

SQL

SQL: historique

- SQL est basé sur l'algèbre et le calcul relationnels
- Il a été intégré à SQL/DS, DB2, puis Oracle, Ingres, ...
- Il existe trois normes :
 - SQL1 (1986) version minimale
 - SQL1 (1989) + intégrité
 - SQL2 (1992) langage complet
- Une version 3 étendue à l'objet vient être proposée.
- La plupart des systèmes supportent SQL1 complet

Rappel algèbre

- Notion de relation
 - Schéma : Nom de relation + ensemble d'attributs
 - Extension : ensemble de tuples (n-uplets)
- Cinq opérations de base
 - Π , Projection
 - σ , Restriction
 - \times , Produit cartésien
 - \cup , Union
 - $-$, Différence
- Autres opérations déduites
 - \bowtie , Jointure
 - \cap , Intersection
 - \div , Division
 - ρ , Renommage

Rappel calcul relationnel ou calcul à variable n-uplet

- Langage formel basé sur la logique des prédicats du premier ordre
- Requêtes en calcul relationnel $\{t \mid \phi(t)\}$
 - t désigne une variable n-uplet et $\phi(t)$ une formule bien formée
- Formules, composées à partir de :
 - termes atomiques : variables t , noms de relations et constantes
 - opérateurs $\in, =, <, >, \dots$
 - connecteurs \wedge, \vee et negation
 - Quantificateurs \exists et \forall
- SQL
 - Version commerciale du calcul relationnel

Exemple de requête

- Schéma :

Viticulteurs (NVT, Nom, Prénom, Ville, Région)

Vins (NV, Cru, Millésime, Degré, NVT, Prix)

Buveurs (NB, Nom, Prénom, Ville)

Abus(NV, NB, Date, Qté)

Quels sont les viticulteurs qui ont produit au moins un vin de 1983 (nom et cru) ?

$$\{t:cru, nom \mid \exists v \exists w (v \in Vins \wedge w \in Viticulteurs \wedge \\ t(cru) = v(cru) \wedge t(nom) = w(nom) \wedge \\ w(nvt) = v(nvt) \wedge v(mill) = 1983 \}$$

Quels sont les viticulteurs qui ont produit au moins un vin de 1983 (nom et cru) ?

$$\{t:cru, nom \mid \exists v \exists w (v \in Vins \wedge w \in Viticulteurs \wedge t(cru) = v(cru) \wedge t(nom)=w(nom) \wedge w(nvt) = v(nvt) \wedge v(mill) = 1983)\}$$

```
SELECT W.nom, V.cru
FROM Viticulteurs W, Vins V
WHERE V.nvt = W.nvt and V.mill = 1983;
```

Algèbre relationnelle?

SQL1 : Composantes de base

Langage de manipulation de données (DML)

SELECT OPEN
INSERT FETCH
UPDATE CLOSE
DELETE

Langage de définition de données (DDL)

CREATE TABLE
CREATE VIEW
DROP TABLE
DROP VIEW

Langage de contrôle de données

GRANT et REVOKE
BEGIN et END TRANSACTION
COMMIT et ROLLBACK

Types de données en SQL

Types atomiques:

- Chaînes de caractères: CHAR(20), VARCHAR(50)
ex. 'Paris '
- Nombres: INT, BIGINT, SMALLINT, FLOAT
- Autres: MONEY, DATETIME, ...

Enregistrements (tuple)

- contiens des attributs atomiques

Table (relation)

- ensemble de tuples

Details

SQL n'est pas « sensible à la casse »

- SELECT = Select = select
- Region = region
- Sauf pour les chaînes de caractères:
 'Paris' ≠ 'paris'

Select (forme de base) :

principe et sémantique

- Une requête SQL est une description dans une syntaxe propre à ce langage d'éléments contenus dans une base de données

- Format simplifié :

SELECT [**DISTINCT**] $\text{Attribut}_1, \text{Attribut}_2, \dots \text{Attribut}_p$

FROM $\text{Relation}_1, \text{Relation}_2, \dots \text{Relation}_k$

WHERE Conditions Q

[**ORDER BY ASC/DESC**]

[{**UNION** | **INTERSECT** | **EXCEPT** } **SELECT** ...]

