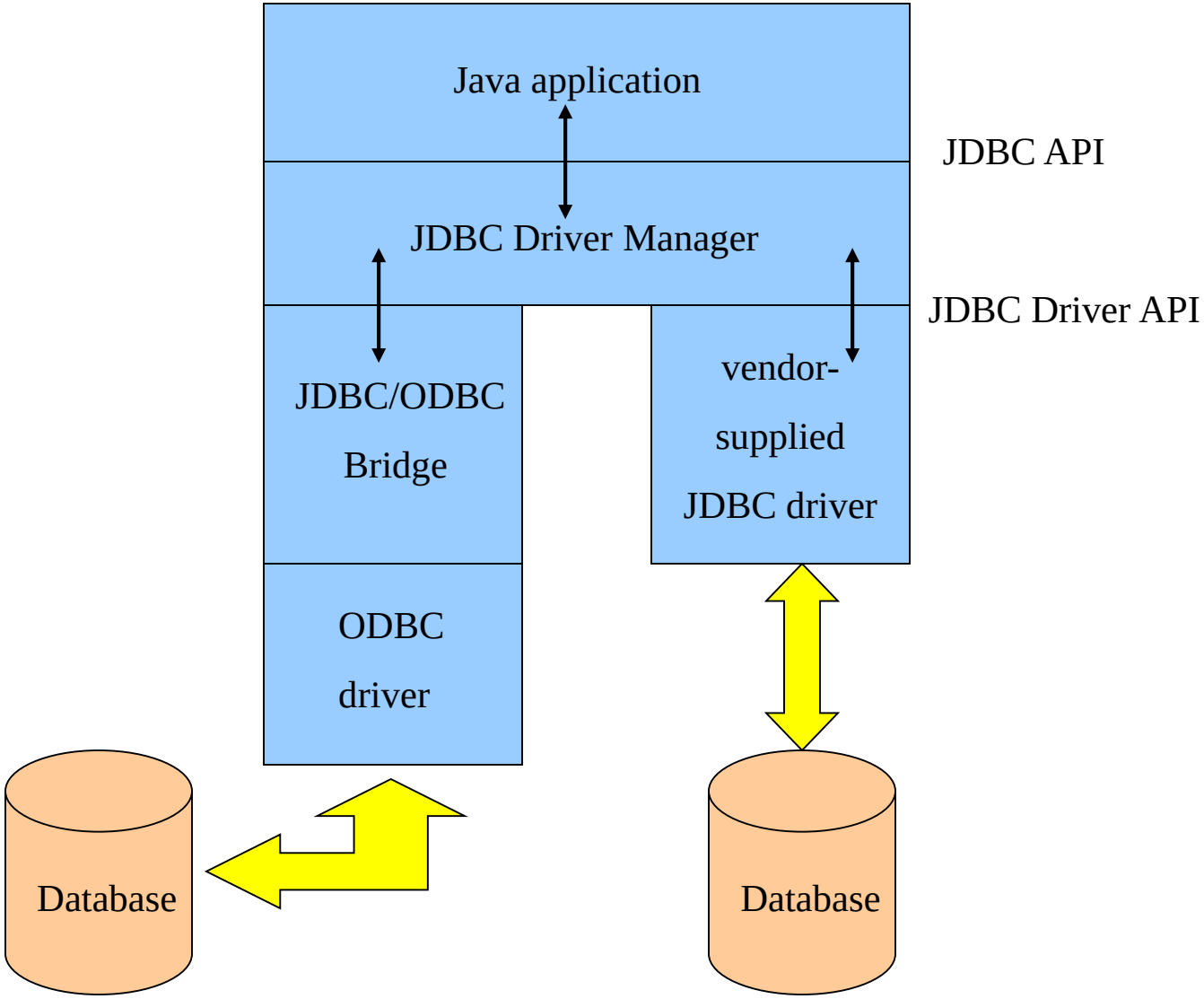
Java Database Connectivity (JDBC)

