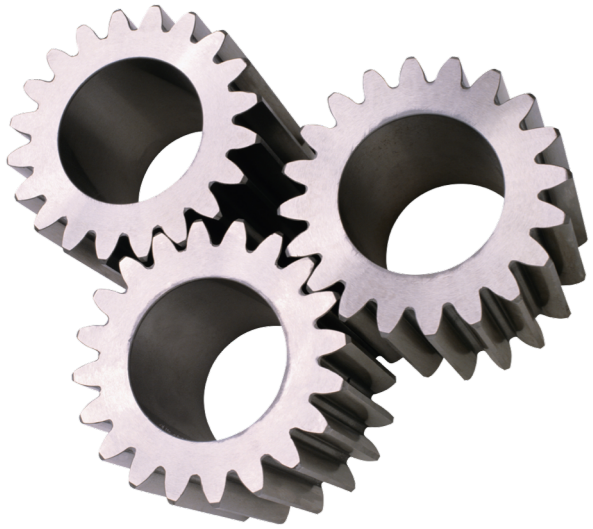
COSC344
Database Theory and Applications



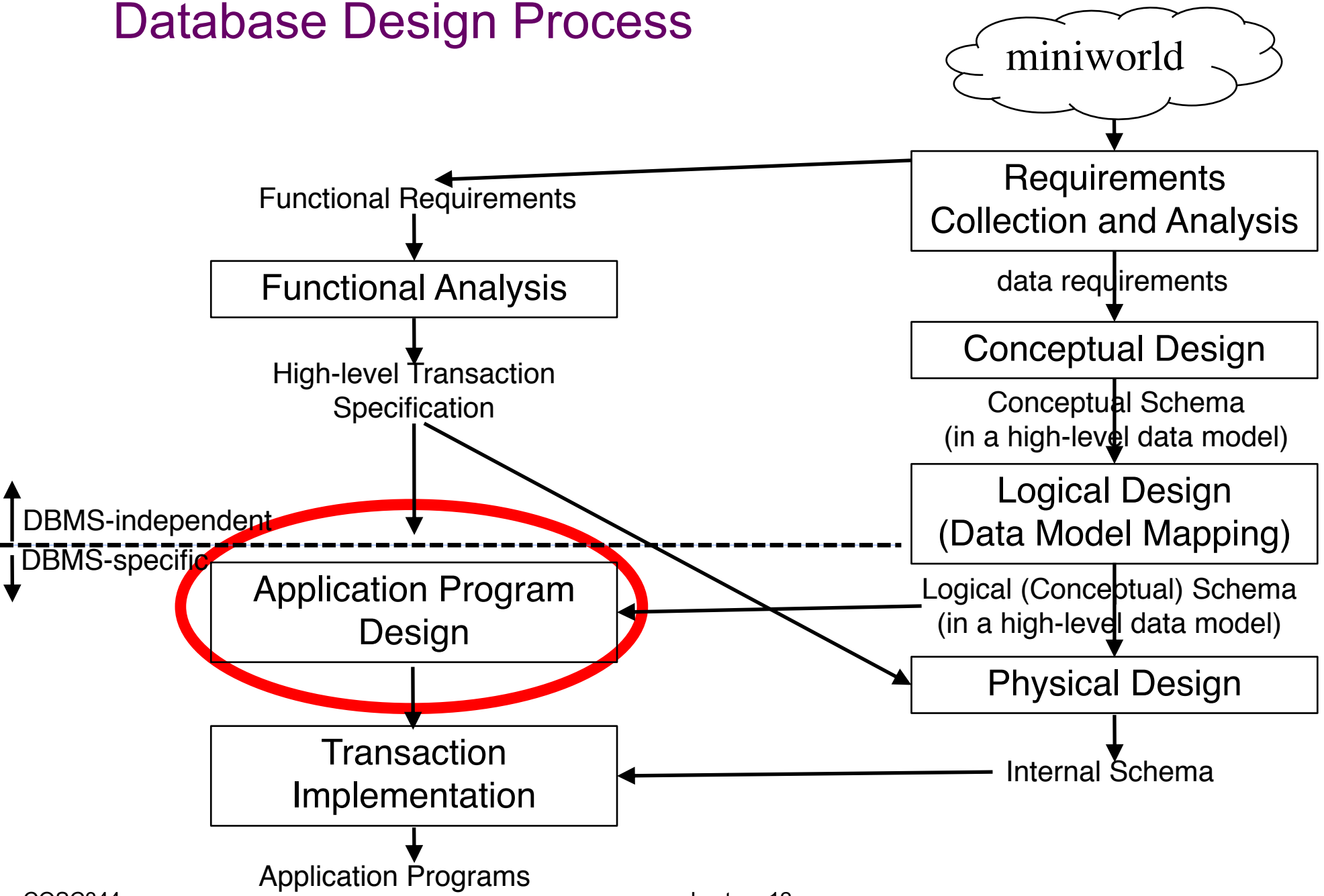
Lecture 12:
Java and SQL

Learning Objectives of This Lecture

- You should
  - understand the four drivers in JDBC
  - understand the basic steps in developing a java program to access databases.
  - get familiar with the classes and functions in JDBC library
  - be able to access and process data in database from a java program

