An introduction to backup, restore, and rollforward recovery in IBM Cloudscape/Apache Derby

Suresh Thalamati

February 17, 2005

Ensure the recoverability of your IBM® Cloudscape™ data. Learn how the backup, restore, and rollforward recovery features work in Cloudscape and Apache Derby. Through examples, find out how to restore your databases.

Introduction

Backup and restore capabilities are key to ensuring data recoverability for any database management system. Applications can crash, disks can fail, and often users can make mistakes that compromise data. For these reasons, it is critical that you understand how backup and recovery features function for your database management system, and that you have a well-planned backup strategy implemented.

Note: IBM Cloudscape is the commercial release of the open source Apache Derby relational database. When this article refers to "Derby," it's referring to either Cloudscape or Apache Derby.

Derby provides two types of restore mechanisms:

- The first returns the database to the state it was in when the backup was taken using a full backup of the database.
- The second restores to the most recent state, using a combination of full backup plus rolling forward the transaction log, called rollforward recovery.

Database backups can be performed online using system procedures, or offline using operating system copy commands. Restore operations are available through connection URL attributes.

You can initiate backup or restore from ij (the Derby command line tool), or from a Java™ Database Connectivity (JDBC) application. Backup and restore operations are supported in both embedded and network server configurations. If you are performing backup/restore from a network server client make sure that the backup locations you have specified are accessible from your network server machine.
Online backup

An online backup is performed when the database system is running using the `SYSCS_UTIL.SYSCS_BACKUP_DATABASE(IN_BACKUPDIR VARCHAR(32762))` procedure. A connection to the database being backed up is required to execute the backup procedure.

What is a transaction log?

The transaction log stores changes that are made to the database; it is required to recover from power failures or application crashes and to restore the database from backups. Derby stores the transaction log as a sequence of files in the file system, the transaction log file names have prefix `log` and suffix `dat`, a number that gets incremented on every new transaction log file creation is added to the prefix, like `log1.dat`, `log2.dat` and etc. The transaction log files are stored in the `log` directory under the database directory or a user-specified location.

In Derby, while backup is in progress, the execution of statements that require writes to the disk like insert, delete, and updates are blocked. Statements that require reads from the disk only, like select, proceed unblocked. Once the backup is complete, the system will let all blocked statements complete their execution. Online backup may decrease system throughput and user response time, so when possible schedule backups when update activity is low.

Listing 1: Performing a backup of *salesdb* database through IJ

```sql
ij>connect 'jdbc:derby:salesdb';
ij>CALL SYSCS_UTIL.SYSCS_BACKUP_DATABASE('D:/dbbackups/');
```

An embedded application can execute backups using the JDBC API; the application client does not need to administrate the backup. In Derby, all the system procedures can be invoked through the JDBC API also.

Listing 2: Performing a backup using JDBC calls

```java
private void backUpDatabase(Connection conn) throws SQLException {
    String sqlstmt = "CALL SYSCS_UTIL.SYSCS_BACKUP_DATABASE(?)";
    CallableStatement cs = conn.prepareCall(sqlstmt);
    cs.setString(1,"D:/dbbackups/");
    cs.execute();
    cs.close();
}
```

The backup process creates a directory with the same name as the database at the specified backup location. This directory contains all the files/directories that are in the database directory. The database directory typically has a data directory (`seg0`) to store all the data files for the tables and indexes, a transaction log directory (`log`) to store all the transaction log files, and a `service.properties` file that contains information to boot the database.

Offline backup

The entire database directory can be copied when the system is offline using operating system commands. If the transaction log directory is different from the database directory, make sure that the transaction log directory is also copied. During the restore operation, transaction logs are necessary to bring the database to a consistent state.
Offline backup can also be performed without bringing the database system offline by freezing the system (blocking all write operations) while copying the database directory. The system procedure to do that is `SYSCS_UTIL.SYSCS_FREEZE_DATABASE()`. Once the copy is complete, unfreeze (allow write operations) the system using the `SYSCS_UTIL.SYSCS_UNFREEZE_DATABASE()` procedure; this procedure call will set the system back to normal state.

