

VERNOTTE Marie  
BAC1-IR



# SYNTHÈSE : BASES DE DONNÉES



Profs : BEIREKDAR Abdo, SMAL Anne

# Table des matières

1. Introduction .....	5
1.1. Historique .....	5
1.2. Les bases de données .....	5
o Caractéristiques d'une BdD .....	5
1.3. S.G.B.D.....	6
2. Schéma conceptuel.....	6
2.1. Comment faire une bonne base de données ? .....	6
2.2. But du schéma conceptuel .....	6
2.3. Formalisme.....	6
2.4. Entités .....	7
o Notation : .....	7
o Identifiant : .....	7
o Attribut facultatif : .....	7
o Occurrence : .....	8
2.5. Relation.....	8
o Notation : .....	8
o Cardinalités : .....	9
o Exemple : .....	10
2.6. Résumé.....	10
2.7. Etapes .....	10
2.8. Contraintes d'intégrité.....	11
2.9. Normalisation .....	11
o 1FN .....	11
o 2FN.....	12
o 3FN.....	12
o 4, 5 et 6FN : .....	12
o Redondance : .....	12
3. Schéma relationnel.....	13

3.1. BD relationnelles .....	13
o Table .....	13
o Ligne .....	13
o Colonne .....	13
o Valeur .....	14
▪ Valeur particulière : valeur null .....	14
o Notion d'ordre .....	14
o Clé primaire .....	14
3.2. Transformation .....	15
3.3. Représentation schéma relationnel .....	15
3.4. Traduire les relations .....	15
o Clé étrangère .....	16
o Règles .....	16
▪ 1-N .....	16
▪ 1-1 .....	17
▪ N-N .....	17
4. SQL .....	18
4.1. Convention de nommage .....	18
4.2. Data definition language .....	18
o Création .....	18
o Contraintes .....	19
▪ Contraintes sur une colonne .....	19
▪ Contraintes sur une table .....	19
▪ FK simple .....	20
▪ FK multiple .....	20
o Exemple .....	20
o Modification .....	21
o Suppression .....	21
4.3. Data manipulation language .....	22
o Insertion de données .....	22
o Modification de données .....	22
o Suppression de données .....	22

4.4. Data query language.....	23
o Sélections .....	23
▪ Opérateurs .....	23
▪ Fonctions .....	24
▪ Fonctions de groupe .....	24
▪ GROUP BY .....	25
▪ HAVING .....	25
o Jointures.....	26
▪ Exemple .....	27
▪ Jointures réflexives .....	27
o Requêtes imbriquées.....	28
▪ EXISTS .....	29

# Synthèse : Bases de données

## 1. Introduction

### 1.1. Historique

Comment stocker des données ?

- Cartes perforées
- Bandes magnétiques
- Fichiers :
  - ⇒ Problème de partage
  - ⇒ Problème de relation
  - ⇒ Problème de sécurité
- Systèmes transactionnels (transaction)
- Système documentaires
- Systèmes de gestion des bases de données.

### 1.2. Les bases de données

Une base de données c'est un moyen de stocker des données

- **Donnée** : Enregistrement dans un code donnée d'un objet, un texte, un concept, un fait..  
En vue de transmettre ou stocker de l'information, interpréter ou effectuer un traitement par l'homme ou la machine, ou encore en déduire de nouvelles informations.

#### o Caractéristiques d'une BdD

- **Relation entre les données.**
- Sauvegarde des données sur un support.
- **Partage des données possibles entre utilisateurs.**
- **Indépendant des applications.**
- **Sans redondance inutile.**
- Contrôle de cohérence.
- Exploitation des données par interrogation.
- Longueur des enregistrements variable.
- Modification possible de la structure.

### 1.3. S.G.B.D

Un Système de Gestion de Base de Données (SGBD) est un système qui gère et fait toutes les tâches de maintenance d'une DB. C'est l'intermédiaire entre l'utilisateur et DB.

C'est un ensemble de services permettant de gérer des bases de données :

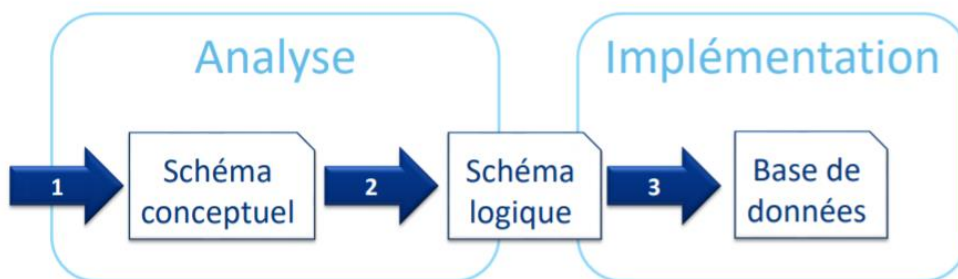
- Accès : Local et Réseau
- Droits : Authentifier et autoriser
- Manipulation : Créer, Modifier et Supprimer

Que doit gérer un SGBD ?

- Accès optimal à toutes les données.
- Traitement simultané des données.
- Validité et cohérence des données.
- Sécurité des données.
- Sauvegarde et récupération.

## 2. Schéma conceptuel

### 2.1. Comment faire une bonne base de données ?



Il faut suivre à la lettre ces étapes.

### 2.2. But du schéma conceptuel

Le schéma conceptuel permet de modéliser un système. On y définit les concepts et les liens qui les unissent → C'est une description du monde réel.

### 2.3. Formalisme

Il existe plusieurs formalismes possibles, dans ce cours se sont les diagrammes entité-relation.

## 2.4. Entités

**Entité** : chose qui existe dans le monde réel

- **Concret** : Une personne, une voiture,...
- **Abstrait** : Un sentiment, une organisation

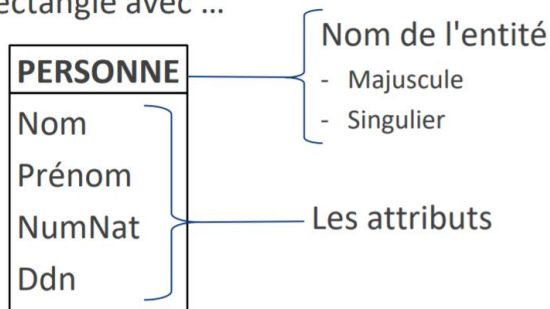
Une entité possède des attributs (caractéristique d'une entité).

**Exemple :**

- **Personne** = nom + prénom + numéro national + ddn +...

