Introduction aux Bases de données

Modèle relationnel et SQL

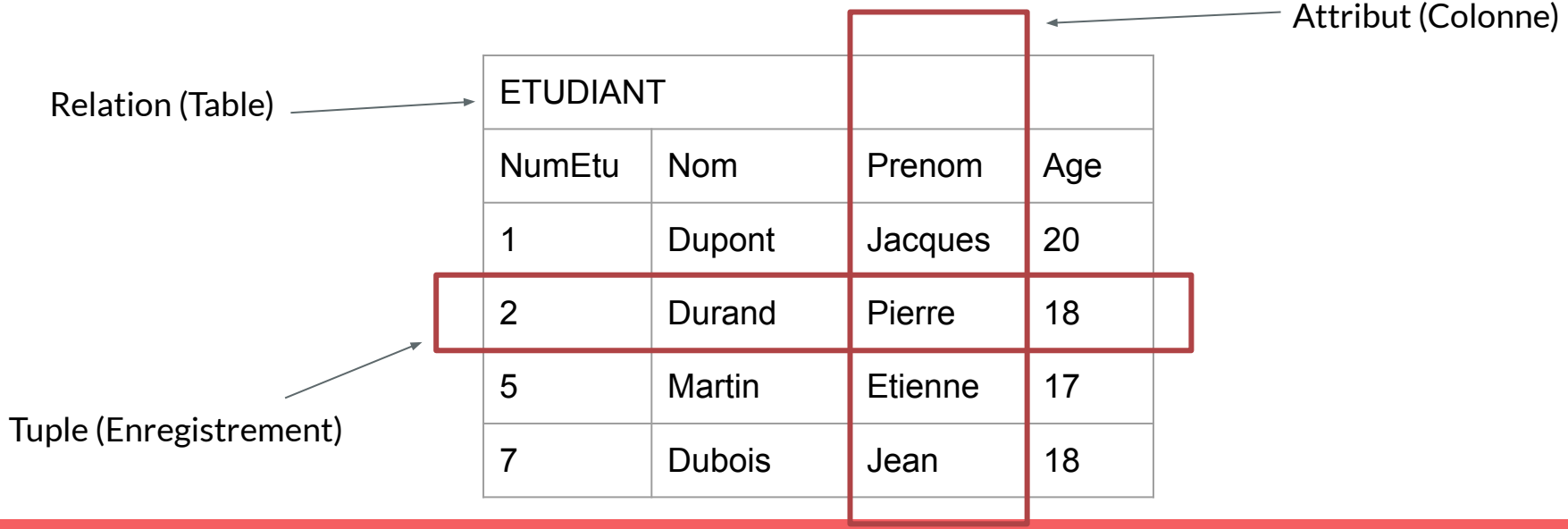
Modèle relationnel

Modèle relationnel

- Formalisé par Edgar Frank Codd en 1969, il est aussi noté aussi MRD
- Basé sur la théorie des relations et l'algèbre relationnelle
- Principe
  - Tables contenant les données sans connaissance de la représentation physique dans la machine
  - Chaque table représentant un ensemble ou relation
- Succès grâce à la simplicité des concepts
- Opérations se résumant à transformer une ou deux tables en une nouvelle table

Eléments du modèle relationnel

- Une **Relation** est une table contenant les données
- Un **Tuple** est une ligne d'une table (ou enregistrement)
- Un **Attribut** est une colonne d'une table



ETUDIANT			
NumEtu	Nom	Prenom	Age
1	Dupont	Jacques	20
2	Durand	Pierre	18
5	Martin	Etienne	17
7	Dubois	Jean	18

Règles d'intégrité structurelle

Unicité de la clé : attribut(s) permettant d'identifier chaque tuple de la relation de manière unique

Contraintes de références : attribut(s) d'une relation devant apparaître comme clé dans une autre relation

Contrainte d'entité : une clé ne peut pas avoir de valeur nulle (non-présence de l'information)

Contrainte de domaine : permet de définir l'ensemble de valeurs possibles pour chaque attribut

Notation

Relation : nom de la relation suivi des attributs et de leur domaine entre parenthèses

Clé primaire : nom de l'attribut (ou des attributs) souligné

Clé externe : ajout du caractère # devant le nom de l'attribut concerné

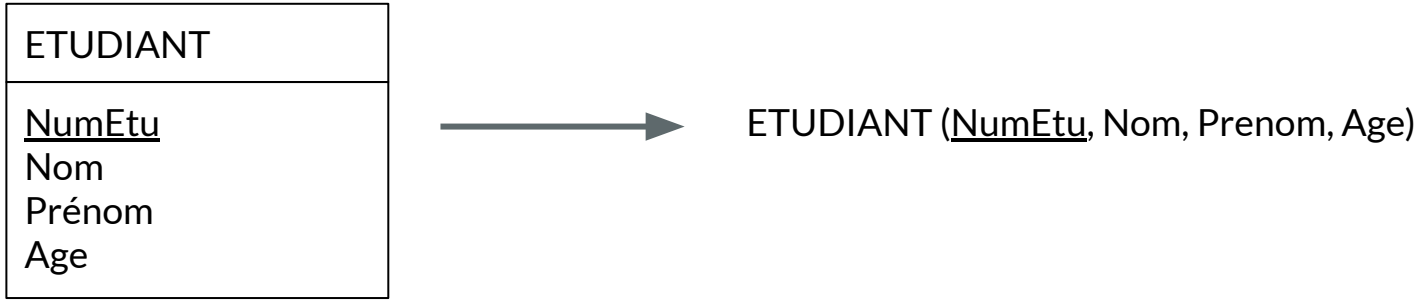
Exemple

- TVA (Code : entier, Valeur : réel)
- PRODUIT(NumPrd : entier, Libellé : chaîne, PrixHT : réel, #Code : entier)

Comment passer d’un MCD à un MRD ?

