JDBC
(Java Database Connectivity)

Agenda

- Introduction to JDBC
- Querying the Database
- Modifying the Database
- Calling Stored Procedures

Agenda

- Retrieving Database Metadata
- Handling Transactions in JDBC
- JDBC Advanced Features and Best Practices

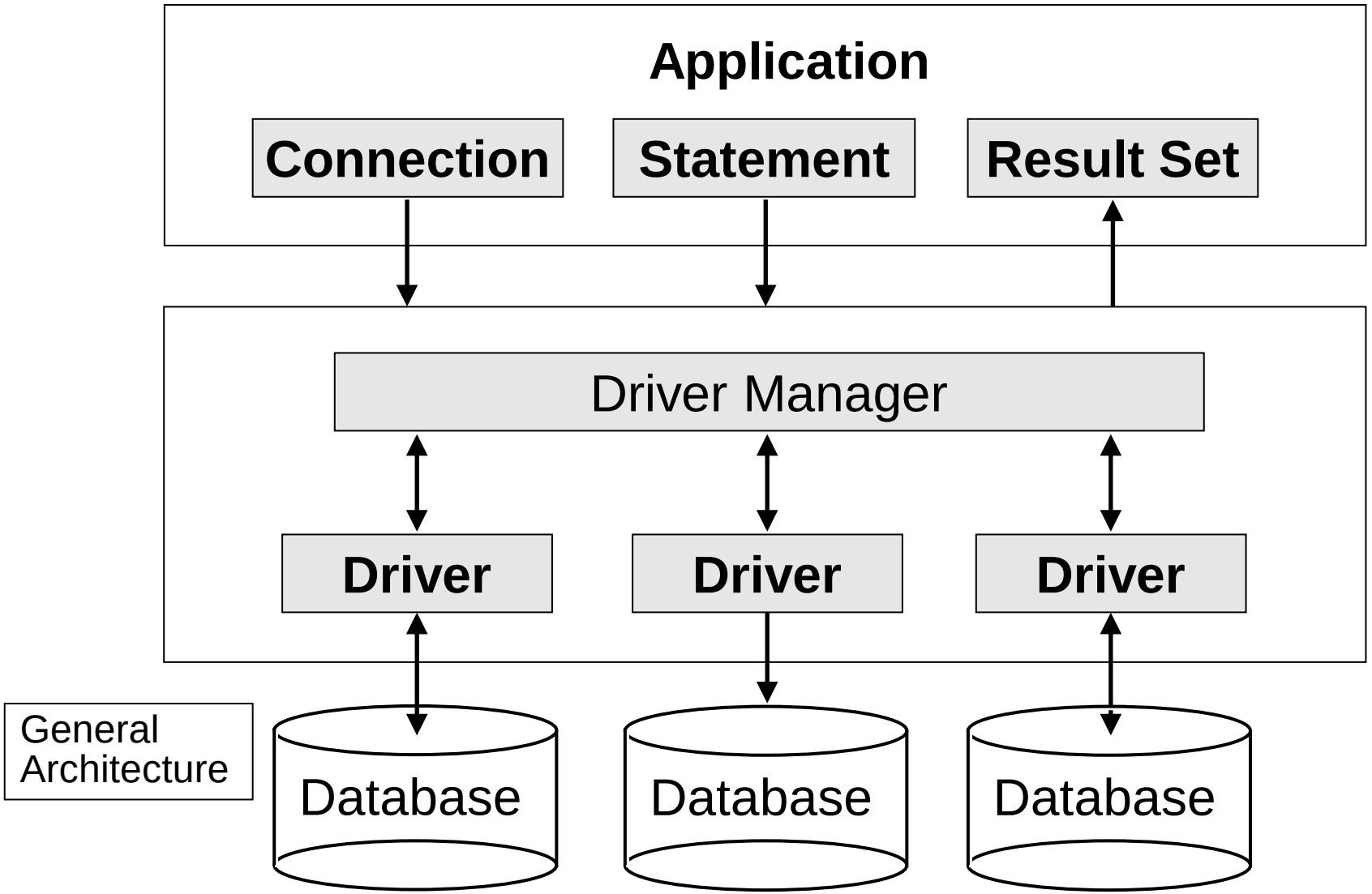
Introduction to JDBC

Technology Overview

Introduction to JDBC

- JDBC is a standard interface for connecting to relational databases from Java
  - The JDBC Core API package is `java.sql`
  - JDBC 2.0 Optional Package API is `javax.sql`
  - JDBC 3.0 API includes the Core API and Optional Package API