- Condition:

- arithmétique ($=, <, >, \neq, \geq, \leq$)
- textuelle (LIKE: exp. réguliers avec caractères spéciaux ' %', ' _')
- sur intervalle (BETWEEN)
- sur liste (IN)

2 sémantiques (interprétations) équivalentes

```
SELECT a1, a2, ..., ap  
FROM   R1 AS x1, R2 AS x2, ..., Rk AS xk  
WHERE  Conditions ;
```

$$\pi_{\text{Attribut1,Attribut2,...,Attributp}} (\sigma_Q (R_1 \times R_2 \times \dots \times R_k))$$

```
Résultat = {}  
for x1 in R1 do  
  for x2 in R2 do  
    .....  
    for xk in Rk do  
      if Conditions  
        then Résultat = Résultat ∪ {(a1,...,ap)}  
return Résultat
```

Quels sont les viticulteurs qui ont produit au moins un vin de 1983 (nom et cru) ?

$$\{t:cru, nom \mid \exists v \exists w (v \in Vins \wedge w \in Viticulteurs \wedge t(cru) = v(cru) \wedge t(nom)=w(nom) \wedge w(nvt) = v(nvt) \wedge v(mill) = 1983)\}$$

```
SELECT W.nom, V.cru
FROM Viticulteurs W, Vins V
WHERE V.nvt = W.nvt and V.mill = 1983;
```

Algèbre relationnelle?

Select : exemples de requêtes (1)

- Liste des crus sans doublons, ordonné.

```
SELECT DISTINCT Cru
FROM Vins
ORDER BY Cru ASC;
```

- Noms des buveurs ayant bus du Beaujolais 87 ou 88.

```
SELECT DISTINCT Nom
FROM      Buveurs B, Vins V, Abus
WHERE B.nb = Abus.nb
AND      Abus.nv = V.nv
AND      Cru LIKE '%Beaujolais%'
AND      millésime IN (1987, 1988) ;
```

Select : exemples de requêtes (2)

- Exemple d'utilisation du OR :
 - Noms et prénoms des buveurs de vins de degré inconnu ou compris entre 11 et 13.

SELECT	Nom, Prénom
FROM	Buveurs B, Vins V, Abus A
WHERE	B.nb = A.nb AND A.nv = V.nv
AND	(degre BETWEEN 11 AND 13 OR degre IS NULL);

Union / Intersection

```
SELECT nvt  
FROM Viticulteurs  
WHERE region= 'Bourgogne' OR/ AND nom=  
    'Dupont' ;
```

```
SELECT nvt  
FROM Viticulteurs  
WHERE region= 'Bourgogne'  
UNION/INTERSECT  
SELECT nvt  
FROM Viticulteurs  
WHERE nom= 'Dupont' ;
```

Difference

```
SELECT nvt  
FROM Viticulteurs  
WHERE region= 'Bourgogne' AND nom <> 'Dupont' ;
```

```
SELECT nvt  
FROM Viticulteurs  
WHERE region= 'Bourgogne'  
EXCEPT/MINUS  
SELECT nvt  
FROM Viticulteurs  
WHERE nom= 'Dupont' ;
```


Imbrication (sous-requêtes)

- Une requête SQL « imbriquée » peut apparaître dans:
 - la partie SELECT (en général à éviter)
 - la partie FROM (en général à éviter)
 - la partie WHERE **parfois incontournable**

Imbrication dans le SELECT

Pour chaque vin la région ou il est produit.

```
SELECT V.nv, (SELECT DISTINCT W.region  
              FROM Viticulteurs W  
              WHERE W.nvt=V.nvt)  
FROM Vins V;
```



Corrélation

The diagram consists of a light green rectangular box with a thin brown border containing the word 'Corrélation'. Two green arrows originate from the top corners of this box. One arrow points to the 'V.nvt' attribute in the 'WHERE W.nvt=V.nvt' clause of the subquery. The other arrow points to the 'FROM Vins V;' clause of the outer query.

Et si un vin est produit dans plusieurs régions ...?

Sans imbrication?

Imbrication dans le SELECT

Pour chaque vin la région ou il est produit.

```
SELECT V.nv, (SELECT DISTINCT W.region  
              FROM Viticulteurs W  
              WHERE W.nvt=V.nvt)  
FROM Vins V;
```

Et si un vin est produit dans plusieurs regions ...?

```
Sans imbrication?  SELECT V.nv, W.region  
                   FROM Viticulteurs W, Vins V  
                   WHERE W.nvt=V.nvt;
```

Imbrication dans le SELECT

Pour chaque vin le nombre de régions ou il est produit.

```
SELECT DISTINCT V.nv, (SELECT count(W.region)
                        FROM Viticulteurs W
                        WHERE W.nvt=V.nvt)
FROM Vins V;
```

Sans imbrication? On va voir le GROUP BY...

Imbrication dans le FROM

```
SELECT *  
FROM (SELECT *  
      FROM Vins V  
      WHERE V.millesime = 1983) as temp  
WHERE temp.prix < 15;
```

Sans imbrication?

