

Chapitre 3 - SQL, un langage relationnel (partie interrogation)

1 SQL - Interrogation

- Introduction
- Clause Select
- Opérateurs ensemblistes
- Produits de relations
- Emboîtement de requêtes
- Aggregations simples
- Aggregations avec partition

Préliminaires

SQL est un langage ANSI/ISO pour interroger et manipuler des données relationnelles.
Conçu pour être un langage lisible par les personnes.

- Opérations de définition des données
- Opérations de modification des données
- Opérations relationnelles
- Opérations d'agrégation

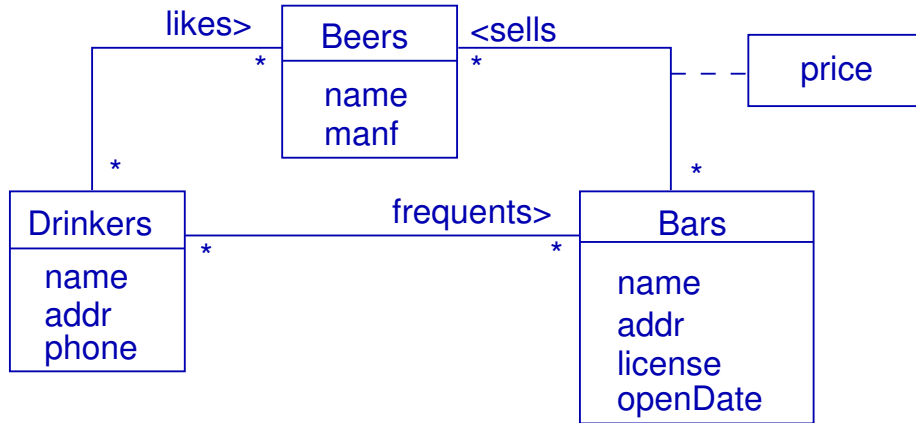
Préliminaires

SQL est un langage ANSI/ISO pour interroger et manipuler des données relationnelles.
Conçu pour être un langage lisible par les personnes.

- Opérations de définition des données
- Opérations de modification des données
- Opérations relationnelles
- Opérations d'agrégation

Facile d'exprimer des requêtes simples
TRÈS difficile d'exprimer des requêtes un peu plus complexes !

Une BD jouet¹



¹Le schéma de cette base de données est tirée de : *Database Systems: The Complete Book (2e édition)*, écrit par Hector Garcia-Molina, Jeff Ullman, and Jennifer Widom, 2008. Les valeurs sont d'inspiration personnelle.

Spécification des relations

Drinkers (name, addr, phone)

Beers (name, manf)

Bars (name, addr, license, opendate)

Likes (drinker, beer)

Likes[drinker] \subseteq Drinkers[name]

Likes[beer] \subseteq Beers[name]

Sells (bar, beer, price)

price > 0

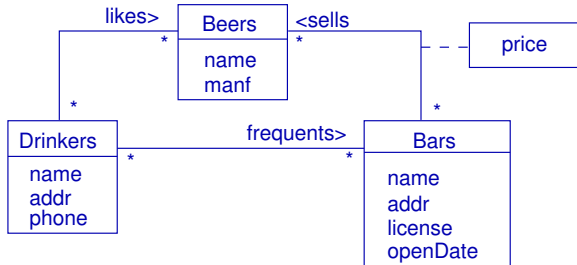
Sells[beer] \subseteq Beers[name]

Sells[bar] \subseteq Bars[name]

Frequents (drinker, bar)

Frequents[drinker] \subseteq Drinkers[name]

Frequents[bar] \subseteq Bars[name]



Il faudrait décrire aussi chacun des domaines.

Valeurs des relations

Bars

name	addr	license	opendate
Australia Hotel	The Rocks	123456	12/1/1940
Coogee Bay Hotel	Coogee	966500	31/8/1980
Lord Nelson	The Rocks	123888	11/11/1920
Marble Bar	Sydney	122123	1/4/2001
Regent Hotel	Kingsford	987654	29/2/2000
Rose Bay Hotel	Rose Bay	966410	31/8/2000
Royal Hotel	Randwick	938500	26/6/1986

Notion de requête

Une requête est un *programme déclaratif* pour retrouver des données d'une base de données.

Notion de requête

Une requête est un *programme déclaratif* pour retrouver des données d'une base de données.

Exemple : *Donner dans quel bar et pour quel prix la bière 'Victoria Bitter' est vendue.*

Notion de requête

Une requête est un *programme déclaratif* pour retrouver des données d'une base de données.

