Advanced Database Management Systems

Lecture 12 – Chapters 9 and 26
SQL Application Interface

Create Table: Naming Constraints

- Name constraints by placing “constraint <name>” at front of constraint clause.

constraint names

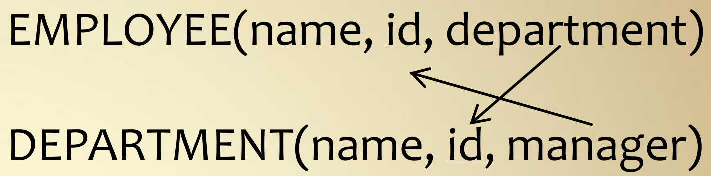
```
constraint PK_EMP Primary Key(Ssn),
constraint FK_EMP_SUPER
Foreign Key(Super_ssn) references EMPLOYEE(Ssn)
```

- This is sometimes necessary to refer to the constraint later.
  - example: removing a constraint with ALTER TABLE

```
alter table EMPLOYEE
drop foreign key FK_EMP_SUPER;
```

Circular Foreign Keys

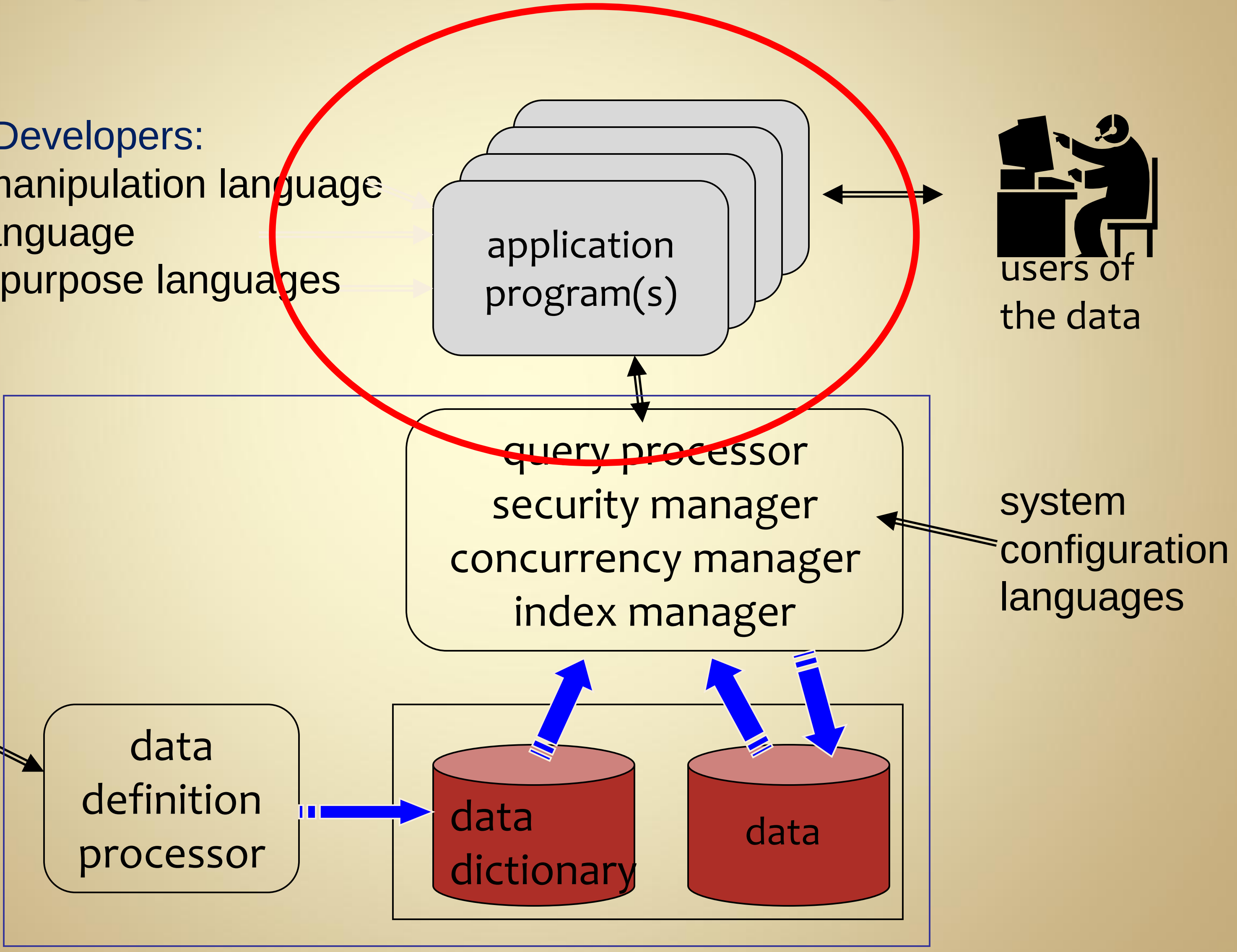
```
create table EMPLOYEE (
name varchar(10),
id integer,
department integer,
constraint PK_EMP primary key (id)
) ENGINE=InnoDB;

create table DEPARTMENT (
name varchar(20),
id integer,
manager integer,
constraint PK_DEPT primary key (id),
constraint FK_DEPT_EMP foreign key (manager) references EMPLOYEE(id)
) ENGINE=InnoDB;

alter table EMPLOYEE
add constraint FK_EMP_DEPT foreign key (department) references
DEPARTMENT(id);
```



