JDBC Avanzado

Universidad de Oviedo

Repositorios de Información

2021-2022

JDBC: Conceptos Avanzados

Statements

Ciclo de Vida

Cuando se llama al método `execute()` del objeto Statement's:

1. El objeto envía la consulta SQL a la BD.
2. La BD compila la consulta SQL analizando su sintaxis.
3. Se realiza un plan de ejecución para dicha consulta.
   - El optimizador de consultas devuelve el plan que considere la mejor alternativa.
   - No siempre es necesario realizarlo, dependerá de lo que se haya estado realizando.
   - La mayoría de los gestores de BD están diseñados para almacenar los planes de ejecución y ejecutarlos varias veces si es necesario. Instrucciones SQL precompiladas.

4. Se ejecuta el plan de ejecución para la consulta SQL ya compilada.
5. Si la consulta devuelve un conjunto de datos, la BD lo almacena en un buffer.
6. Se devuelve el resultado al objeto Statement.
7. Finalmente, la respuesta se envía a la aplicación en forma de ResultSet.

Statements

Interfaces

- **Statement**: Para ejecutar consultas SQL.
  - No se pueden pasar los parámetros a la consulta en tiempo de ejecución al usarlo.
  - Es preferible usarlo cuando la consulta se ejecuta una sola vez.
- **PreparedStatement**: Para ejecutar consultas con parámetros.
  - Extiende de Statement.
  - Se pueden pasar parámetros en tiempo de ejecución.
  - Se recomienda usarlo cuando una consulta se ejecuta varias veces. En este caso el SGBD reutiliza los planes de ejecución.
- **CallableStatement**: Para ejecutar Procedimientos almacenados. Extiende de PreparedStatement.

Statements

PreparedStatement

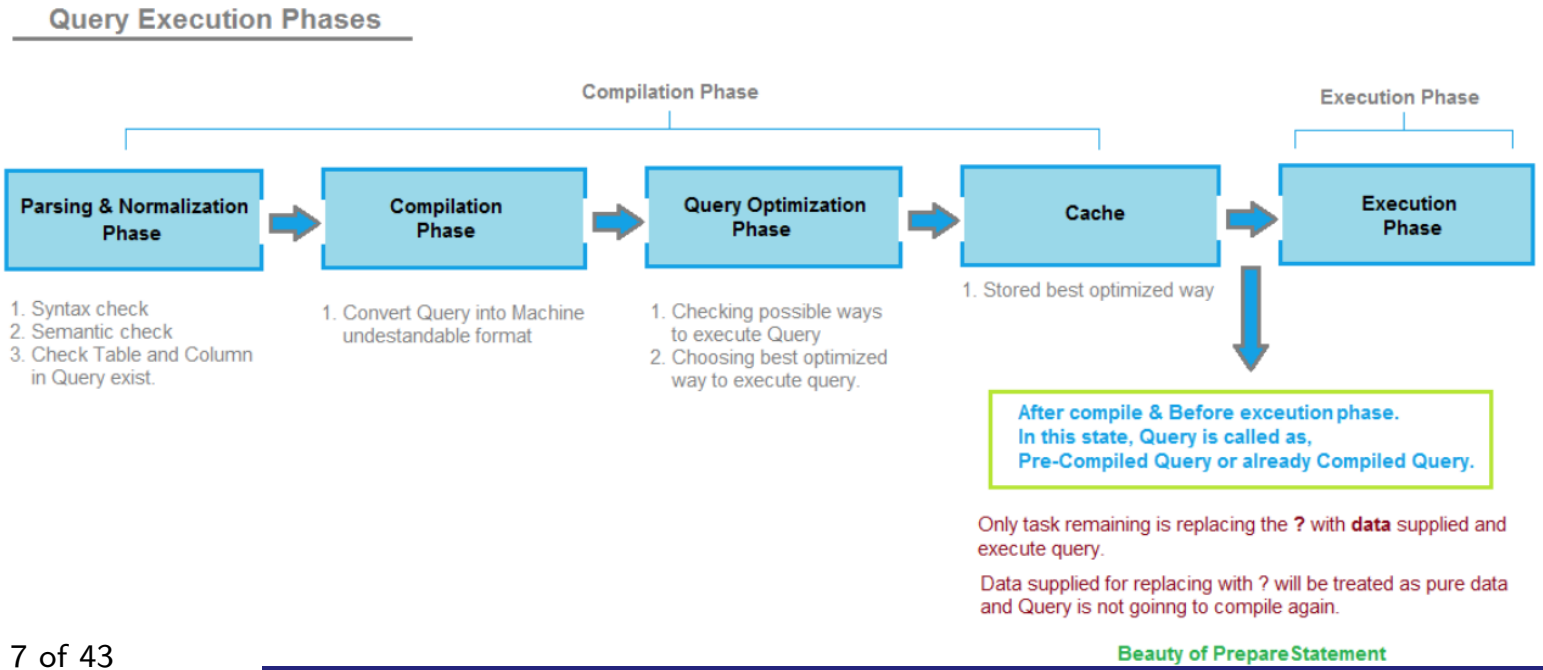
```
PreparedStatement pst = conn.prepareStatement(
    "UPDATE table SET col = ?
     WHERE otherCol = ? AND oneMoreCol = ?");
```