Introduction

- Database
  - Collection of data
- DBMS
  - Database management system
  - Storing and organizing data
- SQL
  - Relational database
  - Structured Query Language
- JDBC
  - Java Database Connectivity
  - JDBC driver

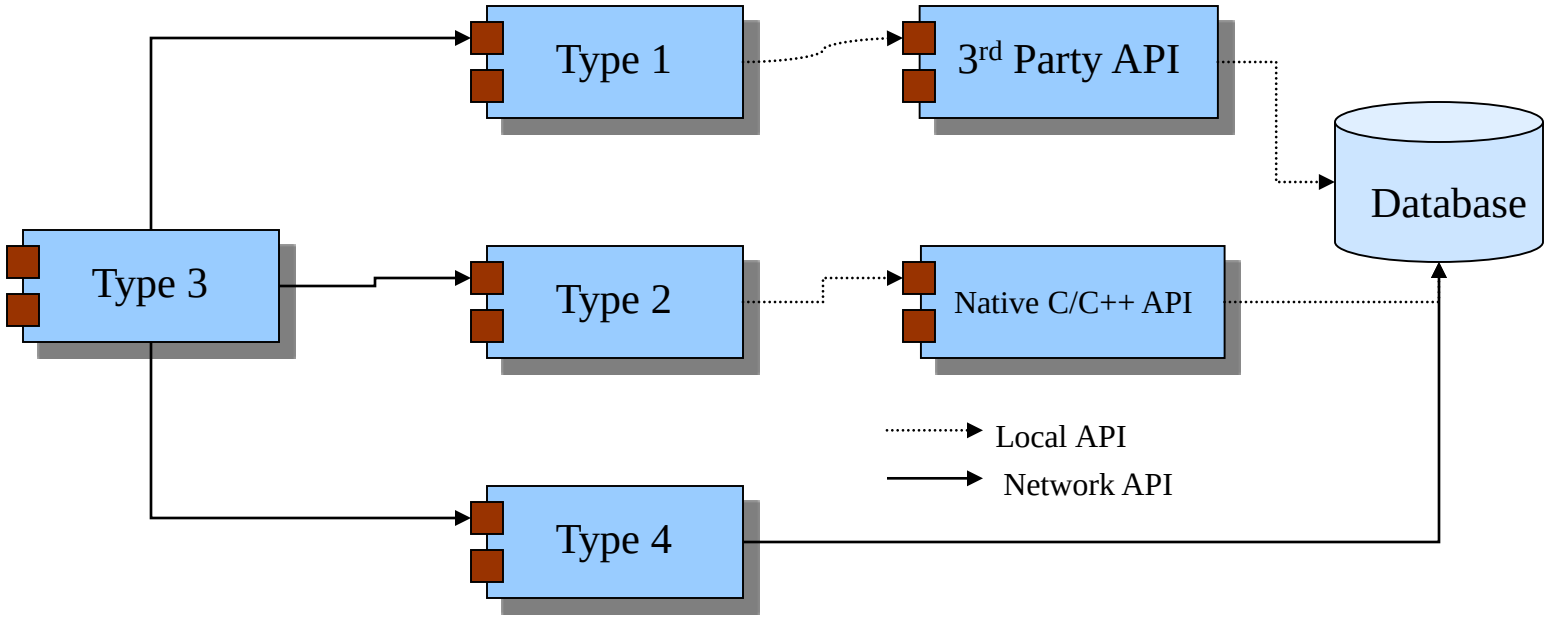
JDBC

- Programs developed with Java/JDBC are platform and vendor independent.
- “write once, compile once, run anywhere”
- Write apps in java to access any DB, using standard SQL statements – while still following Java conventions.
- JDBC driver manager and JDBC drivers provide the bridge between the database and java worlds.


