



DB2 Application Development overview

IBM Information Management Cloud Computing Center of Competence
IBM Canada Lab

Agenda

- **DB2 Application Development overview**
- **Server-side development**
 - ▶ **Stored Procedures**
 - ▶ **User-defined functions**
 - ▶ **Triggers**
- **Client-side development**
 - ▶ **Embedded SQL**
 - ▶ **Static vs. Dynamic SQL**
 - ▶ **CLI/ODBC**
 - ▶ **JDBC / SQLJ / pureQuery**

Supporting reading material & videos

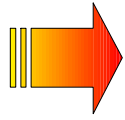
- **Reading materials**

- Getting started with DB2 Express-C eBook
 - Chapter 14: Introduction to DB2 application development
- Getting started with IBM Data Studio for DB2 eBook
 - Chapter 5: Developing SQL Stored Procedures
 - Chapter 7: Developing user-defined functions
- Getting started with DB2 Application Development eBook
 - Chapter 3, section 3.6: Triggers: The big picture

- **Videos**

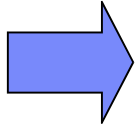
- db2university.com course AA001EN
 - Lesson 12: DB2 application development

Agenda



- **DB2 Application Development overview**
- Server-side development
 - ▶ Stored Procedures
 - ▶ User-defined functions
 - ▶ Triggers
- Client-side development
 - ▶ Embedded SQL
 - ▶ Static vs. Dynamic SQL
 - ▶ CLI/ODBC
 - ▶ JDBC / SQLJ / pureQuery

DB2 Application Development Overview



Server-side development (at the DB2 database server):

- ▶ Routines (Stored Procedures, UDFs)
- ▶ Database objects (Triggers)

■ **Client-side development (at the client):**

- ▶ May require a DB2 client or driver to be installed
- ▶ Database applications (in C/C++, .NET, Cobol, Java, etc)

05/30/2011

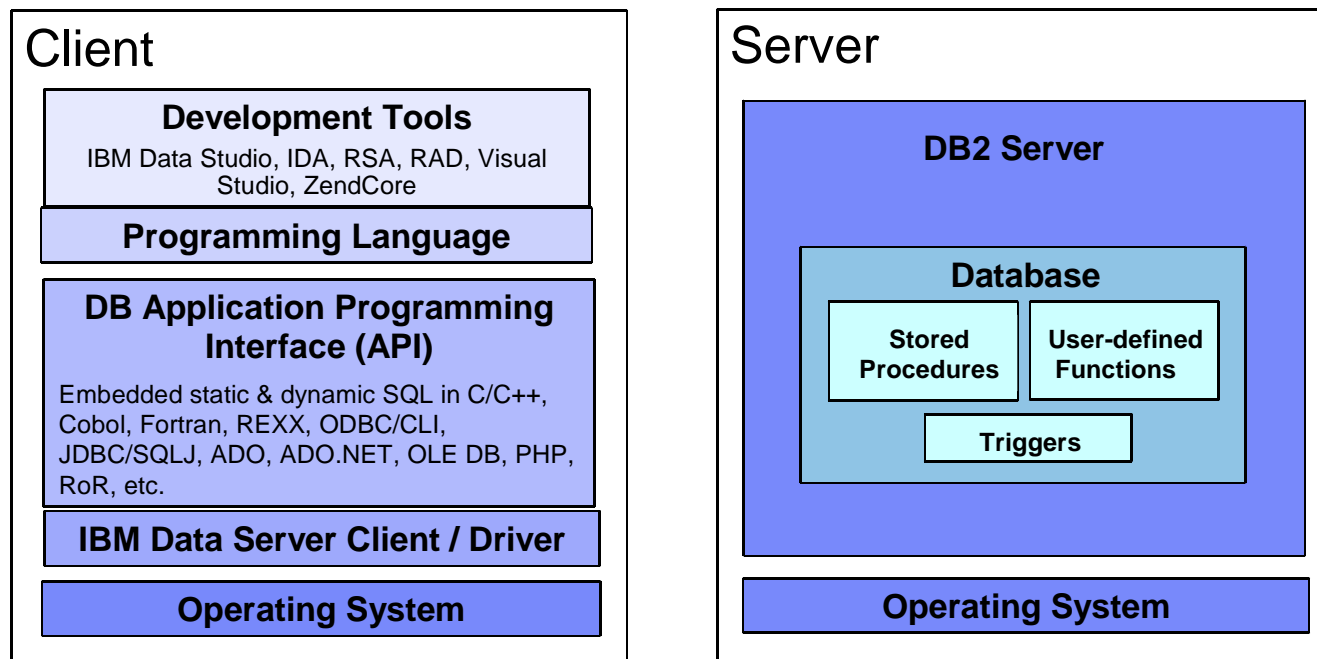
5

Template Documentation

© 2011 IBM Corporation

DB2 application development is divided in two parts, the first part talks about server-side development (Stored procedures, UDFs, etc), and the second part talks about Client-side development (eg: Accessing DB2 from a Java program, for example)

DB2 Application Development Overview



05/30/2011

Template Documentation

6

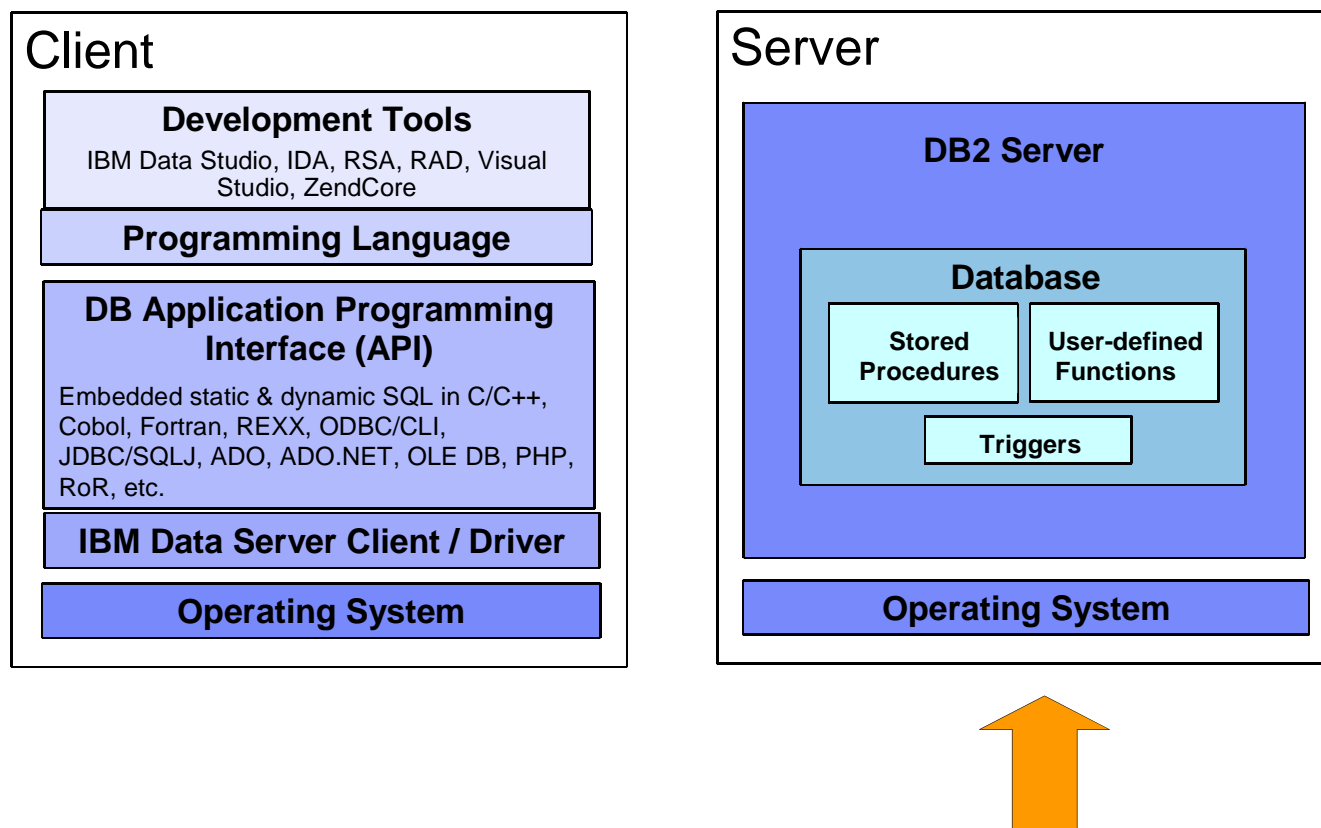
© 2011 IBM Corporation

In the figure, the left side represents a client machine where an application programmer develops and runs his program. In this client machine, in addition to the operating system, an IBM Data Server Client may be installed depending on the type of application being developed. An IBM Data Server client includes the required connection drivers such as the JDBC drivers and the ODBC/CLI drivers. These drivers can also be downloaded independently by visiting the IBM DB2 Express-C Web site at ibm.com/db2/express

Using programming tools such as IBM Data Studio, InfoSphere Data Architect (IDA), Rational Software Architect (RSA), Rational Application Developer (RAD), and so on, you can develop your application in your desired programming language. The API libraries supporting these languages are also included with the IBM Data Server Client, so that when you connect to a DB2 Server, all the program instructions are translated appropriately using these APIs into the SQL or XQuery statements understood by DB2.

