Modélisation Physique de données

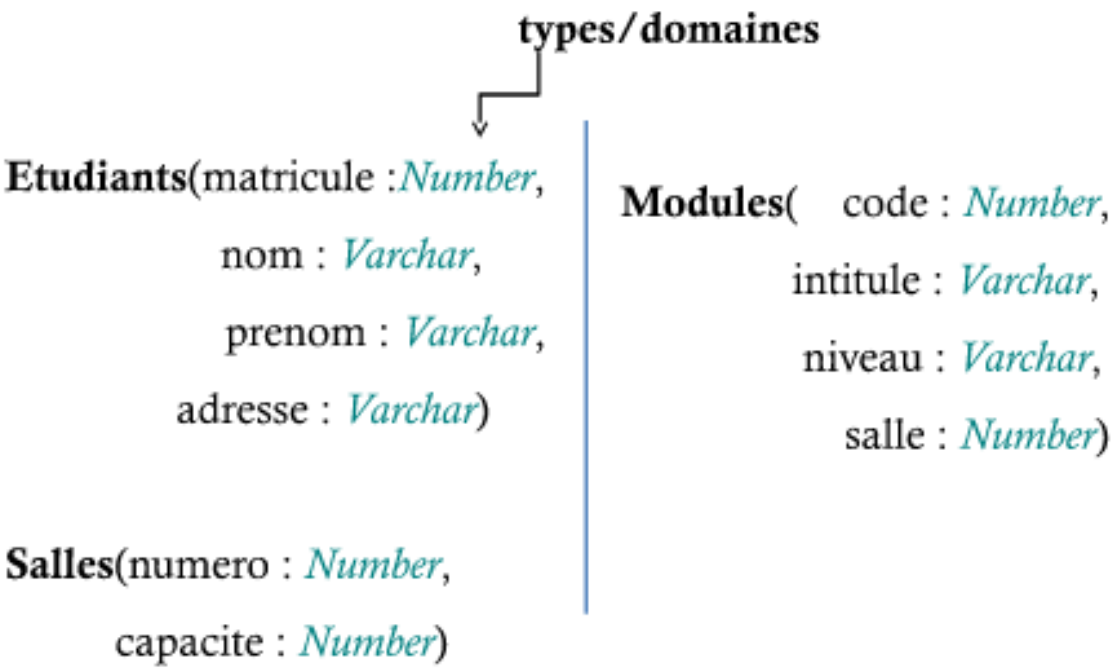
Pourquoi SQL?

1- Définir et modifier le schéma d'une BD

2- Manipuler les données (ajout, suppression, modification)

3- Interroger les données

Concepts clés : Types



Définition des types/Domaines

Types de Bases

Types alphanumériques

- *char(n)* ou character
- *varchar(n)* ou varying character

Types Numériques

- *int* ou *integer*, smallint
- numeric(t,d), *decimal*(t,d)
- *real*, double precision
- float(n)

Définition des types/Domaines

Types alphanumériques

char(n) vs varchar(n)

- *char(n)* réserve *n* cases dans tous les cas, complète par des blancs
  Ex. char(10) |C|O|U|R|S| | | | | |
- *varchar(n)* utilise **au maximum** *n* cases
  Ex. varchar(10) |C|O|U|R|S|
- Précaution lors de la comparaison de deux attributs déclarés comme char(n) et varchar(n) !

Définition des types/Domaines

Types Numériques

numeric(t,d)

Un nombre avec virgule fixe :

- t = nombre total maximum de chiffres (sans signe)
- d = nombre de chiffres après la virgule (partie décimale)
- t-d = nombre de chiffres avant la virgule (partie entière)

Ex. numeric(5,2) peut contenir 123.45, -123.45, 0.56 ou 22, mais pas 1200 ni 12.345

real, **double précision**

Nombre à virgule flottante

float(n)

Nombre à virgule flottante sur *n* bits

Définition des types/Domaines

Types temporels : date, time, timestamp

date

- Date calendaire (année, mois et jour du mois)

time

- Temps d'un jour (heure, minutes, secondes)
- Fuseau horaire

timestamp

- Combinaison des types date et time

Exemples

date

'01-01-2015'

time

'00:30:00'

timestamp

'01-01-2015 0:30:12.50'

Manipulation des types temporels

1 - Conversion d'une chaîne de caractères vers un type temporel

cast (*chaine* as *type*)

type : date, time, timestamp

2 - Extraction d'un champ d'un type temporel

extract (*champ* **from** *type*)

champ : year, month, day, hour(s), minute(s), second(s)

3- Extraction de la date et l'heure système

current_time ex. '10:10:30.123456'

current_date ex. '01-01-2015'

current_timestamp, **localtimestamp**
ex. '01-01-2015 10:10:30.123456'

4- Arithmétique sur les types date

date1-date2 =*nombre de jours*

date1(+/-) entier= date (après/avant) entier jours

Définition des types/Domaines

Définition des domaines

Définition d'un nouveau type à partir d'un type de base

Syntaxe

create domain <*nom*> as <*dom*>[<contrainte>];

Exemples

create domain fin_semaine **as** text

constraint fin_sem **check**(**value** in ('sam','dim'));

create domain sal_min **as numeric(7,2)**

constraint sal_val **check**(**value** >=10 000);

Compte(numero, titulaire, lieu, ouverture, agence)

create domain codeP **as** numeric(5);
create domain codeAg **as** numeric(5);
create table Compte(
numero **numeric(10) not null**,
titulaire **varchar2(10) not null**,
lieu codeP **default** 75001,
ouverture **date**,
agence codeAg);

Langage de définition de données : Création des Tables

create table <*nom*>

(<*Attr1*> <Dom*1*> [**not null**] [**default** <val*1*>],
<*Attr2*> <Dom*2*> [**not null**] [**default** <val*2*>],
....
<*Attrn*> <Dom*n*> [**not null**] [**default** <val*2*>],
<**contrainte***1*>,
...
<**contrainte***n*>
)

- *nom* : nom de la table
- *Atti* : nom d'un attribut
- *Domi* : type/domaine
- *Vali* : valeur par défaut
- **not null** : la valeur doit être renseignée
- contrainte*i* : contrainte d'intégrité

Langage de définition de données (LDD / DDL)
Commandes pour créer, modifier et supprimer les éléments du schéma relationnel tel que relations, vues, indexes, contraintes, database …

CREATE | ALTER | DROP DATABASE
CREATE | ALTER | DROP TABLE
CREATE | ALTER | DROP VIEW
CREATE | ALTER | DROP TRIGGER
CREATE | ALTER | DROP INDEX
CREATE | ALTER | DROP CONSTRAINT
CREATE | ALTER | DROP ...

Langage de définition de données : Création des Bases de données

CREATE DATABASE
Syntaxe: CREATE DATABASE <nom>
Effets: Crée une base de donnée vide et prête à l'emploi
Efface les éventuelles données présentes

ALTER DATABASE
Syntaxe: ALTER DATABASE <nom>
Effets: Permet d'effectuer des opérations de maintenance

DROP DATABASE <nom>
Syntaxe: DROP DATABASE <nom>
Effets: Efface la base de données entière

Langage de définition de données : Création des Tables

```
CREATE TABLE nom_table (
        nom-col type-col [ [CONSTRAINT] contrainte-col]*
        [CONSTRAINT contrainte-table]*
    ) ;
```

Effets: Crée une table vide en spécifiant son nom, ses attributs et ses contraintes
Légende: […] = optionnel, CAPITALE = mot clé,
* = répétition possible (typiquement separés par des virgule)

```
CREATE TABLE AS requête-SQL
```

Effets: Permet de créer et peupler une table à partir d'une requête SQL

Application

Créer une table Artiste contenant un Nom (identifiant), un genre et une nationalité.

Insérer un jeux de données et afficher la table

Contraintes d'intégrités: Contraintes de clés

Clé candidate : attributs dont les valeurs sont distinctes pour tous les n-uplets (valeurs null possibles)

Clé primaire : clé candidate dont chacun des attributs est renseigné

Clé étrangère : attributs dont les valeurs proviennent d'une clé candidate ou d'une clé primaire définie dans la même table ou dans une autre table

Clés candidates

unique(*a1,..,an***)**

Clé primaire

primary key(*a1,..,an***)**

ai et *bj* sont des attributs

Clés étrangères

foreign key(*a1,..,an***) references** *table*
foreign key(*a1,..,an***) references** *table* (*b1,..,bn*)

Concepts clés: Contraintes

Contraintes d'intégrité sur une table

Les contraintes d'intégrité sur une table sont :

* **PRIMARY KEY** (<liste d'attibuts>) : définit les attributs de la liste comme la clé primaire

* **UNIQUE** (<liste d'attibuts>) : interdit que deux tuples de la relation aient les mêmes valeurs pour l'ensemble des attributs de la liste.

* **FOREIGN KEY** (<liste d'attibuts>) REFERENCES <nom table>(<nom colonnes>) : contrôle l'intégrité référentielle entre les attributs de la liste et la table et ses colonnes spécifiées

* **CHECK** (<condition>) : contrôle la validité de la valeur des attributs spécifiés dans la condition dans le cadre d'une restriction de domaine

```
CREATE TABLE Personne (
  N°SS CHAR(13) PRIMARY KEY,
  Nom VARCHAR(25) NOT NULL,
  Prenom VARCHAR(25) NOT NULL,
  Age INTEGER(3) CHECK (Age BETWEEN 18 AND 65),
  Mariage CHAR(13) REFERENCES Personne(N°SS),
  UNIQUE (Nom, Prenom)
);
```

Clé candidate
La clause UNIQUE NOT NULL sur un attribut ou un groupe d'attributs définit une clé candidate non primaire.

Exemple

```sql
CREATE TABLE Personne (
  N°SS CHAR(13) PRIMARY KEY,
  Nom VARCHAR(25) NOT NULL,
  Prenom VARCHAR(25) NOT NULL,
  Age INTEGER(3) CHECK (Age BETWEEN 18 AND 65),
  Mariage CHAR(13) REFERENCES Personne(N°SS),
  Codepostal INTEGER(5),
  Pays VARCHAR(50),
  UNIQUE (Nom, Prenom),
  FOREIGN KEY (Codepostal, Pays) REFERENCES Adresse (CP, Pays)
);

CREATE TABLE Adresse (
  CP INTEGER(5) NOT NULL,
  Pays VARCHAR(50) NOT NULL,
  Initiale CHAR(1) CHECK (Initiale = LEFT(Pays, 1)),
  PRIMARY KEY (CP, Pays)
);
```

```sql
CREATE TABLE Personne (
  N°SS CHAR(13) ,
  Nom VARCHAR(25) NOT NULL,
  Prenom VARCHAR(25) NOT NULL,
  Age INTEGER(3) ,
  Mariage CHAR(13),
  Codepostal INTEGER(5),
  Pays VARCHAR(50),
  PRIMARY KEY (N°SS),
  UNIQUE (Nom, Prenom),
  CHECK (Age BETWEEN 18 AND 65),
  FOREIGN KEY (Mariage) REFERENCES Personne(N°SS),
  FOREIGN KEY (Codepostal, Pays) REFERENCES Adresse (CP, Pays)
);

CREATE TABLE Adresse (
  CP INTEGER(5) NOT NULL,
  Pays VARCHAR(50) NOT NULL,
  Initiale CHAR(1),
  PRIMARY KEY (CP, Pays),
  CHECK (Initiale = LEFT(Pays, 1))
);
```

Contraintes d'intégrités: Contraintes générales

Contraintes sur une seule table

- **check** <*predicat*> où *predicat* est une Condition n'utilisant qu'une seule table

Contraintes sur plusieurs tables

- **create assertion** <nom>
  check <*predicat-complexe*>
  où *predicat-complexe* est une Condition faisant appel à plusieurs tables

Contraintes d'intégrités: Contraintes générales

Contraintes sur une seule table : exemple

Empêcher qu'il y ait plus de 7 modules par niveau d'études

```
create table Module(
    code varchar(10) ,
    intitule varchar(20),
    niveau char(2),
    suit varchar(10),
    …
    constraint modules_max
        check (all (select count(*)
                    from Module
                    group by niveau)<8)
    );
```

Contraintes sur plusieurs tables : exemple

```
create table Etudiant(
    eid numeric(8),
    nom varchar(10),
    moy numeric(4,2)
    …);
```

```
create table Eval(
    eid numeric(8), /*etud*/
    mid char(3), /*module*/
    note numeric(4,2) /*note*/
    …);
```

```
create assertion verif_moy
    check ( not exists (
            select * from Etudiant
            where moy <> ( select avg(note)
                           from Eval
                           where Eval.eid=Etudiant.eid ) ) );
```

L'attribut Etudiant.moy = moyenne des notes (Eval.note) de l'étudiant

Application

Modules

code	intitulé	niveau	suit	salle	resp
LI341	Bases de Données	L3	NULL	105	479
LI345	BD Web	L3	LI341	105	845
LI399	Crypto	L3	LI345	302	214

Salles

numéro	nbPlaces
105	102
302	46
112	10

Enseignants

matricule	nom
479	Josh
845	Lana
214	Liz

Etudiants

matricule	..
1753	
0148	
…	

Inscriptions

etud	module	anneU
1753	LI341	2012
1753	LI345	2013
0148	LI399	2012

annee
2012
2013

- Des clés étrangères peuvent composer la clé candidate/primaire d'une relation:
  - *Exemple:* la clé {etud, module} de la table **Inscriptions**

Ecrire les Modèle Logique et Physique correspondant au schéma suivant.

