Java Basic for Tester

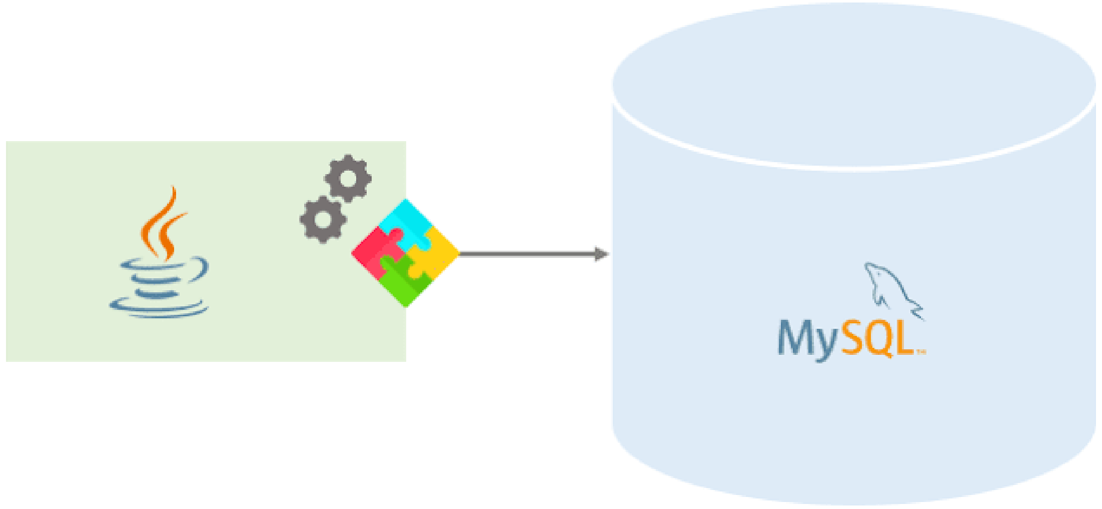
JDBC in Java

Agenda

- *JDBC - Introduction*
- *JDBC - Driver Types*
- *JDBC - Connections*
- *JDBC - Statements*
- *JDBC - Result Sets*
- *JDBC - Data Types*
- *JDBC - Transactions*
- *JDBC - Exceptions*

JDBC - Introduction

JDBC stands for Java Database Connectivity, which is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.



JDBC - Introduction

The JDBC library includes APIs for each of the tasks mentioned below that are commonly associated with database usage.

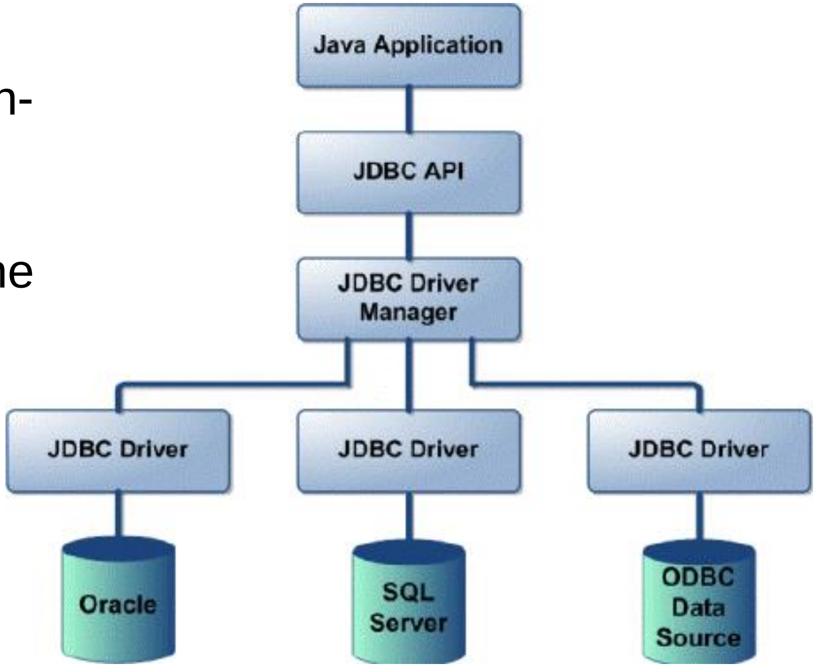
1. Making a connection to a database.
2. Creating SQL or MySQL statements.
3. Executing SQL or MySQL queries in the database.
4. Viewing & Modifying the resulting records.

