SQL

Transparents de Jeffrey ULLMAN
(Stanford University)
Traduits/adaptés par C. Retoré
(Université Bordeaux 1)

Pourquoi SQL?

- SQL langage de haut niveau, évite de spécifier la manipulation des données comme il faudrait le faire par exemple en C++
- SQL fonctionne car les requêtes sont très efficacement optimisées pour des réponses rapides.

Requêtes à une table

SELECT
FROM
WHERE

Select-From-Where

Structure générale d'une requête:

SELECT attributs recherchés
FROM une ou plusieurs tables
WHERE condition sur les n-uplets des tables

Exemple pour tout ce cours

- Requêtes SQL sur le schéma suivant

Beers(name, manf)
Bars(name, addr, license)
Drinkers(name, addr, phone)
Likes(drinker, beer)
Sells(bar, beer, price)
Frequents(drinker, bar)

Les attributs d'une clef sont soulignés

Exemple

- En utilisant Beers(name, manf), quelles beers sont fabriquées par Anheuser-Busch?

```
SELECT name
FROM Beers
WHERE manf = 'Anheuser-Busch';
```


Resultat d'une requête

name
'Bud'
'Bud Lite'
'Michelob'

Relation à un attribut: name
Et les 1-uplest des beers fabriquées par Anheuser-Busch, comme Bud.

Sens d'une requête sur une relation

- Considérer la relation dans le FROM
- Appliquer la selection du WHERE
- Appliquer la projection du SELECT

Sémantique Opérationelle

- Pour implanter l'algorithme
- *Une variable de n-uplet* parcourt la table du FROM.
- Tester si le n-uplet "courrant" satisfait la condition du WHERE.
- SI c'est le cas, calculer les attributs ou expressions du SELECT avec les valeurs du n-uplet courant.

Contenu du champ SELECT

*

Renommage

Constantes

* Dans le SELECT

- Avec une relation dans le FROM clause, * dans le SELECT signifie tous les attributs de la relation
- Exemple avec Beers(name, manf):

```
SELECT *
FROM Beers
WHERE manf = 'Anheuser-Busch';
```


Résultat de la requête:

name	manf
'Bud'	'Anheuser-Busch'
'Bud Lite'	'Anheuser-Busch'
'Michelob'	'Anheuser-Busch'

Ici le résultat a tous les attribut de Beers.

Renommage des Attributs

- On peut changer le nom des attributs avec "AS <nouveau nom>"
- Exemple avec Beers(name, manf):

```
SELECT name AS beer, manf
FROM Beers
WHERE manf = 'Anheuser-Busch'
```


Résultat de la Requête

beer	manf
'Bud'	'Anheuser-Busch'
'Bud Lite'	'Anheuser-Busch'
'Michelob'	'Anheuser-Busch'

Expressions dans le SELECT

- Toute expression calculée à partir des attributs peut apparaître après SELECT.
- Exemple avec Sells(bar, beer, price):

```
SELECT bar, beer,
       price * 120 AS priceInYen
FROM Sells;
```


Résultat de la Requête

bar	beer	priceInYen
Joe's	Bud	300
Sue's	Miller	360
...	...	...

Autre Exemple: Constantes

- Exemple avec Likes(drinker, beer):

```
SELECT drinker,
       'likes Bud' AS whoLikesBud
FROM Likes
WHERE beer = 'Bud';
```


Résultat de la Requête

drinker	whoLikesBud
'Sally'	'likes Bud'
'Fred'	'likes Bud'
...	...

Le contenu du champ WHERE

Conditions composées
Expressions régulières

Conditions composées dans le WHERE

- Avec Sells(bar, beer, price), trouver le prix du Joe's Bar pour Bud:

```
SELECT price
FROM SELLS
WHERE bar = 'Joe''s Bar' AND
    beer = 'Bud';
```


Quelques Remarques

- Deux apostrophes pour représenter une apostrophe dans une chaîne entre apostrophes
- Dans le WHERE on peut utiliser AND, OR, NOT, et des parenthèses.
- SQL ne distingue pas majuscules et minuscules --- sauf entre apostrophes

