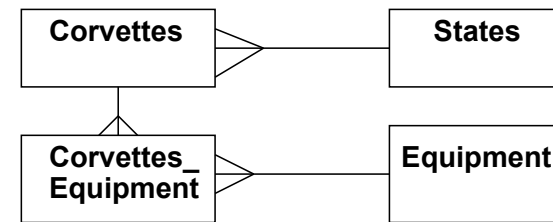


## 13.1 Relational Databases

- A database is a collection of data organized to allow relatively easy access for retrievals, additions, and deletions
- 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
  - Rows are sometimes called *entities*
- 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
  - 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

- A standard language to create, query, and modify databases
  - Supported by all major database vendors
- 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 CREATE TABLE command:

```
CREATE TABLE table_name (  
    column_name1 data_type constraints,  
    ...  
    column_namen data_type constraints)
```

- There are many different data types (INTEGER, FLOAT, CHAR (length), ...)

## 13.2 Intro to SQL (continued)

- There are several constraints possible

e.g., NOT NULL, PRIMARY KEY

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

- The SELECT Command

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

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

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

## 13.2 Intro to SQL (continued)

### - The INSERT Command

```
INSERT INTO table_name (col_name1, ... col_namen)  
VALUES (value1, ..., valuen)
```

### - 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_name1 = value1,  
...  
    col_namen = valuen  
WHERE col_name = value
```

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

## 13.2 Intro to SQL (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)

### - 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 info 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 tables in its **FROM** clause and also has a compound **WHERE** clause
- For our example, we must have three **WHERE** conditions

1. **Vette\_id**s from *Corvettes* and *Corvettes\_Equipment* must match

2. **Equip** from *Corvettes\_Equipment* must match the **Equip\_id** from *Equipment*

3. The **Equip** from *Equipment* must be CD

## 13.2 Intro to SQL (continued)

```
SELECT Corvettes.Vette_id,  
       Corvettes.Body_style, Corvettes.Miles,  
       Corvettes.Year, Corvettes.State,  
       Equipment.Equip  
FROM 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     |

## 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

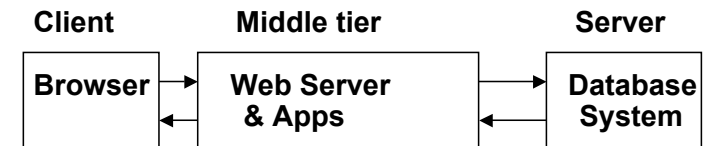
#### - Server tasks:

- 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 maintaining data integrity 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)



### - *Database Access with Embedded SQL*

- SQL commands are embedded in programs written in a host programming language, whose compiler is extended to accept some form of SQL commands
- *Advantage:*
  - One package has computational support of the programming language, as well as database access with SQL

### **13.3 Architectures for Database Access** (continued)

- *Disadvantage (of embedded SQL):*
  - Portability among database systems
- *Microsoft Access Architecture*
  - A tool to access any common database structure
  - Use either the Jet database engine, or go through the Other Database Connectivity (ODBC) standard
    - 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)

- *The Perl DBI/DBD Architecture*
  - Database Interface (DBI) provides methods & attributes for generic SQL commands
  - Database Driver (DBD) is an interface to a specific database system (MySQL, Oracle, etc.)
  - Convenient for Web access to databases, because the Perl program can be run as CGI on the Web server system
- *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

### 13.3 Architectures for Database Access (continued)

- *The Java JDBC Architecture*
- Related to both embedded languages and 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 it is an existing database, but it was not named in the `mysql` command, you must choose one on which to 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 Perl/MySQL

- *Needed:*

1. DBI – a standard object-oriented module
2. A DBD for the specific database system

- *DBI Module*

- Get complete documentation from perldoc DBI

- Interface is similar to Perl's interface to external files – through a filehandle

