SQL et Algèbre relationnelle

PostgreSQL

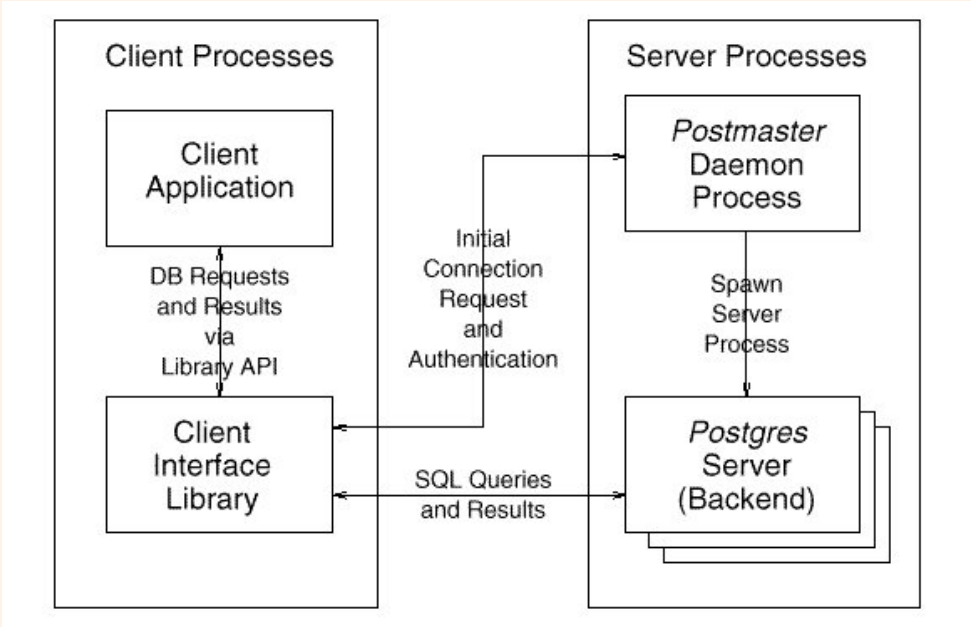
Alexis NEDELEC

Centre Européen de Réalité Virtuelle
Ecole Nationale d'Ingénieurs de Brest

Architecture PostgreSQL

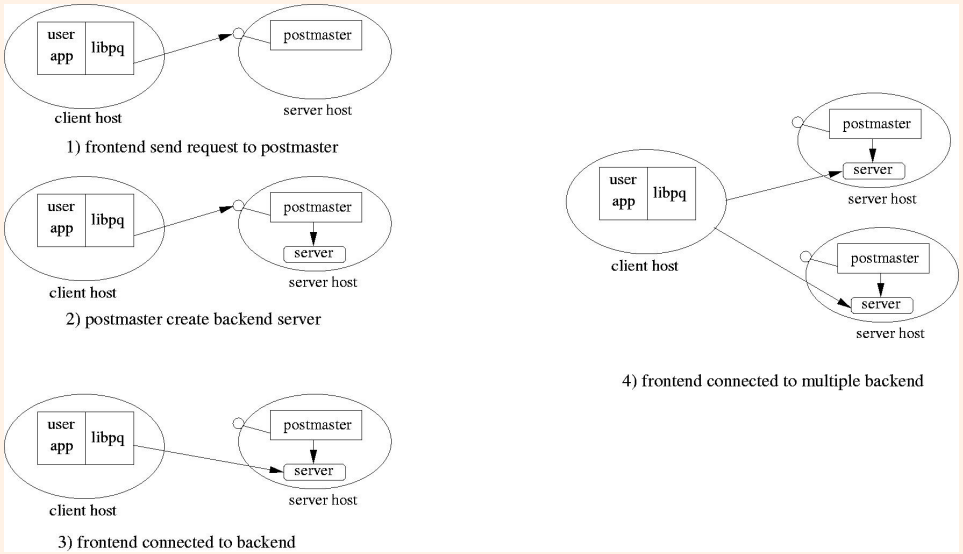
Client-Serveur (www.postgresql.org)

1. un démon “superviseur” : `postmaster`
2. une application cliente, **front-end** (`psql` par exemple)
3. un ou plusieurs serveurs de BD, **back-end** (`postgres`)



Architecture PostgreSQL

Client-Serveur (www.postgresql.org)

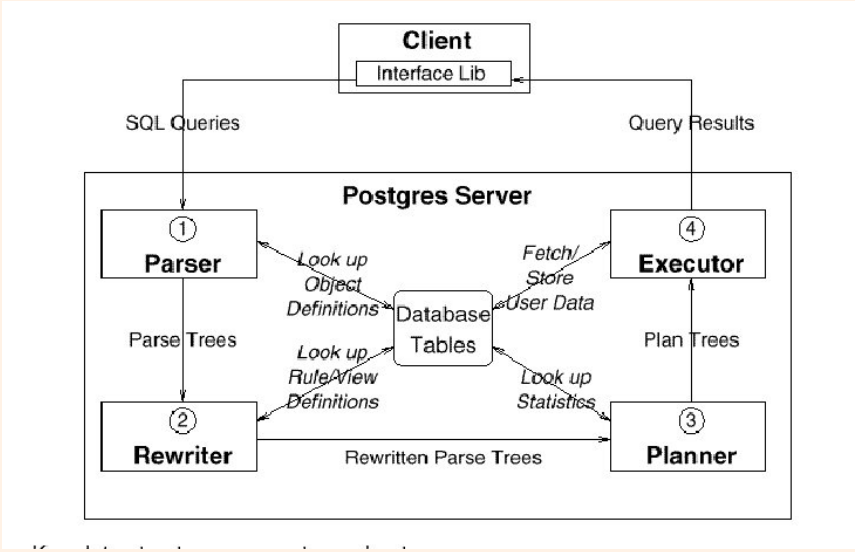


Interfaces de programmation d'applications (API)

- C (`libpq`), Embedded C (`ecpg`)
- C++ (`libpq++`), Java (JDBC), ODBC
- Python (`pygresql`), Tcl/Tk (`libpgtcl`)
- Perl (`pgsql_perl5`), ...

