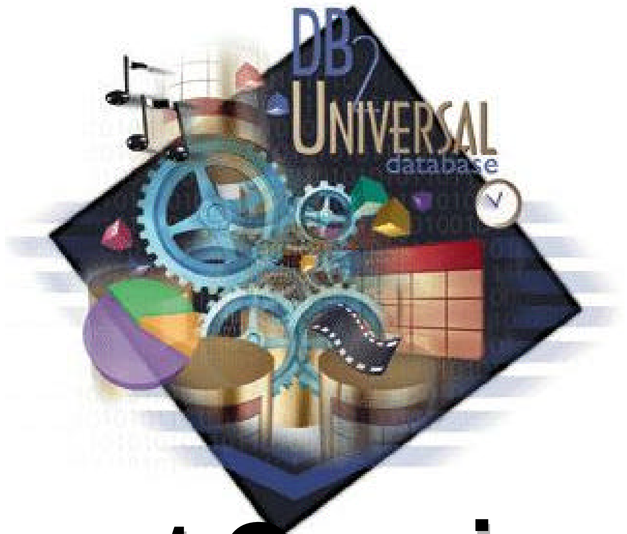
DB2 Universal Database Version 8.1



DB2 Application Development Overview

1. Static & Dynamic Embedded SQL
2. Call Level Interface (CLI)
3. DB2 APIs
4. Java Interfaces
5. Microsoft Data Objects (ADO, RDO, DAO)
6. Other Programming Tools (Perl DBI)

Unit Objectives

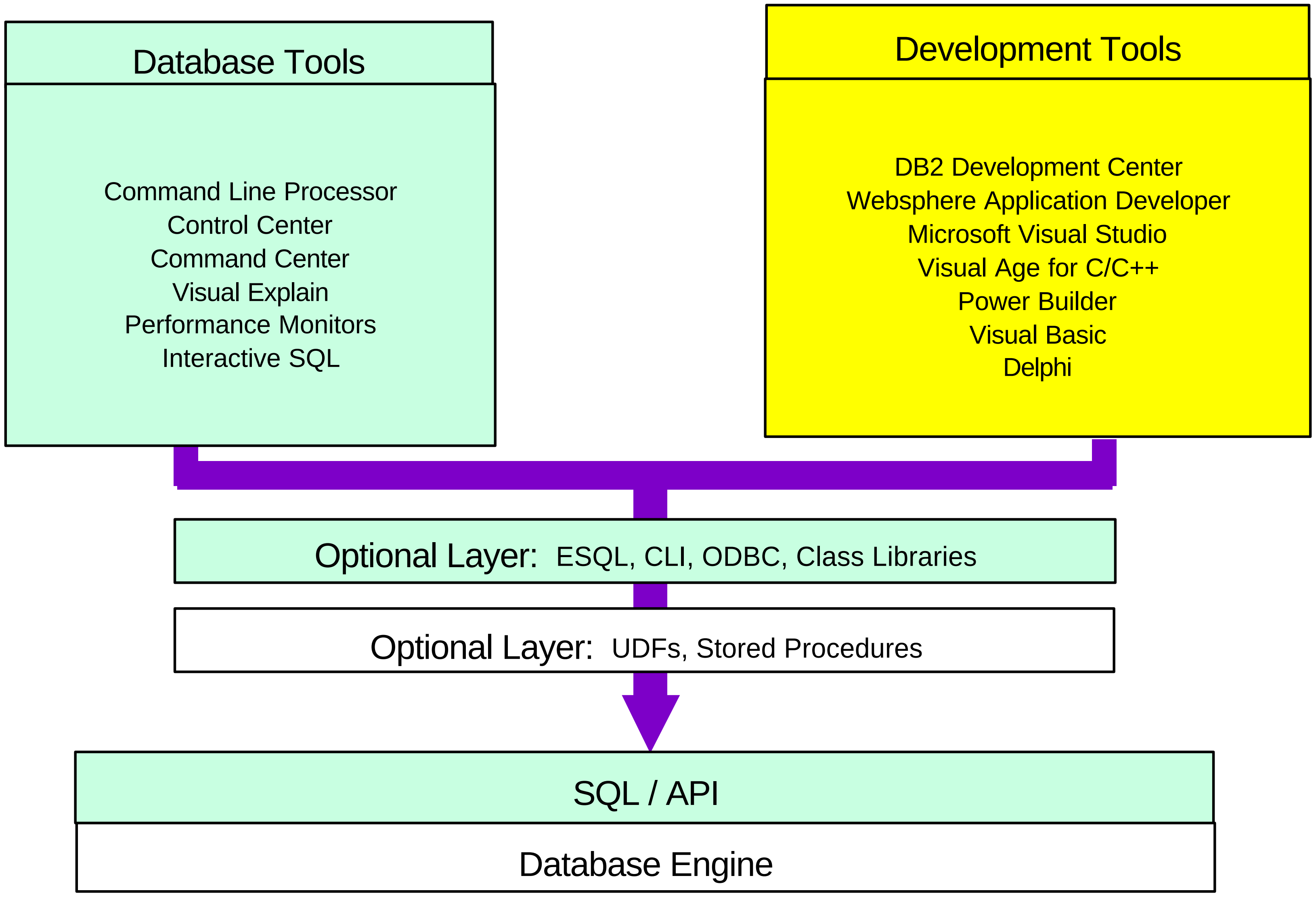
- Describe the application alternatives available to access DB2 data or request other DB2 functions

- List the benefits, and possible disadvantages of the various interfaces

Application Development Considerations

- Where is the database (Operating System)?
- What type of client application?
- What kind of application? (OLTP, DSS / OLAP e.g.)
- Does application perform single transaction across multiple database servers?
- How many & level of skill of application programmers?