Ressources Utiles

Page Wikipedia sur SQL: https://fr.wikipedia.org/wiki/Structured_Query_Language

MySQL(si c'est pas déja fait): http://dev.mysql.com/downloads/mysql/#downloads
MySQL Workbench:
https://dev.mysql.com/downloads/workbench/
https://www.youtube.com/watch?v=X_umYKqKaF0

Tutos sur SQL :

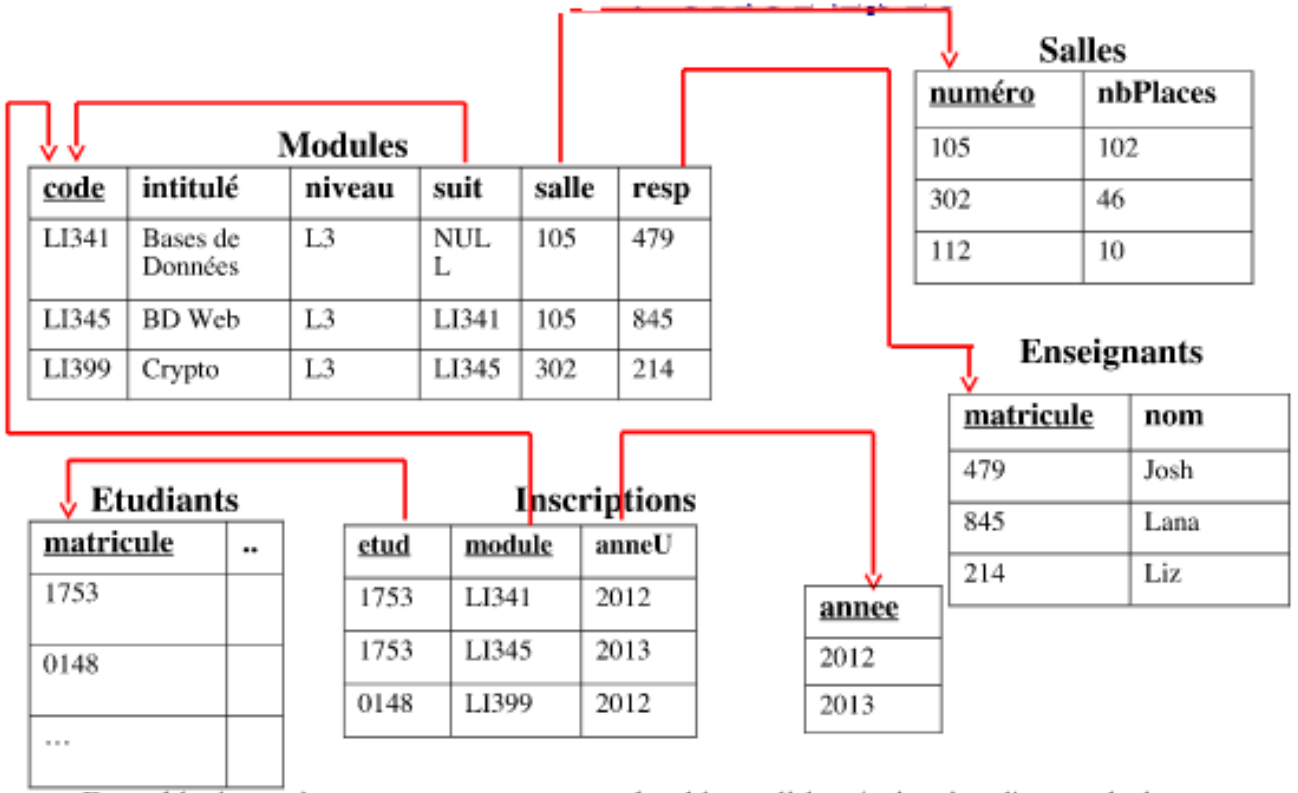
- https://sql.sh/
- https://sql.developpez.com/
- https://www.w3schools.com/sql/
- Bonnes pratiques : https://www.sqlstyle.guide/

https://www.mysqltutorial.org/mysql-data-types.aspx/

Un trés bon mooc très complet!
http://sdz.tdct.org/sdz/administrez-vos-bases-de-donnees-avec-mysql.html

Création du Schéma et les contraintes d’intégrité

Application: Retour sur Université



Salles

numéro	nbPlaces
105	102
302	46
112	10

Modules

code	intitulé	niveau	suit	salle	resp
LI341	Bases de Données	L3	NULL	105	479
LI345	BD Web	L3	LI341	105	845
LI399	Crypto	L3	LI345	302	214

Enseignants

matricule	nom
479	Josh
845	Lana
214	Liz

Etudiants

matricule	..
1753	
0148	
…	

Inscriptions

etud	module	anneU
1753	LI341	2012
1753	LI345	2013
0148	LI399	2012

annee
2012
2013

- Des clés étrangères peuvent composer la clé candidate/primaire d'une relation:
  - *Exemple:* la clé {etud, module} de la table **Inscriptions**

Contraintes d'intégrités: Retour sur Université

Module(code : varchar, intitule : varchar, niveau : varchar,
suit*: varchar, salle* : number)
→ code *clé primaire;* suit ***et*** salle *référencent les tables* Module *et* Salle *respectivement.*

Salles(numero: number, nbPlaces: number)
→ numero *clé primaire*

```
create table Module(
      code varchar(10) ,
      intitule varchar(20),
      niveau char(2),
      suit varchar(10),
      salle numeric(5),
      primary key(code),
      foreign key suit references Module,
      foreign key salle references Salles
      );
```

```
create table Salles(
      numero numeric(5),
      nbPlaces numeric(3),
      primary key(numero)
      );
```

Modification des données: insertion

insert into *table* **values** ('v_1', 'v_2',..,'v_n');
l'ordre des v_i respecte le schéma de *table*

insert into table(a_1, a_2,..,a_m)
values ('v_1', 'v_2',..,'v_m');
l'ordre des v_j respecte (a_1, a_2,..,a_m)
les attributs non considérés mis à Null

Rq : les ' et ' facultatifs pour les types numériques

```
create table Etudiant(
    eid numeric(8),
    nom varchar(10),
    moy numeric(4,2)
    primary key (eid)
);
```

```
create table Module(
    code varchar(10) ,
    intitule varchar(20),
    niveau char(2),
    suit varchar(10),
    primary key(code)
);
```

```
insert into Etudiant values (1243, 'joe',12.5);
insert into Etudiant values (1244, 'doe',null);
insert into Module values ('L234','Techno','L2',null );
insert into Module(code, intitule) values ('L101','logique');
```

Modification des données: insertion

Insérer le résultat d'une requête

insert into *table*

select att1, …

from autreTable, …

where Condition ;

```
create table Etudiant(
    eid numeric(8),
    nom varchar(10),
    moy numeric(4,2)
    primary key (eid)
);
```

```
create table MajorsPromo(
    eid numeric(8),
    promo char(2),
    moy numeric(4,2),
    primary key(eid, promo)
    foreign key (eid) references
        Etudiant);
```

- Insérer dans MajorsPromo le major du L2

```
insert into MajorsPromo
        select eid, 'L2', moy
        from Etudiant
        where moy = (select max (moy) from Etudiant);
```

Modification des données: Mise à jour

Utile pour modifier les valeurs d'un sous-ensemble d'attributs

update *table*
set attr = *Expression* | *valeur*
where *Condition ;*

où

- *Expression* peut combiner attr avec d'autres attributs de *table* ou avec des valeurs
- *Condition* est comme celle de la clause where

```
create table Etudiant(
    eid numeric(8),
    nom varchar(10),
    moy numeric(4,2)
    primary key (eid));
```

```
create table Eval(
    eid numeric(8),
    mid char(3),
    note numeric(4,2),
    primary key(eid, mid));
```

- Remonter la moyenne de tous les étudiants de 10%

```
update  Etudiant set moy=moy*1.1;
```

- Racheter les étudiants ayant obtenu 9.5

```
update  Etudiant set moy=10 where moy between 9.5 and 10;
```

- Affecter aux étudiants ayant obtenu la plus petite moy. null

```
update  Etudiant set moy=null
where moy < 5 and  moy=(select min(moy) from Etudiant);
```

Modification des données: Mise à jour

Utile pour modifier les valeurs d'un sous-ensemble d'attributs

update *table*

set attr = sous-requête ;

- sous-requête doit retourner un singleton et un seul attribut

```
update Etudiant E
set moy = (select avg(note)
           from Eval
           where E.eid=Eval.eid);
```

```
create table Etudiant(
    eid numeric(8),
    nom varchar(10),
    moy numeric(4,2)
    primary key (eid));
```

```
create table Eval(
    eid numeric(8),
    mid char(3),
    note numeric(4,2),
    primary key(eid, mid));
```

- Renseigner la moyenne de chaque étudiant

```
update Etudiant
set moy = (select avg(note)
           from Eval v, Etudiant e where v.eid=e.eid);
```

Que produit cette instruction?

Modification des données: Suppression

Opération inverse de l'insertion

delete from *table*

where Condition ;

La syntaxe ressemble à celle des requêtes,

La clause where est facultative

```
create table Etudiant(
    eid numeric(8),
    nom varchar(10),
    moy numeric(4,2)
    primary key (eid));
```

```
create table Eval(
    eid numeric(8),
    mid char(3),
    note numeric(4,2) not null,
    primary key(eid, mid));
```

Supprimer toutes les notes

```
delete from Eval;
```

Supprimer l'étudiant n'ayant passé aucun examen

```
delete from Etudiant
where eid not in (select eid from Eval);
```

Supprimer les lignes de Eval dont aucune note ne dépasse 7

```
delete from Eval where mid in
 (select mid from Eval group by mid having max (note)<7);
```

Contraintes de clés étrangères et mises à jour

- Par défaut, si une MAJ tente de supprimer/ modifier des nuplets référencés par d'autres elle est **annulée**
- Possibilité de spécifier un traitement alternatif
  - Propagation de la suppression/modification
  - Affectations de valeurs par défaut

foreign key(a_1,..,a_n) **references** *table* (b_1,..,b_n)

on **delete** | **update** : **restrict** | **cascade** | **set default** | **null**

- **restrict** annule la mise à jour
- **set** permet de mettre la valeur par défaut ou null
- **cascade** propage son effet

Suppression nuplets référencés➔suppression nuplets dépendants
modification nuplets référencés➔modification nuplets dépendants

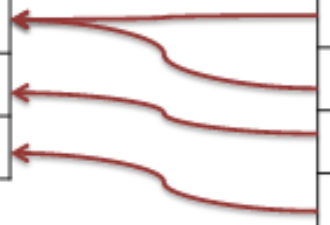
Contraintes de clés étrangères et mises à jour

```
create table Etudiant(
        eid numeric(8),
        nom varchar(10),
        moy numeric(4,2)
        primary key (eid)
);
```

```
create table Eval(
        eid numeric(8),
        mid char(3),
        note numeric(4,2) not null,
        primary key(eid, mid),
        foreign key  (eid) references Etudiant
                on delete cascade,
                on update cascade);
```

eid	nom	moy
12345	Joe	15.25
2471	Doe	13.5
1024	Lars	16

eid	mid	note
12345	L21	12
12345	L17	10
2471	L21	18
1024	L21	14



Modification du schéma d'une BD

Le langage de définition des données permet

- Création des tables (avec contraintes d'intégrité)
- Suppression de table
  drop table nom [**cascade constraints**] ;
- Modification des tables
  alter table nom <instr_modif_table>
  où instr_modif_table peut être
  - **rename to** autre_nom
  - **drop** attrib
  - **add** attrib domaine