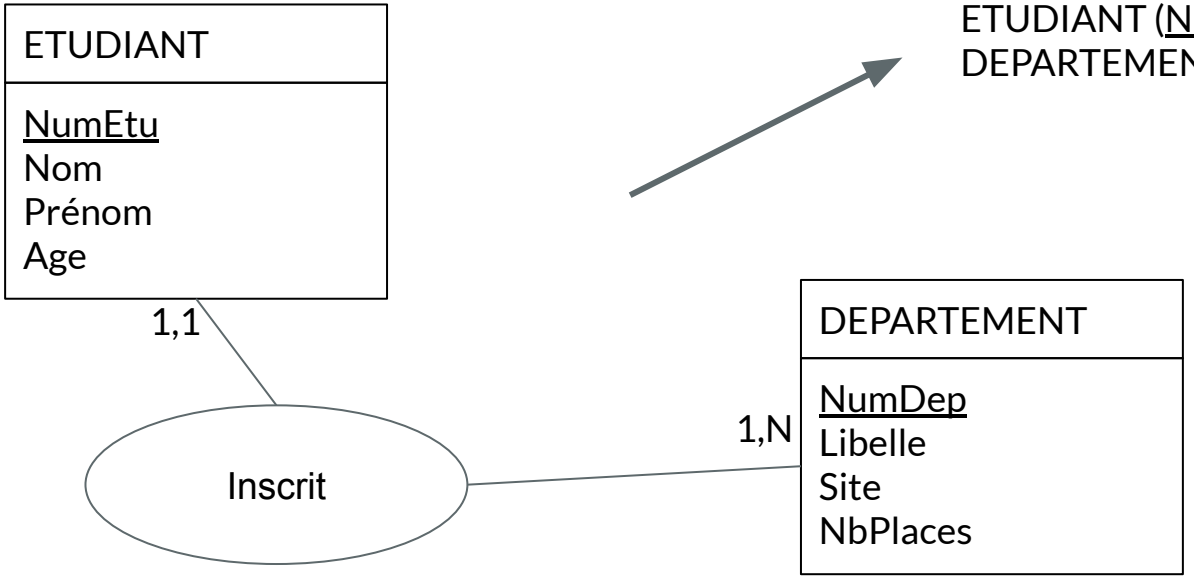
Entité = Relation

Chaque entité devient une relation



Association binaire 1,1 - 0/1,1/N

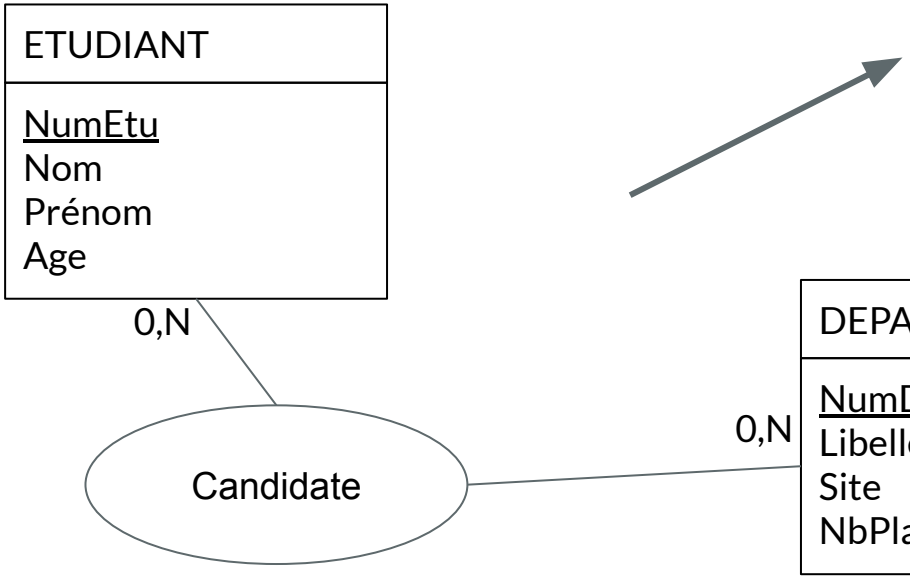
Dans la table du côté 1,1, on intègre en tant que clé externe la clé primaire de l'autre table



ETUDIANT (NumEtu, Nom, Prenom, Age, #NumDep)
DEPARTEMENT (NumDep, Libelle, Site, NbPlaces)

Association binaire 0/1,N - 0/1,N

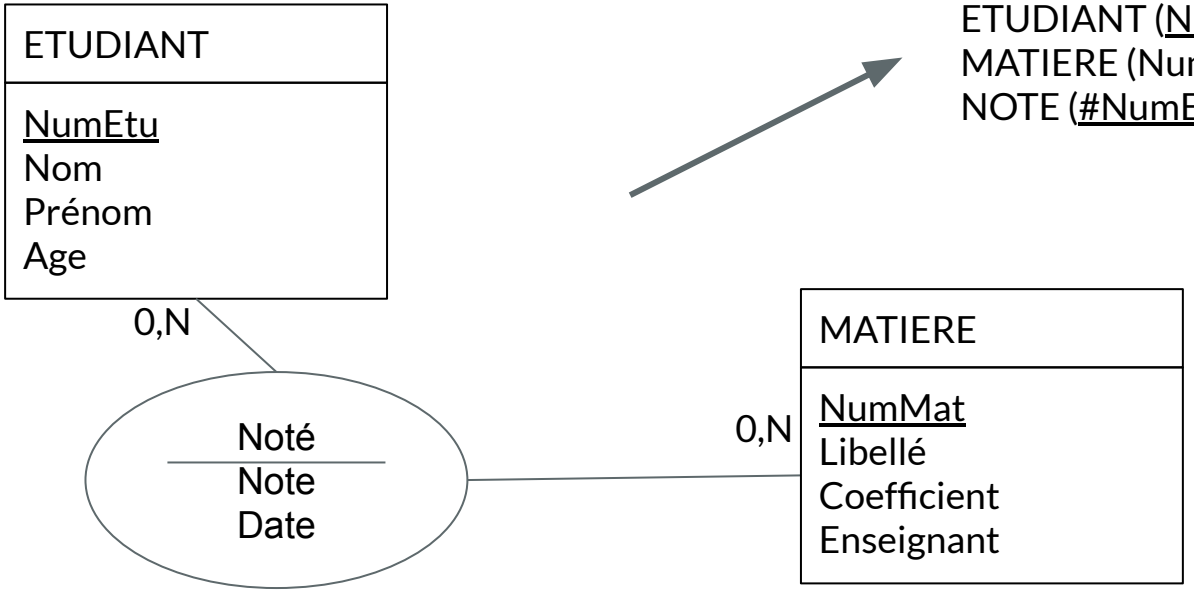
On crée ici une nouvelle relation contenant les clés primaires de chaque table en tant que clé externe. La clé primaire de la table sera l'ensemble de ces clés externes.