Architecture PostgreSQL

Dictionnaire de données



- vérification syntaxique, arbre de requête (`Parser`)
- vérification de règles, transformation d'arbre (`Rewriter`)
- plan d'éxecution, utilisation d'index (`Planner`)
- accès disque, récupération d'enregistrements (`Executor`)

Architecture PostgreSQL

Dictionnaire de Données

Méta-base de données : des tables pour décrire les tables

- `pg_class` : description des tables de la base
- `pg_attribute` : un enregistrement par colonne de table
- `pg_index` : un enregistrement par index
- `pg_relcheck` : informations sur les contraintes de colonnes
- `pg_proc` : description des fonctions
- `pg_language` : langage d'implémentation des fonctions
- `pg_operator` : opérateurs utilisables dans des expressions
- `pg_type` : les types du dictionnaire (`CREATE TYPE`)

Architecture PostgreSQL

Accès au serveur PostgreSQL

```
{logname@hostname} createdb BdM
{logname@hostname} psql -h hostname -U user BdM
 Welcome to the POSTGRESQL interactive sql monitor:
  Please read the file COPYRIGHT ... POSTGRESQL

   type \? for help on slash commands
   type \q to quit
   type \g or terminate with semicolon to execute query
 You are currently connected to the database: BdM

BdM=> SELECT * FROM pg_class;
...
```

Modèle de données

Cahier des charges

On se propose de modéliser un "Système d'Information" sur l'ensemble des bars du monde qui proposent des bières dans leurs services.

Sur une bière on dispose des informations suivantes :

- son nom, sa couleur et son origine (pays) de fabrication

D'un bar on connait :

- son nom et sa localisation (nom de pays).

On sait de plus que, si une bière est servie dans un bar on doit alors connaître le stock de cette bière dans ce bar.

Modèle de données

Cahier des charges

On sait par ailleurs que :

- un bar peut, ou non, servir une ou plusieurs bières.
- une bière peut (ne pas) être servie dans un ou plusieurs bars.

Modélisation

On a donc une association multiple entre deux entités :

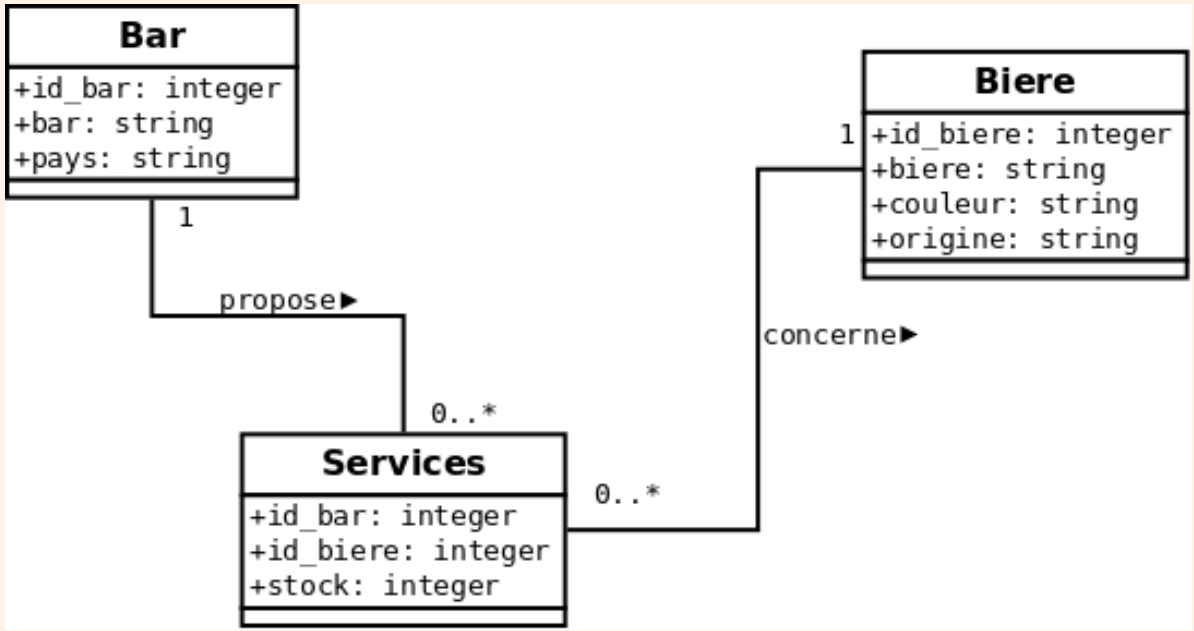
- les `bars` qui servent des `bieres`

Modèle de données

Modélisation UML

Association multiple entre deux entités :

