# Chapter 32 Java Database Programming

# Objectives

☞ To understand the concept of database and database management systems (§32.2).

☞ To understand the relational data model: relational data structures, constraints, and languages (§32.2).

☞ To use SQL to create and drop tables, and to retrieve and modify data (§32.3).

☞ To learn how to load a driver, connect to a database, execute statements, and process result sets using JDBC (§32.4).

☞ To use prepared statements to execute precompiled SQL statements (§32.5).

☞ To use callable statements to execute stored SQL procedures and functions (§32.6).

☞ To explore database metadata using the DatabaseMetaData and ResultSetMetaData interfaces (§32.7).

# What is a Database System?
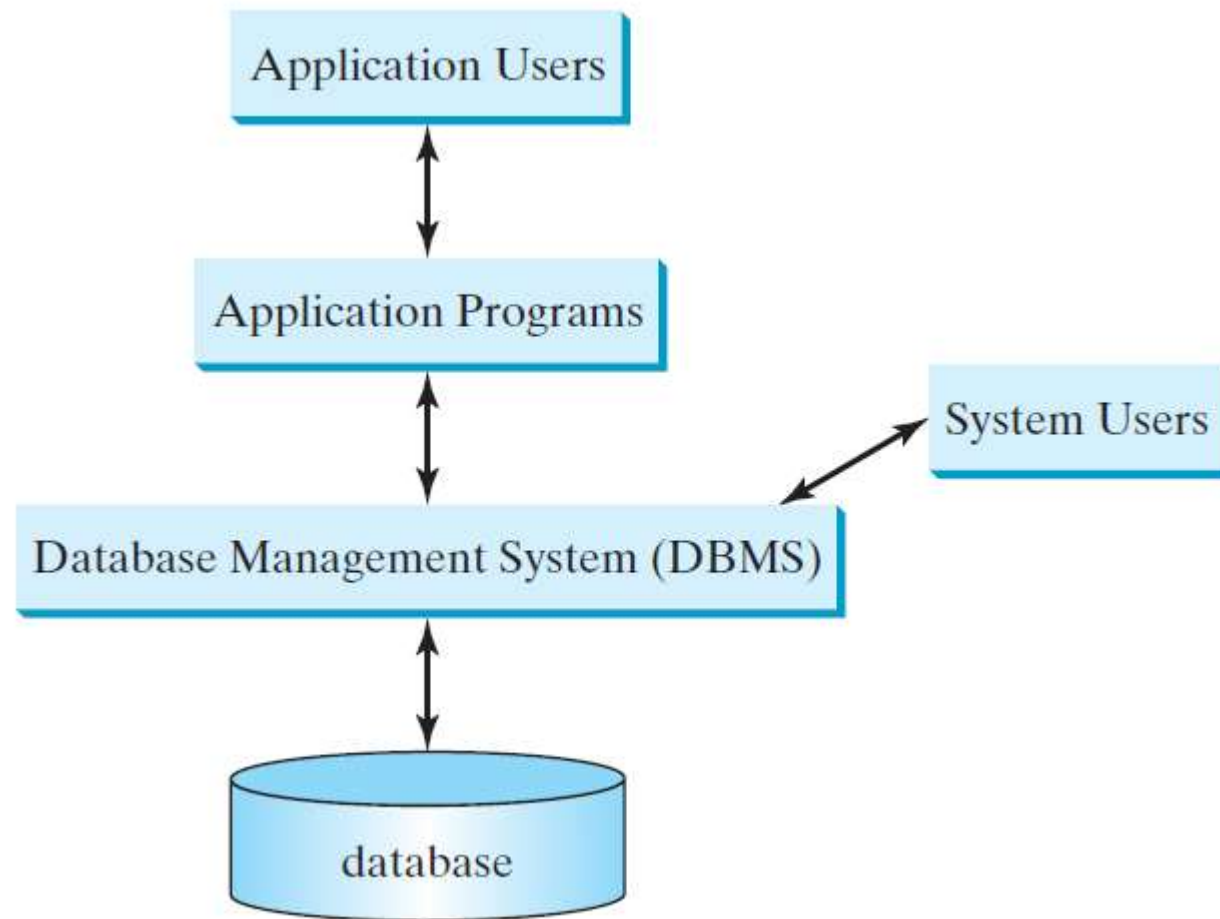
# Database Application Systems
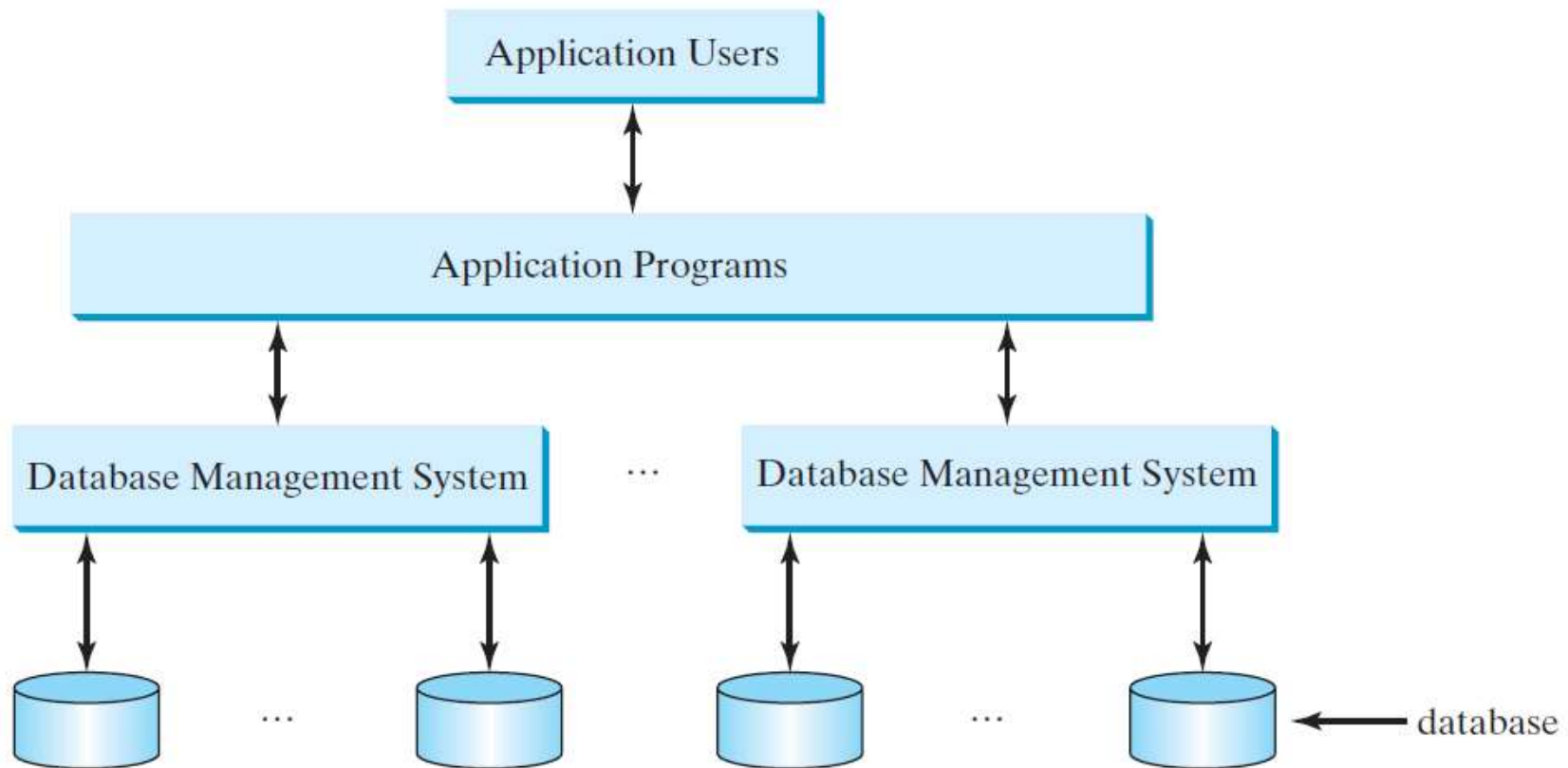
# Rational Database and Relational Data Model

