Fundamentals of JDBC

Walkthrough of Databases

What is a Database

- A repository of data
- Has control and management routines to hold and manage data
- A database is organised into tables.
- Every table has a name and is organised into columns and rows
- Each column has a name and a data type
- Each row represents a record in the database

What is SQL?

- Structured Query Language is a standard way to represent database commands called “queries”
- They can be typed directly into any database engine that supports SQL for example Oracle, DB2 etc.
- There are basically 4 types of SQL commands

What is SQL?

- Structured Query Language is a standard way to represent database commands called “queries”
- They can be typed directly into any database engine that supports SQL for example Oracle, DB2 etc.
- There are basically 4 types of SQL commands
  - **DDL** – Create, Alter, Drop, Truncate

What is SQL?

- Structured Query Language is a standard way to represent database commands called “queries”
- They can be **typed** directly into any database engine that supports SQL for example Oracle, DB2 etc.
- There are basically 4 types of SQL commands
  - **DDL** – Create, Alter, Drop, Truncate
  - **DML** – Select, Insert, Update, Delete

What is SQL?

- Structured Query Language is a standard way to represent database commands called “queries”
- They can be typed directly into any database engine that supports SQL for example Oracle, DB2 etc.
- There are basically 4 types of SQL commands
  - **DDL** – Create, Alter, Drop, Truncate
  - **DML** – Select, Insert, Update, Delete
  - **TCL** – Commit, Rollback

What is SQL?

- Structured Query Language is a standard way to represent database commands called “queries”
- They can be typed directly into any database engine that supports SQL for example Oracle, DB2 etc.
- There are basically 4 types of SQL commands
  - DDL – Create, Alter, Drop, Truncate
  - DML – Select, Insert, Update, Delete
  - TCL – Commit, Rollback
  - DCL – Grant, Revoke

Enter JDBC

Why the need for JDBC

- All databases support SQL (ANSI SQL)
- Different database vendors have introduced their proprietary SQL constructs
- Different database vendors have introduced Application Programming Interfaces for accessing data stored in their respective databases
- Languages such as C++ can directly access these proprietary APIs
- If the database changes, all data access logic has to be entirely re-written
- A need was felt to access data from different databases in a consistent and reliable way

So what is JDBC

- It is not an acronym, but is called Java Database Connectivity
- It is a vendor independent API drafted by Sun to access data from different databases in a consistent and reliable way
- JDBC provides an API by hiding the vendor specific API by introducing the concept of a JDBC driver between the application and the database API
- Hence, JDBC requires a vendor specific driver
- The JDBC driver converts the JDBC API calls from the Java application to the vendor specific API calls

So what is JDBC

- If the database changes, the application can be configured to run with the new database by making very few changes in the code
- The database access in the application must be carefully designed to permit the almost transparent migration to the new database
- JDBC requires the database vendors to furnish runtime implementation of its interfaces

Main goals of JDBC

- JDBC should be an SQL level API
- JDBC should capitalize on the experience of the existing database APIs
- JDBC should provide a simple programming interface

What JDBC does not do?

