least Recently Used), FIFO (First In First Out)
4. Si la page à rejeter a été modifiée pendant son séjour en mémoire, il faut la réécrire sur le disque.
5. Transférer la page  $p$  du disque dans la case  $c$  et retourner  $c$ .

# bd: modèle architectural

## NOTION DE SCHEMA

On appelle **schéma d'une base de données** l'ensemble des informations descriptives (*méta-données*) du contenu de la base.

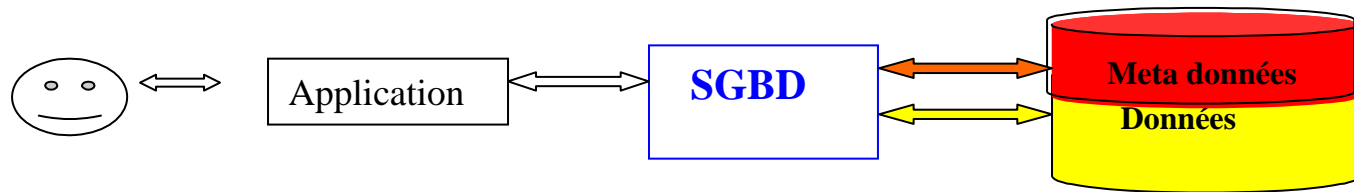




# bd: système de gestion de bd (SGBD)

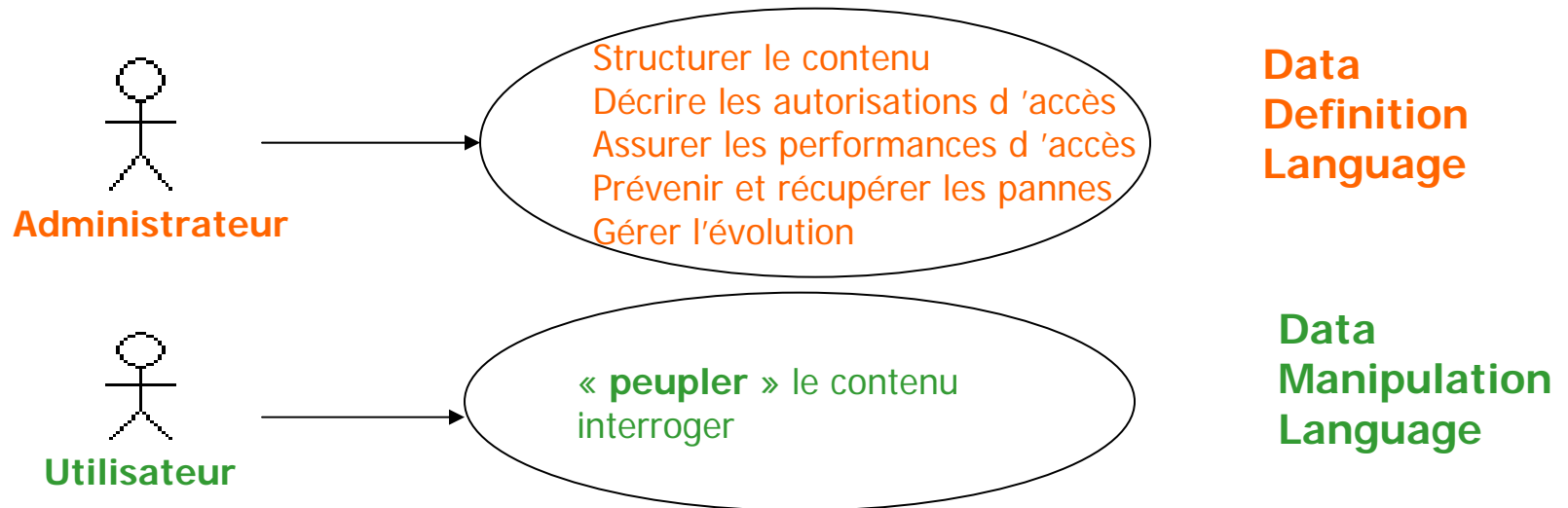
## Définition:

Logiciel réalisant l'interface entre les applications et le contenu de la base et permettant de gérer cette base