Exemple : *Donner dans quel bar et pour quel prix la bière 'Victoria Bitter' est vendue.*

Soit R la relation que décrit la requête. Ci-dessous, on donne sa spécification :

/ $R(\underline{bar}, price) : \langle b, p \rangle \in R \iff b$ est un bar qui vend la bière 'Victoria Bitter' pour le prix p . */*

...

Bars (name, addr, license, opendate)

Likes (drinker, beer)

Sells (bar, beer, price)

...

Notion de requête

Une requête est un *programme déclaratif* pour retrouver des données d'une base de données.

Exemple : *Donner dans quel bar et pour quel prix la bière 'Victoria Bitter' est vendue.*

Soit R la relation que décrit la requête. Ci-dessous, on donne sa spécification :

/ $R(\underline{bar}, price) : \langle b, p \rangle \in R \iff b$ est un bar qui vend la bière 'Victoria Bitter' pour le prix p . */*

...

Bars (name, addr, license, opendate)

Likes (drinker, beer)

Sells (bar, beer, price)

...

SELECT bar, price

FROM Sells

WHERE beer = 'Victoria Bitter';

/ un ensemble de noms d'attributs */*

/ un ensemble de noms de relations */*

/ une expression booléenne */*

Avec un SGBD....

```
SQL> SELECT bar, price  
2 FROM Sells  
3 WHERE beer = 'Victoria Bitter';
```

bar	price
Coogee Bay Hotel	2.3
Marble Bar	2.8
Regent Hotel	2.2
Royal Hotel	2.3

4 rows selected

```
SQL>
```

Projection, sélection

Donner les bars (nom et adresse) qui sont situés à Sydney

Projection, sélection

Donner les bars (nom et adresse) qui sont situés à Sydney

Soit R la relation que décrit la requête. Ci-dessous, on donne sa spécification :

/ $R(\underline{bar}, address) : \langle b, a \rangle \in R \iff b$ est un bar situé à Sydney, a est son adresse. */*

Projection, sélection

Donner les bars (nom et adresse) qui sont situés à Sydney

Soit R la relation que décrit la requête. Ci-dessous, on donne sa spécification :

/ $R(\underline{bar}, address) : \langle b, a \rangle \in R \iff b \text{ est un bar situé à Sydney, } a \text{ est son adresse.} \text{ */}$*

Drinkers (name, addr, phone)

Beers (name, manf)

Bars (name, addr, license, opendate)

...

Projection, sélection

Donner les bars (nom et adresse) qui sont situés à Sydney

Soit R la relation que décrit la requête. Ci-dessous, on donne sa spécification :

/ $R(\underline{bar}, address) : \langle b, a \rangle \in R \iff b$ est un bar situé à Sydney, a est son adresse. */*

Drinkers (name, addr, phone)

Beers (name, manf)

Bars (name, addr, license, opendate)

...

En SQL :

SELECT
FROM
WHERE

Projection, sélection

Donner les bars (nom et adresse) qui sont situés à Sydney

Soit R la relation que décrit la requête. Ci-dessous, on donne sa spécification :

/ $R(\underline{bar}, address) : \langle b, a \rangle \in R \iff b$ est un bar situé à Sydney, a est son adresse. */*

Drinkers (name, addr, phone)

Beers (name, manf)

Bars (name, addr, license, opendate)

...

En SQL :

SELECT

FROM Bars

WHERE

/ relations (une ou plusieurs) */*

Projection, sélection

Donner les bars (nom et adresse) qui sont situés à Sydney

Soit R la relation que décrit la requête. Ci-dessous, on donne sa spécification :

/ $R(\underline{bar}, address) : \langle b, a \rangle \in R \iff b$ est un bar situé à Sydney, a est son adresse. */*

Drinkers (name, addr, phone)

Beers (name, manf)

Bars (name, addr, license, opendate)

...

En SQL :

SELECT

FROM Bars

WHERE addr = 'Sydney';

/ relations (une ou plusieurs) */*
/ sélection */*

Projection, sélection

Donner les bars (nom et adresse) qui sont situés à Sydney

Soit R la relation que décrit la requête. Ci-dessous, on donne sa spécification :

/ $R(\underline{bar}, address) : \langle b, a \rangle \in R \iff b$ est un bar situé à Sydney, a est son adresse. */*

Drinkers (name, addr, phone)

Beers (name, manf)

Bars (name, addr, license, opendate)

...

