♦ 12 FAQ JDBC interview questions and answers

Posted on September 4, 2014 by Arulkumaran Kumaraswamipillai — No Comments ↓

Q1. What are JDBC Statements? What are different types of statements? How can you create them?

A1. A statement object is responsible for sending the SQL statements to the Database. Statement objects are created from the connection object and then executed.

```
1 Statement stmt = myConnection.createStatement();
2 ResultSet rs = stmt.executeQuery("SELECT id, name
3                                  or
4 stmt.executeUpdate("INSERT INTO (field1,field2) v
5
```

The types of statements are:

- Statement (regular statement as shown above).
- PreparedStatement (more efficient than statement due to pre-compilation of SQL).
- CallableStatement (to call stored procedures on the database).

To use **prepared statement**:

```
1 PreparedStatement prepStmt = myConnection.prepare
2 prepStmt.setInt(1, 1245);
```

Callable statements are used for calling stored procedures.

```
1 CallableStatement calStmt = myConnection.prepareC
2 ResultSet rs = cs.executeQuery();
3
```

Q2. What is the difference between statements and prepared statements? Which one would you favor and why?
A2.

#1. Prepared statements offer better performance, as they are pre-compiled. Prepared statements reuse the same execution plan for different arguments rather than creating a new execution plan every time. Prepared statements use bind arguments, which are sent to the database engine. This allows mapping different requests with same prepared statement but different arguments to execute the same execution plan.

#2. Prepared statements are more secured because they use bind variables, which can prevent SQL injection attack.

The most common type of SQL injection attack is SQL manipulation. The attacker attempts to modify the SQL statement by adding elements to the WHERE clause or extending the SQL with the set operators like UNION, INTERSECT etc.

```
1 SELECT * FROM users
```


16 Technical Key Areas

```
2         WHERE username='bob'
3           AND accountid=1234;
```

The attacker can manipulate the SQL as follows

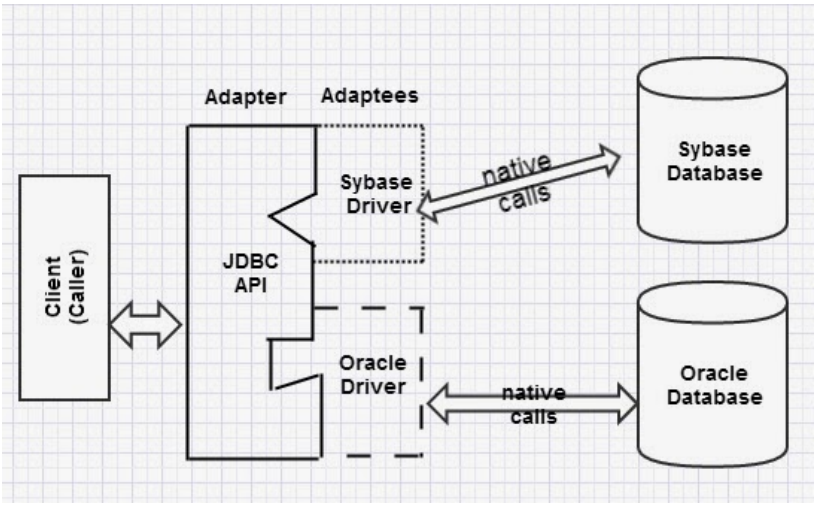
```
1 SELECT * FROM users
2       WHERE username='bob'
3       AND accountid=1234 OR 'a' = 'a' ;
```

The above "WHERE" clause is always true because of the operator precedence. The PreparedStatement can prevent this by using bind variables:

```
1 String strSQL = SELECT * FROM users where usernam
2 PreparedStatement pstmt = myConnection.prepareSta
3 pstmt.setString(1,"bob");
4 pstmt.setLong(2, 1234);
5 pstmt.execute();
6
```

Q3. What design pattern does JDBC use?

A3. Adapter design pattern. An adapter adapts at run time like the decorator design pattern. Adapter design pattern is one of the structural design patterns and its intent is to get two unrelated interfaces work together. Think of using a laptop in UK that was bought in Japan as the sockets are different, and you need an adapter. So, the adapter's intent is to adapt between the Japanese laptop plug with UKs wall socket. The key point is that parties are different. Japanese laptop used in third-party or external (i.e. UK) wall socket.



80+ step by step Java Tutorials

100+ Java pre-interview coding tests

JDBC – Adapter Design Pattern

Adapter is used when you have an abstract interface, for example a JDBC API and you want to map that interface to another object which has similar functional role, but a different interface, for example different JDBC drivers for different databases like Oracle, Sybase, DB2, SQL server, MySQL, etc. The JEE have multiple adaptors for JMS, JNDI, JDBC, JCA, etc. The drivers and implementations are generally provided by the third party vendors. For example, JMS implementations provided by third-party vendors and open source providers web Methods, IBM MQ Series, ActiveMQ, etc.