Expressions régulières

- Le WHERE peut contenir des conditions comparant une chaîne à une expression régulière
- Forme générale:
  <Attribut> LIKE <expr. rég.>
  <Attribut> NOT LIKE <expr. rég.>
- Expr. Rég. % = chaîne quelconque;
  _ = caractère quelconque

Exemple

- avec Drinkers(name, addr, phone) trouver ceux dont le tél commence par 0556:

```
SELECT name
FROM Drinkers
WHERE phone LIKE '0556_____';
```


NULL et UNKNOWN

La (non) valeur NULL
Logique trivaluée

La (non) valeur NULL

- les n-uplets des relations de SQL peuvent avoir des composantes indéfinies: c'est le rôle de NULL
- Le sens dépend du contexte.
  Deux cas courants:
  - *Valeur inconnue Missing value* : personne dont on n'a pas l'adresse
  - *Inapplicable* : emploi pour un enfant

Comparaison avec NULL

- Logique trivaluée:
  TRUE, FALSE, UNKNOWN.
- Comparaison entre une valeur et NULL, donne UNKNOWN.
- Un n-uplet est sélectionné si le WHERE donne TRUE (et pas FALSE ni UNKNOWN).

Logique trivaluée

AND	TRUE	UNKNOWN	FALSE
TRUE	TRUE	UNKNOWN	FALSE
UNKNOWN	UNKNOWN	UNKNOWN	FALSE
FALSE	FALSE	FALSE	FALSE

OR	TRUE	UNKNOWN	FALSE
TRUE	TRUE	TRUE	TRUE
UNKNOWN	TRUE	UNKNOWN	UNKNOWN
FALSE	TRUE	UNKNOWN	FALSE

Exemple Surprenant

- En utilisant la relation Sells:

bar	beer	price
Joe's Bar	Bud	NULL

```
SELECT bar
FROM Sells
WHERE price < 2.00 OR price >= 2.00;
```

UNKNOWN UNKNOWN

UNKNOWN

Requêtes à plusieurs relations

Sens

Méthode d'évaluation

Requêtes à plusieurs relations

- Certaines requêtes utilisent plusieurs relations (jointure)
- On peut mettre plusieurs relations dans le SELECT.
- Il faut distinguer les attributs de même nom appartenant à des relations différentes: "<relation>.<attribute>"

Exemple

- Avec Likes(drinker, beer) et Frequents(drinker, bar), trouver les beers que Likes au moins un des clients du Joe's Bar.

```
SELECT beer
FROM Likes, Frequents
WHERE bar = 'Joe''s Bar' AND
   Frequents.drinker = Likes.drinker;
```

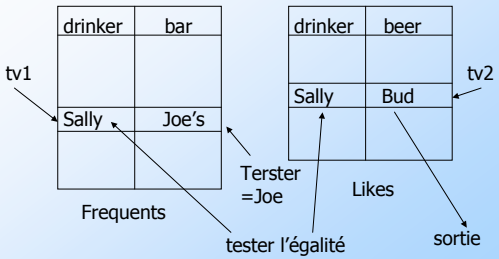

Sens des requêtes

- Ressemble au cas unaire:
  1. Faire le produit des relations du FROM
  2. Sélectionner les n-uplets qui satisfont le WHERE
  3. Projeter sur les attributs du SELECT.

Sémantique Opérationnnelle

- Une variable de n-uplet-variable par relation du FROM
  - Chacune de ces variables parcourt la table de la relation qui lui correspond.
- Si les valeurs des variables de n-uplets satisfont la condition du WHERE On garde les attributs demandés par le SELECT

Example



Variables explicites

- Parfois, plusieurs copies de la même relation.
- On distingue les variables correspondant aux différentes copies de la relation en leur donnant des noms différents dans le FROM
- On peut toujours le faire même si cela n'est pas nécessaire.