### o Notation :

Un rectangle avec ...



### o Identifiant :

Parmi tous les attributs, certains jouent un rôle particulier → celui d'identifier



Un attribut est un **identifiant** pour une entité si sa valeur est distincte pour chaque occurrence d'entité

### o Attribut facultatif :

Si quelqu'un n'a pas de numéro de téléphone ou d'adresse mail, il est possible de mettre l'attribut en facultatif en mettant [0-1] derrière.

PERSONNE
Nom
Prénom
<u>NumNat</u>
Ddn
Tel [0-1]
Mail [0-1]

o Occurrence :

**Occurrence** : Un élément de l'entité

Une occurrence est caractérisée par les valeurs données aux attributs :

**Titre** = Le Petit Prince

**Année** = 1943

**Auteur** = Antoine de Saint-Exupéry

**NbPages** = 93



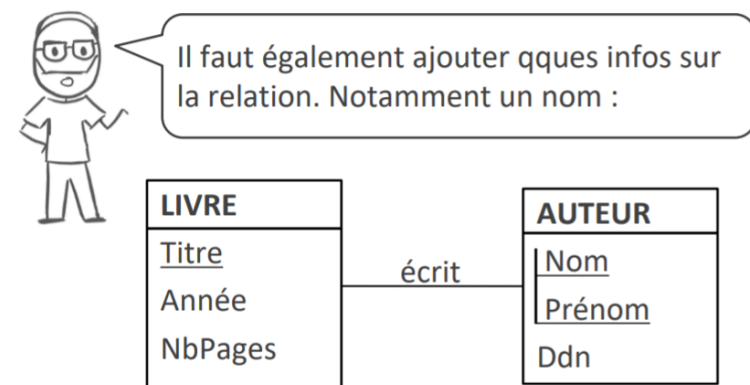
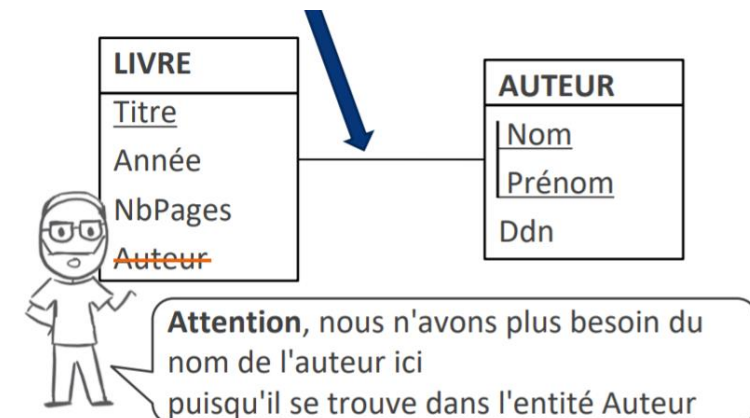
est une  
occurrence de

LIVRE
<u>Titre</u>
Année
NbPages
Auteur

2.5. Relation

**Relation** : Lien entre des entités

o Notation :





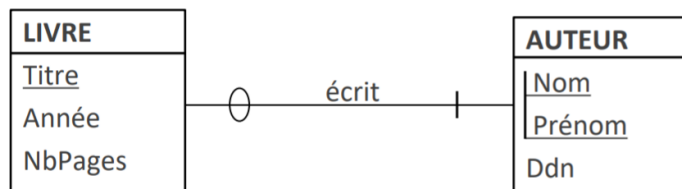
o Cardinalités :

Nombre minimum et maximum d'occurrences d'une identité dans une association.

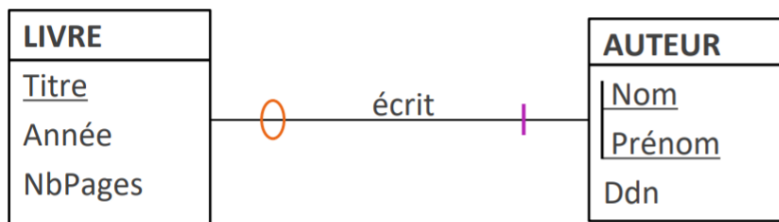
• **Cardinalité minimum : 0/1**

- **0** : on autorise le cas d'occurrences de l'entité considérée qui ne soient pas reliées à l'association.
- **1** : obligation de relier toutes les occurrences de l'entité à l'association

- 0 ➡ 0  
- 1 ➡ |



Comment le lire ?



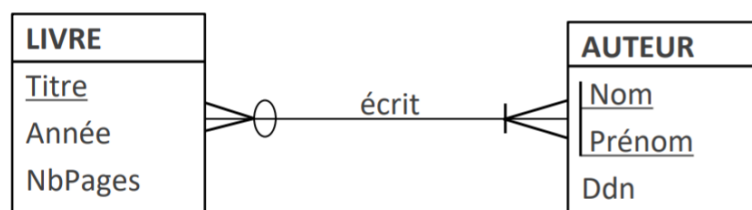
Un livre est écrit par **au moins 1** auteur

Un auteur a écrit **au moins 0** livres









• **Cardinalité maximum : 1/n**

- **1** : seule occurrence de l'association
- **N** : on autorise le cas d'occurrences de l'entité considérée qui soient éventuellement reliées, chacune, à plusieurs occurrences de l'association.

- 1 ➡ |  
- N ➡ <=

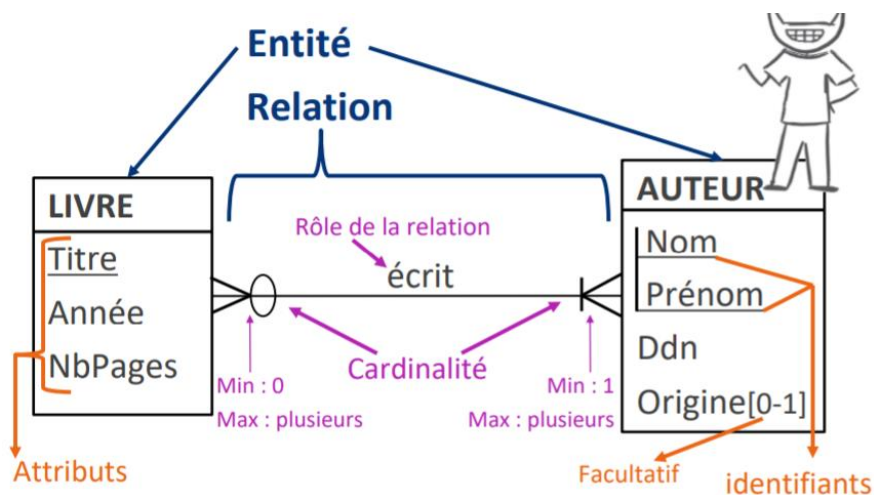


**Combinaisons possibles :**

Au plus 1 (0 ou 1) :		ou	
Exactement 1 :		ou	
0 ou plus :		ou	
1 ou plus :		ou	

o Exemple :

Voir l'exemple dans le cours de la page 25 à la page 29.