Most of today's database systems are relational database systems, based on the relational data model. A relational data model has three key A relational data model has three key components: structure, integrity and languages.

☞ *Structure* defines the representation of the data.

☞ *Integrity* imposes constraints on the data.

☞ *Language* provides the means for accessing and manipulating data.

# Relational Structure

A relational database consists of a set of relations. A relation has two things in one: a *schema* and an *instance* of the schema. The schema defines the relation and an instance is the content of the relation at a given time. An instance of a relation is nothing more than a table with rows and named columns. For convenience with no confusion, we refer instances of relations as just *relations* or *tables*.

# Course Table



| Relation/Table Name | | | | | |
|---|---|---|---|---|---|
| | | | | Columns/Attributes | |
| Course Table | courseId | subjectId | courseNumber | title | numOfCredits |
| Tuples/Rows | 11111 | CSCI | 1301 | Introduction to Java I | 4 |
| | 11112 | CSCI | 1302 | Introduction to Java II | 3 |
| | 11113 | CSCI | 3720 | Database Systems | 3 |
| | 11114 | CSCI | 4750 | Rapid Java Application | 3 |
| | 11115 | MATH | 2750 | Calculus I | 5 |
| | 11116 | MATH | 3750 | Calculus II | 5 |
| | 11117 | EDUC | 1111 | Reading | 3 |
| | 11118 | ITEC | 1344 | Database Administration | 3 |

# Student Table

| Student Table | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| ssn | firstName | mi | lastName | phone | birthDate | | street | zipCode | deptID |
| 444111110 | Jacob | R | Smith | 9129219434 | 1985-04-09 | 99 | Kingston Street | 31435 | BIOL |
| 444111111 | John | K | Stevenson | 9129219434 | null | 100 | Main Street | 31411 | BIOL |
| 444111112 | George | K | Smith | 9129213454 | 1974-10-10 | 1200 | Abercorn St. | 31419 | CS |
| 444111113 | Frank | E | Jones | 9125919434 | 1970-09-09 | 100 | Main Street | 31411 | BIOL |
| 444111114 | Jean | K | Smith | 9129219434 | 1970-02-09 | 100 | Main Street | 31411 | CHEM |
| 444111115 | Josh | R | Woo | 7075989434 | 1970-02-09 | 555 | Franklin St. | 31411 | CHEM |
| 444111116 | Josh | R | Smith | 9129219434 | 1973-02-09 | 100 | Main Street | 31411 | BIOL |
| 444111117 | Joy | P | Kennedy | 9129229434 | 1974-03-19 | 103 | Bay Street | 31412 | CS |
| 444111118 | Toni | R | Peterson | 9129229434 | 1964-04-29 | 103 | Bay Street | 31412 | MATH |
| 444111119 | Patrick | R | Stoneman | 9129229434 | 1969-04-29 | 101 | Washington St. | 31435 | MATH |
| 444111120 | Rick | R | Carter | 9125919434 | 1986-04-09 | 19 | West Ford St. | 31411 | BIOL |

