Rappel

Trois couches : Externe, conceptuelle, interne

L'utilisateur ne voit que la vue de la base qui lui a été attribuée.

Il spécifie ce qu'il veut via un langage de requête déclaratif .

Une fonction défini le passage schéma externe $\longrightarrow$ schéma conceptuel

Une compilation effectue le passage conceptuel $\longrightarrow$ interne

Chacune des couches peut être modifiée indépendamment des autres.

Les performances dépendent de la qualité de la compilation $\longrightarrow$ restriction des possibilité de l'utilisateur : SQL.

Le modèle relationnel

MODÈLE DE BASES DE DONNÉES :

LDD (langage de définition de données) +

LMD (langage de manipulation de données)

MODÈLE RELATIONNEL (Ted Codd 1970) :

Les données sont structurées en tables

LANGAGES : Algèbre, Calcul, SQL

BASES : Calcul des prédicats du 1er ordre

Langage de définition de données

TABLE : *RELATION*, e.g., *Film*

COLONNES : *ATTRIBUT*, e.g., *Titre*

LIGNES : *TUPLE* (ou n-uplet ou enregistrement)

titre	année	genre
Alien	1979	Science-Fiction
Vertigo	1958	Suspense
Volte-face	1997	Thriller
Pulp-Fiction	1995	Policier

Vocabulaire

DOMAINE (de valeur) : int, string, etc...

ATTRIBUTS : colonne d'une table, associé à un domaine.

SCHÉMA de relation: liste de paires attribut, domaine.

Film(titre:string, année:int, genre:string)

SCHÉMA de base de donnée: liste de schémas de relation.

BD = { Film, Cinéma, Acteur }

TUPLE d'une relation (ou n-uplet ou enregistrement): une ligne bien typée d'une table.

un tuple de Film $\in$ string$\times$int$\times$string

INSTANCE d'un schéma de relation : un ensemble de tuples bien typés pour cette relation.

INSTANCE d'un schéma : Associe à chaque nom de relation du schéma une instance.

Vue ensembliste

- En théorie

  Pas deux lignes identiques

  Tous les attributs d'une ligne sont connus

  L'ordre des lignes est à priori sans importance

- En pratique

C'est un peu plus compliqué...

Langage de manipulation de données

- 1989 SQL
- 1992 SQL2, révision majeure de la syntaxe
- 2000 SQL3, rajout d'une dose d'orienté objet

SQL : principe

SQL (Structured Query Language) est le langage de requêtes standard pour les SGBD relationnels

Expression d'une requête par un bloc *SELECT FROM WHERE*

SELECT	<liste des attributs à projeter>
FROM	<liste des relations arguments>
WHERE	<conditions sur un ou plusieurs attributs>

Projection

Soit le schéma de relation **COMMANDES**(numéro,cnom,pnom,quantité)

REQUÊTE: *Information sur toutes les commandes*

SQL:

```
SELECT numéro,cnom,pnom,quantité
FROM   COMMANDES
```

ou

```
SELECT *
FROM   COMMANDES
```

Projection : distinct

Soit le schéma de relation **COMMANDES**(numéro,cnom,pnom,quantité)

REQUÊTE: *Produits commandés*

```
SELECT pnom
FROM   COMMANDES
```

NOTE: Contrairement à la théorie ensembliste, SQL n'élimine pas les dupliqués . Pour les éliminer on utilise DISTINCT :

```
SELECT DISTINCT pnom
FROM   COMMANDES
```

Le DISTINCT peut être remplacé par la clause UNIQUE dans certains systèmes

Sélection

Soit le schéma de relation **COMMANDES**(numéro,cnom,pnom,quantité)

REQUÊTE: *Produits commandés par Jean*

SQL:

```
SELECT pnom
FROM   COMMANDES
WHERE  cnom = 'Jean'
```

REQUÊTE: *Produits commandés par Jean en quantité supérieure à 100*

SQL:

```
SELECT pnom
FROM   COMMANDES
WHERE  cnom = 'Jean'
AND    quantité > 100
```

Conditions de sélection en SQL : Suite

Les comparateurs BETWEEN, LIKE, IS NULL, IN

Soit le schéma de relation **COMMANDES**(numéro,cnom,pnom,quantité)

REQUÊTE: *Clients dont le nom commence par "C"*

SQL:

```
SELECT cnom
FROM   COMMANDES
WHERE  cnom LIKE 'C%'
```

NOTE: Le littéral qui suit LIKE doit être une chaîne de caractères éventuellement avec des caractères jokers (_, %).