Java application
JDBC API
JDBC Driver Manager
JDBC Driver API
JDBC/ODBC
Bridge
vendor-
supplied
JDBC driver
ODBC
driver
Database
Database

ODBC

- JDBC heavily influenced by ODBC
- ODBC provides a C interface for database access on Windows environment.
- ODBC has a few commands with lots of complex options. Java prefers simple methods but lots of them.



• Type 1: Uses a bridging technology to access a database. JDBC-ODBC bridge is an example. It provides a gateway to the ODBC.

• Type 2: Native API drivers. Driver contains Java code that calls native C/C++ methods provided by the database vendors.

• Type 3: Generic network API that is then translated into database-specific access at the server level. The JDBC driver on the client uses sockets to call a middleware application on the server that translates the client requests into an API specific to the desired driver. Extremely flexible.

• Type 4: Using network protocols built into the database engine talk directly to the database using Java sockets. Almost always comes only from database vendors.

Relational-Database Model

- Relational database
  - Table
  - Record
  - Field, column
  - Primary key
    - Unique data
- SQL statement
  - Query
  - Record sets

Structured Query Language (SQL)

- SQL overview
- SQL keywords

SQL keyword	Description
SELECT	Select (retrieve) fields from one or more tables.
FROM	Tables from which to get fields. Required in every **SELECT**.
WHERE	Criteria for selection that determine the rows to be retrieved.
GROUP BY	Criteria for grouping records.
ORDER BY	Criteria for ordering records.
INSERT INTO	Insert data into a specified table.
UPDATE	Update data in a specified table.
DELETE FROM	Delete data from a specified table.

Fig. 8.12 SQL query keywords.

Basic SELECT Query

- Simplest format of a SELECT query
  - **SELECT** * **FROM** tableName
    - **SELECT** * **FROM** **authors**
- Select specific fields from a table
  - **SELECT** **authorID**, **lastName** **FROM** **authors**

authorID	lastName		
1	Deitel		
2	Deitel		
3	Nieto		
4	Santry		

Fig. 8.13 **authorID** and **lastName** from the **authors** table.

WHERE Clause

- specify the selection criteria
  - **SELECT** fieldName1, fieldName2, … **FROM** tableName **WHERE** criteria
    - **SELECT** title, editionNumber, copyright

      FROM titles

      WHERE copyright > 1999
- **WHERE** clause condition operators
  - <, >, <=, >=, =, <>
  - **LIKE**
    - wildcard characters **%** and **_**

WHERE Clause (Cont.)

- **SELECT** authorID, firstName, lastName
  FROM authors
  WHERE lastName **LIKE** 'D%'

authorID	firstName	lastName
1	Harvey	Deitel
2	Paul	Deitel

Fig. 8.15 Authors whose last name starts with `D` from the `authors` table.

WHERE Clause (Cont.)

- **SELECT** authorID, firstName, lastName

  FROM authors

  WHERE lastName **LIKE** '_i%'

authorID	firstName	lastName
3	Tem	Nieto

Fig. 8.16 The only author from the **authors** table whose last name contains **i** as the second letter.

ORDER BY Clause

- Optional **ORDER BY** clause
  - **SELECT** fieldName1, fieldName2, … **FROM** tableName **ORDER BY** field **ASC**
  - **SELECT** fieldName1, fieldName2, … **FROM** tableName **ORDER BY** field **DESC**
- **ORDER BY** multiple fields
  - **ORDER BY** field1 sortingOrder, field2 sortingOrder, …
- Combine the **WHERE** and **ORDER BY** clauses

ORDER BY Clause (Cont.)

- **SELECT** authorID, firstName, lastName
  FROM authors
  ORDER BY lastName **ASC**

authorID	firstName	lastName
2	Paul	Deitel
1	Harvey	Deitel
3	Tem	Nieto
4	Sean	Santry

Fig. 8.17 Authors from table **authors** in ascending order by **lastName**.

ORDER BY Clause (Cont.)