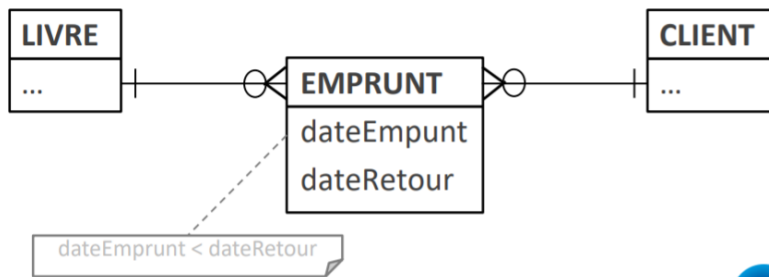
2.6. Résumé2.7. Étapes

Pour élaborer un schéma conceptuel, il existe plusieurs méthodes.. en voilà une :

1. Identifier les entités
2. Identifier les relations
3. Identifier les attributs (et id)
4. Décorer les relations (cardinalité)
5. Donner d'éventuelles contraintes d'intégrité
6. Vérifier :
  - La cohérence du schéma
  - La normalisation.

## 2.8. Contraintes d'intégrité

Ce sont des conditions que le diagramme doit respecter pour bien refléter la réalité → Sous forme de commentaires.



## 2.9. Normalisation

Il s'agit d'un ensemble de bonnes pratiques :

- Permet d'éviter :
  - Les contre-performances
  - La redondance d'information
  - Les anomalies transactionnelles
- Permet d'améliorer :
  - Les mises à jours
  - La maintenance
  - L'évolution

Il existe plusieurs niveaux de normalisation, appelés formes normales. On en parle juste après.

### o 1FN

- Tous les attributs dépendent de l'identifiant
- L'ordre des lignes/colonnes n'a pas d'importance
- Chaque ligne est unique
- Tous les attributs ont une valeur atomique
  - Exemple non atomiques :

→ Adresse = rue / n° / localité

→ Auteurs = Bouraada, Peten, Smal

Produit	Fournisseur
téléviseur	VIDEO SA, HITEK LTD

➔

Produit	Fournisseur
téléviseur	VIDEO SA
téléviseur	HITEK LTD

o 2FN

- 1FN +
- Un attribut ne peut pas dépendre d'une seule partie de l'identifiant.

Exemple : Si l'identifiant est « Produit » + « Fournisseur », l'attribut « Adresse fournisseur » ne peut pas dépendre uniquement de « Fournisseur »

Produit	Fournisseur	Adresse fournisseur
téléviseur	VIDEO SA	13 rue du cherche-midi
écran plat	VIDEO SA	13 rue du cherche-midi
téléviseur	HITEK LTD	25 Bond Street



Produit	Fournisseur
téléviseur	VIDEO SA
téléviseur	HITEK LTD
écran plat	VIDEO SA

Fournisseur	Adresse fournisseur
VIDEO SA	13 rue du cherche-midi
HITEK LTD	25 Bond Street

o 3FN

- 2FN +
- Aucun attribut ne peut dépendre d'un autre attribut à l'exception de l'identifiant.

Exemple : Le pays dépend directement de la ville, et pas du Fournisseur qui est l'identifiant

Fournisseur	Adresse fournisseur	Ville	Pays
VIDEO SA	13 rue du cherche-midi	PARIS	FRANCE
HITEK LTD	25 Bond Street	LONDON	ENGLAND

Fournisseur	Adresse fournisseur	Ville
VIDEO SA	13 rue du cherche-midi	PARIS
HITEK LTD	25 Bond Street	LONDON

Ville	Pays
PARIS	FRANCE
LONDON	ENGLAND



## ➔ Dépendance fonctionnelle

o 4, 5 et 6FN :

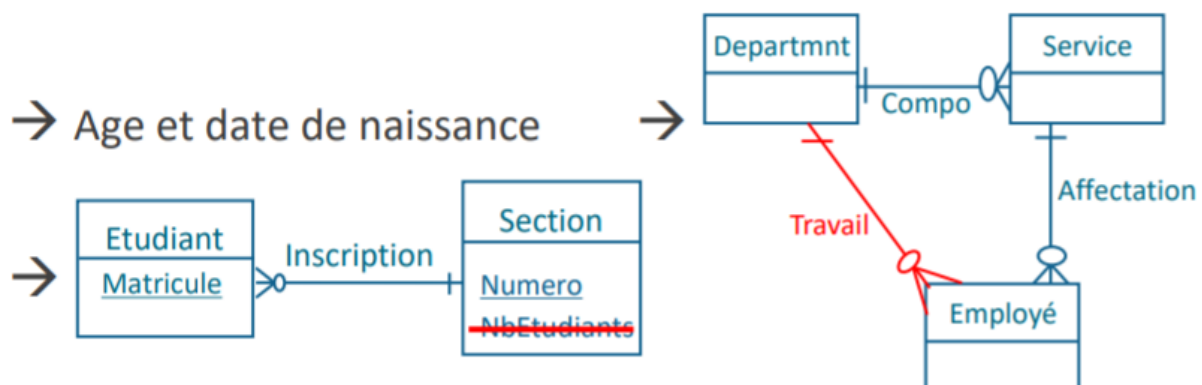
- Concernent les relations

## ➔ Trop coûteux en performances !

o Redondance :

De manière générale, il faut éviter la redondance.

Une information est redondante si elle peut être calculée ou dérivée à partir d'informations déjà stockées dans la BD.



### 3. Schéma relationnel

#### 3.1. BD relationnelles

##### o Table

- Correspond à une entité du diagramme ERD
- Définie par un nom et des attributs.

##### o Ligne

Nom

Attributs

Occurence

AUTEUR		
<u>Nom</u>	<u>Prénom</u>	Ddn
Smal	Anne	21/04/1974
Peten	Jean-Pol	29/02/1994
Bouraada	Mohamed	11/10/1955

Une ligne = une occurrence du type d'entité.