# Enrollment Table

**Enrollment Table**

| ssn | courseId | dateRegistered | grade |
|---|---|---|---|
| 444111110 | 11111 | 2004-03-19 | A |
| 444111110 | 11112 | 2004-03-19 | B |
| 444111110 | 11113 | 2004-03-19 | C |
| 444111111 | 11111 | 2004-03-19 | D |
| 444111111 | 11112 | 2004-03-19 | F |
| 444111111 | 11113 | 2004-03-19 | A |
| 444111112 | 11114 | 2004-03-19 | B |
| 444111112 | 11115 | 2004-03-19 | C |
| 444111112 | 11116 | 2004-03-19 | D |
| 444111113 | 11111 | 2004-03-19 | A |
| 444111113 | 11113 | 2004-03-19 | A |
| 444111114 | 11115 | 2004-03-19 | B |
| 444111115 | 11115 | 2004-03-19 | F |
| 444111115 | 11116 | 2004-03-19 | F |
| 444111116 | 11111 | 2004-03-19 | D |
| 444111117 | 11111 | 2004-03-19 | D |
| 444111118 | 11111 | 2004-03-19 | A |
| 444111118 | 11112 | 2004-03-19 | D |
| 444111118 | 11113 | 2004-03-19 | B |

# Table vs. File

NOTE:

A table or a relation is not same as a file. Most of the relational database systems store multiple tables in a file.

# Integrity Constraints

An integrity constraint imposes a condition that all legal instances of the relations must satisfy. In general, there are three types of constraints: *domain constraint, primary key constraint*, and *foreign key constraint*. Domain constraints and primary key constraints are known as *intra-relational constraints*, meaning that a constraint involves only one relation. The foreign key constraint is known as *inter-relational*, meaning that a constraint involves more than one relation.

# Domain Constraints

| Enrollment Table | ssn | courseId | dateRegistered | grade |
|---|---|---|---|---|
| | 444111110 | 11111 | 2004-03-19 | A |
| | 444111110 | 11112 | 2004-03-19 | B |
| | 444111110 | 11113 | 2004-03-19 | C |
| | ... | | | |

Each value in courseId in the Enrollment table must match a value in courseId in the Course table

| Course Table | courseId | subjectId | courseNumber | title | numOfCredits |
|---|---|---|---|---|---|
| | 11111 | CSCI | 1301 | Introduction to Java I | 4 |
| | 11112 | CSCI | 1302 | Introduction to Java II | 3 |
| | 11113 | CSCI | 3720 | Database Systems | 3 |
| | ... | | | | |

Each row must have a value for courseId, and the value must be unique

Each value in the numOfCredits column must be greater than 0 and less than 5

# Primary Key Constraints

Enrollment Table

| ssn | courseId | dateRegistered | grade |
|-----|----------|----------------|-------|
| 444111110 | 11111 | 2004-03-19 | A |
| 444111110 | 11112 | 2004-03-19 | B |
| 444111110 | 11113 | 2004-03-19 | C |
| ... | | | |

Each value in courseId in the
Enrollment table must match a value
in courseId in the Course table

Course Table

| courseId | subjectId | courseNumber | title | numOfCredits |
|----------|-----------|--------------|-------|--------------|
| 11111 | CSCI | 1301 | Introduction to Java I | 4 |
| 11112 | CSCI | 1302 | Introduction to Java II | 3 |
| 11113 | CSCI | 3720 | Database Systems | 3 |
| ... | | | | |

Each row must have a
value for couserId, and
the value must be unique

*Primary key
constraint*

Each value in the
numOfCredits column must be
greater than 0 and less than 5

# Foreign Key Constraints

Enrollment Table

| ssn | courseId | dateRegistered | grade |
|-----|----------|----------------|-------|
| 444111110 | 11111 | 2004-03-19 | A |
| 444111110 | 11112 | 2004-03-19 | B |
| 444111110 | 11113 | 2004-03-19 | C |
| ... | | | |

Each value in courseId in the
Enrollment table must match a value
in courseId in the Course table

Course Table

| courseId | subjectId | courseNumber | title | numOfCredits |
|----------|-----------|--------------|-------|--------------|
| 11111 | CSCI | 1301 | Introduction to Java I | 4 |
| 11112 | CSCI | 1302 | Introduction to Java II | 3 |
| 11113 | CSCI | 3720 | Database Systems | 3 |
| ... | | | | |

Each row must have a
value for couserId, and
the value must be unique

*Foreign key
constraint*

Each value in the
numOfCredits column must be
greater than 0 and less than 5

# Domain Constraints

*Domain constraints* specify the permissible values for an attribute. Domains can be specified using standard data types such as integers, floating-point numbers, fixed-length strings, and variant-length strings. The standard data type specifies a broad range of values. Additional constraints can be specified to narrow the ranges. You can also specify whether an attribute can be null.

# Domain Constraints Example

```
create table Course (
  courseId char(5),
  subjectId char(4) not null,
  courseNumber integer,
  title varchar(50) not null,
  numOfCredits integer,
  constraint greaterThanOne
      check (numOfCredits >= 1));
```

);

# Superkey

Superkey

Key

Candidate
key

Primary
key

A superkey is an attribute or a set of attributes that uniquely identify the relation. That is, no two tuples have the same values on the superkey. By definition, a relation consists of a set of distinct tuples. The set of all attributes in the relation forms a superkey.

# Key and Candidate Key

Superkey

Key

Candidate key

Primary key

A *key* K is a minimal superkey, meaning that any proper subset of K is not a superkey. It is possible that a relation has several keys. In this case, each of the keys is called a *candidate key*.