Q4. How will you bootsrap the JDBC driver?
A4.

#1. Dynamic class loading with ***Class.forName(….)*** will load the driver and register it with the ***DriverManager***. The driver jar file supplied by the Oracle database vendor or MySQL vendor must be in the classpath. “oracle.jdbc.driver.OracleDriver” is class in the driver jar file.

```
1 Class.forName(“oracle.jdbc.driver.OracleDriver”);
2 String url = jdbc:oracle:thin:@hostname:1526:myDB
3 Connection myConnection =  DriverManager.getConne
4
```

#2. The DataSource interface provides an alternative to the DriverManager for making a connection. DataSource makes the code more portable than DriverManager because it works with JNDI and it is created, deployed and managed separately from the application that uses it. If the DataSource location changes, then there is no need to change the code but change the configuration properties in the server. This makes your application code easier to maintain. DataSource allows the use of connection pooling and support for distributed transactions.

A DataSource is configured on the application server with the following properties

DataSource config

Once the DataSource has been set up, then you can get the connection object as follows:

```
1 Context ctx = new InitialContext();
2 DataSource ds = (DataSource)ctx.lookup("jdbc/myDa
3 Connection myConnection = ds.getConnection(“usern
4
```

Q5. How will you create a DataSource in Spring?

A5. Using **Apache commons-dbcp** package that has the ***org.apache.commons.dbcp.BasicDataSource*** class. The pom.xml file for maven should declare the dependency.

```
1  <properties>
2     <commons-dbcp.version>1.4</commons-dbcp.versi
3  </properties>
4
5  <dependencies>
6   <dependency>
7    <groupId>commons-dbcp</groupId>
8    <artifactId>commons-dbcp</artifactId>
9    <version>${commons-dbcp.version}</version>
10  </dependency>
11 </dependencies>
12
```

Configure the application server specific datasource file. For example, in **JBoss application server** configure **my-ds.xml** with JNDI.

```
1  <datasources>
2     <local-tx-datasource>
3         <jndi-name>jdbc.dataSource.my_jndi</jndi-
4         <use-java-context>false</use-java-context
5         <connection-url>jdbc:sybase:Tds:my-server
6         <driver-class>com.sybase.jdbc3.jdbc.SybDr
7         <user-name>user</user-name>
8         <password>password</password>
```

```
        <max-pool-size>50</max-pool-size>
        <exception-sorter-class-name>org.jboss.re
        <new-connection-sql>select count(1) from
        <check-valid-connection-sql>select count(
    </local-tx-datasource>

</datasources>

```

Finally, the Spring configuration to use the JNDI name

```
<bean id="datasource_abc" class="org.springframe
  <property name="jndiName">
     <value>jdbc.dataSource.my_jndi</value>
  </property>
</bean>

<bean id="jdbcTemplate_abc" class="org.springfra
   <property name="dataSource" ref="datasource_a
</bean>

```

Q6. Why lookup via JNDI for DataSources and other resources like JMS Queues, etc?

A6. JNDI based creation allows you to move an application between environments like development to UAT and then to integration and finally to production. If you configure each application server to use the same JNDI name, you can have different databases in each environment and not required to change your code. You just pick up the same environment free WAR file and drop it in any environment. In other words, the environment details are externalized.

Q7. What are the different ResultSet types, and what do they determine?

A7. The **ResultSet types** determine the ways in which the cursor can be manipulated, and how concurrent changes made to the underlying data source are reflected by the ResultSet object.

The cursor manipulation can be determined by one of **three different ResultSet types:**

TYPE_FORWARD_ONLY: The result set cannot be scrolled; its cursor moves forward only, from before the first row to after the last row. The rows contained in the result set depend

on how the underlying database generates the results. That is, it contains the rows that satisfy the query at either the time the query is executed or as the rows are retrieved.

TYPE_SCROLL_INSENSITIVE: The result can be scrolled; its cursor can move both forward and backward relative to the current position, and it can move to an absolute position. The result set is insensitive to changes made to the underlying data source while it is open. It contains the rows that satisfy the query at either the time the query is executed or as the rows are retrieved.

TYPE_SCROLL_SENSITIVE: The result can be scrolled; its cursor can move both forward and backward relative to the current position, and it can move to an absolute position. The result set reflects changes made to the underlying data source while the result set remains open.

The **default ResultSet type is TYPE_FORWARD_ONLY**. Generally, TYPE_SCROLL_INSENSITIVE is the preferred option. The data contained in the ResultSet object is fixed (a snapshot) when the object is created.