On the right side of the figure a DB2 server is illustrated containing one database. Within this database there are stored procedures, user-defined functions and triggers. We describe all of these objects in more detail in the next slides as we are concentrating on this part first.

DB2 Application Development Overview



05/30/2011

Template Documentation

7

© 2011 IBM Corporation

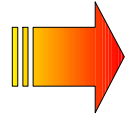
In the figure, the left side represents a client machine where an application programmer develops and runs his program. In this client machine, in addition to the operating system, an IBM Data Server Client may be installed depending on the type of application being developed. An IBM Data Server client includes the required connection drivers such as the JDBC drivers and the ODBC/CLI drivers. These drivers can also be downloaded independently by visiting the IBM DB2 Express-C Web site at ibm.com/db2/express

Using programming tools such as IBM Data Studio, InfoSphere Data Architect (IDA), Rational Software Architect (RSA), Rational Application Developer (RAD), and so on, you can develop your application in your desired programming language. The API libraries supporting these languages are also included with the IBM Data Server Client, so that when you connect to a DB2 Server, all the program instructions are translated appropriately using these APIs into the SQL or XQuery statements understood by DB2.

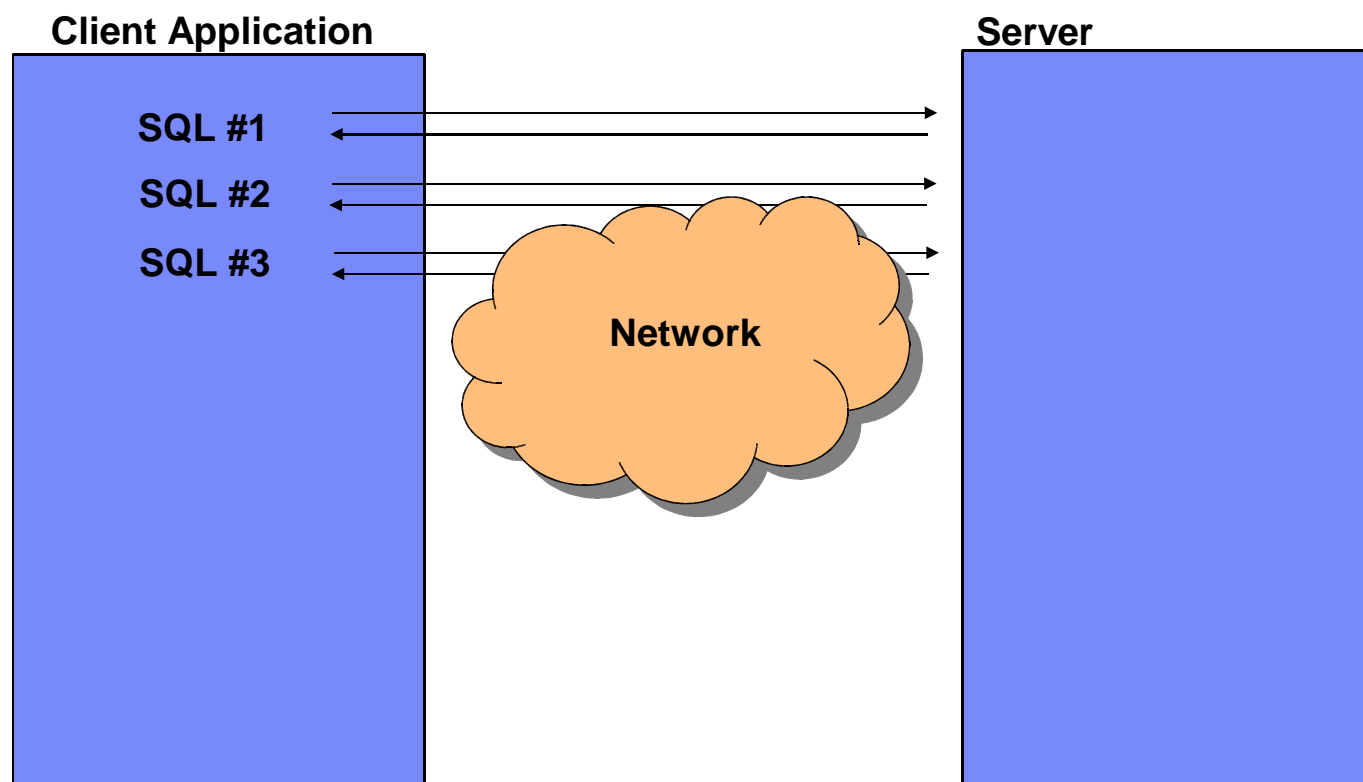
On the right side of the figure a DB2 server is illustrated containing one database. Within this database there are stored procedures, user-defined functions and triggers. We describe all of these objects in more detail in the next slides as we are concentrating on this part first.

Agenda

- DB2 Application Development overview
- **Server-side development**
 - ▶ **Stored Procedures**
 - ▶ User-defined functions
 - ▶ Triggers
- Client-side development
 - ▶ Embedded SQL
 - ▶ Static vs. Dynamic SQL
 - ▶ CLI/ODBC
 - ▶ JDBC / SQLJ / pureQuery



Stored procedures overview



05/30/2011

Template Documentation

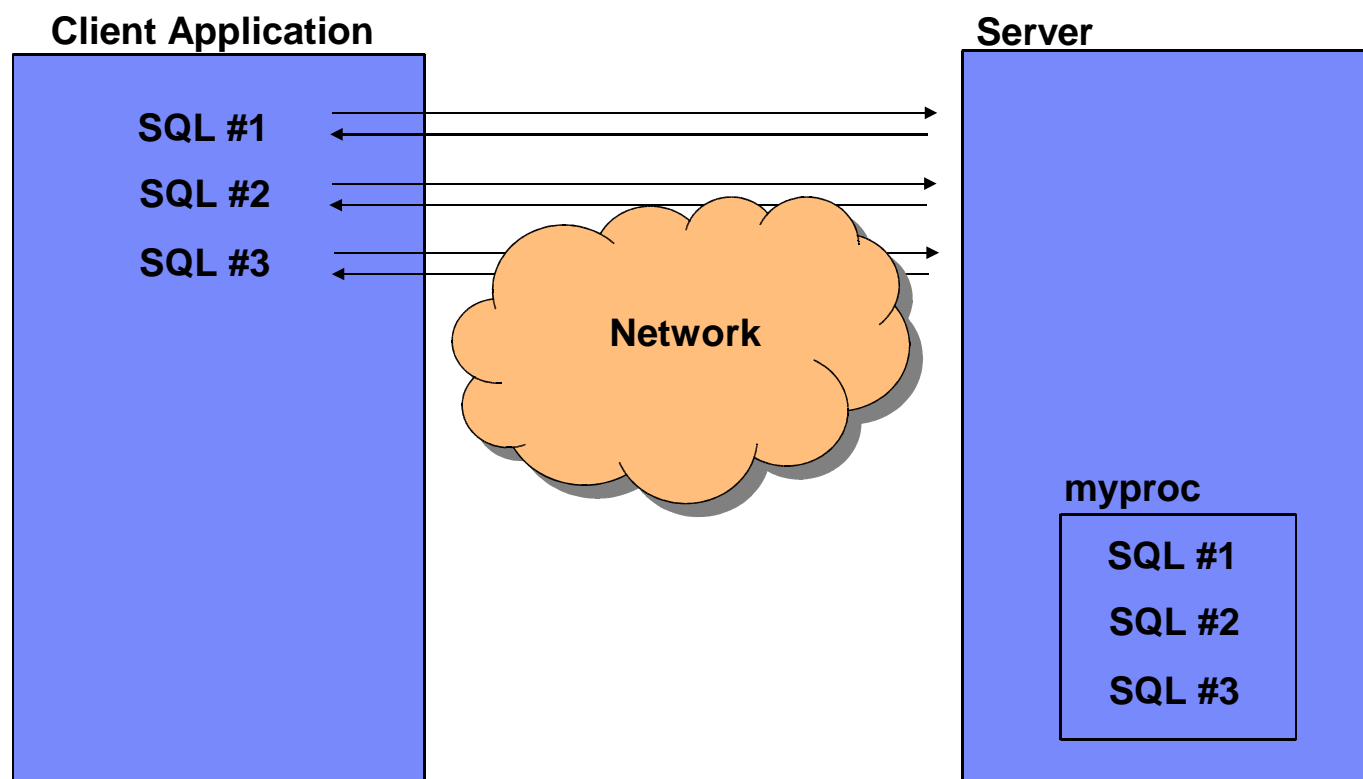
9

© 2011 IBM Corporation

This figure illustrates how Stored Procedures work and why they are needed.

