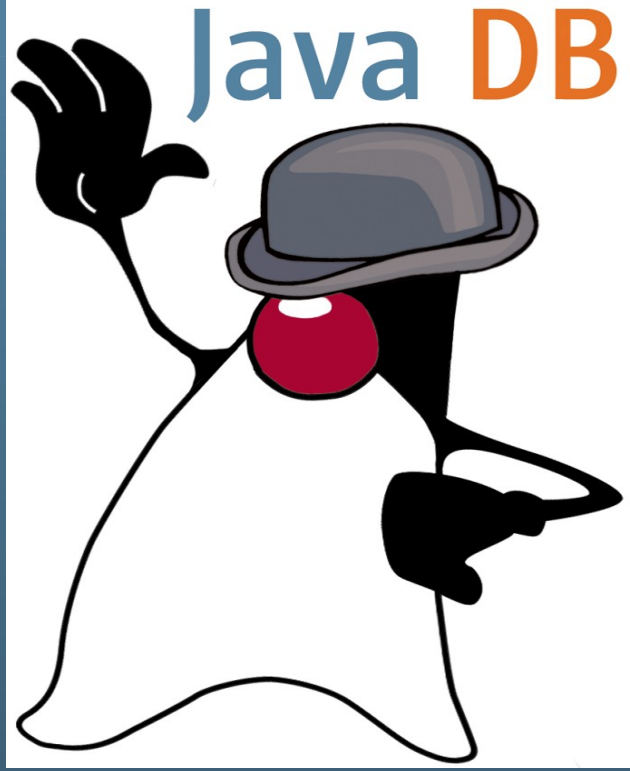

JavaOne™

ORACLE

Java DB in JDK 7: A Free, Feature-Rich, Embeddable SQL Database

dag.wanvik@oracle.com
senior database engineer

CON5141

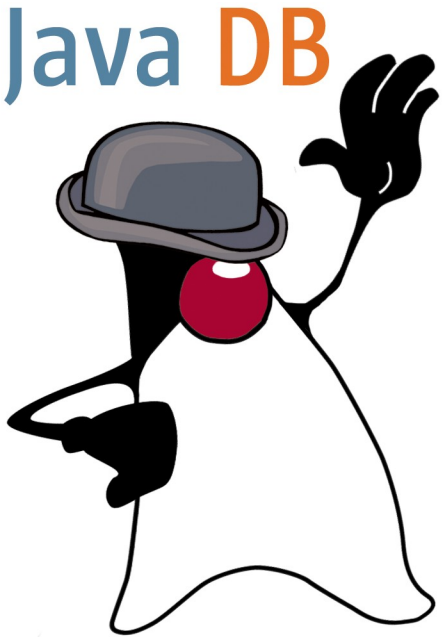


Summary

- In Oracle JDK 7, a small foot-print, pure Java, embeddable SQL data base engine
- Easy to develop and deploy small data rich apps on Java DB
- Zero admin: see no database
- Upgrade on to enterprise level DB when needed

Objectives

- Show how to get quickly started with Java DB in JDK 7
- Show how to build and deploy portable, data-rich applications with Java DB, using code examples
- Give you a sense of its capabilities



Program Agenda

- Availability
- Ease of use: development
- Ease of use: deployment
- Features
- Robustness
- Performance

Where can I find Java DB?

- In the Oracle Java SE JDK (not in the JRE or OpenJDK)
- Java SE Downloads: JDK 7
  - http://www.oracle.com/technetwork/java/javase/downloads/index.html
- After install of JDK, jar files in:
  - Windows: C:\Program Files\Java\jdk1.7.0\db\lib\
  - Linux: /usr/jdk/jdk1.7.0/db/lib
- Also now in Oracle Java Embedded Suite [CON6684 Data Storage for Embedded Middleware]

Where can I find Java DB? [2]

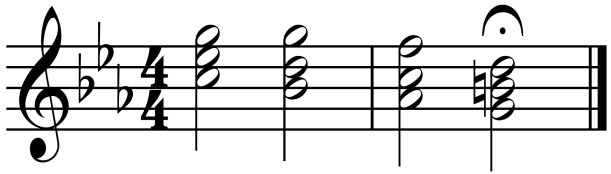
Developed in the open source Apache Derby project; same bits

- Latest version
- Archived versions
- Debug versions
- http://db.apache.org/derby/derby_downloads.html
-

Where can I find Java DB ? [3]

```
$ ls “$JDK_ROOT”/db/lib
derby.jar
derbyclient.jar
derbynet.jar
derbytools.jar
derbyrun.jar
derbyLocale_cs.jar
derbyLocale_de_DE.jar
...
```

Release cadence in Derby



10.8.2.2

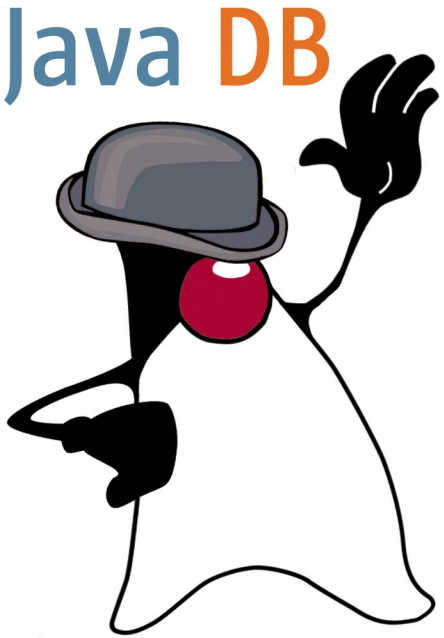
Feature release

Maintenance release

- One feature release per year in Derby project (+/-):

  may contain interface changes

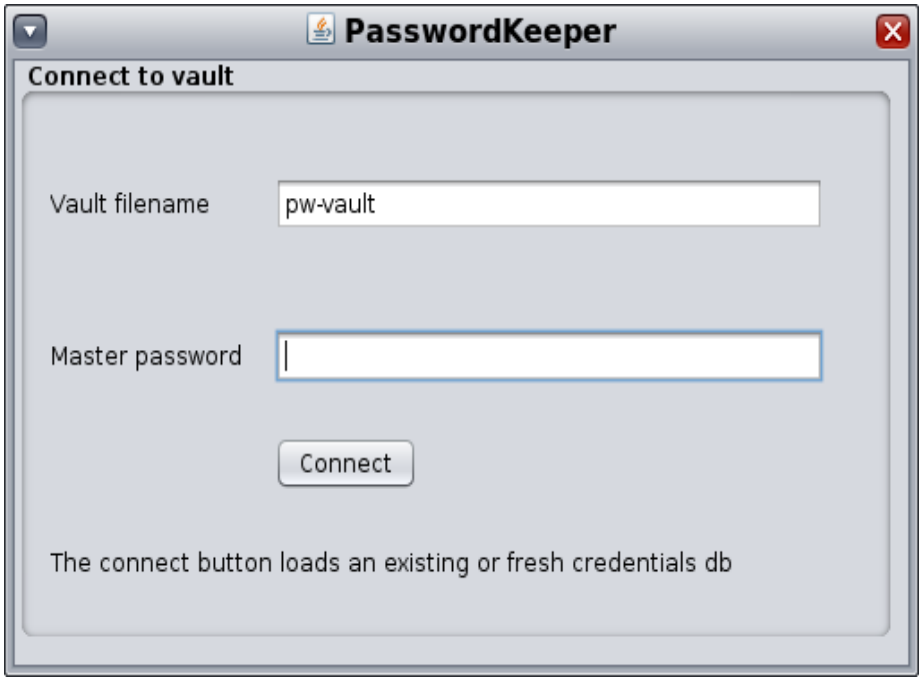
- Maintenance releases as needed: *no interface changes*
- JDK 7u7: Java DB 10.8.2.*2*



Program Agenda

- Availability
- Ease of use: development
- Ease of use: deployment
- Features
- Robustness
- Performance