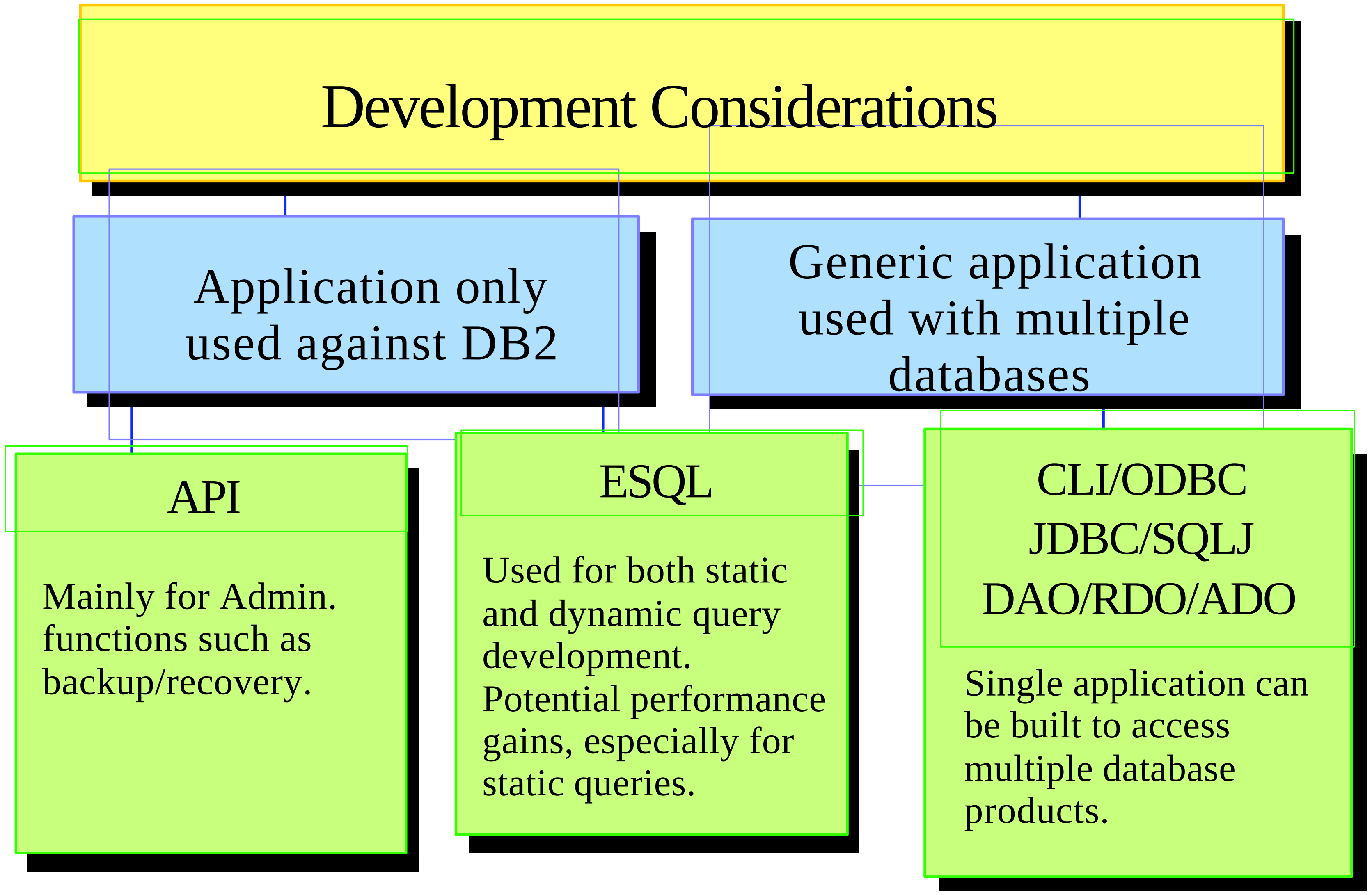
Accessing the DB2 Engine



Database Connect Types

- **DB2 provides two types of database connections**
  - **Type 1** : A transaction acts against a single database and must complete processing before connecting to a different database.
  - **Type 2** : A transaction may connect to multiple databases and can commit / rollback its changes to all databases at the same time. **Also Known as Distributed Unit of Work (DUOW)**
- **The connect type is set**
  - When the application is precompiled
  - For the CLP using the "set client" option

Which method to use?


Development Considerations
Application only used against DB2
Generic application used with multiple databases
API
ESQL
CLI/ODBC
JDBC/SQLJ
DAO/RDO/ADO

Mainly for Admin. functions such as backup/recovery.

Used for both static and dynamic query development. Potential performance gains, especially for static queries.

Single application can be built to access multiple database products.

DB2 Programming Methods

- **Embedded SQL**
  - Static
  - Dynamic
- **Call Level Interface (CLI) / ODBC**
- **Java**
  - JDBC
  - SQLJ
- **DB2 Application Programming Interfaces (APIs)**
- **Microsoft Data Objects (ADO, RDO, DAO)**
- **Other Interfaces and Tools**
  - Perl DBI

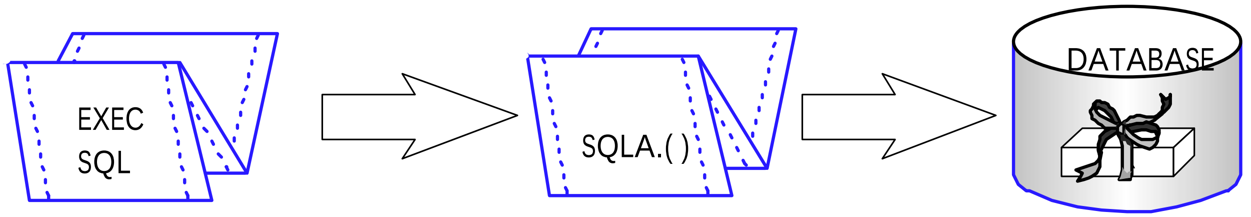
DB2 Universal Database
Version 8.1

DB2 Application Development Overview

1. Static & Dynamic Embedded SQL
2. Call Level Interface (CLI)
3. DB2 APIs
4. Java Interfaces
5. Microsoft Data Objects (ADO, RDO, DAO)
6. Other Programming Tools (Perl DBI)

Static Embedded SQL

- Application development involves combination of SQL with 3GL programming language
- When executed in a program, pre-defined SQL statements bound to a database as application packages



PROGRAM1.xxx
Source Code with
SQL statements:
Languages; C/C++,
COBOL, FORTRAN

PRECOMPILE

PROGRAM1.yyy
Modify Source Code
with DB2 calls
Ready to Compile/Link

COMPILE/LINK

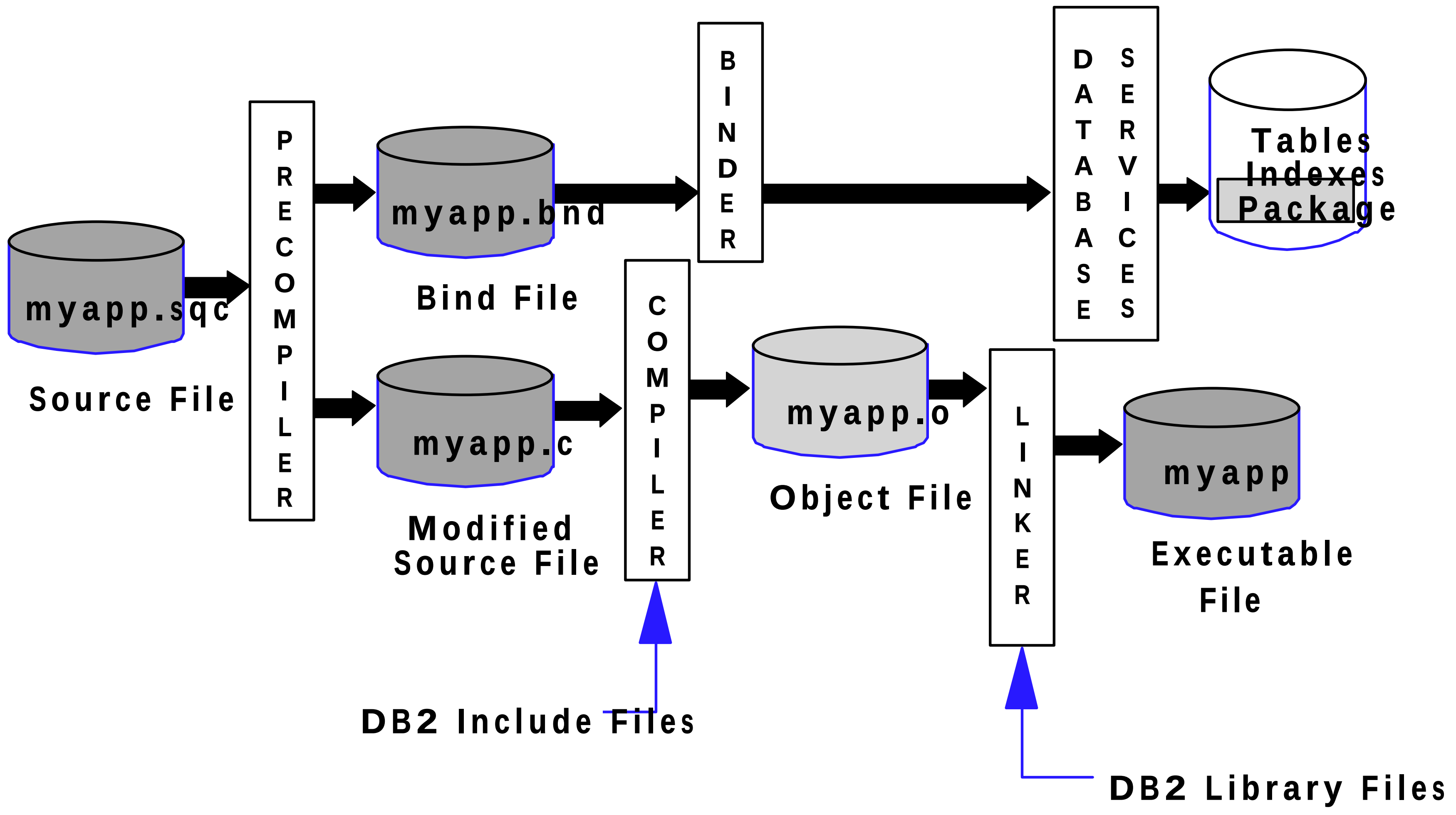
PROGRAM1
Package Stored in
the Database
during BIND

**BIND/EXECUTE
(STATIC)**

Embedded SQL Steps

TASK	*STATEMENT*
Connect to DB	db2 connect to eddb
Prepare Source Code	db2 prep myapp.sqc bindfile
Compile and Link	icc myapp.c db2api.lib
Bind to DB	db2 bind myapp.bnd blocking all
Execute Object Code	myapp > myapp.out

