Programmation SQL avec des fonctions backend

Technique peu utilisée (parce que peu connue), disponible dans les SGBD courants (DB2, Oracle, PostgreSQL et, dans une moindre mesure, MySQL).

Intérêts

- Apporte une solution élégante (souvent) et très efficace (toujours) à des problèmes de clauses qui seraient compliquées à implémenter en SQL, voire impossibles.
- Code serveur, en C, accessible en syntaxe SQL, donc indépendant de l'API utilisée pour l'application (C, C++, Java, Python, Perl, PHP, etc.)
- Se prête bien au développement de librairies spécialisées, sous forme d'un "SQL étendu", utilisables pour différents projets d'un même environnement thématique.

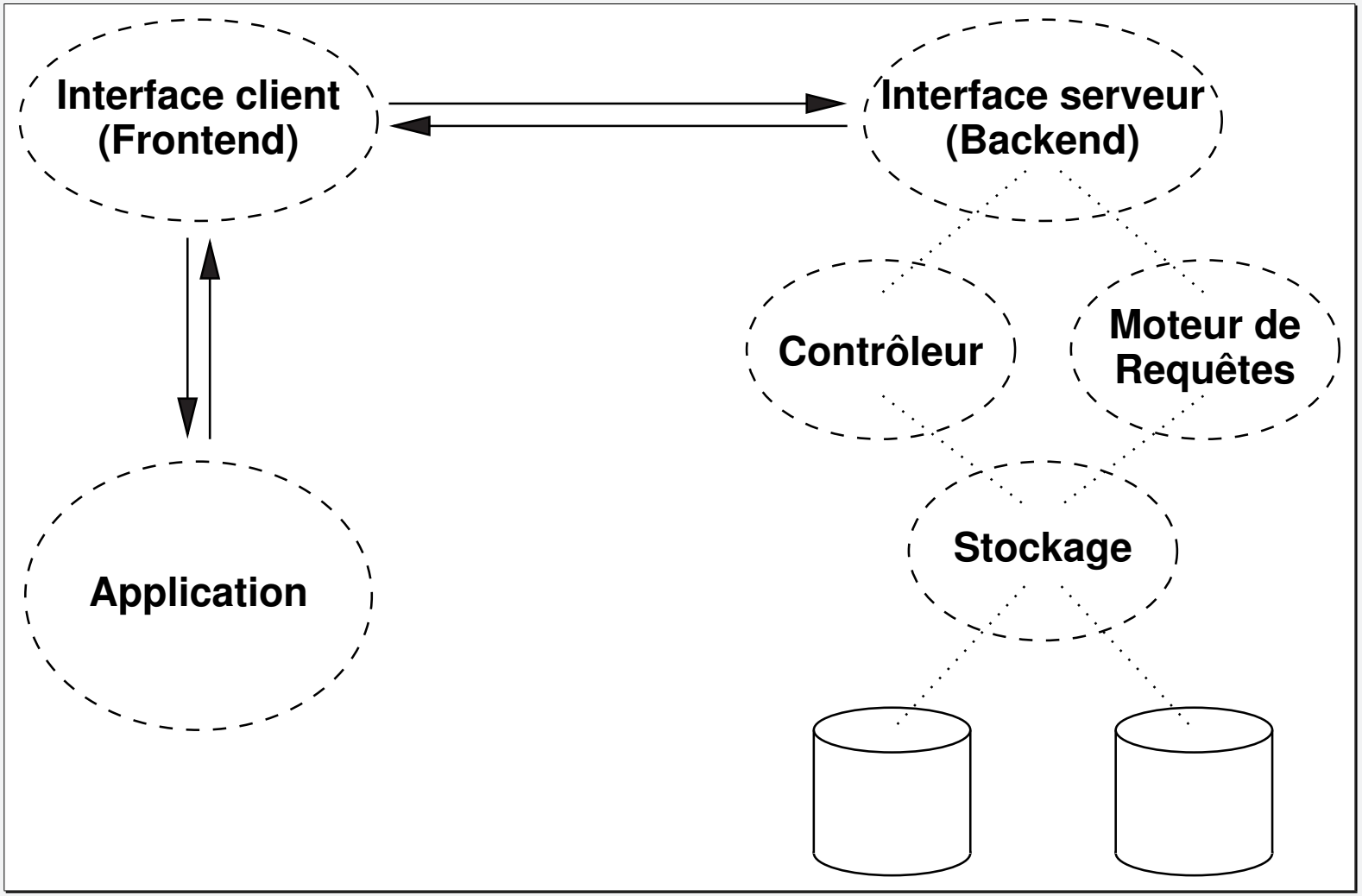
  (Par exemple astronomie en environnement université recherche.)