Running example: simple password keeper



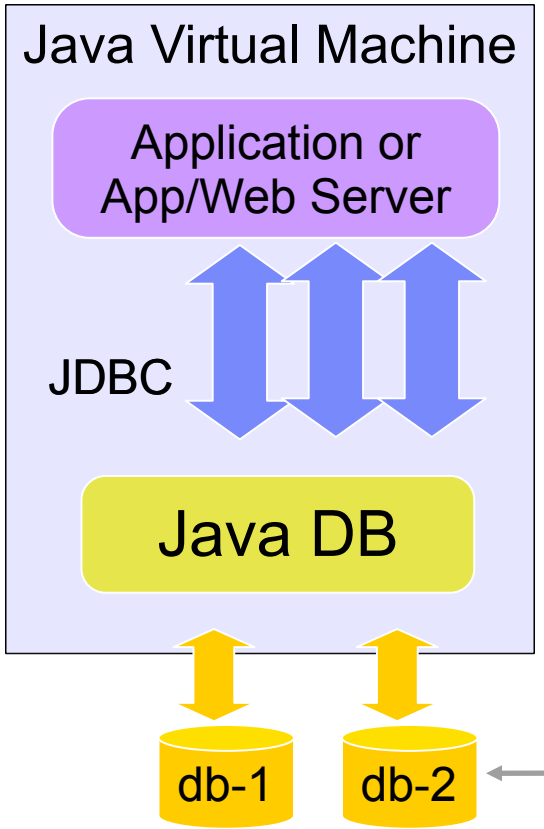
- Simple GUI application to illustrate use of Java DB
- Keep account credentials in an encrypted database
- Easy insert, update, delete and lookup

Source at http://javadesktop.org/javadb/j1_2012_con5141/

Ease of use: cut development time

- Oracle JDK 7: You already have Java DB!
- No install, setup, or complicated configuration: start using it by adding a jar to the app's classpath
- Prototype, test before deploying apps to enterprise DB
- Give query capabilities to apps with persistent data: replace key/value storage or files
- For free: data export/import from/to application, platform neutral format
- Easy schema evolution (e.g. add column)

Embedded Java DB

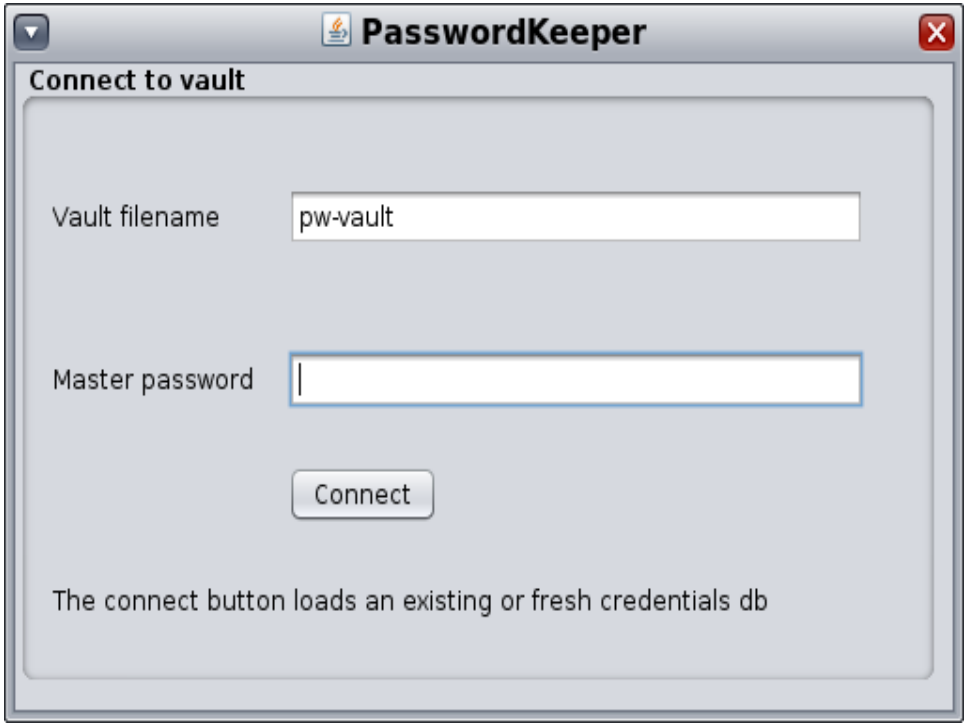


- In application's JVM: JDBC calls call straight into engine
- No administrative set-up
- Hides there's a data base
- Multiple, concurrent connections
- Multiple databases, one engine
- Conservative memory defaults

Embedded Java DB: use cases

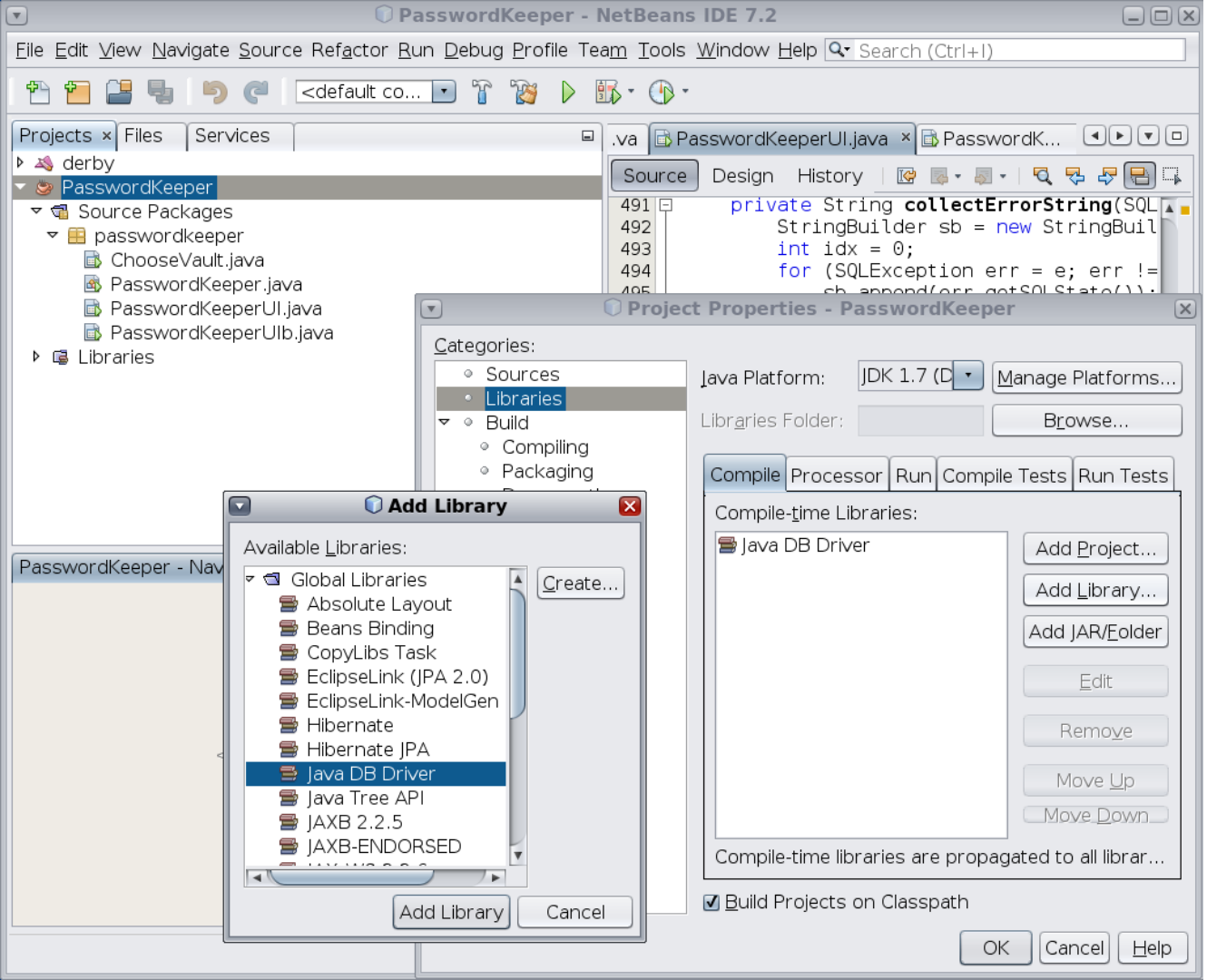
- Need relational database?
  - Queries across object types (key/value pairs not enough), reporting; e.g. joined tables
  - Correctness, no loss of data (ACID)
- Embedded rather than client/server?
  - Reduced cost (local machine only)
  - Ease of development, deployment, no DBA
  - Security (data is local)
  - Speed: no protocol step

