BAB 5

Koneksi Database : SQL dan JDBC

5.1 Pengenalan

Perintah – perintah dalam lingkungan databases adalah :

5.1.1 SELECT

Pernyataan SELECT digunakan untuk query database tentang informasi database yang mana yang ditampilkan sebagai data. Format **dasar** dari pernyataan SELECT adalah :

```
SELECT kolom FROM namatabel where kondisi
```

Pernyataan SQL SELECT dimulai dengan kata kunci SELECT, diikuti oleh tanda koma dari kolom-kolom yang akan ditampilkan, kemudian clause FORM yang menentukan tabel yang berisi data yang akan ditampilkan. Secara bebas, clause WHERE dapat ditambahkan pada pernyataan SELECT, menjelaskan sebuah set dari kondisi yang harus ditemukan oleh data yang akan dikembalikan oleh database. Clause WHERE tidak dipisahkan oleh koma; melainkan, dia tehubung oleh pernyataan AND atau OR yang fungsinya sama dengan logik mereka.

Menetapkan * sebagai nama kolom memberitahu database server untuk meretrieve semua kolom yang tersedia didalam tabel.

Contoh, jika kita ingin meretrieve semua data yang terdapat pada table yang bernama users :

```
SELECT * from users;
```

Jika kita hanya ingin melihat untuk users yang bernama belakang Smith, pernyataan SQL-nya berupa:

```
SELECT * from users where nama ='Smith';
```

SQL tidak bersifat case-sensitive pada kata kuncinya, SQL case-sensitive pada nilainya. Pernyataan berikut ini akan menghasilkan data yang berbeda jika dibandingkan dengan pernyataan diatas:

```
SELECT * from users where nama ='sMith';
```

Operator lain yang dapat digunakan pada conditional statements:

- < - kurang dari
- <= - kurang dari atau sama dengan
- > - lebih besar dari
- >= - lebih besar dari atau sama dengan
- like – sama dengan

5.1.2 INSERT

Pernyataan INSERT digunakan untuk memasukkan baris data yang baru dari informasi tabel database yang aktif.

Struktur dasar dari pernyataan INSERT adalah :

```
INSERT INTO nama-tabel VALUES(nilai1, nilai2, ...)
```

Dimana nama-tabel adalah nama dari tabel yang akan berisi baris data yang baru. Parameter yang diberikan didalam kata kunci VALUES adalah daftar data dari nilai yang akan ditambahkan kedalam tabel. Jika seperti ini dimana hanya tabel yang ditetapkan, SQL akan memasukkan nilai yang diberikan pada pernyataan dengan field didalam database berdasarkan nilai yang dipesan dan field yang ditentukan pada tabel database.

Jika, sebagai contoh, kita mempunyai tabel dengan nama users, dengan field userid, nama, alamat(pada pemesanan), pada baris berikut ini akan menambahkan data yang baru ke tabel:

```
INSERT INTO users VALUES(199700651, 'Jeni Master', 'UP Ayala Technopark');
```

Penting untuk diingat bahwa semua panggilan untuk INSERT harus diikuti aturan integritas pada tabel data. Oleh karena itu, jika sebuah field pada database ditetapkan non-null, berbagai usaha untuk memasukkan ni;ai-null kedalam field tersebut akan menyebabkan error pada database.

5.1.3 UPDATE

Pernyataan UPDATE akan mengupdate baris yang dipilih pada tabel, sebagai lawan dari pernyataan INSERT yang menambahkan baris data baru. Format dasar pernyataan UPDATE adalah:

```
UPDATE nama-tabel set nilai-kolom WHERE kondisi
```

Dimana nama-tabel adalah nama dari tabel yang berisi baris yang akan diupdate, dan nilai-kolom adalah daftar data dari nama kolom dan nilainya. Secara bebas, daftar data dari kondisi dapat ditambahkan secara spesifik dengan baris yang akan dimodofikasi pada tabel. Jika tidak diber kondisi, maka update data akan dilakukan pada tiap-tiap baris didalam tabel yang ditentukan.

Berbagai update harus disesuaikan dengan aturan integritas pada database. Sebagai contoh, menyetting nilai null pada kolom yang sudah ditetapkan dengan nilai NOT NULL akan menyebabkan pernyataan tidak akan dijalankan dan terdapat pesan error pada relasi database.

