جامعة الأهرام الكندية
AHRAM CANADIAN UNIVERSITY

Lecture (06)
Database connectivity (I)

Dr. Ahmed ElShafee

Agenda

- Introduction
- Relational Database
- SQL
- Database Programming
- JDBC Overview
- Connecting DB

Introduction

- Enterprise Systems are Information systems that support many or all of the various parts of a firm.
- They can also refer to many mission-critical applications which are mainframe-based (also referred to as legacy systems).
- Also known as enterprise-wide information systems.
- Information systems that allow companies to integrate information across operations on a company-wide basis.

Introduction (cont,..)

Enterprise systems are broadly categorized into two:

1) Relational - Relational Database Management Systems (RDBMS)
2) Non-Relational
   a) Non Relational Databases (Legacy Database Systems)
   b) Legacy Systems (Older systems like old Cobol Applications)
   c) Enterprise Resource Planning
   d) Customer Relationship Management (CRM)
   e) Supply Chain Management

Introduction (cont,..)

Enterprise Systems Integration

- Enterprise Systems Integration is normally defined as the bringing together of:
  1) People,
  2) Processes and
  3) Information
- to work together in an harmonized way, and supported by appropriate information systems.
- It is also the bringing together of both old and new applications to achieve the overall goal of an organization.

Introduction (cont,..)

Reasons for Integration

- There are many reasons why Enterprise information systems need to be integrated.
- Some are stated below:
  1) need to persist and retrieve information from data repositories.
  2) need to leverage existing systems and resources while adopting and developing new technologies and architectures
  3) concern over the past years, that the mainframe was going away and that all legacy applications would be scrapped and completely rewritten.

Introduction (cont,..)

Java Integration Mechanisms

Java provides some technologies to integrate Enterprise systems:

- 1) Java Database connectivity (JDBC) for connecting applications to relational Database systems
- 2) JavaIDL and Java Connector Architecture (JCA) for connecting to Non- Relational systems
- The focus of this lecture is JDBC.

Relational Database

- Programming is all about data processing; data is central to everything you do with a computer.
- Databases, like filesystems are nothing more than specialized tools for data storage.
- Filesystems are good for storing and retrieving a single volume of information associated with a single virtual location.
- In other words, when you want to save a WordPerfect document, a filesystem allows you to associate it with a location in a directory tree for easy retrieval later.

Relational Database (cont,..)

- Databases provide applications with a more powerful data storage and retrieval system based on mathematical theories about data devised by Dr. E. F. Codd.
- Conceptually, a relational database can be pictured as a set of spreadsheets in which rows from one spreadsheet can be related to rows from another.
- Each spreadsheet in a database is called a table. As with a spreadsheet, a table is made up of rows and columns.
- A database engine is a process instance of the software accessing your database.
- For example Oracle, mySQL, Sybase etc

Relational Database (cont,..)

- Database engines use a standard query language to retrieve information from databases and is called Structured Query Language (SQL).

SQL

- SQL is not much like any programming language you might be familiar with.
- Instead, it is more of a structured English for talking to a database.
- Characteristics:
  1) SQL keywords are case-insensitive
  2) table and column names may or may not be case-insensitive depending on your database engine
  3) the space between words in a SQL statement is unimportant
  4) have a newline after each word, several spaces, or just a single space

SQL (cont,..)

With SQL you can ask the following question:

1) How do you get the data into the database?
2) And how do you get it out once it is in there?

Much of the simplest database access comes in the form of equally simple SQL statements.

Some of these commands are:

1) Create
2) Insert
3) Select
4) Update
5) Delete

SQL (cont,..)

Create

- SQL CREATE statement handles the creation of database entities. The major database engines provide GUI utilities that allow you to create tables without issuing any SQL.
- Uses:

1) To create database

Syntax:

```
CREATE DATABASE database_name
```


SQL (cont,..)

2) To create tables

Syntax:

```
CREATE TABLE table_name (
column_name column_type column_modifiers,
...,
column_name column_type column_modifiers)
```

SQL (cont,..)

```
create table license (id int, name varchar(40), sex varchar(10), date
    varchar(12), licenseType varchar(10));
```


SQL (cont,..)