En SQL :

```
SELECT name, addr
FROM Bars
WHERE addr = 'Sydney';
```

```
/* projection */
/* relations (une ou plusieurs) */
/* sélection */
```

Question

Quel type d'information on met (généralement) dans la clause **SELECT** (ce qui correspond à une Projection) ?

- A. Attributs
- B. Relations
- C. Conditions

Question

Quel type d'information on met (généralement) dans la clause **FROM** ?

- A. Attributs
- B. Relations
- C. Conditions

Question

Quel type d'information on met (généralement) dans la clause **WHERE** (ce qui correspond à une Sélection)?

- A. Attributs
- B. Relations
- C. Conditions

Opérateurs (avec ordre de priorité) :

Priorité	Opérateur	Opération
1	*, /	multiplication, division
2	+, -,	addition, soustraction, concaténation
3	=, !=, <=, LIKE , IN	comparaison
4	NOT	négation logique
5	AND	conjonction
6	OR	disjonction

Valeurs multiples (doublons)

Donner les bières qui coûtent moins de \$2.5

Soit R la relation décrite par la requête.

/ $R(\underline{beer}) : \langle b \rangle \in R \iff b$ est une bière vendue pour moins de \$2.5 */*

Valeurs multiples (doublons)

Donner les bières qui coûtent moins de \$2.5

Soit R la relation décrite par la requête.

$/^ R(\underline{beer}) : \langle b \rangle \in R \iff b \text{ est une bière vendue pour moins de } \$2.5 \text{ } */$*

Le résultat attendu est :

beer
New
Old
Victoria Bitter

Valeurs multiples (doublons)

Donner les bières qui coûtent moins de \$2.5

Soit R la relation décrite par la requête.

$/ R(\underline{beer}) : \langle b \rangle \in R \iff b \text{ est une bière vendue pour moins de } \$2.5 */$*

Le résultat attendu est :

beer
New
Old
Victoria Bitter

En SQL :

```
SELECT beer
FROM Sells
WHERE price <= 2.5;
```

Et le résultat est :

beer
New
Old
Victoria Bitter
New
Victoria Bitter
New
Old
Victoria Bitter

Éliminer les doublons (à l'affichage)

Lorsqu'il est nécessaire d'obtenir un résultat sans répétition de valeurs :

```
SELECT DISTINCT beer  
FROM Sells  
WHERE price <= 2.5;
```

Éliminer les doublons (à l'affichage)

Lorsqu'il est nécessaire d'obtenir un résultat sans répétition de valeurs :

```
SELECT DISTINCT beer  
FROM Sells  
WHERE price <= 2.5;
```

A utiliser avec précaution :

Éliminer les doublons (à l'affichage)

Lorsqu'il est nécessaire d'obtenir un résultat sans répétition de valeurs :

```
SELECT DISTINCT beer  
FROM Sells  
WHERE price <= 2.5;
```

A utiliser avec précaution :

- Penser au coût de cette opération !

Éliminer les doublons (à l'affichage)

Lorsqu'il est nécessaire d'obtenir un résultat sans répétition de valeurs :

```
SELECT DISTINCT beer  
FROM Sells  
WHERE price <= 2.5;
```

A utiliser avec précaution :

- Penser au coût de cette opération !
- Quelques fois inutile : `SELECT name FROM Drinkers;`

Ordonner les nuplets

La clause `order by` s'applique uniquement aux attributs de la projection (c'est-à-dire contenus dans la clause `select`) :

```
SELECT bar, beer, price
```

```
FROM Sells
```

```
ORDER BY price desc, bar asc;
```

Résultat :

bar	beer	price
Lord Nelson	Three Sheets	3.75
Lord Nelson	Old Admiral	3.75
Australia Hotel	Burraborang Bock	3.5
Coogee Bay Hotel	Sparkling Ale	2.8
Marble Bar	New	2.8
Marble Bar	Victoria Bitter	2.8
Marble Bar	Old	2.8
Coogee Bay Hotel	Old	2.5
...		

Union, Intersection, Différence

Dans la suite les requêtes Q1 et Q2 sont de la forme SELECT... FROM.... dont les **schémas sont compatibles** :

- SELECT A, B FROM... n'est pas compatible avec SELECT C FROM...
- SELECT A, B FROM... est compatible avec SELECT C, D FROM... ssi A et C sont comparables, ainsi que B et D.