DB Application Development

Application Developers:
DML: data manipulation language
QL: query language
PL: general purpose languages

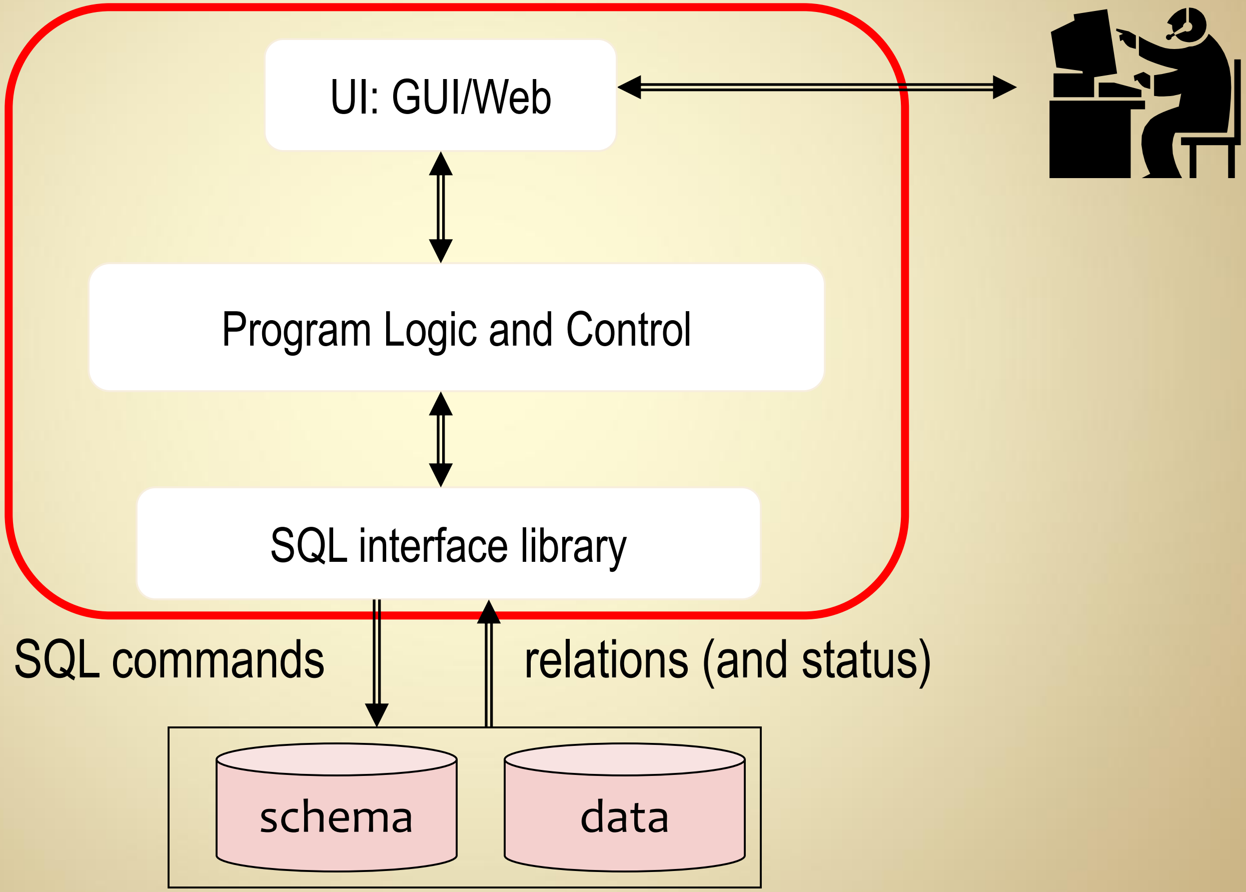
DDL:
data
definition
language



DB Application Development

- Application development requires a general purpose programming language (GPL)
  - most end-users do not want to run SQL commands
  - GPL referred to as “host language”
- GPL provides all non-db functionality
  - gui or web interface
  - error handling
  - application logic
  - etc.

DBMS Applications



GPL/SQL interface(s)

- SLI: statement level interface
  - new kinds of *statements* are added to the host language (i.e. EXEC SQL)
  - preprocessor translates new statements into host language procedures
  - host language compiler used once preprocessed
- CLI: call level interface
  - interface to SQL supplied as *library*
  - applications written entirely in host language
  - no preprocessing required

GPL/SQL interface(s)

- Statement level
  - Embedded SQL
  - Dynamic SQL

older languages:
C, COBOL

- Call level
  - JDBC (Java)
  - ODBC – Open Database Connectivity
  - libraries in PHP, Python, Perl, Visual Basic, etc.

more common
in modern languages

Embedded (Static) SQL

- SQL statements are directly written into program

- SQL is checked against the schema at compile time

- host language variables are used in the SQL statements as parameters and return values