ResultSet Concurrency: The concurrency of a ResultSet object determines what level of **update functionality is supported**.

There are two concurrency levels:

CONCUR_READ_ONLY: The ResultSet object cannot be updated using the ResultSet interface.

CONCUR_UPDATABLE: The ResultSet object can be updated using the ResultSet interface.

The **default** ResultSet concurrency is **CONCUR_READ_ONLY**.

```
1  ResultSet rs;
2  Connection con = null;
3  public void fetchResultSet()
```

```
4  {
5     try {
6          if(con==null || con.isClosed())
7          {
8            Class.forName("sun.jdbc.odbc.JdbcOdbc"
9            con = DriverManager.getConnection("jdb
10         }
11         Statement stmt = con.createStatement(Res
12         String query = "select * from Stocktbl";
13         rs = stmt.executeQuery(query);
14       }catch(Exception ex)
15       {
16          System.out.println(ex);
17          Logger.getLogger(StockScr.class.getName
18
19          try
20          {
21             if(con != null)
22             {
23               con.close();
24             }
25          }catch(Exception x){}
26       }
27 }
28
```

Q8. What is Cursor Holdability?

A8. The cursor holdability feature was added in JDBC 3.0. Calling the method con.commit can close the ResultSet objects that have been created during the current transaction. In some cases, this behavior is not desired. The ResultSet property holdability gives the application control over whether ResultSet objects (cursors) are closed when commit is called. The following ResultSet constants may be supplied to the Connection methods createStatement, prepareStatement, and prepareCall:

HOLD_CURSORS_OVER_COMMIT: ResultSet cursors are not closed; they are holdable: they are held open when the method commit is called. Holdable cursors might be ideal if your application uses mostly read-only ResultSet objects.

CLOSE_CURSORS_AT_COMMIT: ResultSet objects (cursors) are closed when the commit method is called. Closing cursors when this method is called can result in better performance for some applications.
The default cursor holdability varies depending on your DBMS.

Note: Not all JDBC drivers and databases support holdable and non-holdable cursors. The following method, JDBCTutorialUtilities.cursorHoldabilitySupport, outputs the default cursor holdability of ResultSet objects and whether HOLD_CURSORS_OVER_COMMIT and CLOSE_CURSORS_AT_COMMIT are supported

Q9. What is **RowSet**? What is the difference between **RowSet** and **ResultSet**? What are the advantages of using RowSet over ResultSet?
A9. RowSets are a JDBC 2.0 extension to the java.sql.ResultSet interface. Guess what, it makes life a lot easier for all JDBC programmers. No more Connection objects, statement objects, just a single RowSet will do everything for you. RowSet object follows the JavaBeans model for properties and event notification, it is a JavaBeans component that can be combined with other components in an application.

The ResultSet has an 'open connection' to the database whereas a RowSet works in a 'disconnected' fashion. It has the following advantages over a ResultSet.

— Since a RowSet works in a disconnected mode, especially for "read-only" queries, it would have better performance in a highly concurrent system.
— Rowsets have many different implementations to fill different needs. These implementations fall into two broad categories, rowsets that are connected and those that are disconnected.
— Rowsets make it easy to send tabular data over a network. They can also be used to provide scrollable result sets or updatable result sets in special cases when the underlying JDBC driver does not support them.

RowSet disadvantages.

— Rowset keeps all the data from the query result in memory. This is very in-efficient for queries that return huge data.

There are 3 types of RowSets.

JdbcRowSet is a connected type of rowset as it maintains a connection to the data source using a JDBC driver.

```
1 JdbcRowSet jdbcRowSet = new JdbcRowSetImpl();
2 jdbcRowSet.setCommand("SELECT * FROM Course);
3 jdbcRowSet.setURL("jdbc:hsqldb:hsql://localhost/m
4 jdbcRowSet.setUsername("sa");
5 jdbcRowSet.setPassword("pwd");
6 jdbcRowSet.execute();
7
```

CachedRowSet and **WebRowSet** are disconnected types of rowsets as they are connected to the data source only when reading data from it or writing data to it.

```
1 ResultSet rs = stmt.executeQuery("SELECT * FROM C
2 CachedRowSet crset = new CachedRowSetImpl();
3 crset.populate(rs);
4
```

```
1 WebRowSet wrs = new WebRowSetImpl();
2 wrs.populate(rs);
3 wrs.absolute(2)
4 wrs.updateString(1, "JNDI");
5
```

Q10. What is Metadata and why should you use it?
A10. JDBC API has 2 Metadata interfaces — **DatabaseMetaData** & **ResultSetMetaData**. The meta data means data about data, and provides comprehensive information about the database as a whole. The implementation for this interface is implemented by database driver vendors to let users know the capabilities of a Database.

```
1 ResultSet rs = stmt.executeQuery("SELECT a, b, c
2 ResultSetMetaData resultSetMeta = rs.getMetaData(
3 int numberOfColumns = resultSetMeta.getColumnCoun
4 boolean b = resultSetMeta.isSearchable(3);
5
```