Exemple

- Avec Beers(name, manf), on peut trouver les Beers dont le manf est le même.
  - Pas de couples comme (Bud, Bud).
  - Pas deux fois les mêmes avec un ordre différent si (Bud, Miller) alors pas (Miller, Bud).

```
SELECT b1.name, b2.name
FROM Beers b1, Beers b2
WHERE b1.manf = b2.manf AND
   b1.name < b2.name;
```

Sous-requêtes

Sous-requêtes à une solution

IN

ALL

ANY

Sous-requêtes

- Une expression parenthésée SELECT-FROM-WHERE est une sous requête Elle peut être utilisée dans le FROM et le WHERE
- Exemple: on peut interroger une requête comme si elle était une relation prédéfinie en la plaçant dans le FROM
  - Il vaut mieux lui donner un nom comme on l'a fait précédemment.

Sous requêtes produisant UN n-uplet

- On peut utiliser cette sous requête comme valeur
  - La clef peut garantir qu'il n'y a qu'une valeur.
  - On obtient une erreur s'il n'y a pas de n-uplet dans la requête ou s'il y en a plusieurs.

Exemple

- Avec Sells(bar, beer, price), trouver les bars qui Sells Miller au pris où Joe's Bar Sells Bud.
- Deux requêtes:
  1. Trouver le prix de Bud chez Joe's bar
  2. Trouver les bars qui vendent Miller à ce même prix.

Requête + Sous-requête

```
SELECT bar
FROM Sells
WHERE beer = 'Miller' AND
    price = (SELECT price
             FROM Sells
             WHERE bar = 'Joe"s Bar'
               AND beer = 'Bud');
```

Le prix de Bud chez Joe's bar

L'opérateur IN

- <n-uplet> IN <relation> vaut TRUE ssi<n-uplet>appartient à <relation>
- <n-uplet> NOT IN <relation> le contraire
- IN: dans le WHERE
- <relation> est en général une sous-requête

Example

- Avec Beers(name, manf) et Likes(drinker, beer), trouver le nom et le fabriquant de toute Beers que Fred.

```
SELECT *
FROM Beers
WHERE name IN (SELECT beer
               FROM Likes
               WHERE drinker = 'Fred');
```

beers que Fred Likes

L'opérateur EXISTS

- EXISTS(<relation>) vaut TRUE ssi ssi <relation> is n'est pas vide
- NOT EXITS (<relation>) : le contraire
- EXISTS apparaît dans le WHERE
- Example: avec Beers(name, manf), trouver les bières qui sont les seules produites par leur manf

Exemple de requête avec EXISTS

```
SELECT name
FROM Beers b1
WHERE NOT EXISTS(
        SELECT *
        FROM Beers
        WHERE manf = b1.manf AND
              name <> b1.name);
```

Règle de portée: manf renvoie à la relation la plus proche dans l'imprication ayant cet attribut (ici, Beers juste au dessus)

Ens. des Beers différentes avec le même manf que b1

Symbole Différent en SQL

L'opérateur ANY

- *x* = ANY(<relation>) vaut TRUE ssi au moins un n-uplet de <relation> est x
- Même chose avec autres comparaisons:
  - Exemple: *x* >= ANY(<relation>) signifie que x est plus grand que l'un des éléments de <relation>
    Présuppose que le n-uplet x n'a qu'un composant/attribut

L'opérateur ALL

- De même *x* <> ALL(<relation>) vaut TRUE si x diffère de tout n-uplet de <relation>
- <> peut être remlacer par tout opérateur de.
- Example: *x* >= ALL(<relation>) x est plus grand que tous les éléments de relation Présuppose que <relation> n'a qu'un attribut.

Exemple

- Avec Sells(bar, beer, price), trouver la beer(s) qui est vendue le plus cher.

```
SELECT beer
FROM Sells
WHERE price >= ALL(
    SELECT price
    FROM Sells);
```

Le 1er price est plus grand que tous les price de Sells

Opérations ensemblistes Ensembles et multi-ensembles en SQL

Union, Intersection, et Différence

- Union, intersection, et différence:
  - (sous-requête) UNION (sous-requête)
  - (sous-requête) INTERSECT (sous-requête)
  - (sous-requête) EXCEPT (sous-requête)

Exemple

- Avec les relations Likes(drinker, beer), Sells(bar, beer, price) et Frequents(drinker, bar), trouver les drinkers et beers tels que:
  1. drinker Likes beer, et
  2. drinker Frequents au moins un bar qui Sells beer

