13.1 Relational Databases

- A *relational database* is a collection of tables of data, each of which has one special column that stores the primary keys of the table

- Designing a relational database for used Corvettes that are for sale

  - Could just put all data in a single table, whose key would be a simple sequence number

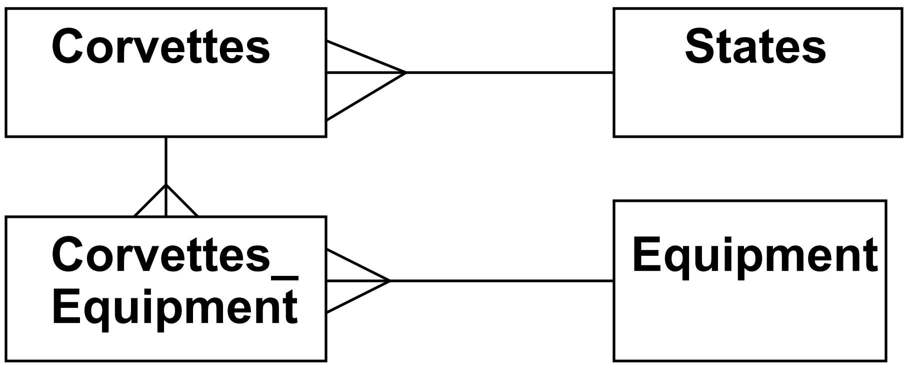
  - The table could have information about various equipment the cars could have

    - Better to put the equipment in a different table and use a cross-reference table to relate cars to equipment

  - To save space, use a separate table for state names, with only references in the main table

13.1 Relational Databases (continued)

- Logical model



- Implementation

Vette_id	Body_style	Miles	Year	State
1	coupe	18.0	1997	4
2	hatchback	58.0	1996	7
3	convertible	13.5	2001	1
4	hatchback	19.0	1995	2
5	hatchback	25.0	1991	5
6	hardtop	15.0	2000	2
7	coupe	55.0	1979	10
8	convertible	17.0	1999	5
9	hardtop	17.0	2000	5
10	hatchback	50.0	1995	7

Figure 13.2 The `Corvettes` table

13.1 Relational Databases (continued)

State_id	State
1	Alabama
2	Alaska
3	Arizona
4	Arkansas
5	California
6	Colorado
7	Connecticut
8	Delaware
9	Florida
10	Georgia

Figure 13.3 The `States` table

Equip_id	Equipment
1	Automatic
2	4-speed
3	5-speed
4	6-speed
5	CD
6	leather

Figure 13.4 The `Equipment` table

13.1 Relational Databases (continued)

Vette_id	Equip
1	1
1	5
1	6
2	1
2	5
2	6
3	1
3	6
4	2
4	6
5	1
5	6
6	2
7	4
7	6
8	4
8	5
8	6
9	4
9	5
9	6
10	1
10	5

Figure 13.5 The `Corvettes-Equipment` cross-reference table

13.2 Intro to SQL

- SQL is a language to create, query, and modify relational databases

- More like structured English than a programming language

- We cover only six basic commands: `CREATE TABLE`, `SELECT`, `INSERT`, `UPDATE`, `DELETE`, and `DROP`

- SQL reserved words are case insensitive

- The `SELECT` Command

  - Used to specify queries
    - Three clauses: `SELECT`, `FROM`, and `WHERE`

  - General form:

    `SELECT` *column names*
    `FROM` *table names*
    `WHERE` *condition*

  - Example:

```
SELECT Body_style FROM Corvettes
   WHERE Year > 1994
```

13.2 Intro to SQL (continued)

- *Joins*

  - If you want all cars that have CD players, you need information from two tables, `Corvettes` and `Equipment`

  - `SELECT` can build a temporary table with data from two tables, from which the desired results can be gotten - this is called a *join* of the two tables