##### o Colonne

Un colonne = un attribut du type d'entité

= rôle joué par chacune des valeurs présentes dans la colonne

AUTEUR		
<u>Nom</u>	<u>Prénom</u>	Ddn
Smal	Anne	21/04/1974
Peten	Jean-Pol	29/02/1994
Bouraada	Mohamed	11/10/1955

o Valeur

La valeur = l'intersection d'une ligne et d'une colonne

= la valeur d'un attribut pour un occurrence particulière de la table.

AUTEUR		
Nom	Prénom	Ddn
Smal	Anne	21/04/1974
Peten	Jean-Pol	29/02/1994
Bouraada	Mohamed	11/10/1955

▪ Valeur particulière : valeur null

- La valeur de l'attribut est inconnue pour certaines occurrences.
- L'attribut ne s'applique pas à certaines occurrences
- Certaines occurrences ne possèdent pas de valeur pour l'attribut.

o Notion d'ordre

L'ordre des lignes et des colonnes n'a pas d'importance.

o Clé primaire

Clé primaire = identifiant

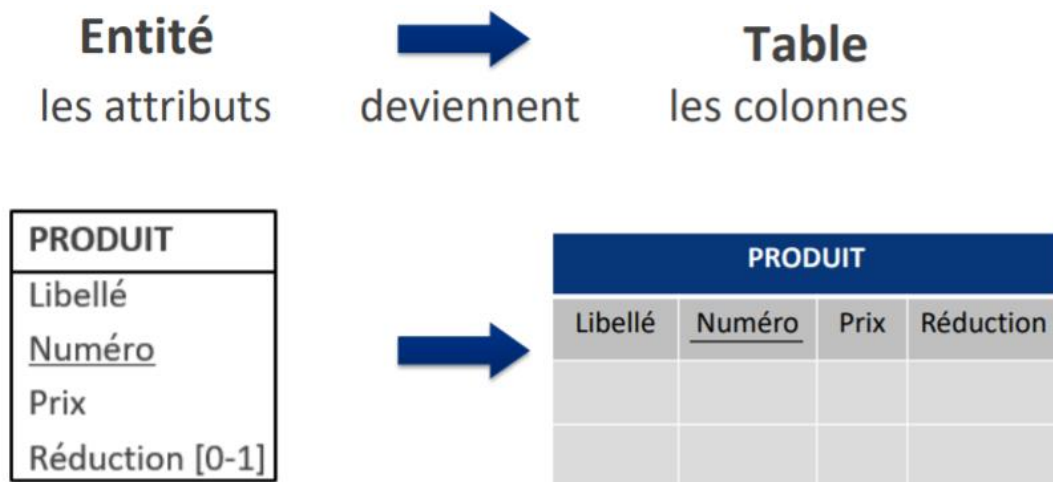
- A chaque valeur de la clé primaire correspond une et une seule ligne
- A une ligne de la table correspond une et une seule valeur de clé primaire

**Rappel** : un attribut est un identifiant pour une entité si sa valeur est distincte pour chaque occurrence d'entité

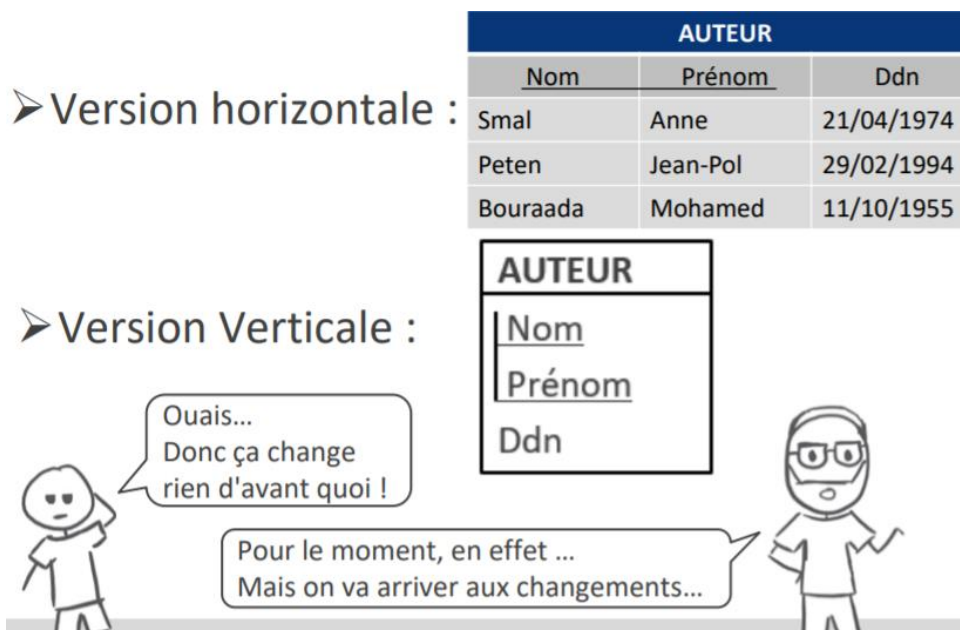
**Remarque** :

- Un bon identifiant est un identifiant invariant = valeur fixe.
- Aucune clé primaire ne peut être null
- En réalité, les identifiants composés sont rarement utilisés. On préférera un identifiant technique.

### 3.2. Transformation



### 3.3. Représentation schéma relationnel



### 3.4. Traduire les relations

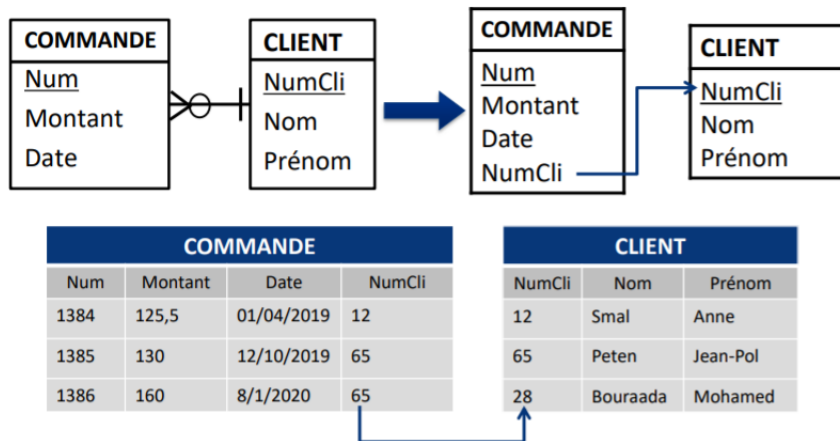
Une BD relationnelle ne contient que des tables.