5.1.4 DELETE

Pernyataan DELETE menghapus baris data pada tabel yang dipilih.

Struktur dasar dari pernyataan DELETE adalah :

```
DELETE FROM nama-tabel WHERE kondisi
```

Dimana nama-tabel adalah nama dari tabel yang berisi baris data yang akan dihapus. Daftar data dari kondisi secara bebas dapat dispesifikasikan sebaik mungkin. Jika tidak diberi kondisi, maka pernyataan akan mneghapus semua baris data pada tabel yang telah ditentukan.

5.2 JDBC

Java menyediakan standard API untuk mengakses database yang disebut Java Database Connectivity (JDBC) API. Dengan menggunakan ini, para pengembang memungkinkan dapat mengakses database tanpa memperdulikan vendornya; para vendor menyediakan implementasi untuk abstract interfaces yang dijelaskan didalam API, penyediaan tersebut sama dengan set dari kemampuan koneksi untuk para pengembang.

Berikut ini merupakan class kunci dari JDBC API, semuanya akan dijelaskan secara detail kemudian :

- java.sql.Connection – membuat sebuah koneksi dengan database. Secara abstrak memberikan detail dari bagaimana cara untuk berkomunikasi dengan database server.
- java.sql.DriverManager – mengatur JDBC driver yang digunakan oleh aplikasi. Pada hubungannya dengan proper driver URL dan proper authentication, dapat menyediakan aplikasi dengan valid instances dari object koneksi.
- javax.sql.DataSource – memisahakan detail (URL, authentication details) dari bagaimana untuk memperoleh sebuah koneksi ke database. Merupakan method terbaru dan yang lebih disukai dari obtaining Connection objects.
- java.sql.Statement – menyediakan method untuk para pengembang dalam mengeksekusi pernyataan SQL.
- java.sql.ResultSet – menyediakan hasil dari sebuah pernyataan SQL. Objects ini sering dikembalikan dari method yang terletak pada Statement object.

5.2.1 java.sql.DriverManager

Dengan menggunakan class ini, pengembang dapat meretrieve sebuah Connection object yang kemudian dapat dia gunakan untuk melakukan aktifitas database. Berikut ini dua langkah yang dianjurkan:

- Pertama, JDBC driver harus diregistrasi dengan DriverManager. Hal ini dapat dikerjakan dengan menggunakan method Class.forName untuk menge-load driver's class definition kedalam memori.
- Kedua, menggunakan method getConnection pada DriverManagerm untuk menyediakan JDBC URL, sebaik username dan password supplying untuk akses database. URL harus mengikuti syntax yang dianjurkan oleh implementasi database tertentu.

Dibawah ini adalah contoh dari bagaimana mendapatkan koneksi dari database PostgreSQL. Sekali lagi, URL dan driver tepat untuk implementasi database yang digunakan. Untuk database yang lain, periksalah dokumen yang disediakan.

```
String jdbcURL = "jdbc:postgresql://localhost:5432/jeni-db";
String user = "jeni";
String password = "j3n1master";
Connection conn = null;

try {
   Class.forName("org.postgresql.Driver");
   conn = DriverManager.getConnection(url, user, password);
 ...
} catch (SQLException e) {
   // perform error handling here
}
```

Saat ini merupakan cara yang valid dari meretrieve sebuah Connection object, method ini menganjurkan para pengembang untuk tetap mengikuti method tersebut dari seperti detail sebagai driver class name, URL dianjurkan untuk akses ke database, sedangkan username dan password tepat untuk penggunaan database. Detail-detail ini paling banyak dipakai pada berbagai aplikasi. Dan juga, mengatur URL dan driver name pada kode membuatnya lebih sulit untuk aplikasi dalam menukar implementasi database, jika hal itu diperlukan.

5.2.2 javax.sql.DataSource

DataSource merupakan interface yang digambarkan pada JDBC API sejak versi 2 dari spesifikasinya. Sekarang saatnya direkomendasikan untuk para pengembang dalam mendapatkan Connection object. Retrieval dari Connection object terjadi secara langsung : cara sederhana memanggil method getConnection() dalam kejadian yang valid dari DataSource. Hal tersebut memperoleh sebuah kejadian dari DataSource yang sekarang dapat menyelesaikan sebuah masalah untuk beberapa pengembang(developer).