ETUDIANT (NumEtu, Nom, Prenom, Age)
DEPARTEMENT (NumDep, Libelle, Site, NbPlaces)
CANDIDATE(#NumEtu, #NumDep)

Association avec propriétés

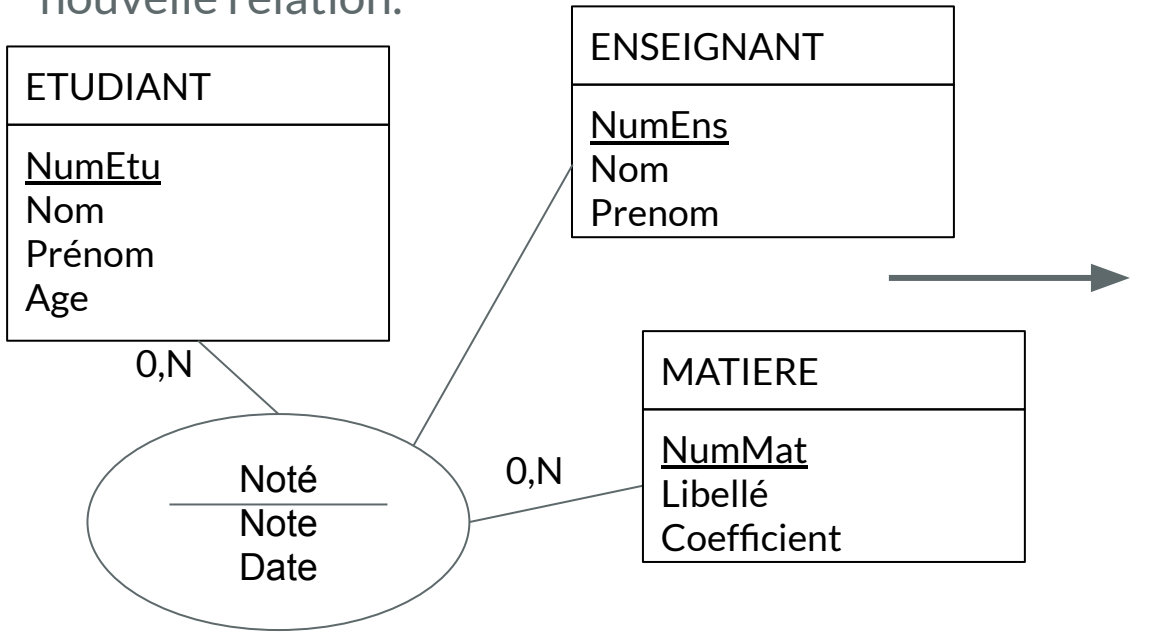
S'il y a des propriétés dans une association, on les intègre soit à la nouvelle relation (dans le cas 0/1,N-0/1,N), soit à la relation existante (dans le cas 1,1-0/1,1/N)



ETUDIANT (NumEtu, Nom, Prenom, Age)
MATIERE (NumMat, Libelle, Coefficient, Enseignant)
NOTE (#NumEtu, #NumMat, Note, Date)

Association ternaire

Dans un MCD, il est éventuellement possible d'avoir des associations à trois entités. Dans ce cas, on a souvent des cardinalités 0/1,N à chaque branche. On crée donc une nouvelle relation.



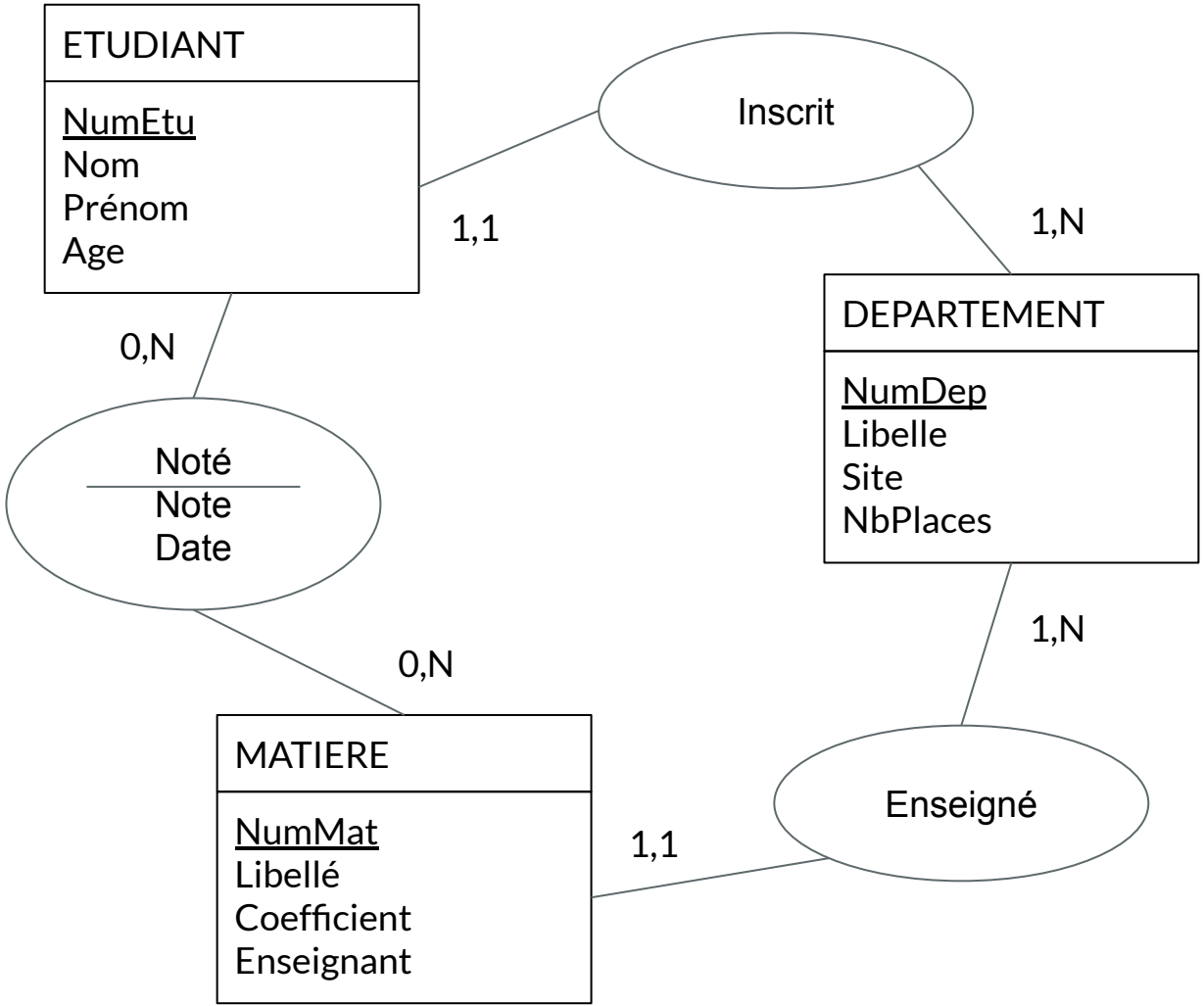
ETUDIANT (NumEtu, Nom, Prenom, Age)
ENSEIGNANT(NumEns, Nom, Prenom)
MATIERE (NumMat, Libelle, Coefficient)
NOTE (#NumEtu, #NumEns, #NumMat,
Note, Date)

