What is JDBC?

- Introduction
- Goals
- Architecture

Introduction

- API
- Describes standard Java lib for accessing data sources
  - Primarily relational databases (db)
  - Using Structured Query Language (SQL)
- Support corp dev’s req for db access from enterprise Java app
SQL

• ANSI standard language
• Used to manage relational dbs
• But, vendors implement unique versions
  • Take advantage of db-specific features

For universal db access

• Standard spec’s for db access drivers ‘
  • “bridges”
• ==> Allow developers build app’s
  • Access, view, modify data
  • From multiple diverse db
• Port new/existing app’s to new data sources
Bridges

- Act as translators
- Take generic db access calls
- Translate to db-specific calls
- Most common drive spec
  - ODBC (Open DataBase Connectivity)
  ...

ODBC

- C-based standard
- Developed by SQL Access Group (SAG) standards committee
- Popularized by Microsoft
- Based on Call-Level Interface (CLI) spec’s of SAG
ODBC, cont.

- Allow users to access data in rel db mngmnt sys
- Provide consistent interface for comm w/ db
- Several drawbacks
  - Slow
  - Limited functionality
  - Drivers are platform specific
  - ==> Need separate driver for ea platform
  - ==> Less portable app’s

ODBC notes

  - Rep’s from many vendors
- More info: www.microsoft.com/data/odbc
JDBC/ODBC bridge

- Also use drivers to create bridges
  - Drivers being developed for many db’s
  - ODBC more mature \implies more drivers
  - Can use JDBC/ODBC bridge (JavaSoft) as db driver
    - Till native driver is available

JDBC/ODBC bridge cont.

- JDBC/ODBC bridge allows use of native Java interface to access db via ODBC wo learning ODBC spec’s.
- But get up and running while vendors develop JDBC drivers
  - Works w/ most db’s
  - Free from JavaSoft
  - Some drawbacks ...
JDBC/ODBC bridge drawbacks

- Degraded performance
- ODBC not as rich as JDBC
- Can’t take full advantage of all JDBC’s capabilities

JDBC drivers

- Drivers list: java.sun.com/products/jdbc/jdbc.drivers.html
Goals for JDBC

- Modeled after ODBC architecture
- Two high-level goals
  - Support common db standards
  - Keep the API simple

Support common db standards

- API must be generic
- Replace functionality of ODBC in C app’s
- Provide interaction w/SQL db’s
- Goals met in three ways ...
Support common standards cont.

- Pass raw SQL statements directly to data source
- Compliant requirement
  - Drivers must at least support ANSI SQL92 entry level standard
- Architecture must support functionality of common db bridges
  - E.g., ODBC

Pass raw SQL statements

- Allow app programmer take advantage of db-specific features
- ==> Has to send appropriate SQL commands
- Drivers provide metadata -- describe
  - Valid commands for data source
  - Data tables, attribute name, type, size
- JDBC: methods to return this info from driver
Support ANSI SQL92

- “JDBC compliant” (Sun trademark) =>
- Minimum level of SQL support =>
- Minimum level of interoperability

Support functionality of common db bridges

- E.g., ODBC
  - Provide superset of ODBC capabilities
  - Improve on ODBC standard
  - Optimized to leverage strengths of Java
  - Use JDBC for all db access calls
  - JDBC/ODBC bridge => take advantage of JDBC w/ any data source
Keep it simple

- Compromises, e.g., SQL vs. Java
  - Different languages, design, use
- API should work well with rest of JavaSoft Java class libraries
- Simplicity guidelines ...

Simplicity guidelines

- Keep common cases simple ...
- Leverage style & benefits of core classes ...
- Create methods that map specific functionality ...
- Use strong, static typing wherever possible ...
Keep common cases simple

- Specific APIs for standard activities
  - E.g., Selecting data using parameters
- Specific methods for common cases
  ==> 
- App programmers write & maintain less code

Leverage style and benefits of core classes

- of Java core classes as much as possible
- API can be easily integrated into app’s
- JDBC 2.0: work with
  - JavaBeans component model
  - Java Transaction Services (JTS)
  - Other Java technologies
Create methods that map specific functionality

- Create rich API
  - Address most db tasks w/ specific method calls
  - ==> Easier for new JDBC prog to understand use of methods
- Alternative
  - Small API ...

