Plan

- Structure d'une BD relationnelle
- Algèbre relationnelle
- Calcul relationnel

Structure d'une BD relationnelle

Les données sont structurées en **tables** (**relations**)

Etant donnés les ensembles $A_1, \ldots A_n$, une `relation` r est un sous ensemble de $A_1 \times A_2 \times \cdots \times A_n$.

Une relation est un `ensemble` de `n-uplets` (ou tuples) de la forme $\langle a_1, \ldots a_n \rangle$ avec $a_i \in A_i$.

Exemple : On a trois ensembles : Nom, Num_Cte et Rue avec
Nom = {Durand, Dupont, Dupond}
Num_Cte = {123, 124, 235 , 226}
Rue = {Neuve, vieille, Courte }

Alors

$\{$ $\langle$ Dupont, 123, Neuve $\rangle$,
$\langle$ Dupont, 124, Neuve $\rangle$,
$\langle$ Dupond, 235, Neuve $\rangle$,
$\langle$ Durand, 123, Vieille $\rangle$ $\}$

est une relation sur Nom $\times$ Num_Cte $\times$ Rue

Schéma de relation

Une table est une relation (au sens mathématique) qui a un nom

$A_1, \ldots, A_n$ sont des `attributs`

$R(A_1, \ldots, A_n)$ est un schéma de relation.
R est le nom du schéma de la relation.

On note $Att(R)$ pour désigner l'ensemble des attributs de R

L'`arité` de R est la cardinalité de $Att(R)$

Le domaine de A_i (noté $dom(A_i)$) est l'ensemble des valeurs associées à A_i. Cet ensemble peut être fini ou non

Instance de relation

Emp	Nom	Num_Cte	Rue
	Dupont	124	Neuve
	Dupond	235	Neuve
	Durand	123	Vieille

$Att(Emp) = \{Nom, Num\_Cte, Rue\}$

Arité(Emp) = 3

$Dom(Num\_Cte)$ = les entiers naturels (infini)

$Dom(Nom)$ = chaînes de moins de 20 caractères (fini)

Langages de requête

Langages qui permettent d'interroger la BD
(i) Langages relationnels "purs"

- Algèbre relationnelle
- Calcul relationnel par n-uplet
- Calcul relationnel par domaine

(ii) Langages pratiques

- SQL (Structured Query Language)
- QUEL (Query Language)
- SEQUEL (Structured English as a Query Language)
- QBE (Query By Example)

Algèbre relationnelle

Six opérations de base

1. Projection
2. Sélection
3. Union
4. Différence
5. Produit cartésien
6. Renommage

Certaines sont unaires d'autres sont binaires

Projection

Notation : $\pi_{A_1,\ldots,A_k}(r)$ où
r : nom de relation et
$\forall 1 \leq i \leq k \ A_i \in Att(r)$.

Le résultat de cette opération est une relation avec k colonnes

Projection (Exemple)

On veut extraire les noms des employés de la relation ¡¡Emp¿¿ ci-dessous

Emp	Nom	Num_Cte	Rue
	Dupont	124	Neuve
	Dupont	235	Neuve
	Durand	123	Vieille

$\pi_{Nom}(Emp) =$

	Nom
	Dupont
	Durand

Sélection

Notation : $\sigma_{Cond}(r)$ où

- r est le nom d'une relation

- $Cond$ est une condition de la forme

  1. $Att_i\,\theta\,Att_j$ ou
     $Att_i\,\theta$ constante avec
     $\theta \in \{<, \leq, =, \geq, >, \neq\}$, ou bien

  2. une `conjonction` ($\wedge$) ou une `disjonction` ($\vee$) de conditions

Le résultat est une relation qui contient tous les n-uplets de r qui satisfont la condition $Cond$

Sélection (Exemple)