At the top left corner of the figure, you see several SQL statements executed one after the other. Each SQL is sent from the client to the data server, and the data server returns the result back to the client. If many SQL statements are executed like this, network traffic increases.

Stored procedures overview



05/30/2011

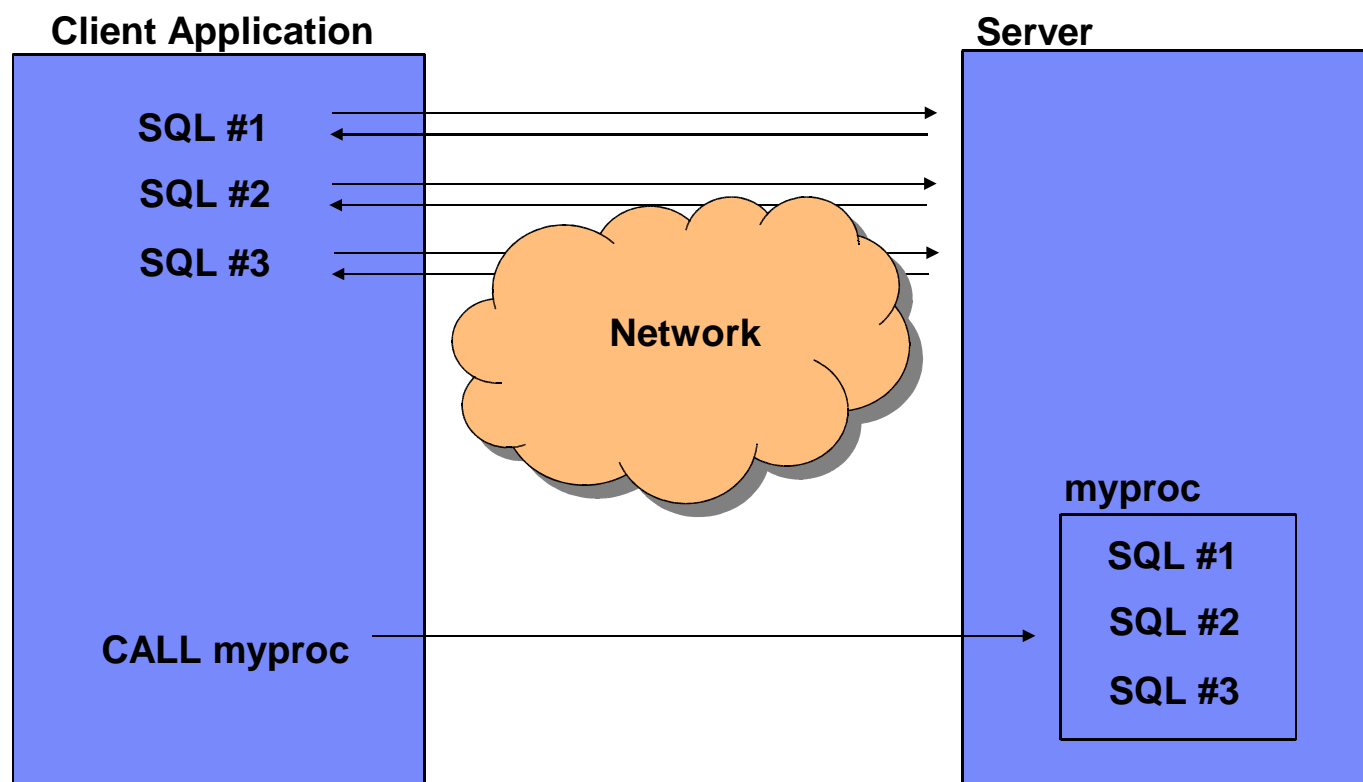
10

Template Documentation

© 2011 IBM Corporation

On the other hand, at the bottom, you see an alternate method that incurs less network traffic. This second method calls a stored procedure **myproc** stored on the server, which contains the same SQL;

Stored procedures overview



05/30/2011

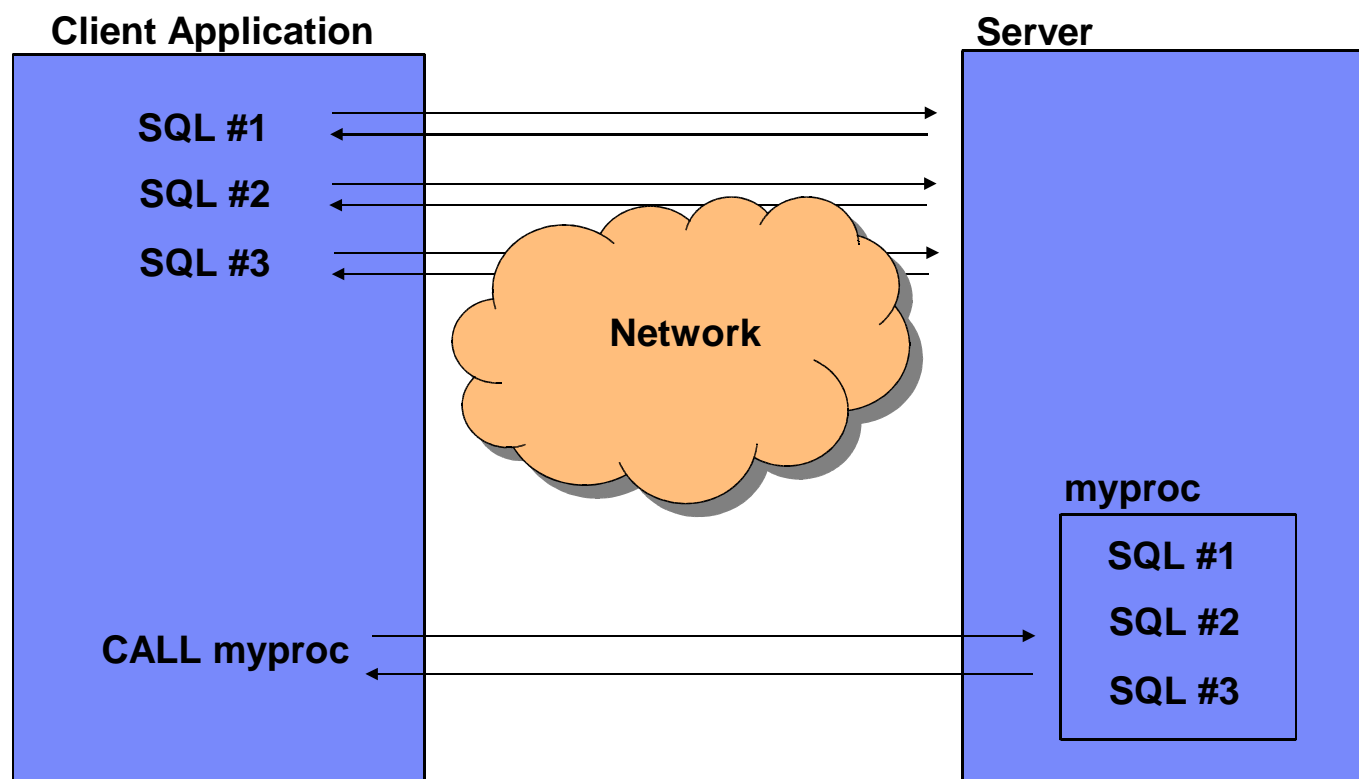
Template Documentation

11

© 2011 IBM Corporation

and then at the client (on the left side), the CALL statement is used to call the stored procedure.

Stored procedures overview



05/30/2011

12

Template Documentation

© 2011 IBM Corporation

This second method is more efficient, as there is only one call statement that goes through the network, and one result set returned to the client.

Stored procedures can also be helpful for security purposes in your database. For example, you can let users access tables or views only through stored procedures; this helps lock down the server and keep users from accessing information they are not supposed to access. This is possible because users do not require explicit privileges on the tables or views they access through stored procedures; they just need to be granted sufficient privilege to invoke the stored procedures.