- Source
  - Textbook: Chapter 10
  - JDBC documentation
  - Program examples

Database Design Process



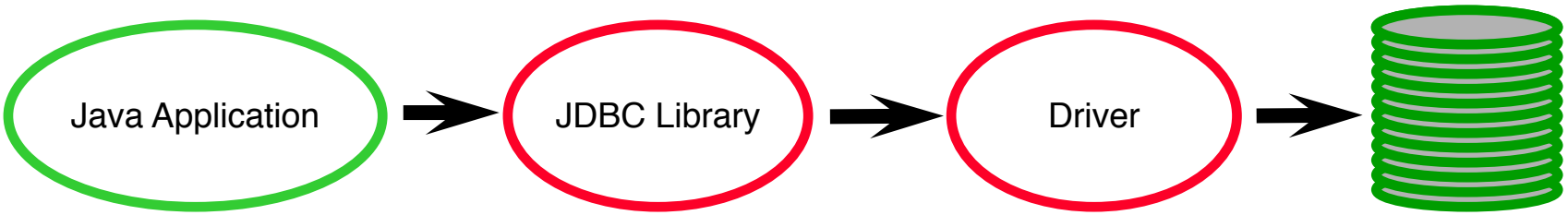
Database Programming Approaches

- Using a library of database functions
  - A library of APIs for accessing databases from programs
  - SQL commands are included as parameters in function calls
  - Java DataBase Connectivity (JDBC) --- this lecture
- Embedding database commands in a general-purpose programming language
  - Embed SQL statements in the host programming language
  - A special prefix (EXEC SQL) for identification
  - A precompiler to extract the SQL for processing in DBMS
  - Pro*C/C++ Precompiler --- next lecture
- Designing a brand-new language
  - Applications with intensive database interactions

What is JDBC?

- JDBC: Java DataBase Connectivity
  - A standard for connecting from Java to relational databases
  - A library of Java APIs for tool/database developers
  - Implemented through the standard *java.sql* interfaces
  - Allows individual providers to implement and extend the standard with their own JDBC drivers.
- Goals:
  - 100% Pure Java
  - SQL-Level
  - Simple and high performance

