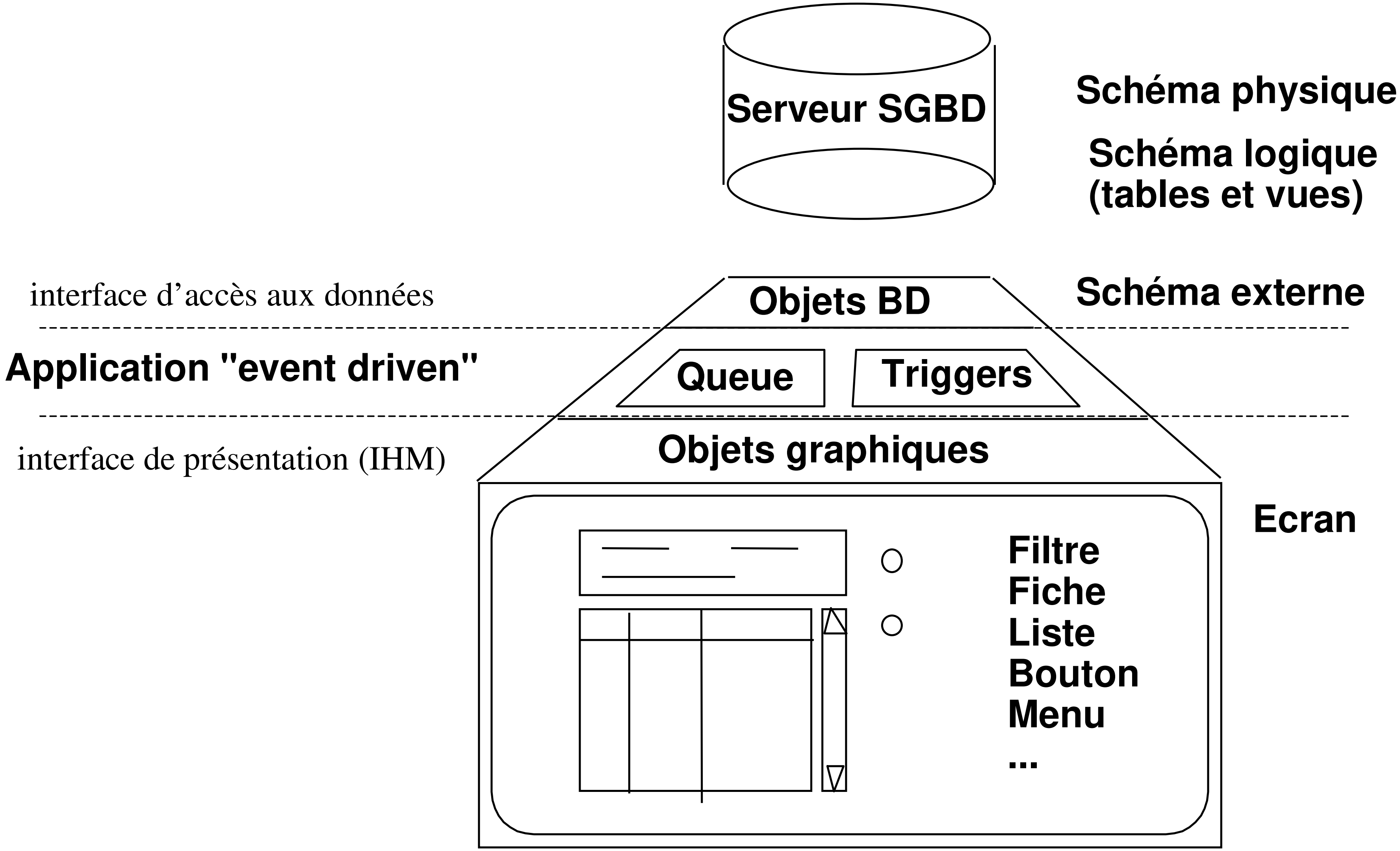
SQL dans les applications (1/2)



SQL dans les applications (2/2)

SQL89, SQL92, SQL99...