Gérer = *Construire, Utiliser, Maintenir, Réorganiser*

## Acteurs:



# bd: modèle architectural

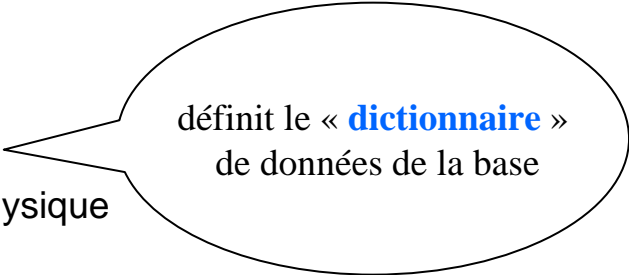
## OUTILS & MODELES

### OUTILS : LANGAGES ASSOCIÉS

#### ■ Le langage de Définition des Données (DDL):

permet de définir:

- le schéma logique de la base
- de préciser certains éléments d'implantation physique
- dans certains cas, des règles d'intégrité
- les schémas externes utilisables



définit le « **dictionnaire** »  
de données de la base

#### ■ Le langage de manipulation des données (DML):

offre, **en conformité avec le DDL**, les primitives:

- concernant l'insertion, la mise à jour, la suppression des données,
- la recherche de données

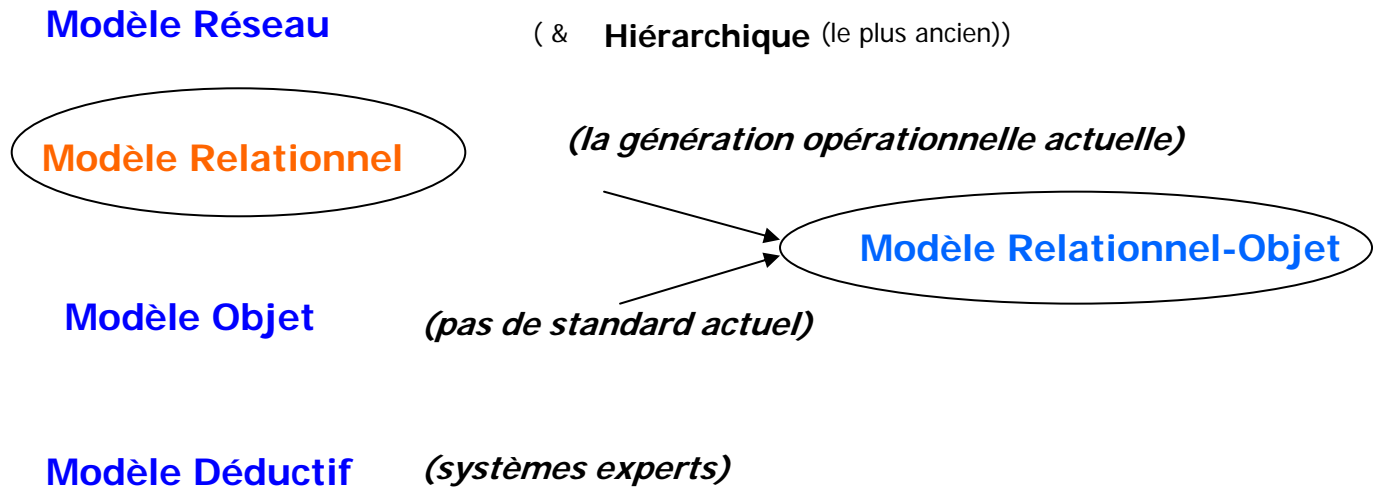
#### ✍ Le langage de contrôle des accès aux données : (DCL)

Ils sont **entièrement dépendants du type de SGBD**

# bd: modèle architectural

## EVOLUTION DES MODELES

### modèles : typologie



# bd: quelques systèmes relationnels

## Systèmes orientés « fichiers »

**Access** (Microsoft)

## Systèmes « client/serveur »

**Oracle** (Oracle Corporation) (>40% du marché)

**SQL Server** (Microsoft)

**Informix** (IBM)

**DB2** (IBM)

**Interbase** (Borland)

**Sybase**

**Postgres** (université de Berkeley)

**MySQL**

# modèle relationnel

## INTRODUCTION

Historique: *Codd* (1970)

Caractéristiques:

- *simplicité* des concepts manipulés,  
✍ bien pour les *utilisateurs*
- *indépendance* du stockage *physique*  
✍ bien pour les *programmeurs*
- *fondement mathématique* clair  
✍ bien pour les *théoriciens*



à la base de la plupart des SGBD actuellement commercialisés

# modèle relationnel

## CONCEPTS DE BASE

### Domaine

Un **domaine** désigne un ensemble de valeurs, caractérisé par un nom.