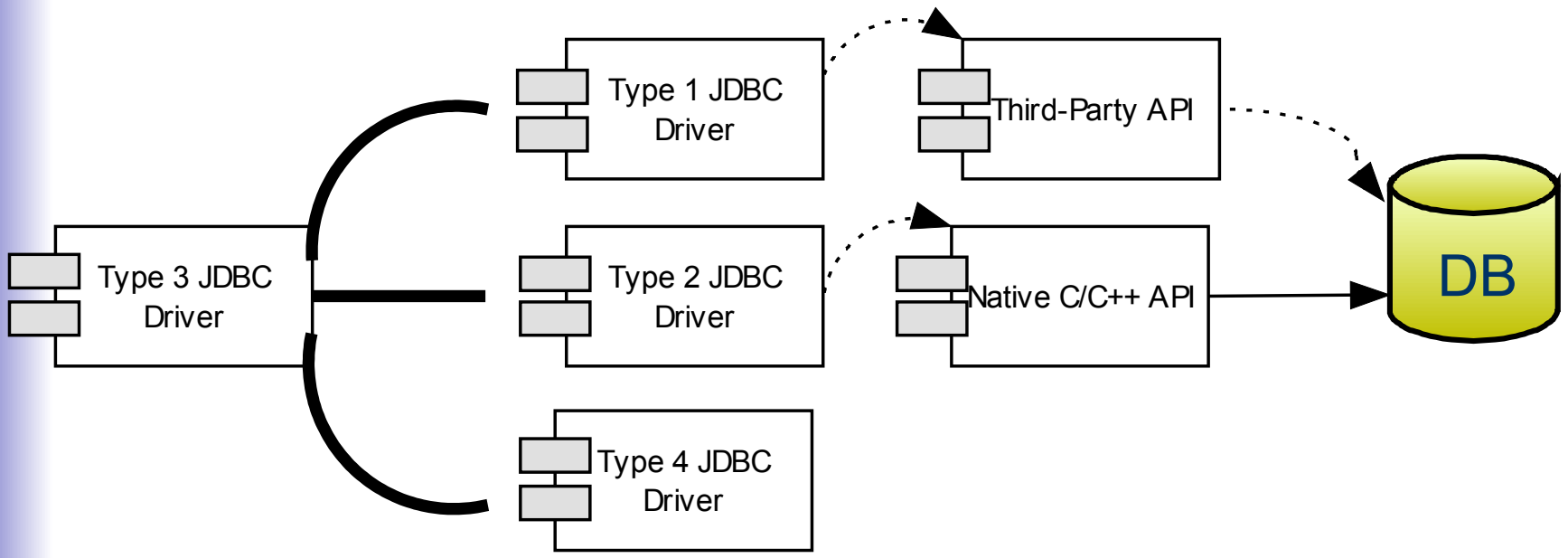
- The API does NOT standardize the SQL syntax
- It does not embed SQL inside it

JDBC Driver Types

There are 4 types of JDBC drivers

- Type 1 – JDBC – ODBC Bridge
- Type 2 – Native API – Partly Java Driver
- Type 3 – Java – Net Protocol Driver
- Type 4 – 100% Java Driver

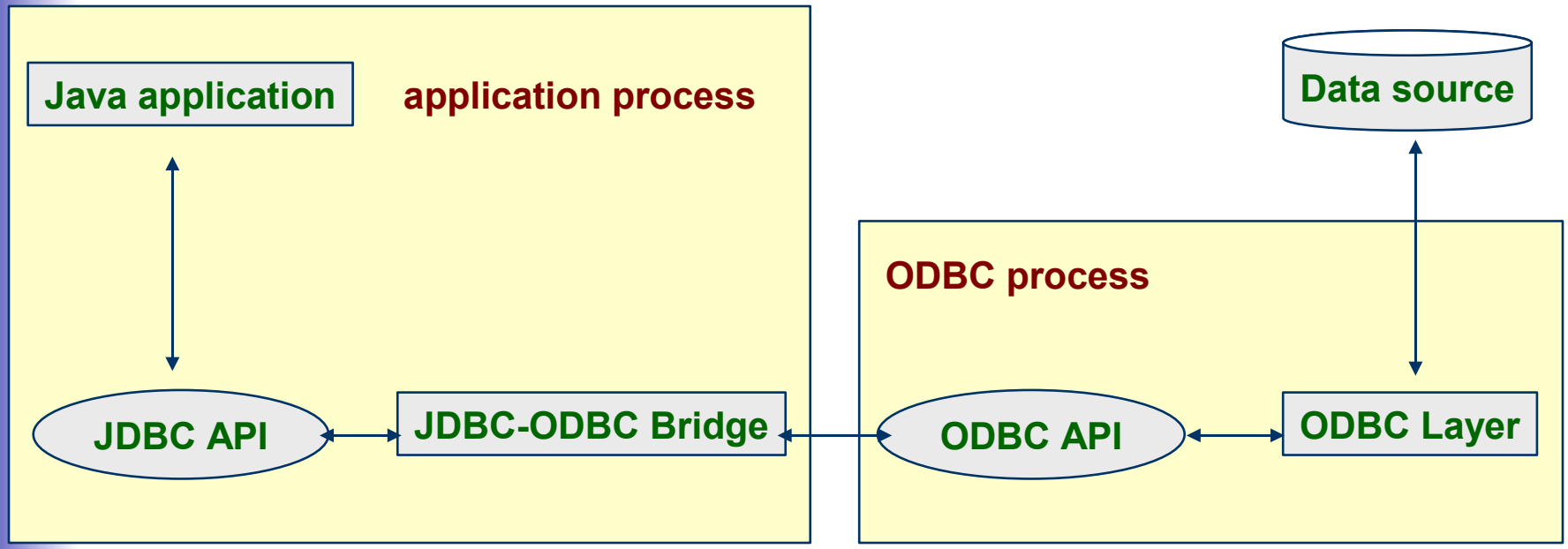
JDBC Driver Types



JDBC Type 1 Driver

- Translates all JDBC calls to ODBC (Open Database Connectivity) calls and sends them to the ODBC driver
- This requires the ODBC driver to be present in the client's machine
- Two stages of data type conversions occurs