Inconvénients

- Facilité de mise en oeuvre variable selon les outils utilisés, SGBD, langage(s) de programmation d'applications.
- Portabilité partielle : code applicatif en C ANSI, donc portable, mais interface spécifique avec le SGBD.
- Nécessite un accès développement au serveur et des droits administrateurs sur le SGBD. (Inutilisable donc chez un hébergeur mais ne pose pas de problème dans le cadre d'une structure de projet qui possède ses propres serveurs.)

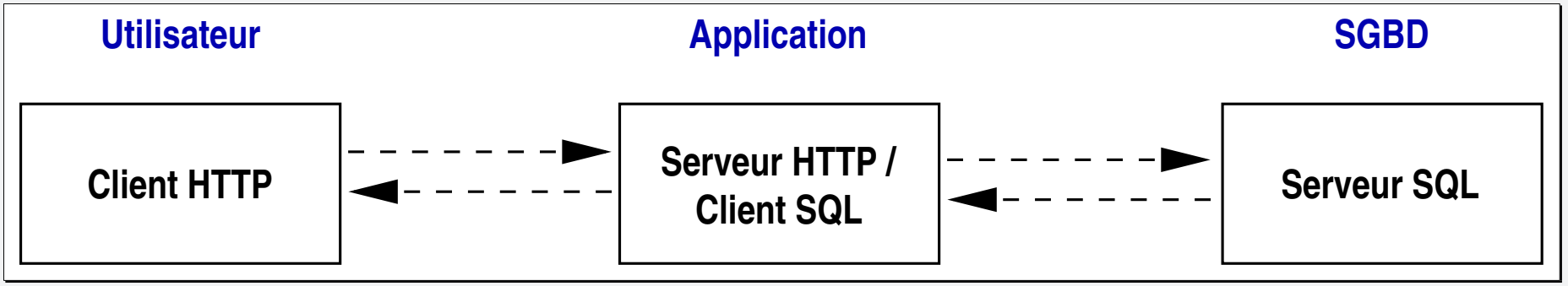
Client / Serveur SQL

Architecture logicielle d'un SGBD



Application SQL + Web

Architecture typique



Possibilité de répartir la programmation

- Sur le client HTTP : petits utilitaires Javascript
- Dans l'application : programme CGI en C, C++, Java, Python, Perl, PHP, etc.
- Dans le SGBD : utilitaires backend

Exemple 1 : distance angulaire

Le problème

On souhaite proposer une recherche dans un catalogue de sources par une clause de type "autour de". La base contient des tables de sources avec leur position, attributs (ra2000, de2000).

L'utilisateur spécifie (formulaire HTML) une position équatoriale (A0, D0) et une distance angulaire maximale Rmax.

La requête

À la soumission du formulaire, on (l'application CGI) doit construire une requête SQL du genre :

```
SELECT xxxx FROM xxxx
WHERE "distance de ra2000, de2000 à A0, D0" inférieure à Rmax;
```

On ne sait pas formuler cela en SQL sauf à disposer d'une fonction spécifique qui permettrait d'écrire quelque chose comme :

```
SELECT xxxx FROM xxxx
WHERE arcdist(ra2000, de2000, A0, D0) < Rmax;
```

Elle n'existe pas, on va la créer...

Fonction SQL distance angulaire

```
#define _XOPEN_SOURCE
#include <math.h>

#include <postgres.h>

#if PG_VERSION_NUM >= 80300
#include <fmgr.h>

#ifdef PG_MODULE_MAGIC
PG_MODULE_MAGIC;
#endif
#endif

static float8 __spinDecl(float8 D)
{
    while( D > M_PI ) D -= 2 * M_PI;
    while( D < - M_PI ) D += 2 * M_PI;
    if( D >  M_PI_2 ) D = M_PI - D;
    if( D < -M_PI_2 ) D = - M_PI - D;
    return D;
}
```

Fonction SQL distance angulaire

```
float8* arcdist(float8* pAs,float8* pDs,float8* pAc,float8* pDc)
{
    float8 A, a, b, cosa;
    float8* result = (float8*)palloc(sizeof(float8));

    /* Compute triangle arcs */
    A = *pAs - *pAc;
    a = __spinDecl(*pDs);
    b = __spinDecl(*pDc);

    /* Compute angular distance */
    cosa = sin(a) * sin(b) + cos(a) * cos(b) * cos(A);

    /* Return result */
    if( cosa < 0.99 ) *result = acos(cosa);
    else {
        float8 D = a - b;
        float8 cosd = cos((a + b) / 2);
        *result = sqrt(A * A * cosd * cosd + D * D);
    }

    return result;
}
```

Fonction SQL distance angulaire

Compilation, installation

Sous forme d'une librairie dynamique, par exemple libastro.so.