- To provide access to DBI and create a DBI object:

```
use DBI;
```

- Access to the object is through the reference variable, DBI

- To connect to the database:

```
$dbh = DBI->connect(  
    "DBI:driver_name:database_name" [, username]  
    [, password]);
```

## 13.5 Database Access with Perl/MySQL (continued)

### - Example:

```
$dbh = DBI->connect("DBI:mysql:cars");
```

- Creates the db handle
- Assumes the user name of the person logged in
- Assumes the db does not need a password

- The `connect` method is usually used with `die`

- A Perl program can have connections to any number of databases

- To create a query, we usually compile the SQL command first, then use it against the database

- To create a compiled query, use `prepare`, as in:

```
$sth = $dbh->prepare("SELECT Vette_id,  
    Body_style, Year, States.State  
FROM Corvettes, States  
WHERE Corvettes.State = States.State_id  
    AND States.State = 'California'");
```

- To execute a compiled query, use `execute`, as in:

```
$sth->execute() or  
die "Error -query: $dbh->errstr\n";
```

## 13.5 Database Access with Perl/MySQL (continued)

- The `$sth` object now has the result of the query

- To display the results, we would like column names, which are stored in a hash

```
$col_names = $sth->{NAME};
```

- Rows of the result are available with the `fetchrow_array` method, which returns a reference to an array that has the next row of the result ( returns false if there are no more rows)

- Note: Putting query results in an HTML document can cause trouble (>, <, ", and &)

- Avoid the problem by using the CGI function, `escapeHTML`

→ SHOW `access_cars.pl`

The query is: `SELECT * FROM Corvettes, States WHERE Corvettes.State = States.State_id AND States.State = "California"`

### Query Results

| Vette_id | Body_style  | Miles | Year | State | State_id | State      |
|----------|-------------|-------|------|-------|----------|------------|
| 5        | hatchback   | 25.0  | 1991 | 5     | 5        | California |
| 8        | convertible | 17.0  | 1999 | 5     | 5        | California |
| 9        | hardtop     | 17.0  | 2000 | 5     | 5        | California |

## 13.5 Database Access with PHP/MySQL (continued)

- To get rid of special characters, use the PHP function, `htmlspecialchars($str)`
  - Replaces the special characters in the string with HTML entities
- Another problem with PHP and HTML forms is the string special characters (" , ' , \, and NULL)
- To fix this, `magic_quotes_gpc` in the PHP.ini file is set to ON by default

- This backslashes the special characters

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

- If the value of `$name` is "O'Shanter"
  - it will prematurely terminate the query string
- With `magic_quotes_gpc` on, it will be converted to "O\'Shanter"
- This creates a new problem – the backslashes can cause trouble

## 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`
- 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 used with `die`
- Sever the connection to the database with `mysql_close`

## 13.5 Database Access with PHP/MySQL (continued)

- To focus MySQL,

```
mysql_select_db("cars");
```

- Requesting MySQL Operations

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

```
$query = "SELECT * from States";  
$result = mysql_query($query);
```

- Dealing with the result:

### 1. Get the number of rows in the result

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

### 2. Get the rows with `mysql_fetch_array`

```
for ($row_num = 0; $row_num < $num_rows;  
    $row_num++) {  
    print "<p> Result row number" .  
        ($row_num + 1) .  
        ". State_id: ";  
    print htmlspecialchars($row["State_id"]);  
    print "State: ";  
    etc.
```

## 13.5 Database Access with PHP/MySQL (continued)

- We have had it easy – the column titles were known

- If they are not known, we must get them

- The result rows are in a PHP array, whose elements actually are double sets of elements

- Each pair has the value, but one has a numeric key and one has a string key

- For example, if the result has the field values (1, Alabama), the array has:

```
((0, 1), (State_id, 1), (1, Alabama),  
 (State, Alabama))
```

- If the row is indexed with numbers, the element values are returned

```
num_fields = sizeof($row);  
for ($field_num = 0;  
    $field_num < $num_fields / 2;  
    $field_num++)  
    print $row[$field_num];
```