Exemple:     `NOMS_DE_PERSONNE = { Paul, Jean, Jacques, Michel.... }`  
                  `AGE = { Nombre_entier_positif }`

### Relation

Une **relation** **R** est un sous-ensemble du produit cartésien

$D = D_1 \times \dots \times D_n$ , où  $(D_i)$  sont des **domaines** (non nécessairement distincts).

✍ Un élément  $e = (e_1, \dots, e_n)$  de **R** est un **n-uplet**, ou "tuple"

Exemple:

La relation **PASSAGER** définie sur **NOMS\_DE\_PERSONNE**  $\times$  **AGE**,  
de contenu:

`PASSAGER = { (Paul, 36), (Jacques, 45), (Michel, 12) }`

# modèle relationnel

## CONCEPTS DE BASE

### Attribut d'une relation

La projection  $proj_{D_i} : R \rightarrow D_i$  est dénotée  $A_i$ .

Le couple  $(A_i : D_i)$  est dit attribut de  $R$

On notera alors la relation  $R[A_1, \dots, A_n]$ .

Exemple:

**PASSAGER** [ **Nom**, **Age**].

où,

domaine (**Nom**)  $\rightarrow$  NOMS\_DE\_PERSONNE

domaine (**Age**)  $\rightarrow$  AGE

### REPRÉSENTATIONS D'UNE RELATION

✍ Représentation en extension: table d'une Relation

<b>PASSAGER</b>	<b>Nom</b>	<b>Age</b>
<i>tuple_1</i>	Paul	36
<i>tuple_2</i>	Jacques	45
<i>tuple_3</i>	Michel	12

✍ Représentation en intention: schéma d'une Relation: *attributs* & *contraintes*

# modèle relationnel

## CONTRAINTES RELATIONNELLES

### DEFINITIONS:

#### ☞ clé d'une relation

Sous-groupe minimal des attributs de la relation permettant d'identifier de façon unique un tuple de cette relation

- Ce sous-groupe **X** peut ne pas être unique
- pour tout **e** de **R**, **X(e)** **identifie** **e** de façon unique

#### ☞ Valeurs nulles

*On appelle **NULL** la valeur conventionnelle permettant de représenter une information inconnue ou inapplicable*

#### Exemple:

Dans la relation

PERSONNE [ Num\_ident, Nom, Naissance, Décès ] ,

où:

Num\_ident ∈ CHAÎNE,

Nom ∈ NOMS\_DE\_PERSONNE

Naissance et Décès ∈ DATE

on peut avoir le tuple:

( 1.50.03.17.125.001, Michel, 7/03/50, NULL )

clé



# modèle relationnel

## CONTRAINTES RELATIONNELLES

### SCHEMA RELATIONNEL:

*Le schéma relationnel d'une base de données =  
schéma de chacune de ses relations +  
contraintes d'intégrité propres au modèle relationnel :*

☞ **contrainte d'ensemble:**

→ *Toute relation doit être munie d'une clé, appelée clé primaire.*

☞ **contrainte d'entité:**

→ *Toute valeur d'un attribut participant à la clé d'une relation doit être non nulle*

☞ **contrainte de « clé étrangère » :**

*clé étrangère* d'une relation = attribut de cette relation qui référence la clé primaire d'une autre relation.

→ *La règle d'intégrité qui en découle consiste à imposer que cette valeur apparaisse effectivement dans la relation cible (intégrité référentielle)*

# modèle relationnel

## EXEMPLE

**PASSAGER** [ Nom, Age ]  
**RESERVATION** [ Nom, Num\_vol, Num\_place ]  
**VOL** [ Num\_vol, Ident\_ligne, Date ]  
**LIGNE** [ Ident\_ligne, Départ, Arrivée ]  
**AEROPORT** [ Nom ]

Schéma

contenu

PASSAGER

<i>Nom</i>	<i>Age</i>
Jean	43
Michel	34

RESERVATION

<i>Nom</i>	<i>Num_vol</i>	<i>Num_place</i>
Michel	303	5
Michel	110	46

VOL

<i>Num_vol</i>	<i>Ident_ligne</i>	<i>Date</i>
110	10	23/02/98
303	12	26/02/98
122	10	03/03/98

AEROPORT

<i>Nom</i>
Paris
Tokyo
Berlin

LIGNE

<i>Ident_ligne</i>	<i>Départ</i>	<i>Arrivée</i>
10	Paris	Tokyo
12	Paris	Berlin

*Num\_vol* → clé primaire

→ → clé étrangère

# algèbre relationnelle



- **Modèle formel** permettant d'exprimer et de calculer les requêtes sur les relations.
- o Constituée d'un ensemble d'opérateurs algébriques. Les plus utilisés sont la sélection, la jointure, la projection et les opérateurs ensemblistes.
  - o Le résultat de l'exécution d'un opérateur est toujours une relation, ce qui permet la composition.
  - o Utilisée également pour l'optimisation de requêtes.
  - o Une requête algébrique peut se présenter sous forme d'un arbre algébrique.