Imbrication dans le WHERE

- Possibilité de blocs imbriqués par :
IN, EXISTS, NOT EXISTS, ALL, ANY

```
SELECT nom  
FROM Viticulteurs W  
WHERE W.nvt IN (SELECT V.nvt  
                FROM Vins V  
                WHERE V.mill = 1983);
```

```
SELECT nom  
FROM Viticulteurs W  
WHERE W.nvt NOT IN (SELECT V.nvt  
                   FROM Vins V  
                   WHERE V.mill = 1983);
```

Imbrication: EXISTS / NOT EXISTS

- Noms des crus bus par au moins un buveur.

```
SELECT DISTINCT cru
FROM Vins V
WHERE EXISTS ( SELECT *
                FROM Abus A
                WHERE A.nv = V.nv );
```

Imbrication: ANY

```
SELECT DISTINCT region
FROM Viticulteurs W
WHERE 10 < ANY (SELECT degre
                FROM Vins V
                WHERE V.nvt=W.nvt
                );
```

Sans imbrication?

Imbrication: ANY

```
SELECT DISTINCT region
FROM Viticulteurs W
WHERE 10 < ANY (SELECT degre
                 FROM Vins V
                 WHERE V.nvt=W.nvt
                 );
```

```
SELECT DISTINCT region
FROM Viticulteurs W, Vins V
WHERE 10 < degre and V.nvt=W.nvt;
```

Les quantificateurs existentiels (ANY, EXISTS, IN) -> requête sans imbrication? Facile!

Imbrication: ALL

```
SELECT *  
FROM Vins V  
WHERE V.degre > ALL (SELECT degre  
                      FROM Vins W  
                      WHERE  
                      W.millesime = 1983);
```

Requêtes monotones

Soit Q une requête sur les relations R, S, T, \dots ; soit $Q(R, S, T, \dots)$ le résultat de Q .

Définition Q est **monotone** ssi :

$$\forall R \subseteq R', S \subseteq S', \dots \Rightarrow Q(R, S, \dots) \subseteq Q(R', S', \dots)$$

Théorème Toutes les requêtes select-from-where sont monotones.

La requête précédente n'est pas monotone.

Quantificateurs universels (NOT IN, NOT EXISTS, ALL) ->
l'imbrication ne peut pas être évitée.

Division

Les buveurs qui ont bu tous les vins.

(1)

```
SELECT B.nb
FROM Buveurs B
WHERE NOT EXISTS
    ((SELECT V.nv
      FROM Vins V)
     EXCEPT
     (SELECT A.nv
      FROM Abus A
      WHERE A.nb=B.nb))
```

(2) SELECT B.nb
FROM Buveurs B
WHERE NOT EXISTS (SELECT V.nv
FROM Vins V

Buveurs nb tels que ... WHERE NOT EXISTS (SELECT A.nv
FROM Abus A
WHERE A.nv=V.nv
AND A.nb=B.nb))

aucun vin nv sans ...

tuple « nb a bu nv »

Fonctions et agrégat

- **Fonction**
 - Fonction de calcul en ligne appliquée sur un ou plusieurs attributs
 - Exemple : $\text{DEGRE} * \text{QUANTITE} / 100$
- **Agrégat**
 - MIN, MAX, AVG, SUM, COUNT
 - A l'exception de COUNT, les agrégats s'appliquent à un seul attributs
 - par défaut, COUNT s'applique au doublons

Exemple: agrégat

```
SELECT AVG(degre*millesime)  
FROM Vins;
```

```
SELECT count(millesime)  
FROM Vins;
```

```
SELECT count(*)  
FROM Vins;
```

COUNT(millesime) ne s'applique pas pour millesime=NULL

```
SELECT count(DISTINCT degre)  
FROM Vins;
```

GROUP BY

SELECT liste de projection
FROM liste de tables
[WHERE critère de jointure AND critère de restriction]
[GROUP BY A1, A2, ..., Ak (attributs de partitionnement)]

Partitionnement horizontal d'une relation selon les valeurs d'un groupe d'attributs, suivi d'un regroupement par une fonction de calcul en colonne (Sum, Min, Max, Avg, Count)

Calculer le degré moyen pour chaque cru.

```
SELECT CRU, AVG(DEGRE) as moyenne  
FROM Vins  
GROUP BY CRU;
```

La liste de projection peut contenir

1. n'importe quels agrégats
2. des attributs parmi A1, ..., Ak, mais pas d'autres attributs!

Sémantique du GROUP BY

- calcul des parties FROM et WHERE
- partitionnement horizontal (grouping) selon les attributs du GROUPBY
- calcul de la partie SELECT: attributs qui apparaissent dans la liste GROUPBY et agrégats.

Observation: chaque groupe sera non-vide.

```
SELECT R.A, count(*)  
FROM R  
WHERE R.B < 55  
GROUP BY R.A;
```




Toujours > 0!

