

## Авторское право



Copyright 2007 Roman V. Babenko.

mailto: [RomanVBabenko@gmail.com](mailto:RomanVBabenko@gmail.com)

## TODO

- Система авторизации и шифрования данных([derbydev.pdf/Derby and Security](#));
- Пример для соединения клиент-сервер;
- Комментарии в примерах;
- Отредактировать вступление «Проект Apache Derby project»
- Описать возможности и сравнить с Firebird;
- Подготовить архив с примерами;
- Дописать «ij. Интерактивное выполнение команд»

## Проект Apache Derby project

Целью проекта Apache Derby project является создание базы данных с открытым исходным кодом, полностью написанной на языке программирования Java, простой в применении, но при этом подходящей для большинства приложений. В 1996 году новой компании под названием Cloudscape, Inc. были выделены средства на проект по созданию сервера базы данных, написанного на языке Java. Первый продукт компании был выпущен через год, и со временем название продукта было изменено на Cloudscape. В 1999 году компания Cloudscape, Inc. была куплена корпорацией Informix Software, Inc., крупным разработчиком баз данных. В 2001 году Informix Software была приобретена корпорацией IBM, и система управления базами данных IBM Cloudscape™ использовалась в качестве встроенного механизма базы данных в нескольких продуктах IBM. В апреле 2004 года IBM безвозмездно передала программу управления базами данных Cloudscape фонду Apache Software Foundation; так родился проект Apache Derby project.

К тому времени база данных Cloudscape насчитывала приблизительно полмиллиона строк Java-кода, поэтому потребовалось некоторое время, чтобы преобразовать ее в Apache Derby project. По окончании инкубационного периода состоялся официальный выпуск Derby в июле 2005 года. Таким образом, хоть Derby и может показаться новичком в группе программ, этот проект разрабатывался более 10 лет.

Apache Derby написана на языке Java, поэтому может выполняться в любой системе, где есть подходящая виртуальная машина Java (JVM). Это означает, что Derby может выполняться практически в любой операционной системе, включая платформы Microsoft Windows®, Macintosh, Linux® и UNIX®. Derby может также выполняться на любой из трех Java-платформ: Java 2 Platform, Micro

Edition (J2ME); Java 2 Platform, Standard Edition (J2SE); и Java 2 Platform, Enterprise Edition (J2EE). Программное обеспечение Derby упаковано в файл Java-архива (JAR), размер которого составляет всего 2 Мбайт. Вследствие своего небольшого размера, база данных Derby может быть без затруднений включена в любое приложение.

Базу данных Derby можно использовать двумя способами:

- Как *встраиваемую базу данных*, при этом пользователь не знает о существовании базы данных. Приложение использует базу данных; как приложение, так и база данных выполняются в одной и той же виртуальной машине; база данных сохраняет данные в локальной файловой системе. В соответствии с этой моделью база данных взаимодействует только с приложением, которое выполняется в этой же виртуальной машине;
- Как *соединение клиент-сервер*; это самая традиционная модель, используется большинством коммерческих разработчиков. В соответствии с данной моделью приложение взаимодействует с базой данных через сетевое соединение, при этом приложение и база данных выполняются в разных виртуальных машинах. Сервер базы данных может осуществлять коммуникации с несколькими клиентскими приложениями.

## Загрузка и установка

Загружаем архив дистрибутива db-derby-10.2.2.0-bin.zip с сайта <http://db.apache.org/derby/>. Распаковываем его в каталог установки(прим. d:\usr\local\derby). Модифицируем файл autoexec.bat следующим образом:

```
@rem JVM Environment

set JAVA_HOME=c:\bin\java\jre1.5.0_06

@rem Apache Derby Environment

set DERBY_INSTALL=d:\usr\local\derby\db-derby-10.2.2.0-bin
set
CLASSPATH=%DERBY_INSTALL%\lib\derby.jar;%DERBY_INSTALL%\lib\derbytools.jar
```

То есть определяем переменную JAVA\_HOME – расположение виртуальной Java машины и DERBY\_INSTALL – каталога установки дистрибутива Derby. Устанавливаем так же переменную CLASSPATH, чтобы виртуальная машина могла находить нужные нам пакеты. Перегружаемся, чтобы изменения конфигурации вступили в силу. На этом этап установки и конфигурации закончен.