- modèle relationnel : créer une table (classe) associative



Modèle de données

Modèle relationnel

- bars(id_bar,bar,pays)
- bieres(id_biere,biere,couleur,origine)
- services(#id_bar,#id_biere,stock)

Notation : clés primaires soulignées, clés étrangères "hashtagées"

bars(id_bar,bar,pays)

```
CREATE TABLE bars ( id_bar SERIAL PRIMARY KEY,
                    bar  VARCHAR(20),
                    pays VARCHAR(20) );
```

SERIAL : gestion automatique de séquence pour les identifiants

Modèle de données

bieres(id_biere,biere,couleur,origine)

```
CREATE TABLE bieres ( id_biere SERIAL PRIMARY KEY,
                      biere   VARCHAR(20),
                      couleur VARCHAR(10),
                      origine VARCHAR(20) );
```

services(#id_bar,#id_biere,stock)

```
CREATE TABLE services (
    id_bar INTEGER NOT NULL,id_biere INTEGER NOT NULL,
    stock SMALLINT,
    PRIMARY KEY(id_bar,id_biere),
    FOREIGN KEY(id_bar) REFERENCES bars,
    FOREIGN KEY(id_biere) REFERENCES bieres );
```

Modèle de données

Insertion d'informations

```
INSERT INTO bars(bar,pays)
  VALUES ('Bar du Coin', 'France');
...
INSERT INTO bieres(biere,couleur,origine)
  VALUES ('Kronenbourg','Blonde','France');
...
INSERT INTO services(id_bar,id_biere,stock)
(
  SELECT id_bar,id_biere,1000
  FROM bars,bieres
  WHERE bar='Bar du Coin' AND biere='Kronenbourg'
    AND couleur='Blonde'
);
...
```

Instances de relations

Etat de la Base de Données : `bars,bieres,services`

bieres

id_biere	biere	couleur	origine
1	Kronenbourg	Blonde	France
2	Guinness	Brune	Irlande
3	Heineken	Blonde	Hollande
4	Newcastle	Rousse	UK
5	Spaten	Blonde	Allemagne
6	Bush	Blonde	USA
7	Kanterbrau	Blonde	France
8	Kronenbourg	Brune	France

bars

id_bar	bar	pays
1	Bar du Coin	France
2	Corners Pub	USA
3	Cafe der Ecke	Allemagne
4	Cafe des Amis	France

services

id_bar	id_biere	stock
1	1	1000
1	2	250
1	3	50
1	4	10
2	1	100
2	6	1500
3	5	5000

Formulation de requêtes

Opérateurs de l'algèbre de relationnelle

- Π : Projection sur certaines informations
  - $\Pi_{pays}(bars)$
- $\times$: Produit cartésien entre deux ensembles
  - $\times(bars, bieres)$
- σ : Restriction par une formulation logique
  - $\sigma_{[pays='France']}(bars)$
- $\wedge, \vee$: connecteurs logiques
  - $\sigma_{[pays='France' \wedge bar='Bar\ des\ Amis']}(bars)$
- $\cup, \cap, \setminus$: opérations ensemblistes
  - $\cup(\Pi_{pays}(bars), \Pi_{origine}(bieres))$
- $\bowtie, \rtimes, \ltimes, \rtimes\!\ltimes$: jointures interne,externe (left,right,full)
  - $\bowtie_{[pays=origine]}(bars, bieres)$

Formulation de requêtes

Langage SQL

- Projection : `SELECT`
- Produit cartésien : `FROM`
- Restriction : `WHERE`
- Connecteurs logiques : `AND,OR`
- opérations ensemblistes : `UNION,INTERSECT,EXCEPT`
- jointures :
  - `INNER JOIN, NATURAL JOIN`
  - `OUTER JOIN (LEFT,RIGHT,FULL)`

Projection : Π

Sélection d'information : $\Pi_{id\_bar}(services)$

```
SELECT id_bar
FROM services;

id_bar
------
   1
   1
   1
   1
   2
   2
   3
(7 rows)
```

```
SELECT DISTINCT id_bar
FROM services;

id_bar
------
   1
   2
   3
(3 rows)
```

Restriction : σ

Filtrage d'information : $\sigma_{[origine='France']}(bieres)$

```
SELECT *
FROM bieres
WHERE origine='France';
```

```
id_biere|biere        |couleur   |origine
--------+-------------+----------+-------
      1 |Kronenbourg |Blonde    |France
      7 |Kanterbrau  |Blonde    |France
      8 |Kronenbourg |Brune     |France
(3 rows)
```

Projection et Restriction : Π, σ

Sélection et Filtrage : $\Pi_{(biere,couleur)}(\sigma_{[origine='France']}(bieres))$

```
SELECT biere, couleur
FROM bieres
WHERE origine='France';
```

```
biere        |couleur
-------------+---------
Kronenbourg |Blonde
Kanterbrau   |Blonde
Kronenbourg |Brune
(3 rows)
```

Connecteurs logiques : $\wedge, \vee$

$\sigma_{[origine='France' \wedge couleur='Brune']}(bieres)$

```
SELECT *
FROM bieres
WHERE origine='France' AND couleur='Brune';
```

```
id_biere|biere       |couleur |origine
--------+------------+--------+-------
      8 |Kronenbourg |Brune   |France
(1 row)
```