On veut avoir les informations concernant les employés dont le nom est Dupont

Emp	Nom	Num_Cte	Rue
	Dupont	124	Neuve
	Dupont	235	Neuve
	Durand	123	Vieille

$\sigma_{(Nom=Dupont)}$ =

Nom	Num_Cte	Rue
Dupont	124	Neuve
Dupont	235	Neuve

Union, Différence et Intersection

- Opérations ensemblistes classiques
- Notation : $r \cup s$; $r - s$; $r \cap s$
- $r \cup s = \{t \mid t \in r \textbf{ ou } t \in s\}$
- $r - s = \{t \mid t \in r \textbf{ et } t \notin s\}$
- $r \cap s = \{t \mid t \in r \textbf{ et } t \in s\}$
- Opérations binaires
- Il faut que $Att(r) = Att(s)$

Union, Différence et Intersection

r	A	B
	α	1
	α	2
	β	1

s	A	B
	α	2
	β	3

$r - s =$

A	B
α	1
β	1

$r \cup s =$

A	B
α	1
α	2
β	1
β	3

$r \cap s =$

A	B
α	2

Produit cartésien

Notation : $r \times s$

$r \times s = \{tv \mid t \in r \textbf{ et } \quad v \in s\}$

tv est la concaténation des tuples t et v

Cette opération n'est pas définie si $Att(r) \cap Att(s) \neq \emptyset$

$Att(r \times s) = Att(r) \cup Att(s)$

Produit cartésien (Exemple)

r	A	B
	α	1
	β	2

s	C	D	E
	α	10	+
	β	10	−

$r \times s =$

A	B	C	D	E
α	1	α	10	+
α	1	β	10	−
β	2	α	10	+
β	2	β	10	−

Renommage

Notation : $\rho_{Att_i \to Att'_i}(r)$

Permet de renommer l'attribut Att_i par Att'_i

Le résultat est la relation r avec un nouveau schéma

Renommage (Exemple)

r	A
	10
	20

$\rho_{\mathsf{A} \to \mathsf{B}}(r) =$

r	B
	10
	20

Composition des opérateurs

On peut appliquer un opérateur de l'algèbre au résultat d'une autre opération

Exemple : $\pi_A\Big(\sigma_{B=20}(r)\Big)$

On dit que l'algèbre relationnelle est un langage `fermé` car chaque opération prend une ou deux relations et retourne une relation.

Soient les schémas de relation $Tit(Id, Nom, Adresse)$ et $Cte(Num, Solde, Id\_Tit)$.

Le compte de numéro Num appartient au client identifié par Id_Tit.

On veut avoir (i) le numéro, (ii) le solde et (iii) le nom du titulaire de chaque compte débiteur.

Id	Nom	Adresse
A25	Dupond	rue neuve
B212	Durand	rue vieille

Tit

Num	Solde	Id_Tit
120	25234.24	A25
135	-100	A25
275	230	B212

Cte

1. $Cte \times Tit$ retourne une relation qui associe à chaque tuple de Cte, tous les tuples de Tit

2. $\sigma_{Id=Id\_Tit}(Cte \times Tit)$ élimine les tuples où le compte n'est pas associé au bon titulaire

3. $\sigma_{Solde<0}\Big(\sigma_{Id=Id\_Tit}(Cte \times Tit)\Big)$ retient les comptes débiteurs

4. $\pi_{Nom,Num,Solde}\Big(\sigma_{Solde<0}\Big(\sigma_{Id=Id\_Tit}(Cte \times Tit)\Big)\Big)$ élimine les attributs non demandés

Comment aurait-on pu faire si dans Cte on avait Id au lieu de Id_Tit comme nom d'attribut ?