- En lugar de codificar directamente las variables de la consulta SQL (`col = 1`), se usa el placeholder (?)
- Se proporcionan los valores de los parámetros **antes** de la ejecución llamando a los métodos `setXXX`.
- También se puede usar PreparedStatement para consultas DML.
- En este caso los métodos `executeUpdate()`, `executeQuery()` y `execute` no reciben parámetros.

Statements

Ventajas de usar PreparedStatements

1. Posibilidad de crear consultas parametrizadas para que los valores de los parámetros se puedan cambiar dinámicamente.
2. Proporciona seguridad contra ataques de inyección SQL.
3. La precompilación de consultas SQL puede mejorar el rendimiento, evitando que se compile la misma consulta varias veces.



Statements

CallableStatement

Los **procedimientos almacenados** son un conjunto de consultas SQL que ejecutan una determinada tarea, reciben una serie de parametros (IN, OUT, INOUT) y pueden devolver valores discretos o conjuntos de datos como parámetros (OUT, INOUT).

```
CallableStatement mycall = conn.prepareCall
    ("{call procedure_name}");
    ("{call procedure_name(?, ?,...)}");
    ("{? = call function_name}");
    ("{? call function_name(?, ?, ...)}");
```

Statements

CallableStatement

Procedimiento Almacenado

```
create procedure getAvgSalary
  ( IN dept varchar(64), OUT avgSalary decimal (10,2))
begin
     select avg(salary) into avgSalary
     from employees
     where department = dept
end
```

Statements

CallableStatement

Código Java

```
// Prepare callable statement
CallableStatement myCall = conn.prepareCall
      ("{call getAvgSalary(?, ?)}");
// Set parameter
mycall.setString(1, "Human Resources");
mycall.registerOutParameter(2, Types.DOUBLE);
// Execute call
myCall.execute();
// Get result
System.out.println("Average salary: "
                   + myCall.getDouble(2));
```

Statements

Using CallableStatement

Otro ejemplo

```
CallableStatement call = conn.prepareCall(
    "{CALL doubleMyInt(?)}");
// for inout parameters, it is good practice to register
// the out parameter before setting the input value
call.registerOutParameter(1, Types.INTEGER);
call.setInt(1,10);
call.execute();
int retval = call.getInt(1);
```

ResultSet Avanzado

ResultSet en JDBC 2.0

- **sensitivity**, el `ResultSet` es consciente de los cambios realizados por otros (desde fuera de la transacción que ejecuta). Presupone que la propia transacción tiene ajustado un nivel de aislamiento que permite que los cambios sean visibles.
- **scrollability** and **positioning**, el cursor puede moverse en ambas direcciones y posicionarse en una fila cualquiera.
- **updatability**, el ResultSet puede ser actualizado en cualquier momento.

ResultSet en JDBC 2.0

El método `createStatement()` del objeto `Connection` está sobrecargado:
`Statement createStatement(int rsType, int rsConcurrency)`
`rsType` establece la scrollability, posición y la sensibilidad;
`rsConcurrency` establece la capacidad para acutalizarse.
Cuidado al establecer estos parámetros porque puede generar sobrecargas [1]

ResultSet avanzado

Desplazamiento del cursor y sensibilidad a los cambios

Constante	Descripción
TYPE_FORWARD_ONLY	Not scrollable, not positionable, and not sensitive.
TYPE_SCROLL_INSENSITIVE	Scrollable, positionable, and not sensitive (**unless the program queries the database again**).
TYPE_SCROLL_SENSITIVE	Scrollable, positionable, and sensitive.

ResultSet avanzado

Capacidad para actualizarse

Constant	Description
CONCUR_READ_ONLY	El ResultSet no puede ser actualizado, por tanto, el contenido de éste no se puede actualizar con sus métodos..
CONCUR_UPDATABLE	El ResultSet puede ser actualizado.