Connecteurs logiques : $\wedge, \vee$

$\Pi_{(biere,couleur)}(\sigma_{[origine='France' \wedge couleur='Brune' \vee couleur='Blonde']}(bieres))$

```
SELECT  biere, couleur
FROM bieres
WHERE origine='France'
  AND couleur='Brune' OR couleur='Blonde';
```

```
biere        |couleur
-------------+----------
Kronenbourg |Blonde
Heineken     |Blonde
Spaten       |Blonde
Bush         |Blonde
Kanterbrau   |Blonde
Kronenbourg |Brune
(6 rows)
```

Connecteurs logiques $\wedge, \vee$: priorité

$\Pi_{(biere,couleur)}(\sigma_{[origine='France' \vee couleur='Brune' \wedge couleur='Blonde']}(bieres))$

```
SELECT  biere, couleur
FROM bieres
WHERE origine='France'
  OR couleur='Brune' AND couleur='Blonde';
```

```
biere                |couleur
---------------------+----------
Kronenbourg          |Blonde
Kanterbrau           |Blonde
Kronenbourg          |Brune
(3 rows)
```

Produit cartésien : $\times$

Recherche sur plusieurs tables : $\times(bars, bieres)$

```
SELECT *
FROM bars, bieres;          -- FROM bars CROSS JOIN bieres
```

```
 id_bar |      bar       |   pays    | id_biere |    biere    | couleur |  origine
--------+----------------+-----------+----------+-------------+---------+-----------
      1 | Bar du Coin    | France    |        1 | Kronenbourg | Blonde  | France
      2 | Corners Pub    | USA       |        1 | Kronenbourg | Blonde  | France
      3 | Cafe der Ecke  | Allemagne |        1 | Kronenbourg | Blonde  | France
      4 | Cafe des Amis  | France    |        1 | Kronenbourg | Blonde  | France
      1 | Bar du Coin    | France    |        2 | Guinness    | Brune   | Irlande
      2 | Corners Pub    | USA       |        2 | Guinness    | Brune   | Irlande
      3 | Cafe der Ecke  | Allemagne |        2 | Guinness    | Brune   | Irlande
      4 | Cafe des Amis  | France    |        2 | Guinness    | Brune   | Irlande
.....................................................................................
      1 | Bar du Coin    | France    |        7 | Kanterbrau  | Blonde  | France
      2 | Corners Pub    | USA       |        7 | Kanterbrau  | Blonde  | France
      3 | Cafe der Ecke  | Allemagne |        7 | Kanterbrau  | Blonde  | France
      4 | Cafe des Amis  | France    |        7 | Kanterbrau  | Blonde  | France
      1 | Bar du Coin    | France    |        8 | Kronenbourg | Brune   | France
      2 | Corners Pub    | USA       |        8 | Kronenbourg | Brune   | France
      3 | Cafe der Ecke  | Allemagne |        8 | Kronenbourg | Brune   | France
      4 | Cafe des Amis  | France    |        8 | Kronenbourg | Brune   | France
(32 lignes)
```

Jointures : $\bowtie$ ($\times$, σ)

$\bowtie_{[bars.id\_bar=services.id\_bar]}$ $(bars, services)$

```
SELECT *
FROM bars b, services s
WHERE b.id_bar = s.id_bar;
```

```
id_bar|bar             |pays      |id_bar|id_biere|stock
------+--------------+----------+------+--------+-----
     1|Bar du Coin   |France    |     1|       4|   10
     1|Bar du Coin   |France    |     1|       3|   50
     1|Bar du Coin   |France    |     1|       2|  250
     1|Bar du Coin   |France    |     1|       1| 1000
     2|CornersPub    |USA       |     2|       6| 1500
     2|CornersPub    |USA       |     2|       1|  100
     3|Cafe der Ecke |Allemagne |     3|       5| 5000
(7 rows)
```

Jointure Naturelle

$\bowtie_{[id\_bar]} (bars, services)$

```
SELECT *
FROM bars NATURAL JOIN  services;
```

```
id_bar|bar             |pays      |id_biere|stock
------+----------------+----------+--------+-----
     1|Bar du Coin     |France    |       4|   10
     1|Bar du Coin     |France    |       3|   50
     1|Bar du Coin     |France    |       2|  250
     1|Bar du Coin     |France    |       1| 1000
     2|CornersPub      |USA       |       6| 1500
     2|CornersPub      |USA       |       1|  100
     3|Cafe der Ecke   |Allemagne |       5| 5000
(7 rows)
```

Jointure interne (INNER JOIN)

$\Pi_{(id\_bar,id\_biere,pays)}(\bowtie_{[pays=origine]} (bars, bieres))$

```
SELECT id_bar,id_biere,pays
FROM bars INNER JOIN bieres ON (pays=origine);
```

id_bar	id_biere	pays
4	1	France
1	1	France
3	5	Allemagne
2	6	USA
4	7	France
1	7	France
4	8	France
1	8	France

Jointure externe (OUTER JOIN)

$\bowtie_{[pays=origine]}(bars, bieres)$

```
SELECT *
FROM bars LEFT OUTER JOIN bieres ON (pays=origine);
INSERT INTO bars  VALUES (5,'Hard Rock Caffee',NULL);
SELECT *
FROM bars LEFT OUTER JOIN bieres ON (pays=origine);
```

$\bowtie_{[pays=origine]}(bars, bieres)$

```
SELECT *
FROM bars RIGHT OUTER JOIN bieres ON (pays=origine);
```