Pour traduire du ERD en schéma relationnel :

- Les entités deviennent des tables
- Que deviennent les relations ? → Des clés étrangères

### o Clé étrangère

Une clé étrangère = colonne additionnelle dans une entité. Cette colonne va référencer l'identifiant (la clé primaire) de la table qu'on veut relier.



!/ le type de la clé étrangère doit être du même type que la colonne référencée



20

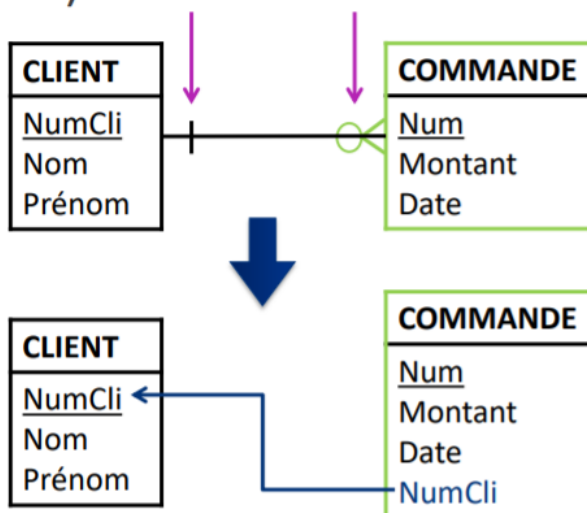
### o Règles

- 1) Relation 1-N
- 2) Relation 1-1
- 3) Relation N-N

#### ▪ 1-N

La clé étrangère va dans la table côté N. Elle va vers le 1.

#### 1) Max 1 – Max N



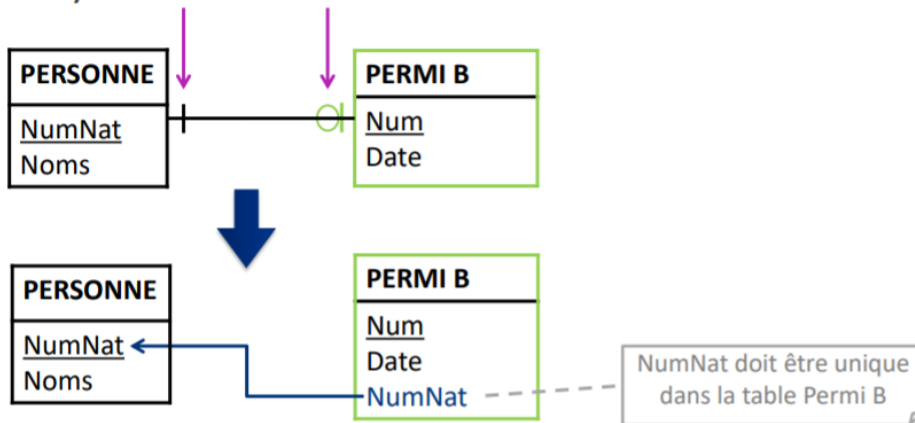


- 1-1

Peu importe le sens mais **!!** contraintes à ajouter.

Si un des deux côtés est facultatif, on préférera mettre la clé étrangère de ce côté là.

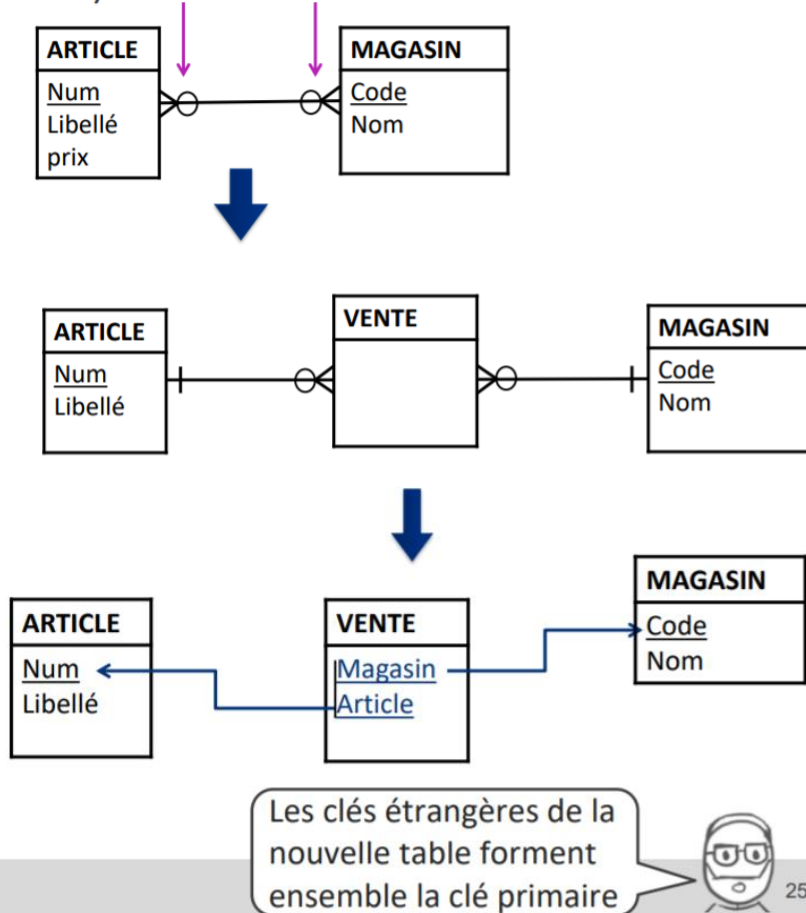
### 1) Max 1 – Max 1



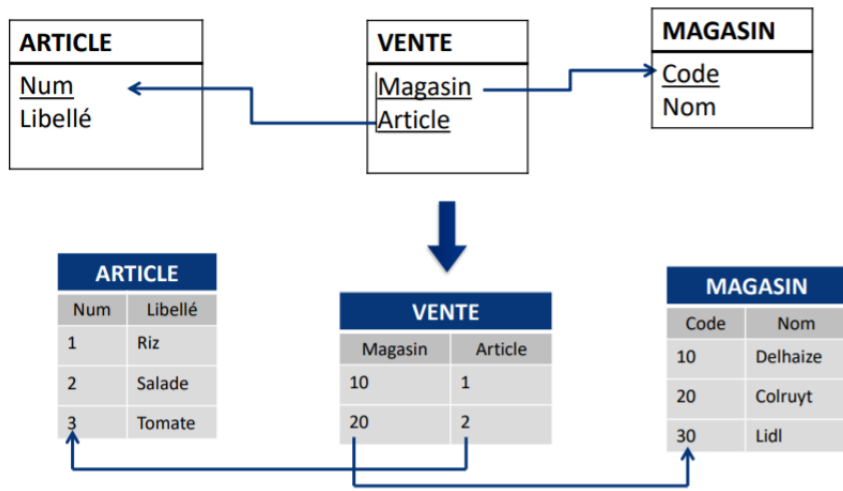
- N-N

La relation devient une table.

