Android & PostgreSQL

Mark Wong
markwkm@postgresql.org

PGCon 2011, University of Ottawa

20 May 2011
Slides available at http://www.sideshare.net/markwkm
Overview

Two topics, mostly the latter:

- Development environment specific for PostgreSQL JDBC
- Code samples for using the PostgreSQL JDBC driver

Any requests?
Audience Survey

Are you familiar with:

- Developed an Android application?
- Java?
- JDBC?
- Eclipse?
- SQL?
- PostgreSQL?
Development Environment

A few details on the following slides about:

- JDK5 or JDK6
- Android SDK
- PostgreSQL JDBC Driver
- Eclipse is optional, current Android SDK r11 requires 3.5 or 3.6

Full system requirement details for Android development:
http://developer.android.com/sdk/requirements.html
PostgreSQL JDBC Driver

http://jdbc.postgresql.org/

- Distributed under BSD License.  
  (http://jdbc.postgresql.org/license.html)
- Supports PostgreSQL 7.2 and laster version
- Thread-safe  
  (http://jdbc.postgresql.org/documentation/head/thread.html)
- JDBC3 vs. JDBC4 (http://jdbc.postgresql.org/download.html)
  - If you are using the 1.6 JVM, then you should use the JDBC4 version.
  - JDK 1.4, 1.5 - JDBC 3. This contains support for SSL and javax.sql, but does not require J2EE as it has been added to the J2SE release.
  - JDK 1.6 - JDBC4. Support for JDBC4 methods is limited. The driver builds, but the several of the new methods are stubbed out.
- Todo list http://jdbc.postgresql.org/todo.html
Using the PostgreSQL JDBC Driver

To add the PostgreSQL JDBC driver for building and packaging edit the hidden `.classpath` file at the top level of the project directory with the full path to the JAR file:

```xml
<classpath>
  ...
  <classpathentry kind="lib"
      path="/workspace/proj/postgresql-9.0-801.jdbc4.jar" />
  ...
</classpath>
```

Alternate instructions when using Eclipse:

http://developer.android.com/resources/faq/commontasks.html#addexternallibrary
Dave Cramer built a special version of the PostgreSQL JDBC driver that works on Android 1.5 and later and posted it in a discussion in the PgAndroid Google Group (this pdf should be built so you can click on the tiny text to open the link):

http://groups.google.com/group/pgandroid/browse_thread/thread/d8b400f039f66d5f/f77b2e2a99370a36?lnk=raot#f77b2e2a99370a36

At this time, build 801 of the JDBC driver works with Android 2.1 and later.
PostgreSQL JDBC Code Examples

- Some simple examples for connecting to a PostgreSQL database and querying some data, and PostgreSQL specific features
- Most examples are variations of those given in the PostgreSQL and PostgreSQL JDBC documentation

Warning: code formatted to fit better on these slides...
Open a Database Connection

Load the PostgreSQL JDBC driver and open a database connection using SSL:

```java
Class.forName("org.postgresql.Driver");
String url;
url = "jdbc:postgresql://pghost:5432/pgdatabase" +
     "?sslfactory=org.postgresql.ssl.NonValidatingFactory" +
     "&ssl=true";
Connection conn = DriverManager.getConnection(url,
                                          "pguser",
                                          "pgpass");

// Don't forget to close the connection when you're done.
// conn.close();
```
Execute a Query

Select the name of all relations from `pg_class` and iterate through every row returned:

```java
String sql;
sql = "SELECT relname FROM pg_class WHERE relkind = 'r';";

Statement st = conn.createStatement();
ResultSet rs = st.executeQuery(sql);
while (rs.next()) {
    // Columns are can be referenced by name.
    String relname = rs.getString("relname");
}
rs.close();
st.close();
```
Execute a Query Using a Cursor

Select the name of all table names from `pg_tables` and iterate through them fetching 10 at a time:

```java
conn.setAutoCommit(false);
String sql = "SELECT tablename FROM pg_tables;"

Statement st = conn.createStatement();
st.setFetchSize(10);
ResultSet rs = st.executeQuery(sql);
while (rs.next()) {
    // Columns are can be referenced by name.
    String relname = rs.getString("relname");
}
rs.close();
st.close();
```
Execute a Query with a Bind Value

Building on the previous slide, select the number of all relations from `pg_class`:

```java
String sql = "SELECT COUNT(*) FROM pg_class WHERE relkind = ?;"

PreparedStatement ps = conn.createStatement();
// Bind variables are enumerated starting with 1;
ps.setString(1, "r");
ResultSet rs = ps.executeQuery(sql);

rs.next();
// Columns are enumerated starting with 1.
long count = rs.getLong(1);

rs.close();
ps.close();
```
Some addition methods for retrieving column data

getBoolean()
getDate()
getDouble()
getFloat()
getInt()

getLong()
getShort()
getString()
getTime()
getTimestamp()
Execute CREATE, ALTER, or DROP SQL statements with the `execute()` method:

```java
Statement st = conn.createStatement();
st.execute("CREATE TABLE films (title VARCHAR(40));");
st.close();
```
Example for INSERT, UPDATE and DELETE

Executing INSERT, UPDATE, or DELETE SQL statements use the `executeUpdate()` as opposed to the `executeQuery()` method shown previously with SELECT statements. Also, `executeUpdate()` returns the number of rows affected as opposed to a ResultSet, otherwise usage is mostly the same:

```java
PreparedStatement st =
conn.prepareStatement(
   "INSERT INTO films (title) VALUES (?)" );
st.setString(1, "On Stranger Tides");
int rows = st.executeUpdate();
st.close();
```
Example for Calling Stored Functions