- **SELECT** authorID, firstName, lastName
  FROM authors
  ORDER BY lastName **DESC**

authorID	firstName	lastName
4	Sean	Santry
3	Tem	Nieto
2	Paul	Deitel
1	Harvey	Deitel

Fig. 8.18 Authors from table **authors** in descending order by **lastName**.

ORDER BY Clause (Cont.)

- **SELECT** authorID, firstName, lastName
  FROM authors
  ORDER BY lastName, firstName

authorID	firstName	lastName
1	Harvey	Deitel
2	Paul	Deitel
3	Tem	Nieto
4	Sean	Santry

Fig. 8.19 Authors from table **authors** in ascending order by **lastName** and by **firstName**.

ORDER BY Clause (Cont.)

- **SELECT** isbn, title, editionNumber, copyright, price
  FROM titles **WHERE** title **LIKE** '%How to Program'
  ORDER BY title **ASC**

isbn	title	edition-Number	copy-right	price
0130895601	Advanced Java 2 Platform How to Program	1	2002	69.95
0132261197	C How to Program	2	1994	49.95
0130895725	C How to Program	3	2001	69.95
0135289106	C++ How to Program	2	1998	49.95
0130895717	C++ How to Program	3	2001	69.95
0130161438	Internet and World Wide Web How to Program	1	2000	69.95
0130284181	Perl How to Program	1	2001	69.95
0134569555	Visual Basic 6 How to Program	1	1999	69.95
0130284173	XML How to Program	1	2001	69.95
013028419x	e-Business and e-Commerce How to Program	1	2001	69.95

Fig. 8.20 Books from table **titles** whose title ends with **How to Program** in ascending order by **title**.

Merging Data from Multiple Tables: Joining

- Join the tables
  - Merge data from multiple tables into a single view
  - **SELECT** fieldName1, fieldName2, …

    FROM table1, table2

    WHERE table1.fieldName = table2.fieldName
  - **SELECT** firstName, lastName, isbn

    FROM authors, authorISBN

    WHERE authors.authorID = authorISBN.authorID

    ORDER BY lastName, firstName

INSERT INTO Statement

- Insert a new record into a table
  - **INSERT INTO** tableName (fieldName1, … , fieldNameN)

    VALUES (value1, … , valueN)
    - **INSERT INTO** authors (firstName, lastName)

      VALUES ('Sue', 'Smith')

authorID	firstName	lastName
1	Harvey	Deitel
2	Paul	Deitel
3	Tem	Nieto
4	Sean	Santry
5	Sue	Smith

Fig. 8.22 Table **Authors** after an **INSERT INTO** operation to add a record.

UPDATE Statement

- Modify data in a table
  - **UPDATE** tableName

    SET fieldName1 = value1, … , fieldNameN = valueN

    WHERE criteria

    - **UPDATE** authors

      SET lastName = 'Jones'

      WHERE lastName = 'Smith' **AND** firstName = 'Sue'

authorID	firstName	lastName
1	Harvey	Deitel
2	Paul	Deitel
3	Tem	Nieto
4	Sean	Santry
5	Sue	Jones

Fig. 8.23 Table **authors** after an **UPDATE** operation to change a record.

DELETE FROM Statement

- Remove data from a table
  - **DELETE FROM** tableName **WHERE** criteria
    - **DELETE FROM** authors

      WHERE lastName = 'Jones' **AND** firstName = 'Sue'

authorID	firstName	lastName
1	Harvey	Deitel
2	Paul	Deitel
3	Tem	Nieto
4	Sean	Santry

Fig. 8.24 Table **authors** after a **DELETE** operation to remove a record.