Introduction to JDBC

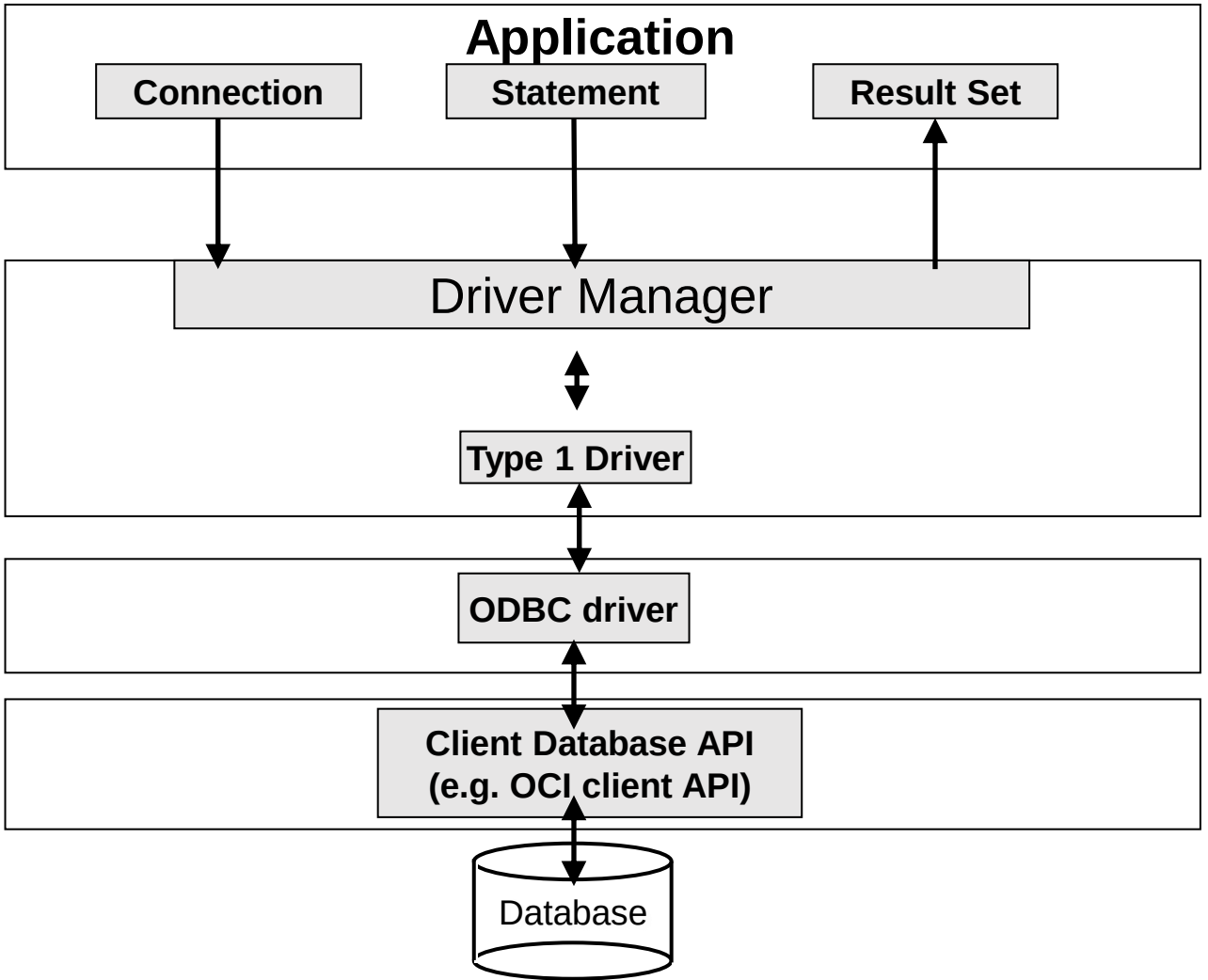


Introduction to JDBC

- Four types of JDBC drivers:
  - Type 1: JDBC-ODBC bridge driver
  - Type 2: Native API driver
  - Type 3: Network protocol driver
  - Type 4: Database protocol driver