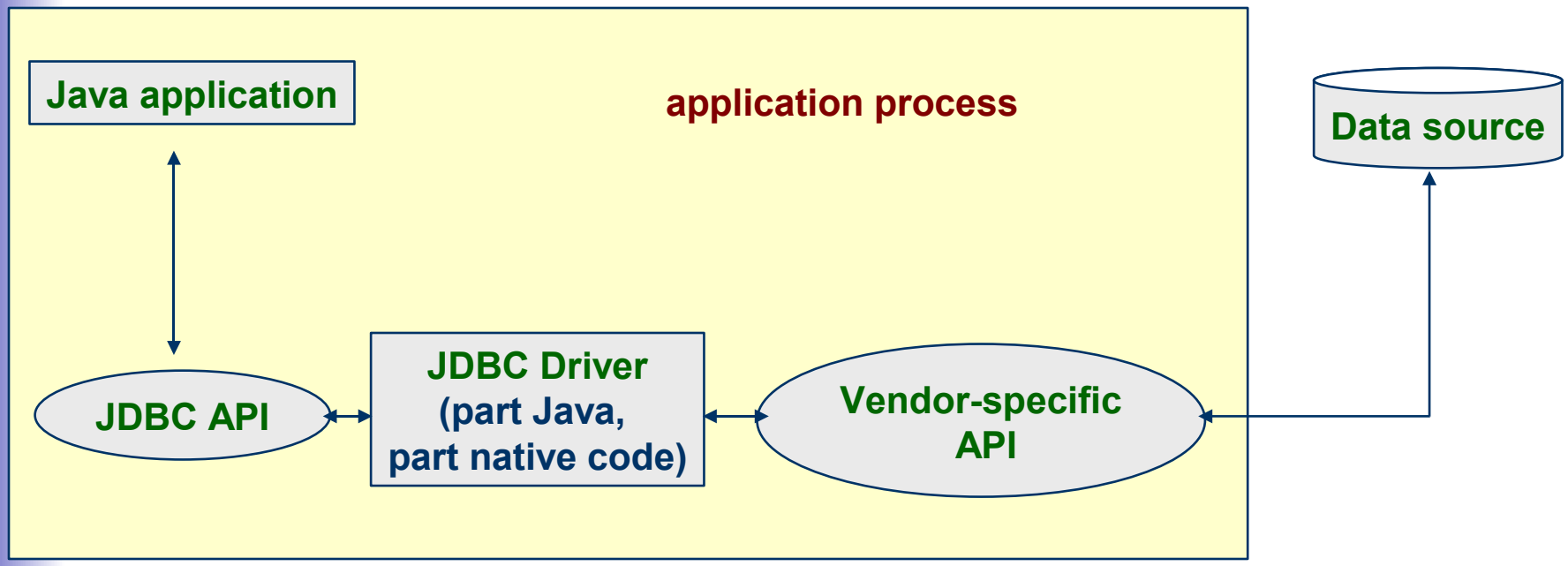
JDBC Type 1 Driver



JDBC Type 2 Driver

- Converts JDBC calls into database specific calls for databases such as Oracle, SQL Server etc.
- Communicates directly with the database server
- Requires some binary code modules to be present on the client machine
- More efficient that JDBC – ODBC bridge as there are fewer layers of communication and translation