# Primary Key

Superkey

Key

Candidate
key

Primary
key

The primary key is one of the candidate keys designated by the database designer. The primary key is often used to identify tuples in a relation.

create table Course(
    subjectCode char(4),
    courseNumber int,
    title varchar(50), numOfCredits int
        constraint greaterThanOne check (numOfCredits >= 1),
    primary key (subjectCode, courseNumber));

# Primary Key

The *primary* key is one of the candidate keys designated by the database designer. The primary key is often used to identify tuples in a relation.

```
create table Course (
    courseId char(5),
    subjectId char(4) not null,
    courseNumber integer,
    title varchar(50) not null,
    numOfCredits integer,
    primary key (courseId)
);
```

# Primary Key Constraints

The *primary key constraint* specifies that the primary key value of a tuple cannot be null and no two tuples in the relation can have the same value on the primary key. The DBMS enforces the primary key constraint. For example, if you attempt to insert a record with the same primary key as an existing record in the table, the DBMS would report an error and reject the operation.

# Foreign Key Constraints

In a relational database, data are related. Tuples in a relation are related and tuples in different relations are related through their common attributes. Informally speaking, the common attributes are foreign keys. The *foreign key constraints* define the relationships among relations.

# Foreign Key Constraints Formal Definition

Formally, a set of attributes *FK* is a *foreign key* in a relation *R* that references relation *T* if it satisfies the following two rules:

☞ The attributes in *FK* have the same domain as the primary key in *T*.

☞ A non-null value on *FK* in *R* must match a primary key value in *T*.

$$R \xrightarrow{\quad FK \quad} T$$

# Foreign Key Example

```
create table Enrollment (
  ssn char(9),
  courseId char(5),
  dateRegistered date,
  grade char(1),
  primary key (ssn, courseId),
  foreign key (ssn) references Student,
  foreign key (courseId) references Course
);
```

# Foreign Key Discussion

A foreign key is not necessarily the primary key or part of the primary in the relation. For example, <u>subjectCode</u> is a foreign key in the <u>Course</u> table that references the <u>Subject</u> table, but it is not the primary key in <u>Course</u>. <u>departmentCode</u> is a foreign key in the <u>Subject</u> table that references <u>Department</u>, but it is not the primary key in <u>Subject</u>.

# Foreign Key Discussion, cont.

The referencing relation and the referenced relation may be the same table. For example, supervisorId is a foreign key in Faculty that references facultyId in Faculty.

# Foreign Key Discussion, cont.

The foreign key is not necessary to have the same name as its referenced primary key as long as they have the same domain. For example, headId is a foreign key in Department that references facultyId in Faculty.

# Foreign Key Discussion, cont.

A relation may have more than one foreign key.
For example, <u>headId</u> and <u>collegeCode</u> are both
foreign keys in <u>Department</u>.

# SQL

Structured Query Language, pronounced S-Q-L, or Sequel

To access or write applications for database systems, you need to use the Structured Query Language (SQL). SQL is the universal language for accessing relational database systems. Application programs may allow users to access database without directly using SQL, but these applications themselves must use SQL to access the database.

# Examples of simple SQL statements

Create table

Drop table

Describe table

Select

Insert

Delete

Update

```
create table Course (
    courseId char(5),
    subjectId char(4) not null,
    courseNumber integer,
    title varchar(50) not null,
    numOfCredits integer,
    primary key (courseId)
);
```

```
create table Student (
    ssn char(9),
    firstName varchar(25),
    mi char(1),
    lastName varchar(25),
    birthDate date,
    street varchar(25),
    phone char(11),
    zipCode char(5),
    deptId char(4),
    primary key (ssn)
);
```

# Examples of simple SQL statements

Create table

**Drop table**

Describe table

Select

Insert

Delete

Update

```
drop table Enrollment;
drop table Course;
drop table Student;
```

# Examples of simple SQL statements

Create table

Drop table

**Describe table**

Select

Insert

Delete

Update

```
describe Course;  -- Oracle
```

# Examples of simple SQL statements

Create table

Drop table

Describe table

Select

Insert

Delete

Update

```
select firstName, mi, lastName
from Student
where deptId = 'CS';


select firstName, mi, lastName
from Student
where deptId = 'CS' and zipCode = '31411';


select *
from Student
where deptId = 'CS' and zipCode = '31411';
```

# Examples of simple SQL statements

Create table

Drop table

Describe table

Select

Insert

Delete

Update

insert into Course (courseId, subjectId, courseNumber, title)
values ('11113', 'CSCI', '3720', 'Database Systems', 3);

# Examples of simple SQL statements

Create table

Drop table

Describe table

Select

Insert

Update

Delete

```
update Course
set numOfCredits = 4
where title = 'Database Systems';
```

# Examples of simple SQL statements

Create table

Drop table

Describe table

Select

Insert

Update

Delete

delete Course
where title = 'Database System';

# Why Java for Database Programming?

- First, Java is platform independent. You can develop platform-independent database applications using SQL and Java for any relational database systems.

- Second, the support for accessing database systems from Java is built into Java API, so you can create database applications using all Java code with a common interface.

- Third, Java is taught in almost every university either as the first programming language or as the second programming language.