Insert Statement

- With the tables in place, you use the INSERT statement to add data to them.
- Its form is:

```
INSERT INTO table_name(column_name, ..., column_name)
VALUES (value, ..., value)
```

- The first column name matches to the first value you specify, the second column name to the second value you specify, and so on for as many columns as you are inserting.
- If you fail to specify a value for a column that is marked as NOT NULL, you will get an error on insert.

SQL (cont,..)

```
insert into license (id, name, sex, date,licenseType)
values (123,"Ayman Mahmoud", "male", "14.03.2011", "Export");
```


SQL (cont,..)

- Update Statement
- The UPDATE statement enables you to modify data that you previously inserted into the database.
- Its form is:

```
UPDATE table_name
SET column_name = value,
...,
column_name = value
WHERE column_name = value
```

- This statement introduces the WHERE clause. It is used to help identify one or more rows in the database.

SQL (cont,..)

```
update license
set id=126,
name="Dina A. Mahmoud"
where id=124;
```


SQL (cont,..)

Select Statement

- The most common SQL command you will use is the SELECT statement.
- It allows you to select specific rows from the database based on search criteria.
- It takes the following form:

```
SELECT column_name, ..., column_name
FROM table_name
WHERE column_name = value
```

SQL (cont,..)

```
select * from license;
select * from license where id=126;
```


SQL (cont,..)

Delete Statement

- The DELETE command looks a lot like the UPDATE statement.
- Its syntax is:

```
DELETE FROM table_name WHERE column_name = value
```

- Instead of changing particular values in the row, DELETE removes the entire row from the table.

SQL (cont,..)

- Database programming has traditionally been a technological Tower of Babel.
- You are faced with dozens of available database products, and each one talks to your applications in its own private language.
- If your application needs to talk to a new database engine, you have to teach it (and yourself) a new language.
- As Java programmers, however, you should not worry about such translation issues.
- Java is supposed to bring you the ability to "write once, compile once, and run anywhere," so it should bring it to you with database programming, as well.

SQL (cont,..)

Stored Procedure

```
DELIMITER //
create procedure get_licenseType (in in_id int, out out_licenseType
    varchar(10) )
begin
select licenseType
into out_licenseType
from license
where id = in_id;
end //
DELIMITER ;
call get_licenseType(124,@licenseType);
select @licenseType;
drop procedure get_licenseType;
```

SQL (cont,..)

```
DELIMITER //
create procedure number_of_license ()
begin
select count(*) as number_of_license
from license;
end //
DELIMITER ;
call number_of_license();
drop procedure number_of_license;
```


JDBC

- JDBC API is a set of interfaces designed to insulate a database application developer from a specific database vendor.
- It enables the developer to concentrate on writing the application – making sure that queries to the database are correct and that the data is manipulated as designed.
- Sun developed a single API for database access—JDBC.
- Three main design goals:
  1) JDBC should be a SQL-level API.
  2) JDBC should capitalize on the experience of existing database APIs.
  3) JDBC should be simple.

JDBC (cont,..)

What does JDBC provide the developer?

1) the developer can write an application using the interface names and methods described in the API, regardless of how they were implemented in the driver
2) the developer writes an application using the interfaces described in the API as though they are actual class implementations
3) the driver vendor provides a class implementation of every interface in the API so that when an interface method is used, it is actually referring to an object instance of a class that implemented the interface.

JDBC (cont,..)

What do the driver vendors provide?

- Driver vendors provide implementations of JDBC interfaces.
- The JDBC API also enables developers to pass any string directly to the driver.
- This makes it possible for developers to make use of custom features of their database without requiring that the application use ANSI SQL

With JBDC you can :