JDBC Architecture

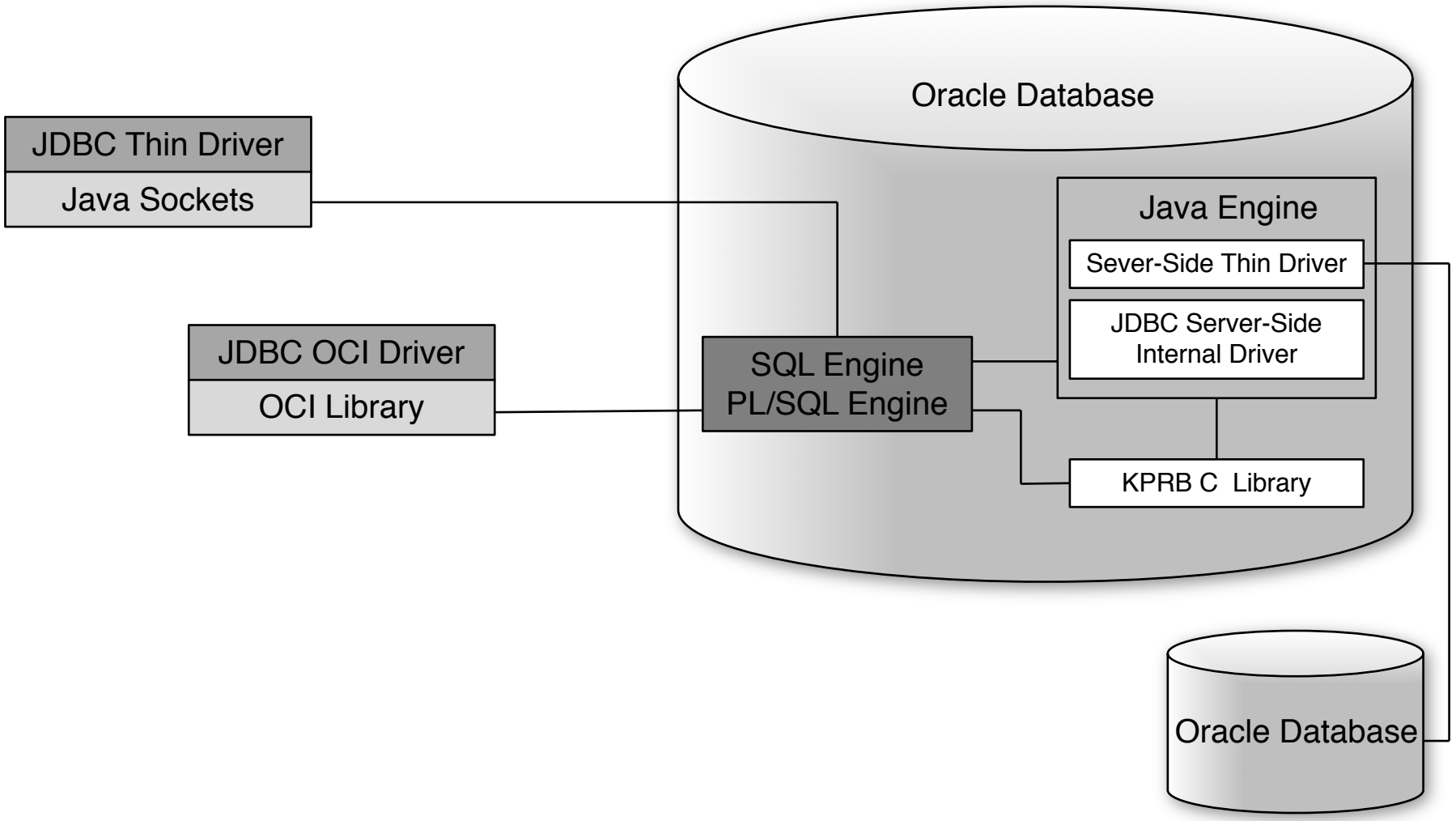


- Java application calls JDBC library.
- JDBC loads a *driver.*
- Driver communicates with a particular database.
- An application can work with several databases by using different drivers.

JDBC Drivers

- JDBC Thin Driver
  - Pure Java driver on the client-side
  - Requires no Oracle software on client-side
  - Implements SQL*Net on top of Java sockets
  - Supports the TCP/IP protocol
  - Applets and applications
- JDBC OCI Driver
  - OCI - Oracle Call Interface
  - Requires Oracle client installation
  - Written in a combination of Java and C
  - Needs OCI libraries, C-entry points, Oracle Net, …

JDBC Drivers (cont.)



JDBC Drivers (cont.)

- JDBC Server-Side Thin Driver
  - Offers the same functionality as Thin driver on client-side
  - Runs inside Oracle Database
  - Accesses a remote database from an Oracle Database
  - Accesses another session from a Java stored procedure
- JDBC Sever-Side Internal Driver
  - Supports any Java code that runs inside Oracle Database
  - Java Virtual Machine(JVM) communicates with SQL Engine
  - Fully consistent with the client-side driver
  - Only support Java 1.5

How to Use JDBC?

- Basic Steps
  - Connect to the database
    - Register the driver and set up a connection
  - Create a statement
  - Execute SQL
  - Process the result set
  - Close result set and statement objects
  - Close the connection
- Interfaces and APIs
  - *java.sql* package
  - Classes and Methods
    - http://docs.oracle.com/javase/7/docs/api/java/sql/package-summary.html

Connect to Database

- Importing Packages

Import statement	Provides
`import java.sql.*;`	Standard JDBC packages.
`import java.math.*;`	The `BigDecimal` and `BigInteger` classes. You can omit this package if you are not going to use these classes in your application.
`import oracle.jdbc.*;`	Oracle extensions to JDBC. This is optional.
`import oracle.jdbc.pool.*;`	`OracleDataSource`.
`import oracle.sql.*;`	Oracle type extensions. This is optional.

Connect to Database (cont.)