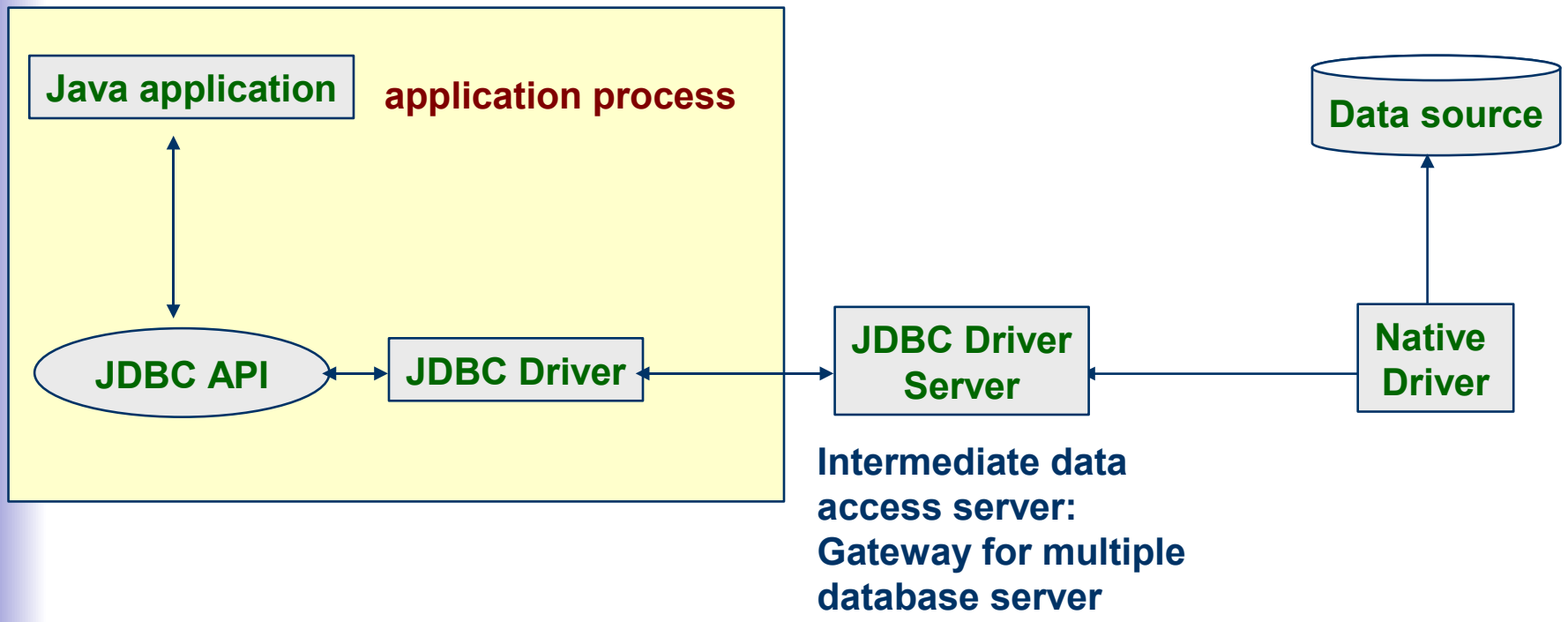
JDBC Type 2 Driver



JDBC Type 3 Driver

- Is a pure Java driver
- Follows a three tiered approach
- JDBC calls are passed through the network to the middle tier server
- The middle tier server then translates the requests to database specific native connectivity interface to further the request to the database server