La valeur NULL est une valeur “spéciale” qui représente une *valeur (information) inconnue*.

Soit le schéma de relation **FOURNISSEUR**(fnom,statut,ville)

REQUÊTE: *Fournisseurs dont l’adresse est inconnue.*

SQL:

```
SELECT fnom
FROM   FOURNISSEUR
WHERE  ville IS NULL
```

Jointure

Soit le schéma de relations

COMMANDES(num,cnom,pnom,quantité)
FOURNITURE(pnom,fnom,prix)

REQUÊTE: *Nom, Coût, Fournisseur des Produits commandés par Jean*

SQL:

```
SELECT COMMANDES.pnom, prix, fnom
FROM   COMMANDES, FOURNITURE
WHERE  cnom = 'Jean' AND
       COMMANDES.pnom = FOURNITURE.pnom
```

NOTE:

```
SELECT COMMANDES.pnom, prix, fnom
FROM   COMMANDES, FOURNITURE
```

équivaut à un produit cartésien des deux relations, suivi d'une projection.

Soit le schéma de relation **EMPLOYE**(enom,dep,sal)

REQUÊTE: *Nom et Salaire des Employés gagnant plus que l'employé de nom 'Jean'*

SQL:

```
SELECT E1.enom, E1.sal
FROM   EMPLOYE E1, EMPLOYE E2
WHERE  E2.nom = 'Jean' AND
       E1.sal > E2.sal
```

Opérations ensemblistes

COMMANDES(num,cnom,pnom,quantité)
FOURNITURE(pnom,fnom,prix)

REQUÊTE: *Produits qui coûtent plus que 1000F ou ceux qui sont commandés par Jean*

SQL:

```
SELECT pnom
FROM   FOURNITURE
WHERE  prix >= 1000
   UNION
SELECT pnom
FROM   COMMANDES
WHERE  cnom = 'Jean'
```

NOTE: L'union élimine les dupliqués. Pour garder les dupliqués on utilise l'opération `UNION ALL` : le résultat contient chaque n-uplet $a+b$ fois, où a et b est le nombre d'occurrences du n-uplet dans la première et la deuxième requête.

La différence ne fait pas partie du standard.

EMPLOYE(enom,dep,sal)

DEPARTEMENT(deptno,dnom,loc)

REQUÊTE: *Départements sans employés*

SQL:

```
SELECT dnom
FROM   DEPARTEMENT
   EXCEPT
SELECT dep
FROM   EMPLOYE
```

NOTE: La différence élimine les dupliqués. Pour garder les dupliqués on utilise l'opération `EXCEPT ALL` : le résultat contient chaque n-uplet $a - b$ fois, où a et b est le nombre d'occurrences du n-uplet dans la première et la deuxième requête.

L'intersection ne fait pas partie du standard.

EMPLOYE(enom,dep,sal)

DEPARTEMENT(deptno,dnom,loc)

REQUÊTE: *Départements ayant des employés qui gagnent plus que 20000F et qui se trouvent à Paris*

SQL:

```
SELECT deptno
FROM   DEPARTEMENT
WHERE  loc = 'Paris'
  INTERSECT
SELECT deptno
FROM   EMPLOYE,DEPARTEMENT
WHERE  sal > 20000
AND    dep=dnom
```

NOTE: L'intersection élimine les dupliqués. Pour garder les dupliqués on utilise l'opération `INTERSECT ALL` : le résultat contient chaque n-uplet $min(a, b)$ fois, où a et b est le nombre d'occurrences du n-uplet dans la première et la deuxième requête.

Requête imbriquées simples

Soit le schéma de relations

COMMANDES(num,cnom,pnom,quantité)
FOURNITURE(pnom,fnom,prix)

REQUÊTE: *Nom, prix et fournisseurs des Produits commandés par Jean*

SQL:

```
SELECT pnom,prix,fnom
FROM   FOURNITURE
WHERE  pnom IN (SELECT pnom
                FROM   COMMANDES
                WHERE  cnom = 'Jean')
```

COMMANDES(num,cnom,pnom,quantité)
FOURNITURE(pnom,fnom,prix)

REQUÊTE: *Nom, prix et fournisseurs des Produits commandés par Jean*

ou

```
SELECT FOURNITURE.pnom,prix,fnom
FROM   FOURNITURE,COMMANDES
WHERE  FOURNITURE.pnom = COMMANDES.pnom
AND    cnom = ``Jean''
```

La différence s'exprime aussi par deux blocs SFW imbriqués

Soit le schéma de relations

EMPLOYE(enom,dep,sal)