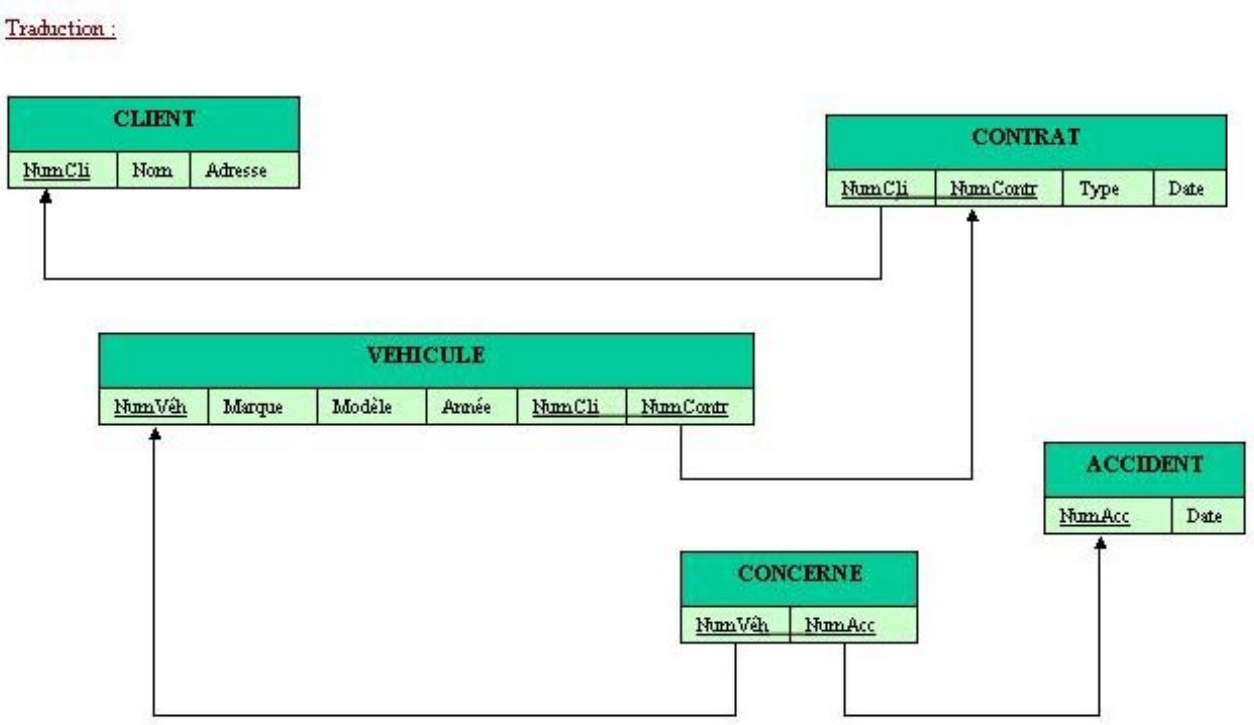
```
create table Etudiant(
    eid numeric(8),
    nom varchar(10),
    moy numeric(4,2)
    primary key (eid));
```

```
create table Eval(
    eid numeric(8),
    mid char(3),
    note numeric(4,2),
    primary key(eid, mid));
```

Supprimer la table Eval
drop table eval;
Renommer la table Etudiant en Etudiants
alter table Etudiant Rename to Etudiants;
Enlever l'attribut moy de la table Etudiant
alter table Etudiant drop moy;
Rajouter à un la table Etudiant attribut date naissance de type date devant être renseigné
alter table Etudiant add dateNaiss Date Not Null;

Application

Traduction du schéma relationnel suivant en SQL



Les Requêtes en SQL

Les Requêtes en SQL

SQL: Syntaxe Simplifiée d'une Requête

```
SELECT liste-colonnes
FROM Table
[WHERE condition] ;
```

Retourne

- Les attributs de liste-colonnes
- Des enregistrements de la table *Table*
- qui vérifient condition

La clause where est facultative mais très utile

SQL: Syntaxe Simplifiée

- La clause from déclare les variables (calcul)
  - § Par défaut nom de la relation : from R, S
  - § on peut renommer : from R v1, S v2...
- Pour retourner toutes les colonnes
  - § Select *
- Sémantique « multi-ensembliste »:
  - § Possibilité d'avoir des doublons

    (parce que les éliminer coûte cher, parce qu'on peut vouloir les compter,...)
  - § Les éliminer avec le mot clé **distinct**

    select DISTINCT

SQL: Syntaxe Simplifiée

- Possibilité d'exprimer des opérations arithmétiques
  - § (att1+att2, att*1.5, etc)
- Possibilité de retourner des chaines entre ' '
- Possibilité de préfixer les attributs par le nom de la table ou une variable
  - § Lever les ambiguïtés de noms d'attributs
- Possibilité de renommer une colonne dans le SELECT avec le mot-clé AS
  - § Lisibilité des résultats

Modèle d’une requête SQL

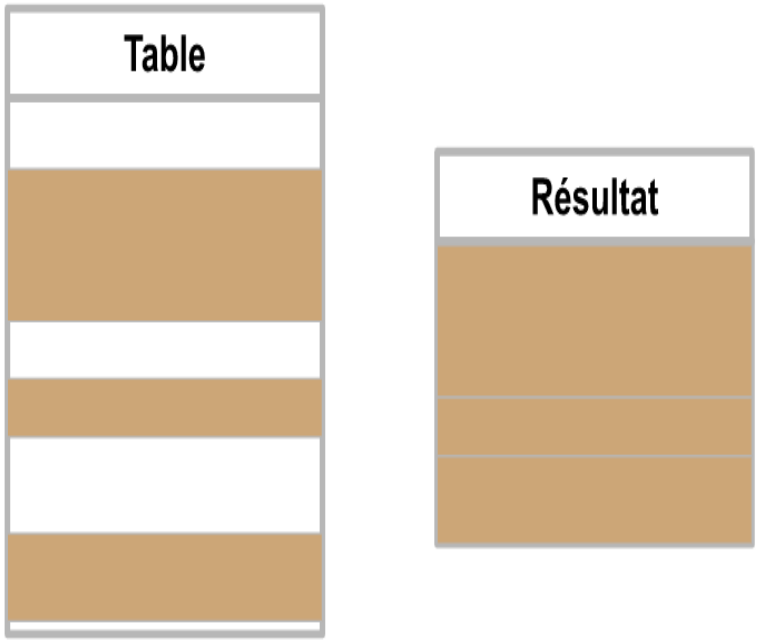
```
SELECT colonnes, calculs
    FROM tables
    WHERE conditions
    GROUP BY colonnes
    HAVING conditions;
```

Requetage Simple: Selection

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



Une restriction est une sélection de lignes **d'une** table, sur la base d'une **condition** à respecter, définie à la suite du terme **WHERE**.

Il permet de ne retenir que les n-uplets répondant à la condition de sélection exprimée derrière la clause WHERE et qui peut être spécifiée à l'aide :
* des opérateurs de comparaison : =, >, <, <=, >=, <>
* des opérateurs logiques : AND, OR, NOT
* des opérateurs : IN, BETWEEN, LIKE, IS

SELECT * FROM table WHERE condition ;

Requetage Simple: Selection

EMP

ENO	ENAME	TITLE
E1	J. Doe	Elect. Eng.
E2	M. Smith	Syst. Anal.
E3	A. Lee	Mech. Eng.
E4	J. Miller	Programmer
E5	B. Casey	Syst. Anal.
E6	L. Chu	Elect. Eng.
E7	R. Davis	Mech. Eng.
E8	J. Jones	Syst. Anal.

```
SELECT * FROM EMP
WHERE TITLE = 'Elect. Eng.' ;
```

ENO	ENAME	TITLE
E1	J. Doe	Elect. Eng
E6	L. Chu	Elect. Eng.

Tous les attributs de la relation sont conservés.

Un attribut peut ne pas avoir été renseigné pour certains n-uplets. Si une condition de sélection doit en tenir compte, on indiquera simplement : nom_attribut "non renseigné".

Requetage Simple

EMP

ENO	ENAME	TITLE
E1	J. Doe	Elect. Eng
E2	**M. Smith**	**Analyst**
E3	A. Lee	Mech. Eng.
E4	J. Miller	Programmer
E5	**B. Casey**	**Analyst**
E6	L. Chu	Elect. Eng.
E7	R. Davis	Mech. Eng.
E8	**J. Jones**	**Analyst**

PAY

TITLE	SALARY
Elect. Eng.	55000
Analyst	70000
Mech. Eng.	45000
Programmer	60000

SELECT **ENO, ENAME, EMP.TITLE, SALARY**
FROM EMP, PAY
WHERE **EMP.TITLE='Analyst'**

ENO	ENAME	EMP.TITLE	SALARY
E2	M. Smith	Analyst	55000
E2	M. Smith	Analyst	70000
E2	M. Smith	Analyst	45000
E2	M. Smith	Analyst	60000
E5	B. Casey	Analyst	55000
E5	B. Casey	Analyst	70000
E5	B. Casey	Analyst	45000
E5	B. Casey	Analyst	60000
E8	J. Jones	Analyst	55000
E8	J. Jones	Analyst	70000
E8	J. Jones	Analyst	45000
E8	J. Jones	Analyst	60000

Une restriction est une sélection de lignes d'une table, sur la base d'une **condition** à respecter, définie à la suite du terme **WHERE**.
Cette condition peut être simple {>, =, <=, >=, <>} ou complexe: une combinaison de comparaisons à l'aide de `AND`, de `OR` et de `NOT` (attention donc aux parenthèses dans ce cas).

IN, BETWEEN, LIKE

- Appartenance à un ensemble de valeurs :

  Att IN (Const1, Const2, ...)

- Appartenance à un intervalle de valeurs :

  Att BETWEEN Constante1 AND Constante2

- Ressemblance à un motif :

  Att LIKE 'MOTIF'

  § où MOTIF combine des chaînes et des joker
    - % pour une chaîne quelconque (y compris vide)
    - _ pour un caractère quelconque et un seul

Les Valeurs Nulles

- La valeur de certains attributs peut
  - ne pas être connue (ex. : année de construction du Louvre)
  - ou ne pas avoir de sens (ex. : nom de jeune fille pour un homme)

on parle alors de valeurs nulles (mot-clé NULL)

- NULL n'est pas une valeur mais une absence de valeur! Les opérations ou les comparaisons ne peuvent lui être appliqué

- Toute opération (+,-,/,*,substr, to_char,...) appliquée à NULL donne NULL

- Toute comparaison avec NULL donne ni vrai ni faux, mais INCONNU

  Notions de sémantique Tri-Valuée abordées en cours 8 : compléments SQL

Requetage Simple: selection

Soit la table **ETUDIANT (N°Etudiant, Nom, Age, CodePostal, Ville)**

* Trouver les étudiants agés entre 19 et 23 ans

*Trouver les étudiants habitant à Paris

*Trouver les étudiants pour lesquels la ville n'est pas renseignée

*Trouver les étudiants pour lesquels la ville est renseignée

Requetage Simple: Projection

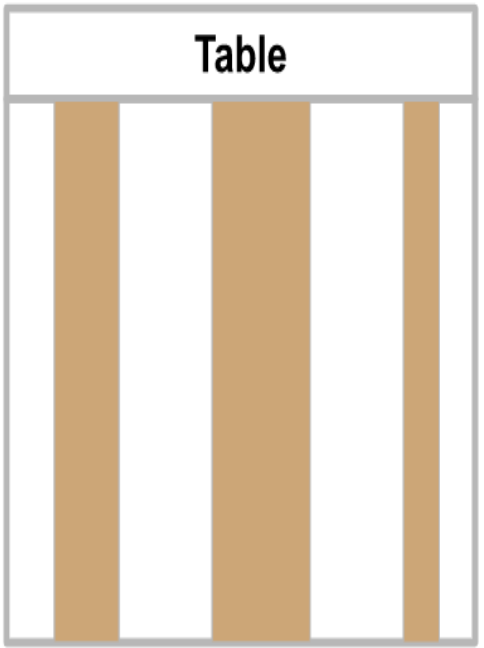
Projection

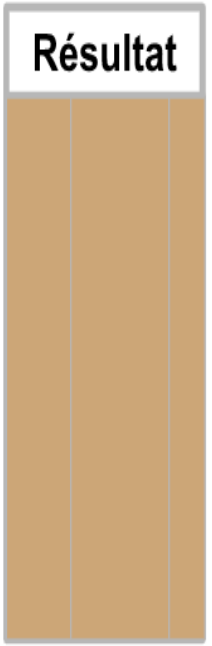
Sélection d'un nombre restreint de colonnes (ou attributs) d'une table

Ex : les noms et prénoms uniquement des étudiants

Attention : il y a risque d'avoir des doublons

Ex : le sexe des étudiants





Une projection est une sélection de colonnes d'une table, sur la base d'une liste d'attributs placés après le `SELECT` et séparés par une virgule.