### Restore from a backup

The online/offline backup image of a database is used to restore the database to the state it was in, when the backup image was taken. The database is restored from the backup by specifying the connection URL attribute `restoreFrom=<backup path>` on the first connection to the database. Note that the backup path must include the database name in the backup, not just the backup location.

The database will be restored to the location specified on the connection URL or to the current `derby.system.home` location if no path is specified on the connection URL. If a database with the same name exists at the current database location it will be removed first, and then restored from the backup.

**Listing 3: Restoring the `salesdb` database from backup:**

```java
String dbURL = "jdbc:derby:salesdb;restoreFrom=D:/dbbackups/salesdb;";
Connection conn = DriverManager.getConnection(dbURL);
```

By default, the transaction log will be copied back to the same location that it was in, when the backup image was taken. The transaction log location can be changed during restore in case there is ever a problem with the disk that has the transaction log. To do this, specify the connection URL attribute `logDevice=<log dir>`.

**Listing 4: Restoring the `salesdb` database from a backup and placing the transaction log in different location:**

```java
String dbURL = "jdbc:derby:salesdb;restoreFrom=D:/dbbackups/salesdb;logDevice=E:/salesdbLog;";
Connection conn = DriverManager.getConnection(dbURL);
```

### Move or clone a database

In Derby, moving databases from one system to another, cloning a database for testing, and making enhancements to a database are easy tasks. Databases can be recreated from the backup by using the connection URL attribute `createFrom=<backup path>` on the first connection to the database. The database will be created at the location specified on the connection URL or at the current `derby.system.home` location if no path is specified on the connection URL. By default, the transaction log is stored under the database directory, the transaction log location can be changed by specifying the `logDevice=<log dir>` attribute on the connection URL.

**Listing 5: Cloning the `salesdb` database using backup image**

```java
String dbURL = "jdbc:derby:salesdb;createFrom=D:/dbbackups/salesdb;logDevice=E:/salesdbLog;";
Connection conn = DriverManager.getConnection(dbURL);
```
With Derby, a database can be recreated from the backup on any operating system or hardware that supports Java. It is not necessary for the new system to have the same configuration as the database on which the backup was taken.

**Rollforward recovery**

Using just the online/offline backup images makes it possible to restore a database to the state it was in when the backup was taken. However, using the rollforward-recovery feature makes it possible to restore a database to the most recent state. Rollforward recovery first does a restore from the backup, and then applies all the transaction log records after the backup image is made, which brings the database to its most recent state. The information that you need to perform rollforward recovery is:

- A full online backup image
- All transaction log files after the backup

By default, only the transaction log files that the system keeps are the ones that are required to recover from a power failure or an application crash, all the other transaction log files are deleted in a successful checkpoint operation. In order to have the option to perform the rollforward recovery, all the transaction log files that are generated after the backup need to be archived. You can do this by enabling the log archive mode as part of the backup. The system procedure to do that is:

```sql
SYSCS_UTIL.SYSCS_BACKUP_DATABASE_AND_ENABLE_LOG_ARCHIVE_MODE(IN BACKUPDIR VARCHAR(32762), IN SMALLINT DELETE_ARCHIVED_LOG_FILES)
```

This procedure will perform the backup, enable the transaction log archive mode, and if a non-zero value is passed as argument for `DELETE_ARCHIVED_LOG_FILES` parameter, it will also delete the transaction log files that were archived to restore from previous backups.

All the archived transaction log files are also stored in the database transaction log directory. Currently, Derby does not have support to ship archived transaction logs to a different location. It is important to make sure that the disk that contains the transaction log directory is well protected using some kind of fault-tolerant mechanism-like mirroring.