# Database Applications Using Java

GUI

Client/Server

Server-Side programming

# The Architecture of JDBC
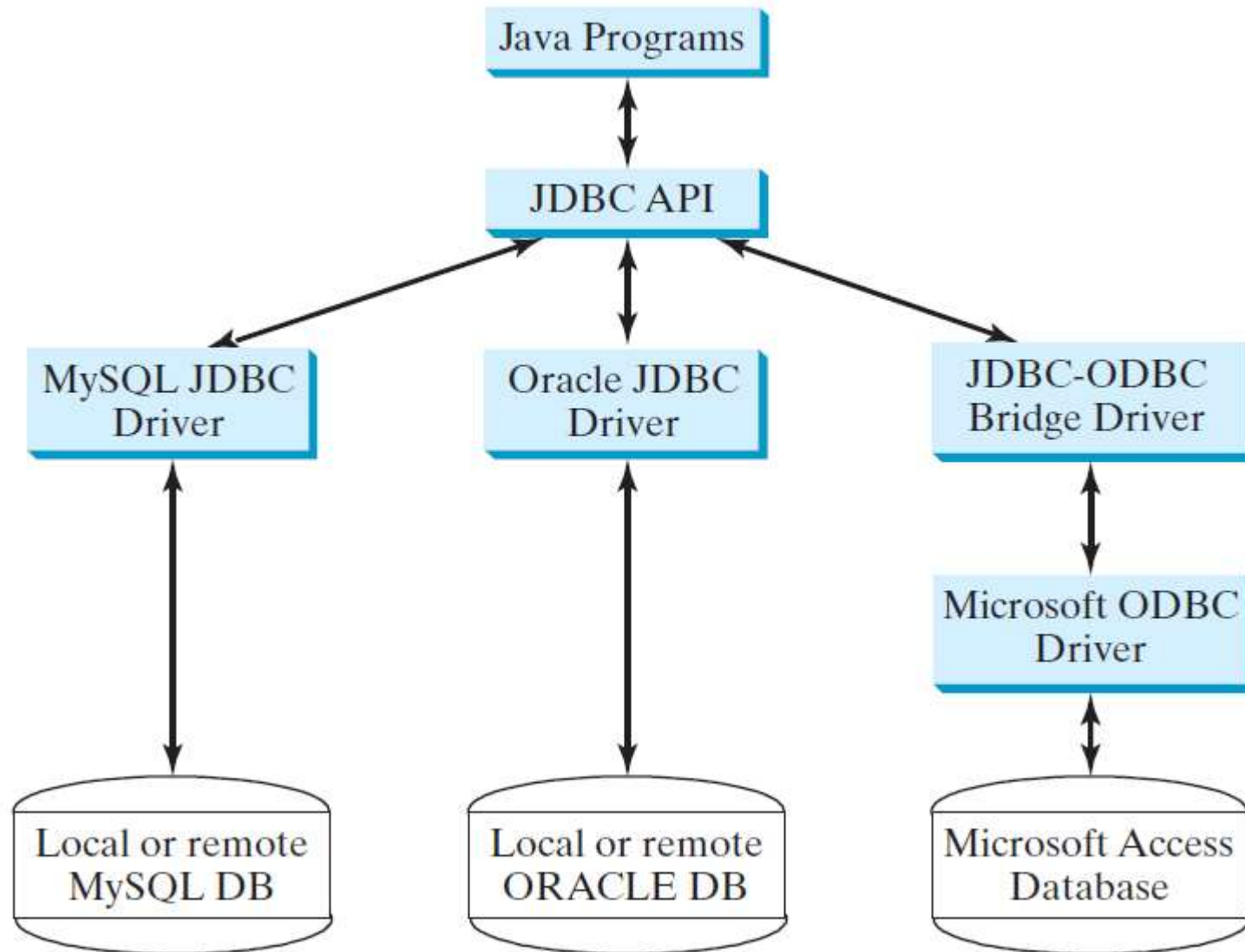
# The JDBC Interfaces
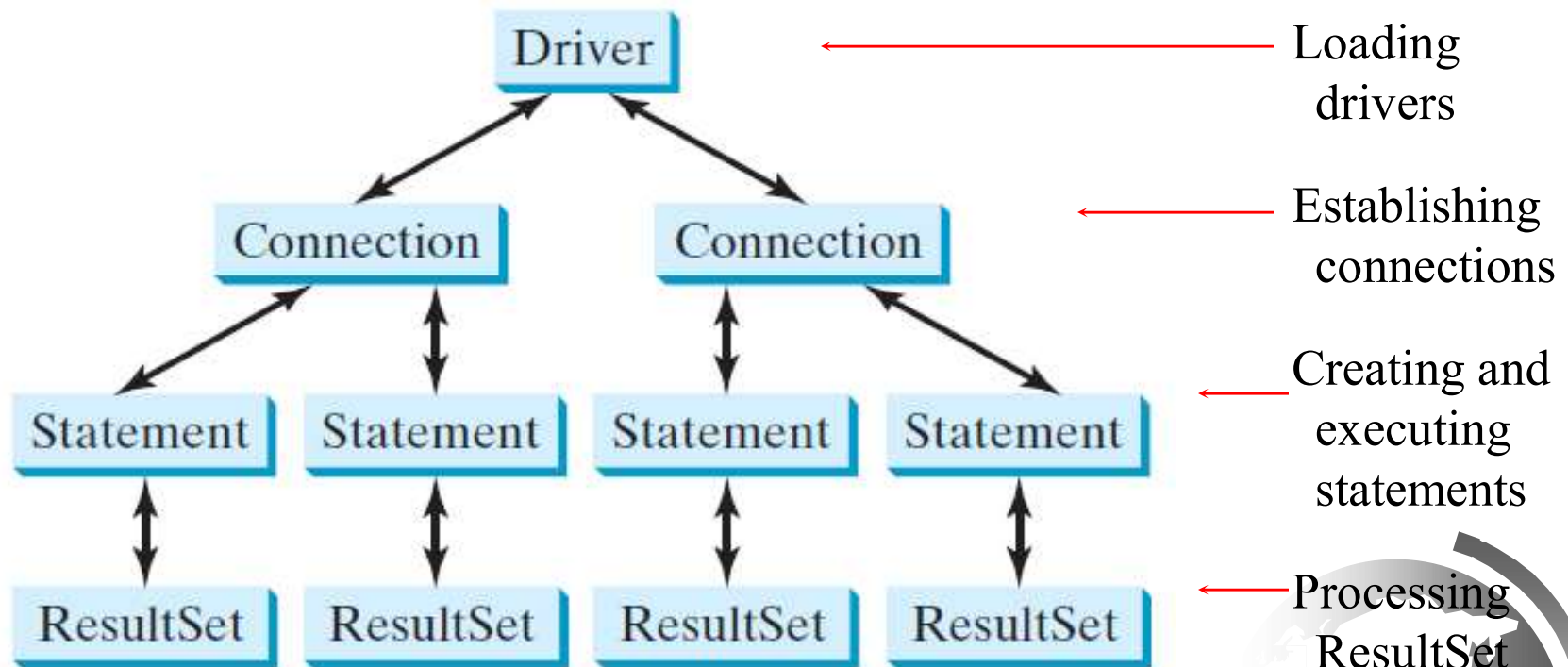