JDBC - Introduction

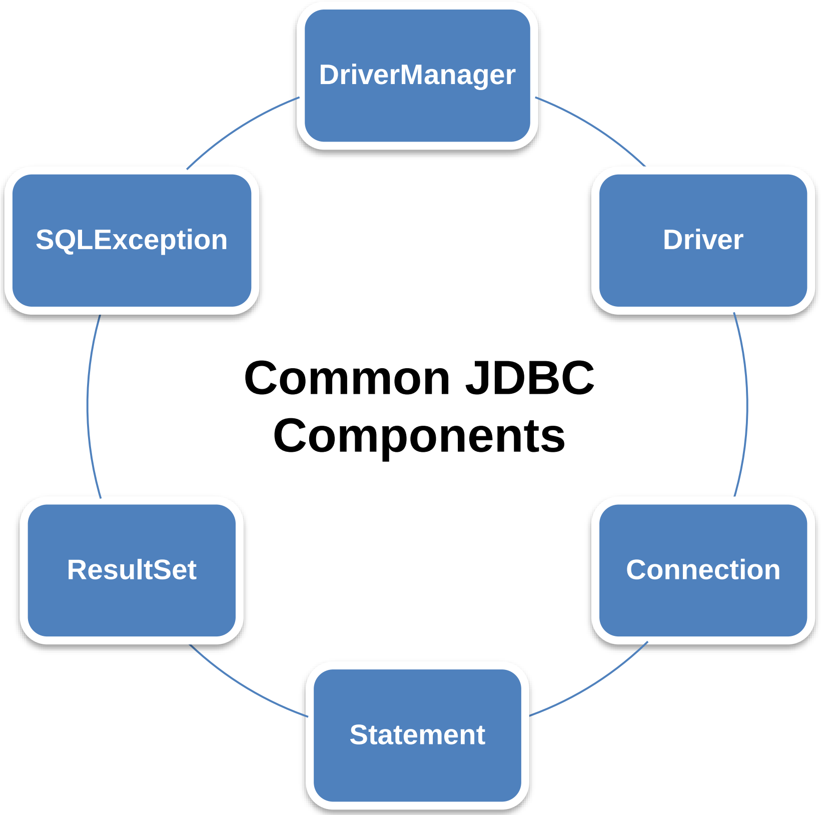
JDBC Architecture

- **JDBC API:** This provides the application-to-JDBC Manager connection.

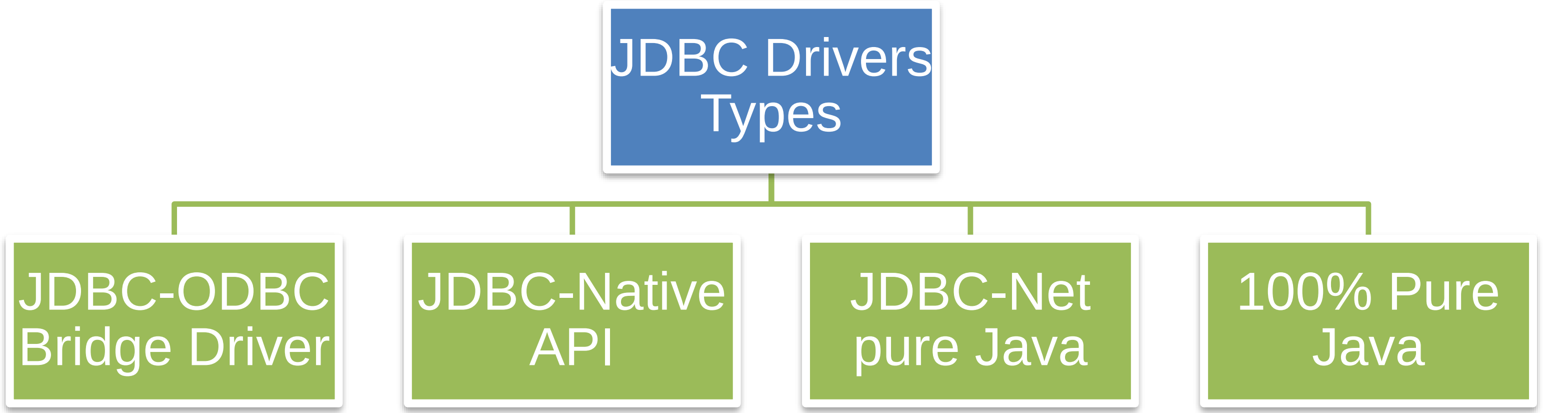
- **JDBC Driver API:** This supports the JDBC Manager-to-Driver Connection.