### 1) Max N – Max N



25



#### 4. SQL

- Insensible à la case.
- Les requêtes finissent par « ; » et peuvent s'écrire en plusieurs ligne.

##### 4.1. Convention de nommage

Les noms de colonne et tables doivent :

- Commencer avec une lettre.
- Faire entre 1 et 30 caractères.
- Contenir seulement A-Z, a-z, 0-9, \_, \$ et #.
- Être différents (pour un même utilisateur).
- Être différents des mots-réservés.

##### 4.2. Data definition language

DDL : définit les structures de base de données.

###### o Création

**Base de données :**

```
CREATE DATABASE <base de données> ;
```

**Table :**

```
CREATE TABLE [<schema>.<table> (
  <colonne> <type> [ DEFAULT <expr> ] [<contrainte>],
  ... );
```

- Exemple :

PERSONNE
<u>NumNat</u>
Noms
Ddn

```
CREATE TABLE PERSONNE (
  NumNat varchar(13) PRIMARY KEY,
  Noms varchar(100),
  Ddn Date ) ;
```

### o Contraintes

Contraintes d'intégrités à ajouter à la création :

- **PRIMARY KEY** : clef primaire.
- **UNIQUE** : valeur unique (appelé aussi clef secondaire, ou clef candidate).
- **NOT NULL** : obligatoire.
- **FOREIGN KEY** : clé étrangère.
- **CHECK** : contraintes additionnelles.

```
CREATE TABLE [<schema>.<table> (
  <colonne> <type> [ DEFAULT <expr> ] [<contrainte de colonne>],
  ...,
  [<contrainte de table>,
  ...]);
```

#### ▪ Contraintes sur une colonne

Se note juste après la définition de la colonne :

- **PRIMARY KEY** : définit la colonne comme clef primaire.
- **UNIQUE** : interdit à 2 valeurs de la colonne d'être les mêmes.
- **NOT NULL** : rend la colonne obligatoire.
- **FOREIGN KEY** : définit la colonne comme clef étrangère.  
→ [FOREIGN KEY] REFERENCES <table>(<colonne ref>)
- **CHECK** : pour définir des contraintes additionnelles  
→ CHECK <condition>

```
[ CONSTRAINT <nom> ] <type>
```

```
create table student
(studentid decimal(5) primary key,
lastname varchar2(50) );
```

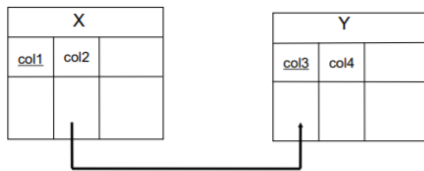
STUDENT
studentid
lastname

#### ▪ Contraintes sur une table

Se note à la fin de la création de la table :

- **PRIMARY KEY** : définit un ensemble de colonnes comme identifiant.  
→ PRIMARY KEY (<colonnes>)
- **UNIQUE** : interdit que 2 lignes aient les mêmes valeurs pour un ensemble de colonnes. → UNIQUE (<colonnes>)
- **FOREIGN KEY** : définit un ensemble de colonnes comme clef étrangère référençant des colonnes d'une autre table.  
→ FOREIGN KEY (<colonnes>) REFERENCES <table>(<colonne >)
- **CHECK** : pour définir des contraintes additionnelles  
→ CHECK <condition>

### ▪ FK simple

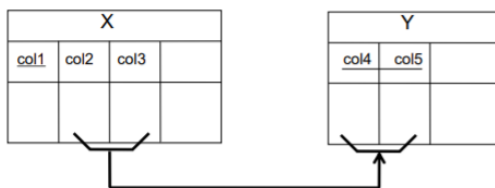


```
CREATE TABLE X
( col1 type1 PRIMARY KEY,
  col2 type2, ... ,
  FOREIGN KEY (col2) REFERENCES Y (col3), ... );
```

ou

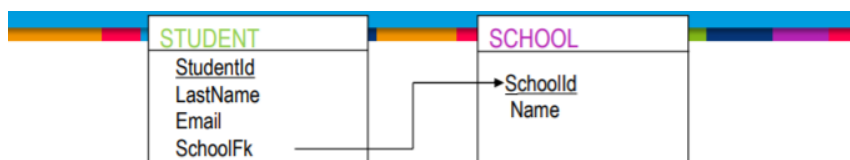
```
CREATE TABLE X
( col1 type1 (long1) PRIMARY KEY,
  col2 type2 (long2) REFERENCES Y (col3),
  ... );
```

### ▪ FK multiple



```
CREATE TABLE X
( col1 type1(long1),
  col2 type2(long2),
  col3 type3(long3),
  ... ,
  PRIMARY KEY (col1),
  FOREIGN KEY (col2,col3) REFERENCES Y (col4,col5) , ... );
```

### ○ Exemple



```
CREATE TABLE school
( schoolid decimal(5) PRIMARY KEY,
  name varchar(50) NOT NULL,
  );
```

```
CREATE TABLE student
( studentid decimal(5),
  lastname varchar(50) not null,
  email varchar(100),
  schoolFk decimal(5),

  CONSTRAINT student_id_pk PRIMARY KEY ( studentid ),
  CONSTRAINT email_uk UNIQUE( email ),
  CONSTRAINT school_fk FOREIGN KEY ( schoolfk )
    REFERENCES school ( schoolid ) );
```

o Modification

**Modifier la structure d'une table :**

ALTER TABLE <table>

- ADD (<colonnes>)
- ALTER COLUMN (<colonnes>)
- DROP (<colonnes>) **OU** DROP COLUMN <colonne>

**Renommer une table :**

EXEC sp\_rename '<ancien nom>', '<nouveau nom>' ;

**Renommer une colonne :**

EXEC sp\_rename '<table>.<ancien nom>', '<nouveau nom>', 'COLUMN' ;

```
ALTER TABLE student
  ADD      (streetname  varchar2(100),
            streetnumber decimal(4) );

ALTER TABLE student
  ALTER COLUMN  lastname      varchar2(100) not null;

ALTER TABLE student
  DROP COLUMN  email ;
```

o Suppression

**Supprimer toute une table :**

DROP TABLE <table> ;

**Vider la table :**

TRUNCATE TABLE <table> ;

### 4.3. Data manipulation language

DML : manipule les données (INSERT, UPDATE, DELETE).