Solution

drinker Frequents bar qui Sells beer

```
(SELECT * FROM Likes)
  INTERSECT
(SELECT drinker, beer
 FROM Sells, Frequents
 WHERE Frequents.bar = Sells.bar
);
```


SQL: ensembles ou multi-ensembles

- SELECT-FROM-WHERE par défaut: multi-ensembles
- UNION INTERSECTION EXCEPT: par défaut: ensembles
  - Les doublons sont éliminés

Raisons: Efficacité

- Multi-ensembles mieux pour projection (attributs du SELECT) et sélection (condition du WHERE) car on procède n-uplet par n-uplet
- Pour intersection et différence, il est plus eficace de trier d'abord et ensuite autant éliminer les doublons (coût ridicule après le tri)

Elimination Contrôlée des Doublons

- Pour imposer d'avoir un ensemble SELECT DISTINCT . . .
- Pour avoir les opérations multi-ensemblistes ajouter ALL après l'opération:
  . . . UNION ALL . . .

Exemple: DISTINCT

- Liste des prix à partir de Sells(bar, beer, price):

```
SELECT DISTINCT price
FROM Sells;
```

- Sans DISTINCT un prix p figurerait autant de fois qu'il y a de couples bar, beer avec le prix de vente p

Exemple: ALL

- Avec les relations Frequents(drinker, bar) et Likes(drinker, beer):

```
(SELECT drinker FROM Frequents)
     EXCEPT ALL
(SELECT drinker FROM Likes);
```

- Drinkers qui fréquentent plus de bars qu'ils ne Likes de beers -- parraissent autant de fois qu'il y a de différences entre ces deux nombres.

Jointures

Naturelle
Conditionnelle
Externe
…

Jointures

- En SQL beaucoup d'opérateurs de jointures
  Ils fonctionnent avec des multi-ensembles.
- Ces expressions peuvent s'utiliser comme requêtes ou dans une requête (dans le FROM ou le WHERE)

Produits et Jointures Naturelles

- Jointure naturelle:
  R NATURAL JOIN S;
- Produit:
  R CROSS JOIN S;
- Exemple:

```
Likes NATURAL JOIN Sells;
```

- Les relations peuvent être des requêtes entre parenthèses

Jointure Conditionnelle

- R JOIN S ON <condition> est une jointure conditionnelle
- Exemple avec Drinkers(name, addr) et Frequents(drinker, bar):

```
Drinkers JOIN Frequents ON
     name = drinker;
```

donne tous les (n, *a, d, b*) tels drinker de nom n habite à l'adresse *a* et frequents le bar *b*.

Jointures externes

- R OUTER JOIN S est le noyau d'une jointure externe, qui peut être modifié par:
  1. Option: NATURAL devant OUTER.
  2. Option: ON <condition> après JOIN.
  3. Option: LEFT, RIGHT, ou FULL après OUTER.
     - LEFT = accepte les n-uplets de R sans n-uplet de S qui corresponde (en utilisant NULL)
     - RIGHT = accepte les n-uplets de S sans n-uplets de R qui leur corresponde (en utilisant NULL)
     - FULL = défaut accepte les n-uplets de R et S même sans n-uplet correspondant dans l'autre relation.

Opérateurs d'agrégation

Opérations
Effet de NULL
Regroupement GROUP BY
Conditionnelles HAVING

Aggrégations

- SUM, AVG, COUNT, MIN, et MAX s'utilisent sur une colonne du SELECT
- COUNT(*) compte les n-uplets de la relation

Exemple: Aggrégation

- Avec Sells(bar, beer, price), trouver la moyenne du prix de 'Bud'

```
SELECT AVG(price)
FROM Sells
WHERE beer = 'Bud';
```


Elimination des Doublons dans les Aggrégations

- DISTINCT dans l'aggrégation supprime les doublons avant l'aggrégation.
- Exemple: trouver les différents prices de Bud:

```
SELECT COUNT(DISTINCT price)
FROM Sells
WHERE beer = 'Bud';
```


NULL: ignoré dans les aggrégation

- NULL n'intervient pas dans une somme une moyenne, etc. et n'est jamais un minimum ou un maximum.
- Mais s'il n'y a que des NULL dans une colonne, son aggrégation fait NULL

Exemple: l'effet de NULL

```
SELECT count(*)
FROM Sells
WHERE beer = 'Bud';
```

Le nombre de bars qui Sells Bud.

```
SELECT count(price)
FROM Sells
WHERE beer = 'Bud';
```

Le nombre de bars qui Sells Bud à un prix connu.