- Host Language Embedded SQL
  - ISO/IEC 9075-5:1999 SQL/Bindings
- Call Level Interface SQL
  - ISO/IEC 9075-3:1999 SQL/CLI
- Persistent Stored Modules SQL
  - ISO/IEC 9075-4:1999 SQL/PSM

C-Embedded SQL (1/4)

```
#include <stdio.h>
      EXEC SQL INCLUDE SQLCA.H ;      /* structures de communication C <=> SQL  */
 main(argc,argv)
 int argc;
 char *argv[];
 {
      char *answer[1];
      EXEC SQL BEGIN DECLARE SECTION ; /* variables communes au C et au SQL */
      VARCHAR db_connect_string[32];
      VARCHAR xx[32];
      VARCHAR yy[16];
      VARCHAR zz[16];
      EXEC SQL END DECLARE SECTION ;
                                                              /* à suivre */
```

C-Embedded SQL (2/4)

```
strcpy(db_connect_string, argv[1]);
EXEC SQL CONNECT : db_connect_string;
                    /* on suppose qu'il existe une relation:  plage(region, nomp, pollution */
EXEC SQL DECLARE C1 CURSOR FOR
SELECT nomp, pollution FROM plage WHERE region = :yy
FOR UPDATE OF pollution;
            /* on veut mettre à jour la pollution pour certaines plages de certaines régions */
while (printf("\nRégion? (\0 pour fin): ") && scanf("%s", zz) && strcmp(zz,'\0') != 0)
    {
    EXEC SQL OPEN C1 ;
    while (sqlca.sqlcode = 0) {                    /* la structure sqlca est dans SQLCA.H*/
          EXEC SQL FETCH C1 INTO :xx, :zz;
          printf("\n%32s, qui était %16s, devient-elle Alarmante? (O|N)", xx, zz) ;
          scanf("%s", answer) ;
                                                                        /* à suivre */
```

C-Embedded SQL (3/4)

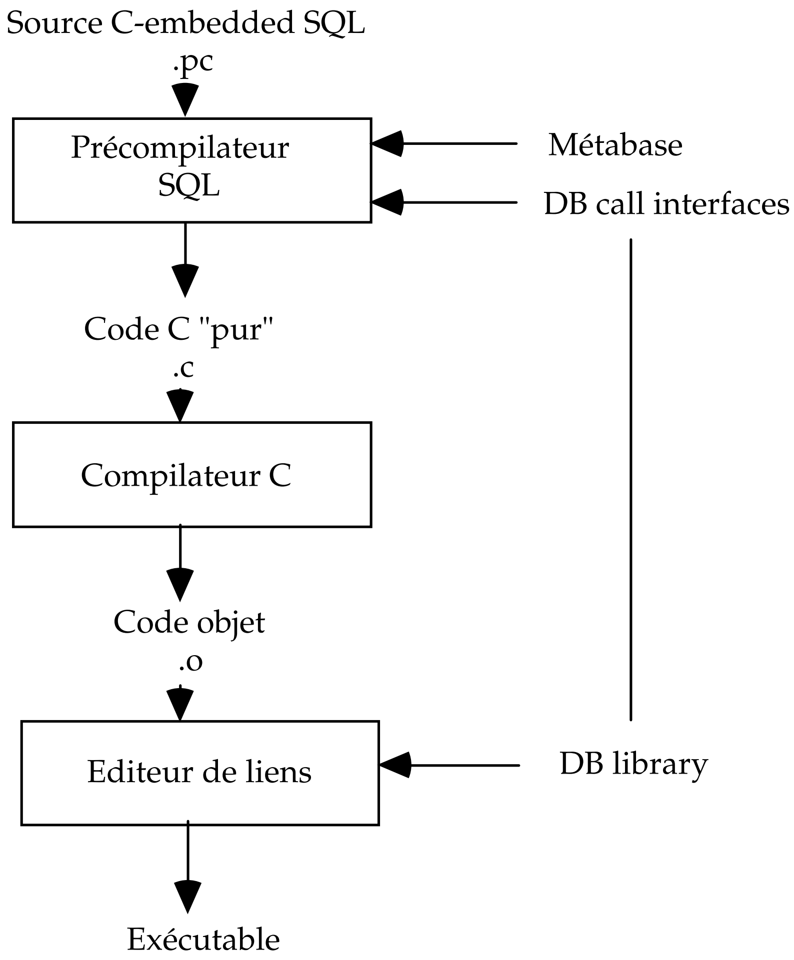
```
      if (answer == 'O')
      EXEC SQL UPDATE PLAGE
      SET pollution = 'Alarmante'
      WHERE CURRENT OF C1 ;
      } /* end of while cursor open */

   EXEC SQL CLOSE C1 ;
   EXEC SQL COMMIT WORK ;

    } /* end of while Région indiquée */

} /* end of main */
```