## Утилиты администрирования

Все необходимые приготовления для вызова системных утилит, входящих в состав Derby мы уже сделали. То есть включили в переменную окружения CLASSPATH пакеты derby.jar и derbytools.jar

Более подробно об использовании обслуживающих утилит смотрите в файле ..\docs\pdf\derbytools.pdf дистрибутива Derby.

## sysinfo

SysInfo – это утилита для получения информации о текущем Java окружении, а так же информации об установленном дистрибутиве Derby.

sysinfo.cmd

```
java org.apache.derby.tools.sysinfo > sysinfo.txt
```

```

----- Java Information -----
Java Version:      1.5.0_06
Java Vendor:       Sun Microsystems Inc.
Java home:         C:\bin\java\jre1.5.0_06
Java classpath:    d:\usr\local\derby\db-derby-10.2.2.0-
bin\lib\derby.jar;d:\usr\local\derby\db-derby-10.2.2.0-
bin\lib\derbytools.jar
OS name:           Windows XP
OS architecture:  x86
OS version:        5.1
Java user name:    romb
Java user home:    C:\Documents and Settings\romb
Java user dir:     D:\devel\java\com.devrona.derbyWorks\bin
java.specification.name: Java Platform API Specification
java.specification.version: 1.5
----- Derby Information -----
JRE - JDBC: J2SE 5.0 - JDBC 3.0
[D:\usr\local\derby\db-derby-10.2.2.0-bin\lib\derby.jar] 10.2.2.0 -
(485682)
[D:\usr\local\derby\db-derby-10.2.2.0-bin\lib\derbytools.jar] 10.2.2.0
- (485682)
-----
----- Locale Information -----
Current Locale :   [українська/Україна [uk_UA]]
...

```

Таким образом мы получили в файле sysinfo.txt следующую информацию и видим, что Derby у нас сконфигурирована правильно. Можно двигаться дальше.

## ij

ij – это интерактивная JDBC утилита для исполнения скриптов.

Возможны два способа применения ij: выполнение команд и пакетное исполнение SQL-скриптов.

## Интерактивное выполнение команд

*UnderConstruction*

## Пакетное выполнение скриптов

```

database.sql
...
-- Подключаемся к базе данных employee
-- jdbc:derby:<база_данных>;[опции]
-- в данном примере мы подключаемся локально по
-- относительному пути с помощью встроенного(embedded)
-- режима. create=true - указывает на то, что в случае
-- отсутствия сказано БД она должна быть создана.

CONNECT 'jdbc:derby:../database/employee;create=true';

-- Создаем две схемы EMP и CUST

CREATE SCHEMA EMP;
CREATE SCHEMA CUST;

-- Создаем таблицы EMPLOYEE и HISTIRY в схеме EMP
-- Поля ID автоинкрементальные и представляют собой
-- сурогатные ключи для уникальной идентификации записей в таблицах

CREATE TABLE EMP.EMPLOYEE (