- programs interact with one specific database (code compiled against schema)

Dynamic SQL

- SQL statements are generated by program (as string values)
- SQL checked against schema at run-time
- SQL variables defined as placeholders in statement
- Programs can interact with multiple databases

Example: Static SQL

```
EXEC SQL  BEGIN  DECLARE SECTION;
  unsigned long num_enrolled;
  char crs_code;
  char  SQLSTATE [6];
EXEC SQL  END  DECLARE  SECTION;
   ……….
EXEC  SQL  SELECT  C.NumEnrolled
     INTO  :num_enrolled
     FROM  Course C
     WHERE  C.CrsCode = :crs_code;
```

variables shared by host and SQL

INTO clause: where to put result

: indicates a host variable

Example: Static SQL

host variable

```
EXEC SQL CONNECT  TO :dbserver;
if (!strcmp (SQLSTATE, “00000”))
       exit (1);
```

status string set by SQL command

Example: Static SQL

```
EXEC SQL DELETE  FROM  Transcript T
   WHERE  T.StudId = :studid
   AND  T.Semester = ‘S2000’
   AND  T.CrsCode = :crscode;
if (!strcmp(SQLSTATE,“00000”))
   EXEC SQL ROLLBACK;
else {
   EXEC SQL UPDATE Course C
            SET  C.Numenrolled = C.Numenrolled – 1
            WHERE  C.CrsCode =  :crscode;
    if (!strcmp(SQLSTATE,“00000”))
       EXEC SQL ROLLBACK;
    else
       EXEC SQL COMMIT;
}
```

Buffer Mismatch Problem

- **Problem:**

  SQL deals with tables (of arbitrary size);