  - A `SELECT` that does a join operation specifies two or more tables in its `FROM` clause and also has a compound `WHERE` clause

  - For our example, to specify cars with CD players, we must have three `WHERE` conditions

    1. The `Vette_id`s column from the `Corvettes` table and the `Corvettes_Equipment` table must match

    2. The `Equip` column from the `Corvettes_Equipment` table must match the `Equip_id` column from the `Equipment` table

    3. The `Equip` column from the `Equipment` table must have the value `'CD'`

13.2 Intro to SQL (continued)

- *Joins* (continued)

```
 SELECT Corvettes.Vette_id,
    Corvettes.Body_style, Corvettes.Miles,
    Corvettes.Year, Corvettes.State,
    Equipment.Equip
FROM Corvettes, Equipment,
        Corvettes_Equipment
WHERE Corvettes.Vette_id =
                Corvettes_Equipment.Vette_id
  AND Corvettes_Equipment.Equip =
                         Equipment.Equip_id
  AND Equipment.Equip = 'CD'
```

This query produces:

```
VETTE_ID BODY_STYLE  MILES YEAR  STATE EQUIP.

1        coupe       18.0  1997  4     CD
2        hatchback   58.0  1996  7     CD
8        convertible 17.0  1999  5     CD
9        hardtop     17.0  2000  5     CD
10       hatchback   50.0  1995  7     CD
```

- To get the state's names:
  1. Replace `Corvettes.State` with `States.State` in the `SELECT` clause
  2. Add `States` to the `FROM` clause
  3. Add `AND Corvettes.State_id = States.State_id` to the `WHERE` clause

13.2 Intro to SQL (continued)

- The `INSERT` Command

`INSERT INTO` *table_name* `(`*col_name*$_1$, … *col_name*$_n$`)`
`VALUES (`*value*$_1$, …, *value*$_n$`)`

  - The correspondence between column names and values is positional

```
INSERT INTO Corvettes(Vette_id,
                Body_style, Miles, Year, State)
VALUES (37, 'convertible', 25.5, 1986, 17)
```

- The `UPDATE` Command

  - To change one or more values of a row in a table

`UPDATE` *table_name*
`SET` *col_name*$_1$ = *value*$_1$,
…
    col_name$_n$ = *value*$_n$
`WHERE` *col_name* = *value*

  - The `WHERE` clause is the primary key of the row to be updated

13.2 Intro to SQL (continued)

- The `UPDATE` Command (continued)

  - Example:

```
      UPDATE Corvettes
      SET Year = 1996
      WHERE Vette_id = 17
```

- The `DELETE` Command

  - Example:

```
 DELETE FROM Corvettes
 WHERE Vette_id = 27
```

  - The `WHERE` clause could specify more than one row of the table

- The `DROP` Command

  - To delete whole databases or complete tables

```
  DROP (TABLE | DATABASE) [IF EXISTS] name
  DROP TABLE IF EXISTS States
```

13.2 Intro to SQL (continued)

- The `CREATE TABLE` command:

`CREATE TABLE` *table_name* `(`
  column_name$_1$ *data_type constraints*,
  …
  column_name$_n$ *data_type constraints*`)`

  - There are many different data types (`INTEGER`, `REAL`, `CHAR(`length`)`, …)

  - There are several constraints possible

    e.g., `NOT NULL`, `PRIMARY KEY`

```
CREATE TABLE States (
   State_id  INTEGER  PRIMARY KEY  NOT NULL,
   State CHAR(20))
```

