To use derby

1. Go to folder jdk/db
2. Run startNetworkServer.bat to start server
3. Run ij.bat it will open ij prompt
4. Give following command to connect to derby server

connect 'jdbc:derby:hsbcdb;create=true;user=hsbc;password=hsbc123';

1. This will create database
2. To use database give following command

connect 'jdbc:derby:c:\mydata\hsbcdb;user=hsbc;password=hsbc123';

--------------------To create table

Create table employee(

Id int,

Name varchar(20),

Sal decimal(9,2),

Doj date

);

Create table employee1(

Id int primary key,

Name varchar(20) not null,

Sal decimal(9,2) check(sal>=2000),

Commission decimal(9,2) default 2000,

Passport\_no int unique

);

-------------insert data

insert into employee values(12,’Kishori’,12345.00,’12/11/2110’); (mm/dd/yyyy)

**CREATE TABLE FIRSTTABLE**

**(ID INT PRIMARY KEY,**

**NAME VARCHAR(12));**

Id,name,sal,doj,deptid

drop table employee

Create table dept(

deptid int primary key,

dname varchar(20) ,

location varchar(20)

)

create table employee(

empid int primary key,

ename varchar(30),

sal decimal(9,2) check(sal>=2000),

doj date,

deptid int references dept(deptid));

insert into dept values(10,’HR’,’Mumbai’);

insert into dept values(11,’Sales’,’Pune’);

insert into dept values(12,’Purchase’,’Pune’);

insert into employee values(12,’Kishori’,12345.00,’12/11/2110’,12);

insert into employee values(13,’Rajan’,11111.00,’12/11/2110’,11);

insert into employee values(14,’Revati’,22222.00,’12/11/2110’,11);

insert into employee values(15,’Anil’,12345.00,’12/11/2110’,10);

select \* from employee;

select empid,ename from employee;

select empid,ename from employee where sal >10000 and sal<20000;

select empid,ename from employee where sal between 10000 and 20000;

select empid,ename

from employee

where sal > (10000,15000,20000);

select \*

from employee

where ename like ‘R%’

Alter table

**ALTER TABLE** [***table-Name***](https://db.apache.org/derby/docs/10.8/ref/rreftablename.html#rreftablename)

**{**

**ADD COLUMN** [***column-definition***](https://db.apache.org/derby/docs/10.8/ref/rrefsqlj81859.html#rrefsqlj81859__rrefaltercolumndef) **|**

**ADD** [***CONSTRAINT clause***](https://db.apache.org/derby/docs/10.8/ref/rrefsqlj13590.html#rrefsqlj13590) **|**

**DROP [ COLUMN ] column-name [ CASCADE | RESTRICT ]**

**DROP { PRIMARY KEY | FOREIGN KEY constraint-name | UNIQUE**

**constraint-name | CHECK constraint-name | CONSTRAINT constraint-name }**

**ALTER [ COLUMN ]** [***column-alteration***](https://db.apache.org/derby/docs/10.8/ref/rrefsqlj81859.html#rrefsqlj81859__rrefsqlj37860)

**Examples**

ALTER TABLE EMPLOYEE

ADD NEW\_COL VARCHAR(25)

ALTER TABLE EMPLOYEE

ADD NEW\_COL CHAR(25) DEFAULT '2' NOT NULL

ALTER TABLE EMPLOYEE

ADD CONSTRAINT chk\_constraint

CHECK (salary <> 0)

ALTER TABLE EMPLOYEE

ADD FOREIGN KEY (DNO)

REFERENCES DEPARTMENT(DNUMBER)

|  |  |  |  |
| --- | --- | --- | --- |
| empid | ename | salary | deptid |
| 1000 | aaaaa | 20000 | 10 |
| 1001 | bbbbb | 10000 | 20 |
| 1002 | cccccc | 12000 | 10 |
| 1003 | ddddd | 10000 | 30 |
| 1004 | xxxxx | 12345 |  |

|  |  |  |
| --- | --- | --- |
| deptid | dname | dlocation |
| 10 | HR | Mumbai |
| 20 | Purchase | Mumbai |
| 30 | Sales | Pune |
| 40 | Accounts | Mumbai |

**Creating a JAR File**

The basic format of the command for creating a JAR file is:

jar cvf *jar-file input-file(s)*

The options and arguments used in this command are:

* The c option indicates that you want to *create* a JAR file.
* v Produces verbose output on stdout while the JAR file is being built. The verbose output tells you the name of each file as it's added to the JAR file.
* The f option indicates that you want the output to go to a *file* rather than to stdout.
* jar-file is the name that you want the resulting JAR file to have. You can use any filename for a JAR file. By convention, JAR filenames are given a .jar extension, though this is not required.
* The input-file(s) argument is a space-separated list of one or more files that you want to include in your JAR file. The input-file(s) argument can contain the wildcard \*symbol. If any of the "input-files" are directories, the contents of those directories are added to the JAR archive recursively.

Java Stored Procedures with Derby

Stored procedures provide a very powerful capability. They enable the execution of many SQL statements within the context of conditional programming logic, via a single database call. Stored procedures execute in the database, thus making them an excellent place for computationally intense db operations. Stored procedures do come at cost. Some database vendors offer extensions to SQL e.g. PL/SQL, T-SQL, PL/pgSQL . These extensions may be proprietary or may comform to different standards. If your application is required to support a multitude of database flavors, it is likely that you must create separate stored procedures for each supported database. One of these standards is SQL/JRT. SQL/JRT defines a means to write stored procedures using the Java programming language and JDBC. Today’s topic is focused in this area.

1. Craft stored procedure logic using the Java programming language

2. Package and deploy the Java logic to Derby

3. Create a stored procedure that calls the Java logic.

4. Call the stored procedure using JDBC.

A. Install and Configure Derby

B. Create a database and import sample data

A. Install and Configure Derby

B. Create a database and import sample data

First, create a file e.g. C:\sample.sql and add the following content:

CREATE TABLE CUSTOMER (

ID BIGINT GENERATED BY DEFAULT AS IDENTITY NOT NULL,

FIRSTNAME VARCHAR(255), LASTNAME VARCHAR(255),

PRIMARY KEY (ID)

);

INSERT INTO CUSTOMER VALUES(default, 'John', 'Doe');

INSERT INTO CUSTOMER VALUES(default, 'Joe', 'Jones');

INSERT INTO CUSTOMER VALUES(default, 'Sally', 'Smith');

Lauch Derby’s interactive SQL scripting tool, ij, and create the sample database.

$DERBYHOME\bin\ij

ij>CONNECT 'jdbc:derby://localhost:1527/sample;create=true;user=APP;password=APP';

The above command, will create a new database named ‘sample’ (if ‘sample’ does not exist). It will also use the database schema, APP.

Next, import data into the sample database using sample.sql.

ij>RUN 'C:\sample.sql';

Finally, to validate the import, ensure the following statement results in three records.

ij>SELECT \* FROM CUSTOMER;

1. Craft stored procedure logic using the Java programming language

Crafting stored procedure logic using Java is very similar to creating a regular Java class.

(When reading the source code below, pay close attention to any source code comments flagged as)

// IMPORTANT!

First, create a new Java class, <code>SampleStoredProcedure.java</code>:

package sample.sp;

public class SampleStoredProcedures {

// class body here

}

Secondly, declare the method to contain our stored procedure logic. For the purpose if this tutorial, our stored procedure will simply fetch a customer’s last name, given that customer’s id.

public class SampleStoredProcedures {

// This method will be registered to and invoked by the database

// IMPORTANT! It must be declared public static! This is required by SQL/JRT.

// IMPORTANT! OUT parameters must be declared as arrays, even if the expected result will always contain 1 element

public static void getCustomerLastNameById(long customerId /\* IN paramater \*/,

String[] customerLastName /\* OUT paramater \*/) throws SQLException {

// method body here

}

}

Lastly, implement the method’s body.

public static void getCustomerLastNameById(long customerId, String[] customerLastName) throws SQLException {

Connection conn = null;

PreparedStatement stmt = null;

ResultSet res = null;

try {

// re-use the existing JDBC connection

// IMPORTANT! "jdbc:default:connection" tells the DriverManager to use the existing connection.

conn = DriverManager.getConnection("jdbc:default:connection");

// prepare the query

String sql = "SELECT LASTNAME FROM CUSTOMER WHERE ID = ?";

stmt = conn.prepareStatement(sql);

// bind parameters

stmt.setLong(1, customerId);

res = stmt.executeQuery();

// set the result in OUT parameter

// IMPORTANT: Notice that we never instantiate the customerLastName array.