Manipulating Databases with JDBC

- Connect to a database
- Query the database
- Display the results of the query

JDBC Classes for DB Connection

- `java.sql.Driver`
  - Unless creating custom JDBC implementation, never have to deal with it. It gives JDBC a launching point for DB connectivity by responding to `DriverManager` connection requests
- `java.sql.DriverManager`
  - Maintains a list of Driver implementations and presents an application with one that matches a requested URL.
  - `getConnection(url, uid, password)`
  - `getDrivers(), registerDriver()`
- `java.sql.Connection`
  - Represents a single logical DB connection; used for sending SQL statements

Database URLs

- Uses syntax similar to net URLs
  - `jdbc:odbc:CoreJava`
  - `jdbc:pointbase:CATS`
- General syntax
  - `jdbc:subprotocol_name:other_stuff`
  - Where `subprotocol` selects the specific driver
  - Format for `other_stuff` depends on subprotocol, but in general:
  - `jdbc:subprotocol://hostname:port/other`
  - `jdbc:odbc://whitehouse.gov:5000/CATS;PWD=Rice` would connect to the `CATS` DB on port 5000 of `whitehouse.gov`, using the ODBC attribute value of `PWD` set to "Rice".

JDBC – Making the Connection

- Register the `Driver` implementation of the DB. JDBC requires a `Driver` class to register itself with `DriverManager` when it is instantiated.
  - Explicitly call `new` to load the driver (needs to be hardcoded)
  - Or, use `Class.forName("DriverClass")`
- Establish a connection with the DB

```
Connection c = DriverManager.getConnection(url, uid,
  password);
```

  - DM searches the registered Drivers until it finds the match.
  - The `Driver` class then establishes the connection, returns a `connection` object to DM, which in turn returns it back.

JDBC – Database Access Classes

- `java.sql.Statement`
  - Most basic class. It performs all the SQL statements
  - `executeQuery( String ), executeUpdate( String ), execute( String )`
- `java.sql.ResultSet`
  - One or more rows of data returned by a query

```
Statement st = c.createStatement();
ResultSet rs = st.executeQuery("…");
```

  - Methods: `next(),getString(column),getInt(..), last(), getRow()`
  - Be careful about SQL NULL and Java NULL
  - Always use `wasNull()`
  - Always call `close()` on all `ResultSet`, `Statement`, and `Connection` objects. Some drivers (e.g., IBM's native DB2) will not close the rs and st objects even when you close the connection.

JDBC – DB Access Classes (Cont.)

- `java.sql.ResultSetMetaData`
  - Provides extra information about (data about data) the ResultSet object
  - `getColumnCount(), getColumnName(column)`

```
ResultSetMetaData md = rs.getMetaData();
```

- `java.sql.DatabaseMetaData`
  - Provides extra information about the database for a given connection object
  - What tables exist, what username is being used, is the DB read-only, what are the primary keys for a table, etc.

```
DatabaseMetaData dmd = c.getMetaData();
```

  - Lets developers write apps that are DB-independent

Prepared SQL

- For each SQL statement received, the DB builds a query plan by
  - parsing the SQL statement
  - reading the SQL to determine what to do
  - formulating a plan for executing the SQL
- Repeatedly executing SQL with same query plan is very inefficient
  - UPDATE account SET balance = *XXX* WHERE id = *YYY*

```
Statement st = c.createStatement();
for (int i=0; i<accounts.length; i++ )
  st.executeUpdate("UPDATE account " +
  "SET balance = " + accounts[i].getBalance() +
  "WHERE id = " + accounts[i].getId());
```

Prepared SQL (contd.)

- DBs enable you to optimize repeated calls through prepared SQL.
- Create a Java instance of a prepared statement that notifies the DB of the kind of SQL call it represents.
- DB can then create a query plan for that SQL even before it is actually executed.
- If the same prepared statement is executed more than once, the DB uses the same query plan without rebuilding a new one.

Prepared SQL (contd.)

```
// create SQL statement with parameters
PreparedStatement st = c.prepareStatement(
                    "UPDATE account " +
                    "SET balance = ? " +
                    "WHERE id = ?");

for (int i=0; i<accounts.length; i++) {
   // bind the parameters
   st.setFloat(1, accounts[i].getBalance());
   st.setInt(2, accounts[i].getId());
   st.execute();
   st.clearParameters();
}
```