JDBC - Introduction



JDBC - Driver Types

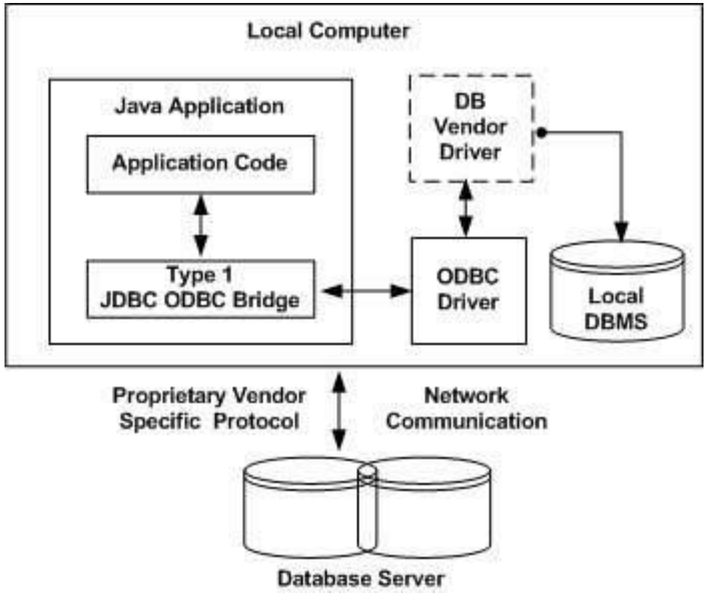


JDBC - Driver Types

Type 1: JDBC-ODBC Bridge Driver

In a Type 1 driver, a JDBC bridge is used to access ODBC drivers installed on each client machine. Using ODBC, requires configuring on your system a Data Source Name (DSN) that represents the target database.

When Java first came out, this was a useful driver because most databases only supported ODBC access but now this type of driver is recommended only for experimental use or when no other alternative is available.

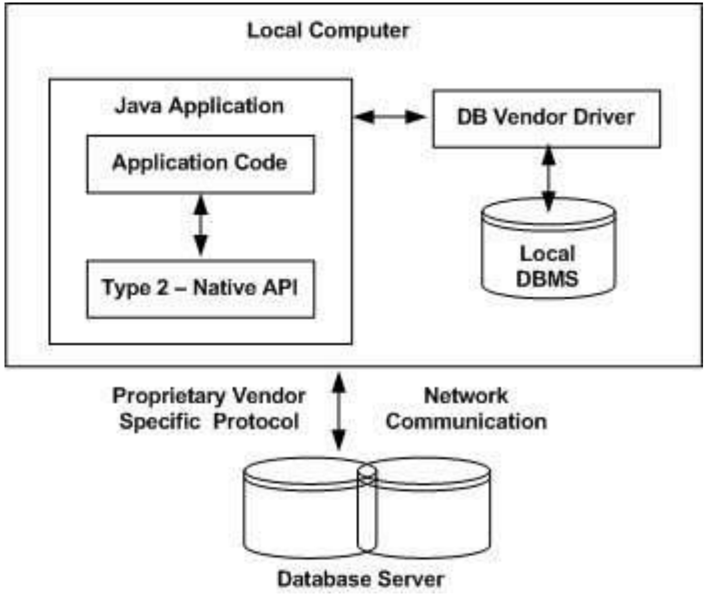


JDBC - Driver Types

Type 2: JDBC-Native API

In a Type 2 driver, JDBC API calls are converted into native C/C++ API calls, which are unique to the database. These drivers are typically provided by the database vendors and used in the same manner as the JDBC-ODBC Bridge. The vendor-specific driver must be installed on each client machine.