Exemple

Transformation du MCD d’un département en MRD

Rappel du MCD

- 3 entités
- 3 associations
  - 2 en 1,1-qqch
  - 1 en 0,N-0,N



MRD obtenu

Les 3 entités deviennent donc chacune une relation

Les 2 associations 1,1-qqch sont intégrées à chaque table du côté du 1,1

La dernière association produit un nouvelle relation NOTE, avec des attributs

DEPARTEMENT(NumDep, Libellé, Site, NbPlaces)

ETUDIANT(NumEt, Nom, Prenom, Age, #NumDep)

MATIERE(NumMat, Libellé, Coefficient, Enseignant, #NumDep)

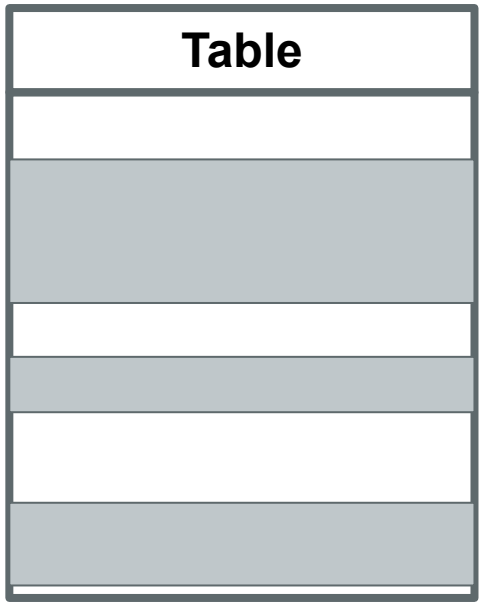
NOTE(#NumEt, #NumMat, Note, DateNote)

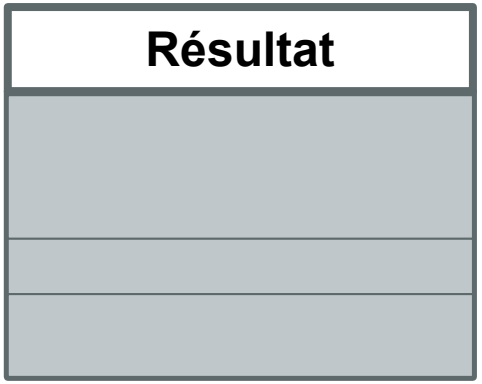
Algèbre relationnelle
Opérations classiques

Restriction

Sélection d'un nombre restreint de tuples (ou lignes) d'une table, selon un ou plusieurs critères

Ex : les étudiants de moins de 19 ans





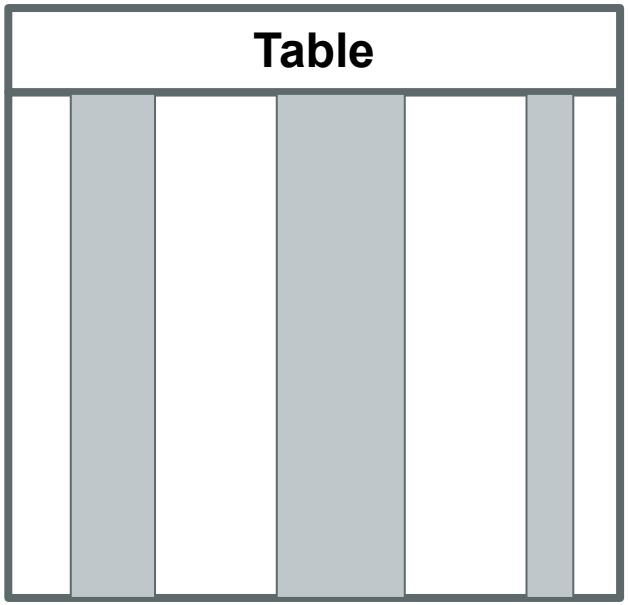
Projection

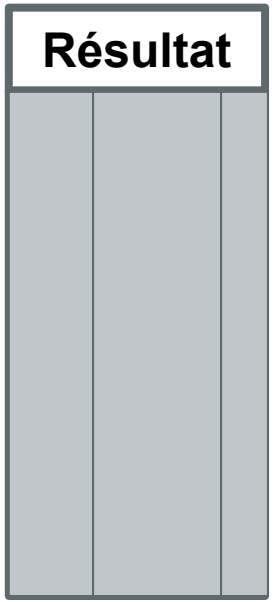
Sélection d'un nombre restreint de colonnes (ou attributs) d'une table