Stored procedures overview

- Usually contain one or more SQL statements as well as procedural (business) logic
- Executed and managed by DB2 (server-side objects)
- Can be written using SQL PL, C/C++, Java, Cobol, CLR supported languages, OLE, PL SQL, etc.
- Benefits for using stored procedures include:
 - ▶ Centralized business logic that promotes code re-use
 - ▶ Improved security
 - ▶ Improved performance
- This workshop focuses on SQL PL procedures because of their popularity, good performance and simplicity

Creating your first stored procedure

- Using the Command Line Processor:

db2=> connect to sample

db2=> create procedure p1 begin end

- Using the IBM Data Studio
(Demo)

05/30/2011

14

Template Documentation

© 2011 IBM Corporation

The procedure created using the CLP is doing nothing. It's used for illustration purposes. Note you need to connect to the database first, because that's where the stored procedure resides.

Basic stored procedure structure

```
CREATE PROCEDURE proc_name [( {optional parameters} )]  
[optional procedure attributes]  
<statement>
```

[optional parameters]

IN	Input parameter
OUT	Output parameter
INOUT	Input and Output parameter

Example:

```
CREATE PROCEDURE proc(IN p1 INT, OUT p2 INT, INOUT p3 INT)  
...
```

Basic stored procedure structure

[optional procedure attributes]

LANGUAGE SQL

RESULT SETS <n> (required if returning result sets)

<statement> is a single statement, or a set of statements grouped by BEGIN [ATOMIC] ... END

05/30/2011

16

Template Documentation

© 2011 IBM Corporation

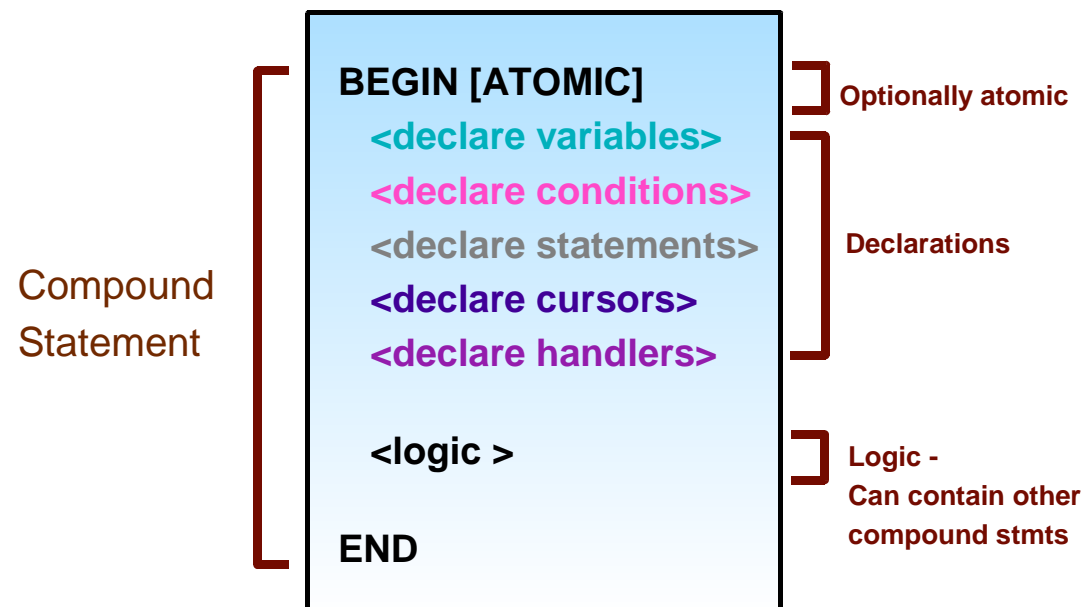
LANGUAGE SQL

This attribute indicates the language the stored procedure will use. LANGUAGE SQL is the default value. For other languages, such as Java or C use LANGUAGE JAVA or LANGUAGE C, respectively.

RESULT SETS <n>

This is required if your stored procedure will be returning n result sets.

Basic stored procedure structure: Compound statements



05/30/2011

Template Documentation

17

© 2011 IBM Corporation

A compound statement in a stored procedure is a statement consisting of several procedural instructions and SQL statements encapsulated by the keywords `BEGIN` and `END`. When the `ATOMIC` keyword follows the `BEGIN` keyword, the compound statement is treated as one unit, that is, all of the instructions or statements in the compound statement must be successful in order for the entire compound statement to be successful. If one of the statements is not, then everything is rolled back.

Note: The order must be followed as shown.

For example, if you try to declare a cursor before variables, you will get errors.

Variable declaration & assignments

```
DECLARE var_name <data type> [ DEFAULT value]
```

Examples:

```
DECLARE temp1 SMALLINT DEFAULT 0;  
DECLARE temp2 VARCHAR(10) DEFAULT 'hello';  
DECLARE temp3 DATE DEFAULT '1998-12-25';
```

```
SET var_name = value
```

Examples:

- **SET total = 100;**
 - Same as VALUES(100) INTO total;
- **SET total = NULL;**
 - any variable can be set to NULL
- **SET total = (select sum(c1) from T1);**
 - Condition is raised if more than one row
- **SET first_val = (select c1 from T1 fetch first 1 row only)**
 - fetch only the first row from a table
- **SET sch = CURRENT SCHEMA;**

05/30/2011

Template Documentation

18

© 2011 IBM Corporation

Example: Stored procedure with parameters

```
CREATE PROCEDURE P2 ( IN      v_p1 INT,  
                     INOUT v_p2 INT,  
                     OUT    v_p3 INT)  
  
LANGUAGE SQL  
SPECIFIC myP2  
BEGIN  
    -- my second SQL procedure  
    SET v_p2 = v_p2 + v_p1;  
    SET v_p3 = v_p1;  
END
```

To call the procedure from the Command Line Processor:

```
db2=> call P2 (3, 4, ?)
```

05/30/2011

Template Documentation

19

© 2011 IBM Corporation

Note:

The question mark is required when calling a stored procedure from an application (other than Data Studio) for OUT parameters.

Example: Stored procedure processing a cursor

```
CREATE PROCEDURE sum_salaries(OUT sum INTEGER)
LANGUAGE SQL
BEGIN
    DECLARE p_sum INTEGER;
    DECLARE p_sal INTEGER;
    DECLARE SQLSTATE CHAR(5) DEFAULT '00000';
    DECLARE c CURSOR FOR
        SELECT SALARY FROM EMPLOYEE;
    SET p_sum = 0;
    OPEN c;
    FETCH FROM c INTO p_sal;
    WHILE(SQLSTATE = '00000') DO
        SET p_sum = p_sum + p_sal;
        FETCH FROM c INTO p_sal;
    END WHILE;
    CLOSE c;
    SET sum = p_sum;
END
```

05/30/2011

Template Documentation

20

© 2011 IBM Corporation

SQLCODE and SQLSTATE

- Access requires explicit declaration:
 - `DECLARE SQLSTATE CHAR(5);`
 - `DECLARE SQLCODE INT;`
- Can be declared ONLY at outermost scope and automatically set by DB2 after each operation
- **SQLCODE**
 - `= 0`, successful.
 - `> 0`, successful with warning
 - `< 0`, unsuccessful
 - `= 100`, no data was found.
 - i.e. `FETCH` statement returned no data
- **SQLSTATE**
 - `SQLSTATE '00000'` = Success
 - `SQLSTATE '02000'` = Not found
 - `SQLSTATE '01XXX'` = Warning
 - Everything else = Exception

Example: Calling a stored procedure from Java application

```
try
{
    // Connect to sample database
    String url = "jdbc:db2:sample";
    con = DriverManager.getConnection(url);
    CallableStatement cs = con.prepareCall("CALL
trunc_demo(?, ?)");
    // register the output parameters
    callStmt.registerOutParameter(1, Types.VARCHAR);
    callStmt.registerOutParameter(2, Types.VARCHAR);
    cs.execute();
    con.close();
}
catch (Exception e)
{
    /* exception handling logic goes here */
}
```

More examples at:

C:\Program Files\IBM\SQLLIB\samples

05/09/2011

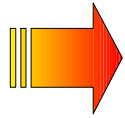
22

Template Documentation

© 2011 IBM Corporation

Agenda

- DB2 Application Development overview
- Server-side development
 - ▶ Stored Procedures
 - ▶ **User-defined functions**
 - ▶ Triggers
- Client-side development
 - ▶ Embedded SQL
 - ▶ Static vs. Dynamic SQL
 - ▶ CLI/ODBC
 - ▶ JDBC / SQLJ / pureQuery



User-defined functions

- Functions always return a value
- Some built-in functions already exist out-of-the-box
 - ▶ Eg: SUM(), AVG(), DIGITS(), etc.
- Can create UDFs in:
 - ▶ SQL PL, C/C++, Java, CLR, OLE, etc.
 - ▶ In this workshop, we focus on SQL PL functions because of their simplicity and popularity

05/30/2011

24

Template Documentation

© 2011 IBM Corporation

A user-defined function (UDF) is a database application object that maps a set of input data values into a set of output values. For example, a function may take a measurement in inches as input, and return the result in centimeters.