$\bowtie_{[pays=origine]}(bars, bieres)$

```
SELECT *
FROM bars FULL OUTER JOIN bieres ON (pays=origine);
```

Jointure : entre plusieurs tables

$\bowtie_{[s.id\_biere=bi.id\_biere]} (\bowtie_{[ba.id\_bar=s.id\_bar]} (bars, services), bieres)$

```
SELECT *
FROM bars ba, services s, bieres bi
WHERE ba.id_bar=s.id_bar AND s.id_biere=bi.id_biere;
```

```
id_bar|bar|pays|id_bar|id_biere|stock|id_biere|biere|couleur|origine
------+---+----+------+--------+-----+--------+-----+-------+-------
  ... |...|... |  ... |   ...  | ... |   ...  | ... |  ...  |  ...
```

Résultat de la requête

- concaténation des colonnes des trois tables
- redondances de colonnes (`id_bar`,`id_biere`)

Jointure naturelle : entre plusieurs tables

$\bowtie_{[id\_biere]} (\bowtie_{[id\_bar]} (bars, services), bieres)$

```
SELECT *
FROM bars NATURAL JOIN services NATURAL JOIN bieres;
```

```
id_biere|id_bar|bar|pays|stock|biere|couleur|origine
--------+------+---+----+-----+-----+-------+-------
```

Ordre des jointures

- $\bowtie (\bowtie (bars, services), bieres)$?
- $\bowtie (\bowtie (bars, bieres), services)$?
- $\bowtie (\bowtie (bieres, services), bars)$?

Création de vues

Encapsulation de requête SQL

```
CREATE VIEW BarsDuMonde AS
  SELECT bar, pays, stock, biere, couleur, origine
  FROM bars NATURAL JOIN services NATURAL JOIN bieres;
```

Utilisation d'une vue : comme une table

```
SELECT * FROM BarsDuMonde
```

```
    bar       | pays |stock|   biere |couleur| origine
--------------+------+-----+---------+-------+----------
Bar du Coin   |France|1000 |Kro...   |Blonde |France
Bar du Coin   |France| 250 |Guinness |Brune  |Irlande
Bar du Coin   |France|  50 |Heineken |Blonde |Hollande
Bar du Coin   |France|  10 |Newcastle|Rousse |UK
Corners Pub   |USA   | 100 |Kro...   |Blonde |France
Corners Pub   |USA   |1500 |Bush     |Blonde |USA
Cafe der Ecke|All...|5000 |Spaten   |Blonde |Allemagne
(7 rows)
```

SQL et Algèbre relationnelle

Exercice

Soient R et S les deux relations suivantes :

- `R(A,B,C)` où A,B,C : attributs de type entier
- `S(A,B)` où A,B : attributs de type entier

Ecrire les requêtes SQL des expressions algébriques :

- $T = \Pi_{(A,B)}(R)$
- $\Pi_A(T)$
- $\Pi_A(\sigma_{[A+B<5]}(T))$
- $\sigma_{[R.A>4]}(\times(R,S))$ que l'on décomposera en
  - $RS = \times(R,S)$
  - $\sigma_{[A>4]}(RS)$

SQL et Algèbre relationnelle

Exercice

- $\times(\sigma_{[A>4]}(R), S)$
- $\bowtie_{[R.A=S.A]} (R, S), \bowtie_{[R.B=S.B]} (R, S)$
- $\bowtie_{[R.A=S.A \wedge R.B=S.B]} (R, S)$

A partir des deux instances suivantes des relations R et S :

R		
A	**B**	**C**
1	2	3
5	6	7
1	2	4
1	6	7

S	
A	**B**
3	4
7	8
1	2
1	6

- Quels résultats obtiendra-t-on en appliquant ces expressions algébriques ?
- Créez la base de données correspondante (PostgreSQL ou SQlite)
- Vérifiez vos résultats en formulant les requêtes SQL correspondantes

Union : $\cup$

Nom des pays ayant des bars et/ou des bières

$\cup(\Pi_{(pays)}(bars), \Pi_{(origine)}(bieres))$

```
(SELECT pays FROM bars)
UNION
(SELECT origine FROM bieres) ;
```

```
   pays
-----------
 Allemagne
 France
 Hollande
 Irlande
 UK
 USA
(6 lignes)
```

Union : ∪

Concaténation sans tri de deux requêtes

```
(SELECT pays FROM bars)
UNION ALL
(SELECT origine FROM bieres) ;
```

Concaténation de deux requêtes triées

```
(SELECT DISTINCT pays FROM bars)
UNION ALL
(SELECT DISTINCT origine FROM bieres) ;
```

Intersection : ∩

Nom des pays ayant des bars et fabriquant de la bière

$\cap(\Pi_{(pays)}(bars), \Pi_{(origine)}(bieres))$

```
(SELECT pays FROM bars)
INTERSECT
(SELECT origine FROM bieres) ;
```

Données communes à deux requêtes :

```
   pays
-----------
 Allemagne
 France
 USA
(3 lignes)
```

Intersection : ∩

Formulation avec requête imbriquée

```
SELECT *
FROM bars
WHERE pays IN (SELECT origine FROM bieres);

SELECT pays
FROM bars
WHERE pays=ANY(SELECT origine FROM bieres);
```

Intersection : ∩

Formulation avec requête imbriquée

```
SELECT pays
FROM bars
WHERE  EXISTS (SELECT *
               FROM bieres
               WHERE pays=origine);
```