- DriverManager Class
  - Manage a set of JDBC drivers
  - Methods

    (1) static void registerDriver (Driver driver)

    //Registers the given driver with the DriverManager

    (2) static void deregisterDriver(Driver driver)

    //Drops a driver from the DriverManager's list

    (3) static Connection getConnection(String url)

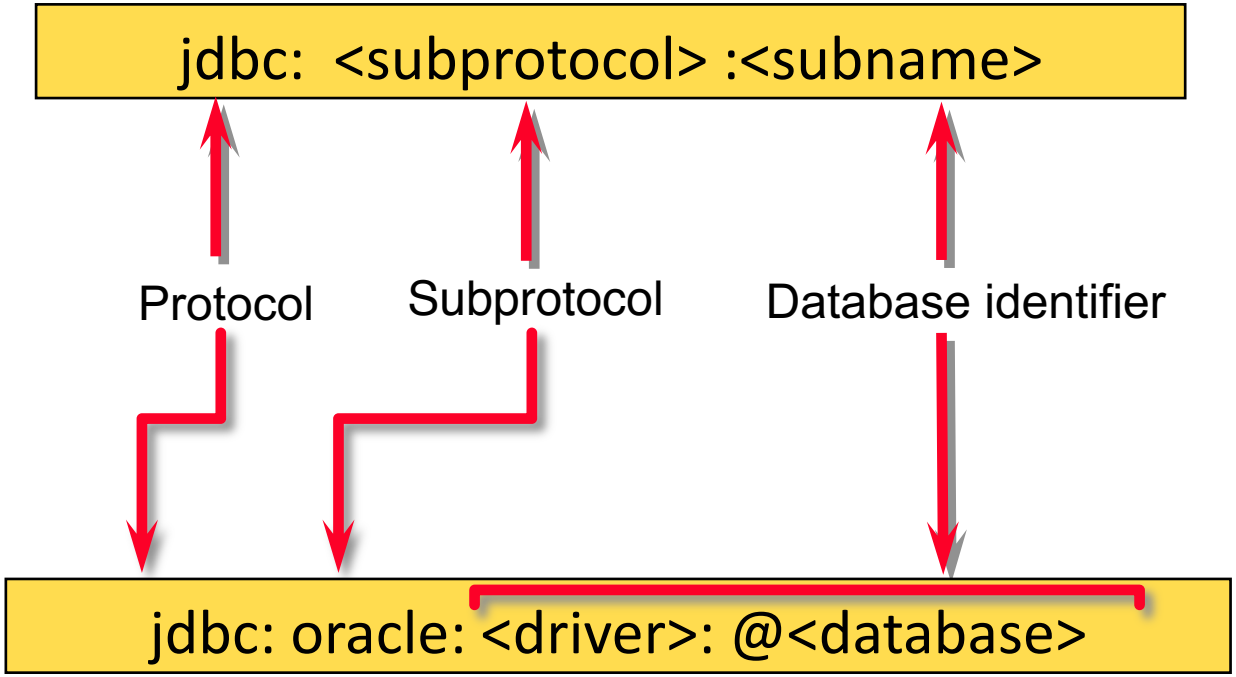
    //Attempts to establish a connection to the given database URL

    (4) static Connection getConnection(String url, String user, String password)

    //Attempts to connect to the given database URL with user name and password

JDBC URLs

- JDBC uses a URL to identify the database connection.



Our Lab jdbc: oracle: thin: @ silver:1521:cosc344

Connect to Database (example)

```
 // Read pass.dat
UserPass login = new UserPass();
String user = login.getUserName();
String pass = login.getPassWord();
String host = "silver";

// Register the driver and connect to Oracle
DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
String url = "jdbc:oracle:thin:@" + host + ":1521:cosc344";
Connection con = null;
con = DriverManager.getConnection(url, user, pass);
```

Create Statement

- Statement Interface
  - Send the SQL statement to the database
  - Return the results produced
  - Need an active connection to create a statement
- Methods supporting Statement (Connection Interface)
  - Statement createStatement()
    - a simple SQL statement with no parameters
  - PreparedStatement prepareStatement()
    - a SQL statement that is executed frequently with or without parameters
  - CallableStatement prepareCall()
    - a CallableStatement object for calling database stored procedures

Execute SQL

- Three methods to execute an SQL statement
  - ResultSet executeQuery()
    - Executes the SQL query and returns the data in a table (ResultSet)

```
ResultSet  results = stmt.executeQuery("SELECT * FROM employee");
```

  - int executeUpdate()
    - Execute INSERT, UPDATE, or DELETE Statements
    - The return is the number of rows affected in the database
    - Support Data Definition Language (DDL) statements

      CREATE TABLE, DROP TABLE, and ALTER TABLE

```
int rows = stmt.executeUpdate ("DELETE FROM employees"+
                                "WHERE dno = 4");
```

Execute SQL (cont.)