Opérateurs

- UNION [all]
Pas d'élimination des doublons avec l'option all
- INTERSECT (intersection)
- MINUS (différence) / EXCEPT (Pour SQLite)

Les doublons sont éliminés sauf lors de l'utilisation de union all.

Différence : exemple

Donner les bières vendues à moins de \$3 et que John n'aime pas

Différence : exemple

Donner les bières vendues à moins de \$3 et que John n'aime pas

```
SELECT beer  
FROM Sells  
WHERE price < 3
```

MINUS

```
SELECT beer  
FROM Likes  
WHERE drinker = 'John';
```

Les deux arguments doivent construire des relations de schémas compatibles.

Intersection : exemple

Donner les bières qui sont aimées et aussi vendues dans au moins un bar

Intersection : exemple

Donner les bières qui sont aimées et aussi vendues dans au moins un bar

```
SELECT beer  
FROM Likes  
INTERSECT  
SELECT beer  
FROM Sells;
```

Union : exemple

Donner les consommateurs qui aiment la Sparkling Ale ou fréquentent le bar Lord Nelson.

Union : exemple

Donner les consommateurs qui aiment la Sparkling Ale ou fréquentent le bar Lord Nelson.

```
SELECT drinker
FROM Likes
WHERE beer = 'Sparkling Ale'
UNION
SELECT drinker
FROM Frequents
WHERE bar = 'Lord Nelson';
```

Produit relationnel (interne)

SELECT

FROM R1 JOIN R2 ON (P)

WHERE

/ \longrightarrow produits de relations */*

Le prédicat **P** est la condition de mise en correspondance du produit de R1 par R2.

Produit relationnel (interne)

```
SELECT  
FROM R1 JOIN R2 ON ( P )  
WHERE C
```

/ → produits de relations */*

/ → sélection */*

Le prédicat P est la condition de mise en correspondance du produit de R1 par R2.

Produit relationnel (interne)

```
SELECT  A1, A2, ..., An  
FROM R1 JOIN R2 ON ( P )  
WHERE  C
```

/ → projection */*

/ → produits de relations */*

/ → sélection */*

Le prédicat P est la condition de mise en correspondance du produit de R1 par R2.

Produit relationnel (interne)

```
SELECT  A1, A2, ..., An  
FROM R1 JOIN R2 ON ( P )  
WHERE  C
```

/ → projection */*

/ → produits de relations */*

/ → sélection */*

Le prédicat P est la condition de mise en correspondance du produit de R1 par R2.

Exemple : *Pour chaque bière vendue par l’Australia Hotel, donner son prix et son brasseur*

Produit relationnel (interne)

```
SELECT  A1, A2, ..., An  
FROM R1 JOIN R2 ON ( P )  
WHERE  C
```

/ → projection */*

/ → produits de relations */*

/ → sélection */*

Le prédicat P est la condition de mise en correspondance du produit de R1 par R2.

Exemple : *Pour chaque bière vendue par l’Australia Hotel, donner son prix et son brasseur*

/ Soit $R(\underline{beer}, price, manf) : \langle b, p, m \rangle \in R \iff b$ est une bière vendue à l’Australia Hotel, m est son brasseur et p son prix. */*

...

Beers (name, manf)

Bars (name, addr, license, opendate)

Sells (bar, beer, price)

Produit relationnel (interne)

```
SELECT  A1, A2, ..., An  
FROM R1 JOIN R2 ON ( P )  
WHERE  C
```

/ → projection */*
/ → produits de relations */*
/ → sélection */*

Le prédicat P est la condition de mise en correspondance du produit de R1 par R2.

Exemple : *Pour chaque bière vendue par l’Australia Hotel, donner son prix et son brasseur*

/ Soit $R(\underline{\text{beer}}, \text{price}, \text{manf}) : \langle b, p, m \rangle \in R \iff b \text{ est une bière vendue à l’Australia Hotel, } m \text{ est son brasseur et } p \text{ son prix.} */$*

...

Beers (name, manf)

Bars (name, addr, license, opendate)

Sells (bar, beer, price)

```
SELECT beer, price, manf  
FROM Beers JOIN Sells ON (name = beer)  
WHERE bar = 'Australia Hotel';
```

/ projection */*
/ condition de produit */*
/ condition de sélection */*

/ La clause “on” peut contenir n’importe quel prédicat pour exprimer le produit entre deux relations (ou plus) */*

Ambigüité de nom dans une requête

Plusieurs occurrences du même nom dans une requête

Pour chaque bar que John fréquente, donner son nom, les bières qu'il vend et leur prix.