Example: simple password keeper



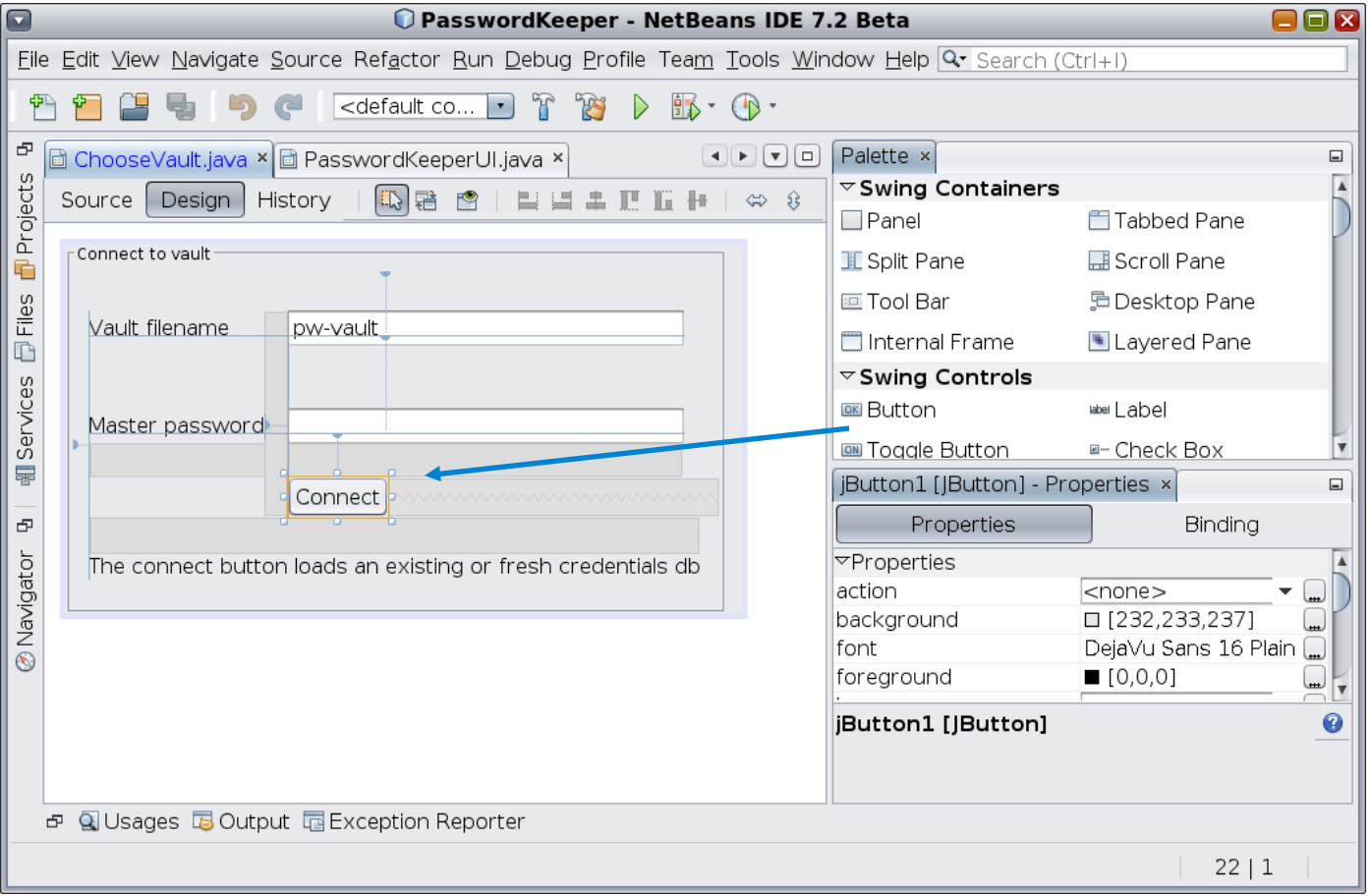
- Self contained app: create db(s), insert, update, delete and query credentials
- Encrypted, embedded db
- NetBeans GUI builder
- Ca 450 LOC to write, 50% db access, 50% GUI.

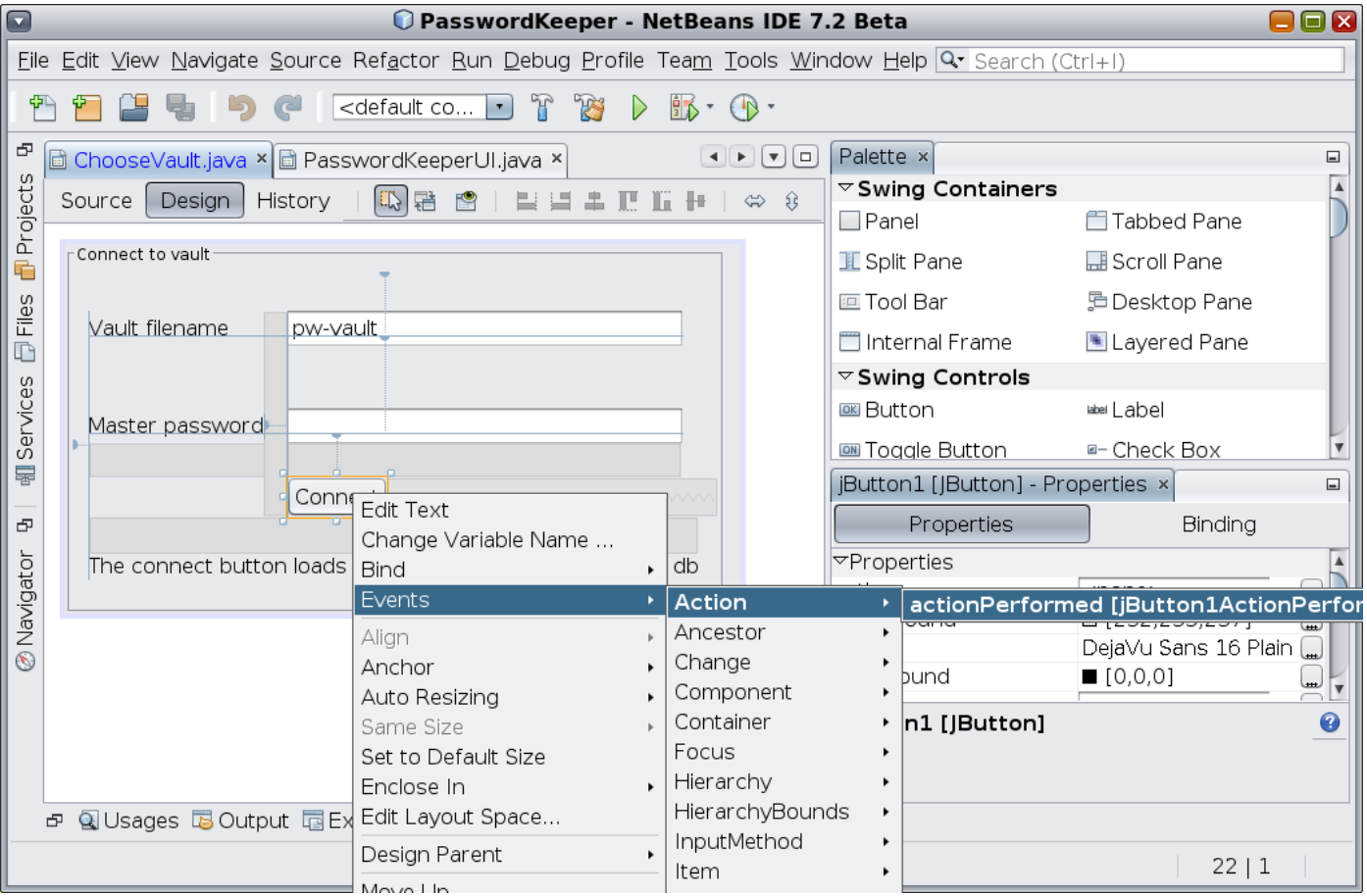
Add Java DB as a library to the NetBeans project



JavaOne ORACLE

GUI builder: drag & drop





GUI builder: drag & drop

Connect button actions to DB operations

Example: simple password keeper - connect [1]

```
// Connect button
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {

    char[] pw = jPasswordField1.getPassword();
    ...

    String dbFileName = jTextField1.getText();
    PasswordKeeperUI ui = (PasswordKeeperUI)getParent();

    boolean success = ui.loadDatabase(dbFileName, pw);


    ...
    if (success) {
        this.dispose();
    }
}
```

Example: simple password keeper - connect [2]