If we change the Database, we have to change the native API, as it is specific to a database and they are mostly obsolete now, but you may realize some speed increase with a Type 2 driver, because it eliminates ODBC's overhead.

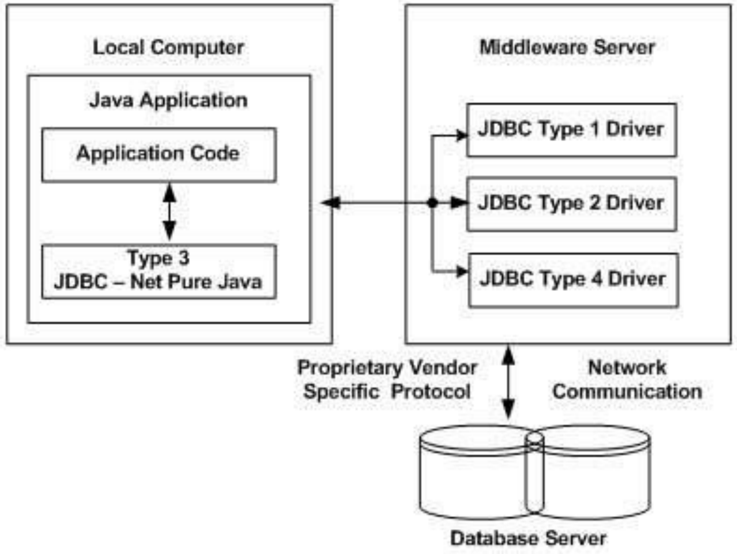


JDBC - Driver Types

Type 3: JDBC-Net pure Java

In a Type 3 driver, a three-tier approach is used to access databases. The JDBC clients use standard network sockets to communicate with a middleware application server. The socket information is then translated by the middleware application server into the call format required by the DBMS, and forwarded to the database server.

This kind of driver is extremely flexible, since it requires no code installed on the client and a single driver can actually provide access to multiple databases.