SELECT DISTINCT liste d'attributs FROM table ;

SELECT liste d'attributs FROM table ;

Syntaxe de tri: Order By

- Dans la clause ORDER BY, on peut avoir des :
  - des noms de colonnes
  - des expressions avec noms de colonnes
  - des numéros de position des colonnes dans la clause SELECT.
- On précise le sens : ASC (par défaut) ou DESC
- Les valeurs nulles sont à la fin par ordre croissant, au début par ordre décroissant.

```
SELECT liste-colonnes
FROM nomtable
WHERE condition
ORDER BY liste-colonnes ;
```

NUMPROJ	NOMPROJ	BUDGET
1	aa	20
2	bb	100
3	ab	30
4	aa	200

select * from project order by nomproj , budget desc;

NUMPROJ	NOMPROJ	BUDGET
4	aa	200
1	aa	20
3	ab	30
2	bb	100

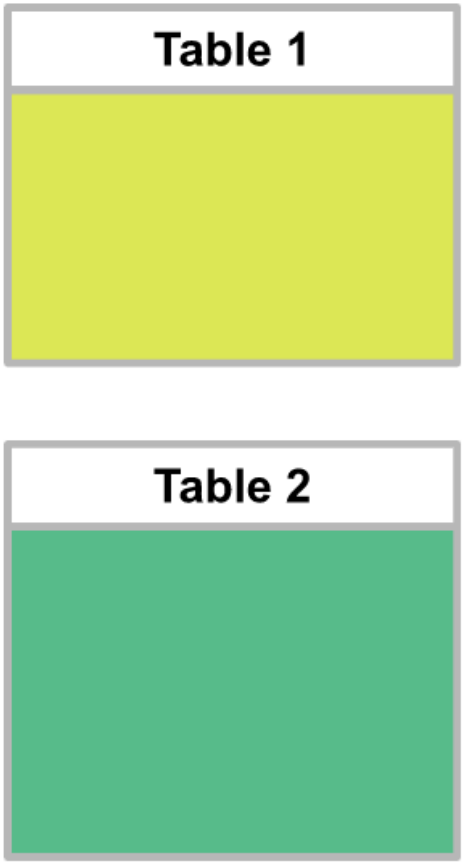
Opérations Ensemblistes

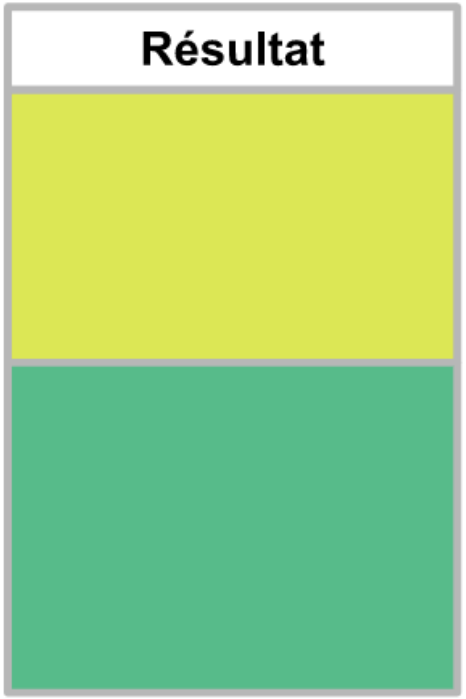
Opérations ensemblistes

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





Opérations Ensemblistes

On peut réaliser des opérations ensemblistes sur les clauses SELECT.

3 opérations ensemblistes

UNION	union de deux ensembles
INTERSECT	intersection de deux ensembles
MINUS	différence de deux ensembles (norme : EXCEPT)

Pour les opérations ensemblistes :

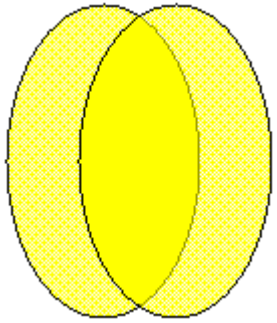
- Pas de lien entre les objets sélectionnés dans les 2 requêtes
- Même schéma dans les SELECT des deux requêtes : c'est à dire même nombre d'attributs et chacun du même type (par forcément le même nom)
- Le schéma en sortie correspond au schéma de la première requête
- Par défaut, les opérations ensemblistes éliminent les doublons (ensemble). Pour garder les doublons (multi-ensemble), il faut ajouter ALL après l'opérateur : UNION ALL, EXCEPT ALL, INTERSECT ALL

Opérations Ensemblistes: Union

Cet opérateur porte sur 2 relations qui doivent avoir le même nombre d'attributs définis dans le même domaine (ensemble des valeurs permises pour un attribut). On parle de relations ayant le même schéma.

La relation résultat possède les attributs des relations d'origine et les n-uplets de chacune, avec élimination des doublons éventuels.

RQ: Les champs que l'on fait correspondre dans les deux tables n'ont pas besoin de porter les mêmes noms ni de se présenter dans le même ordre -- ni même de posséder le même type de données si la transposition est possible (une date en texte, par exemple).



SELECT liste d'attributs FROM table1
UNION
SELECT liste d'attributs FROM table 2 ;

Opérations Ensemblistes: Union

nom	*prénom*
Chose	Jules
Machin	Pierre
Truc	Patrick

Table1

union

nom	*prénom*
Pouf	Jean
Chose	Jules

Table2

=

nom	*prénom*
Chose	Jules
Machin	Pierre
Pouf	Jean
Truc	Patrick

Résultat

```
SELECT nom, prénom
FROM Table1
UNION
SELECT nom, prénom
FROM Table2;
```

nom	*prénom*
Chose	Jules
Machin	Pierre
Truc	Patrick

Table1

union

last-name	*first-name*
Pouf	Jean
Chose	Jules

Table2

=

nom	*prénom*
Chose	Jules
Machin	Pierre
Pouf	Jean
Truc	Patrick

Résultat

```
SELECT nom, prénom
FROM Table1
UNION
SELECT lastname, firstname
FROM Table2;
```

Opérations Ensemblistes: Union

nom	*prénom*
Chose	Jules
Machin	Pierre
Truc	Patrick

Table1

union

nom	*prénom*
Pouf	Jean
Chose	Jules

Table2

=

Col1	*Col2*
Chose	Jules
Machin	Pierre
Pouf	Jean
Truc	Patrick

Résultat

```
SELECT nom AS Col1, prénom AS Col2
FROM Table1
UNION
SELECT nom AS Col1, prénom AS Col2
FROM Table2;
```

nom	*prénom*
Chose	Jules
Machin	Pierre
Truc	Patrick

Table1

union all

nom	*prénom*
Pouf	Jean
Chose	Jules

Table2

=

Col1	*Col2*
Chose	Jules
Machin	Pierre
Truc	Patrick
Pouf	Jean
Chose	Jules

Résultat

```
SELECT nom, prénom
FROM Table1
UNION ALL
SELECT nom, prénom
FROM Table2;
```

Opérations Ensemblistes: Union

nom	*prénom*
Chose	Jules
Machin	Pierre
Truc	Patrick

Table1

union

nom	*prénom*
Pouf	Jean
Chose	Jules

Table2

=

Col1	*Col2*
Chose	Jules
Machin	Pierre
Pouf	Jean
Truc	Patrick

Résultat

```
SELECT nom AS Col1, prénom AS Col2
FROM Table1
UNION
SELECT nom AS Col1, prénom AS Col2
FROM Table2;
```

nom	*prénom*
Chose	Jules
Machin	Pierre
Truc	Patrick

Table1

union all

nom	*prénom*
Pouf	Jean
Chose	Jules

Table2

=

Col1	*Col2*
Chose	Jules
Machin	Pierre
Truc	Patrick
Pouf	Jean
Chose	Jules

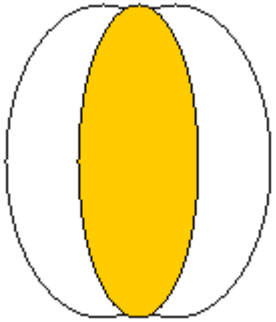
Résultat

```
SELECT nom, prénom
FROM Table1
UNION ALL
SELECT nom, prénom
FROM Table2;
```

Opérations Ensemblistes: Intersection

Cet opérateur porte sur deux relations de même schéma.

La relation résultat possède les attributs des relations d'origine et les n-uplets communs à chacune => ça ne retourne que les lignes communes aux deux tables.



```
SELECT attribut1, attribut2, ... FROM table1
WHERE attribut1 IN (SELECT attribut1 FROM table2) ;

SELECT attribut1, attribut2, ... FROM table1
INTERSECT
SELECT attribut1, attribut2, ... FROM table2 ;
```

Opérations Ensemblistes: Intersection

nom	*prénom*
Chose	Jules
Machin	Pierre
Truc	Patrick

inter

nom	*prénom*
Pouf	Jean
Chose	Jules

=

nom	*prénom*
Chose	Jules

```
SELECT nom, prénom
FROM Table1
INTERSECT
SELECT nom, prénom
FROM Table2;
```

```
SELECT nom, prénom
FROM Table1
WHERE Table1.nom IN (SELECT nom FROM Table2) AND Table1.prénom IN (SELECT prénom FROM Table2);
```

```
SELECT DISTINCT Table1.nom, Table1.prénom
FROM Table1, Table2
WHERE Table1.nom=Table2.nom AND Table1.prénom=Table2.prénom;
```

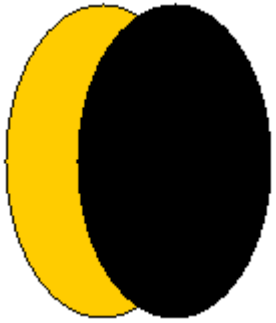
```
SELECT Table1.nom, Table1.prénom
FROM Table1 INNER JOIN Table2 ON Table1.nom = Table2.nom AND Table1.prénom = Table2.prénom
WHERE Table2.nom Is Not Null AND Table2.prénom Is Not Null;
```

Opérations Ensemblistes: Différence

Cet opérateur porte sur deux relations de même schéma.

La relation résultat possède les attributs des relations d'origine et les n-uplets de la première relation qui n'appartiennent pas à la deuxième => ça retourne les enregistrements de la première table qu'on ne retrouve pas dans la seconde

Attention ! DIFFERENCE (R1, R2) ne donne pas le même résultat que DIFFERENCE (R2, R1)



```
SELECT attribut1, attribut2, ... FROM table1
WHERE attribut1 NOT IN (SELECT attribut1 FROM table2) ;

SELECT table1.attribut1, table1.attribut2,...
FROM table1 LEFT OUTER JOIN table2 ON table1.attribut1 =
table2.attribut1
WHERE table2.attribut1 IS NULL ;
```

Opérations Ensemblistes: Différence

nom	*prénom*
Chose	Jules
Machin	Pierre
Truc	Patrick

diff

nom	*prénom*
Pouf	Jean
Chose	Jules

=

nom	*prénom*
Machin	Pierre
Truc	Patrick

```
SELECT nom, prénom
FROM Table1
EXCEPT
SELECT nom, prénom
FROM Table2;
```

```
SELECT nom, prénom
FROM Table1
WHERE Table1.nom NOT IN (SELECT nom FROM Table2) AND Table1.prénom NOT IN (SELECT prénom FROM Table2);
```

```
SELECT Table1.nom, Table1.prénom
FROM Table1 LEFT JOIN Table2 ON Table1.prénom = Table2.prénom AND Table1.nom = Table2.nom
WHERE Table2.nom Is Null AND Table2.prénom Is Null;
```

Opérations Ensemblistes: Produit Cartésien

Cet opérateur porte sur deux relations.

La relation résultat est obtenue en associant tous les enregistrements de la seconde table à chacun des enregistrements de la première.

SELECT * FROM table1, table2 ;

nom
Chose
Machin

X

prénom
Jean
Jules

=

nom	*prénom*
Chose	Jean
Chose	Jules
Machin	Jean
Machin	Jules

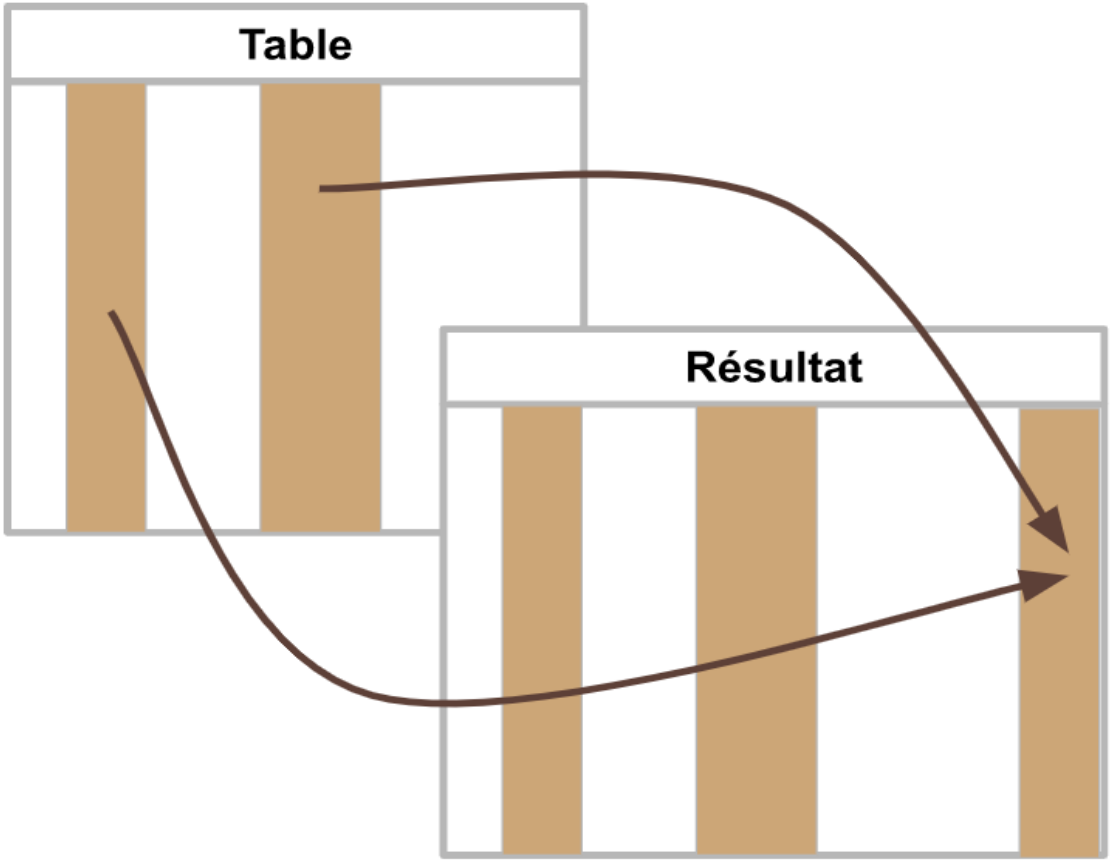
```
SELECT Table1.nom, Table2.prénom
FROM Table1, Table2;
```

Calculs et Fonctions

Calcul

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

Ex : Les initiales de l'étudiant



Fonctions

De nombreuses fonctions existent pour :

- Manipuler les dates
- Manipuler les chaînes de caractères
- Manipuler les chiffres

Cependant, bien qu'une norme existe, elles diffèrent souvent d'un SGBD à l'autre...