Regroupement

- SELECT-FROM-WHERE peut être suivi de GROUP BY et d'une liste d'attributs
- Le resultat du SELECT-FROM-WHERE est regroupé par valeur des attributs du GROUPING L'aggrégation est appliquée à chaque groupe.

Exemple: Group by

- Avec Sells(bar, beer, price), trouver le prix moyen par de chaque beer :

```
SELECT beer, AVG(price)
FROM Sells
GROUP BY beer;
```


Exemple: Group By

- Avec Sells(bar, beer, price) et Frequents(drinker, bar), trouver pour chaque drinker le prix moyen de Bud dans les bar qu'il fréquente :

```
SELECT drinker, AVG(price)
FROM Frequents, Sells
WHERE beer = 'Bud' AND
      Frequents.bar = Sells.bar
GROUP BY drinker;
```

Compute drinker-bar-price of Bud tuples first, then group by drinker.

Restriction sur le contenu de SELECT avec l'Aggrégation

- Si on utilise un opérateur d'aggrégation, tout élement du SELECT est soit:
  1. Aggrégé par un opérateur d'aggrégation,
  2. Un attribut du GROUP BY

Exemple de requête illicite

- Les bars vendant Bud le moins cher:

```
SELECT bar, MIN(price)
FROM Sells
WHERE beer = 'Bud';
```

Requête illicite!

- Bar pas aggrégé,
  et pas dans le GROUP BY non plus

HAVING

- HAVING <condition> peut suivre un GROUP BY.
- Effet: il supprime les groupes qui ne satisfont pas la condition.

Conditions sur HAVING

- Porte sur les relations du FROM.
- Renvoie aux attributs de ces relations, qui ont un sens pour un Groupe:
  1. Attribut du GROUP BY
  2. Attribut aggrégé

Exemple: HAVING

- Avec Sells(bar, beer, price) et Beers(name, manf),
  trouver le prix moyen des beers vendues dans au moins trois bars ou dont le manf est Pete

Solution

```
SELECT beer, AVG(price)
FROM Sells
GROUP BY beer
HAVING COUNT(bar) >= 3 OR
    beer IN (SELECT name
             FROM Beers
             WHERE manf = 'Pete''s');
```

Groupes de beers avec au moins trois bars non NULL qui la vende ou leur manf est Pete

manf: Pete

Modification des tables

Insertion
Suppression
Mise à jour

Modifications de BD

- A modification command does not return a result as a query does, but it changes the database in some way.
- Trois types de modifications:
  1. *Insert* insertion de n-uplets.
  2. *Delete* destruction de n-uplets
  3. *Update* modification des valeurs des n-uplets

Insertion

- Insertion d'un n-uplet:

```
INSERT INTO <relation>
VALUES ( <list of values> );
```

- Exemple: ajouter à Likes(drinker, beer) que Sally likes Bud.

```
INSERT INTO Likes
VALUES('Sally', 'Bud');
```


Attributs précisés par INSERT

- On ajoute à la relation les attributs.
- Pourquoi:
  1. On a oublié l'ordre des attributs.
  2. On n'a pas toutes les valeurs du n-uplets
     On veut que le système mette NULL ou la valeur par défaut.

Exemple avec attributs spécifiés

- Autre manière d'ajouter Sally likes Bud à Likes(drinker, beer):

```
INSERT INTO Likes(beer, drinker)
VALUES('Bud', 'Sally');
```


Insertion de plusieurs n-uplets

- On peut insérer le résulat d'une requête dans une relation:

  INSERT INTO <relation>
  (<sous-requête>);

Exemple: insertion d'une sous-requête

- avec Frequents(drinker, bar), ajouter à une nouvelle relation PotBuddies(name) tous les amis potentiels de Sally, ceux qui fréquente un bar que fréquente aussi Sally