/ R(bar, beer, price) : $\langle a, e, p \rangle \in R \iff$ le bar b , que John fréquente, vend la bière e pour le prix p . */*

```
SELECT bar, beer, price
FROM Sells JOIN Frequents ON (bar = bar)
WHERE drinker = 'John';
```

ERROR at line 2:

ORA-00918: column ambiguously defined

Notation qualifiée

Le nom complet d'un attribut est qualifié par un nom de relation :

```
SELECT Sells.bar, Sells.beer, Sells.price  
FROM Sells JOIN Frequents ON (Sells.bar = Frequents.bar)  
WHERE Frequents.drinker = 'John';
```

- Chaque attribut est qualifié par le nom de la relation à laquelle il appartient (plus facile à lire)
- Nécessaire pour les attributs définis dans plus d'une relation citée dans la clause from.

Donner, pour chaque bar que John fréquente, les bières qu'il vend et pour quel prix.

/ $R(\underline{\text{bar}}, \text{beer}, \text{price}) : \langle a, e, p \rangle \in R \iff \text{le bar } a, \text{ fréquenté par John, vend la bière } e \text{ pour le prix } p.$ */*

Notation qualifiée avec raccourcis

Une relation peut être renommée dans la clause from :

Donner, pour chaque bar que John fréquente, les bières qu'il vend et pour quel prix.

/ R(bar, beer, price) : $\langle a, e, p \rangle \in R \iff$ le bar a , fréquenté par John, vend la bière e pour le prix p . */*

```
SELECT S.bar, S.beer, S.price  
FROM Sells S JOIN Frequents F ON (S.bar = F.bar)  
WHERE F.drinker = 'John';
```

Donner, pour chaque bar que John fréquente, les bières qu'il vend et pour quel prix.

/ R(bar, beer, price) : $\langle a, e, p \rangle \in R \iff$ le bar a , fréquenté par John, vend la bière e pour le prix p . */*

Produit naturel

*La condition de produit porte sur un sous-ensemble des **attributs en commun***

```
SELECT B.name, addr, D.name  
FROM Bars B JOIN Drinkers D USING (addr)  
/* La condition de produit s'applique sur l'attribut commun addr */
```

Une autre version :

```
SELECT B.name, B.addr, D.name  
FROM Bars B JOIN Drinkers D ON (B.addr=D.addr)  
/* La condition de produit est explicite. */
```


Produit cartésien

Donner toutes les combinaisons possibles entre le consommateur John et tous les bars.

```
SELECT *  
FROM Bars CROSS JOIN Drinkers  
WHERE Drinkers.name = 'John';
```

Équivalent à...

```
SELECT *  
FROM Bars, Drinkers  
WHERE Drinkers.name = 'John';
```

Produits : en résumé

Soient R et S définies comme : R (X, Y, Z) et S (Y, Z, T)

- Produit cartésien : `FROM R CROSS JOIN S`
schéma : `R.X, R.Y, R.Z, S.Y, S.Z, S.T`
- Produit relationnel : `FROM R JOIN S ON (P)`
schéma : `R.X, R.Y, R.Z, S.Y, S.Z, S.T`
`P` est un prédicat valide sur `R.X, R.Y, R.Z, S.Y, S.Z, S.T`
La condition de produit est `P`
- Produit naturel : `FROM R JOIN S USING (Y)`
schéma : `R.X, Y, R.Z, S.Z, S.T`
La condition de produit est `R.Y = S.Y`

Citation multiple d'une même relation

Donner les paires de consommateurs différents qui aiment la même bière

Citation multiple d'une même relation

Donner les paires de consommateurs différents qui aiment la même bière

/ R(drinker1, drinker2) : $\langle d1, d2 \rangle \in R \iff$ les personnes $d1$ et $d2$ aiment au moins une bière en commun. */*

```
SELECT L1.drinker, L2.drinker
```

```
FROM Likes L1 JOIN Likes L2
```

```
ON (L1.beer = L2.beer AND L1.drinker <> L2.drinker);
```

Citation multiple d'une même relation

Donner les paires de consommateurs différents qui aiment la même bière

/ R(drinker1, drinker2) : $\langle d1, d2 \rangle \in R \iff$ les personnes $d1$ et $d2$ aiment au moins une bière en commun. */*

```
SELECT L1.drinker, L2.drinker  
FROM Likes L1 JOIN Likes L2  
      ON (L1.beer = L2.beer AND L1.drinker <> L2.drinker);
```

Questions :

La clause *SELECT DISTINCT* est-elle nécessaire pour éliminer les doublons ?