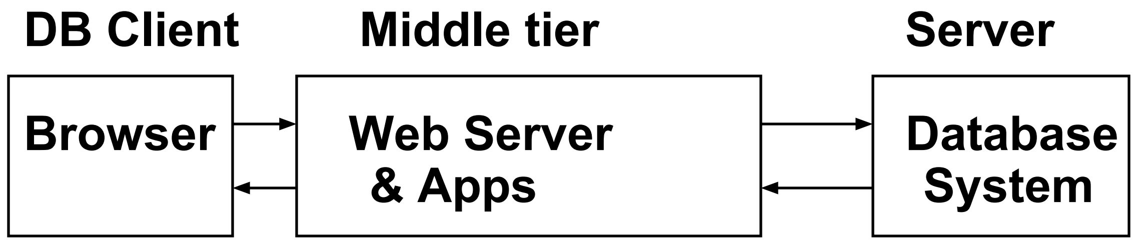
13.3 Architectures for Database Access

- *Client-Server Architectures*
  - Client tasks:
    - Provide a way for users to submit queries
    - Run applications that use the results of queries
    - Display results of queries
  - Server tasks:
    - Implement a data manipulation language, which can directly access and update the database
  - A two-tier system has clients that are connected directly to the server
  - Problems with a two-tier system:
    - Because the relative power of clients has grown considerably, we could shift processing to the client, but then keeping all clients current with application updates is difficult

13.3 Architectures for Database Access (continued)

- A solution to the problems of two-tier systems is to add a component in the middle - create a three-tier system

- For Web-based database access, the middle tier can run applications (client just gets results)



13.3 Architectures for Database Access (continued)

- *Microsoft Open Database Connectivity* (ODBC)
  - ODBC is an API for a set of objects and methods that are an interface to different databases
  - Database vendors provide ODBC drivers for their products – the drivers implement the ODBC objects and methods
  - An application can include SQL statements that work for any database for which a driver is available

13.3 Architectures for Database Access (continued)

- *PHP & Database Access*

  - An API for each specific database system

  - Also convenient for Web access to databases, because PHP is run on the Web server

- *The Java JDBC Architecture*

  - Related to ODBC

  - JDBC is a standard protocol that can be implemented as a driver for any database system

  - JDBC allows SQL to be embedded in Java applications, applets, and servlets

  - JDBC has the advantage of portability over embedded SQL

  - A JDBC application will work with any database system for which there is a JDBC driver

13.4 The MySQL Database System

- A free, efficient, widely used SQL implementation

- Available from `http://www.mysql.org`

- Logging on to MySQL (starting it):

  `mysql` [`-h` host] [`-u` username] [database name] [`-p`]

  - Host is the name of the MySQL server

    - Default is the user's machine

  - Username is that of the database

    - Default is the name used to log into the system

  - The given database name becomes the "focus" of MySQL

- If you want to access an existing database, but it was not named in the `mysql` command, you must choose it for focus

```
use cars;
```

  - Response is: `Database changed`

13.4 The MySQL Database System (continued)

- If the focus has not been set and MySQL gets an SQL command, you get:

```
ERROR 1046: No Database Selected
```

- To create a new database,

```
CREATE DATABASE cars;
```

  - Response:

```
      Query ok, 1 row affected (0.05 sec)
```

- Example:

```
CREATE TABLE Equipment
        (Equip_id INT UNSIGNED NOT NULL
                  AUTO_INCREMENT PRIMARY KEY,
         Equip  INT UNSIGNED
         );
```

- To see the tables of a database:

```
SHOW TABLES;
```

- To see the description of a table (columns):

```
DESCRIBE Corvettes;
```

13.5 Database Access with PHP/MySQL

- When values from a DB are to be put in HTML, you must worry about HTML special characters
  - To get rid of the HTML special characters, use the PHP function, `htmlspecialchars($str)`
    - Replaces the special characters in the string with their corresponding HTML entities
- Another problem with PHP and HTML forms is the string special characters (`'`, `"`, `\`, and `NULL`), which could come from `$_GET` and `$_POST`
  - To fix these, `magic_quotes_gpc` in the `PHP.ini` file is set to `ON` by default
    - This backslashes these special characters

```
$query = "SELECT * FROM Names
          WHERE Name = $name";
```