Jointure

Notation : $r \bowtie s$

$Att(r \bowtie s) = Att(r) \cup Att(s)$

Résultat : Soient $t_r \in r$ et $t_s \in s$. $t_r t_s \in r \bowtie s$ ssi $\forall A \in Att(r) \cap Att(s) t_r.A = t_s.A$

Jointure (Exemple)

r	A	B
	α	10
	α	15
	β	1

s	B	C
	10	$+$
	1	$-$

$r \bowtie s =$

A	B	C
α	10	$+$
β	1	$-$

Jointure (Exemple)

- Noter que le même résultat peut être obtenu comme suit :

  1. $temp_1 := \rho_{B \to B_1}(s)$
  2. $temp_2 := r \times temp_1$
  3. $temp_3 := \sigma_{B=B_1}(temp_2)$
  4. $res := \pi_{A,B,C}(temp_3)$

- La jointure n'est pas une opération de base de l'algèbre relationnelle

Calcul relationnel par n-uplet

- Les requêtes sont de la forme $\{t \mid P(t)\}$
- C'est l'ensemble des n-uplets tels que le `prédicat` $P(t)$ est vrai pour t
- t est une variable n-uplet et $t[A]$ désigne la valeur de l'attribut A dans t
- $t \in r$ signifie que t est un n-uplet de r
- P est une formule de la logique de premier ordre

Rappel sur le calcul des prédicats

- Des ensembles d'attributs, de constantes, de comparateurs $\{<, \dots\}$

- Les connecteurs logiques 'et' $\wedge$, 'ou' $\vee$ et la négation $\neg$

- Les quantificateurs $\exists$ et $\forall$:
  - $\exists t \in r\Big(Q(t)\Big)$: Il existe un tuple t de r tel que Q est vrai
  - $\forall t \in r\Big(Q(t)\Big)$: Q est vrai pour tout t de r

Exemples de requêtes

Considérons les schémas de relations suivants :
Film(Titre, Réalisateur, Acteur) instance f
Programme(NomCiné, Titre, Horaire) instance p
f contient des infos sur tous les films et p concerne le programme à Bordeaux

- Les films réalisés par Bergman

$$\{t \mid t \in f \wedge t[\text{Réalisateur}] = \text{''Bergman''}$$

- Les films où Jugnot et Lhermite jouent ensembles

$$\{t \mid t \in f \wedge \exists s \in f \Big(t[\text{Titre}] = s[\text{Titre}] \wedge$$
$$t[\text{Acteur}] = \text{''Jugnot''} \wedge s[\text{Acteur}] = \text{''Lhermite''} \Big)$$

Exemples de requêtes(suite)

Les titres des films programmés à Bordeaux :

$$\{t \mid \exists s \in p\Big(t[\text{Titre}] = s[\text{Titre}]\Big)\}$$

Les films programmés à l'UGC mais pas au Trianon :

$$\{t \mid \exists s \in p\Big(s[\text{Titre}] = t[\text{Titre}] \wedge s[\text{NomCiné}] = \text{“UGC”} \wedge \neg\exists u \in p\Big(u[\text{NomCiné}] = \text{“Trianon”} \wedge u[\text{Titre}] = t[\text{Titre}]\Big)\Big)\}$$

Les titres de films qui passent à l'UGC ainsi que leurs réalisateurs :

$$\{t \mid \exists s \in p\Big(\exists u \in f\Big(s[\text{NomCiné}] = \text{“UGC”} \wedge s[\text{Titre}] = u[\text{Titre}] = t[\text{Titre}] \wedge t[\text{Réal}] = u[\text{Réal}]\Big)\Big)\}$$

Expressions “non saines”

Il est possible d'écrire des requêtes en calcul qui retournent une relation infinie.

Exemple : Soit NumCte(Num) avec l'instance n et la requête $\{t \mid \neg t \in n\}$ i.e les numéros de compte non recensés. Si on considère que le $Dom(Num) = N$, alors la réponse à cette requête est infinie.

Une requête est *saine si quelle que soit l'instance de la base dans laquelle on l'évalue, elle retourne une réponse finie.*

Dépendance du domaine.

Calcul relationnel par domaine

Les requêtes sont de la forme :

$$\{\langle x_1, \ldots, x_n \rangle \mid P(x_1, \ldots, x_n)\}$$

Les x_i représentent des variables de domaine.

$P(x_1, \ldots, x_n)$ est une formule similaire à celles qu'on trouve dans la logique des prédicats.