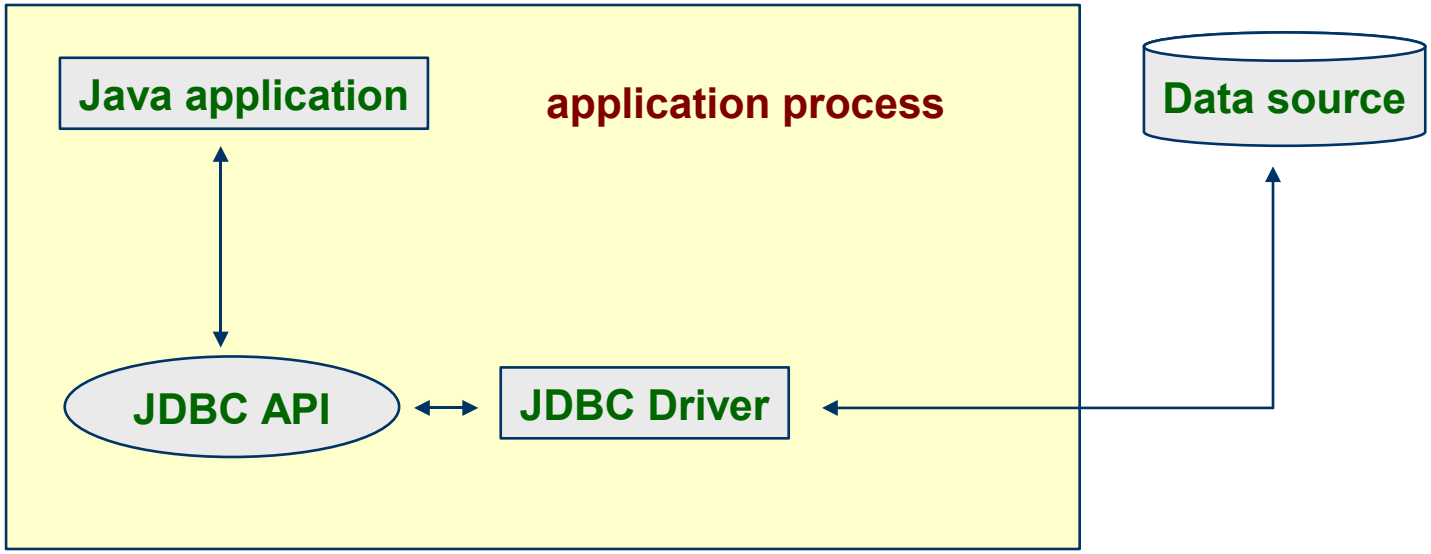
JDBC Type 3 Driver



JDBC Type 4 Driver

- Is a pure Java driver
- Converts JDBC calls to vendor specific database management system protocol
- Hence client applications can directly communicate with the database server
- Completely platform independent, hence are free of deployment and management issues