- If this wasn't done and the value of `$name` is `O'Shanter`, it would prematurely terminate the query string
- But with `magic_quotes_gpc` on, it will be converted to `O\'Shanter`

- Unfortunately, this can create new problems

13.5 Database Access with PHP/MySQL (continued)

- For example, if a `SELECT` clause has a single-quoted part, like `'California'`, the single quotes will be implicitly backslashed, making the query illegal for MySQL

- So, `magic_quotes_gpc` must be turned off, or else the extra backslashes can be removed with `stripslashes`, as in:

```
$query = stripslashes($query);
```

- To connect PHP to a database, use `mysql_pconnect`, which can have three parameters:

  1. host (default is localhost)

  2. Username (default is the username of the PHP script)

  3. Password (default is blank, which works if the database does not require a password)

```
$db = mysql_pconnect();
```

  - Usually checked for failure

- Sever the connection to the database with `mysql_close`

13.5 Database Access with PHP/MySQL (continued)

- To focus MySQL,

```
mysqli_select_db("cars");
```

- Requesting MySQL Operations

  - Call `mysql_query` with a string parameter, which is an SQL command

```
$query = "SELECT * from States";
$result = mysqli_query($db, $query);
```

- Dealing with the result:

  - Get the number of rows in the result

```
$num_rows = mysqli_num_rows($result);
```

  - Get the number of fields in the result

```
$num_fields = mysqli_num_fields($result);
```

  - Get a row of the result

```
$row = mysqli_fetch_assoc($result);
```

13.5 Database Access with PHP/MySQL (continued)

- Display the column names

```
 $keys = array_keys($row);
 for ($index = 0; $index < $num_fields;
       $index++)
   print $keys[$index] . "  ";
 print "<br />";
```

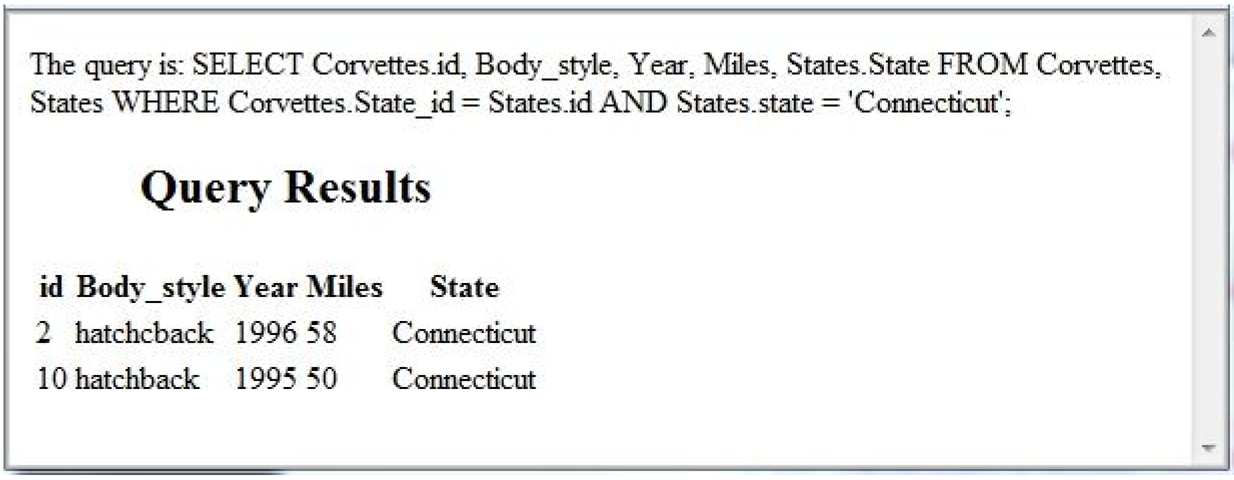
- Display the values of the fields in the rows

```
for ($row_num = 0; $row_num < $num_rows;
    $row_num++) {
 $values = array_values($row);
 for ($index = 0; $index < $num_fields;
        $index++) {
   $value = htmlspecialchars($values[$index]);
   $row = mysql_fetch_array($result);
   print $value . "  ";
 }
 print "<br />";
 $row = mysqli_fetch_assoc($result);
}
```