  host program deals with fixed size buffers

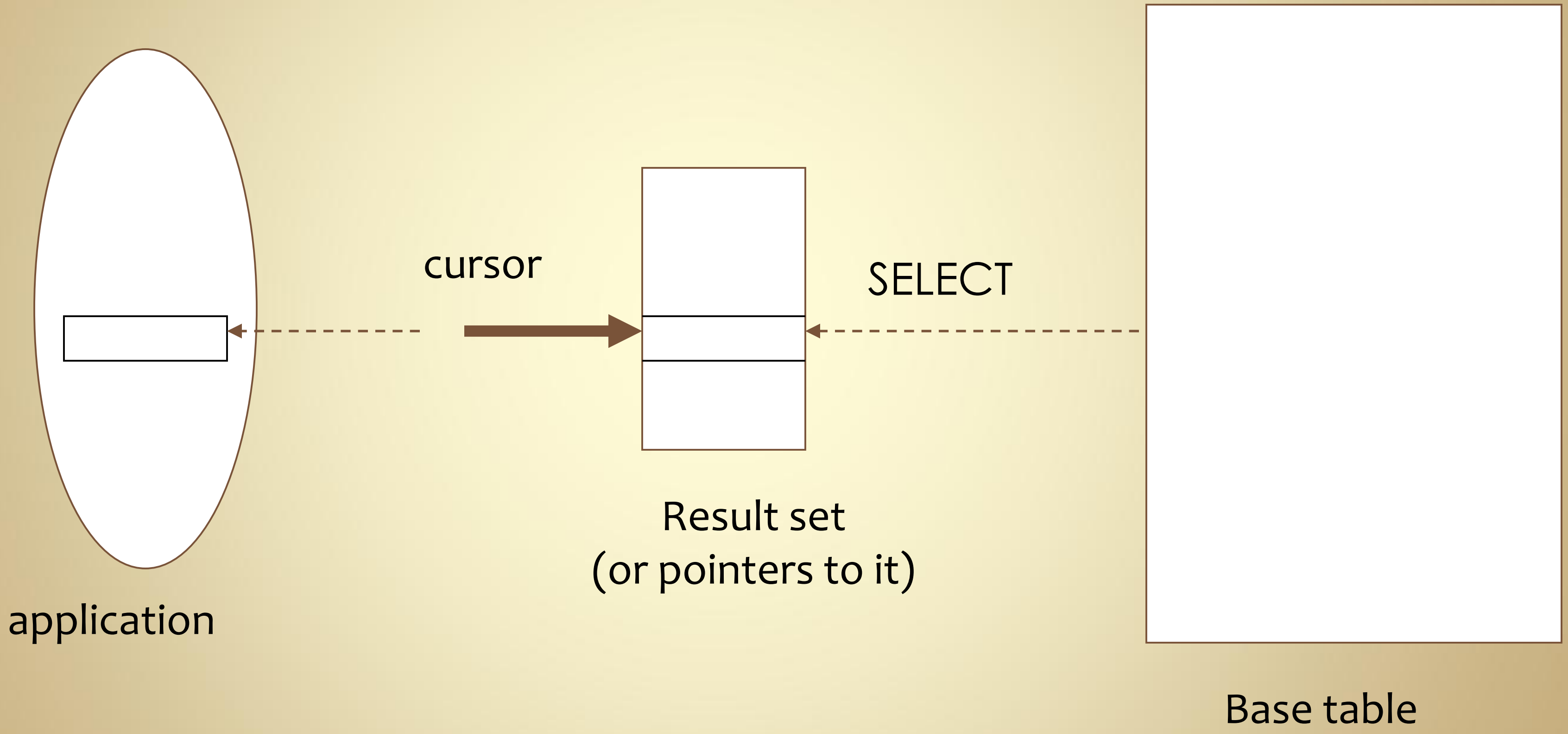
  - How is the application to allocate storage for the result of a SELECT statement?

- **Solution:** Fetch a single row at a time
  - Space for a single row (number and type of *out* parameters) can be determined from schema and allocated in application

Cursors

- ***Result set*** – set of rows produced by a SELECT statement
- ***Cursor*** – pointer to a row in the result set.
  - a cursor is similar to an *iterator*
- ***Cursor operations:***
  - *Declaration*
  - *Open* – execute SELECT to determine result set and initialize pointer
  - *Fetch* – advance pointer and retrieve next row
  - *Close* – deallocate cursor