Call a built-in stored function that returns a value. In this example, call `upper()` to change a string to all upper case characters:

```java
CallableStatement upperProc =
    conn.prepareCall("{ ?= call upper(?)}");
upperProc.registerOutParameter(1, Types.VARCHAR);
upperProc.setString(2, "lowercase to uppercase");
upperProc.execute();
String upperCased = upperProc.getString(1);
upperProc.close();
```
LISTEN & NOTIFY & UNLISTEN

http://jdbc.postgresql.org/documentation/head/listennotify.html
Starting listening to a channel

Register this session as a listener to the notification channel *virtual*:

```
Statement stmt = conn.createStatement();
stmt.execute("LISTEN virtual");
stmt.close();
```
LISTEN: Dummy query

Need to issue a dummy query to contact the PostgreSQL database before any pending notifications are received:

```java
while (true) {
    try {
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery("SELECT 1;");
        rs.close();
        stmt.close();
    }
    // ...
LISTEN: Handle notifications

A key limitation of the JDBC driver is that it cannot receive asynchronous notifications and must poll the backend to check if any notifications were issued.

```
org.postgresql.PGNotification notifications[] = pgconn.getNotifications();
if (notifications != null) {
    for (int i = 0; i < notifications.length; i++) {
        // Handle here: notifications[i].getName();
    }
}

// Wait before checking for more notifications.
Thread.sleep(500);
```
LISTEN: Java Exception Handling

...
Stop listening to a channel

Stop listening to the channel *virtual*:

```java
Statement stmt = conn.createStatement();
stmt.execute("UNLISTEN virtual;" );
stmt.close();
```
NOTIFY: Send a notification

Send a notification to the channel *virtual*:

```java
Statement stmt = conn.createStatement();
stmt.execute("NOTIFY virtual; ");
stmt.close();
```

Note: Starting with PostgreSQL 9.0, there is the stored function `pg_notify(text, text)` that can be used.
Server Prepared Statements

JDBC allows a threshold to be set before the server prepared statement is used. Recommended to use with PostgreSQL 7.4 and later:
http://jdbc.postgresql.org/documentation/81/server-prepare.html

```java
PreparedStatement pstmt = conn.prepareStatement("SELECT ?; ");
// Cast the PreparedStatement to the PostgreSQL specific // PGStatement.
org.postgresql.PGStatement pgstmt =
    (org.postgresql.PGStatement) pstmt;
// Use prepared statement on the 2nd execution onward.
pstmt.setPrepareThreshold(2);
pstmt.setInt(1, 42);
ResultSet rs = pstmt.executeQuery();
pstmt.close();
```
Geometric data types

PostgreSQL data types that include single points, lines, and polygons:
http://jdbc.postgresql.org/documentation/81/geometric.html

Circle example:

```java
stmt.execute("CREATE TABLE geo (mycircle);" );
```
INSERT a circle

PGpoint center = new PGpoint(1, 2.5);
double radius = 4;
PGcircle circle = new PGcircle(center, radius);
PreparedStatement ps = conn.prepareStatement("INSERT INTO geomtest(mycirc) VALUES(?)");
ps.setObject(1, circle);
ps.executeUpdate();
SELECT a circle

```java
ResultSet rs = stmt.executeQuery(
    "SELECT mycirc FROM geo;" );
rs.next();
PGcircle circle = (PGcircle) rs.getObject(1);
PGpoint center = circle.center;
double radius = circle.radius;
```
JDBC Escapes

A special escape syntax that is JDBC specific and database agnostic:
http://jdbc.postgresql.org/documentation/head/escapes.html

Dates and timestamps:

```java
st.executeQuery("SELECT {fn week({d '2005-01-24'})};");
```

Joins:

```java
st.executeQuery("SELECT * FROM {oja LEFT OUTER JOIN b ON (a.i = b.i)};");
```

Scalar functions:

```java
st.executeQuery("SELECT {fn abs(-1)};");
```
Handling binary data

Long examples for using the bytea data type and Large Objects:
http://jdbc.postgresql.org/documentation/head/binary-data.html
Connection pools

Long examples for using JDBC connection pooling:
http://jdbc.postgresql.org/documentation/head/datasource.html
Example application

**PGTop for Android** is a complete application implementing some of the examples shown here:

- Source code: https://github.com/markwkm/pgtop
- Android Market:
  https://market.android.com/details?id=org.postgresql.top
For reference, since not covered in this presentation:

- Hello, World (includes example without using Eclipse)

- More examples without using Eclipse http://
  //developer.android.com/guide/developing/other-ide.html

- Even more sample applications
  http://developer.android.com/resources/samples/
Further JDBC Reading

JDBC API Documentation and JDBC Specification
http://jdbc.postgresql.org/documentation/head/reading.html
markwkm (PGCon2011) Android & PostgreSQL

20 May 2011 35 / 37

Thank you!
Acknowledgements

Hayley Jane Wakenshaw
License

This work is licensed under a Creative Commons Attribution 3.0 Unported License. To view a copy of this license, (a) visit http://creativecommons.org/licenses/by/3.0/us/; or, (b) send a letter to Creative Commons, 171 2nd Street, Suite 300, San Francisco, California, 94105, USA.