Ex : les noms et prénoms uniquement des étudiants

Attention : il y a risque d'avoir des doublons

Ex : le sexe des étudiants

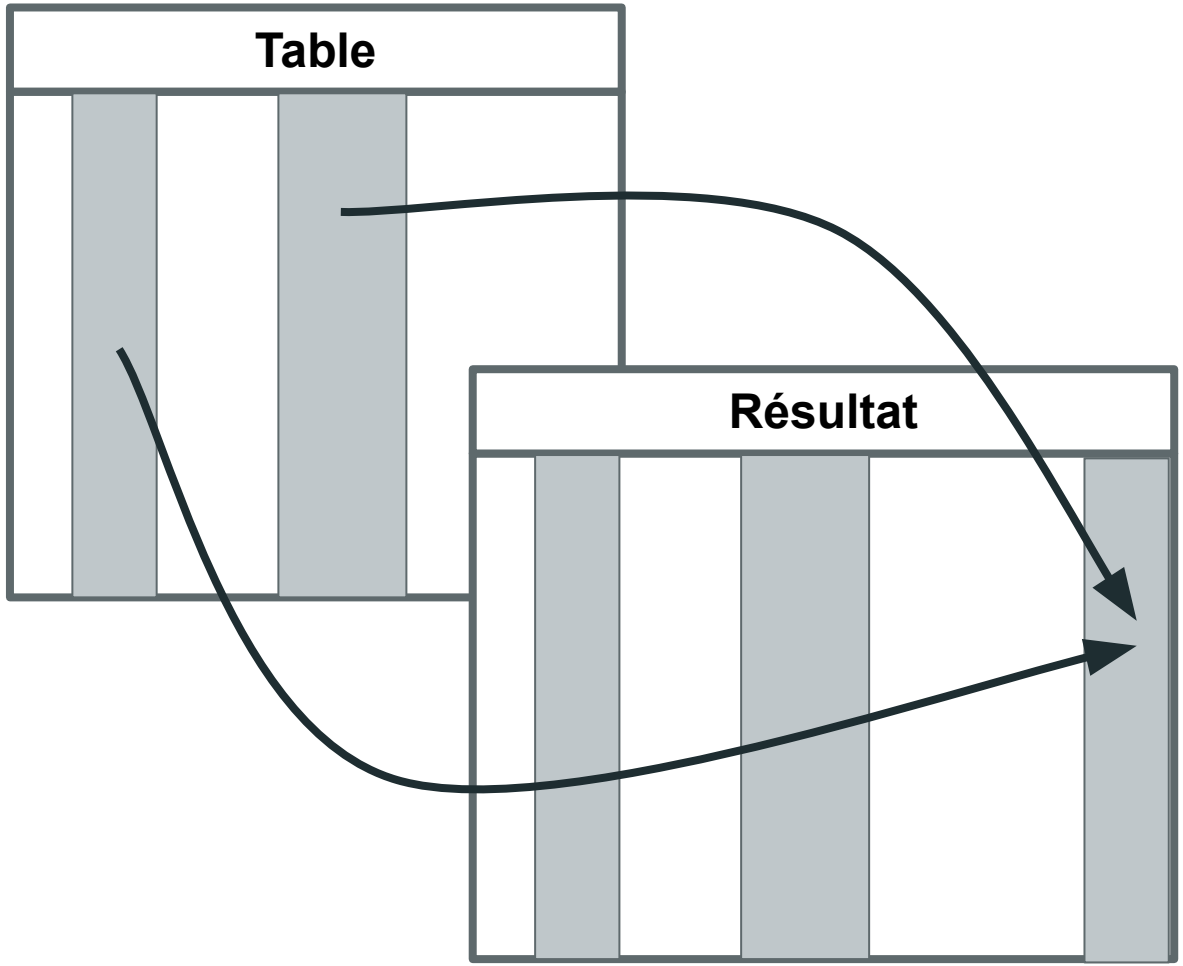




Calcul

Création d'une nouvelle colonne en fonction des autres attributs de la table

Ex : Les initiales de l'étudiant

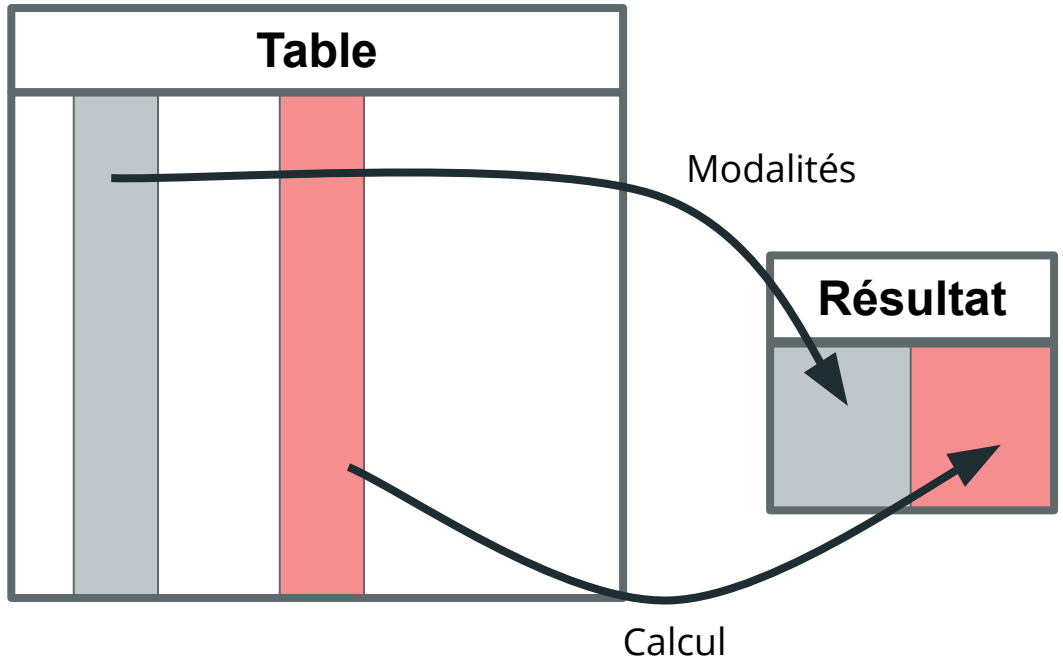


Agrégat

Calcul d'une statistique de base sur un attribut pour chaque modalité d'un autre attribut