C-Embedded SQL (4/4)



Call Level Interface SQL (1/3)

```
/* Embedded-SQL */
EXEC SQL INCLUDE SQLCA;
void main() {
    EXEC SQL CONNECT TO user/passwd@t:serveur:base;
    EXEC SQL insert into messages value('Hello world');
    EXEC SQL commit;
}
```

Pourquoi le CLI ?

- précompiler vers une bibliothèque INDEPENDANTE du SGBD cible
  - une bibliothèque d 'interfaces universelle / SGBD SQL
- différer l'édition de lien dynamiquement à l'exécution du CONNECT
  - une bibliothèque d 'objets « linkables »(« DLL ») propre à chaque SGBD

Call Level Interface SQL (2/3)

/* Call Level Interface -SQL */

```
#include "sql.h"

HENV henv; HDBC hdbc; HSTMT hstmt;

void main() {

    SQLAllocEnv(&henv);

    SQLAllocConnect(henv, &hdbc);

    SQLConnect(hdbc,"t:serveur:base",SQL_NTS,"user",SQL_NTS,"passwd",SQL_NTS );

    SQLAllocStm (hdbc,&hstmt);

    SQLExecDirect(hstmt,"insert into messages value ('Hello world')",SQL_NTS);

    SQLFreeStmt(hstmt,SQL_DROP);

    SQLDisconnect(hdbc);

    SQLFreeConnect(hdbc);

    SQLFreeEnv(henv);

}
```

Call Level Interface SQL (3/3)

Le CLI permet

– l'imbrication « sans couture (seamless)» du SQL dans un langage hôte

classique (Microsoft ODBC)

ou orienté objet (Sun JDBC)

– le client-serveur SQL

(1ère motivation du SQLAccess Group dès 1992, de l 'X/Open DBTG dès 1993)

SQL décomposé en procédures donc RPC possible

N.B. les aspects connectivité et RDA s'ajoutent au SQL strict

Persistent Stored Modules SQL (1/2)

Exemple: PL/SQL d 'Oracle

```
DECLARE
  CURSOR c1 IS
    SELECT ename, sal FROM emp
    ORDER BY sal DESC;
  name     emp.ename%TYPE;
  salary   emp.sal%TYPE;
 BEGIN
DESC;
  OPEN c1;
  FOR i IN 1..5 LOOP
    FETCH c1 INTO name, salary;
    EXIT WHEN c1%NOTFOUND;
    INSERT INTO tmp VALUES(name, salary);
    COMMIT;
   END LOOP;
   CLOSE c1;
 END;
```

Returns the five highest paid employees from the emp table, and stores the result in tmp, a table created for transfer.