// The array is instead initialized and passed in by Derby, our SQL/JRT implementor

customerLastName[0] = (res.next()) ? res.getString(1) : "Customer not Found.";

} finally {

if (res != null) {

// close the result set

res.close();

}

if (stmt != null) {

// close the statement

stmt.close();

}

if (conn != null) {

// close the db connection

conn.close();

}

}

}

2. Package and deploy the Java logic to Derby

Before we deploy, our new Java stored procedure method with Derby, we must first package our SampleStoredProcedures.class in a Jar file. Let’s call our Jar file, SampleStoredProcedures.jar and save it to e.g. c:\myjars.

Note: If you are unfamiliar with Jar files, Oracle has a nice tutorial here.

Next, launch Derby’ ij tool e.g. $DERBY\_HOME/bin/ij from the command line

First, install our Jar file into Derby.

ij>CALL sqlj.install\_jar( 'c:/myjars/SampleStoredProcedures.jar', 'APP.SampleStoredProcedures',0);

Note: APP refers to, the Derby user, APP’s schema.

To replace the installed Jar with a new version:

ij>CALL sqlj.install\_jar( 'c:/mydata/SampleStoredProcedures.jar', 'hsbc.SampleStoredProcedures',0);

Next, update Derby’s classpath to include our Jar file.

ij>CALL SYSCS\_UTIL.SYSCS\_SET\_DATABASE\_PROPERTY('derby.database.classpath', ‘hsbc.SampleStoredProcedures ');

3. Create the stored procedure to call our Java logic.

Continue using $DERBY\_HOME/bin/ij. We will now create the actual stored procedure using the following SQL statement:

ij>CREATE PROCEDURE GETCUSTOMERLASTNAME(STREAM\_ID BIGINT, OUT NAME VARCHAR(255)) LANGUAGE JAVA EXTERNAL PARAMETER STYLE JAVA ‘samplestoredprocedures.SampleStoredProcedures.getCustomerLastName’;

Below is the CREATE PROCEDURE SQL statement in a more sane format.

CREATE PROCEDURE GETCUSTOMERLASTNAME(STREAM\_ID BIGINT,

OUT NAME VARCHAR(255))

LANGUAGE JAVA

EXTERNAL

PARAMETER STYLE JAVA

'samplestoredprocedures.SampleStoredProcedures.getCustomerLastName';

4. Call the stored procedure via JDBC

Finally, we can call our stored procedure. Unfortunately, because our stored procedure has an OUT parameter, we cannot simply invoke it via ij, using the SQL CALL statement. e.g <code>ij>CALL GETCUSTOMERNAME(....</code> Instead, we will use the following simple JDBC client.

public class TestDriver {

public static void main (String[] args) throws Exception {

Connection conn = null;

CallableStatement stmt = null;

try {

// IMPORTANT! The Derby driver e.g. derbyclient.jar must be on the classpath

Class.forName("org.apache.derby.jdbc.ClientDriver").newInstance();

// Connect to the database

conn = DriverManager

.getConnection("jdbc:derby://localhost:1527/gc;user=APP;password=APP");

// Create the CALL statement

stmt = conn.prepareCall("CALL GETCUSTOMERLASTNAME( ?, ? )");

// Bind the customer id to the first parameter

stmt.setLong(1, Long.parseLong(args[0]));

// Register the second parameter as an OUT parameter

stmt.registerOutParameter(2, Types.VARCHAR);

stmt.execute();

// Print result in the OUT parameter - e.g. the customer's last name

System.out.println(stmt.getString(2));

} finally {

// Release db resources

if (stmt != null) {

stmt.close();

}

if (conn != null) {

conn.close();

}

}

}

}

ij>CALL sqlj.install\_jar( 'c:/mydata/SampleProcedure.jar', 'hsbc.SampleStoredProcedures',0);

CREATE PROCEDURE getEmployeeNameById (empid BIGINT, OUT NAME VARCHAR(255)) LANGUAGE JAVA EXTERNAL PARAMETER STYLE JAVA ‘samplestoredprocedures.SampleStoredProcedures.getCustomerLastName’;