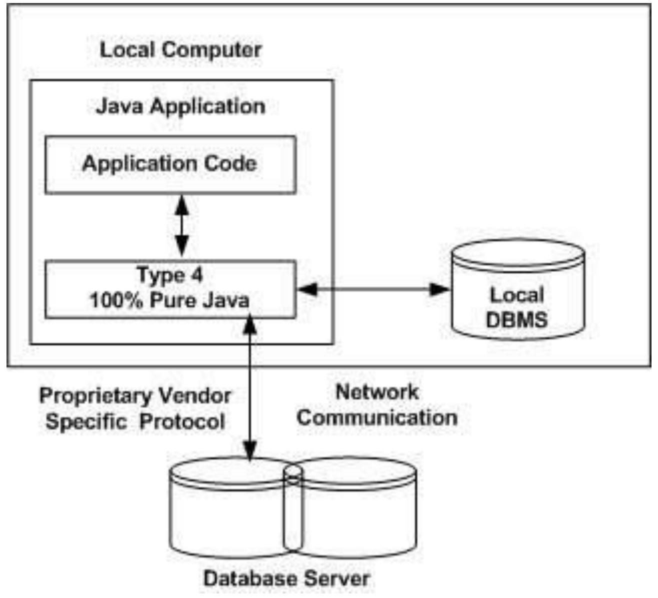


JDBC - Driver Types

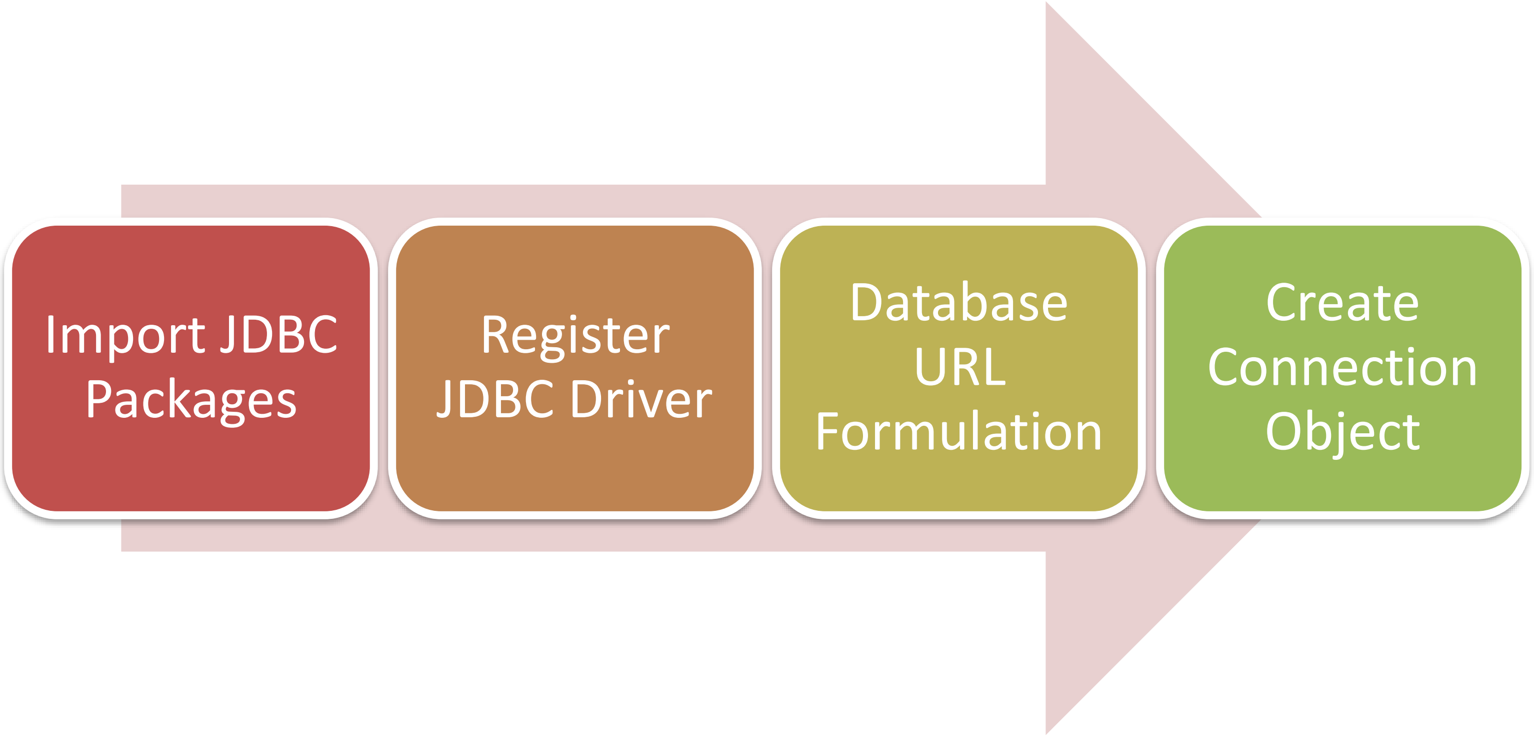
Type 4: 100% Pure Java

In a Type 4 driver, a pure Java-based driver communicates directly with the vendor's database through socket connection. This is the highest performance driver available for the database and is usually provided by the vendor itself.

This kind of driver is extremely flexible, you don't need to install special software on the client or server. Further, these drivers can be downloaded dynamically.



JDBC - Connections



JDBC - Connections

Import JDBC Packages

To use the standard JDBC package, which allows you to select, insert, update, and delete data in SQL tables, add the following imports to your source code −

import java.sql.* ; // for standard JDBC programs
import java.math.* ; // for BigDecimal and BigInteger support