DEPARTEMENT(deptno,dnom,loc)

REQUÊTE: *Départements sans employés*

SQL:

```
SELECT dnom
FROM   DEPARTEMENT
WHERE  dnom NOT IN (SELECT DISTINCT dep
                    FROM   EMPLOYE)
```

EMPLOYE(enom,dep,sal)

DEPARTEMENT(deptno,dnom,loc)

REQUÊTE: *Départements sans employés*

ou

```
SELECT dnom
FROM DEPARTEMENT
EXCEPT
SELECT DISTINCT dep
FROM EMPLOYE
```

Requêtes imbriquées plus complexes : EXISTS

Soit le schéma de relations

FOURNISSEUR(fnom,status,ville)
FOURNITURE(pnom,fnom,prix)

REQUÊTE: *Fournisseurs qui fournissent au moins un produit*

```
SELECT fnom
FROM   FOURNISSEUR
WHERE  EXISTS (SELECT *
               FROM   FOURNITURE
               WHERE  fnom = FOURNISSEUR.fnom)
```

NOTE: La condition EXISTS (SELECT * FROM . . .) est vraie ssi le résultat du bloc (SELECT * FROM . . .) n'est pas vide.

Soit le schéma de relations

FOURNISSEUR(fnom,status,ville)

FOURNITURE(pnom,fnom,prix)

REQUÊTE: *Fournisseurs qui ne fournissent aucun produit*

SQL:

```
SELECT fnom
FROM   FOURNISSEUR
WHERE  NOT EXISTS (SELECT *
                   FROM   FOURNITURE
                   WHERE  fnom = FOURNISSEUR.fnom)
```

NOTE: La condition NOT EXISTS (SELECT * FROM . . .) est vraie ssi le résultat du bloc (SELECT * FROM . . .) est vide.

Si θ est un des opérateurs de comparaison $<, =, >, \ldots$

- La condition x θ ANY (SELECT R_i.y FROM $R_1, \ldots R_n$ WHERE p) est équivalente à

  EXISTS (SELECT * FROM $R_1, \ldots R_n$ WHERE p AND x θ R_i.y)

- La condition x θ ALL (SELECT R_i.y FROM $R_1, \ldots R_n$ WHERE p) est équivalente à

  NOT EXISTS (SELECT * FROM $R_1, \ldots R_n$ WHERE p AND NOT (x θ R_i.y))

REQUÊTE: *Nom, prix et fournisseur des produits commandés par Jean*

```
SELECT pnom, prix, fnom
FROM   FOURNITURE
WHERE  EXISTS  (SELECT * FROM   COMMANDE
                WHERE CNOM = 'Jean'
                  AND pnom = FOURNITURE.pnom)

SELECT pnom, prix, fnom
FROM   FOURNITURE
WHERE  pnom = ANY (SELECT pnom
                   FROM   COMMANDE
                   WHERE  cnom = 'Jean')
```

REQUÊTE: *Fournisseurs qui fournissent au moins un produit avec un coût supérieur au coût des produits fournis par Jean*

```
SELECT DISTINCT P1.fnom
FROM   FOURNITURE P1
WHERE  NOT EXISTS  (SELECT *
                    FROM   FOURNITURE P2
                    WHERE  P2.fnom = 'Jean'
                    AND    P1.prix <= P2.prix)
```

```
SELECT DISTINCT fnom
FROM   FOURNITURE
WHERE  prix > ALL (SELECT prix
                   FROM   FOURNITURE
                   WHERE fnom = 'Jean')
```

Soit le schéma de relations

FOURNITURE(fnom,pnom,quantité)
FOURNISSEUR(fnom,fnom,status,ville)

REQUÊTE: *Fournisseurs qui fournissent tous les produits*

REQUÊTE: *Fournisseurs qui fournissent tous les produits*

SQL:

```
SELECT fnom
FROM   FOURNISSEUR
WHERE  NOT EXISTS
   (SELECT *
    FROM  FOURNITURE F1
    WHERE NOT EXISTS
       (SELECT *
        FROM  FOURNITURE F2
        WHERE F2.fnom = FOURNISSEUR.fnom
        AND   F2.pnom = F1.pnom))
```

Fonctions de Calcul: COUNT, SUM, AVG, MIN, MAX

REQUÊTE: *Nombre de Fournisseurs de Paris*

```
SELECT COUNT(*)
FROM   FOURNISSEUR
WHERE  ville = 'Paris'
```

REQUÊTE: *Nombre de Fournisseurs qui fournissent actuellement des produits*

```
SELECT COUNT(DISTINCT fnom)
FROM   FOURNITURE
```

NOTE: La fonction COUNT(*) compte le nombre des n-uplets du résultat d'une requête sans élimination des dupliqués ni vérification des valeurs nulles. Dans le cas contraire on utilise la clause COUNT(UNIQUE . . .).