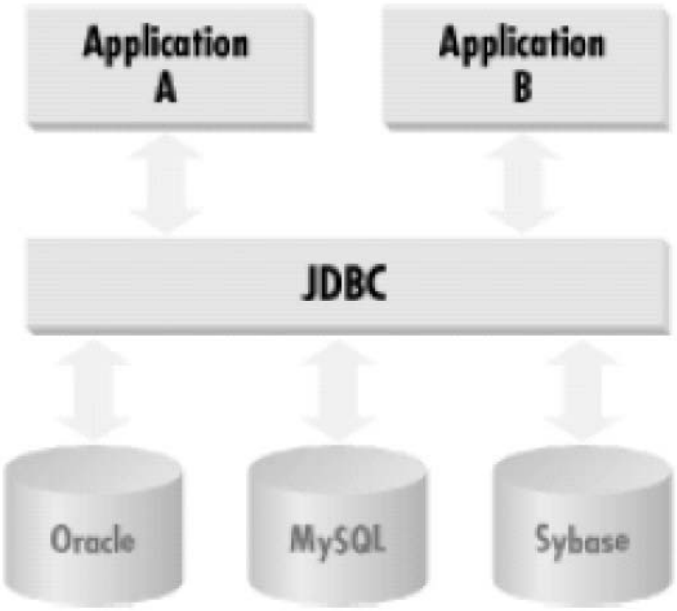
1) establish a connection with a database or access any tabular data source
2) send SQL statement
3) process the results

JDBC (cont,..)

- JDBC accomplishes its goals through a set of Java interfaces, each implemented differently by individual vendors.
- The set of classes that implement the JDBC interfaces for a particular database engine is called a JDBC driver.
- In building a database application, you do not have to think about the implementation of these underlying classes at all.
- The whole point of JDBC is to hide the specifics of each database and let you worry about just your application.

JDBC (cont,..)

JDBC Architecture



JDBC (cont,..)

JDBC Driver Type

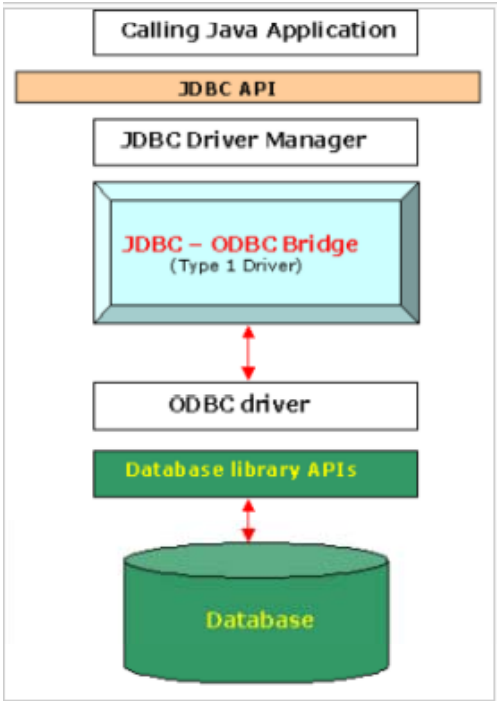
1) Type 1:JDBC-ODBC Bridge plus ODBC Driver
2) Type 2:A native API partially Java technology-enabled
3) Type 3:Pure Java Driver for Database Middleware
4) Type 4:Direct-to-Database Pure Java Driver

JDBC (cont,..)

1. JDBC-ODBC bridge,

is a database driver implementation that employs the ODBC driver to connect to the database.

The driver converts JDBC method calls into ODBC function calls



JDBC (cont,..)

Functions

- Translates query obtained by JDBC into corresponding ODBC query, which is then handled by the ODBC driver.
- Sun provides a JDBC-ODBC Bridge driver. sun.jdbc.odbc.JdbcOdbcDriver. This driver is native code and not Java, and is closed source.
- Client -> JDBC Driver -> ODBC Driver -> Database

Advantages

- Easy to connect.

JDBC (cont,..)