- Insertion de données

```
INSERT
INTO      <table> [ ( <colonnes> ) ]
VALUES    ( <valeurs> );
```

STUDENT
studentid lastname

```
INSERT
INTO      Student (studentid, lastname)
VALUES    ( 18695, "Smith" );
```



```
INSERT
INTO      Student
VALUES    ( 18695, "Smith" );
```

- Modification de données

```
UPDATE
SET        <table>
           <colonne> = <valeur>, ...
[ WHERE    <condition> ] ;
```

```
UPDATE      Student
SET          lastname = "Smithy"
WHERE        studentid = 18695;
```

- Suppression de données

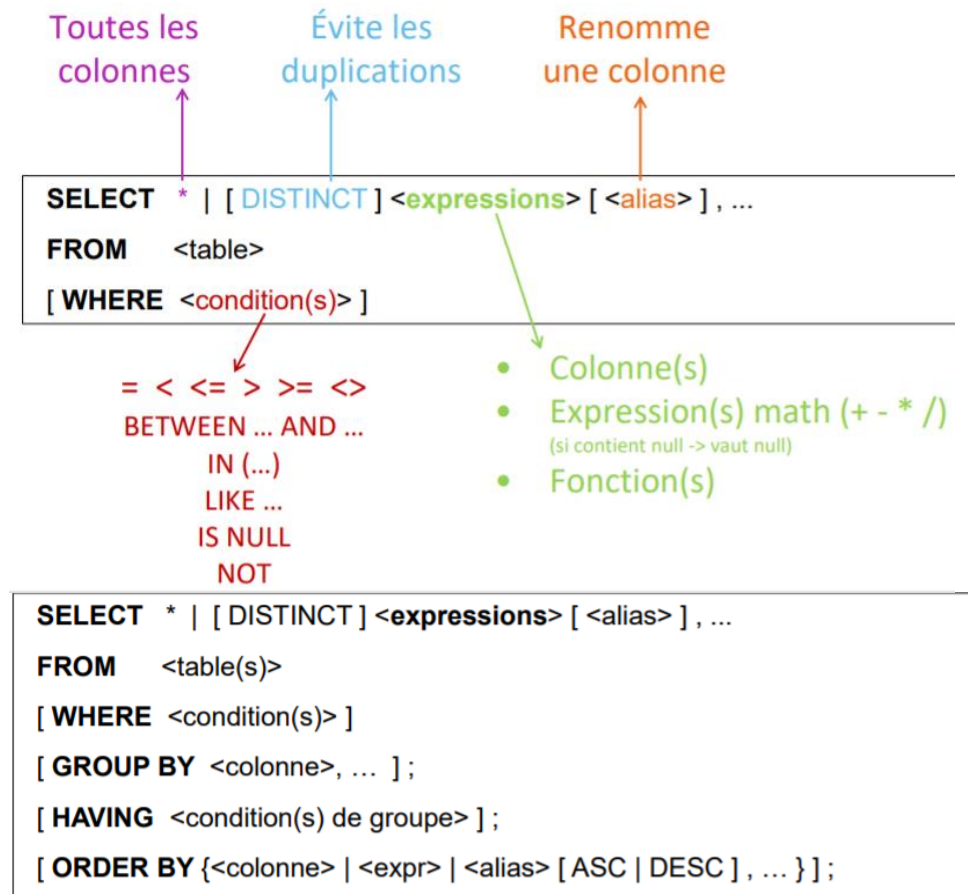
```
DELETE
[ FROM ]    <table>
[ WHERE     <condition> ] ;
```

```
DELETE
FROM        Student
WHERE        studentid = 18695;
```

#### 4.4. Data query language

DQL : permet de sélectionner (SELECT) les données, souvent incluse dans le DML.

##### o Sélections



##### ▪ Opérateurs

- Ordre des opérations :

1	~ (NOT au niveau du bit)
2	* (Multiplication), / (Division), % (Modulo)
3	+ (Positif), - (Négatif), + (Addition), + (Concaténation), - (Soustraction), & (AND au niveau du bit), ^ (OR exclusif au niveau du bit),   (OR au niveau du bit)
4	=, >, <, >=, <=, <>, !=, !>, !< (opérateurs de comparaison)
5	NOT
6	AND
7	ALL, ANY, BETWEEN, IN, LIKE, OR, SOME
8	= (Affectation)

### ▪ Fonctions

**function\_name [ ( <arguments> ) ]**

Peut être utilisé dans le SELECT ou le WHERE

- **LOWER :**
  - LOWER ('HENALLUX Partner') → henallux partner
- **UPPER :**
  - UPPER ('HENALLUX Partner') → HENALLUX PARTNER

```
select *
from employees
where LOWER( last_name ) = 'king';
```

```
select UPPER last_name
from employees ;
```

- **CONCAT :**
  - CONCAT ( 'Hello', 'World' ) → HelloWorld
- **LEN :**
  - LEN( 'Welcome' ) → 7
- **REPLACE :**
  - REPLACE ( 'Hellk wkrd', 'k', 'o') → Hello world
- **ROUND :**
  - ROUND (123.456, 2) → 123.46
- **LEFT :**
  - LEFT ('Welcome',3) → Wel
- **LTRIM :**
  - LTRIM ( ' Welcome',3) → Welcome

### ▪ Fonctions de groupe

Fonctions sur un groupe de lignes → donne un résultat à partir d'un groupe de ligne.



```
SELECT      group_function (column), ...
FROM        table
[ WHERE     condition(s) ];
```

**/!\** sans clause de groupement, toutes les lignes sont considérées comme un groupe !

- **AVG :** la moyenne.
- **COUNT :** le nombre.
- **MAX :** le maximum.
- **MIN :** le minimu.
- **SUM :** la somme.



```
select      avg (salary), sum(salary)
from        employees
```

```
select      max(salary), min(hire_date), max(last_name)
from        employees;
```

```
select      count(distinct department_id)
from        employees;
```