Podéis encontrar un ejemplo en el siguiente enlace:
`https://examples.javacodegeeks.com/core-java/sql/updatable-resultset-example/`

ResultSet avanzado

Limitaciones del ResultSet

Para que el ResultSet pueda ser actualizable:

- La consulta tiene que ser sobre una única tabla, sin joins.
- Para que los INSERT puedan realizarse correctamente, la consulta debe seleccionar columnas no nulas y sin valores por defecto.
- No se puede usar `SELECT *`.
- No pueden seleccionarse columnas agregadas o derivadas de otras (SUM o MAX).

Para que el ResultSet pueda ser scroll-sensitive:

- No se puede usar `SELECT *`.
- La consulta se debe realizar sobre una única tabla.

ResultSet avanzado

Cursor Holdability

JDBC 3.0/JDK 1.4

Cuando se ejecuta un `Connection.commit()` el cursor del ResultSet que se ha creado durante la transacción se cierra. Este comportamiento se puede controlar:

`HOLD_CURSORS_OVER_COMMIT`: Los cursores del ResultSet se mantienen abiertos cuando se ejecuta un `Connection.commit()`. Esto puede ser útil para aplicaciones que usan mayormente ResultSets de sólo lectura.

`CLOSE_CURSORS_AT_COMMIT`: Los cursores del ResultSet se cierran cuando se ejecuta un `Connection.commit()`.

ResultSet avanzado

Resumen de la visibilidad de cambios internos y externos

Result Set Type	Can see internal DELETE?	Can see internal UPDATE?	Can see internal INSERT?	Can see external DELETE?	Can see external UPDATE?	Can see external INSERT?
forward-only	no	yes	no	no	no	no
scroll-sensitive	yes	yes	no	no	yes	no
scroll-insensitive	yes	yes	no	no	no	no

ResultSet avanzado

Ejercicio

Ejecuta los siguientes proyectos (disponibles en el CV) y comprueba qué es lo que sucede y por qué.

1. `ResultSet-scrollability`: Comprueba la capacidad para mover el cursor por el ResultSet.
2. `ResultSet-Sensitivity-Oracle`: Comprueba según el tipo de ResultSet y la operación qué cambios son visibles. **Comprueba detenidamente cómo se implementa la actualización, inserción y borrado de datos en un ResultSet.**

Conexión avanzada

Conexión avanzada

`java.sql.Connection`.

- Para aplicaciones stand-alone o de escritorio se suele usar el método `DriverManager.getConnection(...)`
  - Es necesario un controlador y una URL específica para la BD. Esto hace que la aplicación dependa del controlador y por tanto de la BD.
- La interfaz `DataSource` proporciona una manera totalmente desacoplada de que el cliente JDBC obtenga la conexión (leer [3])

Conexión avanzada

Ciclo de vida del DataSource

Los **Data sources** son objetos estándar de uso general para especificar conexión a BBDD u otros fuentes de datos.
Tienen propiedades para determinar la identidad de la fuente de datos y la manera en que se manejan los datos en una conexión.
Pueden ser modificados en cualquier momento. Si por ejemplo la fuente de datos cambia a un servidor diferente, se cambia la propiedad para ese servidor con facilidad.

Conexión avanzada

Ciclo de vida del DataSource

1. Los objetos Datasource se deben desplegar.
   - Se crea una instancia del objeto
   - Se establecen las propiedades (server, database name, . . .)

```
// Initialize Data Source Properties

OracleDataSource ods = new OracleDataSource();

ods.setDriverType("thin");
ods.setServerName("156.35.94.99");
ods.setDatabaseName("DESA");
ods.setPortNumber(1521);
```

   - Los objetos DataSource objects se vinculan a entidades **Java Naming and Directory Interface** (JNDI) a través de un **nombre lógico**.

```
Hashtable<String, String> env = new Hashtable<String, String>();
env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.rmi.registry.RegistryContextFactory");
/*
 * env.put(Context.PROVIDER_URL,"rmi://localhost:1099");
 */
// Register the Data Source
Context ctx = new InitialContext(env);
ctx.bind(name, ds);
```

Conexión avanzada

Ciclo de vida del DataSource

2. Las aplicaciones pueden usar los nombres lógicos para acceder a servicios, eliminando del código sintaxis especificas de los fabricantes. Esto aumenta la **portabilidad**.
3. Un Datasource se obtiene a través de una búsqueda.

```
Hashtable<String, String> env = new Hashtable<String, String>();
env.put(Context.INITIAL_CONTEXT_FACTORY, "com.sun.jndi.rmi.registry.RegistryContextFactory");
/*
 * env.put(Context.PROVIDER_URL,"rmi://localhost:1099");
 */

return (DataSource) new InitialContext(env).lookup(jndiUrl);
```