Sejak DataSource merupakan sebuah interface, sebuah instance tidak dapat dibuat oleh pengembang dengan menggunakan operator yang baru secara sederhana. Hal tersebut direkomendasikan jika kita memilih aplikasi server dengan menggunakan aturan pembuatan dari DataSource objects untuk kita.

5.2.3 Konfigurasi DataSource pada Sun Application Server 8.1

Masing-masing server memiliki prosedurnya sendiri untuk mengonfigurasi dan mengatur DataSources. Apa yang akan kita bahas akan menjadi prosedur untuk melakukan sesuatu dalam container yang telah gunakan sejauh ini untuk contoh kita yaitu : Sun Application Server 8.1.

3 langkah dalam menyetting datasource untuk AppServer 8.1 :

- Mendaftarkan file JAR yang berisi JDBC driver dengan container.
- Membuat connection pool ke database
- Mendaftarkan sebuah datasource yang digunakan untuk connection pool.

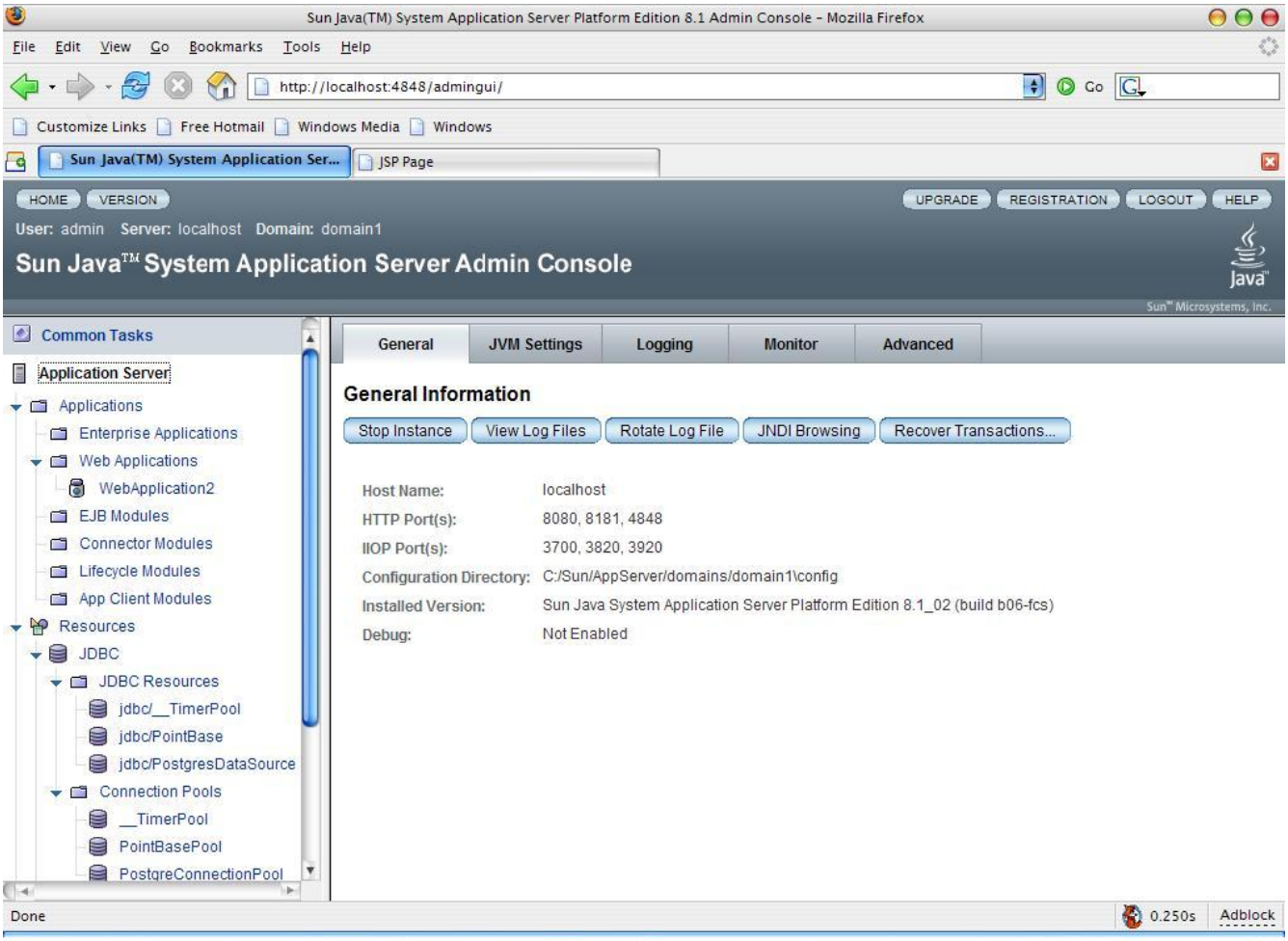
5.2.4 Mendaftarkan file JAR

Langkah pertama akan mengakses console admin untuk server. Sebagai default, console admin dapat diakses dengan menggunakan URL berikut ini pada address bar browser :

http://localhost:4848/

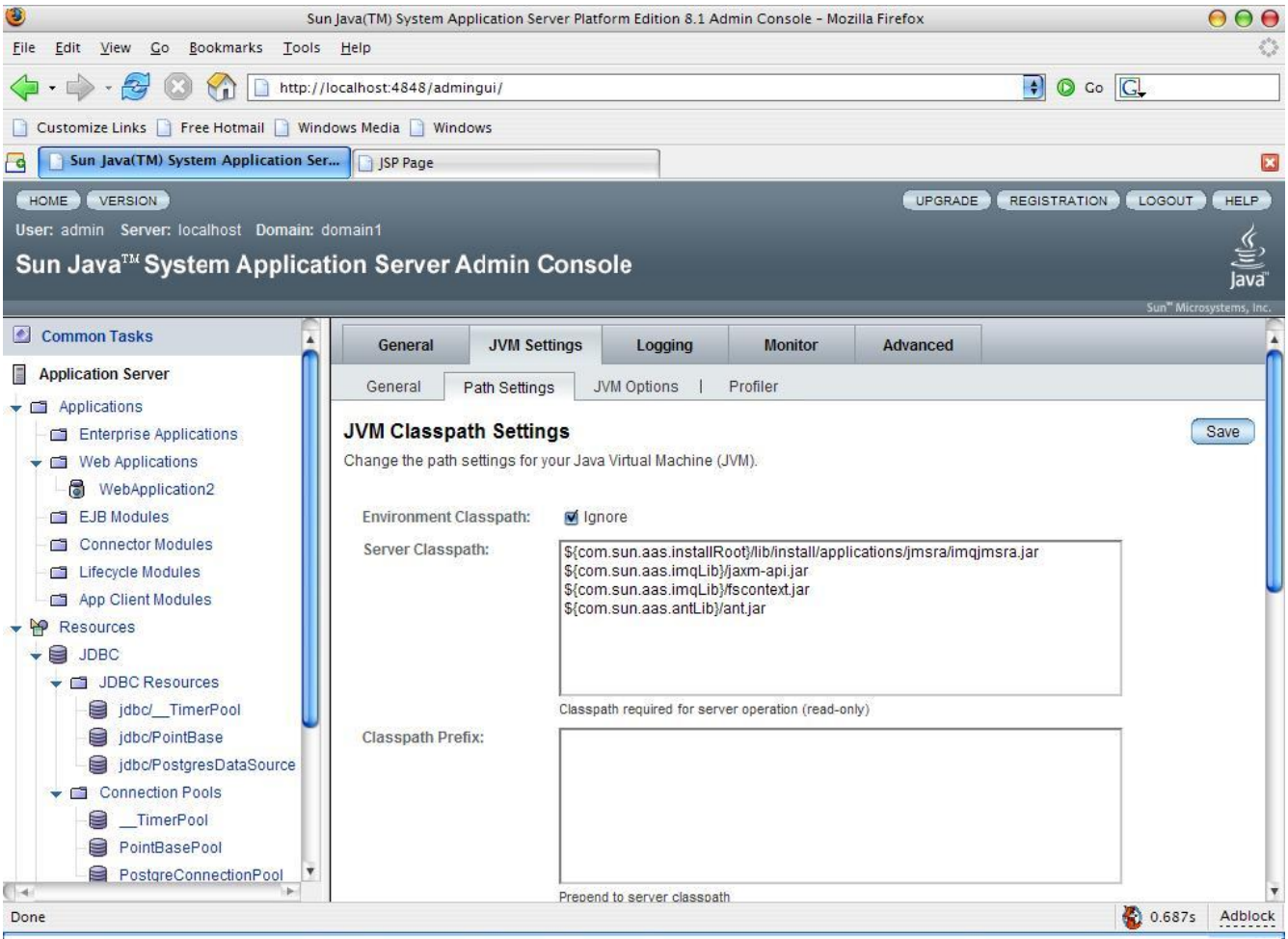
Pada kasus nomer port berbeda yang telah dikonfigurasi sebelumnya untuk console admin Anda selama waktu menginstal, dengan mudah gantilah 4848 dengan nomer port yang ada.