Loading drivers

Establishing connections

Creating and executing statements

Processing ResultSet

# Developing JDBC Programs

**Loading drivers**

**Establishing connections**

**Creating and executing statements**

**Processing ResultSet**

Statement to load a driver:

    Class.forName("JDBCDriverClass");

A driver is a class.  For example:

| Database | Driver Class | Source |
|---|---|---|
| Access | sun.jdbc.odbc.JdbcOdbcDriver | Already in JDK |
| MySQL | com.mysql.jdbc.Driver | Website |
| Oracle | oracle.jdbc.driver.OracleDriver | Website |

The JDBC-ODBC driver for Access is bundled in JDK.

MySQL driver class is in mysqljdbc.jar

Oracle driver class is in classes12.jar

To use the MySQL and Oracle drivers, you have to add mysqljdbc.jar and classes12.jar in the classpath using the following DOS command on Windows:

    classpath=%classpath%;c:\book\mysqljdbc.jar;c:\book\classes12.jar

# Developing JDBC Programs

Loading drivers

Establishing connections

Creating and executing statements

Processing ResultSet

Connection connection = DriverManager.getConnection(databaseURL);

Database     URL Pattern
Access            jdbc:odbc:dataSource
MySQL          jdbc:mysql://hostname/dbname
Oracle          jdbc:oracle:thin:@hostname:port#:oracleDBSID

See Supplement IV.D for creating an ODBC data source

Examples:
For Access:

    Connection connection = DriverManager.getConnection
     ("jdbc:odbc:ExampleMDBDataSource");

For MySQL:

    Connection connection = DriverManager.getConnection
     ("jdbc:mysql://localhost/test");

For Oracle:

    Connection connection = DriverManager.getConnection
     ("jdbc:oracle:thin:@liang.armstrong.edu:1521:orcl",  "scott", "tiger");

# Developing JDBC Programs

Loading
drivers

Establishing
connections

Creating and
executing
statements

Processing
ResultSet

Creating statement:

```
Statement statement = connection.createStatement();
```

Executing statement (for update, delete, insert):

```
statement.executeUpdate
    ("create table Temp (col1 char(5), col2 char(5))");
```

Executing statement (for select):

```
// Select the columns from the Student table
ResultSet resultSet = statement.executeQuery
  ("select firstName, mi, lastName from Student where lastName "
    + " = 'Smith'");
```

# Developing JDBC Programs

Loading drivers

Establishing connections

Creating and executing statements

**Processing ResultSet**

Executing statement (for select):

```
// Select the columns from the Student table
ResultSet resultSet = stmt.executeQuery
  ("select firstName, mi, lastName from Student where lastName "
    + " = 'Smith'");

Processing ResultSet (for select):
  // Iterate through the result and print the student names
  while (resultSet.next())
    System.out.println(resultSet.getString(1) + " " + resultSet.getString(2)
      + ". " + resultSet.getString(3));
```

# Simple JDBC Example

```java
import java.sql.*;
public class SimpleJdbc {
  public static void main(String[] args)
      throws SQLException, ClassNotFoundException {
    // Load the JDBC driver
    Class.forName("com.mysql.jdbc.Driver");
    System.out.println("Driver loaded");

    // Establish a connection
    Connection connection = DriverManager.getConnection
      ("jdbc:mysql://localhost/test");
    System.out.println("Database connected");

    // Create a statement
    Statement statement = connection.createStatement();

    // Execute a statement
    ResultSet resultSet = statement.executeQuery
      ("select firstName, mi, lastName from Student where lastName "
        + " = 'Smith'");

    // Iterate through the result and print the student names
    while (resultSet.next())
      System.out.println(resultSet.getString(1) + "\t" +
        resultSet.getString(2) + "\t" + resultSet.getString(3));

    // Close the connection
    connection.close();
  }
}
```

SimpleJdbc          Run

# Creating ODBC Data Source

Please follow the steps in Supplement on the Companion Website to create an ODBC data source on Windows.

# Example:
# Accessing Database from JavaFX

This example demonstrates connecting to a database from a Java applet. The applet lets the user enter the SSN and the course ID to find a student's grade.