JDBC - Connections

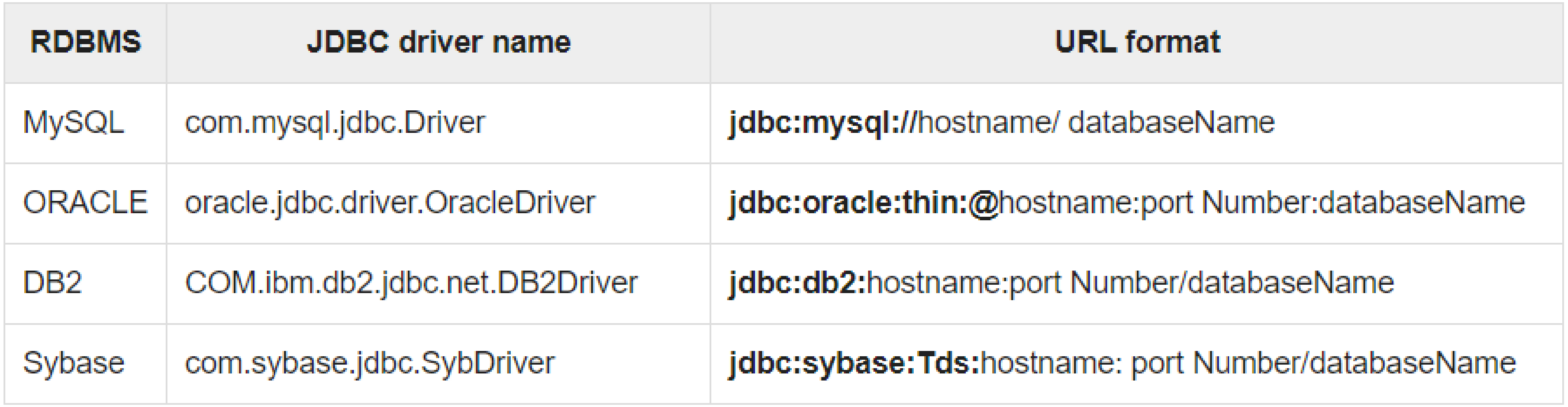
Register JDBC Driver

```
try {
   Driver myDriver = new oracle.jdbc.driver.OracleDriver();
   DriverManager.registerDriver( myDriver );
}
catch(ClassNotFoundException ex) {
   System.out.println("Error: unable to load driver class!");
   System.exit(1);
}
```

JDBC - Connections

Database URL Formulation

getConnection(String url)
getConnection(String url, Properties prop)
getConnection(String url, String user, String password)



RDBMS	JDBC driver name	URL format
MySQL	com.mysql.jdbc.Driver	**jdbc:mysql://**hostname/ databaseName
ORACLE	oracle.jdbc.driver.OracleDriver	**jdbc:oracle:thin:@**hostname:port Number:databaseName
DB2	COM.ibm.db2.jdbc.net.DB2Driver	**jdbc:db2:**hostname:port Number/databaseName
Sybase	com.sybase.jdbc.SybDriver	**jdbc:sybase:Tds:**hostname: port Number/databaseName

JDBC - Connections

Create Connection Object

```
String URL = "jdbc:oracle:thin:username/password@amrood:1521:EMP";
Connection conn = DriverManager.getConnection(URL);
```

Closing JDBC Connections

```
conn.close();
```

JDBC - Statements

Interfaces	Recommended Use
Statement	Use this for general-purpose access to your database. Useful when you are using static SQL statements at runtime. The Statement interface cannot accept parameters.
PreparedStatement	Use this when you plan to use the SQL statements many times. The PreparedStatement interface accepts input parameters at runtime.
CallableStatement	Use this when you want to access the database stored procedures. The CallableStatement interface can also accept runtime input parameters.

JDBC - Statements

The Statement Objects

```
Statement stmt = null;
try {
   stmt = conn.createStatement( );
   . . .
}
catch (SQLException e) {
   . . .
}
finally {
   stmt.close();
}
```

JDBC - Statements

The PreparedStatement Objects

```
PreparedStatement pstmt = null;
try {
   String SQL = "Update Employees SET age = ? WHERE id = ?";
   pstmt = conn.prepareStatement(SQL);
   . . .
}
catch (SQLException e) {
   . . .
}
finally {
   pstmt.close();
}
```

JDBC - Statements

The CallableStatement Objects

```
CallableStatement cstmt = null;
try {
   String SQL = "{call getEmpName (?, ?)}";
   cstmt = conn.prepareCall (SQL);
   . . .
}
catch (SQLException e) {
   . . .
}
finally {
   cstmt.close();
}
```

JDBC - Result Sets

The SQL statements that read data from a database query, return the data in a result set. The SELECT statement is the standard way to select rows from a database and view them in a result set. The java.sql.ResultSet interface represents the result set of a database query.