Program Preparation Steps



Static Embedded SQL

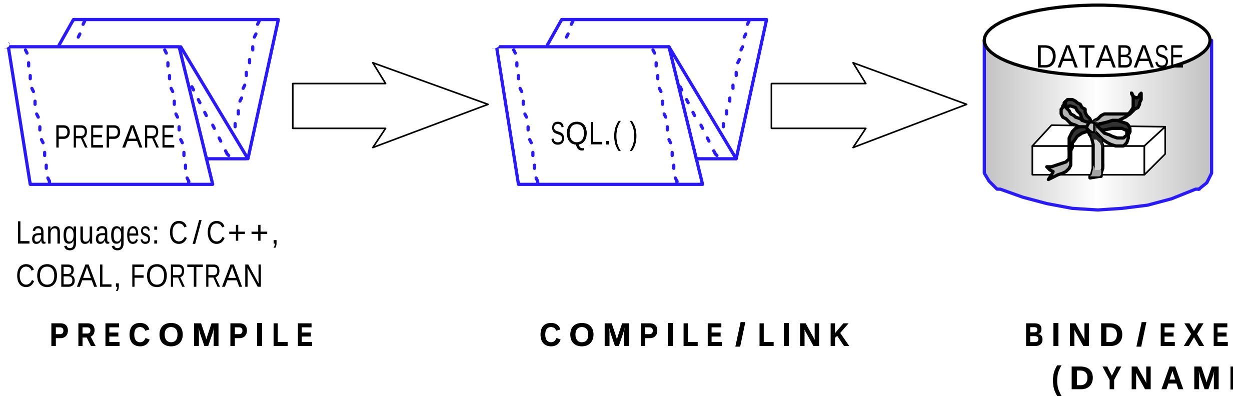
- **Languages supported:**
  - C/C++
  - Java (SQLJ)
  - COBOL
  - FORTRAN
- **Advantages**
  - Optimized packages available at run-time
  - Static SQL statements are persistent
- **Disadvantages**
  - Must know what SQL is needed
  - Requires precompile

Sample - Static Embeded SQL

```
#include <stdio.h>
#include <sql.h>
#include <sqlenv.h>
#include <sqlda.h>
#include <sqlca.h>
EXEC SQL INCLUDE SQLCA;
int main(int argc, char *argv[])
{
    EXEC SQL BEGIN DECLARE SECTION;
        char firstname[13];
        char dbAlias[15] ;
        char user[15] ;
        char pswd[15] ;
 EXEC SQL END DECLARE SECTION;
    /* checks the command line arguments */
    strcpy(dbAlias, argv[1]);
    strcpy(user, argv[2]);
    strcpy(pswd, argv[3]);
    /* initialize the embedded application */
    EXEC SQL CONNECT TO :dbAlias USER :user USING :pswd;

    EXEC SQL SELECT FIRSTNME INTO :firstname
           FROM employee
            WHERE LASTNAME = 'JOHNSON';
    if (sqlca.sqlcode != 0 && sqlca.sqlcode != 100)
          printf("Error");
    else
           printf( "First name = %s\n", firstname );
    EXEC SQL CONNECT RESET ;
}
```

Dynamic Embedded SQL

- Application development includes precompile / compile / link phase
- Binding or select of access plan done AT PROGRAM EXECUTION
- Database objects do not have to exist at precompile only run-time



Dynamic Embedded SQL

- **Languages supported:**
  - C/C++
  - Java (SQLJ)
  - COBOL
  - FORTRAN
  - REXX
- **Advantages**
  - Provide execution of dynamic SQL statements
  - Database objects do not have to exist before runtime
  - More flexible than static SQL statements
- **Disadvantages**
  - Take more time to execute

Sample - Dynamic Embeded SQL

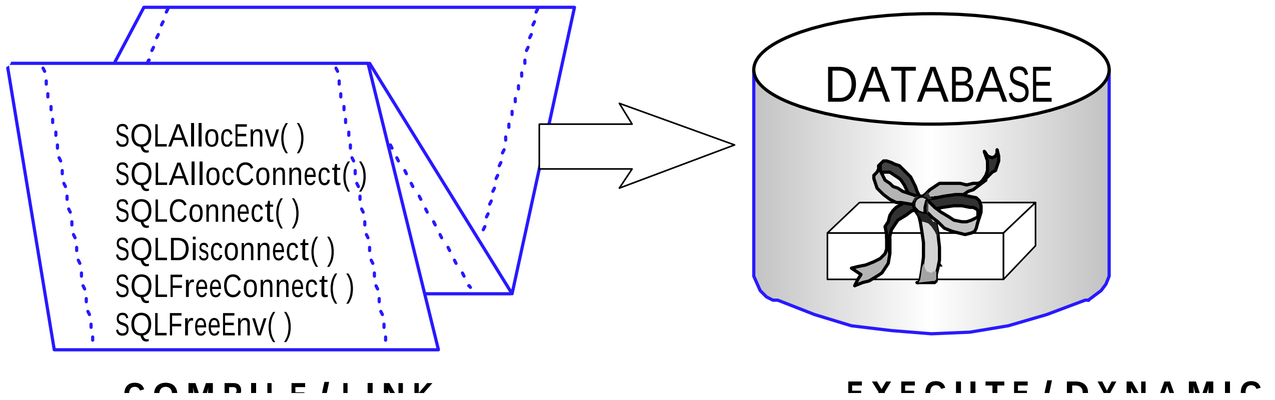
```
#include <stdio.h>
#include <sql.h>
#include <sqlenv.h>
#include <sqlda.h>
#include <sqlca.h>
EXEC SQL INCLUDE SQLCA;
int main(int argc, char *argv[])
{
    EXEC SQL BEGIN DECLARE SECTION;
      char dbAlias[15] ;
      char user[15] ;
      char pswd[15] ;
      char  table_name[25];
      char  st[80];
      char  parm_var[19];
      char firstname[13];
   EXEC SQL END DECLARE SECTION;
    /* checks the command line arguments */
    strcpy(dbAlias, argv[1]);
    strcpy(user, argv[2]);
    strcpy(pswd, argv[3]);
    /* initialize the embedded application */
    EXEC SQL CONNECT TO :dbAlias USER :user USING :pswd;
   strcpy( st, "SELECT tabname FROM syscat.tables" );
   strcat( st, " WHERE tabname <> ? ORDER BY 1" );
   EXEC SQL PREPARE s1 FROM :st;
   EXEC SQL DECLARE c1 CURSOR FOR s1;
   strcpy( parm_var, "STAFF" );
   EXEC SQL OPEN c1 USING :parm_var;
   EXEC SQL OPEN c1;
    do {
     EXEC SQL FETCH c1 INTO :table_name;
     if (SQLCODE != 0) break;
     printf( "Table = %s\n", table_name );
   } while ( 1 );
   EXEC SQL CLOSE c1;
   EXEC SQL COMMIT;

    EXEC SQL SELECT FIRSTNME INTO :firstname
           FROM employee
            WHERE LASTNAME = 'JOHNSON';
    if (sqlca.sqlcode != 0 && sqlca.sqlcode != 100)
          printf("Error");
    else
           printf( "First name = %s\n", firstname );

   EXEC SQL CONNECT RESET ;
}
```

DB2 Universal Database Version 8.1

DB2 Application Development Overview

1. Static & Dynamic Embedded SQL
2. Call Level Interface (CLI)
3. DB2 APIs
4. Java Interfaces
5. Microsoft Data Objects (ADO, RDO, DAO)
6. Other Programming Tools (Perl DBI)

- A callable SQL interface, Similar to ODBC.
  - DB2 specific extensions have been added to help the application programmer specifically exploit DB2 features
- Database resources directly accessed from programming language using Application Programming Interfaces (APIs) rather than embedded SQL calls.
- Dynamic SQL Application development with no precompile or static binding required
- Based on Microsoft's ODBC & X/Open Call Level Interface specifications
- More portable than Embedded SQL.