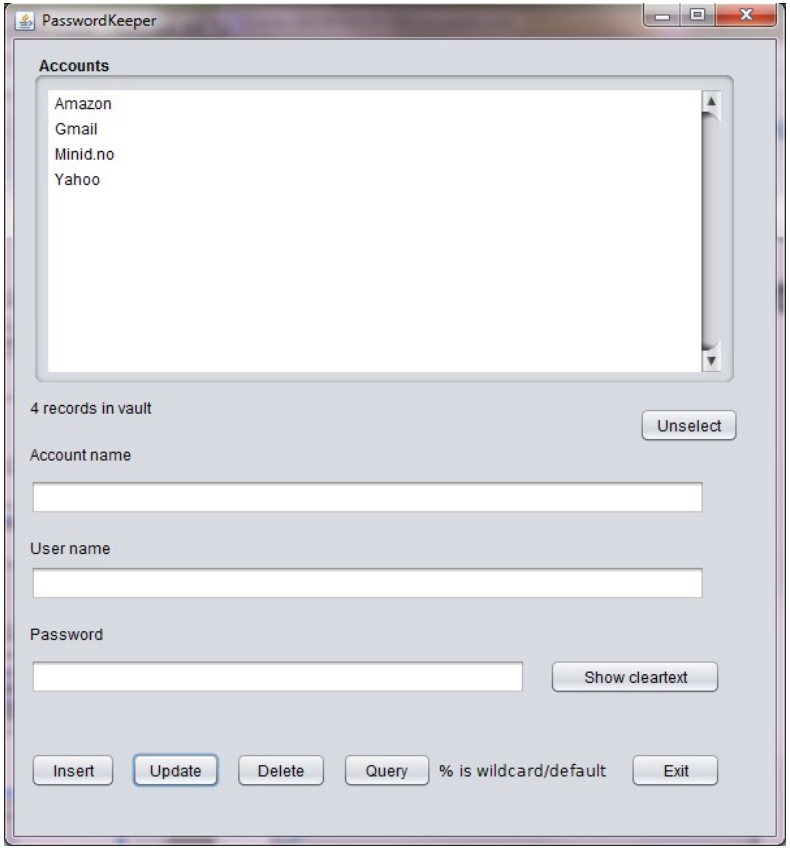
Embedded URL
Starts Java DB engine (if not done)
Creates (or opens) database
Specifies data encryption

```
void loadDatabase(String dbFileName, char[]pw) {
    ...
    try {
        conn = DriverManager.getConnection(
            “jdbc:derby:” + dbFileName + ”;create=true;dataEncryption=true;” +
            “bootPassword=” + String.valueOf(pw));

    ...

} catch (SQLException e) { ...
}
```

Example: simple password keeper - connect [3]

```
boolean loadDatabase(String dbFileName, char[]pw) {
  ...
  try {
     if (db.isDirectory()) {
         conn = DriverManager.getConnection("jdbc:derby:" + dbFileName +
                ";bootPassword=" + String.valueOf(pw));
     } else {
         conn = DriverManager.getConnection("jdbc:derby:" + dbFileName +
                ";create=true;dataEncryption=true;bootPassword=" +
                String.valueOf(pw));

         Statement s = conn.createStatement();
         s.executeUpdate("create table credentials(" +
                         "id VARCHAR(256), " +
                         "principal VARCHAR(256), " +
                         "pw VARCHAR(256))");
         s.close();
```

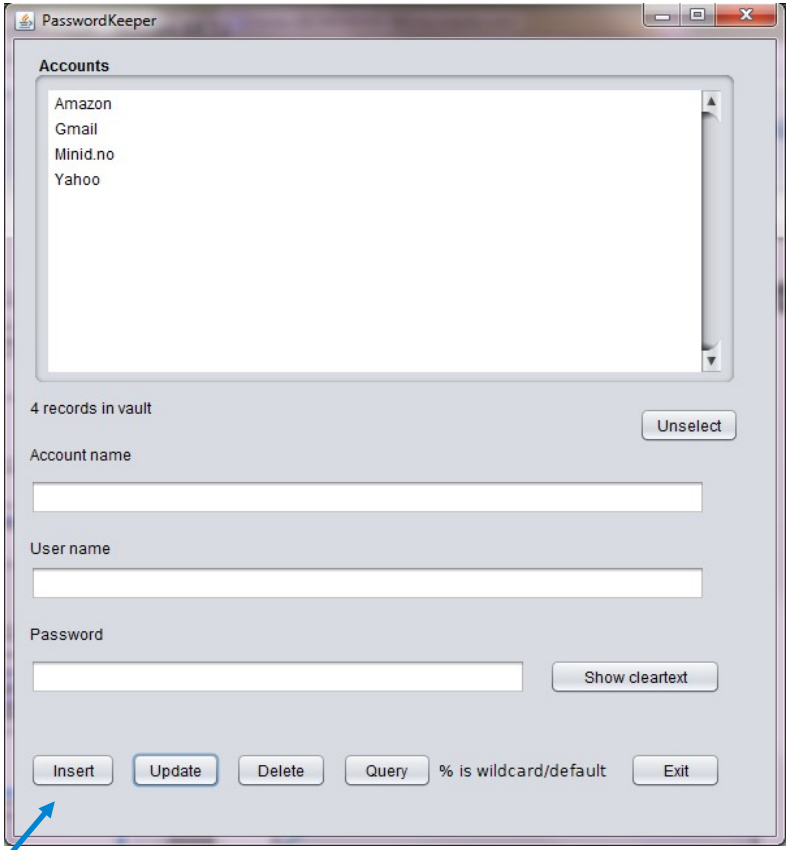
Insert account credentials



New account:

- insert data into the three fields and click “Insert”

Insert account credentials



Insert button is jButton6 in generated code

Example: simple password keeper - insert

```
// Insert button
private void jButton6ActionPerformed(java.awt.event.ActionEvent evt) {
    ...
    insertCredential(
        jTextField2.getText(),
        jTextField4.getText(),
        jPasswordField2.getPassword());
    displayCredentialIds();
    ...
}
```

Example: simple password keeper – insert [2]

```
PreparedStatement insertCredential = conn.prepareStatement(
        "insert into credentials values (?,?,?)");
```

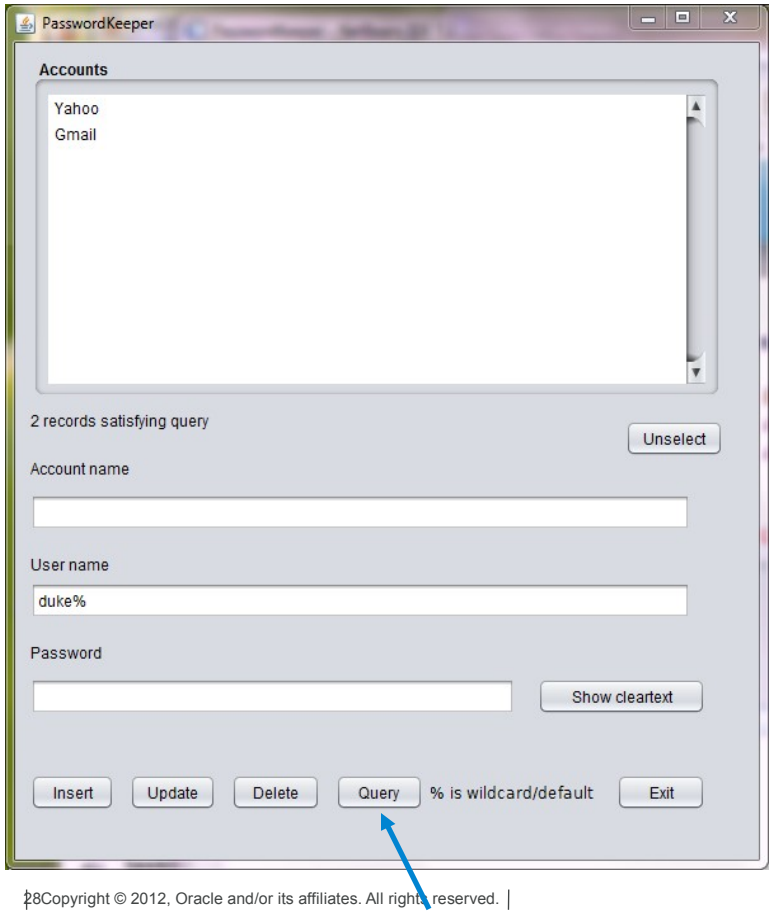
```
// Insert button
private void jButton6ActionPerformed(java.awt.event.ActionEvent evt) {
    ...
    insertCredential(
        jTextField2.getText(),
        jTextField4.getText(),
        jPasswordField2.getPassword());
    displayCredentialIds();
    ...
}

private void insertCredential(String newId, String principal, char[] pw) {
    ...
    try {
        insertCredential.setString(1, newId);
        insertCredential.setString(2, principal);
        insertCredential.setString(3, String.valueOf(pw));
        insertCredential.executeUpdate();
    } catch (SQLException e) {
        ...
    }
}
```

Use prepared statements

```
PreparedStatement lookupAllIds = conn.prepareStatement(
    "select id from credentials order by id");

PreparedStatement lookupCredential = conn.prepareStatement(
    "select * from credentials where id=?");

PreparedStatement deleteCredential = conn.prepareStatement(
    "delete from credentials where id=?");
```