Solution

L'autre drinker

Paires de n-uplets dont les 2 drinkers fréquentent un même bar L'un est Sally et l'autre non.

```
INSERT INTO PotBuddies
(SELECT d2.drinker
 FROM Frequents d1, Frequents d2
 WHERE d1.drinker = 'Sally' AND
   d2.drinker <> 'Sally' AND
   d1.bar = d2.bar
);
```


DELETE

- Destruction des n-uplets satisfaisant une condition:

  DELETE FROM <relation>
  WHERE <condition>;

Exemple: DELETE

- Suppressin de (Sally,Bud) de Likes(drinker, beer) :

```
DELETE FROM Likes
WHERE drinker = 'Sally' AND
    beer = 'Bud';
```


Exemple: suppresion de tous les n-uplets

- Vider Likes:

```
DELETE FROM Likes;
```

- WHERE pas nécessaire.

Exemple: Destruction de plusieurs n-uplets

- Suppresions dans Beers(name, manf) les beers don't le manf fabrique une autre beer.

```
DELETE FROM Beers b
WHERE EXISTS (
  SELECT name FROM Beers
  WHERE manf = b.manf AND
    name <> b.name);
```

Beers with the same manufacturer and a different name from the name of the beer represented by tuple b.

Fonctionnement de la suppression 1

- Supposition: Anheuser-Busch fabrique seulement Bud et Bud Lite.
- On trouve *b* = Bud en premier.
- La sous-requête n'est pas vide, à cause de Bud Lite on détruit Bud.
- Quand *b* est le n-uplet pour Bud Lite, le détruit-on aussi?

Fonctionnement de la suppression 2

- On détruit Bud Lite aussi
- Pourquoi? Suppression en 2 étapes:
  1. On marque dans la relation originale tous les n-uplets qui satisfont la condition du
  2. On supprime les n-uplets marqués.

UPDATE mise à jour

- Pour changer certains attributs de certains n-uplets:

  UPDATE <relation>

  SET <listes d'instanciations d' attributs>

  WHERE <condition sur le n-uplet>;

Exemple: UPDATE

- Changement du téléphone de Fred qui devient 555-1212:

```
UPDATE Drinkers
SET phone = '555-1212'
WHERE name = 'Fred';
```


Exemple: Update global

- Ramener à 4€ le prix de beer s'il est plus élevé:

```
UPDATE Sells
SET price = 4.00
WHERE price > 4.00;
```


Définition d'un Schéma Relationnel

Création
Types en SQL
Déclaration des clefs
Modification du schéma

Définition d'un Schéma de BD

- Déclaration des relations ("tables") de la BD
- On peut déclarer beaucoup d'autres choses dans le schéma --- on verra plus tard.

Déclarer une Relation

- Le plus simple:

  CREATE TABLE <nom> (

  <liste d'éléments>

  );
- On peut supprimer une table:

  DROP TABLE <nom>;

Elts d'une Déclaration d'une Table

- Paire attribut:Types
- Types les plus courants:
  - INT ou INTEGER (synonymes).
  - REAL ou FLOAT (synonymes).
  - CHAR(*n*) = chaîne de *n* caractères.
  - VARCHAR(*n*) =chaîne de moins de *n* caractères.

Exemple: CREATE TABLE

```
CREATE TABLE Sells (
    bar     CHAR(20),
    beer        VARCHAR(20),
    price       REAL
);
```

Dates et Heures

- DATE et TIME sont des types SQL.
- La forme d'une date est:
  DATE 'aaaa-mm-jj'
  - Exemple: DATE '2002-09-30' pour 30 Septembre 2002.

Heures

- Forme générale:
  TIME 'hh:mm:ss'

Éventuellement, décimales en dessous de la seconde.

- Example: TIME '15:30:02.5' =deux seconde et demi après 15 heures trente

Déclaration des Clefs

- Un attribut ou une liste d'attributs peut être déclaré PRIMARY KEY ou UNIQUE.
- Cela signifie que ce(s) attribut(s) détermine les autres.
- On verra d'autres distinctions plus loin.