13.5 Database Access with PHP/MySQL (continued)

→ SHOW `carsdata.html`

→ SHOW `access_cars.php`



```
The query is: SELECT Corvettes.id, Body_style, Year, Miles, States.State FROM Corvettes,
States WHERE Corvettes.State_id = States.id AND States.state = 'Connecticut';

        Query Results

id Body_style Year Miles    State
2  hatchcback 1996 58     Connecticut
10 hatchback  1995 50     Connecticut
```

13.5 Database Access with PHP/MySQL (continued)

- The form display document and the PHP processing document can be combined

- After simply inserting the HTML from the display document into the PHP document, several modifications are required:

  1. Change the value of the `action` attribute of the form to the name of the combined document file

  2. Create a hidden input element that sets its value when the document is first displayed. This provides a way for the document to determine which it is doing, displaying the form or processing the form data

```
<input type = "hidden"  name = "stage"
       value = "1" />
```

The PHP code to test this has the form:

```
$stage = $_POST["stage"];
if (!IsSet($stage))) { … }
```

The then clause includes the form processing; the else clause includes the form display

→ SHOW `access_cars2.php`

13.6 Database Access with JDBC/MySQL

- JDBC is a Java API for database access
- The API is defined in `java.sql` (part of Java distribution)
- *JDBC and MySQL*
  - A database driver for MySQL, as well as other common databases, is included with NetBeans
  - Connecting the application to the driver
    - The `getConnection` method of `DriverManager`, which selects the correct driver
    - This method takes three parameters: a reference to the host and the database, the user, and the password for the database, if there is one

13.6 Database Access with JDBC/MySQL (continued)

- For MySQL and the `cars` database, which resides on the user machine, the reference to the host and database is:

```
jdbc:mysql://localhost/cars
```

- The user for us is `root`

- We assume there is no password for the database, so we can use the empty string

```
myCon = DriverManager.getConnection(
   "jdbc:mysql://localhost/cars", "root", "");
```

- The connection object is used to specify all database operations from the servlet

- SQL commands through JDBC

  - First, you need a `Statement` object

```
Statement myStmt = myCon.createStatement();
```

13.6 Database Access with JDBC/MySQL (continued)

- SQL commands are `String` objects

```
final String sql_com = "UPDATE Corvettes  SET" +
 "Year = 1991 WHERE Vette_id = 7");
```

- Categories of SQL commands

  - Action - `INSERT`, `UPDATE`, `DELETE`, `CREATE TABLE`, and `DROP TABLE`

  - Query - `SELECT`

- The action commands are executed with the `executeUpdate` method of `Statement`

```
  myStmt.executeUpdate(sql_com);
```

  - Returns the number of affected rows

- A `SELECT` is executed by sending it as the actual parameter to the `executeQuery` method of `Statement`

  - The `executeQuery` method returns an object of class `ResultSet`

    - Get rows from `ResultSet` with `next` iterator

13.6 Database Access with JDBC/MySQL (continued)

```
ResultSet result;
final String sql_com =
  "SELECT * FROM Corvettes WHERE Year <= 1990"
result = myStmt.executeQuery(sql_com);

while(result.next()) {
   // access and process the current element
}
```

- Information is extracted from the `ResultSet` object with an access method, for which there is one for each data type

  e.g., If an extracted row is

```
3, "convertible", 13.5, 2001, 1

String style;
style = result.getString("Body_style");
```

  or

```
style = result.getString(2);
```

13.6 Database Access with JDBC/MySQL (continued)

- Metadata - to get table and column names from a database

- Two kinds:

  1. Metadata that describes the database

  2. Metadata that describes a `ResultSet` object

- A `Connection` method, `getMetaData`, creates an object of class `DatabaseMetaData`

```
DatabaseMetaData dbmd = myCon.getMetaData();
```

13.6 Database Access with JDBC/MySQL (continued)

- The `getTables` method of `DatabaseMetaData` takes four parameters, only one of which is necessary

```
String tbl[] = {"TABLE"};
DatabaseMetaData dbmd = myCon.getMetaData();
result = dbmd.getTables(
                    null, null, null, tbl);
System.out.println(
   "The tables in the database are: \n\n");
while (result.next()) {
   System.out.println(result.getString(3));
}
```

- Output from this:

```
The tables in this database are:

    CORVETTES
    CORVETTES_EQUIPMENT
    EQUIPMENT
    STATES
```

- Metadata about query results has a different structure than general database metadata
  - `ResultSetMetaData` object

13.6 Database Access with JDBC/MySQL (continued)

```
ResultSetMetaData resultMd =
                        result.getMetaData();
```

- We can get the number of columns, their names, types, and sizes from the `resultMd` object, using its methods

  - `getColumnCount` returns the number of columns

  - `getColumnLable(i)` returns the ith column's name

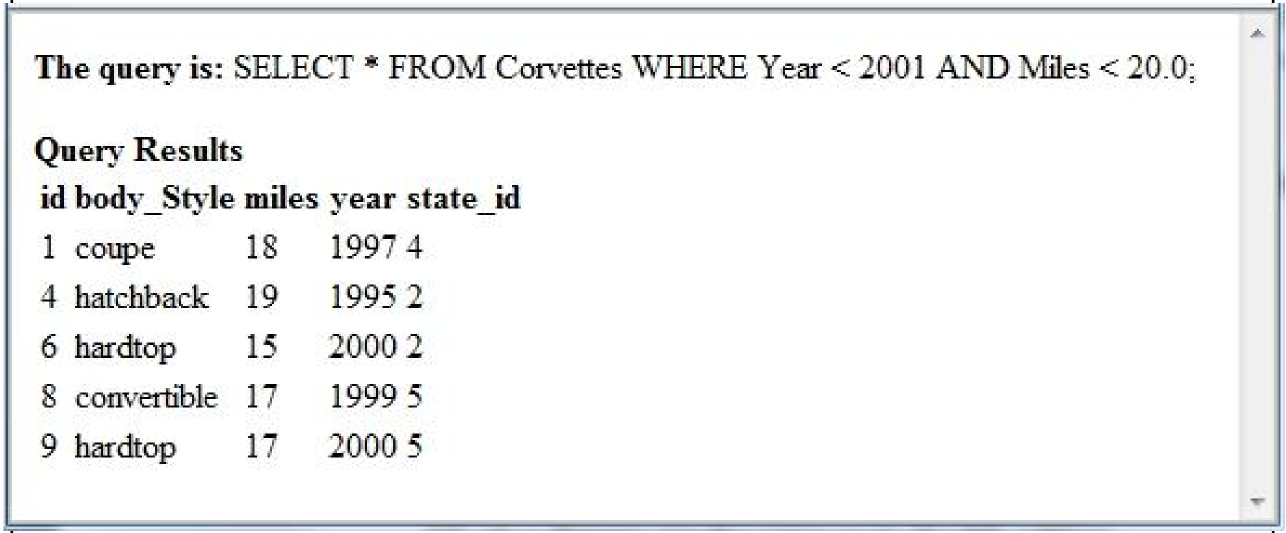
```
// Create an object for the metadata
ResultSetMetaData resultMd =
      result.getMetaData();

// Loop to fetch and display the column names
for (int i = 1; i <= resultMd.getColumnCount();
     i++) {

  String columnName =
                 resultMd.getColumnLabel(i);
  System.out.print(columnName + "\t");
}
System.out.println("\n");
```