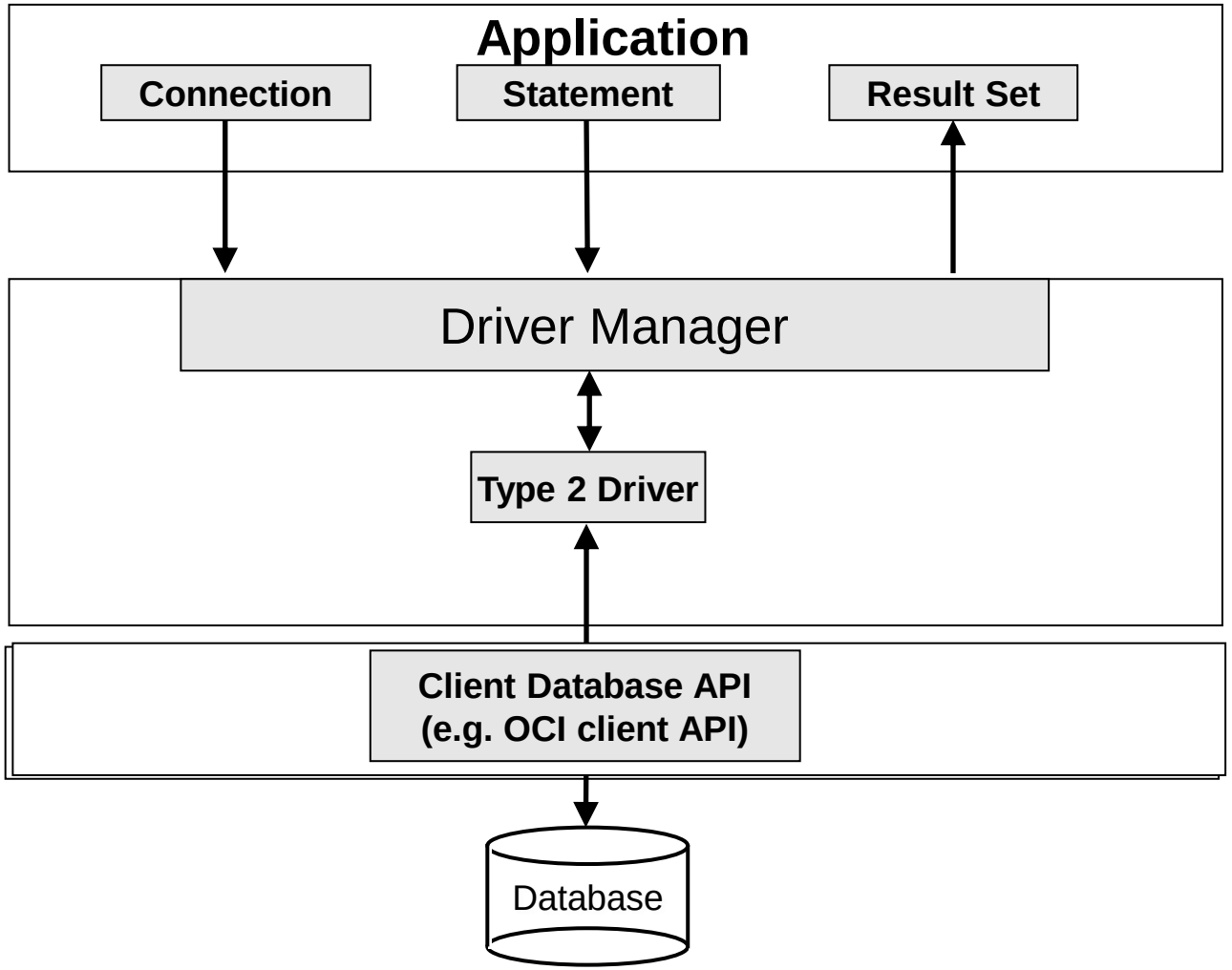
Introduction to JDBC

- Type 1: JDBC-ODBC bridge driver:



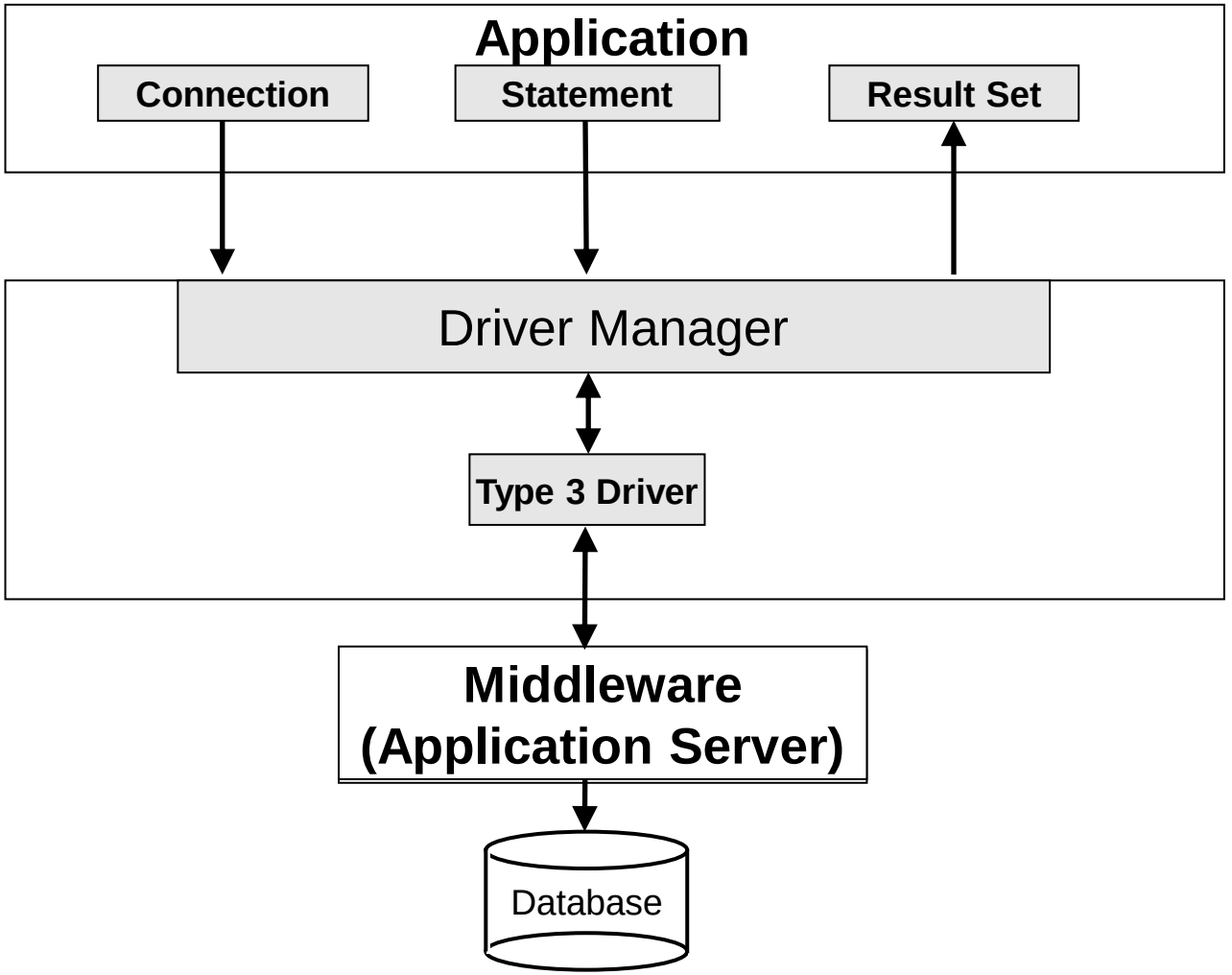
Introduction to JDBC

- Type 2: Native API driver:



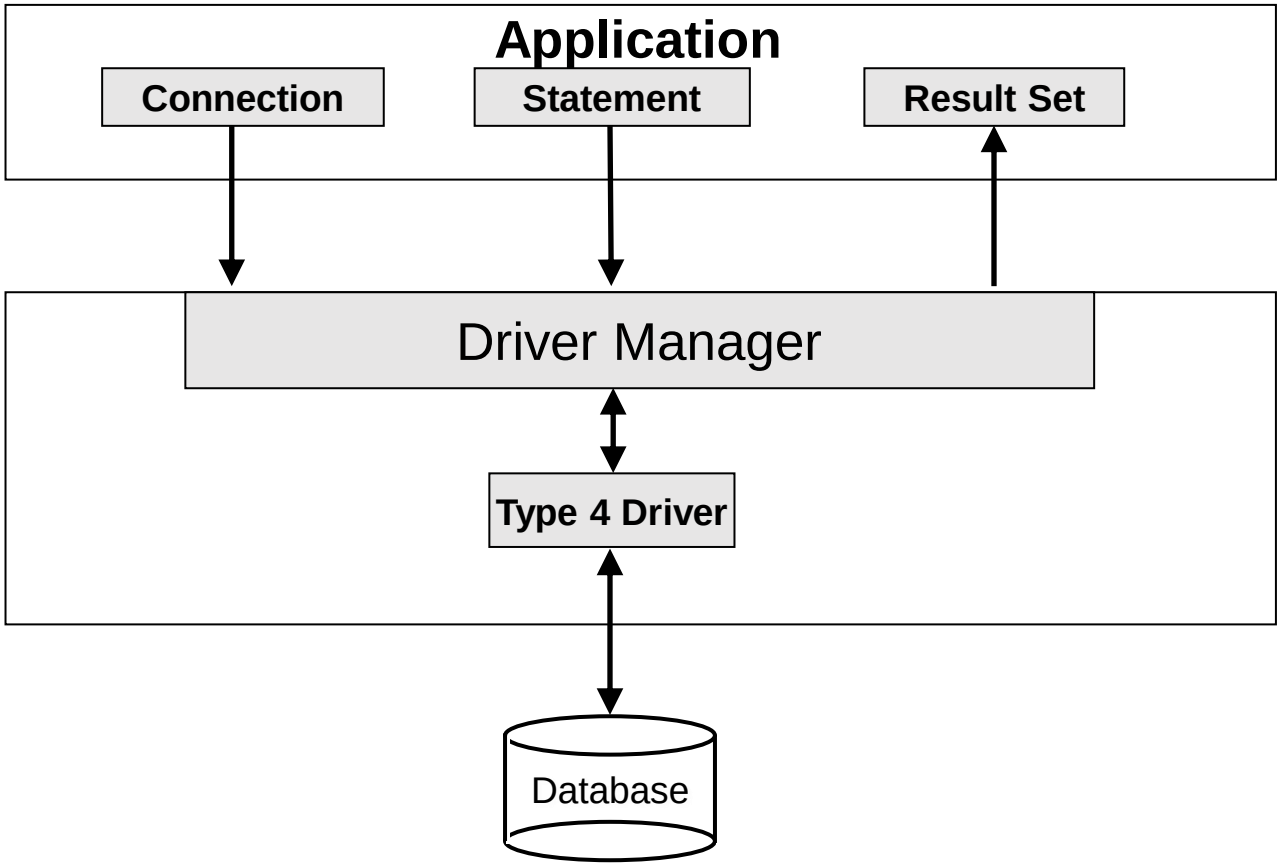
Introduction to JDBC

- Type 3: Network Protocol Driver:



Introduction to JDBC

- Type 4: Database Protocol Driver:



Introduction to JDBC

- Driver Manager - loads database drivers, and manages the connection between application & driver
- Driver - translates API calls to operations for a specific data source (database abstraction)
- Connection - a session between an application and the relational database

Introduction to JDBC

- Statement - a representation of an SQL statement
- Metadata - information about the returned data, driver and the database
- Result Set - logical set of columns and rows returned by executing a statement

Querying the Database

Data Access with JDBC

Querying the Database
(stage 1: loading the JDBC driver)

- Loading the JDBC driver is done by a single line of code:

```
Class.forName("<jdbc driver class>");
```

- Your driver documentation will give you the class name to use
- Loading MySQL or Oracle JDBC driver (not needed as of JDBC 4)

```
Class.forName("com.mysql.jdbc.Driver");
Class.forName("oracle.jdbc.driver.OracleDriver");
```

Querying the Database
(stage 2: establishing a connection)

- The following line of code illustrates the process:

```
Connection con = DriverManager.
  getConnection("url", "user", "pass");
```

- If you are using the JDBC-ODBC bridge driver, the JDBC URL will start with "`jdbc:odbc:`"

- If you are using a JDBC driver developed by a third party, the documentation will tell you what sub-protocol to use

Querying the Database
(stage 2: establishing a connection)

- Connecting to ODBC driver – example
  - ODBC data source is called "`Library`"
  - DBMS login name is "`admin`"
  - The password is "`secret`"

- Establishing a connection:

```
Connection dbCon = DriverManager.
  getConnection("jdbc:odbc:Library",
  "admin",
  "secret");
```

Querying the Database
(stage 2: establishing a connection)

- Connecting to MySQL – example
  - The server is MySQL 5.6, 5.7 or 8.0, locally installed
  - MySQL database schema is called "`HR`"
  - The password is "`hr`"

- Establishing a connection:

```
Connection dbCon =  DriverManager.getConnection(
      "jdbc:mysql://localhost/HR?" +
      "user=hr&password=hr")
```

Querying the Database
(stage 2: establishing a connection)

- Connecting to Oracle – example
  - The server is Oracle 10g Express Edition, locally installed
  - Oracle database schema is called "HR"
  - The password is "hr"

- Establishing a connection:

```
Connection dbCon = DriverManager.getConnection(
  "jdbc:oracle:thin:@localhost:1521/HR",
  "hr",
  "hr");
```

Querying the Database
(stage 3: creating a statement)