(Options gcc/Linux : `-ansi -Wall -O3 -shared -fPIC`)

Installation dans un répertoire du serveur SQL (droits d'accès pour le daemon postgres -> il faut être root),

par exemple :

```
/serv/pgsql/backend/libastro.so
```

Intégration dans le serveur PostgreSQL

```
CREATE FUNCTION "arcdist" (float8, float8, float8, float8)
    RETURNS float8
    AS '/serv/pgsql/backend/libastro.so', 'arcdist' LANGUAGE 'C';
```

Relecture (si librairie modifiée)

```
LOAD '/serv/pgsql/backend/libastro.so';
```

➤ Pas besoin de redémarrer le serveur mais droits administrateur Postgres pour faire le ci-dessus. Et travail sur un serveur de développement !

Exemple 2 : recherche de fréquences

Le problème

On dispose, dans une base, de résultats d'observations avec tous les paramètres de réglage instrumentaux (fréquence LO, largeur de bande, etc.)

On veut proposer à l'utilisateur de chercher dans les observations disponibles celles qui sont susceptibles d'avoir vu telle ou telle fréquence (une raie en spectro moléculaire).

NB: susceptible d'avoir vu ne signifie pas que la raie est effectivement visible dans les résultats d'observation mais simplement que les réglages instrumentaux sont compatibles avec une éventuelle détection de cette fréquence.

La requête

Dans les cas simples, on saura écrire directement en SQL quelque chose comme :

```
SELECT xxxx FROM xxxx
WHERE ((LO - 0.5 * BW) <= F0) AND (F0 <= (LO + 0.5 * BW));
```

(LO et BW sont des attributs de tuples, F0 est un paramètre de requête issu d'un formulaire HTML.)

Exemple 2 : recherche de fréquences

Dans les cas plus compliqués, prise en compte d'un offset oscillateur, application d'une correction Doppler sur les fréquences à partir de la vitesse radiale de la source observée, etc., l'écriture SQL va se compliquer très vite.

On va construire une fonction backend, à valeur booléenne, pour simplifier l'écriture des requêtes et améliorer la performance.

```
/*
 * FREQUENCY RANGE
 * ---------------
 * Checks if the frequency F is "seen" with an instrumental
 * configuration.
 *  F0 is the reference frequency
 *  SV a source velocity, m/s, for Doppler correction
 *  FO a frequency offset
 *  BW a bandwidth
 *  SC a scale factor for "half bandwidth", should be 0.5
 */

bool inrange(float8* pF, float8* pF0, float8* pSV,
    float8* pFO, float8* pBW, float8* pSC)
```

Exemple 2 : recherche de fréquences

Code PostgreSQL

```
bool inrange(float8* pF, float8* pF0, float8* pSV,
    float8* pFO, float8* pBW, float8* pSC)
{
    /* Doppler correction */
    double F0 = *pF0 / (1 + *pSV / 3.E+8);

    /* Plus offset */
    F0 += *pFO;

    /* Check range */
    if( *pF < (F0 - *pBW * *pSC) ) return false;
    if( *pF > (F0 + *pBW * *pSC) ) return false;
    return true;
}
```

Déclaration

```
CREATE FUNCTION "inrange" (float8, float8, float8,
    float8, float8, float8) RETURNS bool
    AS 'path de la librairie.so' LANGUAGE 'C';
```

Exemple 2 : recherche de fréquences

Recherche multi fréquences

Le problème se complique si l'on veut chercher non pas à partir d'une fréquence mais à partir d'une liste F0, F1, F2, Fn, d'un nombre variable de valeurs, e.g. toutes les fréquences de transition d'une molécule donnée.

L'application CGI pourrait construire une requête utilisant la fonction de détection précédente et des clauses OR par fréquence, mais :

1. Très lourd en exécution, le OR est un opérateur coûteux en CPU.
2. On fait des calculs inutiles, e.g. la correction Doppler refaite à l'identique pour chaque fréquence de la liste.

Alternative

L'application CGI encode la liste sous forme d'une chaîne de texte, valeurs numériques séparées par des espaces, et appelle une fonction booléenne dédiée.