#### ▪ GROUP BY

Permet de diviser les lignes en plusieurs groupes → permet d'appliquer les fonctions de groupe sur chaque groupe.

```
SELECT * | [ DISTINCT ] <expressions> [ <alias> ], ...
FROM    <table(s)>
[ WHERE <condition(s)> ]
[ GROUP BY <colonne>, ... ];
[ HAVING <condition(s) de groupe> ];
[ ORDER BY {<colonne> | <expr> | <alias> [ ASC | DESC ], ... } ];
```

```
select      department_id, sum(salary)
from        employees
group by    department_id ;
```

#### ▪ HAVING

Restreint les groupes → ceux qui ne satisfont pas les conditions sont retirés

```
SELECT * | [ DISTINCT ] <expressions> [ <alias> ], ...
FROM    <table(s)>
[ WHERE <condition(s)> ]
[ GROUP BY <colonne>, ... ];
[ HAVING <condition(s) de groupe> ];
[ ORDER BY {<colonne> | <expr> | <alias> [ ASC | DESC ], ... } ];
```

La condition de groupe s'exprime avec des fonctions de groupe :

```
select      department_id, max(salary)
from        employees
where       commission_pct is not null
group by    department_id
having      max(salary) > 10000 ;
```

o Jointures

CLIENT		
Id	Nom	Parrain
1	Alice	
2	Bob	1
3	Chris	
4	David	3

FACTURE		
Id	Total	Client
1	259	1
2	54	3
3	158	1
4	25	6



- Je voudrais le nom du client et le total de chaque facture.
- Je voudrais savoir quel client n'a pas de facture.  
⇒ Besoin de « lier » les tables dans le SELECT

Il existe plusieurs sortes de jointures :

- Inner
- Full outer
- Left
- Right

```
SELECT <colonnes>
FROM <tableA A>
JOIN <tableB B>
ON <A.key> = <B.key>
...
```

condition de jointure

### Exemple

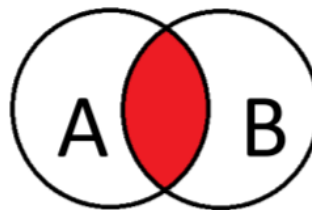
```
SELECT *
FROM CLIENT C
JOIN FACTURE F
ON C.Id = F.Client
```

condition de jointure

CLIENT			FACTURE		
Id	Nom	Parrain	Id	Total	Client
1	Alice		1	259	1
1	Alice		3	158	1
2	Bob	1			
3	Chris		2	54	3
4	David	3			
			4	25	6

CLIENT			FACTURE		
Id	Nom	Parrain	Id	Total	Client
1	Alice		1	259	1
2	Bob	1	2	54	3
3	Chris		3	158	1
4	David	3	4	25	6

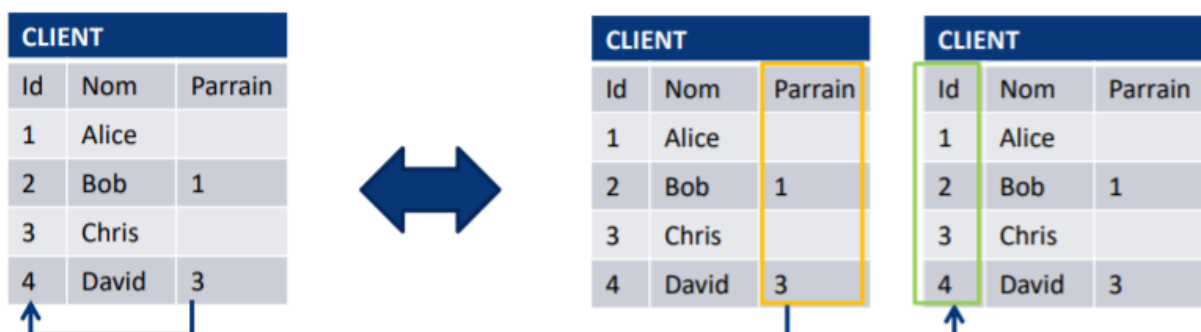
CLIENT			FACTURE		
Id	Nom	Parrain	Id	Total	Client
1	Alice		1	259	1
1	Alice		3	158	1
2	Bob	1			
3	Chris		2	54	3
4	David	3			
			4	25	6



CLIENT			FACTURE		
Id	Nom	Parrain	Id	Total	Client
1	Alice		1	259	1
1	Alice		3	158	1
2	Bob	1			
3	Chris		2	54	3
4	David	3			
			4	25	6

### Jointures réflexives

Lorsqu'une table a une clef étrangère vers elle-même, il faut voir la table comme 2 différentes :



N'importe quel type de jointure peut être fait :

```
SELECT cli.Id, cli.Nom, par.Nom as Parrain
FROM CLIENT cli
INNER JOIN CLIENT par
ON cli.Parrain = par.Id
```

CLIENT		
Id	Nom	Parrain
1	Alice	
2	Bob	1
3	Chris	
4	David	3

Pour éviter la confusion, renommez les table avec des alias différents

```
SELECT cli.Id, cli.Nom, par.Nom as Parrain
FROM CLIENT cli
INNER JOIN CLIENT par
ON cli.Parrain = par.Id
```

CLIENT alias « cli »			CLIENT alias « par »		
Id	Nom	Parrain	Id	Nom	Parrain
1	Alice		1	Alice	
2	Bob	1	2	Bob	1
3	Chris		3	Chris	
4	David	3	4	David	3

Id	Nom	Parrain
2	Bob	Alice
4	David	Chris

#### o Requêtes imbriquées

Il est possible d'imbriquer des requêtes

```
SELECT ...
FROM ...
WHERE <expr> <opérateur>
      ( SELECT ...
        FROM ...
        [ WHERE conditions ] );
```

Renvoie, en fait,  
des valeurs

Quelques opérateurs supplémentaires :

- X IN (<select>)  
→ X se trouve parmi la liste des valeurs renvoyées par le select.
- X <opé> ANY (<select>)  
→ X doit évaluer l'opérateur par rapport à au moins une des valeurs renvoyées par le select.
- X <opé> ALL (<select>)  
→ X doit évaluer l'opérateur par rapport à toutes les valeurs renvoyées par le select

- EXISTS

Teste l'existence de colonnes en résultat d'une sous requête.

- True si au moins une ligne est renvoyée.
- False si aucun ligne n'est renvoyée.

```
SELECT ...  
FROM ...  
WHERE EXISTS  
      ( SELECT ...  
        FROM ...  
        WHERE ...);
```