Output:

```
  Vette_id  Body_style  Miles  Year   State
```

13.6 Database Access with JDBC/MySQL (continued)

→ SHOW `JDBCServlet.java`



The query is: SELECT * FROM Corvettes WHERE Year < 2001 AND Miles < 20.0;

Query Results

id	body_Style	miles	year	state_id
1	coupe	18	1997	4
4	hatchback	19	1995	2
6	hardtop	15	2000	2
8	convertible	17	1999	5
9	hardtop	17	2000	5

13.7 Database Access with ASP.NET and MySQL

- ADO.NET is a library of classes for database management
  - We cover only a small part of it
- Most commercial ASP.NET database applications use SQL Server
- Fundamental aim of ADO.NET is to provide a relationship between markup controls and some data source, internal (e.g., an array) or external (e.g., a database)
- ADO.NET maps controls to the form of the data
  - The data can be manipulated and displayed
- *ADO.NET has two parts*:
  - *Connected part*:
    - Classes to connect to the DB
    - Classes that transmit commands to the data
    - Classes that move data from the source to the application

13.7 Database Access with ASP.NET and MySQL (continued)

- *Disconnected part*:
  - Classes that represent the data that is visible in the application

- Three kinds of classes for the connected part:
  - Connections – one class for each DB vendor
  - Commands – Also one for each DB vendor
    - `ExecuteReader` for `SELECT` commands
    - `ExecuteNonQuery` for non-`SELECT` commands
    - `ExecuteScalar` for `SELECT` commands that return single values
    - Data readers – later……
  - It is more difficult to change an application to use a different vendor with ASP.NET than it is with JDBC
  - We use MySQL, as with the other PHP and JDBC

13.7 Database Access with ASP.NET and MySQL (continued)

- Data-bound controls – data is fetched from a data source and bound to the properties of server controls

- The concept of binding data to markup controls is a significant difference between ADO.NET and JDBC

- Actual data binding is requested with the `DataBind` method of the object that represents the control

  - The `DataSource` property of the control specifies the data source

```
myControl.DataSource = data from some data
                       reader method;
myControl.DataBind();
```

- The `GridView` data-bound control – column-based

  - For relational database sources, the columns are columns of a database table

  - `GridView` has a large collection of properties that allow the developer to have extensive control over the appearance and behavior of the data

13.7 Database Access with ASP.NET and MySQL (continued)

- The information required to connect an ASP.NET source document to a database is stored in a *connection string*

  - A connection string contains information about the server, the specific database, the user id, and the password of the database, if there is one

```
"server=localhost;Database=cars;uid=root"
```

- The source of a driver for ASP.NET and MySQL is

http://dev.mysql.com/downloads/connector/net

  - A developer must download the driver and install it

- *An example – the same one*

  - *Needs*:

    - A text box to collect a `SELECT` command from the user

    - A label element for displaying an error message

    - A `GridView` control to store and display the result from executing the `SELECT` command

13.7 Database Access with ASP.NET and MySQL (continued)

- *The code-behind file*:

  - Must define a string constant for the connection string

  - Must define two methods:

    - One that is a handler for the `Load` event
      - When `IsPostBack` is true, it calls the other method

    - The other one executes the `SELECT` command when called by the `Load` event handler

      - First, create the connection object by calling the connection constructor
      - Second, create the command object by calling the `CreateCommand` method
      - Third, assign the command to the connection object
      - Last, assign the connection string to the `ConnectionString` property of the connection object

```
MySqlConnection con = new MySqlConnection();
 MySqlCommand cmd = con.CreateCommand();
cmd.CommandText = command;
con.ConnectionString = ConnStr;
```

13.7 Database Access with ASP.NET and MySQL (continued)

- *The code-behind file* (continued)

  - Next, call the `Open` method on the connection

```
con.Open();
```

  - Then, call the `ExecuteReader` method of the command object

    (although `ExecuteReader` has several optional parameters, we send just one)