Cursors


cursor
SELECT
application
Result set
(or pointers to it)
Base table

CLI: dynamic buffers

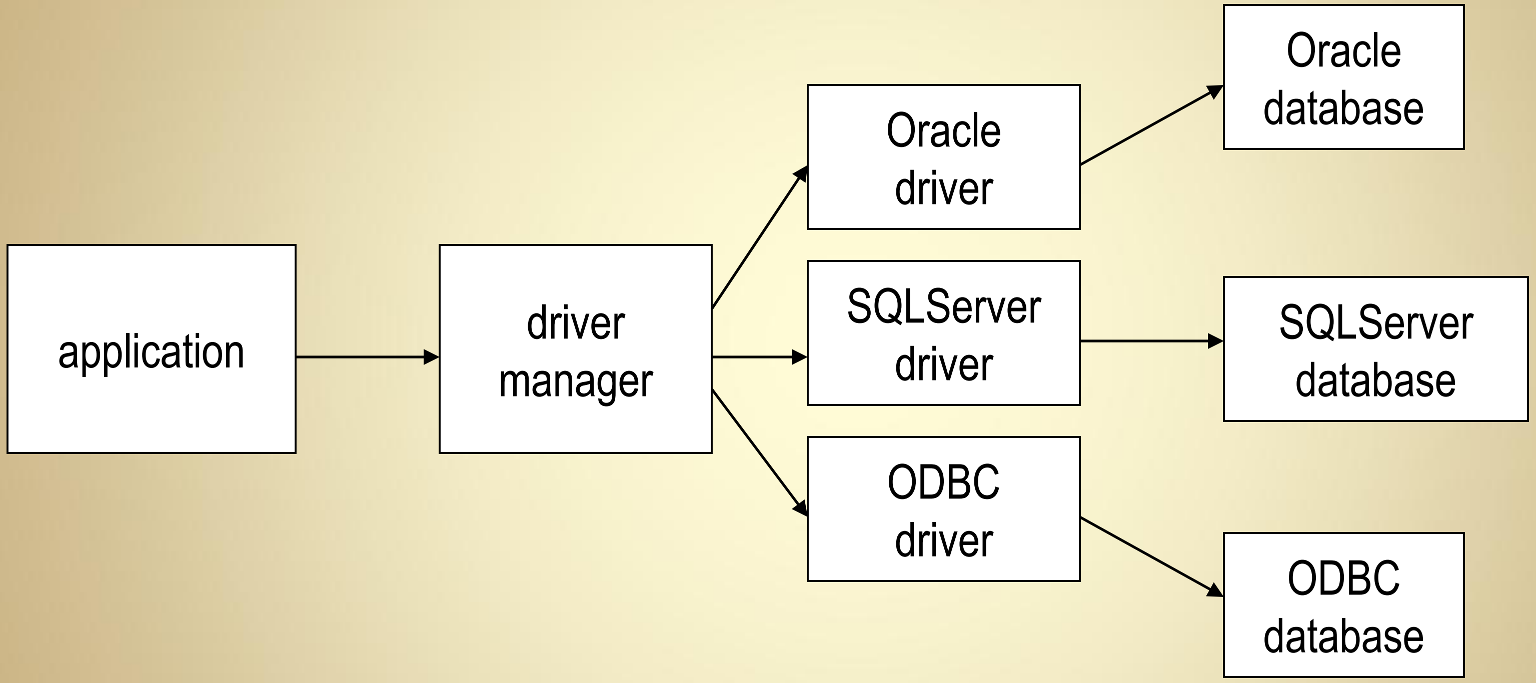
- In a call-level interface (in an appropriate language) SQL can return dynamically sized data structures

- Example: JDBC defines a `ResultSet` class
  - contains meta-data describing the result
  - rows accessed by iteration,
    similar to other Java collections

JDBC

- Call-level interface (CLI)
- Can be used with any DBMS that has a JDBC driver
- SQL statement is constructed at run time as a Java string
- JDBC passes SQL statements to the underlying DBMS and receives result
- Result returned as an instance of ResultSet
- Additional objects handle connections, transactions, etc.

JDBC Run-Time Architecture



using appropriate driver allows generic JDBC commands to be implemented with correct functionality for a particular DBMS

Setting Up JDBC Driver (MySql)

- Download Connector/J 5.1
  - http://dev.mysql.com/downloads/connector/j/5.1.html
- unzip and find mysql-connector-java-5.1.6-bin.jar
- put jar in a convenient place (C:\sql\)
- add jar to your classpath
  - java -classpath .;c:\sql\mysql-connector-java-5.1.6-bin.jar mymain.java

be sure to include current
directory in classpath

JDBC: Connecting to a DB

```
import java.sql.*;

// static method of class loads specified driver
Class.forName(driver_name);

// attempt to connect to DBMS
// If successful, a connection object,
// is created for managing the connection
Connection con =
     DriverManager.getConnection(Url, Id, Passwd);
```

JDBC: Executing a Query

```
// Create a statement object
Statement stat = con.createStatement ();

// Create your SQL command as a string:
String query = "SELECT T.StudId FROM Transcript T" +
               "WHERE   T.CrsCode = 'cse305' " +
               "AND  T.Semester = 'S2000' ";

// execute the statement
ResultSet  res = stat.executeQuery (query);
```

Creates a result set object: res.
Prepares and executes the query.
Stores the result set produced in res (analogous to opening a cursor).
The query string can be constructed at run time.
The input parameters are plugged into the query when the string is formed

Handling Exceptions

```
try {
        ...Java/JDBC code...
}  catch ( SQLException  ex ) {
        …exception handling code...
}
```

- execute all JDBC calls in try blocks

- If an exception is thrown,
  catch the SQLException object

- The exception object has methods to print an error message, return SQLSTATE, etc.