Q11. What are database warnings and why do you need them?
A11. Warnings are issued by a database to inform user of a

problem which may not be very severe. Database warnings do not stop the execution of SQL statements. Warnings are silently chained to the object. You need warnings for the reporting purpose. Warnings may be retrieved from Connection, Statement, and ResultSet objects.

```
1  SQLWarning warning = conn.getWarnings();
2  QLWarning nextWarning = warning.getNextWarning()
3  conn.clearWarnings();
4  ...
5  stmt.getWarnings();
6  stmt.clearWarnings();
7  ...
8  rs.getWarnings();
9  ...
10
```

Q12. What is new in JDBC 3.0?

A12.

#1.Savepoint support, which allows you to define, release, and rollback a transaction to a savepoint. Traditionally, database transactions have been “all or nothing” types of events — start a transaction, insert some rows, do some updates, and then either commit or rollback. With JDBC 3.0, the transactional model is now more flexible. An application might start a transaction, insert several rows and then create a savepoint. This savepoint serves as a bookmark. The application might then perform some if/then/else type of logic such as updating a group of rows. The application might conclude that the updates made were a bad idea but the initial inserts were okay. The application can rollback to the the bookmark (i.e. savepoint), and then commit the group of inserts as if the updates have never been attempted.

#2. Ability to have multiple open ResultSet objects. JDBC 3.0 gives the programmer the flexibility to decide if he/she wants concurrent access to result sets generated from procedures or if he/she wants the resources to be closed when a new result set is retrieved, which is the JDBC 2.0 compliant behavior.

#3. Ability to control how prepared statements are pooled and reused by connections with deployment configurations.

#4. Ability to pass parameters to **CallableStatement** objects **by name**.

#5.Ability to retrieve auto generatet keys. Many databases have hidden columns (aka pseudo columns) that represent a unique key over every row in a table. For example, Oracle and Informix have ROWID pseudo columns. An optional feature of the JDBC 3.0 specification is the ability to retrieve auto generated key information for a row when the row is inserted into a table.

#6. JDBC 3.0 introduces a standard mechanism for updating BLOB and CLOB data. Even though JDBC 2.0 provided mechanisms to read BLOB and CLOB data, but it lacked an update capability for those types.

Popular Posts

♦ 11 Spring boot interview questions & answers

825 views

♦ Q11-Q23: Top 50+ Core on Java OOP Interview Questions & Answers

766 views

18 Java scenarios based interview Questions and Answers

400 views

001A: ♦ 7+ Java integration styles & patterns interview questions & answers

388 views

01b: ♦ 13 Spring basics Q8 – Q13 interview questions & answers

295 views

♦ 7 Java debugging interview questions & answers

293 views

01: ♦ 15 Ice breaker questions asked 90% of the time in Java job interviews with hints

285 views

♦ 10 ERD (Entity-Relationship Diagrams) Interview Questions and Answers

279 views

♦ Q24-Q36: Top 50+ Core on Java classes, interfaces and generics interview questions & answers

239 views

001B: ♦ Java architecture & design concepts interview questions & answers

201 views

Bio | Latest Posts

Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.**945+** paid members. join my LinkedIn Group. **Reviews**

About Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.**945+** paid members. join my LinkedIn Group. **Reviews**

‹ ♦ Java exception handling interview questions and answers

♦ 16 FAQ JMS interview questions & answers ›

Posted in FAQ JEE Job Interview Q&A Essentials, JDBC, member-paid

Tags: Java/JEE FAQs, JEE FAQs

Leave a Reply

Logged in as geethika. Log out?

Comment

Post Comment

Empowers you to open more doors, and fast-track

Technical Know Hows

☀ Java generics in no time ☀ Top 6 tips to transforming your thinking from OOP to FP ☀ How does a HashMap internally work? What is a hashing function?
☀ 10+ Java String class interview Q&As ☀ Java auto un/boxing benefits & caveats ☀ Top 11 slacknesses that can come back and bite you as an experienced Java developer or architect

Non-Technical Know Hows

☀ 6 Aspects that can motivate you to fast-track your career & go places ☀ Are you reinventing yourself as a Java developer? ☀ 8 tips to safeguard your Java career against offshoring ☀ My top 5 career mistakes

Prepare to succeed

☀ Turn readers of your Java CV go from “Blah blah” to “Wow”? ☀ How to prepare for Java job interviews? ☀ 16 Technical Key Areas ☀ How to choose from multiple Java job offers?

Select Category

© Disclaimer

 ↑