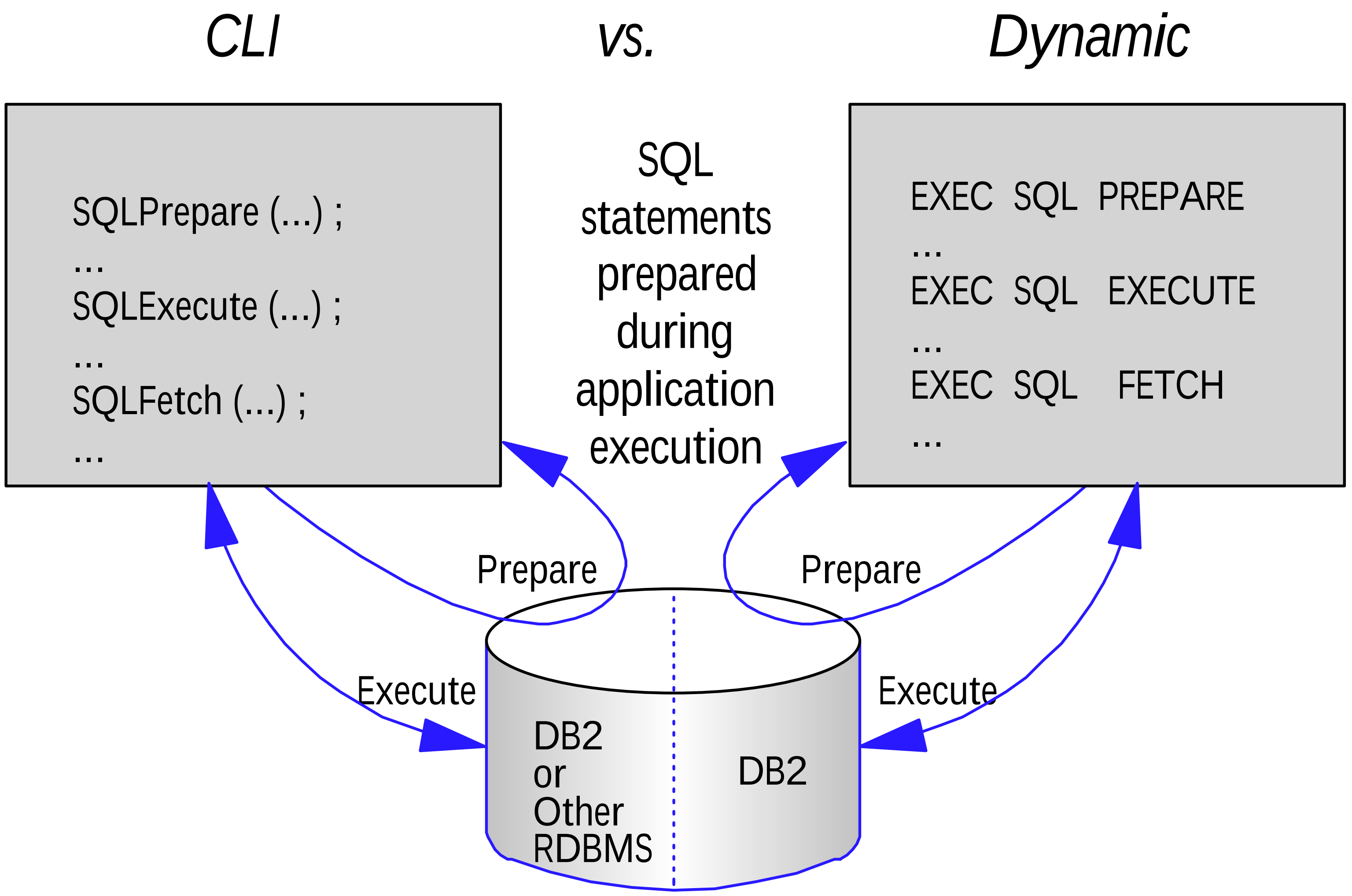


Embedded SQL

CLI differs from Embedded SQL as follows:

- No user defined cursors (internal by DB2).
- No precompile or application packages.
  - Set of CLI packages bound once for all CLI/ODBC applications.
- No COMMIT / ROLLBACK - use SQLEndTran() to commit or rollback a transaction.
- No SQLDA structure required.
- No SQLCA structure required - use SQLSTATES and return codes with special error handing APIs.
- No Host Variables, use Parameter Markers (it is Dynamic).
- Arrays for FETCH and INSERT multiple rows.
- Scrollable bi-directional cursors.
- Predefined APIs for catalog table queries.

Call Level Interface VS Embedded SQL



Call Level Interface

Advantages

- Precompile / Bind is not required
- Current statistics are used
- Can store and retrieve sets of data
- Application can be multi-threaded
- Scrollable Cursors
- Easy application porting

Limitations

- Confined to C / C++
- Dynamic bindings are slower

Setting up your CLI environment

Install the correct DB2 product package

- DB2 Application Development client or any other DB2 client.

BIND the DB2 CLI bind files against the DB2 server

- The CLI/ODBC driver will autobind on the first connection to the database, provided the user has the appropriate privilege or authorization.
- Different set of bind files for each DB2 family platform.
  - db2cli.lst (UNIX, NT, OS/2)
  - ddcsvm.lst (VM)
  - ddcsvse.lst (VSE)
  - ddcsmvs.lst (MVS or OS/390)
  - ddcs400.lst (AS/400)
- BINDADD authority on database is required.

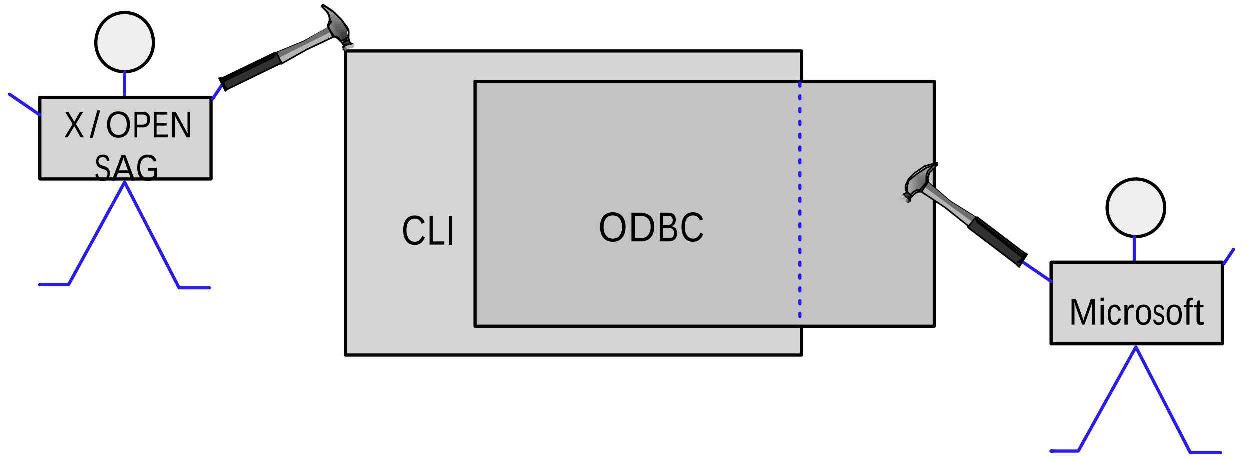
Configure CLI specifically for your environment

- db2cli.ini file
- provides specific tuning for an application, allows changes to the default behavior and enables workarounds for known situations.
- Keywords documented in the *DB2 UDB Call Level Interface Guide and Reference.*

Database must be cataloged

ODBC (Open Database Connectivity)

- The DB2 CLI driver also acts as an ODBC driver when loaded by an ODBC driver manager. It conforms to level 2 of ODBC 2.0, and level 1 of ODBC 3.0. In addition, it also conforms to various ODBC 3.0 level 2 interface conformance items (202, 203, 205, 207, 209, and 211). Information regarding ODBC support and level 2 interface conformance items is provided in the CLI Guide and Reference Appendix C, DB2 CLI and ODBC.
- Can Develop Using ODBC Tools
  - Lotus Approach
  - Microsoft Access
  - Microsoft VisualBasic