Exemples GROUP BY / agrégats (2)


Vins	CRU	MILL	DEGRE	QUANTITE
	CHABLIS	1977	10.9	100
	CHABLIS	1987	11.9	250
	VOLNAY	1977	10.8	400
	VOLNAY	1986	11.2	300
	MEDOC	1985	11.2	200

**SELECT MOY(DEGRE)
FROM Vins;**



AVG	DEGRE
	11.2

**SELECT CRU, SUM(QUANTITE)
FROM Vins
GROUP BY CRU;**



SUM	CRU	QUANTITE
	CHABLIS	350
	VOLNAY	700
	MEDOC	200

GROUP BY vs. imbrication

```
SELECT cru, AVG(degre) as moyenne  
FROM Vins  
WHERE millesime < 2000  
GROUP BY cru;
```

```
SELECT DISTINCT cru, (SELECT AVG(degre)  
                        FROM Vins Y  
                        WHERE Y.nv=X.nv AND  
                        Y.millesime < 2000) AS moyenne  
FROM Vins X  
WHERE X.millesime < 2000;
```

Forme plus générale du GROUP BY

SELECT	liste de projection
FROM	liste de tables
[WHERE	critère de jointure AND critère de restriction]
[GROUP BY	attributs de partitionnement]
[HAVING	critère de restriction sur les partitions]

HAVING : conditions sur des agrégats.

- Calculer le degré moyen et le degré minimum pour tous les crus de 94 dont le degré minimum est supérieur à 12.

```
SELECT cru, AVG(degre), MIN(degre)
FROM Vins
WHERE millesime = 1994
GROUP BY cru
HAVING MIN(degre) > 12;
```

Sémantique avec HAVING

```
SELECT S  
FROM R1,...,Rn  
WHERE Q1  
GROUP BY A1,...,Ak  
HAVING Q2;
```

- Calculer les parties FROM et WHERE (appliquer les conditions Q1)
- Grouper selon A1, ..., Ak
- Appliquer les conditions Q2 sur chaque groupe
- Calculer les agrégats de la liste S et retourner le attributs/agrégats de S

NULLs en SQL

- NULL – interprétations:
 - La valeur n'existe pas
 - La valeur existe mais elle est inconnue
 - La valeur ne s'applique pas
 - etc.
- Dans la spécification du schéma, on peut indiquer si un attribut peut être NULL ou pas.
- Comment gérer les NULLs?

NULLs

- Si $x = \text{NULL}$ alors $3 * (3 - x) \neq \text{NULL}$
- Si $x = \text{NULL}$ alors $x = \text{“Alice”}$ est UNKNOWN
- En SQL nous avons 3 valeurs booléennes
 - FALSE = 0
 - UNKNOWN = 0.5
 - TRUE = 1

NULLs

- $C1 \text{ AND } C2 = \min(C1, C2)$
- $C1 \text{ OR } C2 = \max(C1, C2)$
- $\text{NOT } C1 = 1 - C1$

```
SELECT *  
FROM Personnes  
WHERE (age < 30) AND  
      (taille > 180 OR poids > 80);
```

age=20
taille=NULL
poids=100

En SQL: on garde un tuple uniquement si conditions = TRUE

NULLs

Résultat inattendu:

```
SELECT *  
FROM   Personnes  
WHERE  age < 30 OR age >= 30;
```

Certaines personnes ne sont pas dans le résultat!

NULLs

Syntaxe SQL pour NULLs:

- x IS NULL
- x IS NOT NULL

```
SELECT *  
FROM   Personnes  
WHERE  age < 30 OR age >= 30 OR age IS NULL;
```

Outerjoin (jointure externe)

Les jointure explicite en SQL = “inner joins”:

produit(nom, catégorie)
achat(prodnom, magasin)

```
SELECT produit.nom, achat.magasin  
FROM    produit JOIN achat ON  
        produit.nom = achat.prodnom;
```

Equivalent:

```
SELECT produit.nom, achat.magasin  
FROM    produit, achat  
WHERE   produit.nom = achat.prodnom;
```

Mais les produits qui n'ont pas été achetés sont perdus!

Outerjoin

Left outer join (jointure externe gauche) en SQL:

produit(nom, catégorie)
achat(prodnom, magasin)

```
SELECT produit.nom, achat.magasin  
FROM    produit LEFT OUTER JOIN achat ON  
        produit.nom = achat.prodnom;
```

produit

nom	catégorie
DVD	Visio
Camera	Photo
Portable	Info

achat

Prodnom	magasin
DVD	Fnac
Camera	Surcouf
Camera	Fnac

nom	magasin
DVD	Fnac
Camera	Surcouf
Camera	Fnac
Portable	NULL

Application

Calculer pour chaque produit, le nombre total d'achats en septembre

produit(nom, catégorie)
achat(prodnom, mois, magasin)

```
SELECT produit.nom, count(*)  
FROM    produit, achat  
WHERE   produit.nom = achat.prodnom  
        and achat.mois = 'Septembre'  
GROUP BY produit.nom;
```

Où est le problème ?

Solution

on doit utiliser l'attribut (égal à NULL) pour obtenir 0

```
SELECT produit.nom, count(magasin)
FROM   produit LEFT OUTER JOIN achat ON
        produit.nom = achat.prodnom
WHERE  achat.mois = 'Septembre'
GROUP BY produit.nom;
```

Maintenant on obtient aussi les produits sans achats.