Comment assurer l'antisymétrie ?

R est antisymétrique si :

$$\langle X, Y \rangle \in R \implies (\langle Y, X \rangle \notin R \text{ or } X = Y)$$

On sait déjà que pour tout X, $\langle X, X \rangle \notin R$

Comment assurer l'antisymétrie ?

R est antisymétrique si :

$\langle X, Y \rangle \in R \implies (\langle Y, X \rangle \notin R \text{ or } X = Y)$

On sait déjà que pour tout X, $\langle X, X \rangle \notin R$

```
SELECT DISTINCT L1.drinker, L2.drinker  
FROM Likes L1 JOIN Likes L2  
ON (L1.beer = L2.beer AND L1.drinker < L2.drinker);
```

- Décomposition de la requête en sous-requêtes plus simples.
- Expression et test des sous-requêtes indépendamment les unes des autres.

Opérateur IN / NOT IN

*Donner les noms des bières et brasseur des **bières que John aime**.*

Opérateur IN / NOT IN

Donner les noms des bières et brasseur des bières que John aime.

```
(SELECT beer  
FROM Likes  
WHERE drinker = 'John');
```

Opérateur IN / NOT IN

Donner les noms des bières et brasseur des bières que John aime.

```
SELECT name, manf
FROM Beers
WHERE name IN (SELECT beer
                FROM Likes
                WHERE drinker = 'John');
```

Introduction de noms intermédiaires (clause WITH..AS ())

La clause WITH.. AS.. permet d'introduire des noms intermédiaires :

```
WITH BeersLikedByJohn AS (  
    SELECT beer  
    FROM Likes  
    WHERE drinker = 'John')  
SELECT name, manf  
FROM Beers JOIN BeersLikedByJohn ON (name=beer);
```

La durée de vie du nom introduit par la clause with est celle de la requête.

Introduction de noms intermédiaires (clause WITH..AS ())

La clause WITH.. AS.. permet d'introduire des nomes intermédiaires :

```
WITH BeersLikedByJohn AS (  
    SELECT beer  
    FROM Likes  
    WHERE drinker = 'John')  
SELECT name, manf  
FROM Beers JOIN BeersLikedByJohn ON (name=beer);
```

La durée de vie du nom introduit par la clause with est celle de la requête.

WITH AS pas toujours nécessaire:

```
SELECT name, manf  
FROM Beers JOIN Likes ON (name=beer)  
WHERE drinker = 'John';
```

Aggregation

Pour réduire une liste de valeurs à une seule valeur

- COUNT (*) → nombre de n-uplets
- COUNT (A) → nombre de valeurs en A
- COUNT (DISTINCT A) → nombre de valeurs différents en A
- AVG (A) → valeur moyenne des valeurs en A
- MIN (A) (ou MAX) → valeur minimale (ou maximale) des valeurs en A
- SUM (A) → somme des valeurs de A

Exemples

Quel est le prix moyen des bières vendues par Australia Hotel?

Exemples

Quel est le prix moyen des bières vendues par Australia Hotel?

```
SELECT      price  
FROM Sells  
WHERE bar = 'Australia Hotel';
```


Exemples

Quel est le prix moyen des bières vendues par Australia Hotel?

```
SELECT AVG (price)
FROM Sells
WHERE bar = 'Australia Hotel';
```

Exemples

Quel est le prix moyen des bières vendues par Australia Hotel?

```
SELECT AVG (price)
FROM Sells
WHERE bar = 'Australia Hotel';
```

Combien de bars sont situés dans The Rocks?

Exemples

Quel est le prix moyen des bières vendues par Australia Hotel?

```
SELECT AVG (price)
FROM Sells
WHERE bar = 'Australia Hotel';
```

Combien de bars sont situés dans The Rocks?

```
SELECT      *
FROM Bars
WHERE addr = 'The Rocks';
```

Exemples

Quel est le prix moyen des bières vendues par Australia Hotel?

```
SELECT AVG (price)
FROM Sells
WHERE bar = 'Australia Hotel';
```

Combien de bars sont situés dans The Rocks?

```
SELECT COUNT (*)
FROM Bars
WHERE addr = 'The Rocks';
```

Question

Donner le nom de la bière la moins chère

Beers (name, manf)

Bars (name, addr, license, opendate)

Sells (bar, beer, price)

...