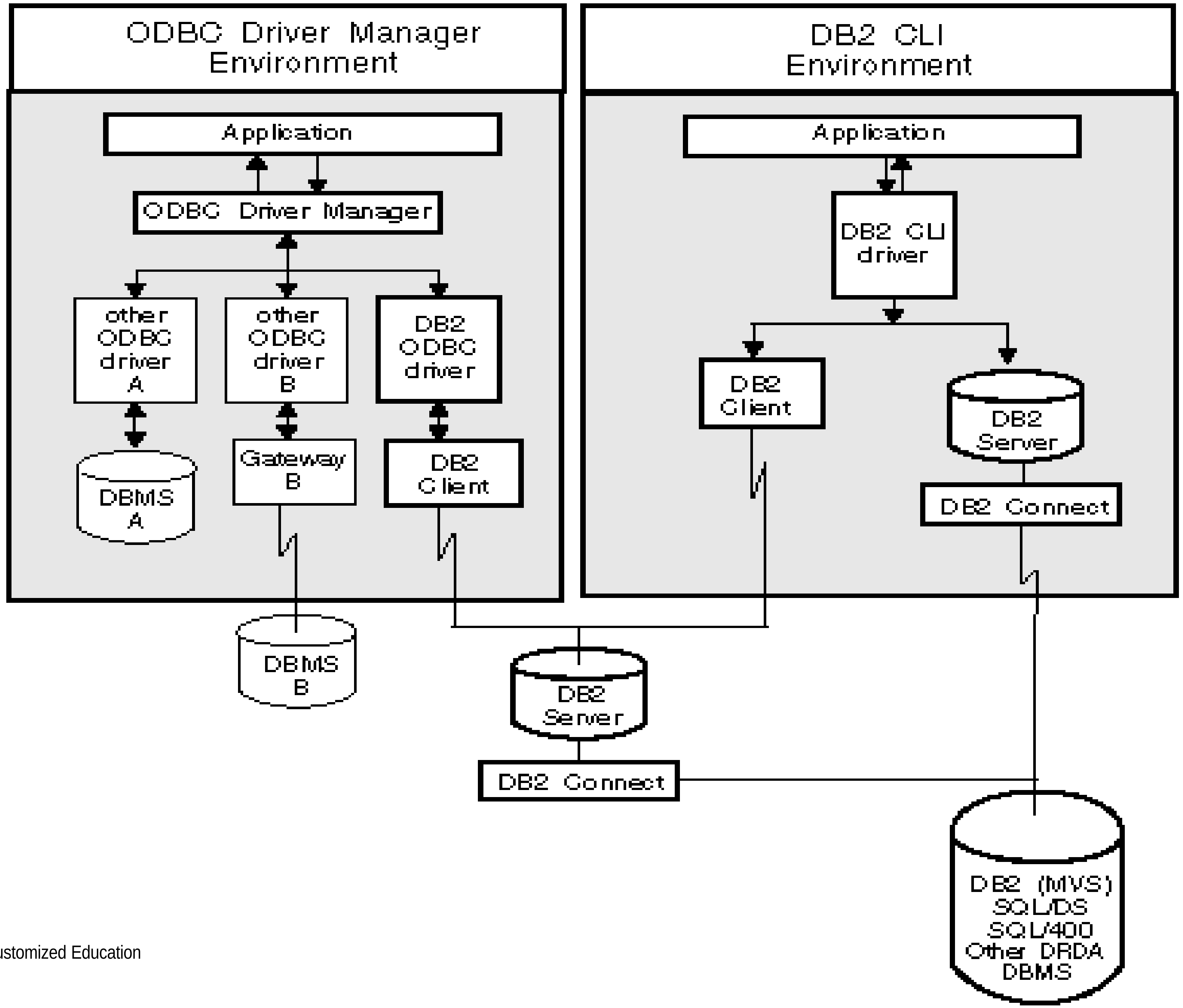


Accessing a DB2 database via ODBC

To access a DB2 database from ODBC, the following is required on the DB2 UDB client where the ODBC application will execute.

- 1. DB2 client must be installed (Application Development, Administration, Run-time)
  - Option exists to install DB2 client code with DB2 server.
- 2. The DB2 database and, optionally, the DB2 server must be catalogued.
- 3. An ODBC Driver Manager must be installed.
  - Provided by Microsoft for all Microsoft operating systems.
  - On Unix, install from an ODBC client application or ODBC SDK.
- 4. The DB2 UDB ODBC driver must be installed and registered with the ODBC driver manager.
  - "IBM DB2 ODBC Driver" is the correct name on Windows platforms.
  - On UNIX, the ODBC driver and databases are specified in the odbc.ini and odbcinst.ini files in the home directory of the user running the ODBC application.
- 5. The DB2 database must be registered as an ODBC data source with the driver manager.
  - Accomplished through DB2 Client Configuration Assistant, the appropriate ODBC Administration tool, or manually.

How does ODBC differ from CLI?



CLI Programming: basic example

```
int main( ) {
    SQLHANDLE henv;
    SQLHANDLE hdbc;
           /* allocate an environment handle */
    SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv );
           /* allocate the connection handle */
    SQLAllocHandle( SQL_HANDLE_DBC, henv, &hdbc );
           /* connect to the db2cert data source */
    SQLConnect( hdbc, "db2cert", SQL_NTS, "userid", SQL_NTS,
                "password", SQL_NTS );

           /*********   Start Transaction Processing  ***********/
           /* allocate statement handle, execute statement, etc.*/
            /*********   End Transaction Processing **************/

              /* disconnect from database */
    SQLDisconnect( hdbc ) ;
             /* free the connection handle */
    SQLFreeHandle( SQL_HANDLE_DBC, hdbc ) ;
              /* free environment handle */
    SQLFreeHandle( SQL_HANDLE_ENV, henv ) ;
     return ( SQL_SUCCESS ) ;
}
```

CLI Programming: Execute statement directly sample

From dbuse.c:

```
int StmtExecDirect( SQLHANDLE hdbc)
{   SQLRETURN   sqlrc = SQL_SUCCESS;
    int         rc = 0;
    SQLHANDLE   hstmt ;  /* statement handle */

    SQLCHAR *   stmt1 = ( SQLCHAR * ) "CREATE TABLE table1(col1 INTEGER)" ;
    SQLCHAR *   stmt2 = ( SQLCHAR * ) "DROP TABLE table1" ;

    printf("\nUSE THE CLI FUNCTIONS\n");
    printf("-SQLSetConnectAttr\n-SQLAllocHandle\n");
    printf("-SQLExecDirect\n-SQLFreeHandle\n");
    printf("TO EXECUTE SQL STATEMENTS DIRECTLY:\n");

    /* set AUTOCOMMIT on */
    sqlrc = SQLSetConnectAttr( hdbc,SQL_ATTR_AUTOCOMMIT,
                               (SQLPOINTER)SQL_AUTOCOMMIT_ON, SQL_NTS) ;
    DBC_HANDLE_CHECK( hdbc, sqlrc);

    /* allocate a statement handle */
    sqlrc = SQLAllocHandle( SQL_HANDLE_STMT, hdbc, &hstmt ) ;
    DBC_HANDLE_CHECK( hdbc, sqlrc);

    /* execute directly statement 1*/
    printf("\n    Execute directly %s.\n", stmt1);
    sqlrc = SQLExecDirect( hstmt, stmt1, SQL_NTS ) ;
    STMT_HANDLE_CHECK( hstmt, sqlrc);

    /* execute directly statement 2 */
    printf("    Execute directly %s.\n", stmt2);
    sqlrc = SQLExecDirect( hstmt, stmt2, SQL_NTS ) ;
    STMT_HANDLE_CHECK( hstmt, sqlrc);

    /* free the statement handle */
    sqlrc = SQLFreeHandle( SQL_HANDLE_STMT, hstmt ) ;
    STMT_HANDLE_CHECK( hstmt, sqlrc);

    return(rc);
}
```

DB2 Universal Database Version 8.1

1. Static & Dynamic Embedded SQL
2. Call Level Interface (CLI)
3. DB2 APIs
4. Java Interfaces
5. Microsoft Data Objects (ADO, RDO, DAO)
6. Other Programming Tools (Perl DBI)

DB2 APIs

- **Supply by DB2**
- **Use to directly manipulate DB2 instances and database (divided by functionality)**
  - Backup/Recovery
  - DB Control
  - DB Manager Control
  - DB Directory Mgt.
  - Node Directory Mgt.
  - Network Support
  - Database Configuration
  - DB Monitoring
  - Operational Utilities
  - Data Utilities
  - General Appl. Programming
  - Appl. Preparation
  - Remote Server Utilities
  - Table Space Mgt.
- **Provide access to some function not available in embedded SQL or through the CLI.**
- **Can be used in embedded SQL or CLI application (sqlaintpl( ), e.g.)**
- **C, C++, COBOL, FORTRAN (must be**

DB2 APIs

Advantages

- Advanced feature of DB2 can be used (table space administration, e.g.)
- No precompile or bind required
- Utilize Host Programming Languages

Disadvantage

- Requires host language compiler / linker
- Can be more difficult to use
- Cannot issue SQL statements
- Not easily ported to other database servers

Sample: DB2 API

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sqle819a.h>
#include <sqlutil.h>
#include <sqlenv.h>
#include "utilapi.h"

int DbDrop(void);

int main(int argc, char *argv[])
{
  int rc = 0;
  char nodeName[SQL_INSTNAME_SZ + 1];
  char user[USERID_SZ + 1];
  char pswd[PSWD_SZ + 1];