JDBC provides the following connection methods to create statements with desired ResultSet

```
createStatement(int RSType, int RSConcurrency);

prepareStatement(String SQL, int RSType, int RSConcurrency);

prepareCall(String sql, int RSType, int RSConcurrency);
```

JDBC - Result Sets

Type of ResultSet

Type	Description
ResultSet.TYPE_FORWARD_ONLY	The cursor can only move forward in the result set.
ResultSet.TYPE_SCROLL_INSENSITIVE	The cursor can scroll forward and backward, and the result set is not sensitive to changes made by others to the database that occur after the result set was created.
ResultSet.TYPE_SCROLL_SENSITIVE.	The cursor can scroll forward and backward, and the result set is sensitive to changes made by others to the database that occur after the result set was created.

JDBC - Result Sets

Concurrency of ResultSet

Type	Description
ResultSet.CONCUR_READ_ONLY	Creates a read-only result set. This is the default
ResultSet.CONCUR_UPDATABLE	Creates an updateable result set.

JDBC - Result Sets

Concurrency of ResultSet

```
try {
   Statement stmt = conn.createStatement(
                           ResultSet.TYPE_FORWARD_ONLY,
                           ResultSet.CONCUR_READ_ONLY);
}
catch(Exception ex) {
   ....
}
finally {
   ....
}
```

JDBC - Result Sets

Navigating a Result Set

S.N.	Methods & Description
1	**public void beforeFirst() throws SQLException** Moves the cursor just before the first row.
2	**public void afterLast() throws SQLException** Moves the cursor just after the last row.
3	**public boolean first() throws SQLException** Moves the cursor to the first row.
4	**public void last() throws SQLException** Moves the cursor to the last row.
5	**public boolean absolute(int row) throws SQLException** Moves the cursor to the specified row.

JDBC - Result Sets

Navigating a Result Set

S.N.	Methods & Description
6	**public boolean relative(int row) throws SQLException** Moves the cursor the given number of rows forward or backward, from where it is currently pointing.
7	**public boolean previous() throws SQLException** Moves the cursor to the previous row. This method returns false if the previous row is off the result set.
8	**public boolean next() throws SQLException** Moves the cursor to the next row. This method returns false if there are no more rows in the result set.
9	**public int getRow() throws SQLException** Returns the row number that the cursor is pointing to.
10	**public void moveToInsertRow() throws SQLException**Moves the cursor to a special row in the result set that can be used to insert a new row into the database. The current cursor location is remembered.
11	**public void moveToCurrentRow() throws SQLExceptionMoves** the cursor back to the current row if the cursor is currently at the insert row; otherwise, this method does nothing

JDBC - Result Sets

Viewing a Result Set

S.N.	Methods & Description
1	**public int getInt(String columnName) throws SQLException** Returns the int in the current row in the column named columnName.
2	**public int getInt(int columnIndex) throws SQLException** Returns the int in the current row in the specified column index. The column index starts at 1, meaning the first column of a row is 1, the second column of a row is 2, and so on.