# algèbre relationnelle

## OPERATIONS UNAIRES

**R** est une relation ayant pour schéma **R[A]** , **A** désignant la liste ses attributs.

### Projection $\pi$

**B** désigne un sous-ensemble de **A** :

$$T[B] = \pi_B(R) \text{ est défini par } \forall r \in R, \text{proj}_B(r) \in T$$

Extraction de colonnes

!!! La projection élimine les n-uplets identiques

Exemple:

**R[A1, A2, A3]**

<b>A1</b>	<b>A2</b>	<b>A3</b>
a1	b1	c1
a2	b1	c2
a3	b3	c3

$\pi_B(R) =$   
Si  $B = \{A2, A3\}$

<b>A2</b>	<b>A3</b>
b1	c1
b1	c2
b3	c3

Si  $B = \{A2\}$

<b>A2</b>
b1
b3

# algèbre relationnelle

## OPERATIONS UNAIRES

### Sélection $\sigma$ (ou restriction)

$T[A] = \sigma_{\text{exp\_select}}(R)$  est défini par:  $T = \{ t \in R \mid \text{exp\_select}(t) = \text{VRAI} \}$

Exemple:

$R[A] =$

Ident_ligne	Départ	Arrivée
10	Paris	Tokyo
12	Paris	Berlin
34	Londres	Berlin

Extraction de lignes

$\text{exp\_log} = (\text{Départ} = \text{'Paris'} \text{ ET } \text{Arrivée} = \text{'Berlin'})$

$\sigma_{\text{exp\_log}}(R) =$

Ident_ligne	Départ	Arrivée
12	Paris	Berlin

# algèbre relationnelle

## OPERATIONS BINAIRES

**Union**  $\cup$

$$T[A] = R[A] \cup S[A]$$

Exemple :

A1	A2
1	2
1	3
2	1

$\cup$

A1	A2
3	1
1	2

A1	A2
1	2
1	3
2	1
3	1

**Différence** -

$$T[A] = R[A] - S[A]$$

**Intersection**  $\cap$

$$T[A] = R[A] \cap S[A]$$

# algèbre relationnelle

## OPERATIONS BINAIRES

### Produit Cartésien X

$T[A+B] = R[A] \text{ X } S[B]$  est défini par:  $T = \{t=(r,s), r \in R \text{ et } s \in S\}$

Exemple :

<b>A1</b>	<b>A2</b>
1	2
1	3
2	1

X

<b>B1</b>	<b>B2</b>
a	b
c	d

<b>R.A1</b>	<b>R.A2</b>	<b>S.B1</b>	<b>S.B2</b>
1	2	a	b
1	2	c	d
1	3	a	b
1	3	c	d
2	1	a	b
2	1	c	d

# algèbre relationnelle

## JOINTURES

**$\theta$ - Jointure :**  $\otimes$

- $R[A]$  et  $S[B]$  désignent 2 relations,
- $\theta_i$  sont des opérateurs de comparaison(  $>$ ,  $<$ ,  $=$ ,  $<>$ ,  $<=$ ,  $>=$ , ... ),
- $C$  est une expression logique bâtie sur les opérateurs  $\theta_i$  et des attributs de  $R$  et  $S$

$$\begin{aligned} T[A+B] &= R[A] \otimes_c S[B] \\ &= \sigma_c (R \times S) \end{aligned}$$

Exemple:

**PASSAGER**

Nom	Age
Jean	43
Michel	34

**RESERVATION**

Nom	Num_vol	Num_place
Michel	303	5
Michel	110	46

**PASSAGER**  $\otimes$  **PASSAGER.Nom=RESERVATION.Nom** **RESERVATION**  
donne :

PASSAGER. Nom	Age	RESERVATION. Nom	Num_vol	Num_place
Michel	34	Michel	303	5
Michel	34	Michel	110	46



# algèbre relationnelle

## JOINTURES

### Jointure naturelle : \*

**Cas particulier important :** *jointure naturelle*, où  $\theta$  est l'égalité et le résultat a pour schéma  $A \cup B$ , i.e. *la fusion des champs qui portent le même nom*.