Setelah menyediakan security credentials dibutuhkan untuk mengakses console, akan tampil sebuah layar seperti gambar berikut ini:



Untuk memproses, klik pada **Application Server** pada pane sebelah kiri, lalu klik pada tab *JVM settings* tab pada pane sebelah kanan.

Kemudian pada layar berikutnya, pilih tab Path Settings pada pane sebelah kanan. akan tampil sebuah layar seperti gambar berikut ini.



Scroll kebawah sampai anda menemukan textarea yang berlabel *Classpath suffix*. Masukkan path leading kedalam file JAR yang berisi JDBC drivers.

5.2.5 Membuat sebuah connection pool

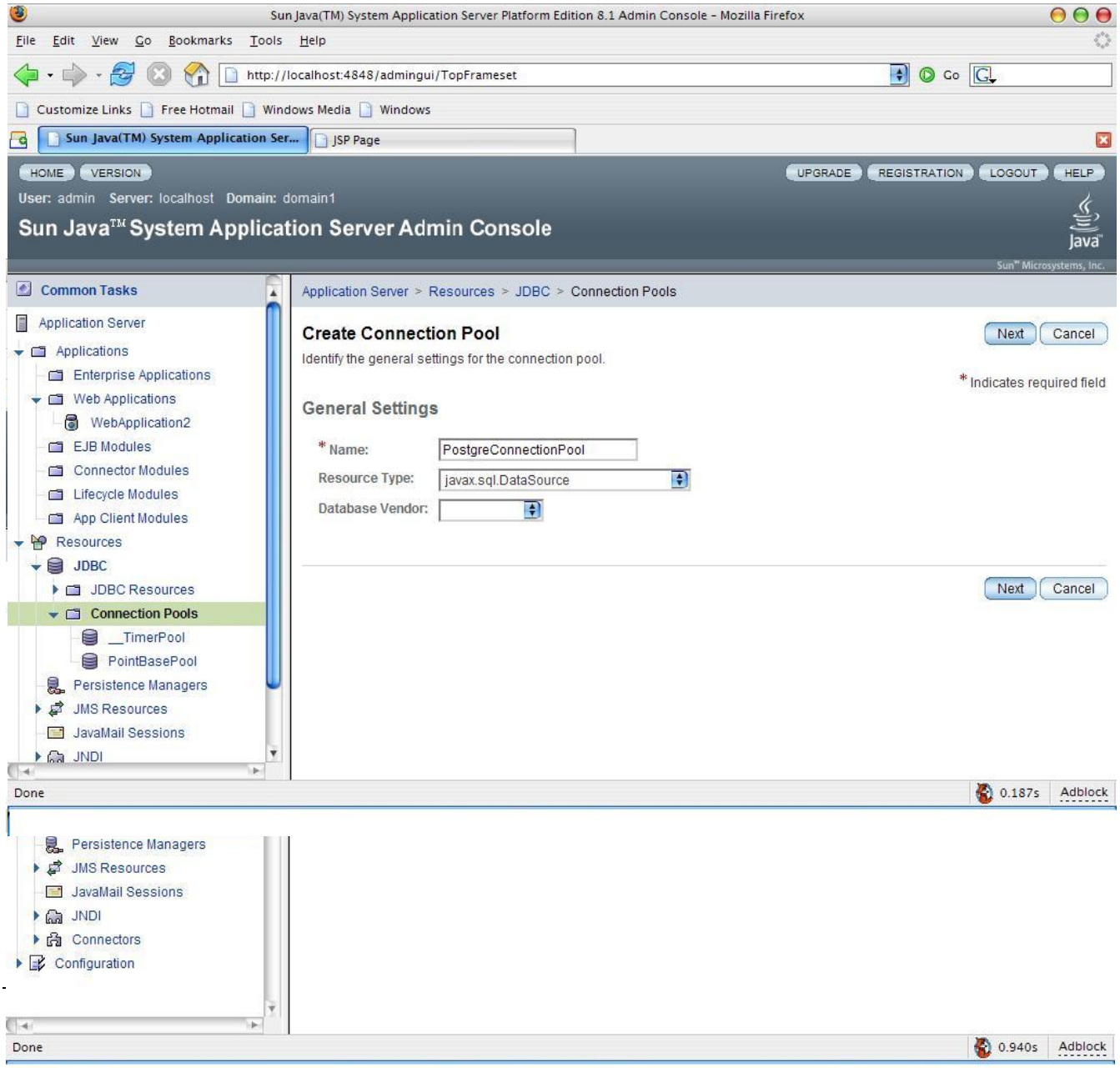
Untuk membuat sebuah connection pool, klik pada link JDBC pada pane sebelah kiri, kemudian klik *Connection Pools* pada pane sebelah kanan.