Exemple 2 : recherche de fréquences

```
/*
 * FREQUENCIES RANGE
 * -----------------
 * Parses a list of frequencies (text string with blank separated
 * floating point values) and check against an instrumental
 * configuration.
 *   F0 is a reference frequency plus offset.
 *   HW is the half bandwith.
 *   DC is a flag signaling whether frequencies are Doppler
 *   corrected or not.
 */

bool checklines(text *list, float8 *pF0, float8 *pHW,
    float8 *pSV, int4 DC)
{
    double Fmin, Fmax;
    unsigned size, maxi;
    const char* text;
    char data[128];

    /* Frequency interval */
    Fmin = *pF0 - *pHW;
    Fmax = *pF0 + *pHW;
```

Exemple 2 : recherche de fréquences

```
/* Doppler correction ? */
if( ! DC ) {
    double scale = (1 + *pSV / 3.E+8);
    Fmin /= scale;
    Fmax /= scale;
}

/* Init lines string data */
maxi = VARSIZE(list) - VARHDRSZ;
text = VARDATA(list);

/* Parse frequencies string */
while( maxi ) {

    /* Skip spacers */
    while( maxi && (*text == ' ') ) maxi--, text++;

    /* Move value */
    size = 0;
    while( maxi && (*text != ' ') ) {
        maxi--;
        data[size++] = *text++;
    }
```

Exemple 2 : recherche de fréquences

```
            /* Parsed data ? */
            data[size] = 0;
            if( size ) {
                double F = atof(data);
                if( (F >= Fmin) && (F <= Fmax) ) return true;
            }
        }

        /* No matches */
        return false;
    }
```

Exemple 3 : recherche de mots-clés

Un exemple de programmation répartie *frontend* / *backend*.

Le problème

On veut proposer une recherche par mots-clés sur des objets d'une base, articles, conférences, observations, etc., plus généralement tous objets disposant d'un attribut texte, notes, commentaires, résumé.

On veut également disposer d'une recherche acceptant des expressions booléennes.

Les besoins

- Assurer le traitement lexicographique usuel pour s'affranchir des majuscules / minuscules et des caractères accentués (utilisateurs anglo-saxons avec clavier US).
- Disposer d'un prédicat booléen efficace pour détecter la présence d'un mot, à chercher, dans un texte a priori quelconque (comparaisons lexicales C).
- Disposer d'un analyseur run-time performant pour traiter des expressions booléennes complètes (AND, OR, NOT) pour chaque *tuple*.

Exemple 3 : recherche de mots-clés

Traitement lexical

Convention de transformation ISO-8859-1 majuscules ou minuscules en US-ASCII minuscules.

Par exemple *Jérôme* sera recodé en *jerome*.

Encodage vocabulaire

Recodage d'un texte libre, par exemple une annonce de séminaire :

```
From the Double Pulsar to Magnetars: Observational constraints
for neutron star magnetospheres,
by Michael Kramer, Manchester University
and
Theory of pulsar magnetosphere: from radio to high-energy pulsed
emission, by Jérôme Pétri, Strasbourg University.
Salle T15
```

en un vocabulaire ordonné (avec recodage lexical) :

```
[[10T8 /constraints /double /emission energy /hennebelle high
/jerome /kramer /magnetars magnetospheres manchester michael
/neutron /observational /petri pulsar pulsed /radio /salle star
strasbourg /t15 theory /university ]
```

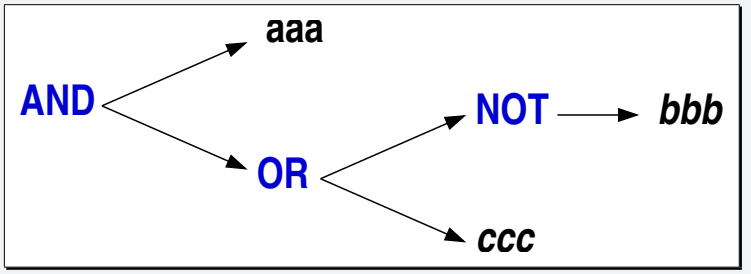
Exemple 3 : recherche de mots-clés

Traitement des expressions booléennes

`aaa AND ( NOT bbb OR ccc )`

La représentation linéaire habituelle des expressions booléennes (ou arithmétiques) est lourde à analyser, priorités d'opérateurs, parenthèses, associativité, analyseurs à décalage / réduction.

On gagne à n'analyser qu'une seule fois l'expression pour la transformer en une forme arborescente compilée, plus efficace à traiter en phase de détection. (Notation dite *polonaise*, postfixée ou mieux, préfixée dans le cas d'expressions booléennes.)