Notation :  $T = R * S$

Exemple:

PASSAGER

Nom	Age
Jean	43
Michel	34

RESERVATION

Nom	Num_vol	Num_place
Michel	303	5
Michel	110	46

( PASSAGER \* RESERVATION ) donne :

Nom	Age	Num_vol	Num_place
Michel	34	303	5
Michel	34	110	46

# algèbre relationnelle

## EXPRESSIONS

Une *expression relationnelle* est définie comme une *composition* des opérateurs définis ci-dessus.

### INTÉRÊT

a) *Expression* = *requête d'interrogation* sur la Base de Données.

"Quelles sont les vols reliant Paris et Berlin?"

$\sigma \left( (\text{Départ} = \text{'Paris'} \text{ ET Arrivée} = \text{'Berlin'}) \text{ OU } (\text{Départ} = \text{'Berlin'} \text{ ET Arrivée} = \text{'Paris'}) \right) (\text{VOL})$

"Quels sont les passagers ayant effectué une réservation?"

$(\text{PASSAGER} * \text{RESERVATION})$

Dans ce modèle, toute requête est ainsi formulée à partir de la construction d'une expression, qui prend en opérandes les relations dont on veut extraire les informations et retourne une relation résultat.

# algèbre relationnelle

## EXPRESSIONS

b) Exprimer *des contraintes* sous la forme :

*résultat d'expression* =  $\emptyset$  (relation vide)

Exemple: *un vol doit relier 2 aéroports distincts:*

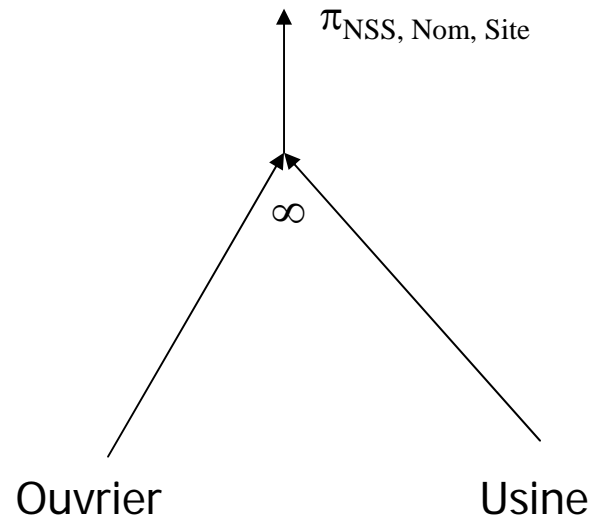
$\sigma_{\text{Depart}=\text{Arrivee}}(\text{VOL}) = \emptyset$

# Arbre algébrique

Un arbre algébrique est un arbre représentant une requête avec:

- Les nœuds feuilles sont les relations de base
- Les nœuds intermédiaires sont les opérateurs
- Le nœud racine est le résultat
- L'arc est un flux de données

Exemple: `SELECT NSS, Nom, Site FROM Ouvrier O, Usine U WHERE O.N°U = U.N°U;`



# le langage SQL

## INTRODUCTION

Ce qu'on va voir:

- le **DML** SQL: interrogations, insertion, suppression, maj de tuples,
- le **DDL** SQL: définition du schéma
- le **DCL** (*Data Control Language*)SQL: gestion des transactions, droits d'accès,

### Historique:

premier langage : **SEQUEL** (IBM)

→ a donné le langage **SQL** (**Structured Query Language**) :

- *standard de l'ANSI* en 1986.
- évolution **SQL2** (SQL 92)
- dernière version : **SQL3** (SQL 99).

### Caractéristiques:

- **fondement théorique = l'Algèbre Relationnelle**
- **Objectif du langage : avoir une expression syntaxique aussi naturelle que possible.**
- **Langage assertionnel**

# le langage SQL

## QUELQUES TYPES DE VALEUR

- le type *chaîne de caractères*: de longueur fixe **CHAR(n)** ou variable **VARCHAR(n)**
- les types *numérique exact*: **NUMERIC** et ses sous-types: **DECIMAL(n,d)**,  
**INTEGER**, **SMALLINT**
- Le type *numérique approché*: **FLOAT** et ses sous-types: **REAL** et **DOUBLE PRECISION**
- le *type chaîne de bits*: de longueur fixe **BIT(n)** ou variable **BIT VARYING(n)**
- les types *date* **DATE** et *heure* **TIME**:  
exemple : '1999-03-09', '08 :00 :00.0'