Requête imbriquée ou jointure ?

```
SELECT DISTINCT pays
FROM bars,bieres
WHERE pays=origine;
```

Différence : \

Nom des pays fabriquant de la bière et n'ayant pas de bars

$\setminus(\Pi_{(origine)}(bieres), \Pi_{(pays)}(bars))$

```
SELECT origine FROM bieres)
EXCEPT
(SELECT pays FROM bars)
```

Données d'une requête qui n'appartiennent pas à l'autre

```
 origine
----------
 Hollande
 Irlande
 UK
(3 lignes)
```

Différence : \

Formulation avec requête imbriquée

```
SELECT *
FROM bieres
WHERE origine NOT IN (SELECT pays FROM bars);

SELECT origine
FROM bieres
WHERE origine <> ALL (SELECT pays FROM bars);
```

Différence : \

Formulation avec requête imbriquée

```
SELECT origine
FROM bieres
WHERE  NOT EXISTS (SELECT *
                   FROM bars
                   WHERE pays=origine);
```

Requête imbriquée ou jointure ?

```
SELECT DISTINCT origine
FROM bieres, bars
WHERE origine <> pays;
```

L'opération ensembliste de différence ne pourra pas être formulée par une telle jointure.

Fonctions d'agrégat

Calculs sur un ensemble

- `COUNT,SUM` : nombre, somme d'occurences
- `MIN,MAX,AVG` : valeur min,max et moyenne
- `STDEV_POP,STDEV_SAMP` : écart-type de population,échantillon
- `VAR_POP,VAR_SAMP` : variance de population,échantillon

Calculs sur un regroupement d'ensemble

- `GROUP BY` : créer des sous-ensembles de l'ensemble

Conditions d'applications des calculs

- `BLOB,CLOB,NCLOB` : calculs inapplicables
- `NULL` : non-comptabilisés dans les calculs

Calculs sur un ensemble

$\Pi_{count(*)}(bars)$: nombre de bars dans la base de données

```
SELECT COUNT(*) FROM bars;
 count
-------
     4
(1 ligne)
```

Calculs sur un ensemble

$\Pi_{count(*)}(bars\_du\_monde)$: nombre de bars qui servent des bières

```
SELECT COUNT(*) FROM bars_du_monde;
 count
-------
     7
(1 ligne)
```

Création de vues : bars_du_monde

$$NJ_1 = \bowtie_{[bars.id\_bar=services.id\_bar]} (bars, services)$$
$$NJ_2 = \bowtie_{[R_1.id\_biere=bieres.id\_biere]} (NJ_1, bieres)$$
$$bars\_du\_monde = \Pi_{(bar,pays,biere,couleur,origine,stock)}(NJ_2)$$

Calculs et groupement

$G_{pays}^{(pays,count(bar))}(bars)$: nombre de bars par pays

```
SELECT pays, COUNT(bar) FROM bars GROUP BY pays
   pays     | count
------------+-------
 Allemagne  |     1
 France     |     2
 USA        |     1
```

$G_{pays}^{(pays,count(bar))}(bars\_du\_monde)$: nombre de services par pays

```
SELECT pays, COUNT(bar) FROM bars_du_monde GROUP BY pays
   pays     | count
------------+-------
 Allemagne  |     1
 USA        |     2
 France     |     4
```

Calculs et groupement

Stocks de bières servies par bar, pour les bars ayant au moins 3 bières en stock

$$G^{(id\_bar, sum(stock))}_{(id\_bar), [count(id\_biere)>3]}(services)$$

```
SELECT id_bar,SUM(stock)
FROM services
GROUP BY id_bar
HAVING COUNT(id_biere) > 3;
 id_bar | sum
--------+------
      1 | 1310
```

Portée des fonctions

Noms des bars et de leur quantité de Kronenbourg en stock supérieur au minimum des stocks de Kronenbourg

Requêtes imbriquées

```
SELECT bar, stock
FROM  bars_du_monde
WHERE biere='Kronenbourg'
AND stock > (
             SELECT MIN(stock)
             FROM   services NATURAL JOIN bieres
             WHERE  biere='Kronenbourg'
 );
     bar      | stock
--------------+-------
 Bar du Coin |  1000
(1 ligne)
```

Portée des fonctions

Noms des bars et de leur quantité de Kronenbourg en stock supérieur au minimum des stocks de Kronenbourg

Requêtes imbriquées : décomposition en vues

on utilise la vue $bars\_du\_monde$ pour récupérer

- les informations sur les bars qui servent de la 'Kronenbourg'
  - $R_1 = \sigma_{[biere='Kronenbourg']}(bars\_du\_monde)$

On récupère les 'Kronenbourg' disponibles dans la base données

- $R_2 = \sigma_{[biere='Kronenbourg']}(bieres)$

qui sont proposées dans des services

- $NJ_3 = \bowtie_{[R_2.id\_biere=services.id\_biere]} (R_2, services)$

Portée des fonctions

Requêtes imbriquées : décomposition en vues

on récupère dans le résultat des bars qui servent de la 'Kronenbourg' (R_1) celles dont le stock de 'Kronenbourg' est supérieur au mini des stocks de 'Kronenbourg' proposées dans des services (NJ_3)