- Faster
- More secure: avoid SQL injection attacks

Query accounts



- Query: specify one or more fields incl. % wildcard
- Query button is jButton7

Example: simple password keeper - query [1]

```
// Query button
private void jButton7ActionPerformed(java.awt.event.ActionEvent evt) {
    ...
    displayCredentialsSubset(new String[]{
        jTextField2.getText(),
        jTextField4.getText()},
        jPasswordField2.getPassword());
    clearCredDetails();
}
```

```
PreparedStatement searchCredential = conn.prepareStatement(
        "select * from credentials " +
        "    where id like ? and principal like ? and pw like ?");
```

Example: simple password keeper - query [2]

```
// Query and update selection list model
private void displayCredentialsSubset(String[] cols, char[] pw) {
    ...
    try {
        for (int i = 0; i < 2; i++) {
            searchCredential.setString(
                i + 1, "".equals(cols[i]) ? "%" : cols[i]);
        }
        searchCredential.setString(
                3, pw.length == 0 ? "%" : String.valueOf(pw));

        ResultSet rs = searchCredential.executeQuery();

        dlm.clear(); // underlying DataListModel of JList1 (selection pane)

        while (rs.next()) {
            dlm.addElement(rs.getString(1));
        }
        ...
    } catch (SQLException e) { … }
}
```

Example: import data using ij tool

- From CSV file “creds.dat”
- Semicolon field delimiter
- Double quote text delimiter

```
$ cat creds.dat
"Hotmail";"dag@hotmail.com";"secret"
"Saxophone club";"dag";"blow"
```

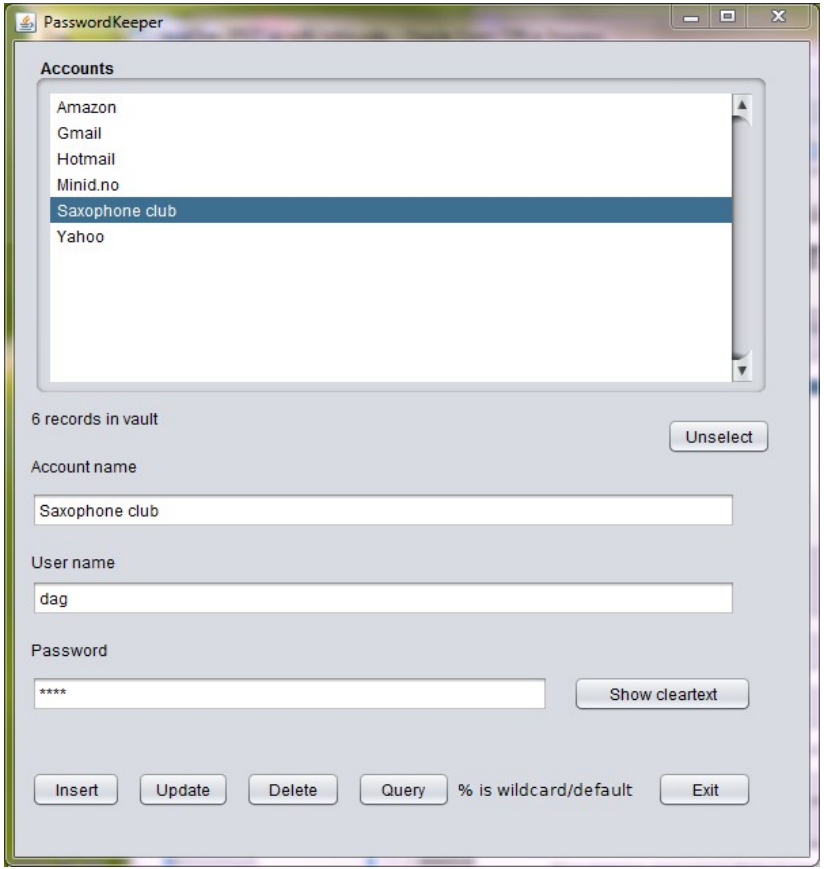
Example: import data using ij tool [2]

- From CSV file `creds.dat`

```
$ java -jar ${JDK_ROOT}/db/lib/derbyrun.jar ij
ij version 10.8
ij> connect 'jdbc:derby:pw-vault;dataEncryption=true;\
                    bootPassword=abracadabra';
ij> call syscs_util.syscs_import_table(null, 'CREDENTIALS',\
                   'creds.dat', ';', '"', null, 0);
```

0=append; 1=replace

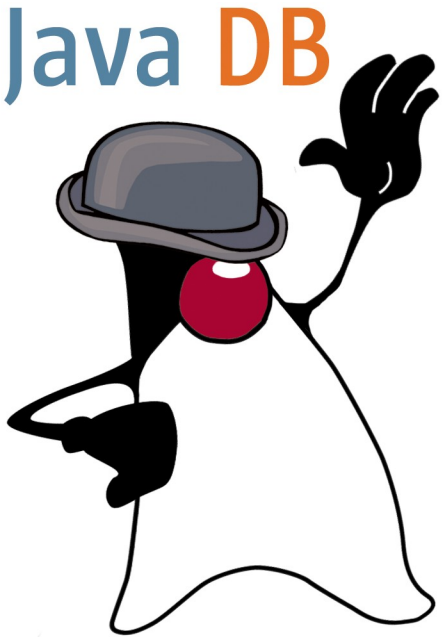
Example: import data using ij tool [3]



Example: schema evolution using ij tool

- Add last updated time stamp to PasswordKeeper

```
ij> connect 'jdbc:derby:pw-vault;dataEncryption=true;bootPassword=abracadabra';
ij> alter table credentials add column modified timestamp;
0 rows inserted/updated/deleted
ij> describe credentials; // subset of DatabaseMetaData#getColumns
COLUMN_NAME         |TYPE_NAME|DEC&|NUM&|COLUM&|COLUMN_DEF|CHAR_OCTE&|IS_NULL&
------------------------------------------------------------------------------
ID                  |VARCHAR  |NULL|NULL|256   |NULL      |512       |YES
PRINCIPAL           |VARCHAR  |NULL|NULL|256   |NULL      |512       |YES
PW                  |VARCHAR  |NULL|NULL|256   |NULL      |512       |YES
MODIFIED            |TIMESTAMP|9   |10  |29    |NULL      |NULL      |YES

4 rows selected
```



Program Agenda

- Availability
- Ease of use: development
- Ease of use: deployment
- Features
- Robustness
- Performance

Deployment friendly

- Permissive license: Apache version 2.0 is free software, but not copy-left
- Bundle Java DB jar(s) with your application
- Compact size: derby.jar ~2.6Mb

Ease of use: deployment

- Ship jars, use MANIFEST.MF in jar

```
$ java -jar PasswordKeeper.jar
```

MANIFEST.MF:

```
Manifest-Version: 1.0
Ant-Version: Apache Ant 1.8.3
Created-By: 1.7.0-b147 (Oracle Corporation)
Class-Path: lib/derby.jar
X-COMMENT: Main-Class will be added automatically by build
Main-Class: passwordkeeper.PasswordKeeper
```

Ease of use: deployment [2]

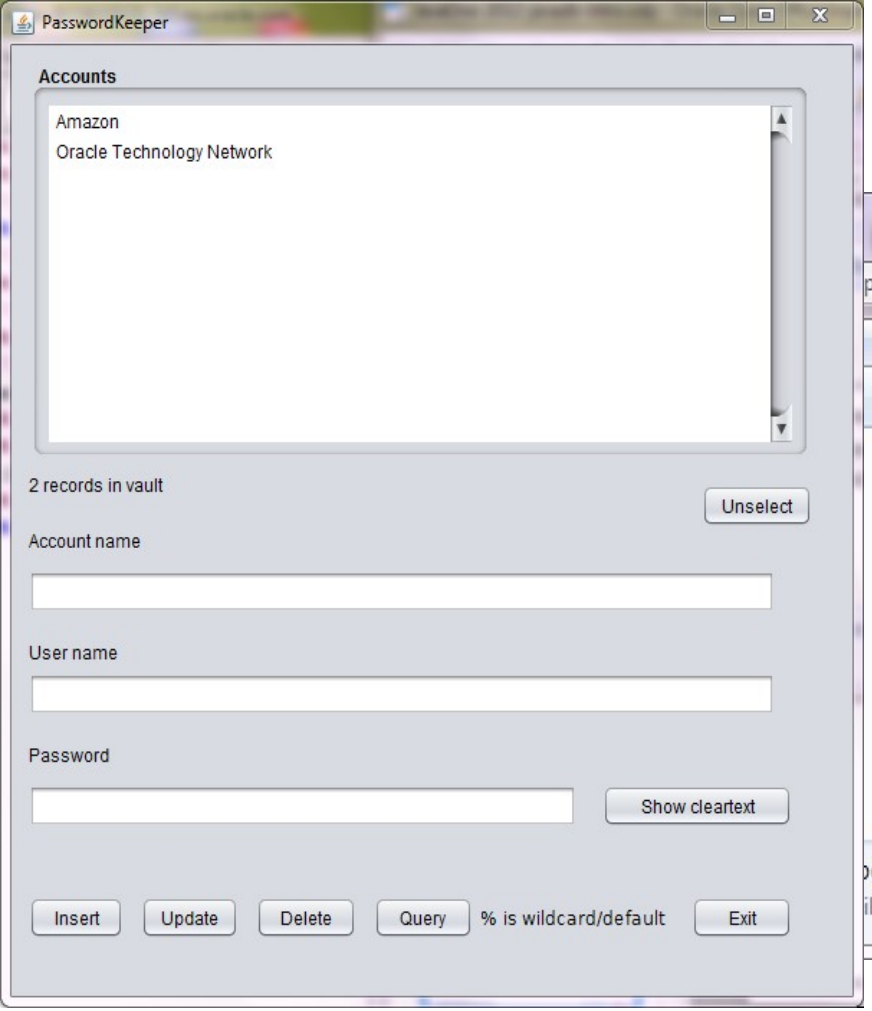
- ZIP that up in PasswordKeeper.zip:

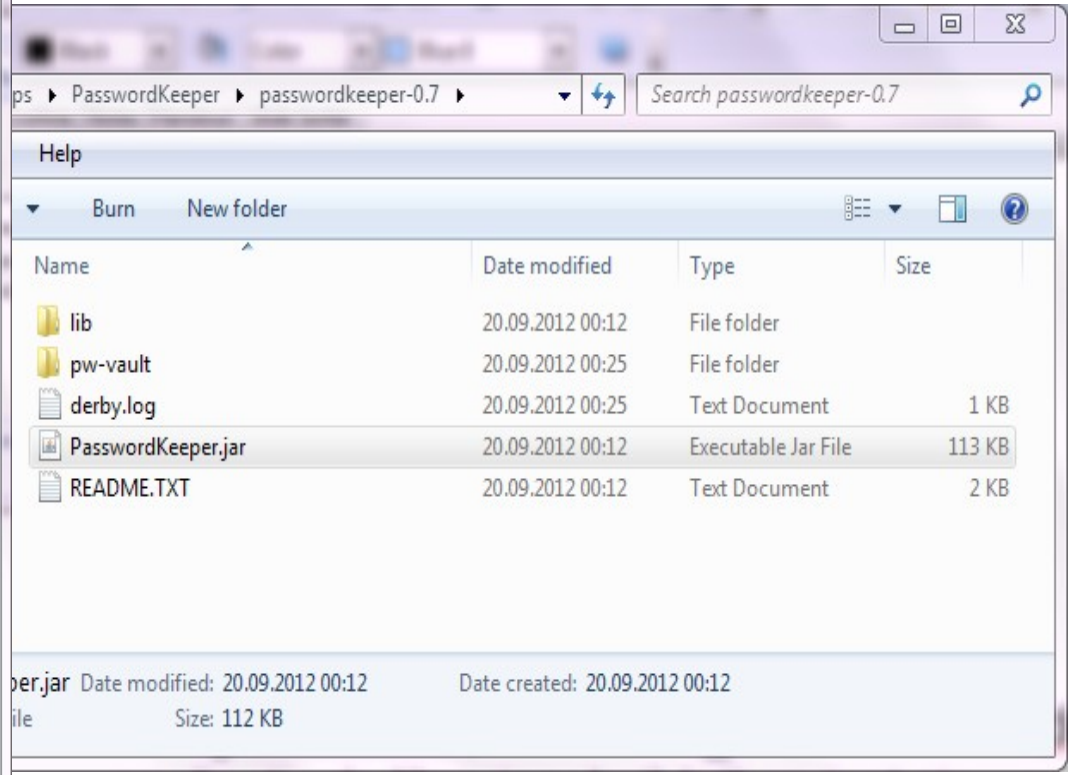
```
lib/derby.jar
PasswordKeeper.jar
```

- Using NetBeans, add <JDK_ROOT>/db/lib/derby.jar to application's libraries; build; then just zip up contents of application's *dist* directory, e.g.

```
$ mv dist passwordkeeper-${VERSION}

$ zip -r passwordkeeper-${VERSION}.zip passwordkeeper-${VERSION}
```

Unzip and go!





JavaOne ORACLE

Dev. & Deployment friendly - Maven

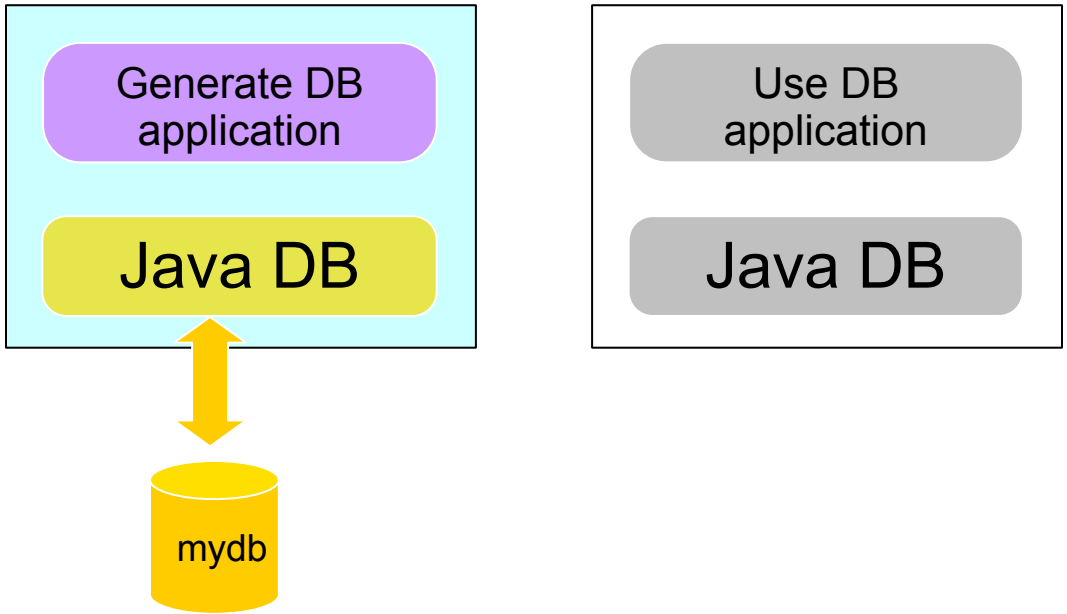
- At http://search.maven.org/, search Derby
- Add dependency to your project's pom.xml and go

```
...
<dependencies>
  ...
  <dependency>
    <groupId>org.apache.derby</groupId>
    <artifactId>derby</artifactId>
    <version>10.9.1.0</version>
    <optional>false</optional>
  </dependency>
</dependencies>
```

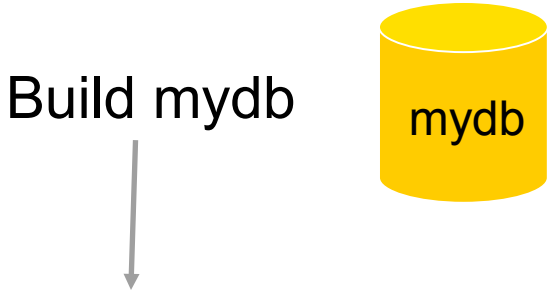
Ease of use: deployment of read-only db

- Deliver shrink-wrapped database: build

```
$ java -jar MyApp.jar build
```



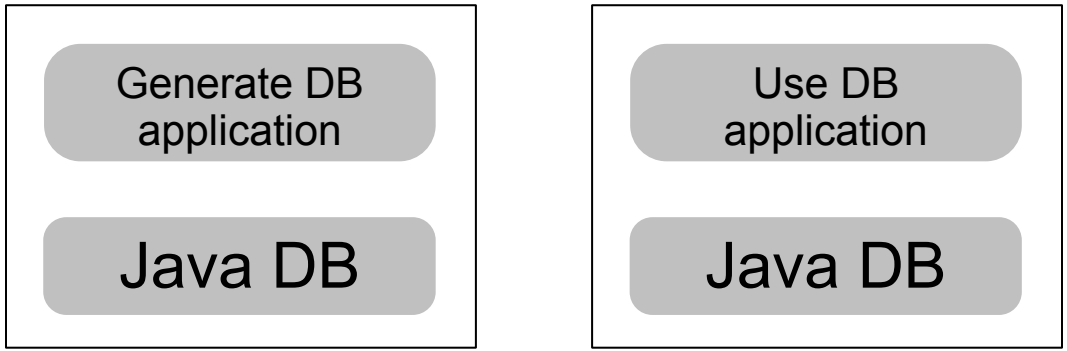
```
package myapp;

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

public class MyApp {
      ...
    if (args.length == 1 && "build".equals(args[0])) {
        Connection c = DriverManager.getConnection("jdbc:derby:mydb;create=true");
        Statement s = c.createStatement();
        s.executeUpdate("create table t (i int)");
        ...
        c.close();
        try {
            DriverManager.getConnection(“jdbc:derby:mydb;shutdown=true”);
        } catch (SQLException e) { … }
    } else {

    }
    ...
```