Les Fonctions de Date

Date/Heure actuelle

Nom	Description
CURDATE()	**CURDATE()** retourne la date actuelle (ex: "2006-01-14"). Lorsque CURDATE() est affectée à un champ de type DATETIME, l'heure 00:00:00 est automatiquement ajoutée (ex: "2006-01-14 00:00:00"), ce qui correspond à la date du jour, à minuit.
CURTIME()	**CURTIME()** retourne l'heure actuelle (ex: "15:41:32").
NOW()	**NOW()** retourne la date et l'heure actuelles (ex: "2006-01-14 15:41:32").
SYSDATE()	**SYSDATE()** retourne la date et l'heure courante. La différence avec NOW() est que SYSDATE() retourne l'heure à laquelle elle est effectivement appelée, alors que NOW() retourne l'heure de début de script.

Les Fonctions de Date

Formatage d'une date

La fonction DATE_FORMAT permet le formatage d'une date dans le format désiré.

D[illegible]mat)

%%	Le caractère '%'
%d	Jour du mois, numérique (00..31)
%m	Mois de l'année, numérique (01..12)
%Y	Année, sur 4 chiffres (YYYY)
%H	Heures, sur 24 heures (00..23)
%i	Minutes (00..59)
%s	Secondes (00..59)
%T	Heure complète (hh:mm:ss)

Les Fonctions de Date

Conversion d'une chaîne en date

Pour créer une date/heure à partir d'une chaine, il suffit généralement de mettre la chaîne dans l'un des format suivants :

*Pour les dates : "YYYY-MM-DD" "YYYYMMDD" "YY-MM-DD" "YYMMDD"

*Pour les heures : "hh:mm:ss" "hhmmss"

*Pour les date-heure : "YYYY-MM-DD hh:mm:ss" "YY-MM-DD hh:mm:ss" "YYYYMMDDhhmmss" "YYMMDDhhmmss"

Nom	Description
STR_TO_DATE()	**STR_TO_DATE(*chaine*, *format*)** est la fonction inverse de DATE_FORMAT. Evalue une chaîne de caractère et la convertit en date/heure en se basant sur le *format* spécifié. (voir la fonction DATE_FORMAT() pour la description du format).

Les Fonctions de Date

Ajout/soustraction d'intervalles de temps

Nom	Description
DATE_ADD()	**DATE_ADD(*date*, INTERVAL *nb uniteDeTemps*)** Ajoute *nb* unités de temps à la date passée en paramètre. Les différentes valeurs d'*uniteDeTemps* sont données dans le tableau suivant. Synonyme: ADDDATE(*date*, INTERVAL *nb uniteDeTemps*).
DATE_SUB()	**DATE_SUB()** se comporte comme DATE_ADD, mais pour la soustraction. Synonyme: SUBDATE(*date*, INTERVAL *nb uniteDeTemps*).
ADDTIME()	**ADDTIME(*date*, *expr*)** ajoute l'expression *expr* à la date/heure passée en paramètre. Exemple: SELECT ADDTIME('1997-12-31 23:59:59.999999', '1 1:1:1.000002'); -> '1998-01-02 01:01:01.000001'
SUBTIME()	**SUBTIME()** se comporte comme ADDTIME, mais pour la soustraction.

Unité de temps	Exemple
MICROSECOND	125 MICROSECOND
SECOND	20 SECOND
MINUTE	30 MINUTE
HOUR	2 HOUR
DAY	3 DAY
WEEK	3 WEEK
MONTH	2 MONTH
QUARTER	3 QUARTER
YEAR	2 YEAR
SECOND_MICROSECOND	20.125 SECOND_MICROSECOND
MINUTE_MICROSECOND	30:20.125 MINUTE_MICROSECOND
MINUTE_SECOND	30:20 MINUTE_SECOND
HOUR_MICROSECOND	2:30:20.125 HOUR_MICROSECOND
HOUR_SECOND	2:30:20 HOUR_SECOND
HOUR_MINUTE	2:30 HOUR_MINUTE
DAY_MICROSECOND	3 2:30:20.125 DAY_MICROSECOND
DAY_SECOND	3 2:30:20 DAY_SECOND
DAY_MINUTE	3 2:30 DAY_MINUTE
DAY_HOUR	3 2 DAY_HOUR
YEAR_MONTH	1-6 YEAR_MONTH

Les Fonctions de Date

Différences entre 2 dates/heures

Nom	Description
DATEDIFF()	**DATEDIFF(expr,expr2)** Retourne le nombre de jours entre la date de début *expr* et la date de fin *expr2*. *expr* et *expr2* sont des expressions de type DATE ou DATETIME. Seule la partie DATE est utilisée dans le calcul. Le résultat peut être négatif. SELECT DATEDIFF('1997-12-31 23:59:59','1997-12-30'); -> 1
TIMEDIFF()	**TIMEDIFF(expr,expr2)** Retourne la durée entre l'heure de début *expr* et l'heure de fin *expr2*. *expr* et *expr2* sont des expressions de type TIME ou DATETIME, et doivent être de même type SELECT TIMEDIFF('2000:01:01 00:00:00', '2000:01:01 00:00:00.000001'); -> '-00:00:00.000001'

Les Fonctions de Date

Comparaison de dates/heures

* Avec les opérateurs de comparaison ==, !=, <, <=, >, >=

```
SELECT * FROM produit WHERE dateLimite < CURDATE() ;
```

* Pour les intervalles de temps, à l'aide de l'opérateur BETWEEN

```
SELECT * FROM produit WHERE dateLimite BETWEEN CURDATE() AND DATE_ADD(CURDATE(), INTERVAL 3 DAY) ;
```

* A l'aide de n'importe quelle fonction sur les dates/heures :

```
SELECT * FROM personne WHERE DATE_FORMAT(dateNaissance, "%d-%m") = DATE_FORMAT(CURDATE(), "%d-%m")
```

Les Fonctions sur les chaines de caractères

Fonction	Desc	Norme	Access	MySQL	Sql Srv	Oracle
Lower/Upper	Mise en minuscules/majusc	O	N	O	O	O
Substring	Extraction sous-chaîne	O	N	O	N	N
Substr	Extraction sous-chaîne	N	N	N	N	O
Position	Position d'une chaîne dans une autre	O	N	O	N	N
Locate	Position d'une chaîne dans une autre	N	O	O	O	O

Les Fonctions sur les chaines de caractères

la longueur d'une chaîne de caractères

CHAR_LENGTH(chaine) , CHARACTER_LENGTH(chaine) , OCTET_LENGTH(chaine) , LENGTH(chaine)

Retourne la longueur de la chaîne de caractères "chaine".

```
SELECT monchamp,CHAR_LENGTH(monchamp) FROM matable;
SELECT monchamp,CHARACTER_LENGTH(monchamp) FROM matable;
SELECT monchamp,LENGTH(monchamp) FROM matable;
SELECT monchamp,OCTET_LENGTH(monchamp) FROM matable;
```

Les fonctions de substitution

LCASE(chaine) , LOWER()
Retourne la chaîne de caractères "chaine" en minuscules.

REPLACE(chaine1,chaine2,chaine3)
Retourne une chaine de caractère dans laquelle toutes les occurences de "chaine2" dans "chaine1" ont été remplacées par "chaine3".

REVERSE(chaine)
Inverse l'ordre des caractères de la chaîne "chaine".

UCASE(chaine), UPPER()
Retourne la chaîne de caractères "chaine" en majuscules.

```
SELECT monchamp,LOWER(monchamp) FROM matable;
SELECT monchamp,REPLACE(monchamp,'Facile',' c''est Facile!') FROM matable;
SELECT monchamp,UPPER(monchamp) FROM matable;
SELECT monchamp,REVERSE(monchamp) FROM matable;
```

Les Fonctions sur les chaines de caractères

CHAR(entier1,entier2,...) : retourne la chaîne de caractères issue de la concaténation des caractères exprimés par leurs codes ascii "entier1","entier2",etc
CONCAT(chaine1,chaine2,…): retourne la chaîne de caractères issue de la concaténation des chaines de caractères "chaine1","chaine2",etc.
ELT(entier,chaine1,chaine2,…):retourne la "entier"-ième chaîne de caractères de la liste "chaine1","chaine2",...
FIELD(chaine1,chaine2,chaine3,...) : retourne la position de "chaine1" dans la liste "chaine2","chaine3",...
FIND_IN_SET(chaine1,chaine2) : retourne la position de "chaine1" dans la liste composée par "chaine2". "chaine2" étant une chaîne de caractère représentant une liste de chaine de caractères séparés par des virgules
INSERT(chaine1,entier1,entier2,chaine2) : insert "chaine2" dans "chaine1" à la position "entier1" en remplaçant "entier2" caractères de "chaine2"
INSTR(chaine1,chaine2) :Retourne la position de "chaine2" dans "chaine1". Le premier caractère étant à la position 1
POSITION(chaine1 IN chaine2) : Retourne la position de "chaine1" dans "chaine2". Le premier caractère étant à la position 1
LEFT(chaine,entier) :Retourne les "entier" premiers caractères de "chaine"
LOAD_FILE(chaine) : retourne la chaîne de caractère correspondant au contenu du fichier "chaine"
LOCATE(chaine1,chaine2) :Retourne la position de "chaine1" dans "chaine2". Le premier caractère étant à la position 1
LOCATE(chaine1,chaine2, entier 1) :Retourne la position de "chaine1" dans "chaine2" en commencant la recherche au "entier1"-ième caractère. Le premier caractère

Les Fonctions sur les chaines de caractères

REPEAT(chaine,entier):retourne une chaîne de caractères composée de "entier" fois "chaine"

RIGHT(chaine,entier) :Retourne les "entier" derniers caractères de "chaine"

SUBSTRING(chaine,entier1) , SUBSTRING(chaine FROM entier1): Extrait de "chaine" la portion de caractères commençant au "entier1" caractère

SUBSTRING(chaine,entier1,entier2), SUBSTRING(chaine FROM entier1 FOR entier2): Extrait de "chaine" la portion de caractères commençant au "entier1" caractère et d'une longueur de "entier2" caractères

=>La Référence est ici: https://dev.mysql.com/doc/refman/8.0/en/string-functions.html

Les Fonctions numériques

Fonction	Desc	Norme	Access	MySQL	Sql Srv	Oracle
Abs	Valeur absolue	N	O	O	O	O
Ceiling	Valeur approchée haute	N	O	O	O	N
Ceil	Valeur approchée haute	N	N	N	N	O
Floor	Valeur approchée basse	N	O	O	O	O
Cos, sin, tan, exp, log,mod, power, sqrt	Opérations diverses	N	O	O	O	O

*Les fonctions d'arrondi
*Les fonctions de puissances et racines
*Les fonctions valeur absolue et signe
*Reste de division
*Plus grande et plus petite valeur

Application

Projet Sakila-DB →voir Brief Projet : requêtes: 1 jusqu’à 15

Requêtes Imbriquées

Requête imbriquée dans la clause WHERE d'une requête externe:

```
SELECT ...
FROM   ...
WHERE  [Opérande] Opérateur (SELECT ...
                             FROM ...
                             WHERE ...)
```

3 imbrications possibles :

- **$(A_1, \ldots A_n)$ IN <sous-req>** exprime inclusion ensembliste
- **EXISTS <sous-req>** exprime condition d'existence
- **$(A_1, \ldots A_n)$ <comp> [ALL|ANY] <sous-req>** **exprime** comparaison avec quantificateur (ANY par défaut)

Requêtes Imbriquées

Une requête imbriquée peut renvoyer trois types de résultats :

* une valeur scalaire

* une colonne

* une table

Requêtes Imbriquées: *une valeur scalaire*

* une valeur scalaire: s'il comporte une seule ligne et une seule colonne

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

```
4
```

* une sous-requête calculant un résultat scalaire qui se place soit comme une colonne supplémentaire, soit comme une valeur servant à évaluer des conditions (WHERE ou HAVING).