VALEUR CONVENTIONNELLE : **NULL**

# le langage SQL

## INTERROGATIONS

!!!

algèbre relationnelle = manipule des relations (sens ensembliste)

SQL manipule des **tables pouvant contenir des duplicata**

### LA COMMANDE SELECT

```
SELECT <spécification de la table résultat>  
FROM {<table_opérande>}  
[ WHERE {<prédicat_de_sélection>} ]  
[<spécification_d_aggrégat>]  
[<spécification_de_présentation>]
```

$\pi_{\langle \text{spécification\_de\_la\_table\_résultat} \rangle}$

$\sigma_{\langle \text{prédicat\_de\_sélection} \rangle}$

$(\langle \text{table\_opérande1} \rangle \bowtie \langle \text{table\_opérande2} \rangle \bowtie \dots)$

# le langage SQL

## INTERROGATIONS

### Clause SELECT

**Les clauses FROM et WHERE fournissent une table résultat:**

SELECT permet de restreindre cette table à des colonnes choisies ou de faire des calculs à partir des colonnes cette table

```
SELECT
  (1) [DISTINCT] {
  (2)          nom_de_table . * |
  (3)          { expression [as nom_de_colonne_resultat ] },
  }
```

- 1) DISTINCT permet d'éliminer les duplicata dans la table résultat,
- 2) La table résultat comporte toutes les colonnes de la table spécifiée
- 3) décrit une colonne obtenue, avec son nom éventuel, par le résultat du calcul spécifié par l'**expression** donnée



# le langage SQL

## INTERROGATIONS

### Clause FROM

Renommage local  
de la table

```
Table_opérande ::= (1) { {nom_de_table [ as alias] },  
                        | (2) nom_de_table JOIN nom_de_table ON condition
```

```
SELECT PILOTE . *  
FROM PILOTE
```

*La table résultat est une copie de la table PILOTE*

Condition de jointure

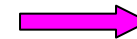
```
SELECT PILOTE . Nom_P, PILOTE . Adresse  
FROM PILOTE
```

*La table résultat comporte les 2 colonnes indiquées de la table PILOTE*

```
SELECT DISTINCT AVION. Base  
FROM AVION
```

table **AVION**:

Ident A	Type	Capacité	Base
100	A340	300	Paris
101	B707	500	Lyon
102	FALCON	15	Lyon



Base
Paris
Lyon

# le langage SQL

## INTERROGATIONS

```
SELECT COUNT(PILOTE) as Total , MAX(PILOTE. Salaire) as Max_salaire
FROM PILOTE
```

table **PILOTE**:

<b>Ident_P</b>	<b>Nom_P</b>	<b>Adresse</b>	<b>Salaire</b>
1	Paul	Paris	12 000
2	Jean	Nice	13 500
3	Roger	Caen	8 000
4	Jean	Lyon	8 500
5	Michel	Lyon	14 000



<b>Total</b>	<b>Max_salaire</b>
5	14 000

```
SELECT PILOTE. Nom_P AS Pilote_en_service
FROM PILOTE JOIN VOL ON PILOTE.Ident_P=VOL.Ident_P
```

Evaluation :

- Calcul de **PILOTE X VOL**,
- **Sélection** sur la colonne **Ident\_P**,
- **Projection** sur la colonne **Nom\_P** avec **renommage**:



<b>Pilote en service</b>
Paul
Jean
Roger
Michel

# le langage SQL

## INTERROGATIONS

### Clause WHERE

```
prédicat_de_sélection ::= (<prédicat_selection>
| <prédicat_selection>< AND | OR >< prédicat_selection>
| NOT <prédicat_selection>
| <prédicat>
```

```
prédicat ::= <prédicat_simple>
| <prédicat_in>
| <prédicat_null>
| <prédicat_between>
| <prédicat_like>
| <prédicat_exists>
```

<prédicat\_simple>

```
SELECT PILOTE . *
FROM PILOTE
WHERE Salaire > 10 000
```

Pilotes dont le salaire est > 10 000 ? →

Ident P	Nom P	Adresse	Salaire
1	Paul	Paris	12 000
2	Jean	Nice	13 500
5	Michel	Lyon	14 000

# le langage SQL

## INTERROGATIONS

<prédicat\_simple>

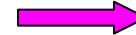
```
SELECT Num_vol as numéro, Départ as Ville
FROM AVION, VOL
WHERE AVION.Base = VOL.Départ
```

table **AVION**:

Ident A	Type	Capacité	Base
100	A340	300	Paris
101	B707	500	Lyon
102	FALCON	15	Lyon

table **VOL**:

Num vol	Ident A	Ident P	Départ	Arrivée
V308	100	1	Paris	Nice
V112	101	2	Paris	Lyon
V144	101	3	Lyon	Marseille
VSP0	102	1	Paris	Marseille
V134	100	5	Nice	Paris



numéro	Ville
V308	Paris
V144	Lyon

Evaluation :

- 1) Calcul de **AVION X VOL**,
- 2) Sélection indiquée,
- 3) **Projection** sur les colonnes indiquées avec **renommage**:

# le langage SQL

## INTERROGATIONS

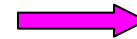
<prédicat\_simple>

```
SELECT Nom_P  
FROM PILOTE as P1, PILOTE as P2  
WHERE P1.Adresse = P2.Adresse
```

*Nom des pilotes ayant la même adresse?*

table **PILOTE**:

Ident P	Nom P	Adresse	Salaire
1	Paul	Paris	12 000
2	Jean	Nice	13 500
3	Roger	Caen	8 000
4	Jean	Lyon	8 500
5	Michel	Lyon	14 000



Nom P
Jean
Michel

Auto jointure

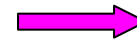
# le langage SQL

## INTERROGATIONS

### <prédicat\_in\_énumération>

```
SELECT Base
FROM AVION
WHERE Type IN ( 'A340', 'B707' )
```

*Base des avions dont le type est A340 ou B707?*

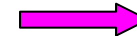


Base
Paris
Lyon

### <prédicat\_in\_sous-question>

```
SELECT Nom_P
FROM PILOTE
WHERE PILOTE.Ident_P IN
( SELECT Ident_P
  FROM VOL
  WHERE VOL.Départ = 'Paris'
```

*Nom des pilotes assurant un vol au départ de Paris?*



Nom P
Jean
Paul

# le langage SQL

## INTERROGATIONS

<prédicat\_exists\_sous-question>

```
SELECT *  
FROM T1  
WHERE EXISTS  
  (SELECT *  
   FROM T2  
   WHERE T1.Ident = T2.Ident)
```

*sous-question  
exécutée pour chaque ligne de T1*

table T1:

Ident	Nom
1	Paul
2	Jean
3	Roger
4	Jean
5	Michel

table T2:

Ident	Adresse
1	Paris
6	Poitiers
3	Lille
7	Lyon



table résultat:

Ident	Nom
1	Paul
3	Roger

# le langage SQL

## INTERROGATIONS

spécification\_d\_agrégat



Lorsque le *SELECT* utilise des fonctions « agrégat »: **MAX, MIN, SUM, COUNT, ...**

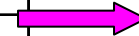
```
GROUP BY {<colonne>}  
[ HAVING <prédicat de sélection> ]
```

```
SELECT Depart, COUNT(*) AS Nombre_vols  
FROM VOL  
GROUP BY Depart
```

« Donner, pour chaque ville Départ, le nombre de vols assurés »

table **VOL**:

Num vol	Ident A	Id P	Départ	Arrivée	H_départ
V308	100	1	Paris	Nice	12
V112	101	2	Paris	Lyon	15
V144	101	3	Lyon	Marseille	16
VSP0	102	1	Paris	Marseille	1
V134	100	5	Nice	Paris	5



Depart	Nombre_vols
Paris	3
Lyon	1
Nice	1



# le langage SQL

## CREATION/SUPPRESSION DE TUPLES

### Insertion de tuples

```
INSERT INTO nom_de_table (liste_attributs_à_valuer)
VALUES (liste_des_valeurs)
```

```
INSERT INTO PILOTE (Ident_P,nom_P) VALUES (7,'Charles')
```

table **PILOTE**:

Ident P	Nom P	Adresse	Salaire
7	Charles	NULL	NULL

```
INSERT INTO nom_de_table (liste_attributs_à_valuer)
<Expression_de_sélection>
```

### suppression de tuples

```
DELETE FROM nom_de_table
WHERE <prédicat_de_sélection>
```

```
DELETE FROM VOL WHERE Num_vol = 'V308'
```

### modification de tuples

```
UPDATE nom_de_table SET {nom_atribut=valeur},
WHERE <prédicat_de_sélection>
```

```
UPDATE RESERVATIONS SET Num_vol='V112'
WHERE Num_vol='V308'
```