Output: SELECT * FROM tmp
ORDER BY earns

```
EMPLOYEE   EARNS
-------------------------
KING        5000
SCOTT       3000
FORD        3000
JONES       2975
BLAK        2850
```

Persistent Stored Modules SQL (2/2)

Les PSM permettent

– d'écrire des « routines » en SQL

ex: procPollution(region IN, plage IN

– d'écrire des « triggers »

(SI événement/données ALORS appel procédure)

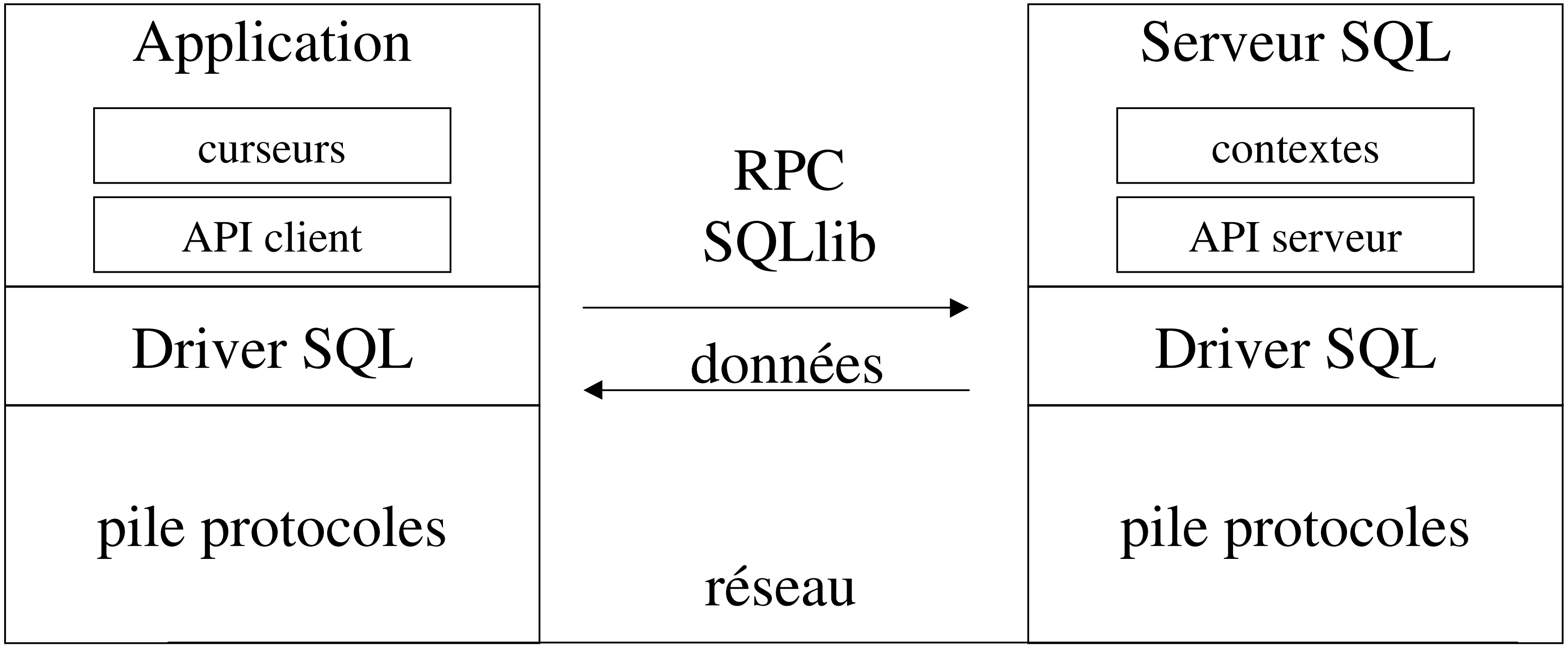
pour

- garder l'intégrité des données
- rendre les bases de données « actives »

– le client-serveur « de 2e génération »

avec beaucoup moins de messages échangés (1 seul RPC/fonction)