```

```

        ID          BIGINT GENERATED ALWAYS AS IDENTITY,
        FIRST_NAME  VARCHAR(255),
        LAST_NAME   VARCHAR(255),
        DESCRIPTION  BLOB
    );

CREATE TABLE EMP.HISTORY (
    ID          BIGINT GENERATED ALWAYS AS IDENTITY,
    REF_ID      BIGINT,
    FULL_NAME   VARCHAR(510)
);

-- Создаем синоним для таблицы EMP.HISTORY в схеме CUST
-- Синоним в запросах неотличим от таблицы на которую он
-- ссылается

CREATE SYNONYM CUST.HISTORY FOR EMP.HISTORY;

-- Создаем триггер для логирования вставок записей
-- в таблицу EMP.EMPLOYEE

CREATE TRIGGER EMP_EMPLOYEE
AFTER INSERT ON EMP.EMPLOYEE
REFERENCING NEW AS NE
FOR EACH ROW MODE DB2SQL
INSERT INTO EMP.HISTORY(REF_ID, FULL_NAME) VALUES (NE.ID, NE.FIRST_NAME
|| ' ' || NE.LAST_NAME);

-- Создаем сранимую внешнюю функцию SGN в схеме CUST
-- Функция SGN - сигнум возвращает знак числа, или 0,
-- если число равно 0. Функция определена в классе
-- java.lang.Math

CREATE FUNCTION CUST.SGN(ARG0 DOUBLE) RETURNS DOUBLE
PARAMETER STYLE JAVA NO SQL LANGUAGE JAVA
EXTERNAL NAME 'java.lang.Math.signum';

-- Делаем вставки в таблицу EMP.EMPLOYEE

INSERT INTO EMP.EMPLOYEE(FIRST_NAME, LAST_NAME) VALUES ('Vasia',
'Pupkin');

INSERT INTO EMP.EMPLOYEE(FIRST_NAME, LAST_NAME) VALUES ('Petia',
'Korjuk');

-- Выходим из пакетного режима

EXIT;

```

Запускаем данный скрипт с помощью пакетного файла нижеследующего содержания, предварительно создав структуру каталогов:

```

database
make
    database.cmd
src
    sqlscripts
        database.sql

```

**database.cmd**

```

@echo off

java org.apache.derby.tools.ij ..\src\sqlscripts\database.sql >
database.out.txt

```

```
@echo on
```

В результате выполнения скрипта в каталоге `.\database` появится системный каталог базы данных `employee`.

## dblook

`dblook` – утилита предназначенная для полного или частичного извлечения метаданных(DDL) в скрипт.

Создадим и выполним пакетный файл `dblook.cmd` для созданной базы данных `employee`

```
dblook.cmd
...
java org.apache.derby.tools.dblook -d
'jdbc:derby:..\database\employee' -o .\employee.sql
```

После выполнения в файле `employee.sql` будет находится скрипт метаданных с помощью которого база была создана.

Возможно так-же частичное извлечение метаданных. Для этого применяются соответствующие опции.

## Архитектура

Derby реализована в двух архитектурах: встроенный сервер и клиент-сервер.

### Встроенный сервер

Архитектура встроенного сервера предоставляет однопользовательский интерфейс доступа к базе данных. В этом варианте Apache Derby являться частью приложения и обеспечивает доступ из одной Java-машины в которой это приложение запущено.

Строка соединения в этом случае выглядит следующим образом:

```
jdbc:derby:<путь_к_базе_данных>; [опции]
```

Имя драйвера для соединения:

```
org.apache.derby.jdbc.EmbeddedDriver
```

### Клиент-сервер

Запуск Derby в режиме клиент-сервер дает возможность осуществлять многопользовательский доступ к базе данных. Тоест несколько Java-машин могут работать с данными одновременно. Причем размещение клиентских приложений не имеет значения: локально, сеть, интернет.

Строка соединения в этом случае выглядит следующим образом:

```
jdbc:derby://<сервер>:<порт>/<путь_к_базе_данных>; [опции]
```

Имя драйвера для соединения:

```
org.apache.derby.jdbc.ClientDriver
```

## Запуск и остановка сервера

Если у вас правильно установлены все переменные окружения, то старт и оста-

новка сервера осуществляются с помощью пакетных файлов, которые находятся в каталоге **\$(DERBY\_INSTALL)\bin**. По умолчанию порт сервера имеет значение **1527**.

```
//Запуск сервера
startNetworkServer.bat

Apache Derby Network Server - 10.2.2.0 - (485682) started and ready to
accept connections on po
rt 1527 at 2007-06-10 11:49:14.046 GMT
...
// Остановка сервера
stopNetworkServer.bat

Apache Derby Network Server - 10.2.2.0 - (485682) shutdown at 2007-06-
10 11:50:26.078 GMT
```