FindGrade

Run

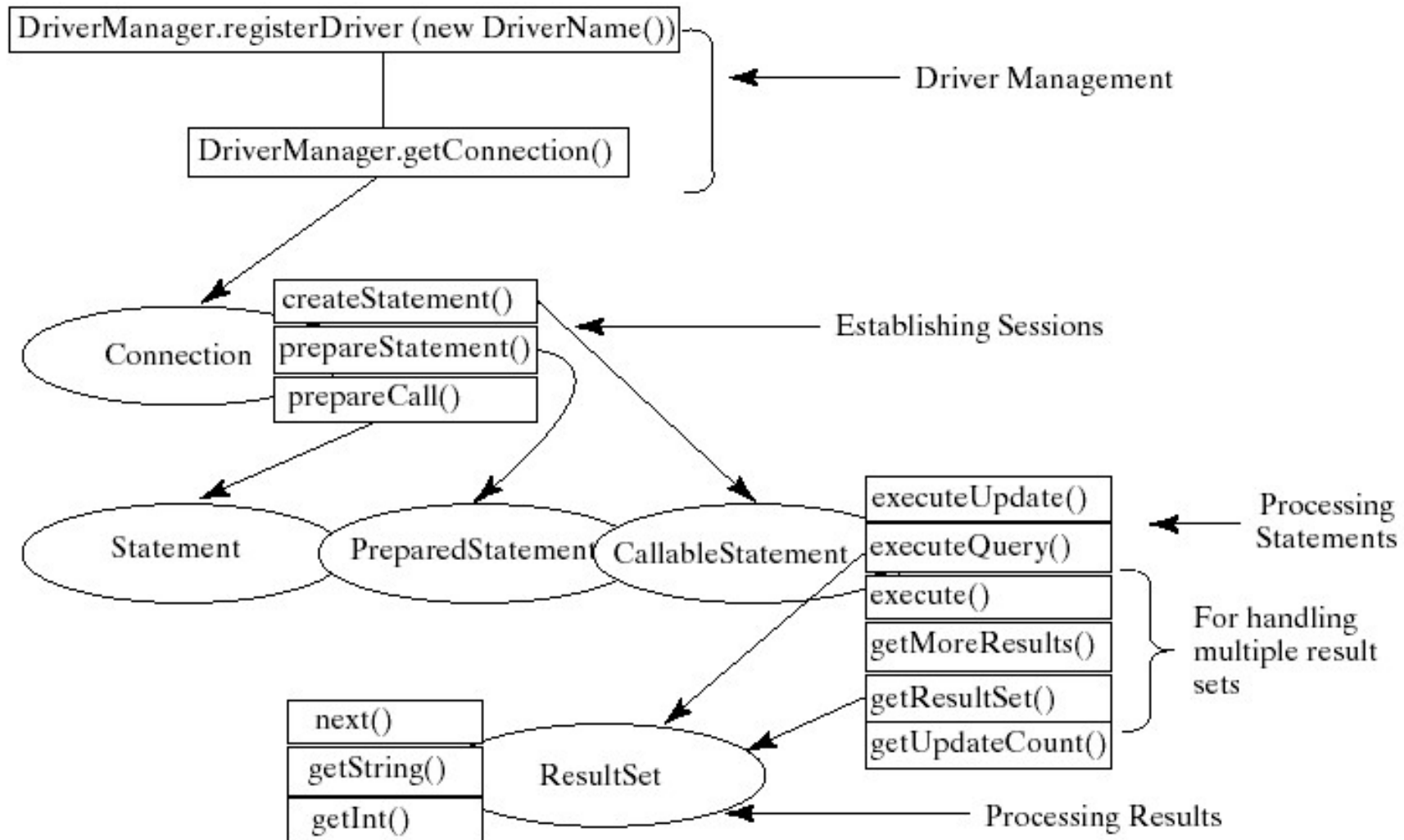NOTE: To run this program from here, you need:
1. To have a MySQL database setup just like the one in the text.
2. Set MySQL JDBC driver in the classpath.

# Processing Statements

Once a connection to a particular database is established, it can be used to send SQL statements from your program to the database. JDBC provides the Statement, PreparedStatement, and CallableStatement interfaces to facilitate sending statements to a database for execution and receiving execution results from the database.

# Processing Statements Diagram



DriverManager.registerDriver (new DriverName()) ← Driver Management

DriverManager.getConnection()

Connection

createStatement()
prepareStatement()
prepareCall() ← Establishing Sessions

Statement    PreparedStatement    CallableStatement

executeUpdate()
executeQuery() ← Processing Statements
execute()
getMoreResults()
getResultSet()
getUpdateCount()
} For handling multiple result sets

next()
getString()
getInt()

ResultSet ← Processing Results

# The execute, executeQuery, and executeUpdate Methods

The methods for executing SQL statements are execute, executeQuery, and executeUpdate, each of which accepts a string containing a SQL statement as an argument. This string is passed to the database for execution. The execute method should be used if the execution produces multiple result sets, multiple update counts, or a combination of result sets and update counts.

# The execute, executeQuery, and executeUpdate Methods, cont.

The executeQuery method should be used if the execution produces a single result set, such as the SQL select statement. The executeUpdate method should be used if the statement results in a single update count or no update count, such as a SQL INSERT, DELETE, UPDATE, or DDL statement.

# PreparedStatement

The PreparedStatement interface is designed to execute dynamic SQL statements and SQL-stored procedures with IN parameters. These SQL statements and stored procedures are precompiled for efficient use when repeatedly executed.

Statement pstmt = connection.prepareStatement

("insert into Student (firstName, mi, lastName) +

values (?, ?, ?)");

# Example:
# Using PreparedStatement to Execute Dynamic SQL Statements

This example rewrites the preceding example using <u>PreparedStatement</u>.