JDBC Example

```
Connection connection = null;
try {
   Class.forName("com.mysql.jdbc.Driver");
   connection = DriverManager.getConnection(
      "jdbc:mysql:///comp163", "mike", "mikepw");
}
catch (SQLException sqlex) {
   sqlex.printStackTrace();
   // abort program?
}
```

`com.mysql.jdbc.Driver` is the name of the Connector/J driver in mysql-connector-java-5.1.6-bin.jar

`jdbc:mysql:///comp163` is the URL of the database on the local machine.

The URL will contain additional information if you are connecting over a network.

JDBC Example

```
ResultSet result_set = null;
try {
   Statement stmt = connection.createStatement();
   String sql_command = "select * from EMPLOYEE;";
   result_set = stmt.executeQuery(sql_command);
}
catch (SQLException sqlex) { … }
```

If `result_set` is not null, it now contains the result of your query.

`executeQuery` is intended for select statements.
It will execute DML commands, but will throw an exception.

`executeUpdate` should be used for DML commands.

JDBC Example

```
Vector attrnames = new Vector();
try {
  ResultSetMetaData metadata = result_set.getMetaData();
  for (int i = 1; i <= metadata.getColumnCount(); ++i)
    attrnames.addElement(metadata.getColumnName( i ));
}
catch (SQLException sqlex) { … }
```

The result set metadata allows you to access the schema of the result.

In this example, names of the columns/attributes are stored in a vector.

JDBC Example

```
Vector tuples = new Vector();
try {
  while (result_set.next())
  {
    Vector data = new Vector();
    for (int i = 1; i <= metadata.getColumnCount(); ++i)
    {
      data.addElement(result_set.getString(i));
    }
    tuples.addElement(data);
  }
}
catch (SQLException sqlex) { … }
```

Move the cursor through the result set by calling next().

In this case, we're extracting all values as strings, other types are possible.

JDBC Example

```
try {
  connection.close();
}
catch (SQLException sqlex) { … }
```

Close the connection when done accessing the database.

JDBC → ODBC

- Open Database Connectivity
  is a standard interface for database connections.
- ODBC is best way to connect to Access databases.
  - set up an ODBC connection to you database
  - start → Programs → Administrative Tools → Data Sources (ODBC)
- JDBC can now connect to Access through ODBC

```
Connection connection = null;
try {
   Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
   connection = DriverManager.getConnection(
      "jdbc:odbc:employee_db");
}
catch (SQLException sqlex) { … }
```

database name
assigned through
ODBC setup

PHP and MySql

- PHP is a dynamic web page language
  - PHP is embedded in HTML code
  - PHP has native library for MySql connections

```
// Connect to the database.
// Parameters are (-h, -u, -p)
$dbc = mysql_connect('localhost', 'mike', 'mikepw');
if ($dbc == null)
   die('<p>connection to mysql failed because:<br>' .
         mysql_error() . '</br></p>');
```

PHP variable names begin with $.
Variable d o not need to be declared.

`die` is a function to print message and terminate program.
Note that the message contains HTML format.

PHP and MySql

```
// Select a database
if (!@mysql_select_db('comp163'))
  die('<p>selection of database failed because:<br>' .
       mysql_error() . '</br></p>');

// Run a query. Result set is $r.
$query = 'select * from department';
$r = mysql_query($query);
if ($r == null)
  die('<p>database query failed because:<br>' .
       mysql_error() . '</br></p>');
```

PHP and MySql

```
// Iterate through result set,
// and print each row into the table.
while ($row = mysql_fetch_array($r))
{
      print '<tr>';
      print "<td>{$row['Dname']}</td>";
      print "<td>{$row['Dnumber']}</td>";
      print "<td>{$row['Mgr_ssn']}</td>";
      print '</tr>';
}

// close the database connection
mysql_close();
```

Iterate through result using mysql_fetch_array.

Retrieve attribute values from each row
by indexing with the attribute names.