Ex : Le nombre d'étudiants ou l'âge moyen de chaque sexe

Calculs les plus courants : dénombrement, somme, moyenne, minimum et maximum

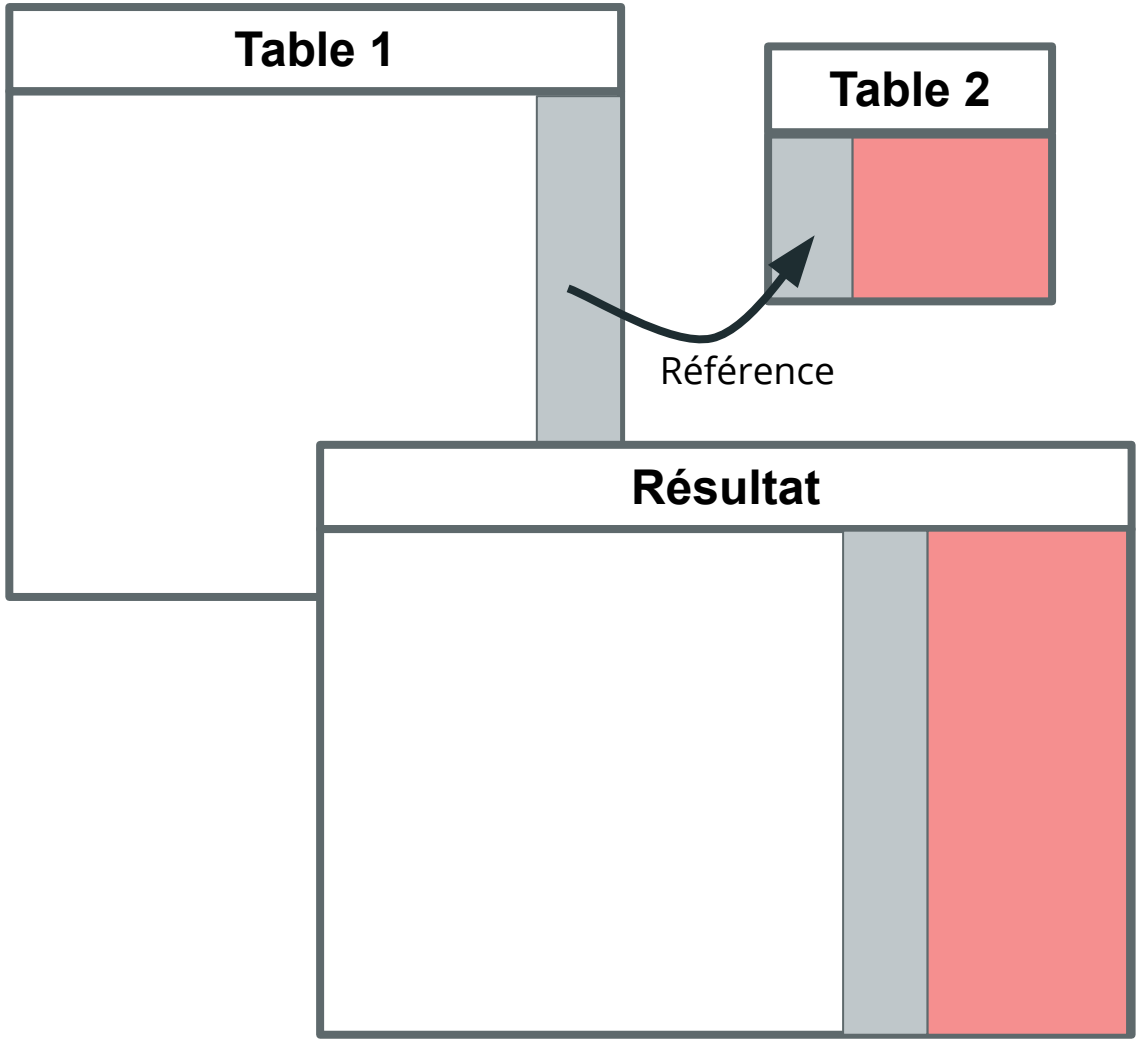


Jointure

Regroupement d'informations présentes dans deux tables, le rapprochement entre les tables se fait sur la base d'un ou plusieurs attributs (généralement les clés externes de l'une et primaires de l'autre)

Ex : Ajout du libellé de la matière pour chaque note

Attention : il existe des cas plus complexes où des lignes d'une table n'ont pas de correspondance dans l'autre

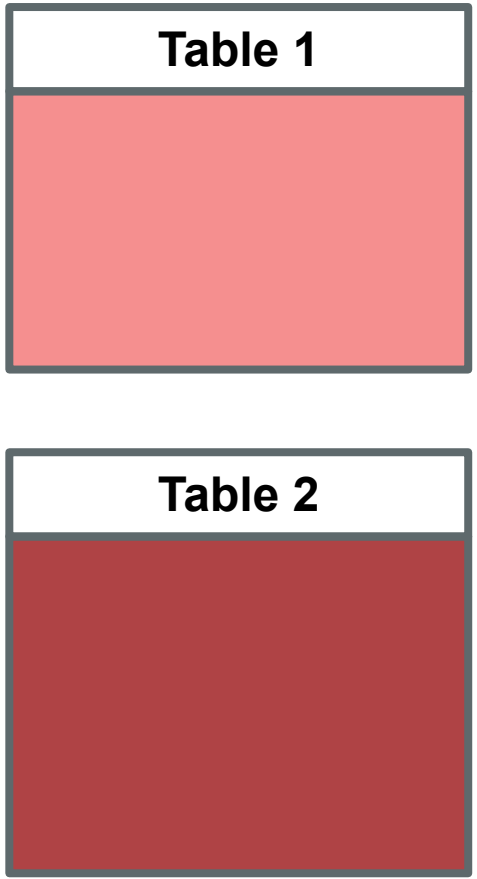


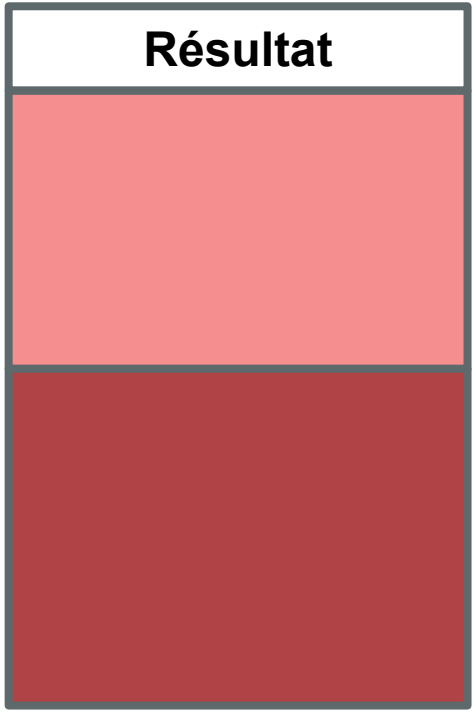
Opérations ensemblistes

Regroupement d'informations présentes dans deux tables ayant exactement les mêmes colonnes

Ex : union des étudiants STID et des étudiants Info

Opérations ensemblistes classiques possibles : union, intersection, différence





Langage SQL

Introduction au SQL

***SQL** : Structured Query Language*

- Besoin d'un langage standard pour la gestion et la manipulation des BD

Notation utilisée

- Mots clés SQL sont en majuscule et en gras
- [] indique un paramètre optionnel
- {} indique les choix possibles, séparés par des |
- … indique un paramètre répétitif possible

Articulation en plusieurs grandes parties

- DDL : Data Definition Language
  - Création, suppression et modification des tables
  - CREATE, ALTER, DROP
- DML : Data Manipulation Language
  - Insertion, suppression et mis à jour des données dans les tables
  - INSERT, UPDATE, DELETE
- DQL : Data Query Language
  - Requêtage (récupération) des données dans les tables
  - SELECT
- DCL/TCL : Data/Transaction Control Language
  - Contrôle des droits (DCL) et des sauvegardes (TCL)
  - GRANT, REVOKE
  - COMMIT, ROLLBACK, SAVEPOINT
  - Pas vu ici