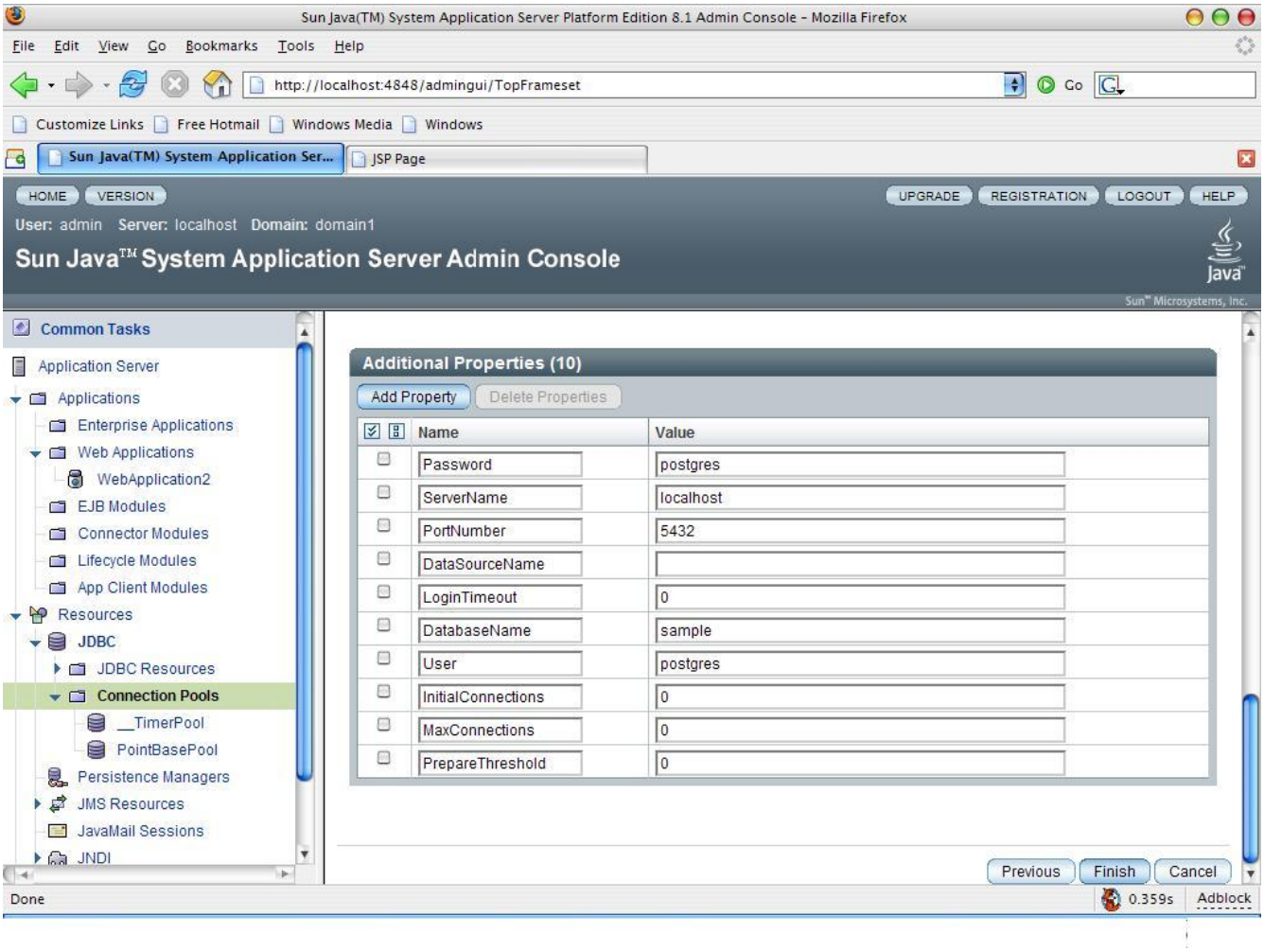
Kemudian pada layar berikutnya, klik pada tombol New... untuk menampilkan layar seperti pada gambar dibawah ini :