4. Una vez recuperado se utiliza para crear la conexión.

```
conn = dataSource.getConnection(USERNAME, PASSWORD);
```

Conexión avanzada

Ventajas del DataSource

- Aplicaciones más portables porque el nombre del controlador la URL no forman parte del código de forma implícita.
- Código más simple y fácil de mantener. Los cambios afectan sólo al Datasource y no al código del cliente.
- Capacidad para trabajar con pool de conexiones de forma automática.

Conexión avanzada

Ejercicios

- `NO-JNDI-DataSource-Oracle`
- Para que funcione el Servidor hay que iniciar el `rmiregistry` desde la línea de comandos: `JNDI-DataSource-Server` and `JNDI-DataSource-Client`

Para más información, leer [4]

Pool de conexiones

Pool de conexiones

Implementaciones

Hay tres tipos de implementaciones:

- **Basic implementation** devuelve un objeto Connection estándar.
- **Connection pooling implementation** devuelve un objeto Connection que forma parte de un pool de conexiones.
- **Distributed transaction implementation** devuelve un objeto Connection para usarse por transacciones distribuidas y que suelen formar parte de un pool de conexiones.

Pool de conexiones

Un pool de conexiones es una caché de objetos Connection a una BD. Los objetos representan conexiones físicas a la BD que pueden ser usadas por una aplicación para conectarse a ella.
Puede ser implementado por el driver o por una librería de terceros:

- OracleConnectionCacheManager (para/de Oracle)
- c3p0
- BoneCP
- Jakarta DBCP

Pool de conexiones

Ciclo de vida

En tiempo de ejecución:

- La aplicación solicita una conexión al **DataSource**. Si el pool contiene una conexión que satisface la solicitud, devuelve la conexión a la aplicación. Si no, se crea una nueva conexión.
- Cuando la aplicación finaliza su ejecución, devuelve la conexión al pool. Dicha conexión estará disponible para una nueva petición.

RowSet

RowSet

Un RowSet es un objeto que encapsula un conjunto de filas por un ResultSet y/o otras tipos de fuentes.

- Es más simple y adaptable que un ResultSet.
- Proporciona actualización y scrollability (por defecto) para cualquier SGBD o controlador.
- Es una copia en memoria de los datos de la BD, por lo que puede ser actualizado en memoria y volcarlo a la BD después (acceptChanges(), ...).
- Como es un componente JavaBean se puede serializar y enviar.
- Soporta eventos JavaBeans permitiendo que se pueda notificar a otros componentes de la aplicación cuando ocurre un evento como un cambio de valor.

Row sets

ResultSet vs RowSet.

ResultSet	RowSet
A ResultSet always maintains connection with the database.	A RowSet can be connected, disconnected from the database.
It cannot be serialized.	A RowSet object can be serialized.
ResultSet object cannot be passed other over network.	You can pass a RowSet object over the network.
ResultSet object is not a JavaBean object You can create/get a result set using the **executeQuery()** method.	ResultSet Object is a JavaBean object. You can get a RowSet using the **RowSetProvider.newFactory().createJdb cRowSet()** method.
By default, ResultSet object is not scrollable or, updatable.	By default, RowSet object is scrollable and updatable.

RowSet

Tipos

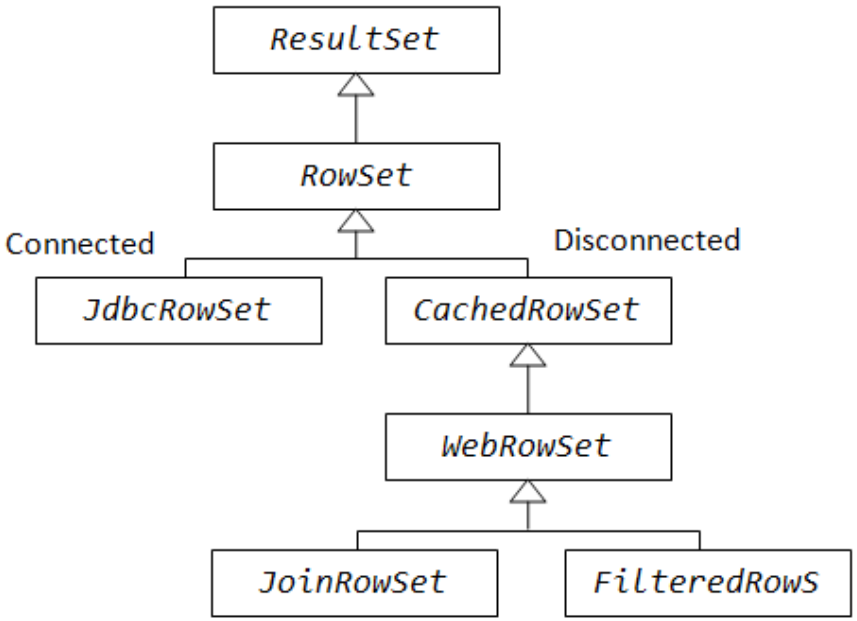
Los RowSet se clasifican **dependiendo de la duración de la conexión**.
Un Rowset **connected** usa el controlador JDBC para establecer una conexión que se mantiene abierta durante toda la vida del objeto.
Un Rowset **disconnected** se conecta para leer o escribir y una vez termina se desconecta de la fuente.