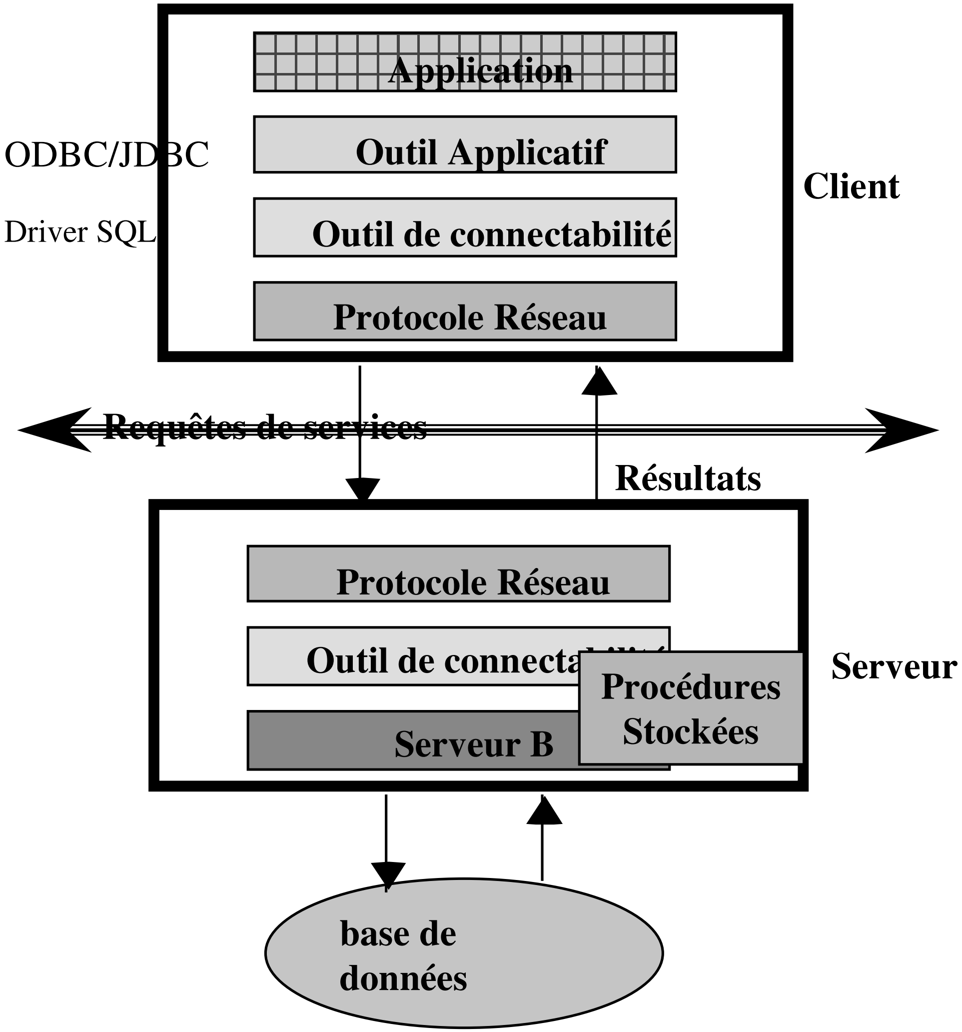
Le Client-Serveur SQL
« data shipping »



C/S de 2e génération (selon G.Gardarin)