Dibawah field *Name,* isilah nama yang mana connection pool ini akan ditunjuk. Dibawah connection pool drop kebawah, pilih javax.sql.DataSource. Tinggalkan Database Vendor yang pilihannya kosong, karena PostgreSQL tidak termasuk dalam list vendor.

Klik next, kemudian jika diminta untuk nama class prompted, masukkan: org.postgresql.jdbc3.Jdbc3PoolingDataSource. klik next.

Pada layar berikutnya, scroll kebawah sampai Anda melihat properti yang akan diasosiasikan dengan connection pool ini:





Parameter-parameter berikut ini harus mempunyai nilai-nilai yang telah disediakan:

- Password
- ServerName
- PortNumber
- DatabaseName
- User

Setelah menyediakan nilai dari semua parameter diatas, klik Finish.

5.2.6 Mendaftarkan DataSource

Untuk mendaftarkan datasource, klik pada link JDBC yang ditemukan pda pane sebelah kiri, lalu klik pada JDBC Resources. Pada layar berikutnya klik pada New...

Field harus diisi dengan ketentuan sebagai berikut:

- JNDI Name – masukkan logical name yang mana aplikasi akan me-retrieve DataSource. Direkomendasikan bahwa namanya memiliki *jdbc/* sebagai prefix-nya, agar lebih mudah bagi server administrators pada masa selanjutnya dalam mengidentifikasi element ini sebagai JDBC resource.
- Pool name – pilih nama dari connection pool yang telah dibuat terlebih dahulu.
- Description – Masukkan text yang menjelaskan tentang DataSource (bebas)

klik pada OK untuk mengakhiri.

5.2.7 Retrieving DataSource

Retrieving merupakan sebuah instance DataSource dari sebuah aplikasi server yang sederhana dan dapat dipenuhi hanya dengan menggunakan beberapa baris dari kode menggunakan bagian dari JNDI API.

Java Naming Directory Interface (JNDI) adalah standard Java API untuk mengakses *directories*. Sebuah direktori merupakan lokasi pusat dimana aplikasi Java dapat retrieve external resources menggunakan logical name.

Detail tambahan untuk JNDI dan bagaimana dia bekerja merupakan lingkup lain pada bahasan ini. Hal yang perlu kita tahu adalah bahwa server aplikasi memelihara sebuah direktori yang mana akan menerbitkan DataSource yang telah kita konfigurasi. Kemudian, aplikasi kita dapat melakukan lookpup sederhana sebuah nama pada direktori tersebut untuk meretrieve resource.