Exemple : Les titres de films programmés à l'UGC de Bordeaux

$$\{\langle t \rangle \mid \exists \langle nc, t, h \rangle \in p(nc = \text{“}UGC\text{”})\}$$

Relation entre les 3 langages

- Toute requête exprimée en algèbre peut être exprimée par le calcul.

- Toute requête "saine" du calcul peut être exprimée par une requête de l'algèbre.

- Les 3 langages sont donc équivalents d'un point de vue puissance d'expression.

- L'algèbre est un langage *procédurale* (quoi et comment) alors que le calcul ne l'est pas (seulement quoi)

Le langage SQL

C'est un langage fourni avec tout SGBD relationnel commercialisé.

C'est un standard reconnu par l'ISO depuis 87 (standard⇒ portabilité)

On en est à la version 2 (SQL92) et la version 3 est annoncée pour bientôt

SQL est un LDD et un LMD. Il est aussi utilisé pour définir des vues, les droits d'accès, manipulation de schéma physique ...

Structure de base

Une requête SQL typique est de la forme :

$$\begin{array}{ll} \textbf{SELECT} & A_1, \ldots, A_n \\ \textbf{FROM} & r_1, \ldots, r_m \\ \textbf{WHERE} & P \end{array}$$

Les A_i sont des attributs, les r_j sont des noms de relations et P est un prédicat.

Cette requête est équivalente à

$$\pi_{A_1,\ldots,A_n}\Big(\sigma_P(r_1 \times \cdots \times r_m)\Big)$$

La clause SELECT

La clause SELECT correspond à la projection de l'algèbre.

Les titres des films :

```
SELECT  Titre
FROM    film
```

L'utilisation de l'astérisque permet de sélectionner tous les attributs

```
SELECT  *
FROM    film
```

La clause SELECT (suite)

SQL autorise par défaut les doublons. Pour le forcer à les éliminer, on utilise la clause **DISTINCT**

```
SELECT DISTINCT  Titre
FROM             film
```

La clause **SELECT** peut contenir des expressions arithmétiques ainsi que le renommage d'attributs

```
SELECT  Prix_HT * 1.206 AS Prix_TTC
FROM    produit
```

La clause WHERE

Correspond au prédicat de sélection dans l'algèbre.

La condition porte sur des attributs des relations qui apparaissent dans la clause **FROM**

```
SELECT DISTINCT  Titre
FROM             film
WHERE            Réalisateur = "Bergman"
                 AND Acteur = "Stewart"
```

SQL utilise les connecteurs **AND**, **OR** et **NOT**

Pour simplifier la clause **WHERE**, on peut utiliser la clause **BETWEEN**

```
SELECT  Num
FROM    compte
WHERE   Solde BETWEEN 0 AND 10000
```

La clause FROM

Elle correspond au produit cartésien de l'algèbre.

Le titre et le réalisateur des films programmés à l'UGC de Bordeaux.

```
SELECT  Titre, Réalisateur
FROM    film, programme
WHERE   film.titre=programme.titre AND
        programme.NomCiné= "UGC"
```

Les variables n-uplets

Elles sont définies dans la clause **FROM**

```
SELECT  Titre, Réalisateur
FROM    film AS f, programme AS p
WHERE   f.titre=p.titre AND p.NomCiné="UGC"
```

Soit Emp(Id,Nom,Id_chef)

```
SELECT  e1.Nom, e2.Nom AS Nom_Chef
FROM    emp e1, emp e2
WHERE   e1.Id_chef = e2.Id
```

La clause ORDER BY

SQL permet de trier les résultats de requête

```
SELECT      *
FROM        programme
WHERE       NomCiné="UGC"
ORDER BY    Horaire ASC, Titre DESC
```

Opérateurs ensemblistes

SELECT ...
...
UNION/ INTERSECT/ EXCEPT
SELECT ...

Attention : Ces opérations éliminent les doublons, pour pouvoir les garder, utiliser à la place **INTERSECT ALL** ...