JDBC Type 4 Driver

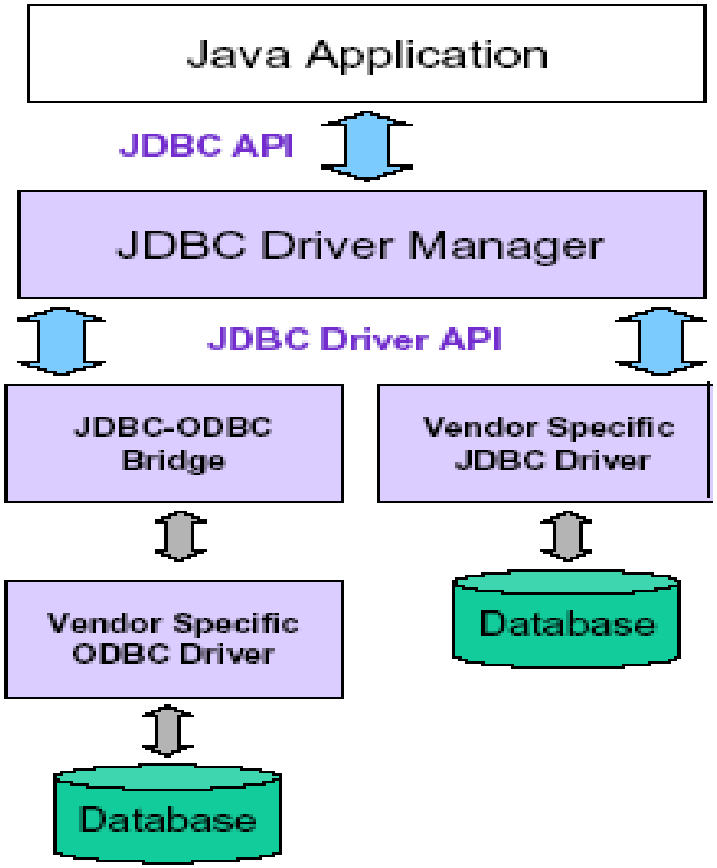


JDBC Architecture

The JDBC architecture mainly consists of two parts

- JDBC API, a purely Java based API
- JDBC Driver Manager

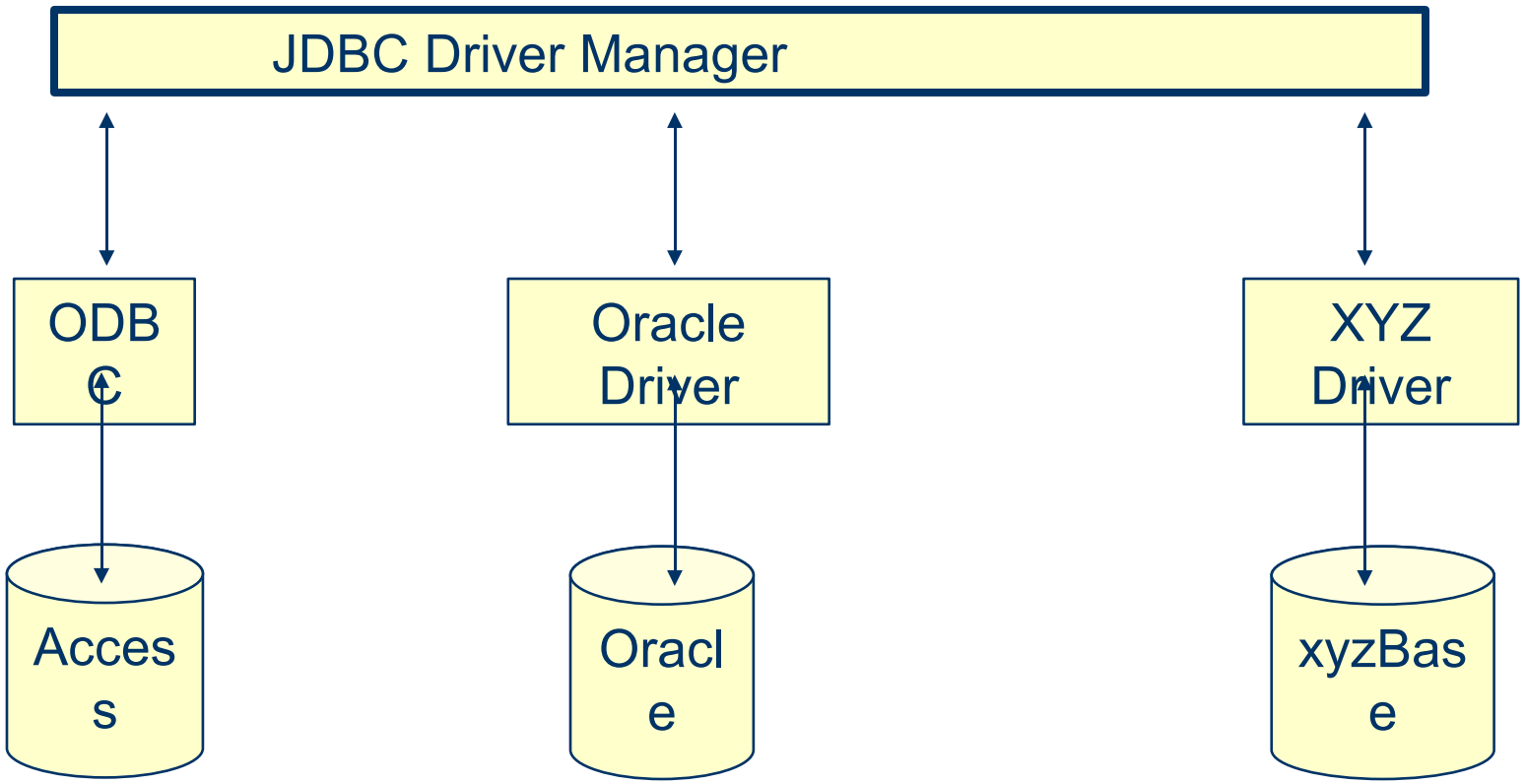
JDBC Architecture



The Driver Manager

- It is possible that an application may need to interact with multiple databases created by different vendors
- The JDBC Driver Manager provides the ability to communicate with multiple databases and keep track of which driver is needed for which database
- Even if you need to interact with one database, you need to do it via the driver manager

The Driver Manager



JDBC Types

JDBC Type	Java Type
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
FLOAT DOUBLE	double
BINARY VARBINARY LONGVARBINARY	byte[]
CHAR VARCHAR LONGVARCHAR	String

JDBC Type	Java Type
NUMERIC DECIMAL	BigDecimal
DATE	java.sql.Date
TIME TIMESTAMP	java.sql.Timestamp
CLOB	Clob*
BLOB	Blob*
ARRAY	Array*
DISTINCT	mapping of underlying type
STRUCT	Struct*
REF	Ref*
JAVA_OBJECT	underlying Java class

*SQL3 data type supported in JDBC 2.0

Programming with JDBC

Configuring JDBC Drivers

- You just need to make the JDBC driver classes available to the program you are writing

  To do this, there is only one step:
- Add the JDBC driver jar to CLASSPATH, or
- Use the –classpath option to make the JDBC driver classes available, or
- Add the JDBC driver jar file to the jre/ext directory

7 Basic steps in using JDBC

- Load the driver
- Define the connection URL
- Establish the database connection
- Create a statement object
- Execute query
- Process results
- Close database connection