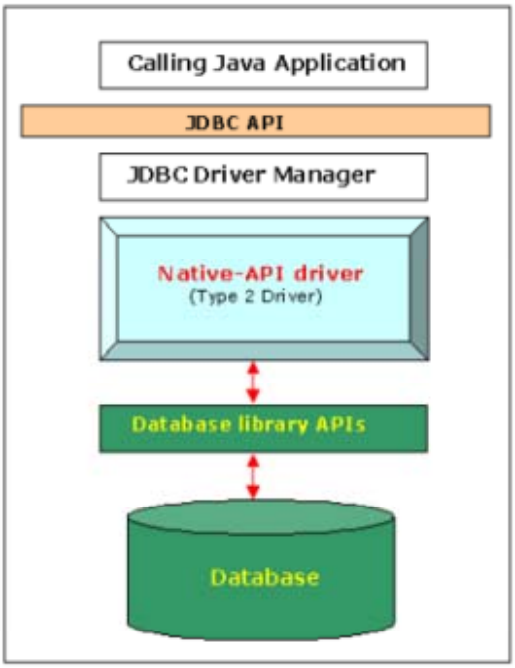
Disadvantages

- Performance overhead since the calls have to go through the jdbc Overhead bridge to the ODBC driver, then to the native db connectivity interface.
- The ODBC driver needs to be installed on the client machine.
- Considering the client-side software needed, this is not suitable for applets.
- Compared to other driver types it's slow.
- The Sun driver has a known issue with character encodings and Microsoft Access databases. Microsoft Access may use an encoding that is not correctly translated by the driver, leading to the replacement in strings of, for example, accented characters by question marks.

JDBC (cont,..)

2. Native-API driver,

- is a database driver implementation that uses the client-side libraries of the database.
- The driver converts JDBC method calls into native calls of the database API.



JDBC (cont,..)

- The type 2 driver is not written entirely in Java as it interfaces with non-Java code that makes the final database calls.
- The driver is compiled for use with the particular operating system.
- For platform interoperability, the Type 4 driver, being a full-Java implementation, is preferred over this driver.

Disadvantages

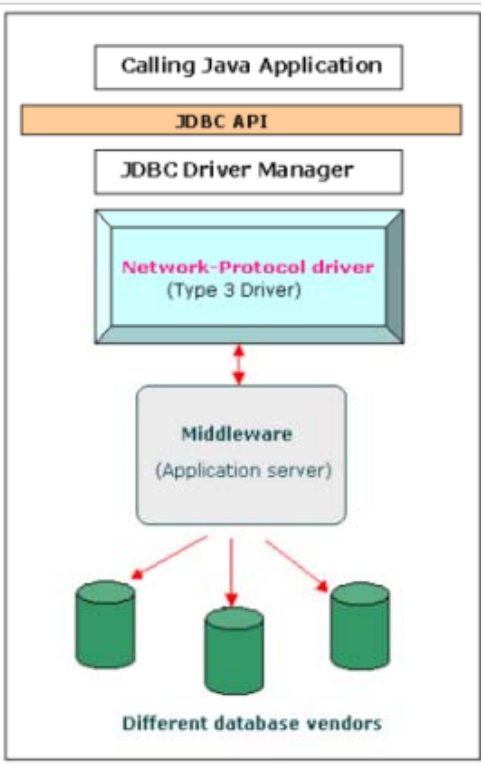
- The vendor client library needs to be installed on the client machine.
- Not all databases have a client side library
- This driver is platform dependent
- This driver supports all java applications except Applets

JDBC (cont,..)

3. Pure Java Driver for Database Middleware,

is a database driver implementation which makes use of a middle tier between the calling program and the database.

The middle-tier (application server) converts JDBC calls directly or indirectly into the vendor-specific database protocol.



JDBC (cont,..)

- This differs from the type 4 driver in that the protocol conversion logic resides not at the client, but in the middle-tier.
- Like type 4 drivers, the type 3 driver is written entirely in Java. The same driver can be used for multiple databases.
- It depends on the number of databases the middleware has been configured to support.
- The type 3 driver is platform-independent as the platform-related differences are taken care by the middleware.
- Also, making use of the middleware provides additional advantages of security and firewall access.

JDBC (cont,..)

Functions

Follows a three tier communication approach.

- Can interface to multiple databases - Not vendor specific.
- The JDBC Client driver written in java, communicates with a middleware-net-server using a database independent protocol, and then this net server translates this request into database commands for that database.
- Thus the client driver to middleware communication is database independent.
- Client -> JDBC Driver -> Network-protocol driver -> Middleware-Net Server -> Any Database,...

JDBC (cont,..)

Advantages

- Since the communication between client and the middleware server is database independent, there is no need for the vendor db library on the client machine.
- Also the client to middleware need not be changed for a new database.
- The Middleware Server (which can be a full fledged J2EE Application server) can provide typical middleware services like caching (connections, query results, and so on), load balancing, logging, auditing etc.
- Can be used in internet since there is no client side software needed.

JDBC (cont,..)

- At client side a single driver can handle any database. (It works provided the middleware supports that database!)