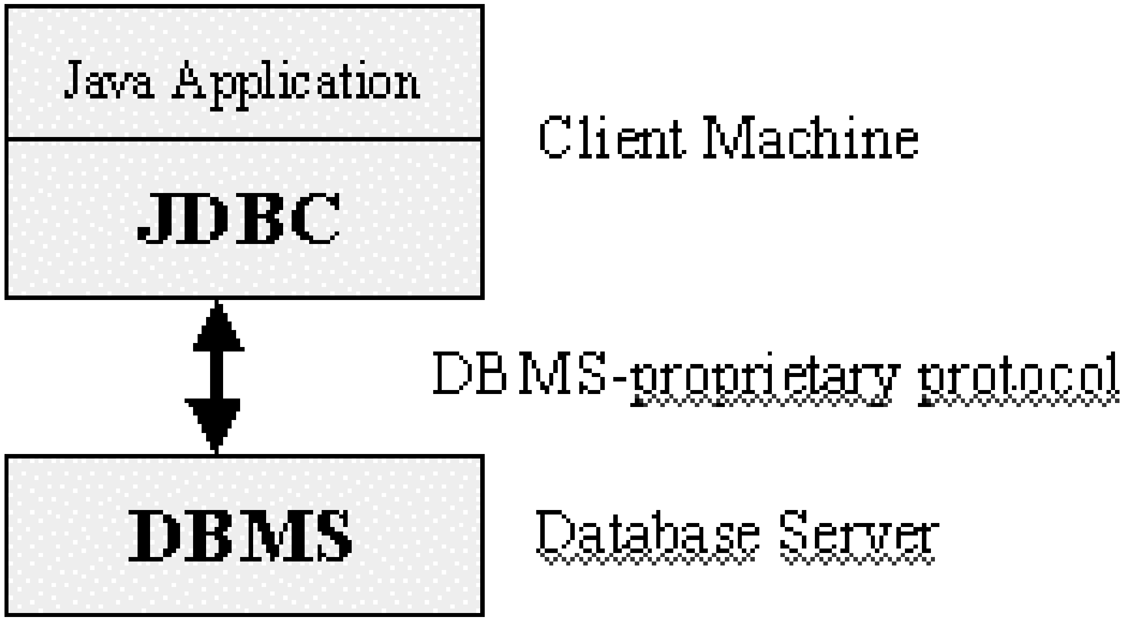
« function shipping »

- Procédure stockée
  - Procédure accomplissant une fonction de service sur les données
  - Exemple : Entrée ou sortie de stock
- Architecture orientée services plutôt que requêtes
  - Distribution des traitements
  - Peut être automatisée
- Évolution et passage à l'échelle
  - Possibilité de serveurs multiples, avec redondances
  - Possibilité de données privées sur les clients



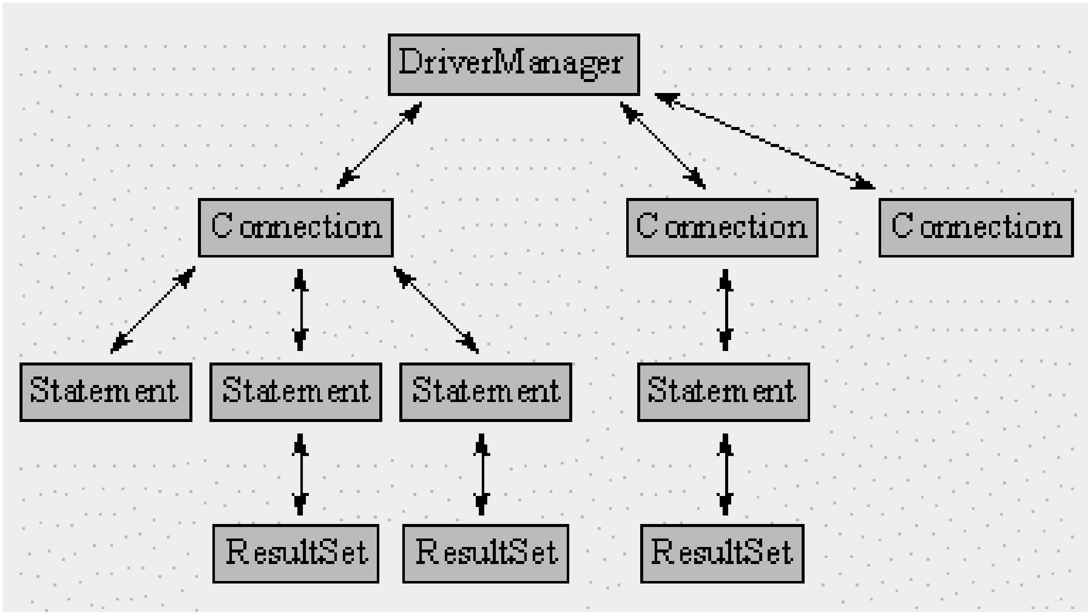
CLI SQL: JDBC (1/3)