    - The type of the return value is `MySqlDataReader`

    - The return value is assigned to the `DataSource` attribute of the `GridView` control, `results`

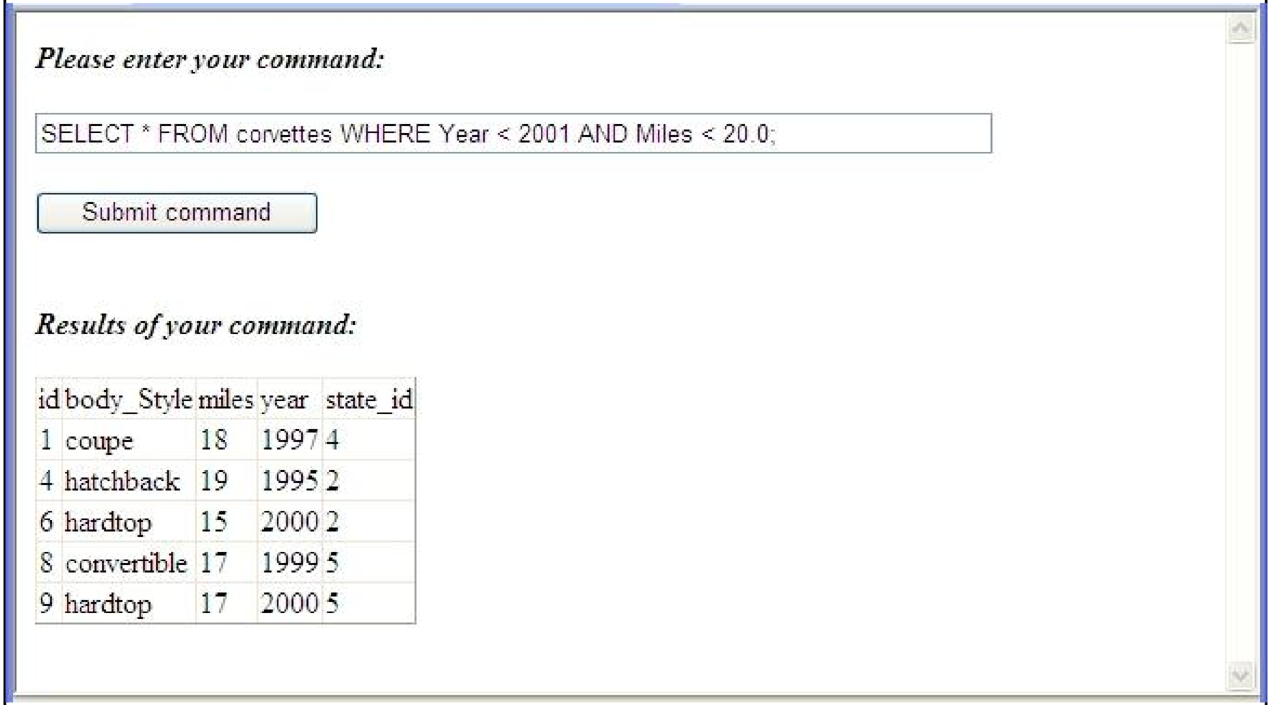
```
MySqlDataReader reader = cmd.ExecuteReader(
            CommandBehavior.CloseConnection);
results.DataSource = reader;
```

  - Finally, `DataBind` must be called

```
results.DataBind();
```

→ **SHOW** `sqlcars.aspx` **and** `sqlcars.aspx.cs`

13.7 Database Access with ASP.NET and MySQL (continued)



Please enter your command:

SELECT * FROM corvettes WHERE Year < 2001 AND Miles < 20.0;

Submit command

Results of your command:

id	body_Style	miles	year	state_id
1	coupe	18	1997	4
4	hatchback	19	1995	2
6	hardtop	15	2000	2
8	convertible	17	1999	5
9	hardtop	17	2000	5