REQUÊTE: *Quantité totale de Briques commandées*

```
SELECT SUM (quantité)
FROM   COMMANDES
WHERE  pnom = 'Brique'
```

REQUÊTE: *Coût moyen de Briques fournies*

```
SELECT AVG (prix)
FROM   FOURNITURE
WHERE  pnom = 'Brique'
```

ou

```
SELECT SUM (prix)/COUNT(prix)
FROM FOURNITURE
WHERE pnom = 'Brique'
```

REQUÊTE: *Le prix des briques qui sont les plus chères.*

```
SELECT MAX (prix)
FROM   FOURNITURE
WHERE  pnom = 'Brique';
```

REQUÊTE: *Fournisseurs de briques dont le prix est en dessous du prix moyen*

```
SELECT fnom
FROM   FOURNITURE
WHERE  pnom = 'Brique' AND
       prix < (SELECT AVG(prix)
               FROM   FOURNITURE
               WHERE pnom = 'Brique')
```

Opérations d’Agrégation: GROUP BY

REQUÊTE: *Nombre de fournisseurs* ***par ville***

```
SELECT ville, COUNT(fnom)
FROM   FOURNISSEUR
GROUP BY ville
```

La base et le résultat :

ville	fnom
PARIS	TOTO
PARIS	DUPOND
LYON	DURAND
LYON	LUCIEN
LYON	REMI

ville	COUNT(fnom)
PARIS	2
LYON	3

NOTE: La clause GROUP BY permet de préciser les attributs de partitionnement des relations déclarées dans la clause FROM. Par exemple on regroupe les fournisseurs par ville.

REQUÊTE: *Donner pour chaque produit fourni son prix moyen*

```
SELECT pnom, AVG (prix)
FROM   FOURNITURE
GROUP BY pnom
```

Résultat:

pnom	AVG (prix)
BRIQUE	10.5
ARDOISE	9.8

REQUÊTE: *Produits fournis par deux ou plusieurs fournisseurs avec un prix supérieur à 100 Euros*

```
SELECT pnom
FROM   FOURNITURE
WHERE  prix > 100
GROUP BY pnom
HAVING COUNT(*) >= 2
```

Avant la clause HAVING:

pnom	fnom	prix
BRIQUE	TOTO	105
ARDOISE	LUCIEN	110
ARDOISE	DURAND	120

Après la clause HAVING

pnom	fnom	prix
ARDOISE	LUCIEN	110
ARDOISE	DURAND	120

NOTE: La clause HAVING permet d'éliminer des partitionnements, comme la clause WHERE élimine des n-uplets du résultat d'une requête. Par exemple on garde les produits dont le nombre des fournisseurs est $>= 2$. De cette façon des conditions de sélection peuvent être appliquées avant le calcul d'agrégat (clause WHERE) mais aussi après (clause HAVING).

REQUÊTE: *Nom et prix moyen des produits fournis par des fournisseurs Parisiens et dont le prix minimum est supérieur à 1000 Euros*

```
SELECT pnom, AVG(prix)
FROM   FOURNITURE, FOURNISSEUR
WHERE  ville = 'Paris' AND
       FOURNITURE.fnom = FOURNISSEUR.fnom
GROUP BY pnom
HAVING MIN(prix) > 1000
```

ORDER BY

En général, le résultat d'une requête SQL n'est pas trié. Pour trier le résultat par rapport aux valeurs d'un ou de plusieurs attributs, on utilise la clause `ORDER BY` :

```
SELECT ville, fnom, pnom
FROM FOURNITURE, FOURNISSEUR
WHERE FOURNITURE.fnom = FOURNISSEUR.fnom
ORDER BY ville, fnom DESC
```

Le résultat est trié par les villes (ASC) et le noms des fournisseur dans l'ordre inverse (DESC).

SQL : vue ensembliste

- En théorie

  Pas deux lignes identiques

  Tous les attributs d'une ligne sont connus

  L'ordre des lignes est à priori sans importance

- En pratique

  On n'élimine pas les duplicata sauf si explicitement demandé

  Certains attributs peuvent ne pas être connus

  L'ordre des lignes d'un résultat peut être imposé

Exercices

Traduire les requêtes de la feuille de TD2 en SQL.