JDBC entre application et driver SQL du SGBD



CLI SQL: JDBC (2/3)

La logique des héritages entre objets :



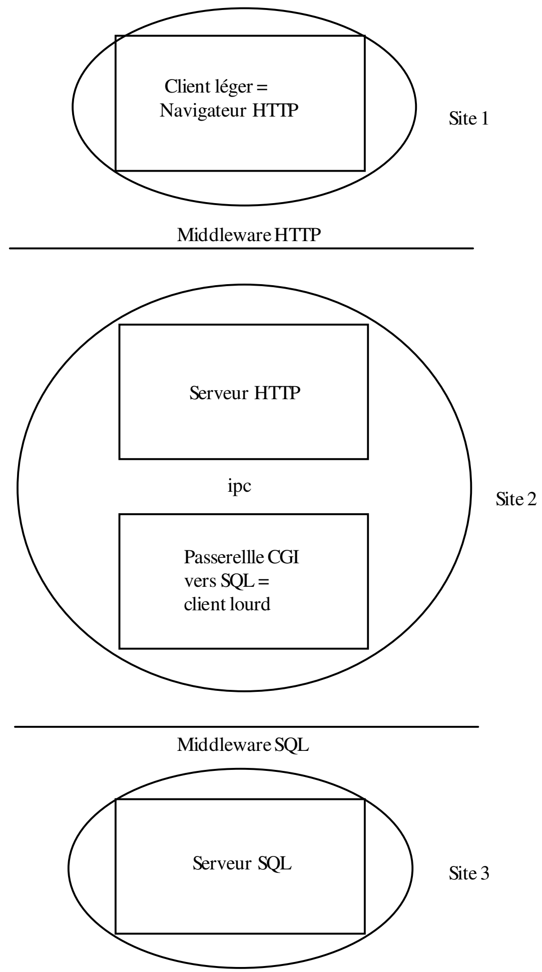
java.sql.DriverManager, …, java.sql.CallableStatement, java.sql.ResultSet

CLI SQL: JDBC (3/3)

```
import java.net.URL;
import java.sql.*;
class jdbc_query {
    public static void main(String argv[]) { // usage: java jdbc_query URL ...
        try {
        Class.forName("jdbc.odbc.JdbcOdbcDriver ");    // JDBC-ODBC bridge
        String url = argv[0];              // may be user=argv[1]; pwd=argv[2];
        Connection con = DriverManager.getConnection(url, …);
        Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery("select tablename from cat");
        while(rs.next()) {
                String name = rs.getString("tablename");
                System.out.println(name);     }
        stmt.close();
        con.close; } catch(Exception e) {System.out.println(e.getMessage()); }
    } // end main    } // end class
```

L'architecture « 3 rangs »