Outer Joins

- Left outer join:
 - Inclut les tuples gauches même s' il n' y a pas de correspondant
- Right outer join:
 - Inclut les tuples gauches même s' il n' y a pas de correspondant
- Full outer join:
 - Inclut les tuples gauches et droites même s' il n' y a pas de correspondant

Create table

```
CREATE TABLE Vins (  
    NV          NUMBER(6) UNIQUE,  
    CRU         CHAR(10) ,  
    ANNEE       NUMBER(4),  
    DEGRE       NUMBER(5, 2) ,  
    NVT         NUMBER(3),  
    PRIX        NUMBER(5, 2) );
```

```
DROP TABLE Vins;
```

```
ALTER TABLE Vins ADD COLUMN type : char;
```


Create view

```
CREATE VIEW Gros_Buveurs AS
  SELECT      nb, nom, prenom,
  FROM        Buveurs, Abus
  WHERE       buveurs.nb = Abus.nb
  AND         Abus.quantite > 100;
```

```
DROP VIEW ...
```

Insert

INSERT INTO nom de relation [(attribut [, attribut] ...)]
VALUES (valeur [, valeur] ...) | spécification de requête

Valeurs manquants sont mises à NULL

On peut ne pas donner les noms des attributs si dans l'ordre

- Exemples

```
INSERT INTO Vins (nv, cru, millesime)  
VALUES (112, "JULIENAS", NULL);
```

```
INSERT INTO Buveurs (nb, nom, prenom)  
SELECT nvt, nom, prenom  
FROM Viticulteurs  
WHERE ville LIKE '%DIJON% ';
```

Update

UPDATE nom de relation
SET attribut = expression de valeur | NULL
[attribut = expression de valeur | NULL] ...
[WHERE critère de recherche]

- Exemple

```
UPDATE Abus
SET qte = qte * 1.1
WHERE Abus.nv IN
  (SELECT nv
   FROM Vins
   WHERE cru = 'VOLNAY' AND millesime = 1990;
```

Delete

DELETE FROM nom de relation
[WHERE critère de recherche]

Exemple:

```
DELETE FROM Abus
WHERE nv IN
  SELECT nv
  FROM Vins
  WHERE degre IS NULL;
```

Question: peut-on effacer une seule occurrence d' un tuple qui apparaît plusieurs fois dans une relation?

Contraintes d'intégrité en SQL

Types:

- Clés primaires/secondaires
- Contraintes sur les valeurs d'un attribut
- Contraintes sur un tuple
- Contraintes globale: assertions

Clés

- Clé primaire et contrainte référentielle

CREATE TABLE Vins

(NV NUMBER(6) PRIMARY KEY,

CRU CHAR(10),

ANNEE NUMBER(4),

DEGRE NUMBER(5,2) ,

NVT NUMBER(3) REFERENCES VITICULTEURS,

PRIX NUMBER(5,2) DEFAULT 40)

Référence la clé primaire de VITICULTEURS

Clés

- Clé primaire, contrainte référentielle, unicité

```
CREATE TABLE Vins
( NV NUMBER(6),
  CRU CHAR(10),
  ANNEE NUMBER(4),
  DEGRE NUMBER(5,2) ,
  NVT NUMBER(3),
  PRIX NUMBER(5,2) DEFAULT 40,
  PRIMARY KEY(NV,CRU),
  UNIQUE (NV, ANNEE),
  FOREIGN KEY (nvt)
  REFERENCES Viticulteurs(nvt)
)
```

On peut avoir un seul PRIMARY KEY, mais plusieurs UNIQUE

Contraintes sur un attribut

- VALEURS PAR DEFAULT

```
CREATE TABLE Vins  
( NV NUMBER(6) UNIQUE,  
  CRU CHAR(10),  
  ANNEE NUMBER(4) NOT NULL,  
  DEGRE NUMBER(5,2) ,  
  NVT NUMBER(3),  
  PRIX NUMBER(5,2) DEFAULT 40 )
```

- CONTRAINTES DE DOMAINES

```
SALAIRE NUMBER(8, 2) CHECK BETWEEN 6000 AND 100000
```


Norme actuelle (SQL2 - 92)

Trois niveaux :