DB2 supports creating functions using SQL PL, PL/SQL, C/C++, Java, CLR (Common Language Runtime), and OLE (Object Linking and Embedding) (i.e: .NET technologies).

You can think of a UDF as a way to allow users to extend the SQL language to their needs. You invoke functions from a **SELECT** or a **VALUES** statement

Type of functions

- **Scalar functions**

- ▶ Return a single value
- ▶ Cannot change database state (i.e. no INSERT, UPDATE, DELETE statements allowed)
 - Example: COALESCE(), SUBSTR()

- **Table functions**

- ▶ Return values in a table format
- ▶ Called in the FROM clause of a query
- ▶ Can change database state (i.e. allow INSERT, UPDATE, DELETE statements)
 - Example: SNAPSHOT_DYN_SQL(), MQREADALL()

- **Others type of functions (not covered in this course):**

- ▶ Row functions
- ▶ Column functions

Scalar functions

- Scalar functions take input values and return a single value
- They cannot be used to modify table data

```
CREATE FUNCTION deptname(p_empid VARCHAR(6))
RETURNS VARCHAR(30)
SPECIFIC deptname
BEGIN ATOMIC
    DECLARE v_department_name VARCHAR(30);
    DECLARE v_err VARCHAR(70);
    SET v_department_name = (
        SELECT d.deptname FROM department d, employee e
        WHERE e.workdept=d.deptno AND e.empno= p_empid);
    SET v_err = 'Error: employee ' || p_empid || ' was not found';
    IF v_department_name IS NULL THEN
        SIGNAL SQLSTATE '80000' SET MESSAGE_TEXT=v_err;
    END IF;
    RETURN v_department_name;
END
```

05/30/2011

Template Documentation

26

© 2011 IBM Corporation

In the above listing, the function name is **deptname** and it returns the department number of an employee based on the employee id.

Invoking a scalar function

- Scalar UDFs can be invoked in SQL statements wherever a scalar value is expected, or in a VALUES clause

```
▶ SELECT DEPTNAME('000010') FROM SYSIBM.SYSDUMMY1  
  
▶ VALUES DEPTNAME('000010')
```

05/30/2011

27

Template Documentation

© 2011 IBM Corporation

A scalar UDF can be invoked using the VALUES statement. It can also be invoked from a SQL statement wherever a scalar value is expected. For example, try the following from the DB2 Command Window or from a Linux or UNIX terminal:

```
db2 "values (deptname ('000300'))"
```

or

```
db2 "select (deptname ('000300')) from sysibm.sysdummy1"
```

Note in the second example that the SYSIBM.SYSDUMMY1 is used. This is a special dummy table with one row and one column. It is used to ensure that only one value is returned. If you try the same SELECT statement with any other table that had more rows, the function would be invoked as many times as the table has.

Table UDFs

- Returns a table
- Used in the FROM clause of a query
- Typically used to return a table and keep an audit record

Example: A function that enumerates a set of employees of a department

```
CREATE FUNCTION getEnumEmployee(p_dept VARCHAR(3))  
RETURNS TABLE  
    ( empno CHAR(6),  
      lastname VARCHAR(15),  
      firstnme VARCHAR(12))  
SPECIFIC getEnumEmployee  
RETURN  
    SELECT e.empno, e.lastname, e.firstnme  
    FROM employee e  
    WHERE e.workdept=p_dept
```

05/30/2011

Template Documentation

28

© 2011 IBM Corporation

Table functions return a table of rows. You can call them using the FROM clause of a query. Table functions, as opposed to scalar functions, can change the database state; therefore, INSERT, UPDATE, and DELETE statements are allowed. Some built-in table functions are `SNAPSHOT_DYN_SQL()` and `MQREADALL()`. Table functions are similar to views, but since they allow for data modification statements (INSERT, UPDATE, and DELETE) they are more powerful. Typically they are used to return a table and keep an audit record.

Calling a table UDFs

- Used in the FROM clause of an SQL statement
- The TABLE() function must be applied and must be aliased.

```
SELECT * FROM  
TABLE (getEnumEmployee('E01')) T
```

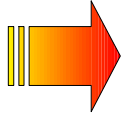
TABLE() function

alias

A table UDF has to be invoked in the FROM clause of an SQL statement since it returns a table. The special TABLE() function must be applied and an alias must be provide after its invocation.

Agenda

- DB2 Application Development overview
- Server-side development
 - ▶ Stored Procedures
 - ▶ User-defined functions
 - ▶ **Triggers**
- Client-side development
 - ▶ Embedded SQL
 - ▶ Static vs. Dynamic SQL
 - ▶ CLI/ODBC
 - ▶ JDBC / SQLJ / pureQuery



Triggers

- A trigger is a database object defined on a table and fired when an INSERT, UPDATE, or DELETE operation is performed.
- Activate (“fire”) automatically
- Operations that cause triggers to fire are called *triggering* SQL statements

05/30/2011

Template Documentation

31

© 2011 IBM Corporation

Triggers are database objects associated with a table that define operations to occur when an INSERT, UPDATE, or DELETE operation is performed on the table. Triggers are activated automatically. The operations that cause triggers to fire are called triggering SQL statements.

Types of triggers

- **BEFORE**

- ▶ Activation before row is inserted, updated or deleted

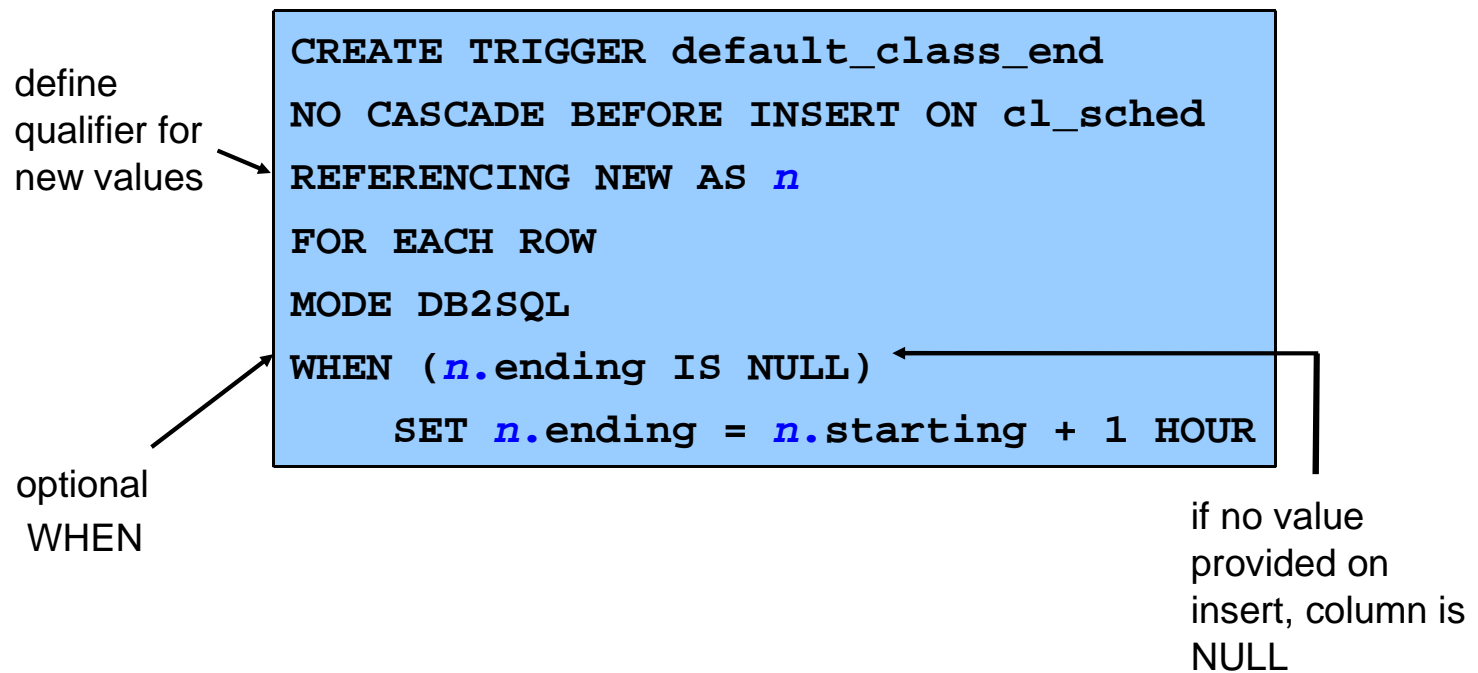
- **AFTER**

- ▶ Activated after the triggering SQL statement has executed to successful completion