DDL

Data Definition Language

- Définition des tables
- Expression des contraintes d'intégrité
- Définition de vues
- Suppression des tables
- Modification des tables

Création de tables

Les noms de table et d'attribut doivent être sans espace et idéalement sans accent

DEFAULT permet de définir une valeur par défaut

CONSTRAINT permet de donner un nom à la contrainte (d'attribut ou de relation)

```
CREATE TABLE nom_table (
    att type [DEFAULT exp] [[CONSTRAINT ctr] ctr_att …],
    …,
    [[CONSTRAINT ctr] ctr_rel] …
);
```

Création de tables

- Quelques types classiques
  - Caractères : `VARCHAR2(n)`, `CHAR(n)`
  - Numérique : `NUMBER`, `INTEGER` (ou `INT`)
  - Temporel : `DATE`, `TIMESTAMP`
- Contraintes d'attributs
  - `NOT NULL` : interdit l'absence de valeur pour cet attribut
  - `PRIMARY KEY` : définit un seul attribut comme clé primaire
  - `UNIQUE` : définit un seul attribut comme clé secondaire
  - `REFERENCES table [(attribut)]` : définit l'attribut comme clé externe, en indiquant quelle table (et quel attribut si le nom est différent) est pointée
  - `CHECK(condition)` : définit des contraintes de valeurs sur l'attribut

Création de tables

- Contraintes de relation
  - `PRIMARY KEY (att1, …)` : définit la clé primaire, qu'elle est simple ou multiple attributs
  - `UNIQUE(att1, …)` : idem mais pour clé secondaire
  - `FOREIGN KEY (att1, …) REFERENCES tab [(att, …)]` : indique une référence à une table, que celle-ci soit sur un seul ou plusieurs attributs (en indiquant ceux-ci si les noms différent)
  - `CHECK(condition)` : définit une condition sur un ou plusieurs attributs

Exemples

```
CREATE TABLE Etudiant (
    NumEt INT NOT NULL PRIMARY KEY,
    Nom VARCHAR2(50),
    Prenom VARCHAR2(50),
    Age INT CHECK(Age > 0),
    NumDep INT REFERENCES DEPARTEMENT
);
CREATE TABLE Departement (
    NumDep INT NOT NULL CONSTRAINT pk_Departement PRIMARY KEY,
    Libelle VARCHAR2(100),
    Site VARCHAR2(100),
    NbPlaces INT CHECK(NbPlaces > 0)
);
```

Création de vues

Une vue est une table virtuelle, créée à partir d'une requête et qui peut être utilisée comme une table classique

L'intérêt d'une vue est que la requête est exécutée lors de l'appel de celle-ci. Les données dedans sont donc à jour.

```
CREATE VUE nom_vue AS
    requête;
```

```
CREATE VUE vue_etudiant AS
    SELECT * FROM ETUDIANT;
```

Suppression de tables

Il faut faire attention lors de la suppression d'une table, s'il y a des références sur celle-ci. Cela produira par défaut une erreur.

Il existe des mécanismes pour contourner cela, mais l'idéal est de supprimer d'abord les tables qui pointent sur celle-ci ou les références à celle-ci.

```
DROP TABLE nom_table;
```

```
DROP TABLE ETUDIANT;
```

Modification de tables

La première modification possible est d'ajouter ou de modifier un attribut à la table.

La syntaxe à utiliser est la même que pour la création de tables

```
ALTER TABLE nom_table [{ADD | MODIFY}] (
    att type [contraintes]
);
```

```
ALTER TABLE Etudiant ADD (
    Sexe CHAR(1) CHECK(Sexe IN c('H','F'))
);
```

Modification de tables

On peut aussi vouloir ajouter une contrainte de relation.

La syntaxe à utiliser est la même que pour la création de tables

```
ALTER TABLE nom_table ADD
    [CONSTRAINT nom] contrainte
);
```

```
ALTER TABLE Etudiant ADD
    CONSTRAINT ck_Etudiant_Age_inf100 CHECK(Age < 100);
```

Modification de tables

Enfin, on peut aussi supprimer des contraintes à la table, surtout si elles sont nommées

```
ALTER TABLE nom_table DROP
    { PRIMARY KEY |
      UNIQUE(att) |
      CONSTRAINT nom_contrainte };
```

```
ALTER TABLE Etudiant DROP
    CONSTRAINT ck_Etudiant_Age_inf100;
```

DML

Data Manipulation Language

Insertion

Modification

Suppression

Insertion de valeurs

On insère ici une nouvelle ligne à une table. L'insertion est effective si les contraintes de la table sont respectées. On peut ne renseigner que certains attributs.

```
INSERT INTO nom_table [(att, ...)]
    VALUES (exp, ...)
);
```

```
INSERT INTO Etudiant
    VALUES (7, “Bond”, “James”, 40, 1);
INSERT INTO Etudiant (NumEt, Nom)
    VALUES (1, “Aristote”);
```

Suppression de valeurs

La suppression est effective si la ligne n'est référencée nulle part dans une autre table. Les conditions s'écrivent de la même façon que dans le DQL (cf plus loin).

```
DELETE FROM nom_table
    [WHERE condition];
```

```
DELETE FROM Etudiant;
DELETE FROM Etudiant
    WHERE NumEt = 1;
```

Modification de valeurs

La mise à jour peut concerner toutes les lignes, où quelques unes seulement (voire une seule).

```
UPDATE nom_table
    SET attribut = expression
    [WHERE condition];
```

```
UPDATE Etudiant
    SET Age = Age + 1;
UPDATE Etudiant
    SET Age = 2404
    WHERE NumEt = 1;
```

DQL

Data Query Language

Interrogation du contenu

Sûrement la partie la plus importante dans le SQL pour un spécialiste de la data

Notations

- Nom d'attribut : `Table.Attribut`
  - Table pouvant être omis si aucune ambiguïté
- Opérateurs arithmétiques classiques : `+ - * / ( )`
- Concaténation de chaînes : `exp || exp`
- Opérateurs de comparaison
  - Classiques : `exp { = | <> | < | <= | > | >= } exp`
  - Combinaison : `exp { AND | OR } exp`
  - Négation : `NOT exp`
  - Par rapport à une liste : `IN (exp, ...)`
  - Par rapport à un intervalle : `BETWEEN exp AND exp`
  - Par rapport à un format : `LIKE "format"`
    - `_` : un et un seul caractère
    - `%` : une suite de caractères quelconques (éventuellement nulle)