- $R_3 = \sigma_{[stock > \Pi_{min(stock)}(NJ_3)]}(R_1)$

On récupère au final le nom des bars et leur quantité de 'Kronenbourg' en stock supérieur au minimum des stocks de 'Kronenbourg'

- $P = \Pi_{(bar,stock)}(R_3)$

Créer les vues et vérifier qu'on obtient le même résultat que la requête initiale (sans les vues) en exécutant la requête :

- `SELECT * FROM P;`

Portée des fonctions

Quantificateur universel (∀)

Nom des bars qui servent toutes les bières :

- Quel que soit la bière, elle est servie dans les bars à trouver.
- trouver les bars qui servent un nombre de bières différentes égal au nombre total de bières de la base de données

Regroupement avec conditions

```
SELECT bar
FROM bars NATURAL JOIN services
GROUP BY bar
HAVING COUNT(DISTINCT id_biere) = (
                                  SELECT COUNT(id_biere)
                                  FROM bieres
                                      );
```

Division relationnelle

Algèbre relationnelle

En algèbre relationnelle le quantificateur universel ($\forall$) :

- $\forall y \in S, (x, y) \in R$

entre deux ensembles (R, S) s'exprime par l'opérateur ($\div$) :

- $T[X] = \div(R[X, Y], S[Y])$

de division relationnelle qui permettra d'obtenir :

- les valeurs des groupes d'attributs X des éléments de R
- qui sont en relation
- avec tous les éléments de S
- sur les valeurs des groupes d'attributs Y

Division relationnelle

Algèbre relationnelle

$\forall$ n'existe pas en SQL mais peut s'exprimer :

- par une formulation de double négation
  - `NOT EXISTS` imbriqués
- en combinant les opérateurs
  - $\Pi, \times, \setminus$
- en appliquant des fonctions d'agrégats sur des regroupements
  - `GROUP BY HAVING, COUNT()`

Division relationnelle

Double négation (NOT EXISTS)

- Trouver les $X \in R(X, Y)$
- tel qu' il n'existe pas de $Y \in S(Y)$
- pour lequel il n'existe pas d'association entre ce X et ces Y

Requête SQL

```
SELECT r1.X  FROM R r1
WHERE NOT EXISTS (SELECT * FROM S
                  WHERE NOT EXISTS (
                       SELECT * FROM R r2
                       WHERE r1.X=r2.X AND S.Y=r2.Y
                  )
);
```

Division relationnelle

Ainsi pour trouver : "les bars qui servent toutes les bières" :

Trouver les *id_bar* tel qu' il n'existe pas de bière pour laquelle ...

```
SELECT id_bar
FROM services s1
WHERE NOT EXISTS( SELECT *
                  FROM bieres b
```

... il n'existe pas de service associant ce bar et cette bière

```
    WHERE NOT EXISTS( SELECT *
                      FROM services s2
                      WHERE s1.id_bar=s2.id_bar
                       AND b.id_biere = s2.id_biere
                    )
                )
```

Division relationnelle

A l'aide des opérateurs $\Pi, \times, \setminus$

$T[X] = \div(R[X,Y], S[Y])$

$T[X] == \setminus(\ \Pi_{(X)}(R)\ ,\ \Pi_{(X)}(\ \setminus(\ \times(\Pi_{(X)}(R)\ ,\ S\ ), R\ )\ )$

Décomposition en suite d'opérations algébriques

- $T_1 = \Pi_{(X)}(R)$: éléments X dans l'ensemble R
- $T_2 = \times(T_1, S)$: lier tous les X de R avec tous les (Y) de S
- $T_3 = \setminus(T_2, R)$: les liaisons (X, Y) qui n'existent pas dans R
- $T_4 = \Pi_{(X)}(T_3)$: seulement les X de T_3
- $T[X] = \setminus(T_1, T_4)$: les X de R qui ne sont pas dans les X des liaisons (X, Y) qui n'existent pas
  $\Leftrightarrow$ les (X) de R liés à tous (Y) de S