- **INSTEAD OF**

- ▶ Defined on views
- ▶ Logic defined in the trigger is executed *instead of* the triggering SQL statement

Example of a BEFORE trigger



05/30/2011

33

Template Documentation

© 2011 IBM Corporation

The trigger ***default_class_end*** will be triggered before an INSERT SQL statement is performed on the table CL_SCHED. This table is part of the SAMPLE database, so you can create and test this trigger yourself while connected to this database. The variable *n* in the trigger definition represents the new value in an INSERT, that is, the value being inserted. The trigger will check the validity of what is being inserted into the table. If the column ENDING has no value during an INSERT, the trigger will ensure it has the value of the column STARTING plus 1 hour. To test the trigger you can do:

db2 insert into cl_sched (class_code, day, starting) values ('abc',1,current time)

db2 select * from cl_sched

Example of an AFTER trigger

- Similar to BEFORE triggers, except that INSERT, UPDATE and DELETE are supported
- Prereq:
 - CREATE TABLE audit (mytimestamp timestamp, comment varchar (1000))

```
CREATE TRIGGER audit_emp_sal
AFTER UPDATE OF salary ON employee
REFERENCING OLD AS o NEW AS n
FOR EACH ROW
MODE DB2SQL

INSERT INTO audit VALUES (
CURRENT TIMESTAMP, ' Employee ' || o.empno || '
salary changed from ' || CHAR(o.salary) || ' to
' || CHAR(n.salary) || ' by ' || USER)
```

05/30/2011

Template Documentation

34

© 2011 IBM Corporation

The trigger **audit_emp_sal** is used to perform auditing on the column SALARY of the EMPLOYEE table. When someone makes a change to this column, the trigger will be activated to write the information about the changed made to the salary into another table called AUDIT. The OLD as o NEW as n line indicates that the prefix **o** will be used to represent the old or existing value in the table, and the prefix **n** will be used to represent the new value coming from the UPDATE statement. Thus, **o.salary** represents the old or existing value of the salary, and **n.salary** represents the updated value for the column salary data.

Example of an INSTEAD OF trigger

- It is activated when performing changes to a view
- Prereq:

```
CREATE TABLE countries (id int, country varchar(50),
                        region varchar (50), average_temp int)
CREATE VIEW view1 (id, continent, temperature) as
SELECT id, region, average_temp from countries
```

```
CREATE TRIGGER update_view1
  INSTEAD OF UPDATE ON view1
  REFERENCING OLD AS o NEW AS n
  FOR EACH ROW
  MODE DB2SQL
  BEGIN ATOMIC
    UPDATE countries
    SET region = n.region
    WHERE region = o.region;
  END
```

05/30/2011

Template Documentation

35

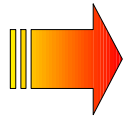
© 2011 IBM Corporation

Instead of triggers are defined on views. Since views are defined dynamically using a SELECT statement that accesses one or more tables, views cannot be updated. However, using this type of trigger, you can give users the illusion that a view can be updated because the logic defined in the trigger is executed instead of the triggering SQL statement. For example, if you perform an update operation on a view, the instead of trigger will be activated to actually perform the update on the base tables that form the view.

Triggers cannot be created from IBM Data Studio unless you use the “Script” folder. They can be created from the Control Center or from the Command line tools (DB2 Command Window, Command Line Processor)

Agenda

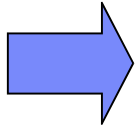
- DB2 Application Development overview
- Server-side development
 - ▶ Stored Procedures
 - ▶ User-defined functions
 - ▶ Triggers
- **Client-side development**
 - ▶ Embedded SQL
 - ▶ Static vs. Dynamic SQL
 - ▶ CLI/ODBC
 - ▶ JDBC / SQLJ / pureQuery



DB2 Application Development Overview

Server-side development (at the DB2 database server):

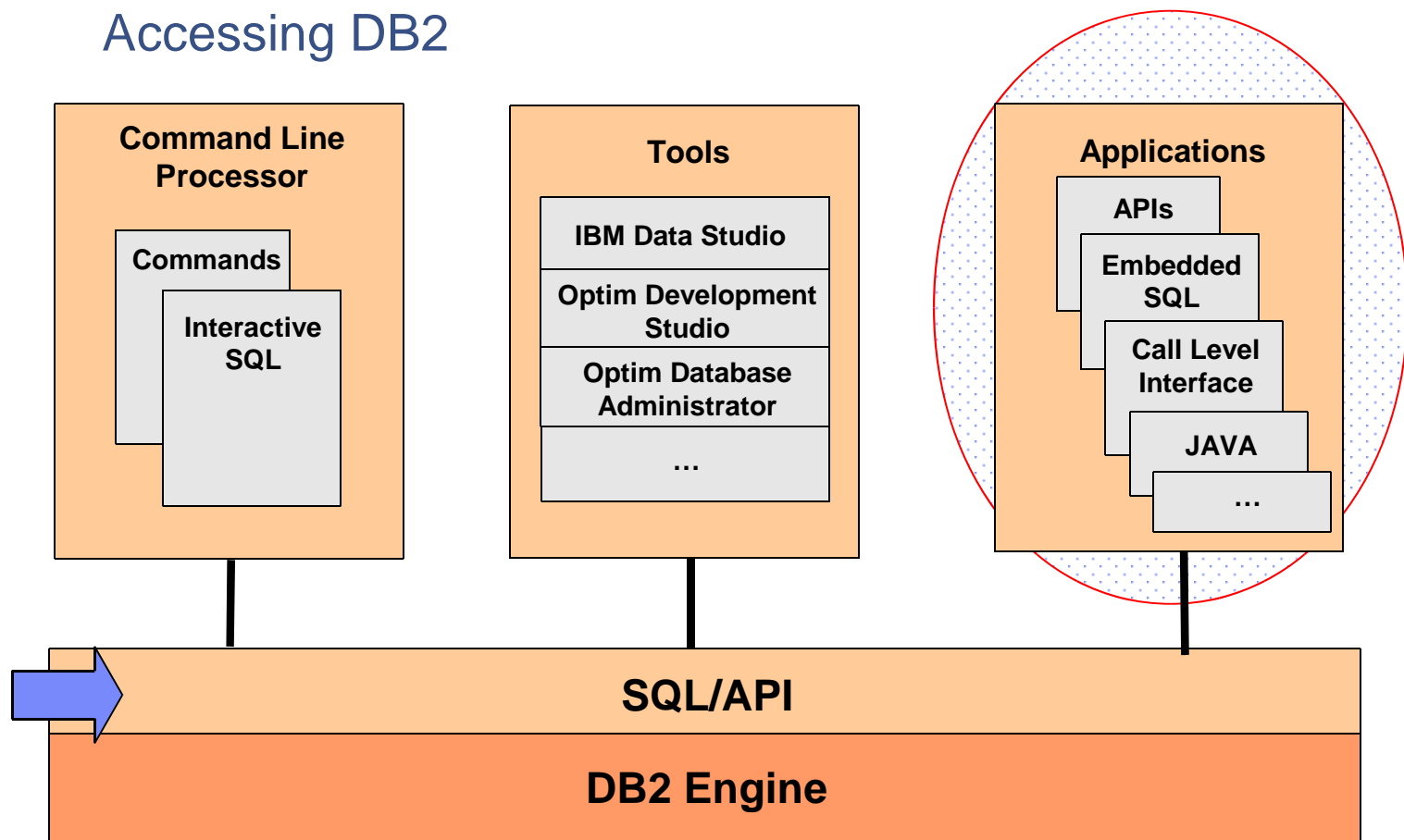
- ▶ Routines (Stored Procedures, UDFs)
- ▶ Database objects (Triggers)



Client-side development (at the client):

- ▶ May require a DB2 client or driver to be installed
- ▶ Database applications (in C/C++, .NET, Cobol, Java, etc)

Accessing DB2



05/30/2011

Template Documentation

38

© 2011 IBM Corporation

In the end, the SQL/API layer converts everything into SQL that DB2 can understand.