Colonne fictive

On peut ajouter une colonne dans une requête, et choisir comme valeurs pour cette colonne le résultat d'une requête. Ce type de requête est souvent une alternative à GROUP BY

```
SELECT nomprod, (SELECT COUNT(*)
 FROM PROPOSER PR
 WHERE PR.numprod = P.numprod) AS NB_FOURNISSEURS
FROM PRODUIT P
```

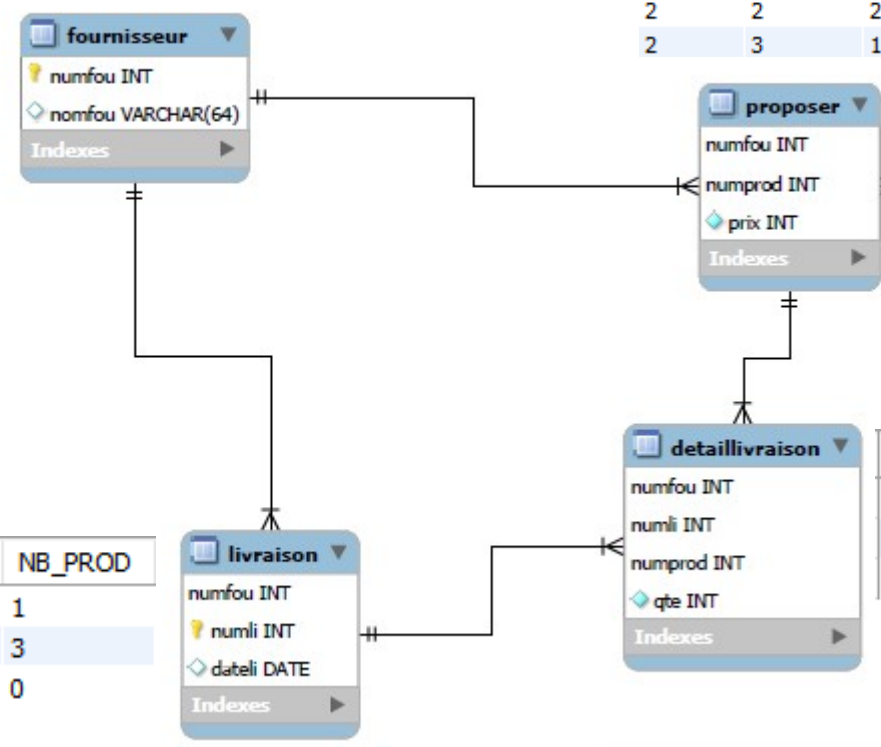
nomprod	NB_FOURNISSEURS
Bocal de cornichons	2
Tube de dentifrice	1
Flacon de lotion anti-escarres	1
Déodorant fraîcheur 96 heures	0

Requêtes Imbriquées: *une valeur scalaire*

numfou	nomfou
1	Bocaux Gérard
2	Paramédical Gisèle
3	Tracteurs Raymond

numfou	numprod	prix
1	1	2
2	1	3
2	2	2
2	3	1

numprod	nomprod
1	Bocal de cornichons
2	Tube de dentifrice
3	Flacon de lotion anti-escarres
4	Déodorant fraîcheur 96 heures



numfou	NB_PROD
1	1
2	3
3	0

numfou	NB_PROD
1	1
2	3
3	0

Requêtes Imbriquées: *une valeur scalaire*

Conditions complexes

On peut construire une condition en utilisant le résultat d'une requête.

```
CREATE VIEW NB_PROD_PAR_FOU AS
 SELECT numfou, (SELECT COUNT(*)
         FROM PROPOSER P
         WHERE P.numfou = F.numfou) AS NB_PROD
 FROM FOURNISSEUR F
```

une vue contenant le nombre d'articles proposés par chaque fournisseur

numfou	NB_PROD
1	1
2	3
3	0

```
SELECT nomfou
FROM FOURNISSEUR F, NB_PROD_PAR_FOU N
WHERE F.numfou = N.numfou
AND NB_PROD = (SELECT MAX(NB_PROD)
 FROM NB_PROD_PAR_FOU)
```

les noms des fournisseurs proposant le plus de produits

nomfou
Paramédical Gisèle

Requêtes Imbriquées: *une valeur scalaire*

INSERT et UPDATE

On peut placer dans des instructions de mises à jour ou d'insertions des requêtes imbriquées..

```
INSERT INTO PERSONNE (numpers, nom, prenom)
VALUES ((SELECT MAX(numpers) + 1 FROM PERSONNE),
'Darth', 'Vador');
```

Requêtes Imbriquées: *une colonne*

* Sous requêtes renvoyant une colonne, On considère une colonne comme une liste de valeurs, on peut tester l'appartance d'un élément à cette liste à l'aide de l'opérateur IN.

On peut s'en servir comme une alternative aux jointures,

```
SELECT N.numfou
FROM NB_PROD_PAR_FOU N
WHERE NB_PROD = (SELECT MAX(NB_PROD)
 FROM NB_PROD_PAR_FOU)
```

numfou	NB_PROD
2	3

numfou	NB_PROD
1	1
2	3
3	0

```
SELECT nomfou
FROM FOURNISSEUR F
WHERE F.numfou IN (SELECT N.numfou
 FROM NB_PROD_PAR_FOU N
 WHERE NB_PROD = (SELECT MAX(NB_PROD)
         FROM NB_PROD_PAR_FOU))
```

nomfou
Paramédical Gisèle

numfou	nomfou
1	Bocaux Gérard
2	Paramédical Gisèle
3	Tracteurs Raymond

Requêtes Imbriquées: *une table*

* Sous requêtes non correlées renvoyant une table;
on peut remplacer le nom d'une table dans la clause FROM par une sous-requête.

```
SELECT
      (SELECT COUNT(*)
          FROM PROPOSER PR
         WHERE PR.numfou = F.numfou
      ) AS NB_PROD
    FROM FOURNISSEUR F;
```

NB_PROD
1
3
0

pour chaque fournisseur, le nombre de produits proposés

Requêtes Imbriquées:

Une sous-requête peut être de deux types :

* simple : Elle est évaluée avant la requête principale
* correlée : Elle est évaluée pour chaque ligne de la requête principale

```
SELECT numfou,
      (SELECT SUM(qte)
       FROM DETAILLIVRAISON D
       WHERE D.numfou = F.numfou
        ) NB_PROD_L
    FROM FOURNISSEUR F;
```

numfou	NB_PROD_L
1	7
2	21
3	NULL

```
SELECT nomfou, NB_PROD_L
    FROM FOURNISSEUR F,
        (SELECT numfou,
          (SELECT SUM(qte)
           FROM DETAILLIVRAISON D
           WHERE D.numfou = F.numfou
          ) NB_PROD_L
        FROM FOURNISSEUR F
         ) L
  WHERE F.numfou = L.numfou;
```

nomfou	NB_PROD_L
Bocaux Gérard	7
Paramédical Gisèle	21
Tracteurs Raymond	NULL

Opérateur IN

```
SELECT ...
FROM ...
WHERE (A1,...,An) [NOT] IN (SELECT B1,...,Bn
                            FROM ...
                            WHERE ...)
```

Sémantique : la condition est vraie si le n-uplet désigné par $(A_1, \ldots, A_n)$ de la requête *externe* appartient (n'appartient pas) au résultat de la requête *interne*.

- La requête interne est effectuée dans un premier temps et son résultat est stocké (temporairement) en RAM (ou sur disque si trop gros!)
- La requête interne n'utilise pas la (les) table(s) de la requête externe
- Le IN étant ensembliste, les nuplets retournés par la requête interne doivent être du même format que le nuplet avant le IN (même nombre d'attributs et de même domaine, par forcément de même nom)
- Le SELECT de la requête externe ne peut retourner que des attributs appartenant aux tables placées dans son FROM

Opérateur Exists

Q

```
SELECT ...
FROM ...
WHERE
```

Q'

```
[NOT] EXISTS (SELECT *
              FROM ...
              WHERE P)
```

- *Sémantique opérationnelle :*
  - pour chaque n-uplet *x* de la requête externe Q, exécuter la requête interne Q'; s'il existe au moins un [s'il n'existe aucun] n-uplet y dans le résultat de la requête interne, alors sélectionner *x*.

- Les deux requêtes sont *corrélées* : la condition *P* dans la requête interne Q' exprime une jointure entre les tables de Q' et les tables de la requête externe Q.
  => requête interne exécutée pour chaque nuplet de la requête externe
- Seule l'existence d'un résultat importe d'où le * dans le SELECT de la requête interne
- Le SELECT de la requête externe ne peut retourner que des attributs appartenant aux tables placées dans son FROM

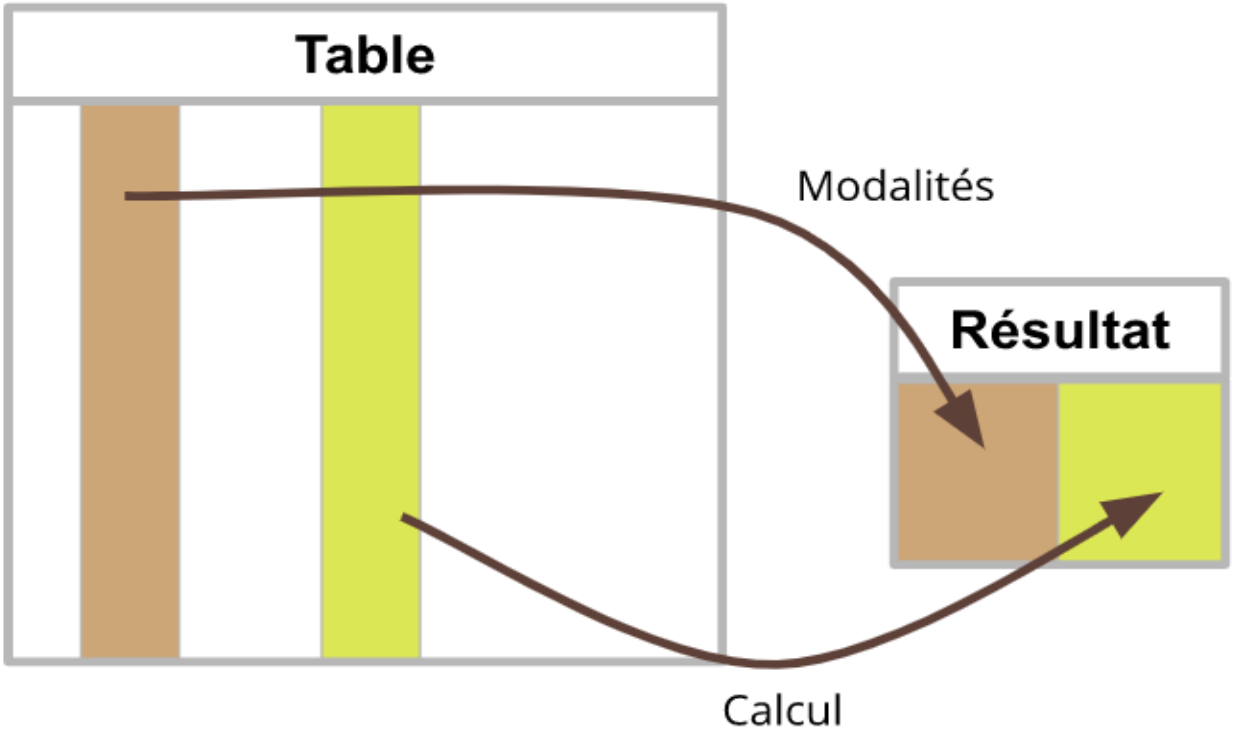
Aggrégats

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



Aggrégats

Jusqu'à présent les requêtes retournent des n-uplets (multi ensemble)

Les fonctions d'agrégation permettent de synthétiser le contenu des données en (statistique descriptive) :

- COUNT(A) ou COUNT(*) : nombre de valeurs ou n-uplets,
- SUM(A) : somme des valeurs,
- MAX(A) : valeur maximale,
- MIN(A) : valeur minimale,
- AVG(A) : moyenne des valeurs

dans *l'ensemble des valeurs* désignées par A

Appliquées aux valeurs NON NULLES seulement!

PN	Pname	Budget	City
01	Dev web	100k	Paris
02	Dev web	200k	Lyo
03	Rech info	120k	Paris
04	Maintenanc	100k	Pau
05	Assurances	80k	Lyo

count(*) count(city)

count
5

count(distinct city)

count
3

Aggrégats

On applique des fonctions de calcul sur le résultat d'une requête :

```
SELECT Agg1(A_i), …, AggN(A_j)
FROM   R_1, ..., R_m
WHERE  Conditions
```

- On exécute la requête (clause from et where).
- On applique les différentes fonctions Aggk sur l'ensemble du résultat.