- Los datos pueden cambiar mientras que el RowSet está desconectado.
- Los datos modificados se pueden actualizar en la base de datos después de que se restablezca la conexión.

RowSet

Implementaciones

Como ocurre con los pool de conexión, la implementación de un RowSet no necesariamente la proporciona el controlador. Hay librerías de terceros que lo implementan.



Interfaces Rowset

JdbcRowSet

Son Connected RowSet que hacen de envoltorio del ResultSet para que el controlador JDBC se parezca a un componente JavaBean.

```
JdbcRowSet jdbcRs = new JdbcRowSetImpl();
jdbcRs.setUsername("scott");
jdbcRs.setPassword("tiger");
jdbcRs.setUrl("jdbc:oracle://localhost:3306/test");
jdbcRs.setCommand("select * from employee");
jdbcRs.execute();
while(jdbcRs.next()) {
System.out.println(jdbcRs.getString("emp_name"));
}
```

Interfaces Rowset

CachedRowSet

Son Disconnected RowSet.
Mantiene los datos en memoria.
Es serializable.
No es apropiado para usar con grandes volúmenes de datos pero ideal para clientes pequeños (PDA, . . .).

```
CachedRowSet cachedRs = new CachedRowSetImpl();
cachedRs.setUsername("scott");
cachedRs.setPassword("tiger");
cachedRs.setUrl("jdbc:oracle://localhost:3306/test");
cachedRs.setCommand("select * from employee");
cachedRs.setPageSize(4);
cachedRs.execute();
while (cachedRs.nextPage()) {
while (cachedRs.next()) {
System.out.println(cachedRs.getString("emp_name"));
}
}
```

RowSet interfaces

WebRowSet, JoinRowSet, FilteredRowSet

- **WebRowSet** Extiende de CachedRowSet. Puede leer y escribir en formato XML.
- **JoinRowSet** Extiende de WebRowSet. Permite combinar datos de dos RowSet diferentes. Útil cuando hay datos relacionados en dos fuentes diferentes.
- **FilteredRowSet** Extiende de WebRowSet. Proporciona opciones de filtrado (WHERE implementado por next()).

RowSet

Modelo de eventos

Un RowSet, a través de eventos JavaBeans, podría **notificar a diferentes componentes** que estén escuchando cuando suceda algún evento. El mecanismo para que esto suceda involucra a ambas partes, el componente a modificar y el RowSet.

1. Los **componentes** a los que se quiere modificar, deben implementar la interfaz RowSetListener.
2. El objeto **RowSet** debe registrar cada componente añadiéndolo a su lista de componentes a los que quiere notificar (`RowSet.addRowSetListener()`).

RowSet

Modelo de eventos

- **Database cursor movement events** indica que el cursor se ha desplazado (`cursorMoved()`)
- **Row change events** indica que una fila ha sido insertada, actualizada o borrada (`rowChanged()`).
- **Rowset change events** indica que el contenido del RowSet ha cambiado (`rowSetChanged()`).

Connection pools

Ejemplos

Echa un vistazo al proyecto `CachedRowSetTest` y ejecútalo.
Ojo con las credenciales Oracle.

Lectura Complementaria (1)

[1] https://docs.oracle.com/cd/B19306_01/java.102/b14355/resltset.htm

[2] https://docs.oracle.com/cd/A84870_01/doc/java.816/a81354/resltse7.htm

[3] Read Oracle Database JDBC Developer's Guide and Reference, chapter 8

[4] https://docs.oracle.com/javase/tutorial/jdbc/basics/sqldatasources.html