- A `Statement` object sends the SQL commands to the DBMS
  - `executeQuery()` is used for `SELECT` statements
  - `executeUpdate()` is used for statements that create/modify tables
- An instance of active `Connection` is used to create a `Statement` object

```
Connection dbCon = DriverManager.getConnection(
  "jdbc:odbc:Library", "admin", "secret");
Statement stmt = dbCon.createStatement();
```

Querying the Database
(stage 4: executing a statement)

- `executeQuery()` executes SQL command through a previously created statement
  - Returns the results in a `ResultSet` object

```
Connection dbCon =
  DriverManager.getConnection(
  "jdbc:odbc:Library", "admin", "secret");

Statement stmt = dbCon.createStatement();

ResultSet rs = stmt.executeQuery(
  "SELECT first_name FROM employees");
```

Querying the Database
(stage 4: executing a statement)

- `executeUpdate()` is used to submit DML/DDL SQL statements
  - DML is used to manipulate existing data in objects (using `UPDATE`, `INSERT`, `DELETE` statements)
  - DDL is used to manipulate database objects (`CREATE`, `ALTER`, `DROP`)

```
Statement stmt = dbCon.createStatement();
int rowsAffected = stmt.executeUpdate(
  "UPDATE employees SET salary = salary*1.2");
```

Querying the Database
(stage 5: processing the result)

- The `ResultSet` object
  - Maintains a cursor pointing to its current row of data
  - Provides methods to retrieve column values

```
ResultSet rs = stmt.executeQuery(
  "SELECT last_name, salary FROM employees");
while (rs.next()) {
  String name = rs.getString("last_name");
  double salary = rs.getDouble("salary");
  System.out.println(name + " " + salary);
}
```

Querying the Database
(stage 6: closing the connection)

- Explicitly close a `Connection`, `Statement`, and `ResultSet` to release resources that are no longer needed

```
try {
  Connection conn = ...;
  Statement stmt = ...;
  ResultSet rset = stmt.executeQuery(...);
  ...
} finally
    // clean up
    rset.close();
    stmt.close();
    conn.close();
}
```

SQLException

- SQL statements can throw `java.sql.SQLException` during their execution

```
try  {
  rset = stmt.executeQuery(
    "SELECT name, phone, email FROM employees");
  } catch (SQLException sqlex) {
      ... // Handle SQL errors here
  } finally {
    // Clean up all used resources
    try {
      if (rset != null) rset.close();
    } catch (SQLException sqlex)  {
      ... // Ignore closing errors
    }
  ...
}
```

Querying MySQL Database through JDBC

Demo

Modifying the Database

`Statement` and `PreparedStatement`

Modifying the database

1. To execute a DML statement create an empty statement object

```
Statement stmt = conn.createStatement();
```

2. Use `executeUpdate()` to execute the statement

- Example:

```
int count = stmt.executeUpdate(sql_dml_statement);
```

```
Statement stmt = conn.createStatement();

int rowsDeleted = stmt.executeUpdate("DELETE FROM
  order_items WHERE order_id = 2354");
```

Modifying the database

1. To execute a DDL statement create an empty statement object

```
Statement stmt = conn.createStatement();
```

2. Use `executeUpdate()` to execute the statement

```
int count = stmt.executeUpdate(sql_ddl_statement);
```

- Example:

```
Statement stmt = conn.createStatement();

stmt.executeUpdate("CREATE TABLE
  temp(col1 NUMBER(5,2), col2 VARCHAR2(30))");
```

Modifying the database

1. To execute an unknown statement create an empty statement object

```
Statement stmt = conn.createStatement();
```

2. Use `execute()` to execute the statement

```
boolean result = stmt.execute(SQLstatement);
```

3. Process the statement accordingly

```
if (result) { // was a query - process results
  ResultSet r = stmt.getResultSet(); ...
}
else { // was an update or DDL - process result
  int count = stmt.getUpdateCount(); ...
}
```

Modifying the database

- `PreparedStatement` is used to
  - execute a statement that takes parameters
  - execute a given statement many times

```
String insertSQL = "INSERT INTO employees(" +
  "first_name, last_name, salary) VALUES(?,?,?)";
PreparedStatement stmt =
  con.prepareStatement(insertSQL);
stmt.setString(1, "Martin");
stmt.setString(2, "Toshev");
stmt.setDouble(3, 1000.0);
stmt.executeUpdate();
```

Modifying the database

- Use prepared statements always when you need parameterized queries
  - Do not build a query using the "+" operator
  - Prefer named parameters

```
String insertSQL = "INSERT INTO employees(" +
  "first_name, last_name) VALUES(@fn, @ln)";
PreparedStatement stmt =
  con.prepareStatement(insertSQL);
stmt.setString("@fn", "Martin");
stmt.setString("@ln", "Toshev");
stmt.executeUpdate();
```