**Listing 6: Perform a backup, enable the transaction log archive mode, and preserve archived transaction logs before this backup**

```java
public void backupAndEnableLogArchiveMode(Connection conn) throws SQLException {
    String sqlstmt = "CALL SYSCS_UTIL.SYSCS_BACKUP_DATABASE_AND_ENABLE_LOG_ARCHIVE_MODE(? , ?)";
    CallableStatement cs = conn.prepareCall(sqlstmt);
    cs.setString(1, "D:/dbbackups/");
    cs.setInt(2, 0);
    cs.execute();
    cs.close();
}
```
Listing 7: Perform a backup, enable the transaction log archive mode, and delete archived transaction logs before this backup

```java
public void backupAndEnableLogArchiveMode(Connection conn) throws SQLException {
    String sqlstmt = "CALL SYSCS_UTIL.SYSCS_BACKUP_DATABASE_AND_ENABLE_LOG_ARCHIVE_MODE(?, ?)";
    CallableStatement cs = conn.prepareCall(sqlstmt);
    cs.setString(1, "D:/dbbackups/");
    cs.setInt(2, 1);
    cs.execute();
    cs.close();
}
```

Restoring the database using rollforward-recovery

A database can be restored from the backup with rollforward recovery by specifying the connection URL attribute `rollForwardRecoveryFrom=<backup path>` on the first connection to the database. Note that the backup path must include the database name in the backup, not just the backup location.

The database will be restored to the location specified on the connection URL or to the current derby.system.home location if no path is specified on the connection URL. By default rollforward recovery assumes that this location is where the database existed previously. When you restore the database to a new location use `logDevice=<log dir>` attribute to specify the transaction log location, this should be in the same place where the transaction log was stored before.

Listing 8: Restoring the `salesdb` from backup with rollforward recovery:

```java
String dbURL = "jdbc:derby:salesdb;rollForwardRecoveryFrom=D:/dbbackups/salesdb";
Connection conn = DriverManager.getConnection(dbURL);
```

Disabling the transaction log archive mode

Enabling the transaction log archive mode to have the option to perform rollforward recovery in case of failure is excellent, but if the system is running out of transaction log storage space it might be better to disable the transaction log archive mode temporarily than let the system go offline. The system procedure to disable transaction log archiving is `SYSCS_UTIL.SYSCS_DISABLE_LOG_ARCHIVE_MODE(IN_SMALLINT_DELETE_ARCHIVED_LOG_FILES )`.

One side effect of enabling the transaction log archive mode is all the data changes (inserts) get logged, whereas by default they are not logged in the following two cases:

- Index creation on a table that contains data
- Import of external data into an empty table using Import procedures

If performance is important in the above cases, it would be a good idea to disable transaction log archive mode for the duration of their execution.
Listing 9: Disable the transaction log archive mode, and delete all the archived transaction log files

```java
public void disableLogArchiveMode(Connection conn) throws SQLException {
    String sqlstmt = "CALL SYSCS_UTIL.SYSCS_DISABLE_LOG_ARCHIVE_MODE(?)";
    CallableStatement cs = conn.prepareCall(sqlstmt);
    cs.setInt(1, 1);
    cs.execute();
    cs.close();
}
```

Checking if the transaction log archive mode is on

You can find out if the transaction log archive mode is enabled for a database by checking the database property `derby.storage.logArchiveMode`. If the transaction log archive mode is enabled, this property will be set to `true`.

Listing 10: Checking for the transaction log archive mode status?

```java
private boolean isLogArchiveModeOn(Connection conn) throws SQLException {
    Statement stmt = conn.createStatement();
    String sqlstmt = "VALUES SYSCS_UTIL.SYSCS_GET_DATABASE_PROPERTY('derby.storage.logArchiveMode')";
    ResultSet rs = stmt.executeQuery(sqlstmt);
    rs.next();
    boolean logArchiveStatus = rs.getBoolean(1);
    rs.close();
    stmt.close();
    return logArchiveStatus;
}
```

Backup scheduler

Taking backup images in an ad-hoc manner works for applications with less-frequent changes to the data. If your database is being changed more frequently, it is good to have some kind of automatic mechanism to schedule backups at regular intervals. If the application already performs timed tasks, backups could be added to the existing list; otherwise, it is important to implement some mechanism that will schedule database backups at regular intervals. Following is example code that you can use in an application code to schedule backups once a day:

Listing 11: BackupScheduler.java

```java
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.CallableStatement;
import java.sql.SQLException;
import java.util.Timer;
import java.util.TimerTask;
import java.util.Calendar;
import java.util.Date;
import java.text.SimpleDateFormat;

/*
 * This class implements a backup scheduler to schedule database backups once a day at a specified time after a day it is started.
 * Example usage in an application:
 * BackupScheduler backups = new BackupScheduler("jdbc:derby:salesdb", "D:/backups");
 */
An introduction to backup, restore, and rollforward recovery in IBM Cloudscape/Apache Derby

```java
public class BackupScheduler extends TimerTask {
    private final static long BACKUP_INTERVAL = 1000*60*60*24; // once a day
    private String dbURL;
    private String backupPath;
    private Timer timer;

    /* Constructor for Backup Scheduler
     * @param dbURL connection URL that should be used to connect to the database.
     * @param backupPath Location where the backup should be placed.
     */
    public BackupScheduler(String dbURL, String backupPath) {
        this.dbURL = dbURL;
        this.backupPath = backupPath;
    }

    /**
     * start the backup scheduler.
     * @param backupTime the time of the date at which to start the backup
     * in 24-hour clock notation.(eg: 1PM 13)
     */
    public void start(int backupTime) {
        timer = new Timer();
        // schedule a backup task everyday at the specified time, starting tomorrow.
        timer.scheduleAtFixedRate(this,
                                   getTomorrowTime(backupTime),
                                   BACKUP_INTERVAL);
    }

    /**
     * Implements TimerTask's run method to perform backups.
     */
    public void run() {
        try {
            new org.apache.derby.jdbc.EmbeddedDriver();
            Connection conn = DriverManager.getConnection(dbURL);
            backupDatabase(conn);
            conn.close();
        } catch (Exception ex) {
            System.out.println("backup failed on:" + dbURL);
            ex.printStackTrace();
        }
    }

    /**
     * Performs back up of the database
     * @param conn Connection to the database that is to be backed up.
     */
    private void backupDatabase(Connection conn) throws SQLException {
        SimpleDateFormat dateFormat = new SimpleDateFormat("MM-dd-yyyy");
        String backupDirectory = backupPath + "/" + dateFormat.format(new Date());
        String sqlstmt = "CALL SYSCS_UTIL.SYSCS_BACKUP_DATABASE(?)";
        CallableStatement cs = conn.prepareCall(sqlstmt);
        cs.setString(1, backupDirectory);
        cs.execute();
        cs.close();
    }
}
```
/**
 * Gets the tomorrow's calendar time at the specified hour.
 * @param hourOfDay hour the time to be calculated.
 * @return returns tomorrow time.
 */
private Date getTomorrowTime(int hourOfDay)
{
    Calendar tomorrow = Calendar.getInstance();
tomorrow.roll(Calendar.DATE, true);
tomorrow.set(Calendar.HOUR_OF_DAY, hourOfDay);
tomorrow.set(Calendar.MINUTE, 0);
tomorrow.set(Calendar.SECOND, 0);
tomorrow.set(Calendar.MILLISECOND, 0);
return tomorrow.getTime();
}

Conclusion

In conclusion, IBM Cloudscape/Apache Derby makes performing backup and restore functions very easy. A database can be protected from the disk failures to a varying degree, depending on the resources and the application by setting up the system with any one of the following configurations:

- Scheduled online/offline backups at regular intervals, using this configuration database, can be restored to the state it was at the last successful backup.
- Online backups with transaction log archive mode enabled at regular intervals and the transaction logs are stored on a fault-tolerant disk. Using this configuration the database can be restored to the most recent state using rollforward recovery.
- Hardware/software fault-tolerant techniques like mirroring to protect the disks on which the database and transaction log information is stored. In this case no need of backup/restore as disks are well protected.
Related topics

- The IBM Cloudscape documentation will provide you with in-depth details on Derby.
- The Apache Derby site contains lot of information about the Derby engine and hosts mail lists for developers and users.

© Copyright IBM Corporation 2005
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)