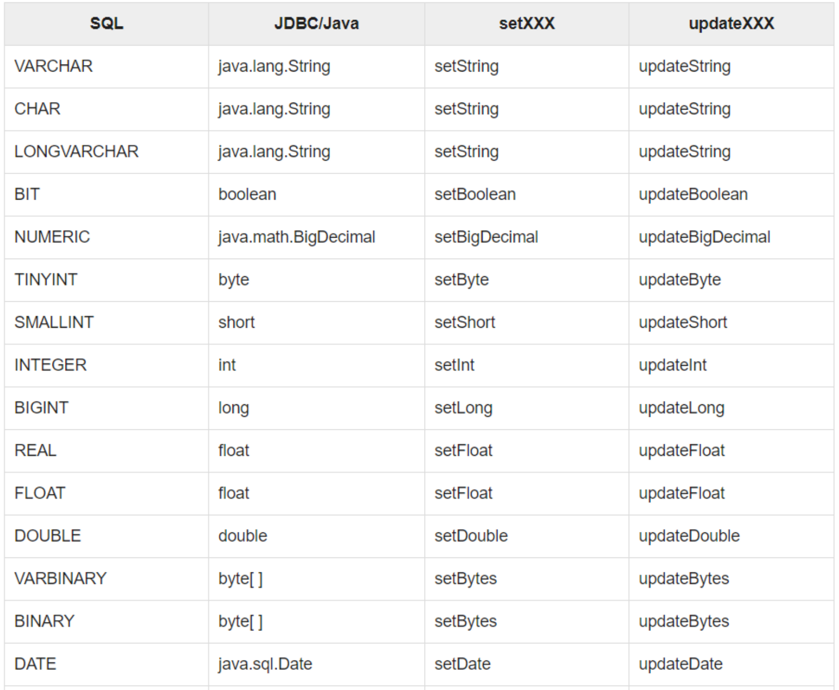
JDBC - Result Sets

Updating a Result Set

S.N.	Methods & Description
1	**public void updateString(int columnIndex, String s) throws SQLException** Changes the String in the specified column to the value of s.
2	**public void updateString(String columnName, String s) throws SQLException** Similar to the previous method, except that the column is specified by its name instead of its index.

JDBC - Data Types

The JDBC driver converts the Java data type to the appropriate JDBC type, before sending it to the database.



SQL	JDBC/Java	setXXX	updateXXX
VARCHAR	java.lang.String	setString	updateString
CHAR	java.lang.String	setString	updateString
LONGVARCHAR	java.lang.String	setString	updateString
BIT	boolean	setBoolean	updateBoolean
NUMERIC	java.math.BigDecimal	setBigDecimal	updateBigDecimal
TINYINT	byte	setByte	updateByte
SMALLINT	short	setShort	updateShort
INTEGER	int	setInt	updateInt
BIGINT	long	setLong	updateLong
REAL	float	setFloat	updateFloat
FLOAT	float	setFloat	updateFloat
DOUBLE	double	setDouble	updateDouble
VARBINARY	byte[ ]	setBytes	updateBytes
BINARY	byte[ ]	setBytes	updateBytes
DATE	java.sql.Date	setDate	updateDate

JDBC - Transactions

Transactions enable you to control if, and when, changes are applied to the database. It treats a single SQL statement or a group of SQL statements as one logical unit, and if any statement fails, the whole transaction fails.

To enable manual- transaction support instead of the auto-commit mode that the JDBC driver uses by default, use the Connection object's setAutoCommit() method. If you pass a boolean false to setAutoCommit( ), you turn off auto-commit. You can pass a boolean true to turn it back on again.

```
conn.setAutoCommit(false);

conn.rollback( );
```

JDBC - Transactions

Using Savepoints

The new JDBC 3.0 Savepoint interface gives you the additional transactional control. Most modern DBMS, support savepoints within their environments such as Oracle's PL/SQL.
When you set a savepoint you define a logical rollback point within a transaction. If an error occurs past a savepoint, you can use the rollback method to undo either all the changes or only the changes made after the savepoint.

setSavepoint(String savepointName): Defines a new savepoint. It also returns a Savepoint object.
releaseSavepoint(Savepoint savepointName): Deletes a savepoint. Notice that it requires a Savepoint object as a parameter. This object is usually a savepoint generated by the setSavepoint() method.

JDBC - Exceptions

An SQLException can occur both in the driver and the database. When such an exception occurs, an object of type SQLException will be passed to the catch clause. The passed SQLException object has the following methods available for retrieving additional information about the exception −

Method	Description
getErrorCode( )	Gets the error number associated with the exception.
getMessage( )	Gets the JDBC driver's error message for an error, handled by the driver or gets the Oracle error number and message for a database error.
getSQLState( )	Gets the XOPEN SQLstate string. For a JDBC driver error, no useful information is returned from this method. For a database error, the five-digit XOPEN SQLstate code is returned. This method can return null.
getNextException( )	Gets the next Exception object in the exception chain.
printStackTrace( )	Prints the current exception, or throwable, and it's backtrace to a standard error stream.
printStackTrace(PrintStream s)	Prints this throwable and its backtrace to the print stream you specify.
printStackTrace(PrintWriter w)	Prints this throwable and it's backtrace to the print writer you specify.

Thank you