1 – Loading the JDBC Driver

This stage involves the following steps:

- Load the JDBC driver class
- Create an instance for the JDBC driver class
- Register the instance with the driver manager
  - Set the jdbc.drivers property using –D option. If multiple drivers are there, separate them by colons (:)
  - java –Djdbc.drivers=org.postgressql.Driver MyJdbcProgram
  - Use System.setProperty to set the jdbc.drivers property
  - System.setProperty (“jdbc.drivers”, “org.postgressql.Driver”);
- jdbc.drivers property contains a list of drivers that the DriverManager will register at startup

1 – Loading the JDBC Driver

- Manually register using Class.forName()

```
try {
        Class.forName ("oracle.jdbc.driver.OracleDriver");
}
catch (ClassNotFoundException ex) {
        System.out.println ("Error while loading driver class" + ex);
}
```

- A static block in the class automatically creates the instance and registers it with the driver manager

2 – Define the connection URL

- This step basically specifies the location of the database server
- URLs are used to specify the location of the database server
- The URLs must conform to the jdbc: protocol
- The URL must define the server host, the port, and the database name
- The exact format of the JDBC URL is defined in the JDBC driver's documentation

```
String host = "phx-etc-dora.nac.travel.aexp.com";
//could be an IP address as well
String dbName = "TBMDEV";
int port = 1521;
String oracleURL = "jdbc:oracle:thin:@" + host + ":" + port + ":"
dbName;
```

3 – Establishing the connection

- Connection object represent a DB connection
- This step creates the actual network connection with the database server
- You need to pass the URL, user name, and password to the DriverManager's getConnection () method
- getConnection () throws SQLException

```
try {
        String userName = "POLICY";
        String password = "POLICY";
        Connection conn = DriverManager.getConnection (oracleURL,
userName, password);
}
catch (SQLException ex) {
        System.out.println ("Error while loading driver class" + ex);
}
```

3 – Establishing the connection

- An optional step is to retrieve the database information after the connection has been established

```
try {
        //Assume that the connection "conn" has already been
created
        DatabaseMetaData metaData = conn.getMetaData ();
        String databaseProductName =
metaData.getDatabaseProductName ();
        String databaseVersion =
metaData.getDatabaseProductVersion ();
}
catch (SQLException ex) {
        System.out.println ("Exception while retrieving meta-data");
}
```

4 – Creating a Statement

- Statement object is used to send queries to the database
- It can be retrieved from the Connection object

```
try {
        //Assume that Connection object “conn” has already been
created
        Statement stmt = conn.createStatement ();
}
catch (SQLException ex) {
        System.out.println (“Exception while retrieving meta-data”);
}
```

- The different types of statements will be discussed later

5 – Execute a query

- Once you have a statement object, you can start executing queries

```
try {
        //Assume that Connection object "conn" has already been created
        //Assume that Statement object "stmt" has already been created
        String sqlQuery = "SELECT FIRSTNAME, LASTNAME, EMPID FROM
EMPLOYEE";
        ResultSet rs = stmt.executeQuery (sqlQuery);
}
catch (SQLException ex) {
        System.out.println ("Exception while retrieving meta-data");
}
```

- To modify the database, once can use the executeUpdate () method of the Statement object
- Use the setQueryTimeOut to specify a maximum delay to wait for the results

6 – Process results

- The results of query execution are stored in a ResultSet object
- The standard practice is to process one row in the ResultSet at a time
- The ResultSet.next () method is used for such processing
- ResultSet supports getXXX () methods to retrieve column values. E.g. getString (), getInt ()
- getXXX () methods can be used with either column index or column name
- First column in a ResultSet has index 1, not 0

6 – Process results

```
try {
        //Assume Connection object “conn” has been created
        //Assume Statement object “stmt” has been created
        //Assume ResultSet object “resultSet” has been obtained
        while (resultSet.next ()) {
                String firstName = resultSet.getString (1);        //getString
(“FIRSTNAME”)
                String lastName = resultSet.getString (2);         //getString
(“LASTNAME”)
                int empID = resultSet.getInt (3);                  //getString
(“EMPID”)
        }
}
catch (SQLException sqle) {
        System.out.println (“Error processing records”);
}
```

7 – Closing the connection

- Remove the connection between the client and the database server
- To explicitly close the connection, use the close() method

```
try {
        conn.close ();
}
catch (SQLException sqle) {
        System.out.println ("Error occurred while closing the connection");
}
```