Modifying the database

- Some databases support "auto increment" primary key columns
  - E. g. MS SQL Server, MS Access, MySQL, ...
  - JDBC can retrieve auto generated keys

```
// Insert row and return PK
int rowCount = stmt.executeUpdate(
  "INSERT INTO Messages(Msg) VALUES ('Test')",
  Statement.RETURN_GENERATED_KEYS);

// Get the auto generated PK
ResultSet rs = stmt.getGeneratedKeys();
rs.next();
long primaryKey = rs.getLong(1);
```

Modifying the database

- Oracle does not support "auto increment" primary key
  - Use sequences to generate unique values

```
stmtSeq = dbCon.createStatement();
rsNextId = stmtSeq.executeQuery(
  "SELECT <some_sequence>.nextval FROM dual");
rsNextId.next();
long nextId = rsNextId.getLong(1);

psIns = dbCon.prepareStatement(
  "INSERT INTO Table(id, ...) VALUES(?, ?)");
psIns.setLong(1, nextId);
psIns.setString(2, ...);
psIns.executeUpdate();
```

Calling Stored Procedures

`CallableStatements`

Calling stored procedures

- `CallableStatement` interface
  - Is used for executing stored procedures
  - Can pass input parameters
  - Can retrieve output parameters

- Example:

```
CallableStatement callStmt =
  dbCon.prepareCall("call SP_Insert_Msg(?,?)");
callStmt.setString(1, msgText);
callStmt.registerOutParameter(2, Types.BIGINT);
callStmt.executeUpdate();
long id = callStmt.getLong(2);
```

Modifying the Database and Calling Stored Procedures

Demo

Retrieving Database Metadata

Retrieving database metadata

- Each RDBMS provides a way to retrieve database metadata (e.g. list of tables in the database or list of columns in table) - in Oracle this is the database dictionary

- JDBC provides a way to retrieve RDBMS metadata independent of the particular RDBMS implementation

Retrieving database metadata

- To retrieve metadata from the connection (e.g. user schemas, schema tables):

```
DatabaseMetaData metadata = dbCon.getMetaData();
ResultSet hrmTables = metadata.getTables(null,
"HRM",null, null);
```

- To retrieve metadata from the result set (e.g. table column names and types):

```
ResultSetMetaData resultMetadata =
hrmTables.getMetaData();
int columnCount = resultMetadata.getColumnCount();
```

Handling Transactions in JDBC

Handling transactions in JDBC

- A database transaction is a set of database operations that must be either entirely completed or aborted

- A simple transaction is usually in this form:
  1. Begin the transaction
  2. Execute several SQL DML statements
  3. Commit the transaction

- If one of the SQL statements fails, rollback the entire transaction

Handling transactions in JDBC

- JDBC transaction mode:
  - Auto-commit by default
  - Can be turned off by calling `setAutoCommit(false)`
- In auto-commit mode each statement is treated as a separate transaction
- If the auto-commit mode is off, no changes will be committed until `commit()` is invoked
- Auto-commit mode can be turned back on by calling `setAutoCommit(true)`

Handling transactions in JDBC

- If we don't want certain changes to be made permanent, we can issue `rollback()`

```
dbCon.setAutoCommit(false);
try {
  Statement stmt = con.createStatement();
  stmt.executeUpdate("INSERT INTO Groups " +
    "VALUES (101, 'Administrators')");
  stmt.executeUpdate("INSERT INTO Users " +
    "VALUES (NULL, 'Mary', 101)");
  dbCon.commit();
} catch (Exception ex) {
  dbCon.rollback();
  throw ex;
}
```

Transactions in JDBC

Demo

JDBC Advanced Features

JDBC Advanced Features

- You can pass database specific information to the database by using Properties object

- Example for Oracle database:

```
Properties props = new java.util.Properties();

props.put("user", "scott");
props.put("password", "tiger");
props.put("defaultRowPrefetch", "30");
props.put("defaultBatchValue", "5");

Connection dbCon = DriverManger.getConnection(
  "jdbc:oracle:thin:@hoststring", props);
```

JDBC Advanced Features

- Connection pool contains a number of open database connections

- There are a few choices when using connection pool:

  1. Depend on application server

  2. Use JDBC 2.0 interfaces (`ConnectionPoolDataSource` and `PooledConnection`)

  3. Create your own connection pool

JDBC Advanced Features

- You can set the transaction isolation level by calling `setTransactionIsolation(`*`level`*`)`