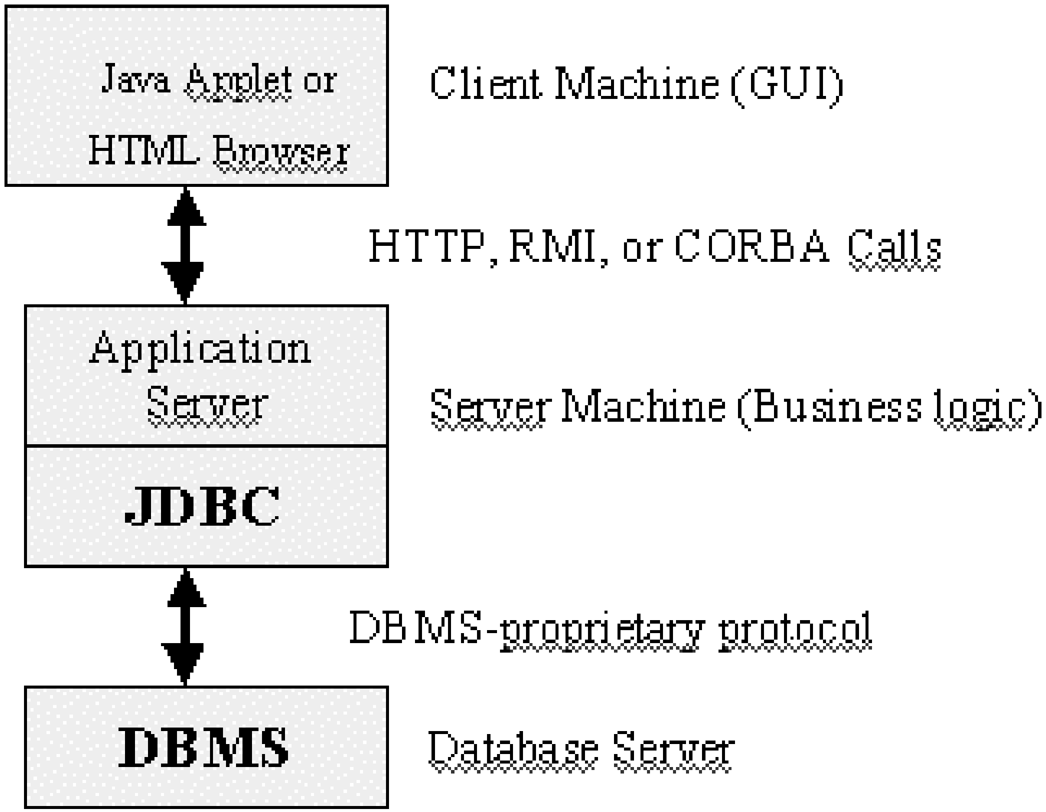
- Terminal: Navigateur
- *Client-Serveur HTTP*
- Frontal: Serveur + passerelle
- *Client-Serveur SQL*
- Dorsal: Serveur de données



Le SQL dans des CGI

- Formulaires spécialisés // actions spécialisées
  - scripts HTML + scripts (ou code source) CGI
- CGI en C et Embedded SQL (précompilé, compilé)
- CGI en C et SQL/CLI (ODBC) (compilé)
- CGI en PERL et SQL/CLI (interprété)
- CGI en Java et SQL/CLI (JDBC) (précompilé, interprété)

3-tiers JDBC



Java Applet or
HTML Browser
Client Machine (GUI)
HTTP, RMI, or CORBA Calls
Application
Server
JDBC
Server Machine (Business logic)
DBMS-proprietary protocol
DBMS
Database Server

Le SQL dans l'HTML

- But: écrire formulaires et actions dans le même script
- Solution: de nouvelles balises et une interprétation dynamique
  - propriétaire fermé : Oracle Web (// O2 Web)
  - propriétaire ouvert : SQL~Surfer (1994-98)
  - freeware : PHP/FI/mSQL (1995-)

    Personal Home Page tools / Form Interpreter / milliSQL

Le serveur d’applications PHP (1/2)

```
<html><head><title>My First PHP-SQL script!</title></head>
<body>
<?php
   $conn = Ora_Logon("login", "passwd") or die;
   $curs = Ora_Open($conn);
   $query = sprintf("select * from cat");
   Ora_Parse($curs, $query) or die;
   Ora_Exec($curs);
   $ncols = Ora_Numcols($curs);
   while (Ora_Fetch($curs)) {
        for ($i=0; $i<$ncols; $i++) {
        printf("%s = %s",ora_columnname($curs,$i),ora_getcolumn($curs,$i));
        printf("
        "); }     }
   Ora_Close($curs);
   Ora_Logoff($conn);
 ?>
</body>
</html>
```

Le serveur d'applications PHP (2/2)