Si t apparaît m fois dans r et n fois dans s alors il apparaît

- $m+n$ fois dans r **UNION ALL** s
- $\min(m,n)$ fois dans r **INTERSECT ALL** s
- $\max(0,m-n)$ fois dans r **EXCEPT ALL** s

Les fonctions d'aggrégats

Ce sont des fonctions qui agissent sur des ensembles (multi-ensembles) de valeurs :

AVG : la valeur moyenne de l'ensemble
MIN : la valeur minimale
MAX : la valeur maximale
SUM : le total des valeurs de l'ensemble
COUNT : le nombre de valeur dans l'ensemble

Les fonctions d'aggrégats (suite)

SELECT COUNT(Titre)
FROM Programme

Cette requête retourne le nombre de films programmés à Bordeaux.

Attention : Un même titre peut être compté plusieurs fois s'il est programmé à des heures différentes et dans des salles différentes.

SELECT COUNT(**DISTINCT** Titre)
FROM Programme

Aggrégats et GROUP BY

Le nombre de films programmés dans chaque salle

```
SELECT NomCiné, COUNT(DISTINCT Titre)
FROM Programme
GROUP BY NomCiné
```

Les attributs apparaissant dans la clause **SELECT** en dehors des aggrégats *doivent* être associés à la clause **GROUP BY**

Aggrégats et la clause HAVING

Les salles où sont programmés plus de 3 films

SELECT NomCiné, **COUNT**(**DISTINCT** Titre)
FROM Programme
GROUP BY NomCiné
HAVING COUNT(**DISTINCT** Titre) $>$ 3

Le prédicat associé à la clause **HAVING** est testé après la formation des groupes définis dans la clause **GROUP BY**

Requêtes imbriquées

SQL fournit un mécanisme qui permet d'imbriquer les requêtes.

Une sous requête est une requête SQL (SELECT-FROM-WHERE) qui est incluse dans une autre requête. Elle apparaît au niveau de la clause **WHERE** de la première requête.

Les films programmés à l'UGC non programmés au Triaon

```
SELECT Titre
FROM Programme
WHERE NomCiné="UGC" and Titre NOT IN (
      SELECT Titre
      FROM Programme
      WHERE NomCiné="Trianon")
```

Requêtes imbriquées (suite)

Compte(Num, Solde, NomTit)

Trouver les comptes dont les soldes sont supérieurs aux soldes des comptes de Durand

```
SELECT *
FROM Compte
WHERE Solde > ALL (
      SELECT Solde
      FROM Compte
      WHERE NomTit="Durand" )
```

En remplaçant **ALL** par **SOME**, on obtient les comptes dont les soldes son sup. au solde d'au moins un compte de Durand.

Requêtes imbriquées (suite)

Les cinémas qui passent tous les films programmés à l'UGC

```
SELECT NomCiné
FROM programme p1
WHERE NOT EXISTS (
      (SELECT DISTINCT Titre
      FROM programme
      WHERE NomCiné="UGC")
      EXCEPT
      (SELECT DISTINCT Titre
      FROM programme p2
      WHERE p1.NomCiné=p2.NomCiné))
```

Test d'absence de doublons

La clause **UNIQUE** permet de tester si une sous requête contient des doublons.

Les titres de films programmés dans une seule salle et un seul horaire

```
SELECT p.Titre
FROM programme p
WHERE UNIQUE (
      (SELECT p1.Titre
      FROM programme p1
      WHERE p.Titre=p1.Titre)
```

Les relations dérivées

Titulaire(Nom, Adresse)

Compte(Num, Solde, NomTit)

Donner le solde moyen des comptes de chaque personne ayant un solde moyen supérieur à 1000

```
SELECT NomTit, SoldeMoyen
FROM (
        SELECT NomTit, AVG(Solde)
        FROM Compte
        GROUP BY NomTit )
    AS Result(NomTit, SoldeMoyen)
WHERE SoldeMoyen > 1000
```

Noter qu'on aurait pu exprimer cette requête en utilisant la clause **HAVING**

Les vues

Permettent de définir des relations *virtuelles* dans le but de (i) cacher certaines informations à des utilisateurs, (ii) faciliter l'expression de certaines requêtes (iii) améliorer la présentation de certaines données.