Untuk tujuan kita, sudah cukup bagi kita untuk membuat sebuah context JNDI menggunakan default constructor.Context JNDI ini memisahkan detail dari pengkoneksian direktori, membuat resource lookup semudah memanggil sebuah method single. Ingatlah bahwa nama yang digunakan untuk lookup resource harus sama dengan nama yang digunakan pada konfigurasi DataSource.

```
...
Context ctxt = null;
DataSource ds = null;

try {
   // membuat sebuah instance dari JNDI context yang mana akan melakukan lookup
   ctxt = new InitialContext();

   // retrieve DataSource dari direktori menggunakan logical name
   ds = (DataSource)ctxt.lookup("jdbc/PostgreSQLDS");
}  catch (NamingException ne) {
   System.err("Specified DataSource cannot be found");
}
```

Sekali kita mempunyai valid DataSource instance, mendapatkan sebuah Connection object adalah semudah :

```
Connection conn = ds.getConnection();
```

5.2.8 java.sql.Connection / java.sql.Statement

java.sql.Connection objects menghadirkan connections yang nyata ke database. Sekali kita mempunyai sebuah instance dari object ini, kita dapat membuat sebuah instance dari sebuah Statement object, dimana kemudian kita dapat menggunakan query SQL.

Statement object menyediakan beberapa method untuk mengeksekusi query SQL. Dua method yang sering digunakan adalah:

- executeQuery – menggunakan pernyataan SELECT dan mengembalikan hasil dari operasi sebagai ResultSet object.
- executeUpdate – menggunakan pernyataan INSERT, UPDATE, atau DELETE dan mengembalikan jumlah dari baris yang diperngaruhi sebagai integer primitive.

Dibawah ini adalah bagian dari contoh kode outlining prosedur, bersama dengan beberapa error yang sangat dasar – menangani prosedur.

```
Context ctxt = null;
DataSource ds = null;
Connection conn = null;
Statement stmt = null;
ResultSet rs = null;

try {
   ctxt = new InitialContext();
   ds = (DataSource)ctxt.lookup("jdbc/PostgreSQLDS");

   conn = ds.getConnection();
   stmt = conn.createStatement();

   rs = stmt.executeQuery("SELECT * FROM users");

} catch (NamingException e) {
   System.err("Cannot find named datasource");
} catch (SQLException se) {
   System.err("Error occurred while performing query");
}
```

5.2.9 java.sql.ResultSet

Sebuah ResultSet object encapsulates merupakan hasil dari sebuah query ke database. Data didalam ResultSet object dapat divisualisasikan sebagai tabel.kemudian informasinya dapat diretrieve satu baris pada saat itu juga, dengan ResultSet object menjaga proses tetap pada baris yang ditentukan.

Untuk iterasi diatas baris yang diarahkan pada ResultSet, telah disediakan method yang disebut next(). Memanggil method next() memindahkan internal pointer menjaga ResultSet object ke point baris berikutnya. Method ini mengembalikan nilai true jika ditemukan baris berikutnya, dan nilai false jika tidak ada baris berikutnya.

```
while (rs.next()) {
   //membaca data dari baris sebelum disini
}
```

Contoh dari iterasi ResultSet

Untuk meretrieve data dari tiap-tiad baris, ResultSet object menyediakan beberapa get method. Yaitu sebuah method getString untuk meretrieve data sebagai String, method getInt untuk meretrieve data integer, method getBoolean untuk meretrieve data boolean, dll. Dalam berbagai kasus, method-method ini juga menerima sebagai parameter nomer kolom dari kolom yang berisi data, atau nama kolom.

```
Context ctxt = null;
DataSource ds = null;
Connection conn = null;
Statement stmt = null;
ResultSet rs = null;

try {
   ctxt = new InitialContext();
   ds = (DataSource)ctxt.lookup("jdbc/PostgreSQLDS");

   conn = ds.getConnection();
   stmt = conn.createStatement();

   rs = stmt.executeQuery("SELECT * FROM users");

   while (rs.next()) {
       String userName = rs.getString("name");
       String address = rs.getString("address");
       int userID = rs.getInt("userid");

       // perform operations on retrieved data here.
   }

} catch (NamingException e) {
   System.err("Cannot find named datasource");
} catch (SQLException se) {
   System.err("Error occurred while performing query");
}
```

Ringkasan pada operasi database :

- Memperoleh sebuah Connection object baik menggunakan DriverManager atau memperolehnya dari DataSource object(disarankan).
- Membuat sebuah Statement object menggunakan method createStatement() yang tersedia pada Connection object.
- Melakukan query SQl menggunakan Statement object dan retrieve results.
- Jika hasil dari query adalah sebuah ResultSet object, iterasi diatas baris dengan memanggil method berikutnya secara berulang-ulang ketike meretrieve data di setiap baris.
- Menutup semua databaseyang berhubungan dengan object.