- A. SELECT MIN(price)
FROM Sells
- B. **SELECT DISTINCT beer** *une bière peut être vendue au prix min dans plusieurs bars*
FROM SELLS
WHERE price IN (SELECT MIN(price) FROM Sells)
- C. SELECT beer, MIN(price)
FROM Sells

Partition

Partitionner une relation pour appliquer une agrégation sur chaque classe séparément

Combien de bières aime chaque consommateur ?

Résultat attendu :

drinker	beer
Adam	3
Gernot	2
John	4
Justin	3

Comment faire une partition

1. Construire une partition sur Likes

Adam	Crown Lager
Adam	Fosters
Adam	New
Gernot	Premium Lager
Gernot	Sparkling Ale
John	80/-
John	Bigfoot Barley Wine
John	Fosters
John	Three Sheets
Justin	Sparkling Ale
Justin	Fosters
Justin	Victoria Bitter

Comment faire une partition

1. Construire une partition sur Likes

Adam	Crown Lager
Adam	Fosters
Adam	New
Gernot	Premium Lager
Gernot	Sparkling Ale
John	80/-
John	Bigfoot Barley Wine
John	Fosters
John	Three Sheets
Justin	Sparkling Ale
Justin	Fosters
Justin	Victoria Bitter

En SQL :
SELECT ...
FROM Likes
GROUP BY drinker

2. Réduire à un n -uplet chaque classe de la partition

liste de drinkers (critère de partition)

→ drinker (une des valeurs)

liste de beer

→ entier (nombre de valeurs)

2. Réduire à un n -uplet chaque classe de la partition

liste de drinkers (critère de partition)

→ drinker (une des valeurs)

liste de beer

→ entier (nombre de valeurs)

En SQL :

```
SELECT drinker, COUNT (beer)
```

```
FROM Likes
```

```
GROUP BY drinker
```

Impact sur la sémantique de la clause select

Dans une requête contenant la clause GROUP BY, la clause select contient uniquement :

- Un ou plusieurs attributs parmi ceux du critère de la partition
- Une ou plusieurs agrégations appliquées aux autres attributs

Expression incorrecte :

```
SELECT drinker, addr, COUNT (beer)
FROM Likes JOIN Drinkers ON (drinker=name)
GROUP BY drinker
```

Expression correcte :

```
SELECT drinker, addr, COUNT (beer)
FROM Likes JOIN Drinkers ON (drinker=name)
GROUP BY drinker, addr
```

Filtrer une partition

Pour chaque bar qui vend plus de 2 bières, donner le nombre de bières qu'il vend et le nombre de consommateurs qui le fréquentent.

Filtrer une partition

Pour chaque bar qui vend plus de 2 bières, donner le nombre de bières qu'il vend et le nombre de consommateurs qui le fréquentent.

```
FROM Sells S JOIN Frequents F USING (bar)
```

Filtrer une partition

Pour chaque bar qui vend plus de 2 bières, donner le nombre de bières qu'il vend et le nombre de consommateurs qui le fréquentent.

```
FROM Sells S JOIN Frequents F USING (bar)
GROUP BY bar
```

Filtrer une partition

Pour chaque bar qui vend plus de 2 bières, donner le nombre de bières qu'il vend et le nombre de consommateurs qui le fréquentent.

```
FROM Sells S JOIN Frequents F USING (bar)
GROUP BY bar
HAVING COUNT (DISTINCT beer) > 2
```

Filtrer une partition

Pour chaque bar qui vend plus de 2 bières, donner le nombre de bières qu'il vend et le nombre de consommateurs qui le fréquentent.

```
SELECT bar,  
       COUNT (DISTINCT S.beer) AS nbBeers,  
       COUNT (DISTINCT F.drinker) AS nbDrinkers  
FROM Sells S JOIN Frequents F USING (bar)  
GROUP BY bar  
HAVING COUNT (DISTINCT beer) > 2
```


Encore des exemples...

Quels sont les bars qui vendent toutes les bières ?

Les bars qui vendent toutes les bières sont ceux, qui dans la relation Sells sont associés à un ensemble (soit A) de bières égal à l'ensemble des bières (vendues ou connues) soit B.

Or, on sait que : $|A| = |B| \wedge A \subseteq B \implies A = B$

En SQL, tester l'égalité de deux ensembles A et B lorsque $A \subseteq B$ revient donc à tester l'égalité des cardinalités des ensembles $|A|$ et $|B|$.