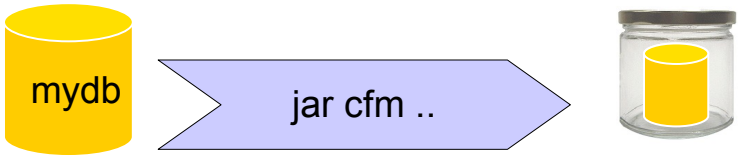


Ease of use: deployment of read-only db [2]

- Deliver shrink-wrapped database: wrap

```
$ jar cfm MyApp.jar MANIFEST.MF lib mydb myapp
```

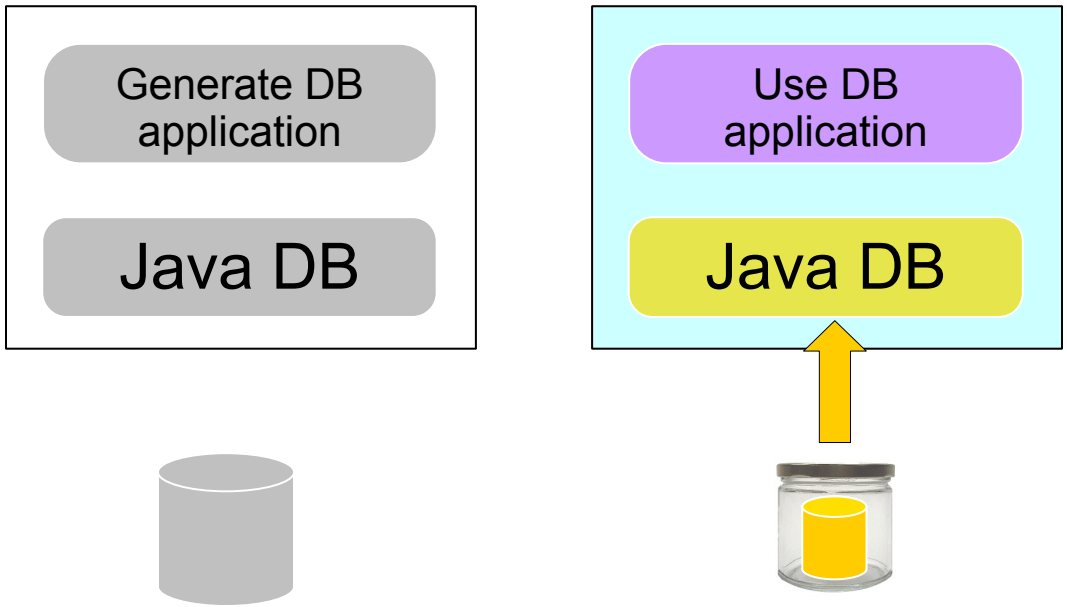




Ease of use: deployment of read-only db [3]

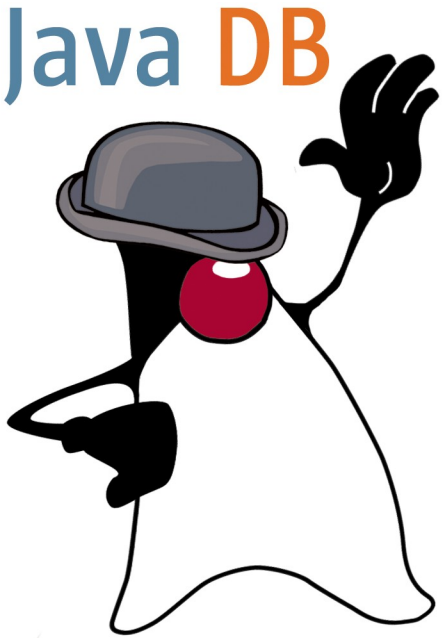
- Deliver shrink-wrapped database: then use!

```
$ java -jar MyApp.jar
```



```
package myapp;

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

public class MyApp {
    ...
    if (args.length == 1 && "build".equals(args[0])) {
        ...
    } else {
        Connection c = DriverManager.getConnection(
                                          "jdbc:derby:jar:(MyApp.jar)mydb");
        Statement s = c.createStatement();
        ResultSet rs = s.executeQuery("select * from t");
        ...
    }
    ...
```

Read-only DB in a jar



Program Agenda

- Availability
- Ease of use: development
- Ease of use: deployment
- Features
- Robustness
- Performance

Standards based

- SQL
  - Based on SQL99 and SQL2003: core/mandatory set

    http://wiki.apache.org/db-derby/SQLvsDerbyFeatures
- Runs on Java 5 - Java 7, JDBC 4.1
  - JSR 169: JDBC for Java ME CDC
- DRDA V5 (database network protocol, by OpenGroup)
- OSGi bundle
- X/Open: XA (distributed transactions)

Standards based [2]

- Ease upgrade path to enterprise level database (e.g. Oracle, DB2), but no silver bullet; SQL differences
  - Data type differences (e.g. range, precision, collation) => helps to know eventual target DB
  - SQL standard ensures minimum hassle, ORM can help: JPA (TopLink, Hibernate,...), JDO
  - Schema extraction: $ java -jar derbyrun.jar dblook ...
  - Export data to CSV: call syscs_util.syscs_export_table

Powerful relational engine

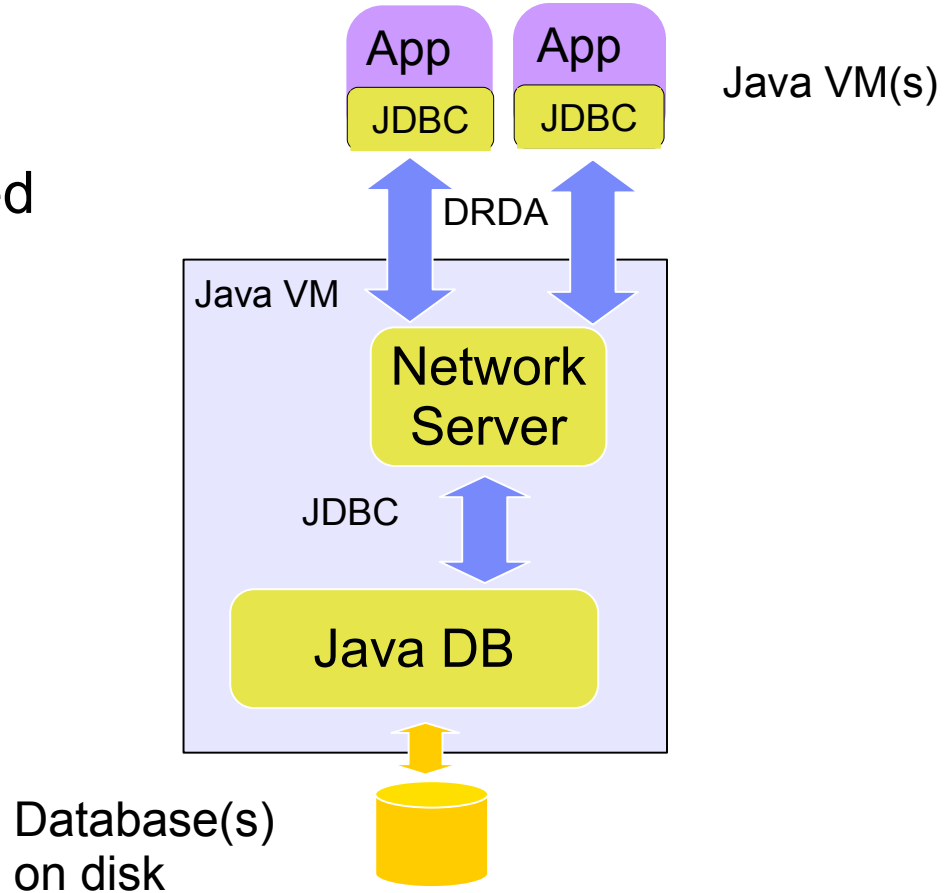
- Multi-user, transactions, isolation levels, deadlock detection, crash recovery, lock escalation
- SQL: schemas, tables, indexes, views, triggers, procedures, functions, UDTs, table functions, collations, sequences
- SQL: foreign keys and check constraints
- SQL: joins, cost based optimizer, automatic statistics
- Data caching, statement caching, write ahead logging (Aries), group commit
- Online backup/restore, space compression, asynch. replication, XA client