FindGradeUsingPreparedStatement  Run

# Retrieving Database Metadata

Database metadata is the information that describes database itself. JDBC provides the DatabaseMetaData interface for obtaining database wide information and the ResultSetMetaData interface for obtaining the information on the specific ResultSet.

# DatabaseMetadata, cont.

The DatabaseMetaData interface provides more than 100 methods for getting database metadata concerning the database as a whole. These methods can be divided into three groups: for retrieving general information, for finding database capabilities, and for getting object descriptions.

# General Information

The general information includes the URL, username, product name, product version, driver name, driver version, available functions, available data types and so on.

# Obtaining Database Capabilities

The examples of the database capabilities are whether the database supports the GROUP BY operator, the ALTER TABLE command with add column option, supports entry-level or full ANSI92 SQL grammar.

# Obtaining Object Descriptions

the examples of the database objects are
tables, views, and procedures.

```
DatabaseMetaData dbMetaData = connection.getMetaData();

System.out.println("database URL: " +
  dbMetaData.getURL());
System.out.println("database username: " +            Examples
  dbMetaData.getUserName());
System.out.println("database product name: " +
  dbMetaData.getDatabaseProductName());
System.out.println("database product version: " +
  dbMetaData.getDatabaseProductVersion());
System.out.println("JDBC driver name: " +             Sample run on
  dbMetaData.getDriverName());                         next slide
System.out.println("JDBC driver version: " +
  dbMetaData.getDriverVersion());
System.out.println("JDBC driver major version: " +
  new Integer(dbMetaData.getDriverMajorVersion()));
System.out.println("JDBC driver minor version: " +
  new Integer(dbMetaData.getDriverMinorVersion()));
System.out.println("Max number of connections: " +
  new Integer(dbMetaData.getMaxConnections()));
System.out.println("MaxTableNameLentgh: " +
  new Integer(dbMetaData.getMaxTableNameLength()));
System.out.println("MaxColumnsInTable: " +
  new Integer(dbMetaData.getMaxColumnsInTable()));
connection.close();
```
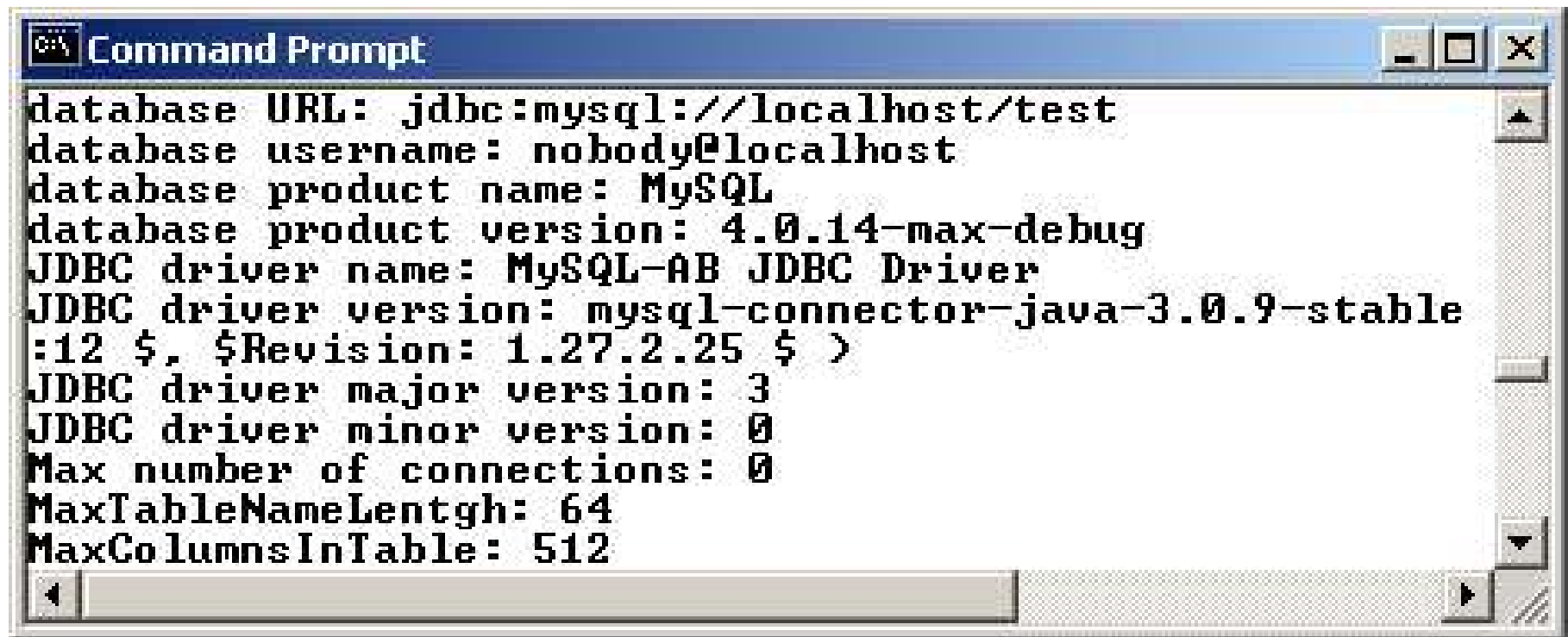
# Sample Run



```
database URL: jdbc:mysql://localhost/test
database username: nobody@localhost
database product name: MySQL
database product version: 4.0.14-max-debug
JDBC driver name: MySQL-AB JDBC Driver
JDBC driver version: mysql-connector-java-3.0.9-stable
:12 $, $Revision: 1.27.2.25 $ )
JDBC driver major version: 3
JDBC driver minor version: 0
Max number of connections: 0
MaxTableNameLentgh: 64
MaxColumnsInTable: 512
```