  /* check the command line arguments */
  rc = CmdLineArgsCheck2(argc, argv, nodeName, user, pswd);
  if (rc != 0)
  {
    return rc;
  }

  printf("\nTHIS SAMPLE SHOWS HOW TO CREATE/DROP A DATABASE.\n");

  /* attach to a local or remote instance */
  rc = InstanceAttach(nodeName, user, pswd);
  if (rc != 0)
  {
    return rc;
  }

  rc = DbDrop();

/* detach from the local or remote instance */
  rc = InstanceDetach(nodeName);
  if (rc != 0)
  {
    return rc;
  }

  return 0;
} /* main */

int DbDrop(void)
{
  struct sqlca sqlca;
  char dbLocalAlias[SQL_ALIAS_SZ + 1];

  printf("\n-----------------------------------------------------------");
  printf("\nUSE THE DB2 API:\n");
  printf("  sqledrpd -- DROP DATABASE\n");
  printf("TO DROP A DATABASE:\n");

  /* drop a database  */
  strcpy(dbLocalAlias, "dbcreate");
  printf("\n  Drop a [remote] database and uncatalog it locally.\n");
  printf("    local database alias: %s\n", dbLocalAlias);

  /* drop database */
  sqledrpd(dbLocalAlias, &sqlca);
  DB2_API_CHECK("Database – Drop");

  return 0;
} /* DbDrop */
```

DB2 Universal Database Version 8.1

DB2 Application Development Overview

1. Static & Dynamic Embedded SQL
2. Call Level Interface (CLI)
3. DB2 APIs
4. Java Interfaces
5. Microsoft Data Objects (ADO, RDO, DAO)
6. Other Programming Tools (Perl DBI)

Java Interfaces (JDBC and SQLJ)

Programming language used to develop applications and applets that access and manipulate data in DB2 databases

- Java applications (requires DB2 Runtime Client)
- Java applets (does not require DB2 components on the client)
- Java servlets
- UDFs and stored procedures

JDBC Support

- DB2 supports Sun Microsystem's Java Database Connectivity (JDBC) API via driver
- JDBC API provide standard way to access databases from Java code

SQLJ Support

- embedded SQL for Java
- static SQL
- standard developed by SQLJ Consortium

Develop user defined functions and stored

Java Interfaces (JDBC and SQLJ)

Advantages

- Increased portability to other database systems and operating platforms
- Easy access to database across the Internet from multiple client platforms
- Object-oriented application development and data access model

Disadvantages

- Must have Java programming skills
- Can be slower since Java is interpreted

Java Database Connectivity (JDBC) API

Purpose: access relational database in Java

- Connect to database
- Execute SQL Statements
- Process the results

Specs: http://java.sun.com/products/jdbc/

- Specifies minimum SQL92-Entry

Industry standard by Sun

- 1996: JDK 1.0.2 + JDBC1.0
- 1997: JDK 1.1 (JDBC1.22), part of the Core API package
- 1998/1999: JDBC 2.0/JDBC 2.1

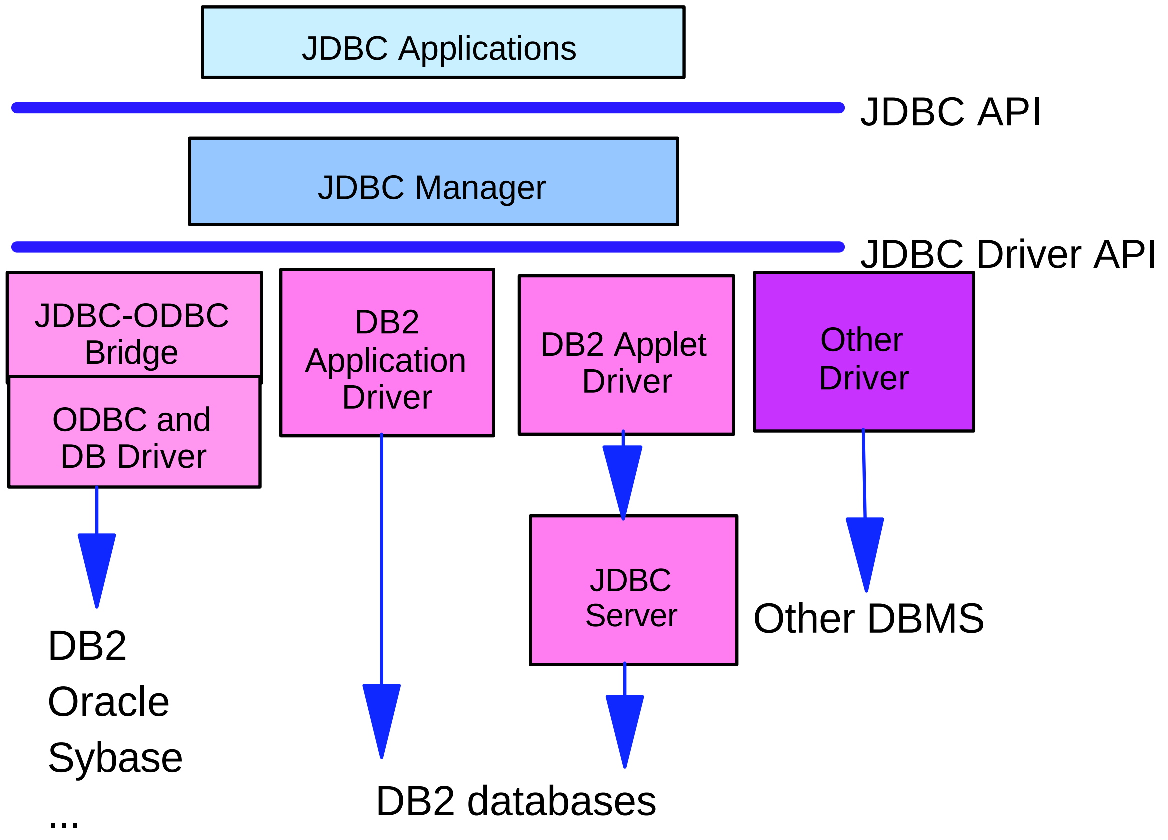
OO version of ODBC

- Object representation of handles
- Dynamic SQL queries-No precompile, bind steps

JDBC Driver Manager and Drivers

- **DriverManager from Sun (java.sql)**
  - Management layer between application and JDBC driver
  - Keeps track of available drivers
  - Establishes connections
- **To load a JDBC driver**
  - To use Class.forName("drivername");
  - To update "jdbc.drivers" system property
- **JDBC Driver registers itself with DriverManager**
- **JDBC Driver implements Standard Interfaces**
- **Connection URL syntax**
  - jdbc:<subprotocol>:<subname>

JDBC DriverManager and Drivers



Types of JDBC Drivers

Type 1: The JDBC-ODBC bridge

- JDBC Access via most ODBC drivers, from JavaSoft
- Vendor neutral, DB2, Oracle, Sybase, Informix
- Available on WIN NT/95, Sun Solaris

Type 2: A native-API partly-Java driver

- Converts JDBC calls into calls on the client API for DBMS
- Some binary code be loaded on each client machine

Type 3: A net-protocol all-Java driver

- Translates JDBC calls first into DBMS-independent net protocol
- The Server then translates it to DBMS protocol
- Net server middleware is able to connect its all-Java clients to many different databases

Type 4: A native-protocol all-Java driver

- Converts JDBC calls into the network protocol used by DBMS's protocol
- Uses proprietary protocols, DBMS specific

Types of JDBC Drivers

- TYPE 2 - "app" driver (COM.ibm.db2.jdbc.app.DB2Driver)
  - db2java.zip
  - J2EE 1.3 Certified & recommended for WebSphere Application Server
  - DB2 client required
  - **example URL: "jdbc:db2:sample"**
- TYPE 3 - "net" driver (COM.ibm.db2.jdbc.net.DB2Driver)
  - db2java.zip
  - Requires listener process at server & DB2 Client not required
  - **example URL: "jdbc:db2://host.mydomain.com:5678/sample"**
- TYPE 4 (com.ibm.db2.jcc.DB2Driver)
  - db2jcc.jar
  - Native DRDA
  - DB2 Client not required
  - **example URL: "jdbc:db2://host.mydomain.com:50000/sample"**

Creating JDBC Application / Applet

- import the JDBC DriverManager package / classes (java.sql)
- load the appropriate JDBC driver(s)
  - the JDBC-ODBC bridge driver: sun.jdbc.odbc.JdbcOdbcDriver
  - DB2 application driver: COM.ibm.db2.jdbc.app.DB2Driver
  - DB2 applet driver: COM.ibm.db2.jdbc.net.DB2Driver
- connect to a database using JDBC URL (defined in JDBC spec)
  - the JDBC-ODBC URL -- jdbc:odbc:dbname
  - DB2 application driver URL -- jdbc:db2:dbname
  - DB2 applet driver URL -- jdbc:db2://servername:portno/dbname
- pass SQL statements to the database
- receive the results (example: via ResultSet)
- close the connection/statement/resultset

Sample: JDBC Application