- Entry SQL2

SQL89 + manques

- Intermediate SQL2

Compléments relationnels

- Full SQL2

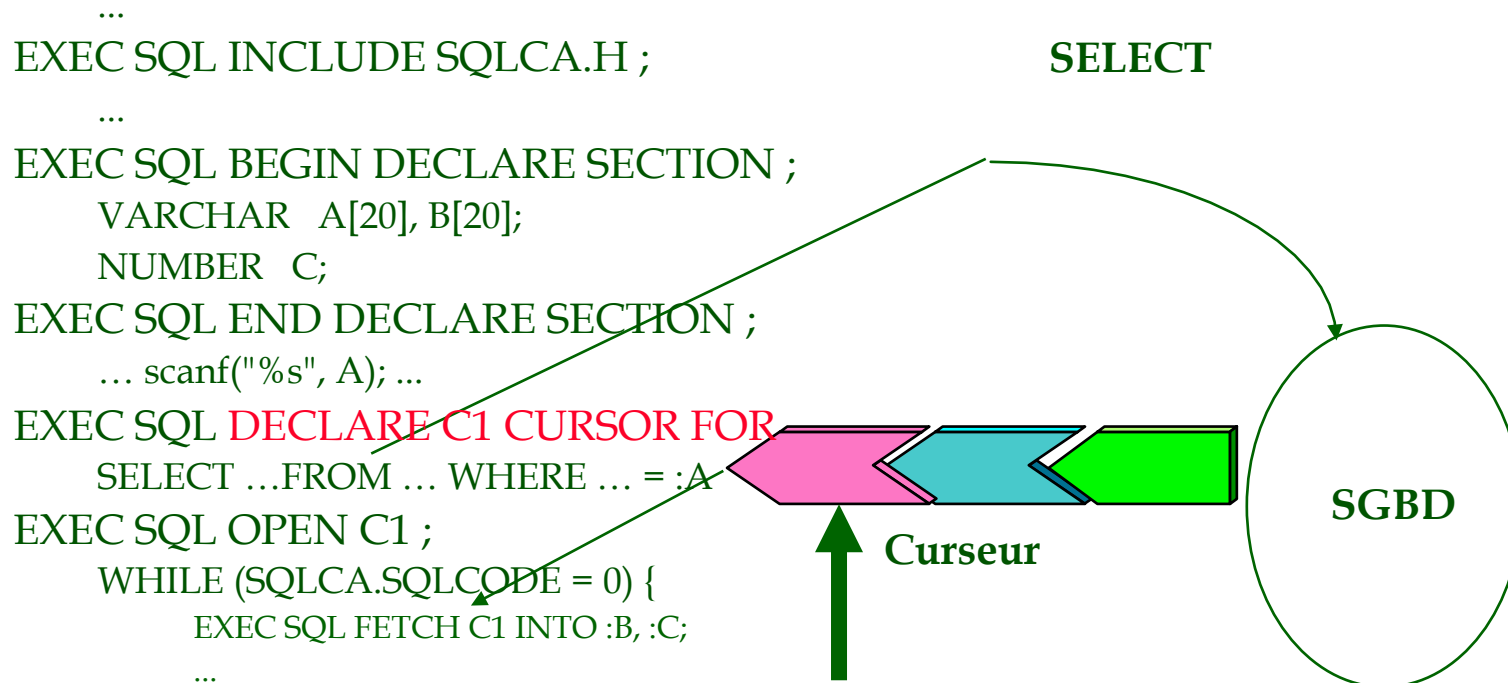
Gadgets en plus

Solutions BD pour le développement d'applications

- Embedded SQL
- Procédures stockées (stored procedures)
- Déclencheurs (triggers)

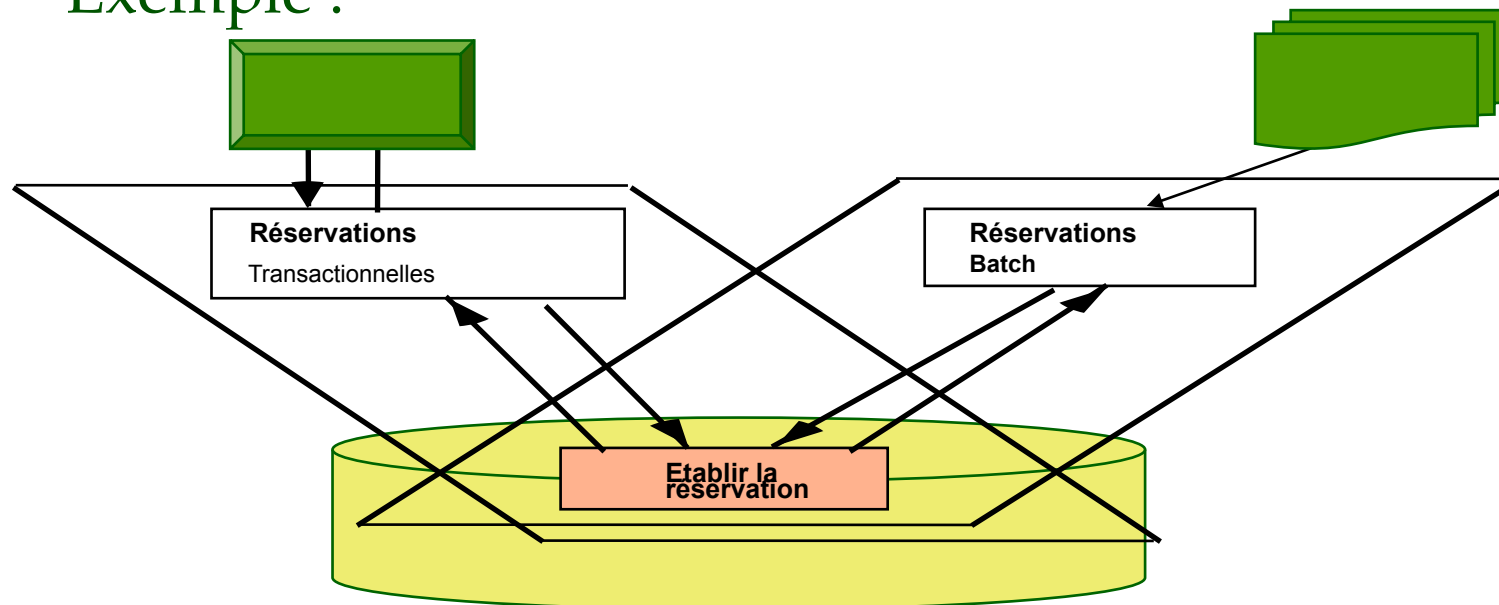
ESQL : SQL dans un langage de programmation