# le langage SQL

## CREATION DU SCHEMA (DDL)

```
schéma ::= CREATE SCHEMA ( [ { <definition_de_domaine> } ]  
                           [ <definition_de_table> }  
                           [ { <definition_de_vue> } ]  
                           [ { <contraintes_globales> } ] )
```

### Domaines

```
definition_de_domaine ::=  
    CREATE DOMAIN nom_de_domaine  
    AS <type_de_base> [DEFAULT valeur]  
    [<contrainte_de_domaine>] ;  
contrainte_de_domaine ::= CHECK (VALUE <op> constante)
```

```
CREATE DOMAIN numéro AS SMALLINT  
CHECK (VALUE>0) ;
```

**Définition du  
domaine de  
valeurs numéro  
(entier>0)**

**!!! Très rarement implémenté**

# le langage SQL

## CREATION DU SCHEMA (DDL)

### Tables

```
definition_de_table ::= CREATE TABLE nom_de_la_table  
    (  
        { <definition_de_colonne> },  
        { <contrainte_de_clé> },  
        [ { <contrainte_unicité> } ],  
        [ { <contraintes_locales> } ]  
    );
```

```
definition_de_colonne ::= nom <type> [NOT NULL [UNIQUE]]
```

```
contrainte_de_clé : := <clé_primaire> [<clé_étrangère>]
```

```
clé_primaire : := PRIMARY KEY( {nom_de_colonne}, )
```

```
clé_étrangère : := FOREIGN KEY( nom_de_colonne ) REFERENCES  
    nom_de_table(nom_de_colonne)
```

```
contrainte_unicité ::= UNIQUE( {nom_de_colonne} )
```

```
contraintes_locales : := CHECK ( <expression_logique> )
```

# le langage SQL

## CREATION DU SCHEMA (DDL)

```
CREATE SCHEMA (  
  CREATE DOMAIN numéro AS SMALLINT  
    CHECK (VALUE>0) ;  
  CREATE TABLE PILOTE  
    (  
      Ident_P  numéro ,  
      Nom_P   VARCHAR(20) NOT NULL ,  
      Adresse VARCHAR(80) ,  
      Salaire DECIMAL(2,10) ,  
      PRIMARY KEY(Ident_P)  
    ) ;  
  CREATE TABLE AVION  
    (  
      Ident_A  SMALLINT ,  
      Type    VARCHAR(12) NOT NULL ,  
      Capacité INTEGER ,  
      Base    VARCHAR(18) ,  
      PRIMARY KEY(Ident_A)  
    ) ;  
  CREATE TABLE VOL  
    (  
      Num_vol  VARCHAR(10)  
      Ident_A  SMALLINT ,  
      Ident_P  numéro ,  
      Départ  VARCHAR(18) ,  
      Arrivée  VARCHAR(18) ,  
      Heure_départ TIME ,  
      PRIMARY KEY(Num_vol) ,  
      FOREIGN KEY Ident_A REFERENCES AVION(Ident_A) ,  
      FOREIGN KEY Ident_P REFERENCES PILOTE(Ident_P) ,  
      CHECK (Départ <> Arrivée)  
    ) ;  
  ....  
)
```

*Clé primaire*

*Clé étrangère*

*Contrainte locale*

# le langage SQL

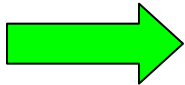
## CREATION DU SCHEMA (DDL)

### Vues

- Vue** =
- relation virtuelle, *résultat d'une requête* sur les tables de base et/ou les autres vues
  - utilisable comme une table normale

```
definition_de_vues ::= CREATE VIEW nom_de_vue ({<nom_de_colonne>},)  
                      AS <question>
```

```
CREATE VIEW VOL_AIRBUS ( Numero, Départ, Arrivée )  
  AS SELECT Num_vol, Départ, Arrivée  
     FROM VOL  
     WHERE Ident_A IN  
           ( SELECT Ident_A FROM AVION  
             WHERE Ident_A = 'A340' )
```



Cette notion est très importante:  
elle définit les *schémas externes visibles*

# le langage SQL

## CREATION DU SCHEMA (DDL)

### Contraintes globales

```
contraintes_globales : := CREATE ASSERTION nom CHECK (
                        <expression_logique> )
```

```
CREATE ASSERTION regle_de_depart CHECK (
  NOT EXISTS
  (SELECT PILOTE.Ident_P
   FROM VOL, PILOTE
   WHERE
     VOL.Ident_P = PILOTE.Ident_P
   AND
     VOL.Depart <> PILOTE.Adresse
  )
)
```

# « Moteur » SQL