– boolean execute()

- Generic method for executing stored procedures and prepared statements
- May or may not return a ResultSet (statement.getResultSet)
- Two or more result sets may be produced
- Rarely used, exception for multiple return result sets

Execute SQL (example)

- Using Statement

```
Statement     stmt = con.createStatement();
ResultSet    reset  = stmt.executeQuery("SELECT * FROM employee");
   int          rows = statement.executeUpdate ("DELETE FROM employee"+
                                                "WHERE dno = 4")
```

- Using PreparedStatement

```
PreparedStatement  pstmt = con.prepareStatement("INSERT INTO"+
                                "employee(ird, lname) values (?,?)");

//Add Bob as employee number 1500
pstmt.setInt (1,150000000);       //The first ? Is for ird
pstmt.setString(2,"Bob");         //The second ? Is for lname
pstmt.executeUpdate();
```

Process the Result

- ResultSet Interface
  - A table representing a result set generated by a query
  - A ResultSet maintains a cursor pointing its current row of data
  - Provides a set of methods to manipulate the data in ResultSet
  - Uses a *while loop* and *next()* to iterate through the ResultSet

```
while (rset.next()) {
   ....
}
```

- Cursor Management Methods
  - boolean first()
  - boolean next()
  - boolean previous()
  - boolean last()
  - boolean isFirst()
  - boolean isLast()

Process the Result (cont.)

- Data Retrieval Methods
  - byte getByte(int columnIndex)
  - double getDouble(int columnIndex)
  - int getInt(int columnIndex)
  - Date getDate(int columnIndex)
  - ……
- Data Update Methods
  - void updateInt(int columnIndex, int x)
  - void updateNull(int columnIndex)
  - void insertRow()
  - ……
- More Methods
  - refer to ResultSet Interface in java.sql package

Process the Result (cont.)

- ResultSetMetaData Interface
  - Used to get information about the types and properties of the columns in a *ResultSet* object

```
ResultSet    reset  = stmt.executeQuery("SELECT * FROM employee");
ResultSetMetaData rsmd = reset.getMetaData();
```

  - Methods

(1) int getColumnCount();

(2) String getColumnLabel(int column)

(3) String getColumnName(int column)

(4) int getColumnType(int column)

(5) boolean isNullable(int column)

(6) boolean isReadOnly(int column)

JDBC Data Types

JDBC Type	Java Type	JDBC Type	Java Type
BIT	boolean	NUMERIC DECIMAL	BigDecimal
TINYINT	byte	DATE	Java.sql.Date
SMALLINT	short	TIME TIMESTAMP	Java.sql.Timestamp
INTEGER	int		
BIGINT	long	CLOB	Clob
REAL	float	BLOB	Blob
FLOAT DOUBLE	double	ARRAY	Array
		DISTINCT	Mapping of underlying type
BINARY VARBINARY LONGVARBINARY	byte[]	STRUCT	Struct
		REF	Ref
CHAR VARCHAR LONGVARCHAR	String	JAVA_OBJECT	Underlying java class

Close the Connection

- Close the ResultSet object

```
reset.close();
```

- Close the Statement object

```
stmt.close();
```

- Close the connection

```
con.close();
```

JDBC Exceptions

- SQLException: provides information on a database access error or other errors

```
try {

   // Java-SQL statement

}
catch (SQLException e){
    quit(e.message());
}

private void quit (String message){
    System.err.println(message);
    System.exit(1);
}
```

JDBC Exceptions (cont.)

- try…catch…finally statement

```
try {

   //Register the driver and connect to Oracle
   //Execute SQL
   //Process the result
}

catch (SQLException e){

   //Process exception

}

finally {

   //close connection

}
```

Examples

- A query example

Retrieves from the EMPLOYEE table the first and last names and department number of all employees whose first name begins with a specified letter.

- An update example

Updates the salary field for a specified employee. Ask for the IRD of an employee and a new salary. Updates the salary and commits.

- An example showing the use of wasNull()

The use of various data types and the use of wasNull()

Source code

– /coursework/344/pickup/oracle-java

Useful References

- Oracle's JDBC documentation on the course web page under Resources
- JDBC Package java.sql

  Oracle 12 documentation -> JDBC Java API Reference
- Reese, G; Database Programming with JDBC and JAVA, O'Reilly, 2nd edition
- Example programs are in /coursework/344/pickup/oracle-java

Assignment 2

- 15% of the final mark
- Due at 4pm on May 17 (Tuesday)
- Tasks
  - Revise the ERD according to the feedback
  - Convert the ERD to relational schema
  - Normalize all relations to BCNF
  - Implement in Oracle and populate data