- Intégration de deux systèmes de types
 - utilisation d'un pré-compilateur
- Passage du traitement ensembliste au traitement tuple à tuple
 - utilisation de curseurs et Fetch
- Exemple de programme en C/SQL



Procédure stockée

- Procédure écrite en L3G/SQL ou L4G/SQL définie au niveau du schéma de la base et stockée dedans.
- Avantages :
 - partitionnement des traitements entre client et serveur
 - limiter le trafic sur le réseau, partager des procédures
 - Rapidité d'exécution
- Exemple :



Déclencheur (Trigger)

- Action base de données déclenchée suite à l'apparition d'un événement particulier
- Forme :
 - {BEFORE | AFTER} <événement> THEN <action>
 - Un événement peut être :
 - une opération sur une table (m-à-j)
 - un événement externe (heure, appel, etc.)
 - Une action peut être :
 - une requête BD (mise à jour)
 - un abort de transaction
 - l'appel à une procédure stockée

Déclencheur avec condition (Règle)

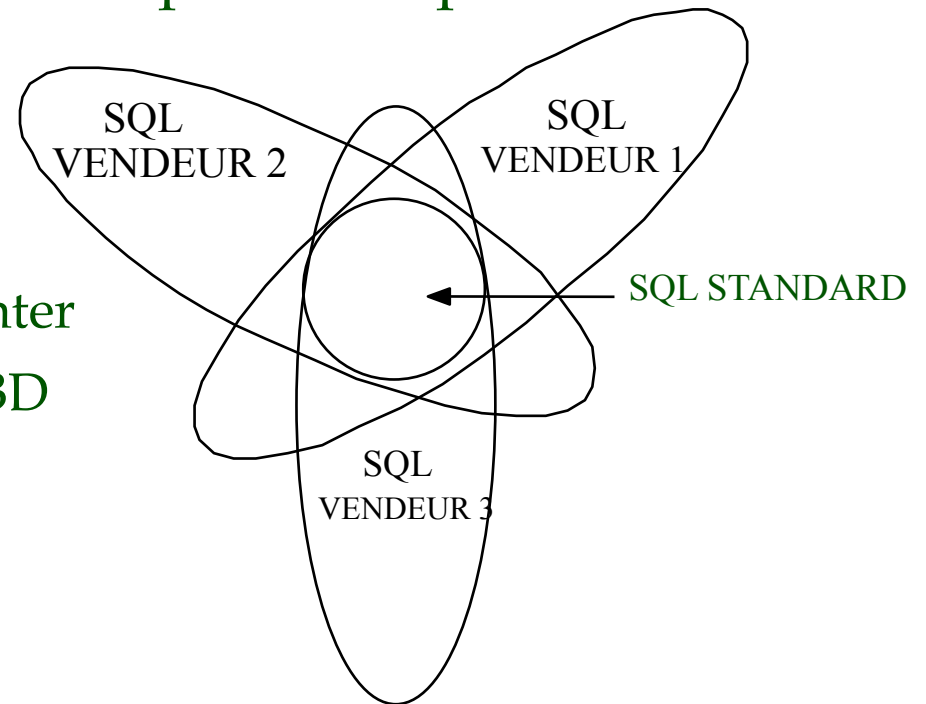
- Il est possible d'ajouter une condition au déclenchement de l'action
 - Une condition est une qualification portant sur la base.
- Exemples :

```
BEFORE      UPDATE FROM EMPLOYE  
IF          SALAIRE > 100.000  
THEN ABORT TRANSACTION
```

Conclusion

- Un standard de plus en plus complet et de plus en plus suivi

- Approximations et imitations incomplètes
- Une référence pour implémenter et utiliser chaque aspect des BD



- SQL réussira-t-il à bien intégrer l'objet ?