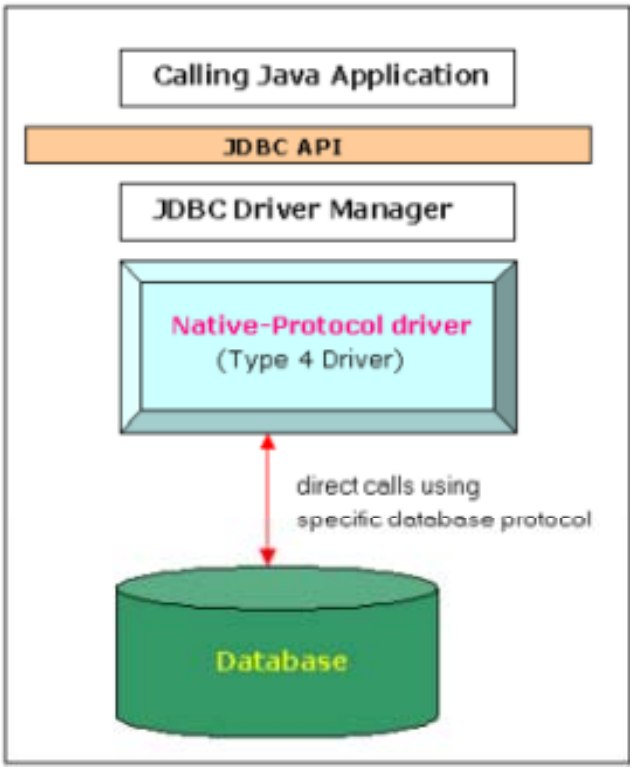
Disadvantages

- Requires database-specific coding to be done in the middle tier.
- An extra layer added may result in a time-bottleneck. But typically this is overcome by providing efficient middleware services .

JDBC (cont,..)

4. The JDBC type 4 driver, also known as the Direct to Database Pure Java Driver,

is a database driver implementation that converts JDBC calls directly into a vendor-specific database protocol.



JDBC (cont,..)

- Written completely in Java, type 4 drivers are thus platform independent.
- They install inside the Java Virtual Machine of the client.
- This provides better performance than the type 1 and type 2 drivers as it does not have the overhead of conversion of calls into ODBC or database API calls.
- Unlike the type 3 drivers, it does not need associated software to work.
- As the database protocol is vendor-specific, the JDBC client requires separate drivers, usually vendor-supplied, to connect to different types of databases.

JDBC (cont,..)

Functions

- Type 4 drivers, coded entirely in Java, communicate directly with a vendor's database, usually through socket connections. No translation or middleware layers are required, improving performance.
- The driver converts JDBC calls into the vendor-specific database protocol so that client applications can communicate directly with the database server.
- Completely implemented in Java to achieve platform independence.

JDBC (cont,..)

Advantages

- These drivers don't translate the requests into an intermediary format (such as ODBC), nor do they need a middleware layer to service requests. This can enhance performance considerably.
- The JVM can manage all aspects of the application-to-database connection; this can facilitate debugging.

JDBC (cont,..)

Disadvantages

- Drivers are database dependent.

JDBC (cont,..)

- This type includes (for example) the widely-used Oracle thin driver - oracle.jdbc.driver.OracleDriver which connects using a format configuration of jdbc:oracle:thin:@URL
- Client -> Native-protocol JDBC Driver -> database server

Connecting DB

- The JDBC Connection process is the most difficult part of JDBC to get right.
- There are generally two basic connection problems:

1) Connection fails with the message "Class not found“

Solution: Set your JDBC driver in your CLASSPATH

2) Connection fails with the message "Driver not found"

Solution: register the JDBC driver with the DriverManager class

Connecting DB (cont,..)

Connection steps:

1. Load driver

```
Class.forName ("com.mysql.jdbc.Driver").newInstance ();
```

2. Create connection

```
Connection conn = null;
String url = "jdbc:mysql://localhost/testDB";
String userName = "root";
String password = "root";
conn = DriverManager.getConnection (url, userName, password);
```


Connect01.java

```
String userName = "root";
        String password = "root";
        String url = "jdbc:mysql://localhost/testDB";
        Class.forName ("com.mysql.jdbc.Driver").newInstance ();
        conn = DriverManager.getConnection (url, userName, password);
        System.out.println ("Database connection established");
```

Thanks,

See you next Week, isA