```
import java.sql.*;
class DB2Appl {

  static {
    try {
      // register the driver with DriverManager
      // The newInstance() call is needed for the sample to work with
      // JDK 1.1.1 on OS/2, where the Class.forName() method does not
      // run the static initializer. For other JDKs, the newInstance
      // call can be omitted.
      Class.forName("COM.ibm.db2.jdbc.app.DB2Driver").newInstance();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  public static void main(String argv[]) {
    Connection con = null;

    // URL is jdbc:db2:dbname
    String url = "jdbc:db2:sample";

    try {
      if (argv.length == 0) {
        // connect with default id/password
        con = DriverManager.getConnection(url);
        }
      else if (argv.length == 2) {
        String userid = argv[0];
        String passwd = argv[1];

        // connect with user-provided username and password
        con = DriverManager.getConnection(url, userid, passwd);
        }
      else {
        System.out.println("\nUsage: java DB2Appl [username password]\n");
        System.exit(0);      }
```

```
// retrieve data from the database
System.out.println("Retrieve some data from the database...");
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("SELECT * from employee");

System.out.println("Received results:");

// display the result set
// rs.next() returns false when there are no more rows
while (rs.next()) {
  String a = rs.getString(1);
  String str = rs.getString(2);

  System.out.print(" empno= " + a);
  System.out.print(" firstname= " + str);
  System.out.print("\n");
}

rs.close();
stmt.close();

// update the database
System.out.println("\n\nUpdate the database... ");
stmt = con.createStatement();
int rowsUpdated = stmt.executeUpdate("UPDATE employee set firstnme = 'SHII

System.out.print("Changed "+rowsUpdated);

if (1 == rowsUpdated)
  System.out.println(" row.");
else
  System.out.println(" rows.");

stmt.close();
con.close();
} catch( Exception e ) {
e.printStackTrace();     }  }}
```

Sample: JDBC Applet

```
import java.sql.*;
import java.awt.*;
import java.applet.Applet;

public class DB2Applt extends Applet {

  static {
    try {
      // register the driver with DriverManager
      // The newInstance() call is needed for the sample to work with
      // JDK 1.1.1 on OS/2, where the Class.forName() method does not
      // run the static initializer. For other JDKs, the newInstance
      // call can be omitted.

      Class.forName("COM.ibm.db2.jdbc.net.DB2Driver").newInstance();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  Connection con;

  public void init() {

    try {
      // get parameter values from the html page
      String server = getParameter("server");
      String port = getParameter("port");

      // construct the URL ( sample is the database name )
      String url = "jdbc:db2://"+server+":"+port+"/sample";

      String userid = getParameter("userid");
      String password = getParameter("password");

      // connect to database with userid and password
      con = DriverManager.getConnection(url, userid, password );

    } catch( Exception e ) {
      e.printStackTrace();
    }
```

```
public void paint(Graphics g) {
  try {
    // retrieve data from database
    g.drawString("First, let's retrieve some data from the database...", 10, 10);

    Statement stmt = con.createStatement();
    ResultSet rs = stmt.executeQuery("SELECT * from employee");
    g.drawString("Received results:", 10, 25);

    // display the result set
    // rs.next() returns false when there are no more rows
    int y = 50;
    int i = 0;
    while (rs.next() && (i<2)) {
      i++;
      String a= rs.getString(1);
      String str = rs.getString(2);
      String oneLine = " empno= " + a + " firstname= " + str;
      g.drawString(oneLine, 20, y );
      y = y + 15;

    }
    stmt.close();

    // update the database
    g.drawString("Now, update the database...", 10, 100);
    stmt = con.createStatement();
    int rowsUpdated = stmt.executeUpdate("UPDATE employee set firstnme = '

    // display the number of rows updated
    String msg = "Updated " + rowsUpdated;

    if (1 == rowsUpdated)
      msg = msg +" row.";
    else
      msg = msg +" rows.";
    y = y + 40;
    g.drawString(msg, 20, y);

    stmt.close();
  } catch( Exception e ) {
```

SQLJ: Embedded SQL for Java

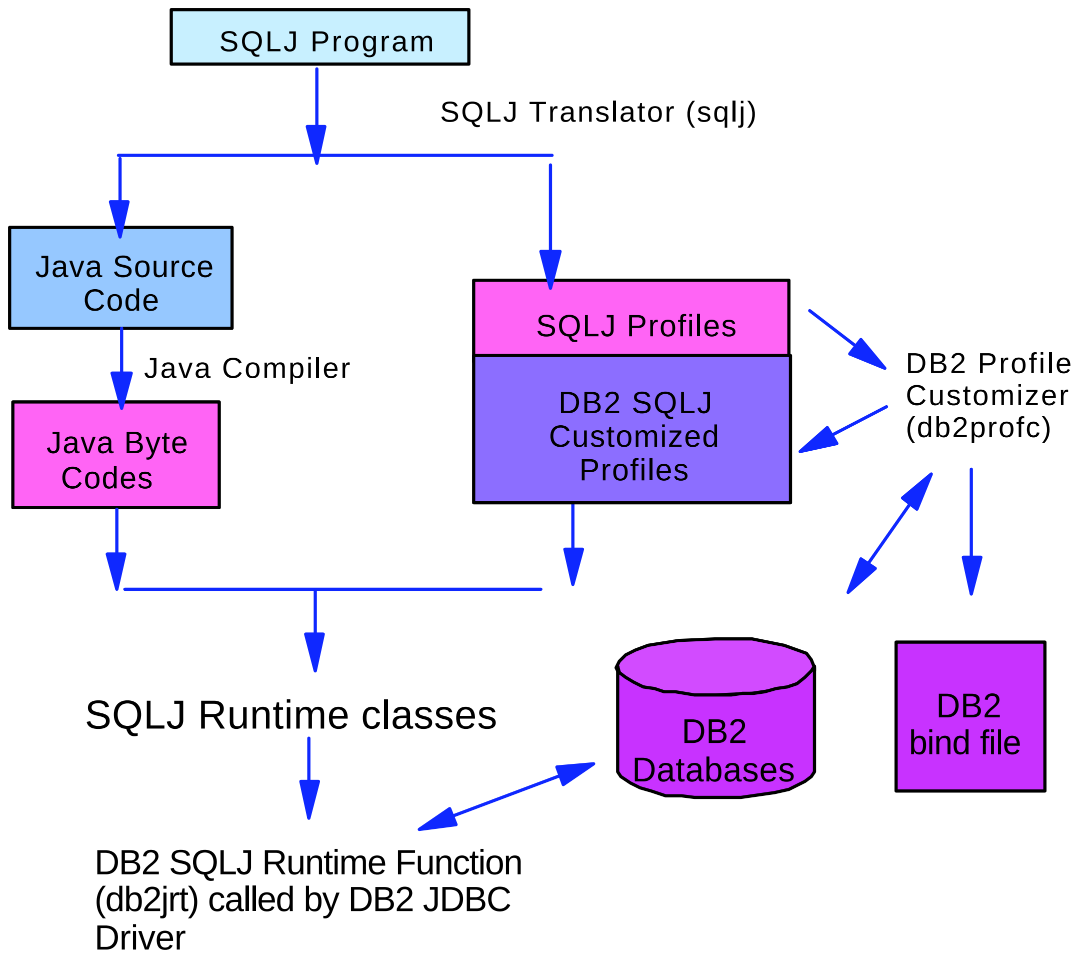
SQLJ standard developed by SQLJ Consortium

- (SQLJ Consortium now includes IBM, Oracle, Tandem, Sybase, Informix, Microsoft and Sun)

SQJ vs. JDBC

- SQLJ supports only static SQL constructs
- SQLJ relies upon JDBC for support of dynamic SQL
- Similar tradeoffs of static vs. dynamic for SQLJ vs. JDBC
  - less flexible at run-time
  - allows error checking at development time
  - pre-compilation of SQL
  - may improve execution time
  - provides opportunity for certain query optimizations
  - SQLJ programs may be smaller than JDBC application

Running an SQLJ Program



Sample: SQLJ