Application development freedom

- Ruby on Rails
- C/C++ (ODBC and Static SQL)
- JDBC and SQLJ
- Borland
- Python
- PHP
- Perl
- .NET languages
- OLE-DB
- ADO
- Web Services
- SQL
- MS Office: Excel, Access, Word



05/30/2011

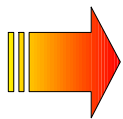
Template Documentation

39

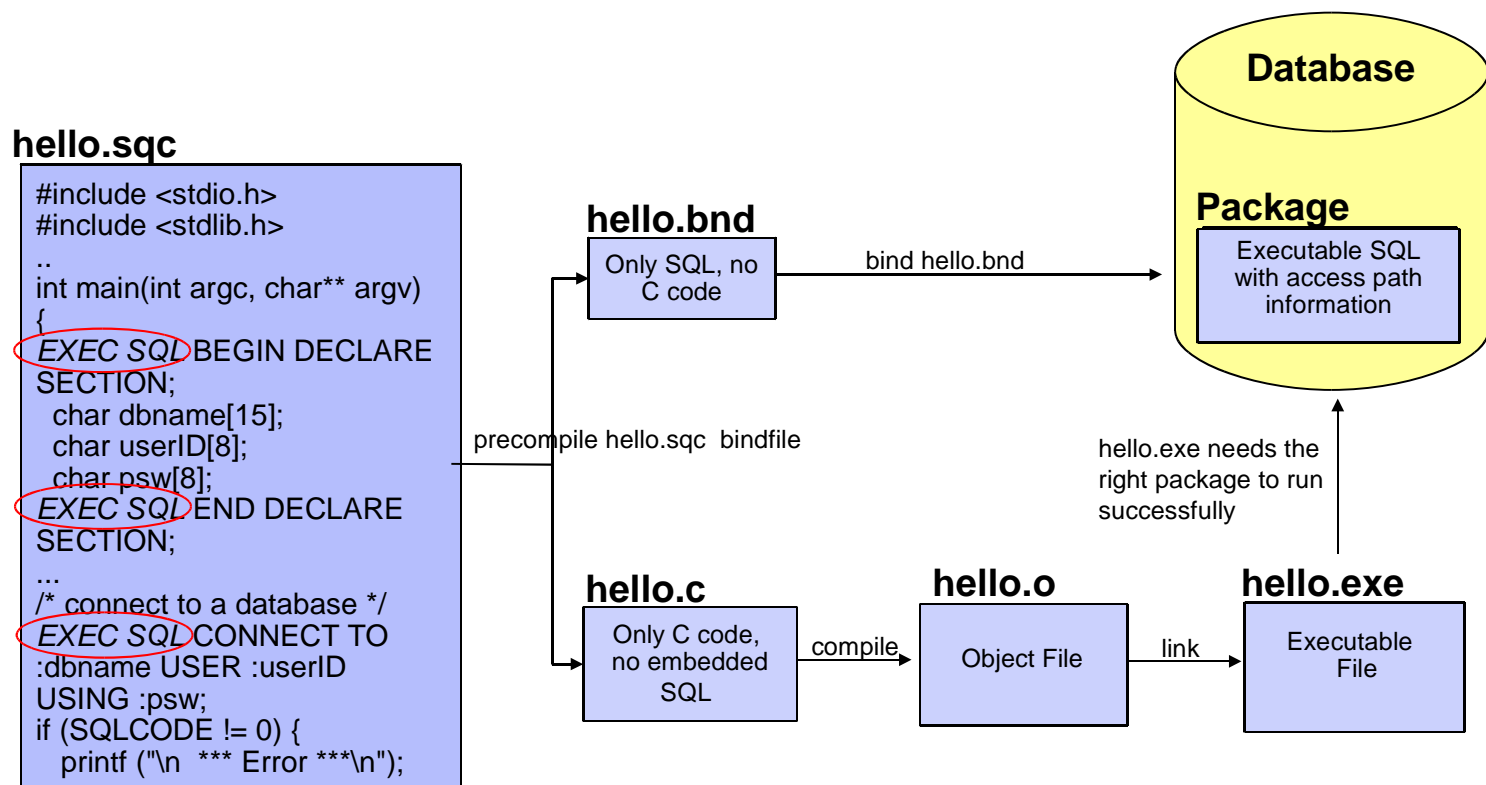
© 2011 IBM Corporation

Agenda

- DB2 Application Development overview
- Server-side development
 - ▶ Stored Procedures
 - ▶ User-defined functions
 - ▶ Triggers
- Client-side development
 - ▶ **Embedded SQL**
 - ▶ Static vs. Dynamic SQL
 - ▶ CLI/ODBC
 - ▶ JDBC / SQLJ / pureQuery



Embedded SQL



05/30/2011

41

Template Documentation

© 2011 IBM Corporation

Embedded SQL applications are applications where SQL is embedded into a host language such as C, C++, or COBOL. The embedded SQL application can include static or dynamic SQL.

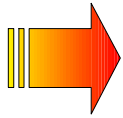
In the figure, the C program `hello.sqc` contains embedded SQL. The embedded SQL API for the C language uses `EXEC SQL` (highlighted in *Figure 1.2*) as a delimiter to allow the PRECOMPILER that comes with DB2 to identify the SQL statements to be translated versus the actual C code.

You may also note in the `hello.sqc` listing that some variables are prefixed with a colon, as in `:dbname`, `:userID`, and `:psw`. These are called host variables. Host variables are variables from the host language that are referenced in the embedded SQL statements.

Issuing the **precompile** command (also known as the **prep** command) with the **bindfile** option generates two files, the `hello.bnd` bind file containing only SQL statements and the `hello.c` file containing only C code. The bind file will be compiled using the **bind** command to obtain a **package** that is stored in the database. A package includes the compiled/executable SQL and the access path DB2 will follow to retrieve the data. To issue the **bind** command, a connection to the database must exist. At the bottom of the figure, the `hello.c` file is compiled and linked like any regular C program. The resulting executable file `hello.exe` has to match the package stored in the database to successfully execute.

Agenda

- DB2 Application Development overview
- Server-side development
 - ▶ Stored Procedures
 - ▶ User-defined functions
 - ▶ Triggers
- Client-side development
 - ▶ Embedded SQL
 - ▶ **Static vs. Dynamic SQL**
 - ▶ CLI/ODBC
 - ▶ JDBC / SQLJ / pureQuery



Static SQL

- The SQL statement structure is fully known at precompile time.

```
SELECT lastname, salary FROM employee
```

- The names for the columns (lastname, salary) and tables (employee) referenced in a statement are fully known at precompile time.
- Host variables values can be specified at run time (but their data type must still be precompiled).

```
SELECT lastname, salary  
FROM employee  
WHERE firstname = :fname
```

- You precompile, bind, and compile statically executed SQL statements before you run your application.
- Static SQL is best used on databases whose statistics do not change a great deal.

Template Documentation

Static SQL statements are the ones where the SQL structure is fully known at precompile time.

In this second example, a host variable **:fname** is used as part of an embedded SQL statement. Though the value of the host variable is unknown until runtime, its data type is known from the program, and all the other objects (column names, table names) are fully known ahead of time. DB2 software uses estimates for these host variables to calculate the access plan ahead of time; therefore, this case is still considered static SQL.

You precompile, bind, and compile statically executed SQL statements before you run your application. Static SQL is best used on databases whose statistics do not change a great deal

Dynamic SQL

- The SQL is built and executed at run-time.

```
SELECT ?, ? FROM ?
```

- ▶ The names for the columns and tables referenced in a statement are not known until runtime.
- The access plan is determined at runtime.
 - ▶ Normally static SQL performs better than dynamic SQL since the access plan is calculated ahead of time
 - ▶ For tables whose statistics change often, dynamic SQL may provide a more accurate access plan.
- When working with dynamic SQL, use parameter markers (?) to reduce the amount of times an access plan is calculated. (see following example)

05/30/2011

Template Documentation

44

© 2011 IBM Corporation

In this example, the names for the columns and table referenced by the statement are not known until runtime. Therefore the access plan is calculated only at runtime and using the statistics available at the time. These types of statements are considered **Dynamic SQL** statements.

Some programming APIs, like JDBC and ODBC, always use dynamic SQL regardless of whether the SQL statement includes known objects or not. For example, the statement **SELECT lastname, salary FROM employee** has all the columns and table names known ahead of time, but through JDBC or ODBC, you do not precompile the statements. All the access plans for the statements are calculated at runtime.

Dynamic SQL

- Example:

Case 1:

```
EXECUTE IMMEDIATELY SELECT name from EMP where dept = 1  
EXECUTE IMMEDIATELY SELECT name from EMP where dept = 2
```

Case 2:

```
strcpy(hVStmtDyn, "SELECT name FROM emp WHERE dept = ?");  
PREPARE StmtDyn FROM :hVStmtDyn;  
EXECUTE StmtDyn USING 1;  
EXECUTE StmtDyn USING 2;
```

- In case 1, each statement is treated as different SQL, therefore DB2 will calculate the access plan for each.
- In case 2, there is only one SQL statement:
"SELECT name FROM emp WHERE dept = ?"
- Therefore, the access plan will only be calculated once, and cached in memory.