Interrogation de contenu

La requête renvoie le résultat, sans le stocker dans la base

```
SELECT exp, ...
    [FROM table, ... ]
    [WHERE condition];
```

```
SELECT DATE(“now”);
SELECT *
    FROM Etudiant;
```

Ordre et limitation

- ORDER BY : tri du résultat sur un ou plusieurs attributs
  - ascendant par défaut
  - ajout de DESC après un attribut pour indiquer l'ordre décroissant pour celui-ci
- LIMIT : ne renvoie que les n premières lignes
  - Toujours en dernier
- La combinaison permet donc de faire des TOPn, très prisés en entreprise
  - à connaître donc

```
SELECT exp, ...
    [FROM table, ... ]
    [ORDER BY att [DESC], ...]
    [LIMIT n];
```

```
SELECT *
    FROM Etudiant
    ORDER BY Age DESC, Nom, Prenom
    LIMIT 100;
```

De l’algèbre relationnelle au langage SQL

Restriction

Sélection de certaines lignes de la table, selon une ou plusieurs conditions

```
SELECT *
    FROM table
    WHERE condition;
```

```
SELECT *
    FROM Etudiant
    WHERE Age > 20;
```

Projection

Sélection de certaines colonnes de la table.

Attention aux doublons ainsi créés : on peut les supprimer avec DISTINCT

```
SELECT [{DISTINCT | ALL}] attribut, …
    FROM table;
```

```
SELECT Nom
    FROM Etudiant;
SELECT DISTINCT Sexe
    FROM Etudiant;
```

Calcul

Ajout d'une nouvelle colonne, résultant d'un calcul (ou expression) sur un ou plusieurs autres attributs (fait sur chaque ligne)

Il est possible de renommer le résultat avec AS

```
SELECT ..., expression [AS alias]
    FROM table;
```

```
SELECT Nom, Prenom, Age, 2020 - Age AS AnneeNaissance
    FROM Etudiant;
```

Calcul d'agrégat simple

Calcul d'une statistique basique (principalement dénombrement, somme, moyenne, minimum, maximum) sur l'ensemble des lignes

Fonctions existantes : COUNT(), SUM(), AVG(), MIN(), MAX(), ...

```
SELECT ..., fct(att) [AS alias]
    FROM table;
```

```
SELECT COUNT(*)
    FROM Etudiant;
```

- COUNT(*)
  - Compte toutes les lignes
- COUNT(attribut)
  - Compte toutes les valeurs non nulles de cet attribut
- COUNT(DISTINCT attribut)
  - Compte toutes les valeurs distinctes de cet attribut

Calcul d'agrégat selon un critère

Calcul de la statistique pour chaque groupe créé par une modalité d'un attribut (ou d'un ensemble de modalités pour un ensemble d'attributs)

Les attributs listés dans le SELECT doivent être identiques à ceux listés dans le GROUP BY

```
SELECT liste, fct(att) [AS alias]
    FROM table
    GROUP BY liste ;
```

```
SELECT Sexe, COUNT(*)
    FROM Etudiant
    GROUP BY Sexe;
```

Restriction d'agrégat

On souhaite ne sélectionner que certaines modalités selon le résultat du calcul de la statistique

Il est impossible de la faire dans le WHERE dans ce cas

```
SELECT liste, fct(att) [AS alias]
    FROM table
    GROUP BY liste
    HAVING condition;
```

```
SELECT Age, COUNT(*)
    FROM Etudiant
    GROUP BY Age
    HAVING COUNT(*) > 10;
```

Jointure naturelle

L'idée d'une jointure est donc de récupérer les informations d'une table B pour les "coller" à une table A, en liant les 2 tables selon un attribut ayant les mêmes valeurs (n'ayant pas forcément pas le même nom - et éventuellement sur plusieurs attributs)

La jointure naturelle se fait sur des attributs ayant exactement les mêmes noms

```
SELECT ...
    FROM tableA NATURAL JOIN tableB;
```

```
SELECT *
    FROM Etudiant NATURAL JOIN Departement;
```

Jointure interne

La jointure interne se fait sur des attributs ayant exactement des noms différents

Il faut noter que pour cette jointure (ainsi que la jointure naturelle), le résultat contient les lignes présentes dans les 2 tables

```
SELECT ...
    FROM tableA INNER JOIN tableB
        ON condition;
```

```
SELECT *
    FROM Etudiant INNER JOIN Departement
        ON Etudiant.NumDep = Departement.NumDep;
```

Jointure externe

La jointure externe permet de garder les lignes de la table A (pour jointure gauche), ou de la table B (pour jointure droite) ou les deux (pour jointure complète), non présentes dans l'autre table

```
SELECT ...
    FROM tableA {LEFT | RIGHT | FULL } [OUTER] JOIN tableB
        ON condition;
```

```
SELECT *
    FROM Etudiant LEFT OUTER JOIN Departement
        ON Etudiant.NumDep = Departement.NumDep;
```

Jointure “à la main”

On peut aussi faire les jointures internes “à la main”, i.e. faire les restrictions dans le WHERE et lister toutes les tables dans le FROM

```
SELECT ...
    FROM tableA , tableB
    WHERE condition;
```

```
SELECT *
    FROM Etudiant, Departement
    WHERE Etudiant.NumDep = Departement.NumDep;
```

L’une ou l’autre des jointures donnent le même résultat, le choix dépendant des habitudes de l’entreprise ou du développeur souvent

Opération ensembliste

Ceci correspond à l'union, l'intersection et la différence

```
requêteA
UNION [ALL] | INTERSECT | EXCEPT
requêteB;
```

```
SELECT *
    FROM Etudiant
    WHERE Age > 18
INTERSECT
SELECT *
    FROM Etudiant
    WHERE Sexe = "H";
```

Les 2 requêtes doivent retourner strictement les mêmes colonnes

Division

On cherche ici dans une requête des lignes pour lesquelles ils existent (ou non) un résultat obtenue dans une sous-requête

```
SELECT ...
    FROM TableA
    WHERE [NOT] EXISTS (sous-requête incluant TableA);
```

```
SELECT *
    FROM Departement
    WHERE EXISTS (SELECT *
                  FROM Etudiant
                  WHERE NumDep = Departement.NumDep);
```