Division relationnelle

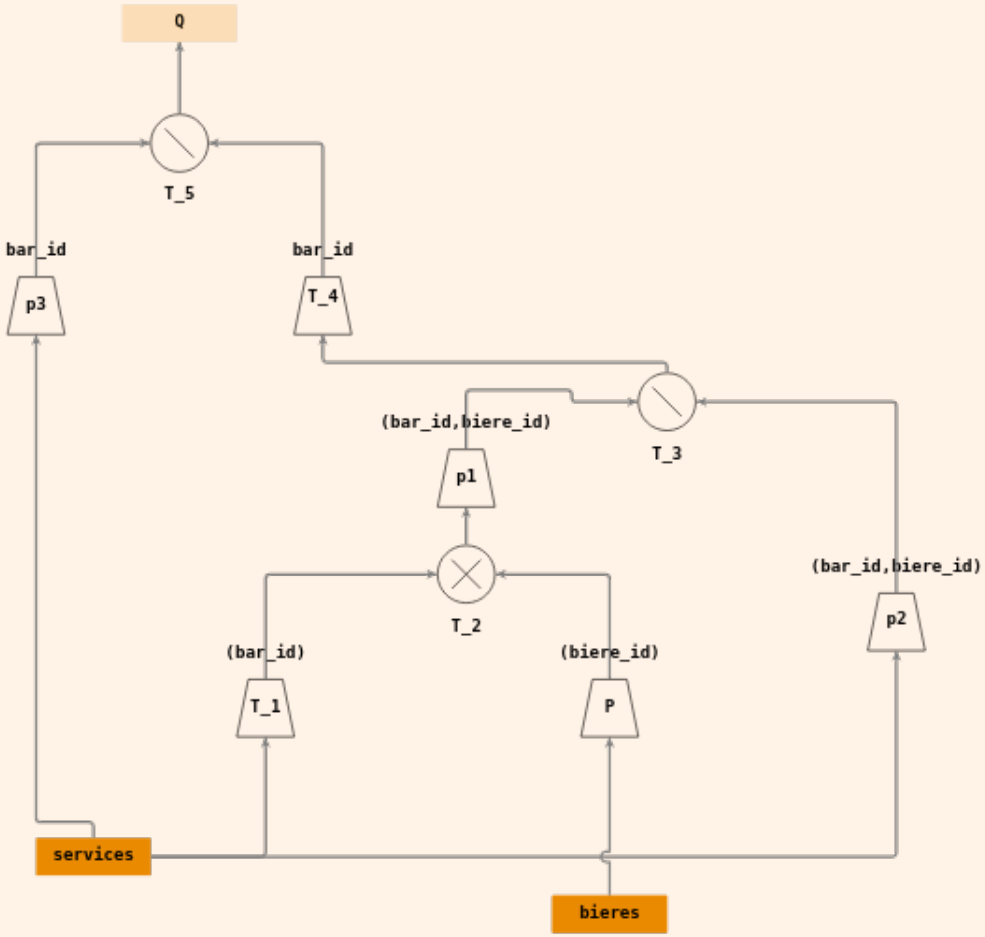
$T[id\_bar] = \div(services[id\_bar, id\_biere], S[id\_biere])$

```
CREATE VIEW T1 AS SELECT R.id_bar FROM services R;
CREATE VIEW T2 AS
SELECT T1.id_bar,S.id_biere FROM T1,bieres S;
CREATE VIEW T3 AS
SELECT id_bar, id_biere FROM T2
EXCEPT
SELECT R.id_bar, R.id_biere FROM services R;
CREATE VIEW T4 AS SELECT id_bar FROM T3;
CREATE VIEW T5 AS
SELECT id_bar, FROM T1  EXCEPT  SELECT id_bar FROM T4;

SELECT * FROM T5;
```

Division relationnelle

Arbre de requêtes



Gestion du Temps

Types Date, Time, Timestamp

```
SELECT CURRENT_DATE AS today_is;
  today_is
------------
 2020-04-02
SELECT CURRENT_TIME AS time_is;
      time_is
--------------------
 15:25:14.266754+02
SELECT CURRENT_TIMESTAMP AS timestamp_is;
         timestamp_is
-------------------------------
 2020-04-02 15:26:09.228392+02
```

Gestion du Temps

ALTER TABLE : modification de relation

```
ALTER TABLE services
ADD COLUMN depot DATE  DEFAULT CURRENT_DATE;
```

```
SELECT * FROM services;
 id_bar | id_biere | stock |   depot
--------+----------+-------+------------
      1 |        1 |  1000 | 2020-04-02
      1 |        2 |   250 | 2020-04-02
      1 |        3 |    50 | 2020-04-02
      1 |        4 |    10 | 2020-04-02
      2 |        1 |   100 | 2020-04-02
      2 |        6 |  1500 | 2020-04-02
      3 |        5 |  5000 | 2020-04-02
(7 lignes)
```

Création d'utilisateurs

utilisateurs et droits

```
=>CREATE GROUP A3S2;
CREATE GROUP
=>CREATE USER A3S2B1 WITH PASSWORD  'A3S2B1'
->NOCREATEDB  NOCREATEUSER IN GROUP A3S2;
CREATE USER
```

Modification des droits utilisateurs

```
=>ALTER USER A3S2B1 CREATEDB
ALTER USER
=>GRANT SELECT ON services TO A3S2B1;
GRANT
```

Transactions utilisateurs

Transaction et verrouillage de tables

```
=>BEGIN TRANSACTION;
BEGIN
=>LOCK TABLE services  IN ACCESS SHARE MODE;
LOCK TABLE
```

Modification de table dans une transaction

```
=>INSERT INTO services  VALUES (4,2,10000);
INSERT 17297 1
```

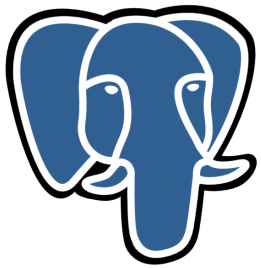
Validation de transaction

- `COMMIT` : nouvel état de la BD
- `ROLLBACK` : état de la BD avant lancement de transaction

Conclusion

PostgreSQL

- Initiateur Michael Stonebraker (DARPA, projet POSTGRES)
- SGBDR Open source en constante évolution
- conforme au standard ANSI-SQL 2008
- langage procédural PL/pgSQL similaire au PL/SQL d'Oracle
- supporte d'autres langages (Python, Java,Perl, C/C++ ..)
- réplication de données sur différents serveurs
- outils d'administration `pgadmin,phpPgAdmin ...`



PostgreSQL

Bibliographie

Adresses “au Net”

- CRD ENIB
- PostgreSQL
- SQlite
- Cours et tutoriaux sur SQL
- Tutoriaux W3C sur SQL
- Site de Georges Gardarin
- Cours de Maude Manouvrier
- Références SQL du site developpez.com