Small API alternative

- Few methods
  - Each taking several parameters
    - Allow prog to configure method
  - ==> Multiple meanings based on parameter values
- E.g., Execute method for all types of SQL calls
  - Parameter specifies if return result expected
Small API, cont.

- Easier for programmer to learn all available methods
- But harder to learn how to use each method
- JDBC 2.0:
  - Significantly larger API
  - Methods simple to understand & use

Use strong, static typing wherever possible

- ==> Provide compile-time checking
- ==> Catch more errors at compile time
  - Rather than run time
- But SQL: dynamic language, ==> ...
**SQL: dynamic language**

- Raw SQL passed to db & interpreted at run time
- ==> Data types determined as result of query at run time
- ==> Must monitor carefully warnings & exceptions

**JDBC feature**

- Can quickly develop access strategies for app’s
- With very small amount of code
  - Create connection to db
  - Query db
  - Update values
- JDBC supports transaction model ...
Transaction model

- Can make several db modifications
- Undo all in single transaction (if necessary)

JDBC architecture

- Basic architecture
- Typical scenarios
Basic architecture

- Simple
- DriverManager class
  - Provides service for managing set of JDBC drivers
  - Attempts to load driver classes ref’d in jdbc.drivers system property
  - Can load drivers explicitly using Class.forName()
Notes

- `Class.forName()` loads driver class
- On load, driver registers w/ `DriverManager`
- `DriverManager.getConnection()`
  - look for registered driver
    - can handle data source described by URL using `jdbc:` protocol
  - return object from driver
    - implements connection interface

Notes, cont.

- `Connection`
  - represents session w/ data source
  - includes methods for executing db op’s
  - provides info about data source
    - meta-data
      - includes info about data source structure
Notes, cont.

- Protocol used to specify data source
  - format jdbc:subprotocol:datasourcename
    - datasource: name of registered ODBC data source
      - e.g., MS Access db / Oracle db
  - getConnection() method takes optional username & password
    - For registration w/data sources that require such

Typical scenarios

- Differ based on
  - Location of db
  - Driver
  - Application
  - Communication protocols used
Typical scenarios, cont.

• Standalone applications
• Applets communicating w/ Web server
• App & applets communicating w/ db via JDBC/ODBC gateway
• App’s accessing remote resources using mechanism, e.g.,
  • RMI

Typical scenarios, cont.

• Grouped into 2 architectures
  • 2-tier ...
  • 3-tier ...
Two-tier

- App resides on same machine as db driver
  - Driver access db running on db server
  - db driver responsible for handling networked communications

Two-tier example

- Java app running on client machine using JDBC local driver
  - Driver uses vendor-specific client lib
    - Access db remotely
    - Java app accesses resource transparently
  - Never deals w/ net comm issues
Two-tier note

- db may reside on same machine as app and still considered 2-tier architecture because app resides in separate address space from db
- driver must handle inter-process comm.

Three-tier architecture

- app / applet running on one platform accessing db driver on another
- db driver accessed through
  - Applet via Web server
  - App access remote server program that communicates locally w/ db driver
  - App communicate w/ app server that accesses db
Three-tier example notes

- Applet on client Web browser --> server app --> db behind firewall
- All db access logic in JDBC driver
- ==> Driver vendor handles issues of comm w/db & db vendor client lib
- ==> Can write app’s that function in 2-tier or 3-tier env
  - W/ few or no changes to code
- JDBC design ==> access relational db wo extra skill
Summary

- Standard mechanism & API
- Java programs to access
  - Relational databases
  - Other data sources

JDBC programs can

- Send queries & updates
- Ask about info re
  - Data source
  - Data in it
Access to specific databases

- Managed by drivers
  - Implement common set of Java interfaces
- ==> db connectivity provider & driver programmer
  - Use mechanisms needed to create high-performance db connection
  - Change db wo changing much code