L'analyse est faite une seule fois à la préparation de la requête (réception d'arguments CGI depuis le client HTTP), la détection est à faire pour chaque tuple lors de l'extraction.

Exemple 3 : recherche de mots-clés

Répartition de la programmation

- **Encodage du vocabulaire : effectué uniquement lors de la création ou de l'édition d'un objet, à partir d'un attribut texte.**

  Fonction application (*frontend*), exécutée lors d'un INSERT ou UPDATE SQL, conservation du résultat dans la base, sous forme d'un attribut dédié en plus du texte original.

- **Compilation d'expression booléenne : effectuée pour chaque requête, lors de la réception des arguments CGI, transformation forme linéaire en forme arborescente.**

  Fonction *frontend*, compilation avant de construire la requête SQL.

- **Détection booléenne : effectuée pour chaque tuple au cours de l'extraction.**

  Fonction *backend* à deux arguments, expression compilée et attribut vocabulaire; retourne un résultat booléen à utiliser dans une clause WHERE.

- **Recodage lexical : fonction *frontend*, utilisée pour la construction des vocabulaires et la compilation des expressions (opérandes texte).**

Discussion sur les outils

Support fonctions backend dans les SGBD courants

DB2	Implémenté, sûrement très bien (on est chez IBM !), pas d'expérience personnelle.
Oracle	Très bien implémenté, exécution très efficace.
PostgreSQL	Idem Oracle.
MySQL	Implémentation médiocre et minimaliste, utilisable mais pas très utilisée.
MS-SQL server	Idem MySQL (rumeur publique, pas d'expérience personnelle).
Autres...	?

Stratégies de choix (quand on n'a pas d'argent)

En environnement académique, université recherche, on privilégie les SGBD gratuits PostgreSQL et MySQL. Le souhait (ou le besoin) de développer une application disposant d'une librairie *métier* de fonctions backend peut-être un élément de choix, à prendre en compte avec d'autres contraintes.

Portabilité des développements backend

Partielle ! Code applicatif portable (C ANSI), code interface spécifique. La migration d'une application d'un SGBD vers un autre, e.g. PostgreSQL vers Oracle, nécessite de reprendre l'interface avec le moteur SQL du SGBD cible.

Discussion sur les outils

Langage de programmation utilisé pour les applications CGI

Pour des raisons de confort de développement et de maintenance, il est très souhaitable qu'un développement réparti *frontend* / *backend* (cf. exemple mots-clés) soit fait avec le même langage. (Homogénéïté des prototypes, partage de symboles, etc.)

➤ Ce langage commun est du C, programmation *backend* oblige.

Selon le langage retenu pour l'écriture du programme CGI, on aura trois niveaux de confort (de développement) :

1. Programme CGI écrit en C, C++. C'est l'idéal, l'utilisation d'une librairie *frontend* en C est immédiate, un simple *link* suffit.
2. Programme CGI écrit dans un langage disposant d'une très bonne interface avec du C, e.g. Python, ou d'une ... interface avec du C, e.g. Java + JNI.

   Nécessite la construction d'un *wrapper*, explicite ou via des outils de génération, e.g. SWIG.
3. Programme CGI écrit dans un langage où l'interface avec des librairies C est plus problématique, e.g. Perl ou PHP.

   Implémentation plus problématique. Se poser la question de savoir si le jeu en vaut la chandelle ou s'il est préférable de modifier la stratégie.

Discussion sur les outils

Migration de code frontend vers code backend

Lorsque le langage de programmation de l'application CGI ne se prête pas simplement à l'utilisation de code C, on peut déplacer du code *frontend* vers le serveur.

Exemple précédent de l'encodage d'un texte libre en vocabulaire ordonné :

- Applicatif CGI en C, C++, Python.

  Implémentation *frontend* sans hésiter. On encode le texte (par exemple issu d'un `textarea` de formulaire HTML) en vocabulaire, on met à jour la base (INSERT ou UPDATE) avec les deux attributs.

```
UPDATE ...
SET texte = '...', vocab = '...' WHERE ident = ...;
```

- Applicatif CGI en Perl, PHP.
  1. On implémente la fonction d'encodage en *backend*.
  2. On met à jour la base avec uniquement le texte libre :

```
UPDATE ...
SET texte = '...' WHERE ident = ...;
```

  3. On fait une seconde mise à jour pour exécuter la fonction d'encodage :

```
UPDATE ...
SET vocab = encodage(texte) WHERE ident = ...;
```

  Le code applicatif CGI ne comporte alors plus que des requêtes SQL, sans besoin de *greffes* de librairies C. Tout langage convient.

Conclusion

Revoir l'introduction...