- The column names are the odd-numbered elements of the result rows

## 13.5 Database Access with PHP/MySQL (continued)

- To get all column names:

```
while (next_element = each($row)) {  
    $next_element = each($row);  
    $next_key = $next_element['key'];  
    print $next_key . " ";  
}
```

The query is: SELECT Vette\_id, Body\_style, Year, Miles, States.State FROM Corvettes, States WHERE Corvettes.State = States.State\_id AND States.State = 'Connecticut'

### Query Results

| Vette_id | Body_style | Year | Miles | State       |
|----------|------------|------|-------|-------------|
| 2        | hatchback  | 1996 | 58.0  | Connecticut |
| 10       | hatchback  | 1995 | 50.0  | Connecticut |

## 13.7 Database Access with JDBC/MySQL

- Approaches to using JDBC outside the Web
- JDBC is a Java API for database access
- The API is defined in java.sql (part of Java distribution)
- Can use a two-tier configuration
  - **Disadvantage: Every client must have a driver for every database vendor**
- Can also use a three-tier configuration
  - The application runs on the client side, the middle machine runs JDBC, and the third system runs the database system
- *JDBC and MySQL*
- Connecting the application to the driver
  - The `getConnection` method of `DriverManager`, which select the correct driver from those that are registered

## 13.7 Database Access with JDBC/MySQL (continued)

- The general form of a reference to a database for the connection operation is:

```
jdbc:subprotocol_name:more_info
```

- The “subprotocol” specifies the driver

- For the JDBC-ODBC bridge, it is `odbc`
- For the MySQL, it is `mysql`

- The “more info” part depends on the specific database being used

- For MySQL and the cars database,

```
jdbc:mysql://localhost/cars?user=root
```

- Two ways to register a database driver:

1. The general way is to have the system property `jdbc.drivers` maintain a list of registered drivers

- Add one for `mysql` with

```
jdbc.drivers = org.gjt.mm.mysql.Driver;
```

## 13.7 Database Access with JDBC/MySQL (continued)

2. Manual registration, using the `forName` method of the `Class` class, passing the name of the driver

```
Class.forName(  
    "org.gjt.mm.mysql.Driver").newInstance();
```

- The actual connection is made by creating a `Connection` object with the `getConnection` method of the `DriverManager` class

```
DriverManager.getConnection(database_address,  
    database_user_id, password)
```

- If the application owner owns the database, public can be used for both the user id and the password

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

- SQL commands through JDBC

- First, you need a `Statement` object

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

## 13.7 Database Access with JDBC/MySQL (continued)

- SQL commands are `String` objects

```
final String sql_com = "UPDATE Corvettes " +  
"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

## 13.7 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);
```

→ SHOW `Query.java`

## 13.7 Database Access with JDBC/MySQL (continued)

1993-2001 Corvettes For Sale

| 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     |
| 6        | hardtop     | 15.0  | 2000 | 2     |
| 8        | convertible | 17.0  | 1999 | 5     |
| 9        | hardtop     | 17.0  | 2000 | 5     |
| 10       | hatchback   | 50.0  | 1995 | 7     |

- 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();
```

## 13.7 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

## 13.7 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

- `getColumnLabel(i)` returns the column names

```
// 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
```

## 13.7 Database Access with JDBC/MySQL (continued)

- JDBC and Servlets

→ SHOW `JDBCServlet.java`

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

| Query Results |             |       |      |       |
|---------------|-------------|-------|------|-------|
| Vette_id      | Body_style  | Miles | Year | State |
| 1             | coupe       | 18.0  | 1997 | 4     |
| 4             | hatchback   | 19.0  | 1995 | 2     |
| 6             | hardtop     | 15.0  | 2000 | 2     |
| 8             | convertible | 17.0  | 1999 | 5     |
| 9             | hardtop     | 17.0  | 2000 | 5     |