Une vue est définie par une expression de la forme :

CREATE VIEW V **AS** `requête`

`requête` : est une expression quelconque de requête

V : est le nom de la vue

Les vues (suite)

Emp(NumE, Salaire, Dept, Adresse)

CREATE VIEW EmpGen **AS** (
  SELECT NumE, Dept, Adresse
  FROM Emp)

Toutes les informations concernant les employés du département 5.

```
SELECT *
FROM EmpGen
WHERE Dept = 5
```

Modification des relations : Suppression

- Supprimer tous les employés du département 5
  DELETE FROM Emp
  WHERE Dept=5

- Supprimer du programme tous les films programmés à l'UGC où un des acteurs est DiCaprio.

  DELETE FROM programme
  WHERE NomCiné = "UGC" **AND EXISTS** (
  SELECT Titre
  FROM film
  WHERE programme.Tite = film.Titre **AND**
  film.Acteur ="DiCaprio")

Modification des relations : Suppression

Supprimer les comptes dont le solde est inférieur à la moyenne des soldes de tous les comptes.

DELETE FROM compte
WHERE Solde $<$ (**SELECT AVG**(Solde) **FROM** compte)

Remarque : Si les n-uplets sont supprimés un à un de la relation compte, on peut penser qu'à chaque suppression on a une nouvelle valeur de **AVG**(Solde).

Solution de SQL : D'abord, calculer **AVG**(Solde) et ensuite supprimer les tuples satisfaisant le test sans re-calculer à chaque fois la nouvelle valeur de **AVG**(Solde).

Modification des relations : Insertion

- Insérer un n-uplet dans la relation "compte"

  INSERT INTO compte(Num, Solde, NomTit) **VALUES**
  (511, 1000, "Dupont")

  ou bien

  INSERT INTO compte **VALUES**
  (511, 1000, "Dupont")

- Insère un n-uplet avec un solde *inconnu*.

  INSERT INTO compte **VALUES**
  (511, `NULL`, "Dupont")

  ou bien

  INSERT INTO compte(Num, NomTit) **VALUES**
  (511, "Dupont")

  Les 2 dernières m.à.j sont équivalentes sauf si une valeur par défaut du Solde a été spécifiée lors de la définition de la table `compte`.

Modification des relations : Insertion

Supposons qu'on a crée une relation TitMoy(NomTit, Moyenne) qui doit contenir le nom des clients de la banque ainsi que la moyenne des soldes de leurs comptes.

INSERT INTO TitMoy(NomTit, Moyenne)
SELECT NomTit, **AVG**(Solde)
FROM compte
GROUP BY NomTit

Modification des relations : Update

Rajouter 1% à tous les comptes dont le solde est inférieur à 500

UPDATE compte
SET Solde = Solde * 1.01
WHERE Solde $\leq$ 500

La condition qui suit la clause **WHERE** peut être une requête SQL.

SQL en tant que LDD

- Le schéma des relations
- Les domaines des attributs
- Les contraintes d'intégrité
- La gestion des autorisations
- La gestion du stockage physique
- Les index associés à chaque relation

Domaines

- **char(n)** : chaîne de caractères de taille fixe **n**
- **varchar(n)** : chaîne de caractères de taille variable mais inférieure à **n**
- **int** : Entier (un sous ensemble fini des entiers, dépend de la machine)
- **smallint** : Entier. Sous ensemble de **int**
- **numeric(p,d)** : Un réel codé sur p digits et au max **d** digits pour la partie à droite de la décimale.