## Derby4Eclipse

Eclipse (<http://eclipse.org>) – *UnderConstruction*

Для интеграции Derby и Eclipse необходимо скачать архивы derby\_core\_plugin\_10.2.2.485682.zip и derby\_ui\_plugin\_1.1.0.zip и распаковать их в каталог eclipse\plugins. После чего перезапустить eclipse с ключем clean:

```
eclipse.exe -clean
```

## Примеры работы с Derby на Java

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class DerbyEmbeddedConnection {
    static final String drv = "org.apache.derby.jdbc." +
        "EmbeddedDriver";

    static final String databaseURL = "D:\\devel\\java\\"
        + "com.devrona.derbyWorks\\database\\employee";

    static final String url = "jdbc:derby:" + databaseURL +
        ";create=true";

    public static void main(String[] args) throws
        InstantiationException, IllegalAccessException,
        ClassNotFoundException, SQLException {
        Connection conn = null;
        Statement stmt = null;
        ResultSet rs = null;
        Class.forName(drv).newInstance();
        conn = DriverManager.getConnection(url);
        stmt = conn.createStatement();
        rs = stmt.executeQuery("SELECT * FROM EMP.EMPLOYEE");
        while (rs.next()) {
            System.out.println(rs.getString("FIRST_NAME")
                + " " + rs.getString("LAST_NAME"));
        }
        rs.close();
        stmt.close();
    }
}
```

```

        conn.close();
    }
}

```

```

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class DerbyNetworkConnection {
    static final String drv = "org.apache.derby.jdbc." +
        "ClientDriver";

    static final String databaseURL = "D:\\devel\\java\\"
        + "com.devrona.derbyWorks\\database\\employee";

    static final String url = "jdbc:derby://localhost:1527/"
        + databaseURL + ";create=true";

    public static void main(String[] args) throws
        InstantiationException, IllegalAccessException,
        ClassNotFoundException, SQLException {
        Connection conn = null;
        Statement stmt = null;
        ResultSet rs = null;
        Class.forName(drv).newInstance();
        conn = DriverManager.getConnection(url);
        stmt = conn.createStatement();
        rs = stmt.executeQuery("SELECT * FROM CUST.HISTORY");
        while (rs.next()) {
            System.out.println(rs.getString("FULL_NAME"));
        }
        rs.close();
        stmt.close();
        conn.close();
    }
}

```

## Firebird & Derby

\* Имена метаданных:

Длина имен метаданных до 128 символов.

\* Схемы:

Позволяют группировать таблицы в слабосвязанные системы по какому либо признаку.

Например:

SYS.\* - системные таблицы

SALES.\* - таблицы относящиеся к продажам.

\* Синонимы:

Позволяет создавать символьные ссылки на таблицы. Синоним может быть как в пределах схемы в которой расположена таблица, так и в другой схеме. В запросах синоним ведет себя аналогично таблице или виду.

Например:

```
CREATE SYNONYM CUST.HISTORY FOR EMP.HISTORY;
```

\* Домены:

отсутствуют;

\* Генераторы и последовательности:

отсутствуют.

Идентификация происходит с помощью автоинкрементных полей определенных как GENERATE ALWAYS AS IDENTITY или GENERATE BY DEFAULT AS IDENTITY. Второй вариант позволяет вставлять значения самостоятельно и следовательно не гарантирует уникальности.

Возможны установки начального значения и шага инкремента((START WITH 2, INCREMENT BY 1)).

После создание автоинкрементального поля индекс для него не создается автоматически.

Автоинкрементальные столбцы вне транзакций. То есть откат транзакции не вызывает отката

счетчика для столбца.

\* Функции и процедуры:

SQL реализация отсутствует. Писать можно на Java проецируя на классы.

Хранимые процедуры скорее похожи на UDF хотя и могут возвращать набор данных.