Project

PN	Pname	Budget	City
01	Dev web	100k	Paris
02	Dev web	200k	Lyo
03	Rech info	120k	Paris
04	Maintenanc	100k	Pau
05	Assurances	80k	Lyo

Select city, count(*) , sum(budget)
From Project
Where city='Paris';

→

City	*count*	*sum*
Paris	2	220k

Aggrégats

Emp (Eno, Ename, Title, City)
Pay(Title, Salary)
Project(Pno, Pname, Budget, City)
Works(Eno, Pno, Resp, Dur)

Nombre d'employés parisiens ?

```
SELECT COUNT(*)
FROM   Emp
WHERE  City='Paris';
```

Plus grand salaire et plus petit salaire de toutes les professions?

```
SELECT MAX(Salary), MIN(Salary)
FROM   Pay;
```

Budgets totaux des budgets des projets de Paris?

```
SELECT SUM(Budget)
FROM   Project
WHERE  City = 'Paris';
```

Nombre de villes où il y a un projet avec l'employé E4?

```
SELECT COUNT(DISTINCT  City)
FROM   Project, Works
WHERE  Project.Pno = Works.Pno
AND    Works.Eno = 'E4';
```

Aggrégation et Imbrication

Emp (Eno, Ename, Title, City)
Pay(Title, Salary)

Project(Pno, Pname, Budget, City)
Works(Eno, Pno, Resp, Dur)

Pour utiliser des attributs de tables du FROM et une agrégation, il faut souvent passer par une sous-requête :

Noms des professions qui payent le plus (et les salaires correspondant) ?

```
SELECT Title, Salary
  FROM Pay
 WHERE Salary = (SELECT MAX(Salary)
                 FROM Pay);
```

Noms des projets dont le budget est supérieur au budget moyen?

```
SELECT Pname
  FROM Project
 WHERE Budget > (SELECT AVG(Budget)
                 FROM Project);
```

← singleton

Remarque : dans le cas où la requête imbriquée retourne le résultat d'un agrégat (donc un seul résultat), pas besoin d'utiliser un IN, ANY ou ALL (mais pas faux) après l'opérateur =, <, >, <=, >=

Aggrégation et Partitions (Groupage)

- Par défaut, agrégation de toutes les valeurs
- Possibilité d'appliquer les fonctions d'agrégation sur des groupes de valeurs → partitionnement

Project

PN	Pname	Budget	City
01	Dev web	100k	Paris
02	Dev web	200k	Lyo
03	Rech info	120k	Paris
04	Maintenanc	100k	Pau
05	Assurances	80k	Lyo

Grouper par **city** →

PN	Pname	Budget	City
01 03	Dev web Rech info	100k 120k	Paris
02 05	Dev web Assurances	200k 80k	Lyo n
04	Maintenanc	100k	Pau

Requêtes de Groupement : Group By

Pour *partitionner* les n-uplets résultats en fonction des valeurs de certains attributs :

```
SELECT A_i, …, A_n, agg1, agg2, …
FROM   R_1, …, R_m
WHERE  P
GROUP BY A_j …, A_k ;
```

Les opérations d'agrégation considérées portent sur la totalité des lignes retournées par les requêtes, l'instruction GROUP BY permet de former des paquets à l'intérieur desquels les données seront agrégées.

Principes :

- On exécute la requête FROM-WHERE
- On regroupe le résultat en paquets d'enregistrements en plaçant dans un paquet tous les enregistrements ayant la même valeur pour A_j …, A_k
- On fait le select en appliquant les agrégations à chaque paquet

Requêtes de Groupement : Group By

Emp (Eno, Ename, Title, City)
Pay(Title, Salary)

Project(Pno, Pname, Budget, City)
Works(Eno, Pno, Resp, Dur)

Numéros des projets avec leurs effectifs ?

```
SELECT Pno, Count(Eno)
FROM   Works
GROUP BY  Pno;
```

Pour chaque ville, nombre d'employés par profession ?

```
SELECT  City, Title, Count(*)
FROM   Emp
GROUP BY  City, Title;
```

Requêtes de Groupement : Group By

afficher la liste des nombres de produits proposés par chaque fournisseur

```
SELECT nomfou, COUNT(*) AS NB_PRODUITS_PROPOSES
FROM FOURNISSEUR F, PROPOSER P
WHERE F.numfou = P.numfou
GROUP BY nomfou;
```

numfou	nomfou
1	Bocaux Gérard
2	Paramédical Gisèle
3	Tracteurs Raymond

numfou	numprod	prix
1	1	2
2	1	3
2	2	2
2	3	1

nomfou	NB_PRODUITS_PROPOSES
Bocaux Gérard	1
Paramédical Gisèle	3

Supposons que nous ne souhaitions garder que les lignes pour lesquelles la valeur NB_PRODUITS_PROPOSES est égale à 1.

=>Ajouter une condition dans WHERE serait inutile, le filtrage occasionné par WHERE est effectué avant l'agrégation.
Il nous faudrait une instruction pour n'inclure que des groupes de données répondant à certains critères.
L'instruction utilisée pour ce faire est HAVING.

```
SELECT nomfou, COUNT(numprod) AS NB_PRODUITS_PROPOSES
FROM FOURNISSEUR F, PROPOSER P
WHERE F.numfou = P.numfou
GROUP BY nomfou
HAVING COUNT(numprod) = 1
ORDER BY nomfou DESC;
```

nomfou	NB_PRODUITS_PROPOSES
Bocaux Gérard	1

Prédicats sur les groupes: Having

Pour *garder (éliminer) les groupes (partitions)* qui satisfont (ne satisfont) pas une certaine condition :

```
SELECT A_i, …, A_n, agg1,agg2,...
FROM   R_1, …, R_m
WHERE  P
GROUP BY A_j …, A_k
HAVING Q ;
```

Principe : une fois la requête FROM-WHERE exécutée, on regroupe en agrégat avec le GROUP BY et on ne garde ensuite que les agrégats satisfaisants le HAVING

Prédicats sur les groupes: Having,OrderBy

Emp (Eno, Ename, Title, City) **Project**(Pno, Pname, Budget, City)
Pay(Title, Salary) **Works**(Eno, Pno, Resp, Dur)

Villes dans lesquelles habitent plus de 2 employés?

```
SELECT City
FROM   Emp
GROUP BY  City
HAVING COUNT(ENO) > 2 ;
```

Projets demandant plus de 1000 jours/homme?

```
SELECT Pno, Pname
FROM   Project, Works
WHERE Projet.Pno=Work.Pno
GROUP BY  Pno, Pname
HAVING SUM(Dur) > 1000 ;
```

```
SELECT ...
FROM ...
WHERE ...
GROUP BY...
HAVING <condition>
ORDER BY ...
```

Emp (Eno, Ename, Title, City) **Project**(Pno, Pname, Budget, City)
Pay(Title, Salary) **Works**(Eno, Pno, Resp, Dur)

Numéros et noms des projets ayant des employés venant de plus de 5 villes différentes ?

```
SELECT Pno, Pname
FROM   Project, Works, Emp
WHERE Projet.Pno=Work.Pno AND Work.Eno=Emp.Eno
GROUP BY  Pno,Pname
HAVING COUNT(DISTINCT Emp.City) > 5 ;
```

Liste des employés triés par temps total de travail décroissants ?

```
SELECT Eno, Ename, SUM(Dur)
FROM   Works, Emp
WHERE  Work.Eno=Emp.Eno
GROUP BY  Eno,Ename
ORDER BY SUM(Dur) DESC ;
```

Prédicats sur les groupes: Having,OrderBy

Emp (Eno, Ename, Title, City) **Project**(Pno, Pname, Budget, City)
Pay(Title, Salary) **Works**(Eno, Pno, Resp, Dur)

Effectif maximal des projets

```
SELECT MAX(COUNT(*))
FROM   Works
GROUP BY  Pno ;
```

Nombre moyen d'employés provenant de chaque ville ?

```
SELECT AVG(COUNT(*))
FROM   Emp
GROUP BY  City ;
```

Comment connaître le nom du (des) projet(s) avec le plus de travailleurs?

```
SELECT Pno,MAX(COUNT(*))
FROM   Works
GROUP BY  Pno ;
```

=> INTERDIT ! Le GROUP BY sert pour le COUNT, pas de GROUP BY pour le MAX, donc pas Pno dans le SELECT

Solution ? Il faut imbriquer !

```
SELECT Pno,COUNT(*)
FROM   Works
GROUP BY  Pno
HAVING COUNT(*)=     (SELECT MAX(COUNT(*))
                      FROM  Works
                      GROUP BY Pno) ;
```

Requêtes de Groupement : HAVING

Afficher les noms des fournisseurs qui ont livré strictement plus d'un produit différent (toutes livraisons confondues)

```
SELECT nomfou
FROM FOURNISSEUR F, DETAILLIVRAISON D
WHERE F.numfou = D.numfou
GROUP BY F.numfou, nomfou
HAVING count(DISTINCT numprod) > 1;
```

numfou	nomfou
1	Bocaux Gérard
2	Paramédical Gisèle
3	Tracteurs Raymond

nomfou
Paramédical Gisèle

Aggrégations et Division

Une autre solution pour faire la division repose sur les agrégats

On procède en 3 étapes :

- On compte le nombre d'enregistrements correspondant à l'ensemble de référence
- On compte pour chaque candidat potentiel à combien d'éléments de l'ensemble de référence il peut être associé
- Si c'est égal au total, on retourne ce candidat

Aggrégations et Division

Emp (Eno, Ename, Title, City) **Project**(Pno, Pname, Budget, City)
Pay(Title, Salary) **Works**(Eno, Pno, Resp, Dur)

Quels sont les employés ayant travaillé sur tous les projets?

```
SELECT Eno,Ename
FROM   Emp
WHERE NOT EXISTS (SELECT *
                  FROM Project
                  WHERE NOT EXISTS (SELECT *
                                    FROM Work
                                    WHERE Work.Eno=Emp.Eno
                                    AND Work.Pno=Project.Pno));
```

Bien observer que la (les) table(s) faisant le lien entre les 2 autres tables se trouve(nt) dans la 2ème imbrication et réalise(nt) les 2 jointures

on calcule les employés pour lesquels il n'existe pas de projets auxquels ils n'ont pas participé
=> Soit une double-négation !

```
SELECT Eno,Ename
   FROM Emp, Work
   WHERE Emp.Eno=Work.Eno
   GROUP BY Eno, Ename
   HAVING COUNT(DISTINCT Pno)=(SELECT COUNT(*)
                               FROM Projet);
```

utilisation indispensable du GROUP BY dans ce type de division

Aggrégations et Division

Emp (Eno, Ename, Title, City) **Project**(Pno, Pname, Budget, City)
Pay(Title, Salary) **Works**(Eno, Pno, Resp, Dur)

Quels sont les villes dans lesquelles tous les salaires d'employés sont représentés?

```
SELECT DISTINCT City
FROM   Emp A
WHERE NOT EXISTS (SELECT *
                  FROM Pay P1
                  WHERE NOT EXISTS (SELECT *
                                    FROM Emp B, Pay P2
                                    WHERE B.Title=P2.Title
                                    AND P2.Salary=P1.Salary
                                    AND B.City=A.City));
```

```
SELECT City
FROM Emp, Pay
WHERE Emp.Title=Pay.Title
GROUP BY City
HAVING COUNT(DISTINCT Salary) = (SELECT COUNT(DISTINCT Salary)
                                 FROM Pay);
```

Aggrégations et Division

Emp (Eno, Ename, Title, City) **Project**(Pno, Pname, Budget, City)
Pay(Title, Salary) **Works**(Eno, Pno, Resp, Dur)

Quels sont les projets parisiens pour lesquels toutes les professions gagnant plus de 3000 euros ont participé?

```
SELECT Pno,Pname
FROM   Projet
WHERE City='Paris'
AND NOT EXISTS (SELECT *
                FROM Pay
                WHERE Salary>3000
                AND NOT EXISTS (SELECT *
                                FROM Work, Emp
                                WHERE Work.Eno=Emp.Eno
                                AND Work.Pno=Project.Pno
                                AND Emp.Title=Pay.Title)
```

```
SELECT Pno,Pname
FROM Projet, Work, Emp, Pay
WHERE Projet.City='Paris'
AND Projet.Pno=Work.Pno
AND Work.Eno=Emp.Eno
AND Emp.Title=Pay.Title
AND Salary>3000
GROUP BY Pno, Pname
HAVING COUNT(DISTINCT Title)= (SELECT COUNT(Title)
                               FROM Pay
                               WHERE Salary>3000);
```