Transaction Level	Permitted Phenomena			Impact
	Dirty Reads	Non-Repeatable Reads	Phantom Reads	
TRANSACTION_NONE	-	-	-	FASTEST
TRANSACTION_READ_UNCOMMITED	YES	YES	YES	FASTEST
TRANSACTION_READ_COMMITED	NO	YES	YES	FAST
TRANSACTION_REPEATABLE_READ	NO	NO	YES	MEDIUM
TRANSACTION_SERIALIZABLE	NO	NO	NO	SLOW

JDBC Advanced Features

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

```
ResultSet rs = stmt.executeQuery(
  "SELECT * FROM employees");
ResultSetMetaData rsm = rs.getMetaData();
int number = rsm.getColumnCount();
for (int i=0; i<number; i++) {
  System.out.println(rsm.getColumnName(i));
}
```

JDBC Best Practices

JDBC Best Practices
(close your connection)

- Closing connections, statements and result sets explicitly allows garbage collector to recollect memory and resources as early as possible
- Close statement object as soon as you finish working with them
- Use `try-finally` statement to guarantee that resources will be freed even in case of exception

JDBC Best Practices
(use proper statements)

- Use `PreparedStatement` when you execute the same statement more than once

- Use `CallableStatement` when you want result from multiple and complex statements for a single request

JDBC Best Practices
(batch your queries)

- Send multiple queries to reduce the number of JDBC calls and improve performance:

```
statement.addBatch("sql_query1");
statement.addBatch("sql_query2");
statement.addBatch("sql_query3");

statement.executeBatch();
```

- You can improve performance by increasing number of rows to be fetched at a time

```
statement.setFetchSize(30);
```

JDBC Best Practices
(batch your queries)

- Bad:

```
Statement stmt = con.createStatement();

ResultSet rs = stmt.executeQuery(
  "SELECT * FROM EMPLOYEE WHERE ID=1");
```

- Good:

```
Statement stmt = con.createStatement();

ResultSet rs = stmt.executeQuery(
  "SELECT SALARY FROM EMPLOYEE WHERE ID=1");
```

Problems (1)

1. Write a program that prints the names of all employees and their salaries from the standard HR schema in MySQL. Don't forget to ensure that all exceptions are handled appropriately and used resources are cleaned.

2. Write a program that reads a last name and a salary range from the console and prints all matched employees and their salaries from the standard HR schema MySQL. Use `PreparedStatement` with parameters. Handle the possible exceptions and close all used resources.

Problems (2)

3. Write a program that creates a table `Countries(country_id, country_name)` and a sequence `SEQ_Countries`. Define a class `Country` with the same fields like the columns in the `Countries` table. Write a method to insert new country. Write a method to list all countries (it should return `Country[]`). Write a method to find country by `country_id`. Write a method to find country by part of its name. Finally write a method to drop the `Countries` table and the sequence `SEQ_Countries`. Test all methods.

Problems (3)

4. Create tables `People(person_id, person_name)`, `Accounts(account_id, acc_holder_id, amount)` and `Log(log_id, msg_date, log_msg)`. Define sequences for populating the primary keys in these tables (without triggers).

   Write stored procedures for inserting persons, accounts and log messages and Java methods that call them.

   Write method for transferring funds between accounts. It should keep track of all successful transfers in the `Log` table. Use transactions to maintain data consistency.

Additional Problems (1)

1. Write a program that prints the names of all employees, their managers and departments from the standard HR schema in Oracle 11g. Handle the possible exceptions and close all used resources.

2. Write a program that reads a department name from the console and prints all employees in this department and their average salary. Use the standard HR schema in Oracle 11g. Use `PreparedStatement` with parameters. Handle the possible exceptions and close all used resources.

Additional Problems (2)

3. Write a program that creates tables `Users(user_id, user_name, group_id)` and `Groups(group_id, group_name)` along with sequences fro populating their primary keys. Write classes `User` and `Group` that correspond to these tables. Write methods for adding new users and groups. Write methods for listing all groups, all users and all users by given group. Write methods for updating and deleting users and groups. Finally write a method to drop the tables `Users` and `Groups` and their sequences. Test all these methods. Handle the exceptions appropriately and close all used resources.

Additional Problems (3)

4. Modify the previous program to add also the table `Log(log_id, msg_date, msg_text)` that keeps track of all changes in the all other tables. Modify all methods to maintain the log. Don't forget to use transactions if a business operation modifies several tables.

   Add a method for adding user and group in the same time (by given user name and group name). If the group does not exists, it should be created on demand. Use transactions to guarantee the data consistency. Don't forget to register the operations in the logs.