Domaines

- **real** : Un réel flottant.
- **date** : YYYY-MM-DD (année, mois, jours)
- **time** : HH:MM:SS (heure, minute, seconde)
- Les valeurs nulles (`NULL`) sont possibles dans tous les domaines. Pour déclarer qu'un attribut ne doit pas être nul, il faut rajouter **NOT NULL**
- **CREATE DOMAIN** nom-client **char(20)**

Création des tables

- On utilise la clause **CREATE TABLE**
  CREATE TABLE compte (
  Num **int NOT NULL**,
  Solde **int**,
  NomTit **varchar(20)**)

- Rajout de contraintes :
  CREATE TABLE compte (
  Num **int NOT NULL**,
  Solde **int DEFAULT** 0,
  NomTit **varchar(20)**,
  PRIMARY KEY (Num),
  CHECK (Num $\geq$ 1))

- En SQL92, si un attribut est clé alors il est différent de `NULL`

Clé étrangère

Soient Personne(NSS, Nom) et
Voiture(Matricule, modèle, Proprio).

"Proprio" correspond au NSS du propriétaire. C'est une *clé étrangère* dans le schéma `Voiture` car c'est une clé dans un autre schéma.

```
CREATE TABLE Voiture (
      Matricule CHAR(8),
      Modele    CHAR(10),
      Proprio   CHAR(3),
      PRIMARY KEY(Matricule),
      FOREIGN KEY(Proprio) REFERENCES Personne
              ON [DELETE | UPDATE] CASCADE|
                                   RESTRICT|
                                   SET NULL
      )
```

CASCADE : Si une personne est supprimée, alors les voitures qu'elle possède sont supprimées.
RESTRICT : Le système refuse la suppression d'une personne s'il y a des voitures qui lui sont rattachées. C'est l'option par défaut.
SET NULL : Si une personne est supprimée, alors l'attribut `Proprio` prend la valeur `NULL`

L'insertion d'une voiture ne peut se faire que si le "proprio" existe dans `Personne` (ou bien valeur nulle)

Valeurs nulles

Employé	Nom	Salaire
	Dupont	10000
	Martin	NULL

SELECT *
FROM Employé
WHERE Salaire ¿ 12000

Ne retourne aucun tuple. Pareil si la condition est :
WHERE Salaire ¡ 8000

SELECT SUM(Salaire) **FROM** Employé;
Retourne 10000
SELECT COUNT(Salaire) **FROM** Employé;
Retourne 2
SELECT AVG(Salaire) **FROM** Employé;
Retourne 10000
SELECT COUNT(*) **FROM** Employé
WHERE Salaire IS NOT NULL;
Retourne 1

Mise à jour des vues

Personne(Nom,Salaire). Supposons que la table Personne est vide.
CREATE VIEW Gros_Salaire AS
SELECT *
FROM Personne
WHERE Salaire $>$ 10000

INSERT INTO Gros_Salaire VALUES(" Martin" , 5000)

Si on fait

SELECT * FROM Gros_Salaire; on n'obtient aucun tuple.

Si à la création de la vue on rajoute l'option

WITH CHECK OPTION

alors l'insertion est refusée.

Les mises à jours des vues sont traduites en des mises à jours des tables sous-jacente. *La traduction n'est pas toujours unique*.
$\Longrightarrow$ Certaines vues ne permettent pas des mises à jour.

Jointure externe

Si on fait Personne⋈ Voiture, on n'aura que les personnes qui ont une(des) voiture(s) qui sont dans le résultat.

```
SELECT  *
FROM    Personne P Left Outer Join Voiture V
        ON P.NSS = V.Proprio
```

Cette requête retourne aussi les personnes n'ayant pas de voiture.
Ces tuples auront des *valeurs nulles* pour les champs provennant de Voiture.

Si on met juste **Outer Join** alors on aura les personnes sans voitures et les voitures sans Propriétaire.

La jointure est exprimée par : T1 **Inner Join** T2 **On** Condition

Manipulation de schéma

- La commande **DROP TABLE** permet de supprimer une table.
  Ex : **DROP TABLE** compte.

- Si une vue est définie sur la table `compte` alors il faut utiliser
  DROP TABLE compte **CASCADE**

- La commande **ALTER TABLE** permet de modifier le schéma d'une relation. Exemple :
  ALTER TABLE compte **ADD** Date_ouverture **date**
  ALTER TABLE compte **DROP** Solde **CASCADE**