```
import java.sql.*;
import sqlj.runtime.*;
import sqlj.runtime.ref.*;

class Static
{  static
   {  try
      {  Class.forName ("COM.ibm.db2.jdbc.app.DB2Driver").newInstance ();
      }
      catch (Exception e)
      {  System.out.println ("\n  Error loading DB2 Driver...\n");
         System.out.println (e);
         System.exit(1);
      }
   }

   public static void main(String argv[])
   {  try
      {  System.out.println ("  Java Static Sample");

         String url = "jdbc:db2:sample";       // URL is jdbc:db2:dbname
         Connection con = null;

         // Set the connection                /* :rk.3:erk. */
         if (argv.length == 0)
         {  // connect with default id/password
            con = DriverManager.getConnection(url);
         }
         else if (argv.length == 2)
         {  String userid = argv[0];
            String passwd = argv[1];

            // connect with user-provided username and password
            con = DriverManager.getConnection(url, userid, passwd);
         }
         else
         {  throw new Exception("\nUsage: java Static [username password]
         }

         // Set the default context
         DefaultContext ctx = new DefaultContext(con);
         DefaultContext.setDefaultContext(ctx);


         String firstname = null;

         #sql { SELECT FIRSTNME INTO :firstname
                FROM employee
                WHERE LASTNAME = 'JOHNSON' } ;   /* :rk.4:erk. */


         System.out.println ("First name = " + firstname);
      }
      catch( Exception e )
      {  System.out.println (e);
      }
   }
}
```

DB2 Universal Database
Version 8.1

DB2 Application Development Overv

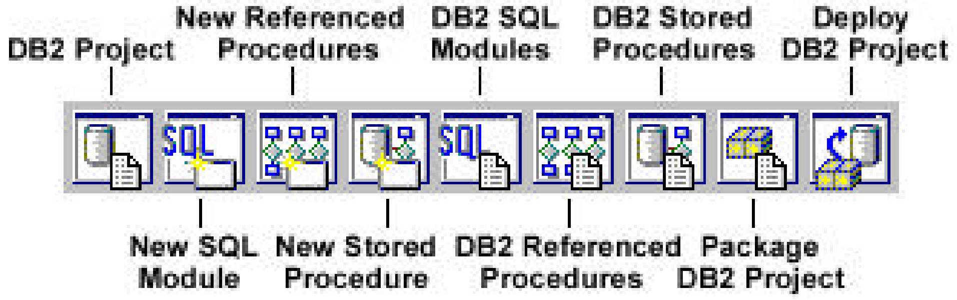
1. Static & Dynamic Embedded SQL
2. Call Level Interface (CLI)
3. DB2 APIs
4. Java Interfaces
5. Microsoft Data Objects (ADO, RDO, DAO)
6. Other Programming Tools (Perl DBI)

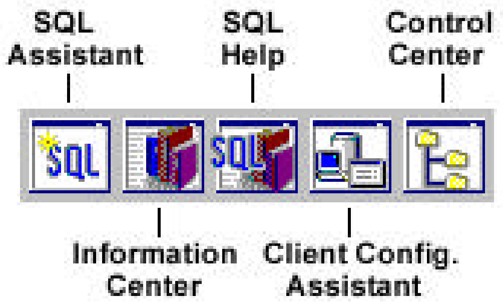
Microsoft Data Objects

- **Support for Data Access Object (DAO), Remote Data Object (RDO), and ActiveX Data Object (ADO).**
- **Supported through DB2 UDB's ODBC (CLI) Driver, through OLE:ODBC bridge, or through native OLE DB driver.**
- **Advantages**
  - Provide standardized programming model independent of data source.
- **Disadvantages**
  - Data objects available on Microsoft Windows platforms only.

Windows Integration

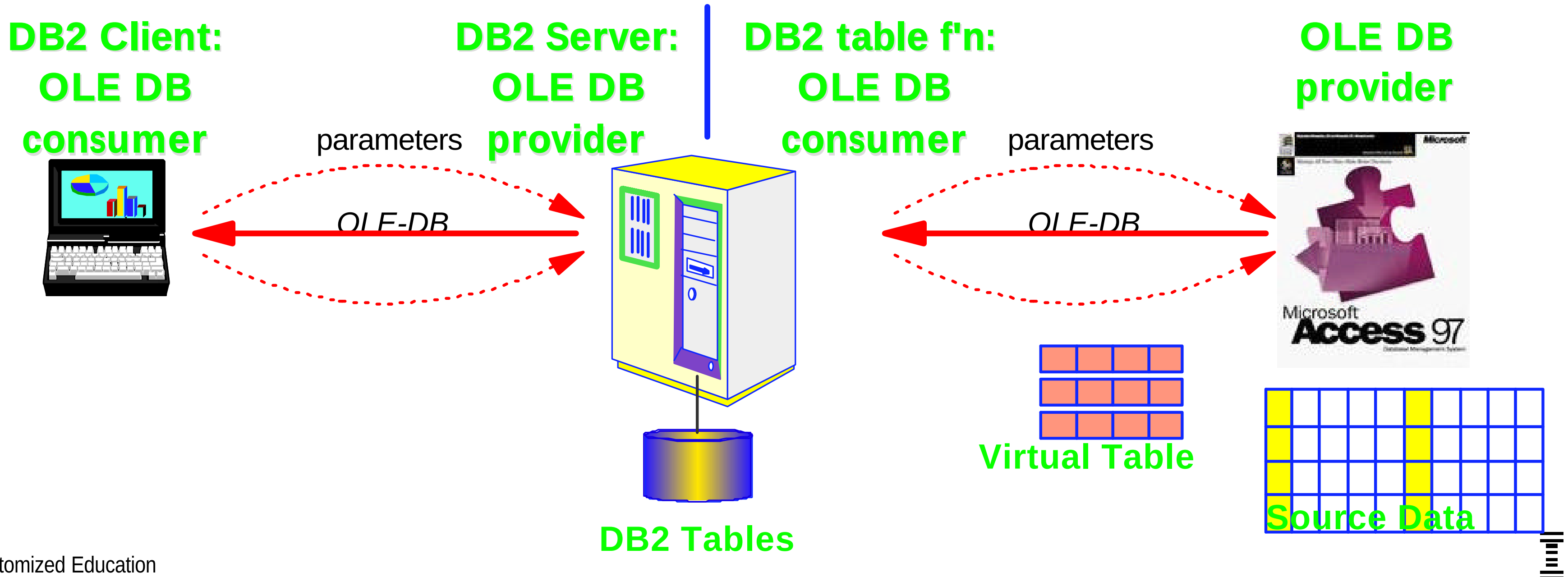
- **DB2 client support for OLE DB 2.0**
- **Integrated support for OLE stored procedures**
- **Visual Studio Integration**
  - Plugins for Visual C++ and Visual Basic
  - Improved and expanded DB2 UDB Samples & Examples - Included with DB2 Application development client
- **Light Directory Access Protocol (LDAP) on Windows 2000**
- **Extended userid support**
- **Support for Microsoft Transaction Server (MTS)**





OLE DB Applications

- **DB2 client support for OLE DB 2.0**
  - Native, high-performance implementation
  - Supports applications using ADO API:
    - Visual C++, Visual Basic, and C/C++ apps that use ADO APIs
- **Integrated support for OLE stored procedures**
  - Makes it easy to deploy stored procedures that use OLE automation (e.g. Visual Basic stored procedures)OLE DB Consumer Support (Windows Only)



DB2 Universal Database Version 8.1

DB2 Application Development Overview

1. Static & Dynamic Embedded SQL
2. Call Level Interface (CLI)
3. DB2 APIs
4. Java Interfaces
5. Microsoft Data Objects (ADO, RDO, DAO)
6. Other Programming Tools (Perl DBI)

Other Programming Tools - Perl DBI

- APIs for database access from the Perl Language.
- Standardized programming interface independent of underlying database system.
- DB2 UDB support is through the DBD::DB2 driver which sits on top of the DB2 CLI driver.

Unit Summary

- Describe the application alternatives available to access DB2 data or request other DB2 functions
  - Embedded SQL
  - CLI
  - DB2 APIs
  - Java
  - Microsoft Data Objects
  - Other programming tools
- List the benefits, and possible disadvantages of the various interfaces