- Closing the connection can be deferred if more database operations are desired since the overheads involved in opening a connection are very large

JDBC Objects – A recap

- The following JDBC objects have been examined
  - Connection
  - Statement
  - ResultSet
- These are basically interfaces in the java.sql package

Using Statement

- The Statament object is used to execute SQL queries against the database
- There are 3 types of Statament objects
  - Statement
    - ➔For executing simple SQL statements
  - PreparedStatement
    - ➔For executing pre-compiled SQL statements
  - CallableStatament
    - ➔For executing database stored procedures

Using Statement methods

- executeQuery ()
  - Executes SQL query and returns data in a table (ResultSet)
  - The resulting table may be empty, but never null

    ResultSet rs = statement.executeQuery (“SELECT * FROM EMPLOYEE”);
- executeUpdate ()
  - Used to execute INSERT, UPDATE, and DELETE SQL statements
  - It returns an int that represents the number of rows that were affected in the table
  - Supports Data Definition Language (DDL) statements
    - ➔CREATE TABLE, DROP TABLE, ALTER TABLE

      int nRows = statement.executeUpdate (“DELETE FROM EMPLOYEE WHERE EMPID = 22”);

Using Statement methods

- execute ()
  - Generic method for executing stored procedures and prepared statements
  - Rarely used for returning multiple result sets
  - The statement execution may or may not return a ResultSet
  - If statement.getResultSet returns true, two or more result sets were produced

Using Statement methods

- getMaxRows/setMaxRows
  - Determines the maximum number of rows a ResultSet may contain
  - If not set, the number of rows is unlimited and the return value is 0
- getQueryTimeout/setQueryTimeout
  - Specifies the amount of time a driver will wait for a Statement to complete before throwing the SQLException

PreparedStatment (Pre-compiled Queries)

- The basic concept
  - An SQL query when executed against a database undergo a parsing and compilation stage
  - Once a query has been compiled, a query plan is created. These two stages are resource intensive
  - If similar queries are fired multiple times, there is no need for the compilation and parsing to happen multiple times
  - In such scenarios, PreparedStatements are very efficient

PreparedStatment (Pre-compiled Queries)

- The basic usage
  - Create a statement in standard form that is sent to the database for compilation before actually being used
  - Each time you use it, simply replace some of the marked parameters using the setXXX () methods
- PreparedStatement is an interface that inherits from Statement. The following methods are inherited
  - execute ()
  - executeQuery ()
  - executeUpdate ()

PreparedStatment Example

```
try {
        //Assume, the url, userName, and password strings have been
initialized
        Connection connection = DriverManager.getConnection (url,
userName, password);
        String sqlQuery = “UPDATE EMPLOYEE SET SALARY = ? WHERE
EMPID = ?”;
        PreparedStatement ps = connection.prepareStatement (sqlQuery);
        int[] newSalaries = getSalariesForAllEmployees();
        int[] employeeIDs = getAllIds ();

        for (int i = 0; i < employeeIDs.length; i++) {
                ps.setInt (1, newSalaries[i]);
                ps.setInt (2, employeeIDs[i]);
                ps.executeUpdate ();
        }
```

PreparedStatment Methods

- setXxx ()
  - Sets the indicated parameter (?) in the SQL statement to the passed in value
- clearParameters
  - Clears all set parameter values in the statement
  - execute ()
  - executeQuery ()
- executeUpdate ()

CallableStatment

- Stored procedures can greatly improve performance since they are pre-compiled units lying on the database server
- Usually involves making a choice between splitting business logic between the Java program and the database code
- Used to invoke stored procedures

```
try {
    CallableStatement cstmt = conn.prepareCall("{?=call
GETBOOKID(?)}");
    cstmt.registerOutParameter(1, java.sql.Types.INTEGER);
    cstmt.setString(2, "The Big Sleep");
    cstmt.execute();
}
catch (SQLException ex)
{…}
```

Transactions

- Basic idea
  - By default, after each SQL statement is executed, the changes are automatically committed to the database
  - Sometimes you handle a task/a group of tasks that need to be completed in an assured manner for any meaningful operation to be successful
  - Turn auto-commit off to group two or more statements together in a transaction
    - connetion.setAutoCommit (false);
  - Calling commit () permanently records the changes in the databse
  - Call rollback () if any error occurs