Déclaration des clefs à un attribut

- Ecrire PRIMARY KEY ou UNIQUE après le type dans la déclaration de l'attribut
- Exemple:

```
CREATE TABLE Beers (
    name    CHAR(20) UNIQUE,
    manf    CHAR(20)
);
```


Clefs Multi-Attributs

- Déclaration qui suit celle des attributs dans CREATE TABLE
- Peut s'utiliser pour un seul attribut aussi.
- Cette forme est nécessaire si la clef comporte plusieurs attributs

Exemple: Clef Multi-Attributs

- bar et beer est la clef de Sells:

```
CREATE TABLE Sells (
    bar     CHAR(20),
    beer    VARCHAR(20),
    price   REAL,
    PRIMARY KEY (bar, beer)
);
```


PRIMARY KEY ou UNIQUE ?

- Le standard SQL standard ne précise pas la distinction entre PRIMARY KEY et UNIQUE.
  - Par exemple le SGBD peut créer un index pour PRIMARY KEY et pas pour UNIQUE.

Distinctions Obligatoires

- SQL demande que
  1. Une seule PRIMARY KEY pour une relation, mais plusieurs attributs UNIQUE.
  2. Un attribut d'une PRIMARY KEY ne peut jamais valoir NULL. Par contre un attribut UNIQUE peut valoir NULL et même pour plusieurs n-uplets.

Autres déclaration pour les Attributs

1. NOT NULL
   attribut qui ne vaut jamais NULL.
2. DEFAULT <value>précise la valeur par défaut

Exemple: Valeurs par Défaut

```
CREATE TABLE Drinkers (
   name CHAR(30) PRIMARY KEY,
   addr CHAR(50)
       DEFAULT '123 Sesame St.',
   phone CHAR(16)
);
```


Valeurs par défaut -- 1

- Sally fait partie de drinker, mais on ignore son adresse et téléphone.
- INSERT avec une liste d'attributs incomplète fonctionne:

```
INSERT INTO Drinkers(name)
   VALUES('Sally');
```


Valeurs par défaut -- 2

- Quel n-uplet dans Drinkers?

name	addr	phone
'Sally'	'123 Sesame St'	NULL

- SI téléphone NOT NULL, l'insertion aurait été refusée

Ajout d'Attributs

- Ajouter un attribut (une "colonne"):
  ALTER TABLE <name> ADD
  <attribute declaration>;
- Exemple:

```
ALTER TABLE Bars ADD
phone CHAR(16)DEFAULT 'unlisted';
```


Suppressions d'Attributs

- Suppression d'un attribut dans une table:
  ALTER TABLE <name>
  DROP <attribute>;
- Exemple: suppression de la license:

```
ALTER TABLE Bars DROP license;
```


Suppressions d'Attributs

- Suppression d'un attribut dans une table:
  ALTER TABLE <name>
  DROP <attribute>;
- Exemple: suppression de la license:

```
ALTER TABLE Bars DROP license;
```


VUES

(tables aux possibilités restreintes)

Vues

- Vue: table virtuelle définie à partir des autres tables et vues
- Déclaration:
  CREATE VIEW <name> AS <query>;
- Les autres tables sont dites tables de *base.*

Exemple: Définition d'une vue

- CanDrink(drinker, beer) est une vue avec les couples drinker-beer tel que drinker fréquente au moins un bar qui vend beer:

```
CREATE VIEW CanDrink AS
   SELECT drinker, beer
   FROM Frequents, Sells
   WHERE Frequents.bar = Sells.bar;
```

Exemple: accès à une vue

- Une vue s'interroge comme une table.
  - On peut modifier une vue si cela a un sens pour les tables sous-jacentes.
- Exemple:

```
SELECT beer FROM CanDrink
WHERE drinker = 'Sally';
```


Comment la vue est-elle utilisée?

- Le SGBD interprète la requête comme si la vue était une table de base. Il en fait une formule de l'algèbre relationnelle dans laquelle intervient la vue V.
- Cette table V est remplacée par sa définition en algèbre relationnelle.

Exemple: Expansion d'une Vue

Optimisation du SGBD

- La requête obtenue par remplacement de la vue par sa définition n'est pas optimale.
- Optimisations:
  1. Faire descendre les sélections près des tables de base
  2. Eliminer les projections inutiles.

Exemple: Optimisation