05/30/2011

Template Documentation

45

© 2011 IBM Corporation

In general, two statements are used to treat a SQL statement as dynamic:

PREPARE: This statement prepares or compiles the SQL statement calculating the access plan to use to retrieve the data

EXECUTE: This statement executes the SQL

Alternatively you can execute a **PREPARE** and **EXECUTE** in one single statement:
EXECUTE IMMEDIATELY

Static vs. Dynamic SQL

- Embedded SQL applications support static & dynamic SQL
- Example of a static SQL in an embedded SQL C program

```
EXEC SQL SELECT name, dept
          INTO :name, :dept
          FROM staff WHERE id = 310;
printf( ...)
```

- Example of a dynamic SQL in an embedded SQL C program

```
...
strcpy(hostVarStmtDyn,
       "UPDATE staff SET salary = salary + 1000 WHERE dept = ?");
EXEC SQL PREPARE StmtDyn FROM :hostVarStmtDyn;
EXEC SQL EXECUTE StmtDyn USING :dept;
```

05/30/2011

Template Documentation

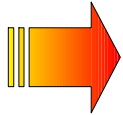
46

© 2011 IBM Corporation

With respect to performance, static SQL will normally perform better than dynamic SQL since the access plan in static SQL is performed at precompile time and not at runtime. However, for environments where there is a lot of activity such as INSERTs and DELETEs, the statistics calculated at precompile time may not be up-to-date, and therefore, the access plan of the static SQL may not be optimal. In this case, dynamic SQL may be a better choice, assuming a RUNSTATS command is frequently executed to collect current statistics.

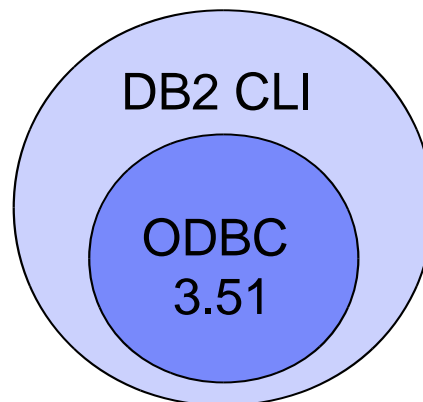
Agenda

- DB2 Application Development overview
- Server-side development
 - ▶ Stored Procedures
 - ▶ User-defined functions
 - ▶ Triggers
- Client-side development
 - ▶ Embedded SQL
 - ▶ Static vs. Dynamic SQL
 - ▶ **CLI/ODBC**
 - ▶ JDBC / SQLJ / pureQuery



CLI / ODBC

- CLI = Call Level Interface
- DB2 CLI can be used as the ODBC Driver when loaded by an ODBC Driver Manager
- DB2 CLI conforms to ODBC 3.51



05/30/2011

48

Template Documentation

© 2011 IBM Corporation

DB2 CLI conforms to ODBC 3.51 and can be used as the ODBC Driver when loaded by an ODBC Driver Manager. The figure can help you picture DB2 CLI support for ODBC.

CLI / ODBC

- To **run** a CLI/ODBC application all you need is the DB2 CLI driver. This driver is installed from either of these:
 - ▶ IBM Data Server Client
 - ▶ IBM Data Server Runtime Client
 - ▶ IBM Data Server Driver for ODBC and CLI

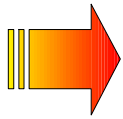
- To **develop** a CLI/ODBC application you need the DB2 CLI driver and also the appropriate libraries. These can be found only on:
 - ▶ IBM Data Server Client

CLI / ODBC

- CLI/ODBC characteristics:
 - ▶ The code is easily portable between several RDBMS vendors
 - ▶ Unlike embedded SQL, there is no need for a precompiler or host variables
 - ▶ It runs dynamic SQL
 - ▶ It is very popular

Agenda

- DB2 Application Development overview
- Server-side development
 - ▶ Stored Procedures
 - ▶ User-defined functions
 - ▶ Triggers
- Client-side development
 - ▶ Embedded SQL
 - ▶ Static vs. Dynamic SQL
 - ▶ CLI/ODBC
 - ▶ **JDBC / SQLJ / pureQuery**



05/30/2011

51

Template Documentation

© 2011 IBM Corporation

JDBC / SQL / pureQuery

- **JDBC characteristics:**

- ▶ Like in ODBC, the code is easily portable between several RDBMS vendors
- ▶ Dynamic SQL
- ▶ It is very popular

- **SQLJ**

- ▶ Embedded SQL in Java
- ▶ Static SQL
- ▶ Not that popular

- **pureQuery**

- ▶ Eclipse-based plug-in to manage relational data as objects
- ▶ IBM's paradigm to develop Java database applications
- ▶ New since mid-2007, available with Optim Development Studio

05/30/2011

Template Documentation

52

© 2011 IBM Corporation

Java Database Connectivity (JDBC) is a Java programming API that standardizes the means to work and access databases. In JDBC the code is easily portable between several RDBMS vendors. The only changes required to the code are normally which JDBC driver to load and the connection string. JDBC uses only dynamic SQL and it is very popular.

SQLJ is the standard for embedding SQL in Java programs. It is mainly used with static SQL, though it can inter-operate with JDBC as shown in the figure. Though it is normally more compact than JDBC programs and provides better performance, it has not been widely accepted. SQLJ programs must be run through a preprocessor (the SQLJ translator) before they can be compiled.

In the figure (see next slide), a DB2 client may or may not be required depending on the type of JDBC driver used.

pureQuery is an IBM Eclipse-based plug-in to manage relational data as objects. Available since 2007, pureQuery can automatically generate the code to establish an object-relational mapping (ORM) between your object oriented code and the relational database objects. You start by creating a Java project with Optim™ Development Studio (ODS), connect to a DB2 database, and then have ODS discover all the database objects. Through the ODS GUI you can pick a table and then choose to generate the pureQuery code which would transform any of the underlying relational table entities into a Java object. Code is generated to create the relevant SQL statements and parent Java objects that encapsulate those statements. The generated Java objects and the contained SQL statements can be further customized. With pureQuery, you can decide at runtime whether you want to run your SQL in static or dynamic mode. pureQuery supports both Java and .NET.

JDBC / SQLJ – Supported drivers

Driver Type	Driver Name	Packaged as	Supports	Minimum level of SDK for Java required
Type 2	DB2 JDBC Type 2 Driver for Linux, UNIX and Windows (Deprecated)	db2java.zip	JDBC 1.2 and JDBC 2.0	1.4.2
Type 2 and Type 4	IBM Data Server Driver for JDBC and SQLJ	db2jcc.jar and sqlj.zip	JDBC 3.0 compliant	1.4.2
		db2jcc4.jar and sqlj4.zip	JDBC 4.0 and earlier	6

- **Type 2 drivers need to have a DB2 client installed**
- **Deprecated means it is still supported, but no longer enhanced**
- **Note that the same file (for example db2jcc.jar) supports type 2 and 4**

05/30/2011

Template Documentation

53

© 2011 IBM Corporation

Though there are several types of JDBC drivers such as type 1, 2, 3 and 4; type 1 and 3 are not commonly used, and DB2's support of these types has been deprecated. For type 2, there are two drivers as we will describe shortly, but one of them is also deprecated.

Type 2 and type 4 are supported with DB2 software, as shown in the *Table*. Type 2 drivers need to have a DB2 client installed, as the driver uses it to establish communication to the database. Type 4 is a pure Java client, so there is no need for a DB2 client, but the driver must be installed on the machine where the JDBC application is running.

As mentioned earlier and shown also in the *Table*, Type 2 is provided with two different drivers; however the DB2 JDBC Type 2 Driver for Linux, UNIX and Windows, with filename db2java.zip is deprecated.

JDBC / SQLJ – Supported drivers

- **db2java.zip, db2jcc.jar, sqlj.zip, db2jcc4.jar and sqlj4.zip are included with:**
 - ▶ IBM DB2 for Linux, UNIX and Windows servers
 - ▶ IBM Data Server Client
 - ▶ IBM Data Server Runtime Client
 - ▶ IBM Data Server Driver for JDBC and SQLJ

05/30/2011

54

Template Documentation

© 2011 IBM Corporation

When you install a DB2 server, a DB2 client or the IBM Data Server Driver for JDBC and SQLJ, the db2jcc.jar and sqlj.zip files compliant with JDBC 3.0 are automatically added to your classpath.



Thank you!

Use the forum in the db2university.com course AA001EN if you have technical questions about the materials covered in this course. Fellow students, faculty and IBMers can help you!