Encore des exemples...

Quels sont les bars qui vendent toutes les bières ?

Les bars qui vendent toutes les bières sont ceux, qui dans la relation Sells sont associés à un ensemble (soit A) de bières égal à l'ensemble des bières (vendues ou connues) soit B.

Or, on sait que : $|A| = |B| \wedge A \subseteq B \implies A = B$

En SQL, tester l'égalité de deux ensembles A et B lorsque $A \subseteq B$ revient donc à tester l'égalité des cardinalités des ensembles $|A|$ et $|B|$.

Pour chaque bar, combien de bières ?

Soit BieresParBar(bar,nbBeers) la relation associée :

```
SELECT bar, COUNT(beer) AS nbBeers FROM Sells GROUP BY bar
```

Encore des exemples...

Quels sont les bars qui vendent toutes les bières ?

Les bars qui vendent toutes les bières sont ceux, qui dans la relation Sells sont associés à un ensemble (soit A) de bières égal à l'ensemble des bières (vendues ou connues) soit B.

Or, on sait que : $|A| = |B| \wedge A \subseteq B \implies A = B$

En SQL, tester l'égalité de deux ensembles A et B lorsque $A \subseteq B$ revient donc à tester l'égalité des cardinalités des ensembles $|A|$ et $|B|$.

Pour chaque bar, combien de bières ?

Soit BieresParBar(bar,nbBeers) la relation associée :

```
SELECT bar, COUNT(beer) AS nbBeers FROM Sells GROUP BY bar
```

Combien de bières vendues au total ?

Soit BieresTot(nbTot) la relation associée :

```
SELECT COUNT(name) AS nbTot FROM Beers
```

Encore des exemples...

Quels sont les bars qui vendent toutes les bières ?

La requête est donc :

```
WITH BieresParBar AS (  
    SELECT bar, COUNT(beer) AS nbBeers  
    FROM Sells  
    GROUP BY bar  
    – <b, nbb> ∈ BieresParBar  $\iff$  le bar b vend nbb beers  
) , BieresTot AS (  
    SELECT COUNT(name) AS nbTot  
    FROM Beers  
    – <nbt> ∈ BieresTot  $\iff$  il y a nbt beers  
)  
SELECT bar  
FROM BieresParBar JOIN BieresTot ON (nbBeers = nbTot);
```

Encore des exemples...

*Quels sont les bars qui vendent toutes les bières ?
Même requête en utilisant l'opérateur HAVING*

Encore des exemples...

Quels sont les bars qui vendent toutes les bières ?

Même requête en utilisant l'opérateur HAVING

La requête est donc :

```
SELECT bar
FROM Sells
GROUP BY bar
HAVING COUNT(beer) =
  (SELECT COUNT(name)
   FROM Beers);
```

Quels sont les bars qui vendent le plus de bières ?

- Pour chaque bar combien de bières ? voir BieresParBar ci-dessus.
- Dans BieresParBar, quel est le plus grand nombre de bières ? Soit MaxBieres(nbMax) la relation associée :

```
SELECT MAX(nbBeers) AS nbMax  
FROM BieresParBar
```

- $\langle m \rangle \in \text{MaxBieres} \iff$ le bar qui vend le plus de bières, vend m bières

La requête est finalement :

```
WITH BieresParBar AS (  
  SELECT bar, COUNT(beer) AS nbBeers  
  FROM Sells  
  GROUP BY bar  
) , MaxBieres AS (  
  SELECT MAX(nbBeers) as nbMax  
  FROM BieresParBar  
)  
SELECT bar  
FROM BieresParBar JOIN MaxBieres ON (nbBeers = nbMax);
```

Sémantique opérationnelle des requêtes

- 5 SELECT *projection ou agrégations*
- 1 FROM *produits de relations*
- 2 WHERE *sélection sur la relation construite par les produits du from*
- 3 GROUP BY *partition*
- 4 HAVING *filtre sur la partition*

- Ce chapitre n'a couvert que la partie interrogation de SQL (la partie définition de relation, et modification de données est étudiée plus tard).
- Il y a souvent plusieurs expressions possibles pour une même requête.
 - ▶ Laquelle choisir ? Ce n'est pas une question simple...
 - ▶ Comme les requêtes sont optimisées par le SGBD, le programmeur n'a pas à se soucier du problème...
 - ▶ Cependant, dans le cas de certains SGBD, le temps de réponse peut être un problème.