Exemple : Bars à Bière (schéma Entité/Association)

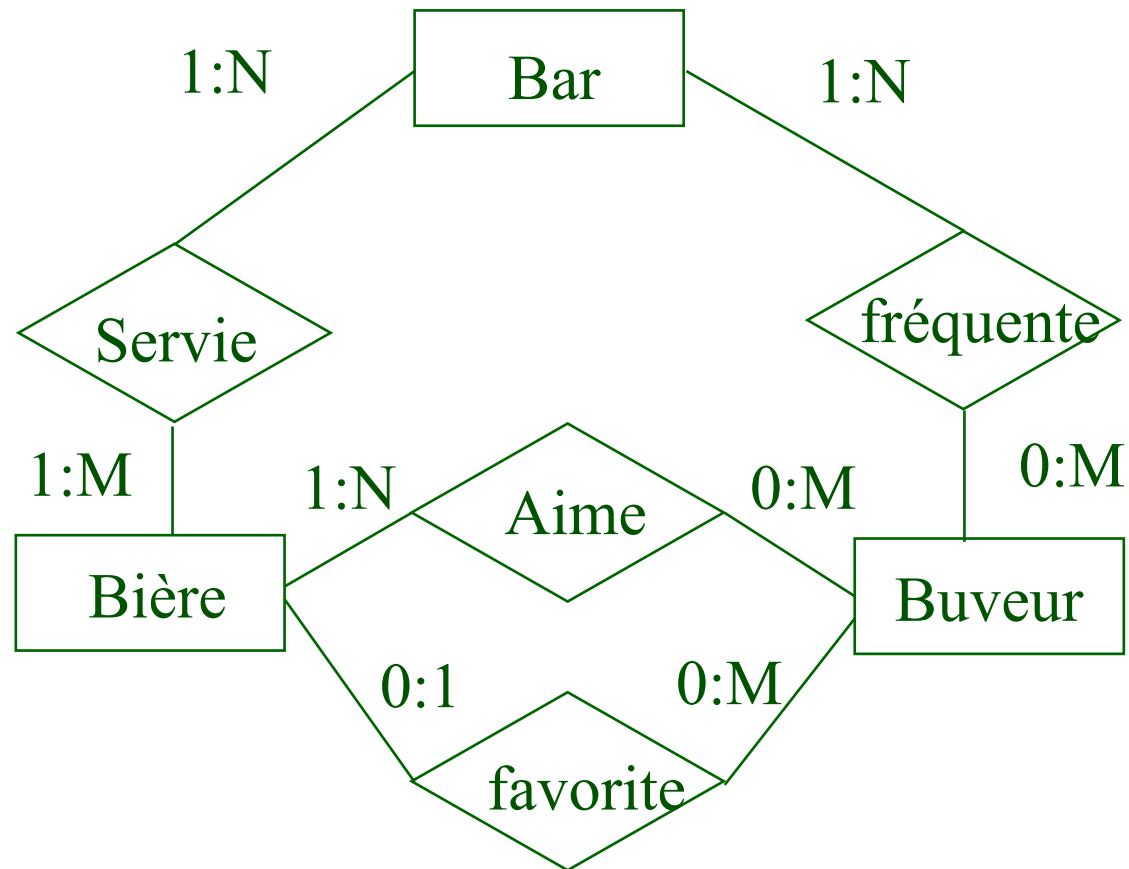


Schéma relationnel

- Aimer (bière, buveur)
- Servir (bière, bar)
- Fréquenter (buveur, bar)
- Bar (nom, ville, rue)
- Bière (nom, type, degré)
- Buveur(nom, prénom, favorite)

Requêtes

- Q1 - Les bars qui servent une bière appréciée par 'Dupont'
- Q2 - Les buveurs qui vont dans les mêmes bars que Dupont
- Q3 - Les buveurs qui fréquentent au moins un bar où l'on sert une bière qu'ils aiment
- Q4 - Les buveurs qui ne fréquentent aucun bar où l'on sert une bière qu'ils aiment
- Q5 - Les buveurs qui fréquentent tous les bars
- Q6 - Les buveurs qui fréquentent tous les bars qui servent au moins une bière qu'ils aiment
- Q7 - Les buveurs qui ne fréquentent que les bars qui ne servent que les bières qu'ils aiment
- Q8 - Donner pour chaque buveur, le nombre de bars servant une bière qu'ils aiment
- Q9 - Les buveurs qui fréquentent au moins 2 bars où l'on sert une bière qu'ils aiment
- Q10 - Le degré moyen par type de bière
- Q11 - La bière ayant le degré le plus élevé
- Q12 - Le type de bière ayant le degré moyen le plus élevé