Security

- Native, LDAP and pluggable authentication, pw expiration
- Client/server data flow, network authentication: SSL/TLS
- SQL grant/revoke fine grained privileges support, incl. SQL roles, routines: definer's or invoker’s rights
- Java Security Manager (server default)
- Conservative file visibility by default in server
- On-disk database encryption

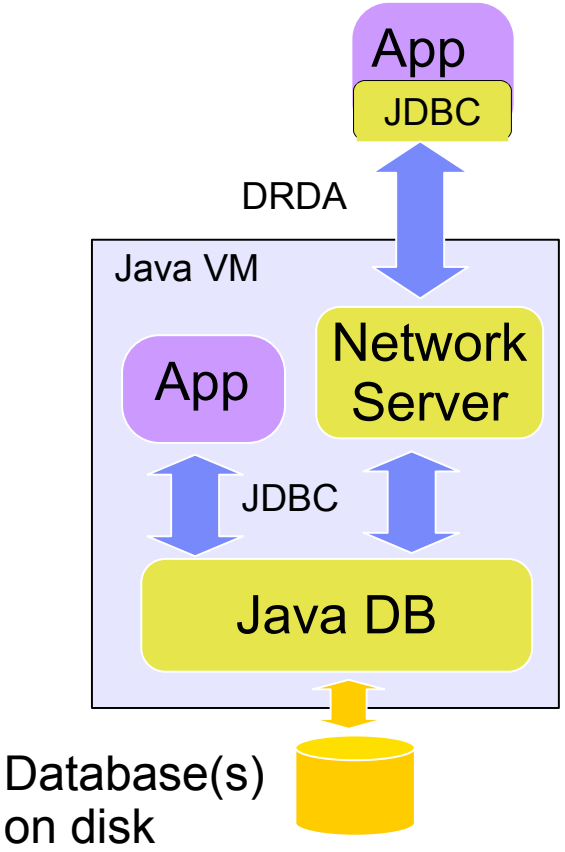
Client server mode

- Network Server uses embedded driver against Java DB
-
```
$ java -jar derbyrun.jar \
  server start &


$ java -jar derbyrun.jar \
  server shutdown
```

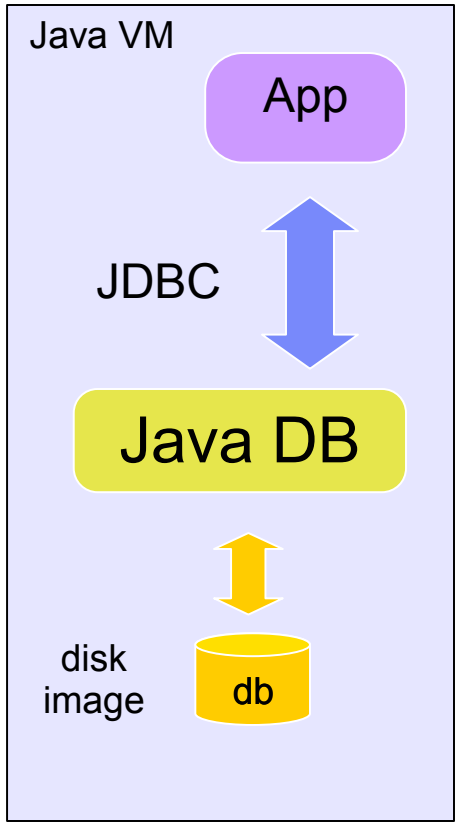


Embedded + network server

- Provides access to database from outside the application's JVM: monitoring, reporting
- derby.drda.startNetworkServer=true or via API
- Embeddable in app server frameworks, e.g. WebLogic, GlassFish app servers (Java DB is default db)



In-memory operation



- Speed over durability
- Use case: transient data
- Simple: just add “:memory” in URL
- Mix: update on-disk version at end of session:

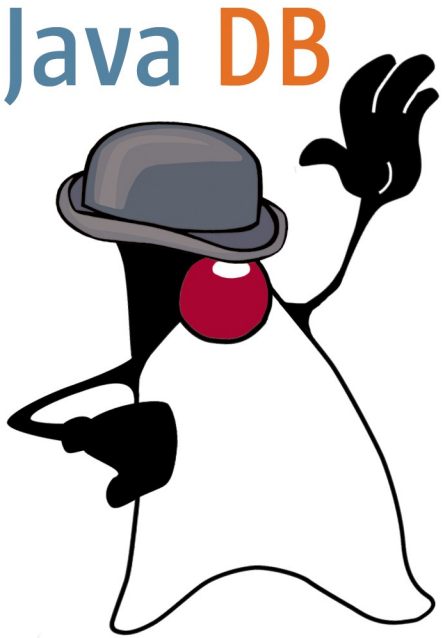
```
call syscs_util.syscs_backup_database(<dir>);

connect('jdbc:derby:memory:db;restoreFrom=<dir>/db');
```

Example: simple password keeper - connect [v.2]

```
boolean loadDatabase(String dbFileName, char[]pw) {
  ...
  try {
    if (db.isDirectory()) {

      conn = DriverManager.getConnection("jdbc:derby:memory:" +
             dbFileName + ";restoreFrom=" + dbFileName +
             ";bootPassword=" + String.valueOf(pw));

    } else {

      conn = DriverManager.getConnection("jdbc:derby:memory:" +
             dbFileName +
             ";create=true;dataEncryption=true;bootPassword=" +
             String.valueOf(pw));
      ...
```

Example: simple password keeper - exit

```
void shutdownDbAndExit() {
  ...
  try {
    // Back up database before we close

    PreparedStatement s = conn.prepareCall(
        "call syscs_util.syscs_backup_database(?)");
    File db = new File(dbFileName);
    String parentDir = new File(db.getCanonicalPath()).getParent();
    s.setString(1, parentDir);
    s.execute();
    ...
```



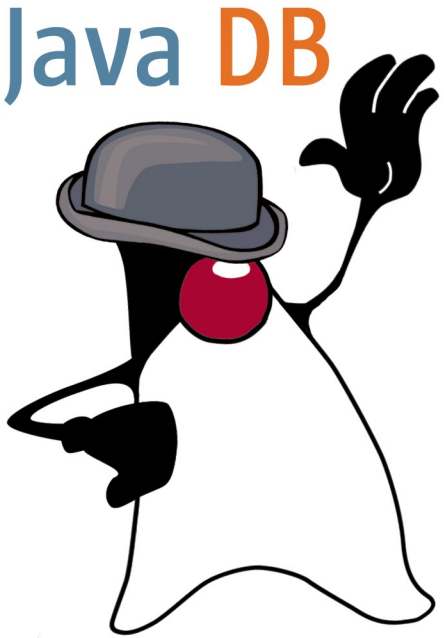
Program Agenda

- Availability
- Ease of use: development
- Ease of use: deployment
- Features
- Robustness
- Performance

Robustness

- Mature database (15+ years), maintained by devs from both IBM and Oracle plus volunteers: responsive community
- Proven track record, samples at

  http://wiki.apache.org/db-derby/UsesOfDerby
- Open development process in Apache Derby
- Frequent releases
- Good upward compatibility



Program Agenda

- Availability
- Ease of use: development
- Ease of use: deployment
- Features
- Robustness
- Performance

Performance

- Comparable with other Open Source databases
- Performs well: queries compiled to Java byte code: JIT optimized
- Sweet spot when data can't fit in memory: has a good cache replacement algorithm and group commits
- Scales well
- See also: TS-45170 Java 2007 “Java DB Performance”

Performance hints

- Use and reuse prepared statements:

  Statement cache hit chance:

  ```
  “select * from t where id = ?”
  ```

  ```
  “select * from t where id=” + val
  ```

- Cache reused across connections
- Avoid auto-commit on long transactions, e.g. large inserts
- Log files on a separate disk (logDevice connection attribute)
- Consider in-memory operation

Performance hints [2]

- Tune page cache size (default 4MB):

  derby.storage.pageCacheSize=...

- Use indexes to avoid table scans: validate with

  derby.language.logQueryPlan=true

- Make sure statistics are up-to-date: automatically or manually

  => optimizer does a good job.

- If not, use override:

```
-- DERBY-PROPERTIES index = t_idx
```

Summary

- Easy-to-use, small foot-print, performant, pure Java, embeddable DB with capable SQL via JDBC
- Shrink-wrapped apps: see no database
- Easy to develop on Java DB, deploy on enterprise DB

It's there, in your copy of the Oracle JDK, check it out!

Q & A

MAKE THE FUTURE JAVA

ORACLE


JavaOne™