Application

Projet Sakila-DB →voir Brief Projet : requêtes: 16 jusqu’à 21

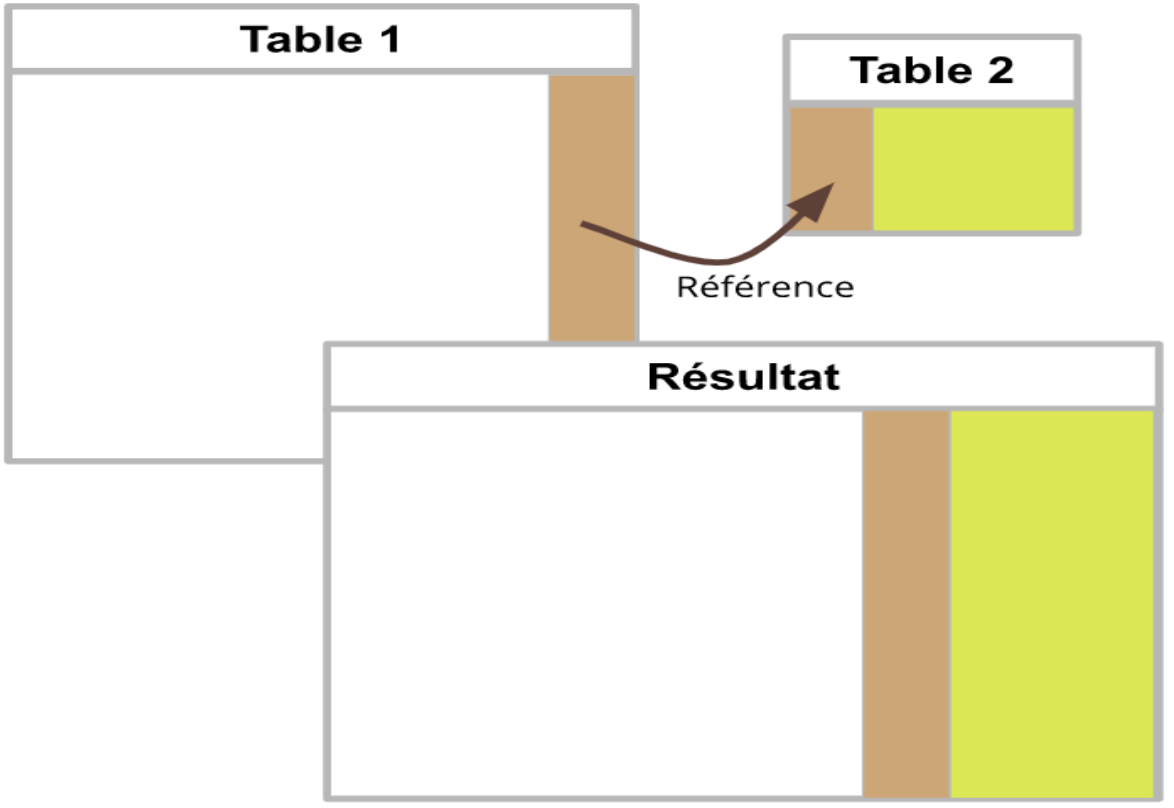
Jointures

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



Jointures

EMP

ENO	ENAME	TITLE
E1	J. Doe	Elect. Eng
E2	**M. Smith**	**Analyst**
E3	A. Lee	Mech. Eng.
E4	J. Miller	Programmer
E5	**B. Casey**	**Analyst**
E6	L. Chu	Elect. Eng.
E7	R. Davis	Mech. Eng.
E8	**J. Jones**	**Analyst**

PAY

TITLE	SALARY
Elect. Eng.	55000
Analyst	70000
Mech. Eng.	45000
Programmer	60000

```
SELECT ENO, ENAME, EMP.TITLE, SALARY
FROM EMP, PAY
WHERE EMP.TITLE=PAY.TITLE
        And PAY.TITLE='Analyst';
```

- Un tuple de la table *T1* apparait dans le résultat de le jointure s'il joint **avec au moins 1** tuple de *T2*
- Très souvent, les jointures permettent de « traverser » les associations du schéma E/A
- Si un attribut de même nom dans les 2 tables alors nécessité de préfixer l'attribut par le nom de la table (ou par son renommage si la table est renommée)
- Si oubli de la condition de jointure... produit cartésien ! Donc si k tables dans le from, en général au minimum k-1 conditions de jointure dans le WHERE

Jointures

Emp (Eno, Ename, Title, City) **Project**(Pno, Pname, Budget, City)
Pay(Title, Salary) **Works**(Eno, Pno, Resp, Dur)

Noms, salaires et titres des employés ?

```
SELECT Eno, Ename, Emp.Title, Salary
FROM   Emp, Pay
WHERE  Emp.Title = Pay.Title ;
```

Que se passe-t'il pour les employés exerçant un emploi dont on ne connait pas le salaire (qui n'ont pas d'entrée dans la table Pay) ?

Jointures internes

EMP

ENO	ENAME	TITLE
E1	J. Doe	Elect. Eng
E2	M. Smith	Analyst
E3	A. Lee	Mech. Eng.
E4	J. Miller	Programmer
E5	B. Casey	Analyst
E6	L. Chu	Elect. Eng.
E7	R. Davis	Mech. Eng.
E8	J. Jones	Analyst

PAY

TITLE	SALARY
Elect. Eng.	55000
Mech. Eng.	45000
Programmer	60000

```
SELECT ENO, ENAME, EMP.TITLE, SALARY
FROM EMP, PAY
WHERE EMP.TITLE=PAY.TITLE;
```

ENO	ENAME	EMP.TITLE	SALARY
E1	J. Doe	Elect. Eng.	55000
E3	A. Lee	Mech. Eng.	45000
E4	J. Miller	Programmer	60000
E6	L. Chu	Elect. Eng.	55000
E7	R. Davis	Mech. Eng.	45000

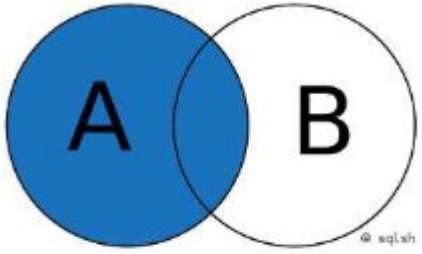
Les employés E2, E5 et E8 ne font pas parti du résultat Puisque leur Title n'apparaît pas dans la table Pay

Jointures externes gauches

```
SELECT liste-colonnes
FROM T1, T2
WHERE Condition;
```

Le résultat = tous les nuplet de T1

- associés aux nuplets de T2 avec lesquels ils sont joignables
- associés à NULL sinon



Jointure externe gauche

EMP

ENO	ENAME	TITLE
E1	J. Doe	Elect. Eng
E2	M. Smith	Analyst
E3	A. Lee	Mech. Eng.
E4	J. Miller	Programmer
E5	B. Casey	Analyst
E6	L. Chu	Elect. Eng.
E7	R. Davis	Mech. Eng.
E8	J. Jones	Analyst

PAY

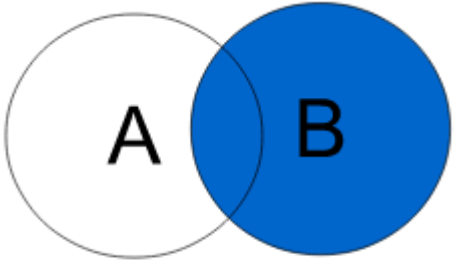
TITLE	SALARY
Elect. Eng.	55000
Mech. Eng.	45000
Programmer	60000

```
SELECT ENO, ENAME, EMP.TITLE, SALARY
FROM EMP, PAY
WHERE EMP.TITLE=PAY.TITLE(+);
```

ENO	ENAME	EMP.TITLE	SALARY
E1	J. Doe	Elect. Eng.	55000
E2	M. Smith	Analyst	NULL
E3	A. Lee	Mech. Eng.	45000
E4	J. Miller	Programmer	60000
E5	B. Casey	Analyst	NULL
E6	L. Chu	Elect. Eng.	55000
E7	R. Davis	Mech. Eng.	45000
E8	J. Jones	Analyst	NULL

Jointures externes droites

SELECT liste-colonnes
FROM table1 **RIGHT** OUTER JOIN table2 ON table1.att=table2.att
WHERE conditions;



Jointure externe droite
(+) du côté gauche

EMP

ENO	ENAME	TITLE
E1	J. Doe	Elect. Eng
E2	M. Smith	Analyst
E3	A. Lee	Mech. Eng.
E4	J. Miller	Programmer
E5	B. Casey	Analyst
E6	L. Chu	Elect. Eng.
E7	R. Davis	Mech. Eng.
E8	J. Jones	Analyst

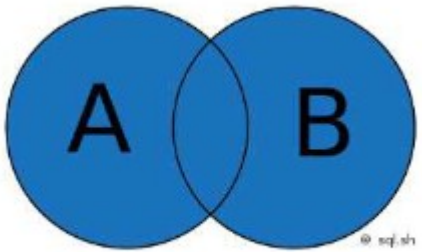
PAY

TITLE	SALARY
Elect. Eng.	55000
Mech. Eng.	45000
Programmer	60000
Doctor	55000

```
SELECT *
FROM EMP, PAY
WHERE EMP.TITLE(+)=PAY.TITLE;
```

ENO	ENAME	EMP.TITLE	SALARY	PAY.TITLE
E1	J. Doe	Elect. Eng.	55000	Elect. Eng.
E3	A. Lee	Mech. Eng.	45000	Mech. Eng.
E4	J. Miller	Programmer	60000	Programmer
E6	L. Chu	Elect. Eng.	55000	Elect. Eng.
E7	R. Davis	Mech. Eng.	45000	Mech. Eng.
NULL	NULL	NULL	55000	Doctor

Jointures externes complètes



Jointure externe complète

EMP

ENO	ENAME	TITLE
E1	J. Doe	Elect. Eng
E2	M. Smith	Analyst
E3	A. Lee	Mech. Eng.
E4	J. Miller	Programmer
E5	B. Casey	Analyst
E6	L. Chu	Elect. Eng.
E7	R. Davis	Mech. Eng.
E8	J. Jones	Analyst

PAY

TITLE	SALARY
Elect. Eng.	55000
Mech. Eng.	45000
Programmer	60000
Doctor	55000

```
SELECT *
FROM EMP, PAY
WHERE EMP.TITLE(+)=PAY.TITLE(+);
```

ENO	ENAME	EMP.TITLE	SALARY	PAY.TITLE
E1	J. Doe	Elect. Eng.	55000	Elect. Eng.
E2	M. Smith	Analyst	NULL	NULL
E3	A. Lee	Mech. Eng.	45000	Mech. Eng.
E4	J. Miller	Programmer	60000	Programmer
E5	B. Casey	Analyst	NULL	NULL
E6	L. Chu	Elect. Eng.	55000	Elect. Eng.
E7	R. Davis	Mech. Eng.	45000	Mech. Eng.
E8	J. Jones	Analyst	NULL	NULL
NULL	NULL	NULL	55000	Doctor

(left outer join) union (right outer join)

```
SELECT liste-colonnes
FROM table1 FULL OUTER JOIN table2 ON table1.att=table2.att
WHERE conditions;
```

Jointures externes complètes

Emp (Eno, Ename, Title, City) **Project**(Pno, Pname, Budget, City)
Pay(Title, Salary) **Works**(Eno, Pno, Resp, Dur)

Noms des employés Nantais et leur salaire quand il est connu (null s'il n'est pas connu) en gardant les informations sur les salaires des emplois non affectés?

```
SELECT Ename, Salary
FROM Emp FULL OUTER JOIN Pay ON Emp.Title = Pay.Title
WHERE City='Nantes' ;
```

Jointures Internes

```
SELECT liste-colonnes
FROM T1 INNER JOIN T2 ON T1.att=T2.att
WHERE conditions;
```

Où condition ne contient pas de condition liant 2 tables.

Equivalent à

```
SELECT liste-colonnes
FROM T1, T2
WHERE conditions AND T1.att=T2.att;
```

Emp (Eno, Ename, Title, City) **Project**(Pno, Pname, Budget, City)
Pay(Title, Salary) **Works**(Eno, Pno, Resp, Dur)

Noms et salaires des employés dont on connait le salaire?

```
SELECT Ename, Salary
FROM Emp INNER JOIN Pay ON Emp.Title = Pay.Title ;
```

Jointures Internes

Emp (Eno, Ename, Title, City) **Project**(Pno, Pname, Budget, City)
Pay(Title, Salary) **Works**(Eno, Pno, Resp, Dur)

Noms et salaires des employés dont on connait le salaire?

```
SELECT Ename, Salary
FROM Emp INNER JOIN Pay ON Emp.Title = Pay.Title ;
```

Noms et salaires des employés dont on connait le salaire?

```
SELECT Ename, Salary
FROM Emp INNER JOIN Pay ON Emp.Title = Pay.Title ;
```

Ou encore avec la jointure naturelle :

```
SELECT Ename, Salary
FROM Emp NATURAL JOIN Pay ;
```

Jointures Internes

Emp (Eno, Ename, Title, City) **Project**(Pno, Pname, Budget, City)
Pay(Title, Salary) **Works**(Eno, Pno, Resp, Dur)

Noms des employés et leur salaire quand il est connu?

```
SELECT Ename, Salary
FROM Emp LEFT OUTER JOIN Pay ON Emp.Title = Pay.Title ;
```

Application

Projet Sakila-DB →voir Brief Projet : requêtes: 24 jusqu'à 37