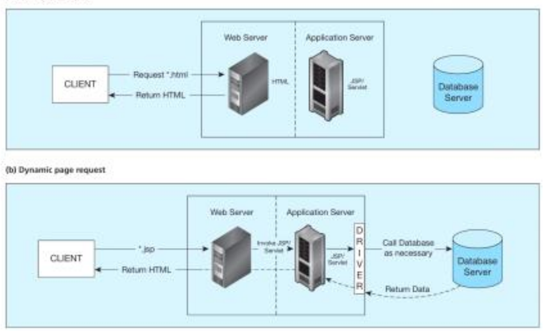
DATABASES IN THREE-TIER APPLICATIONS

Static vs Dynamic page request

FIGURE 7-5 Information flow in a three-tier architecture

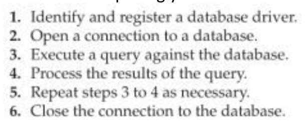
(a) Static page request

(b) Dynamic page request



Note:

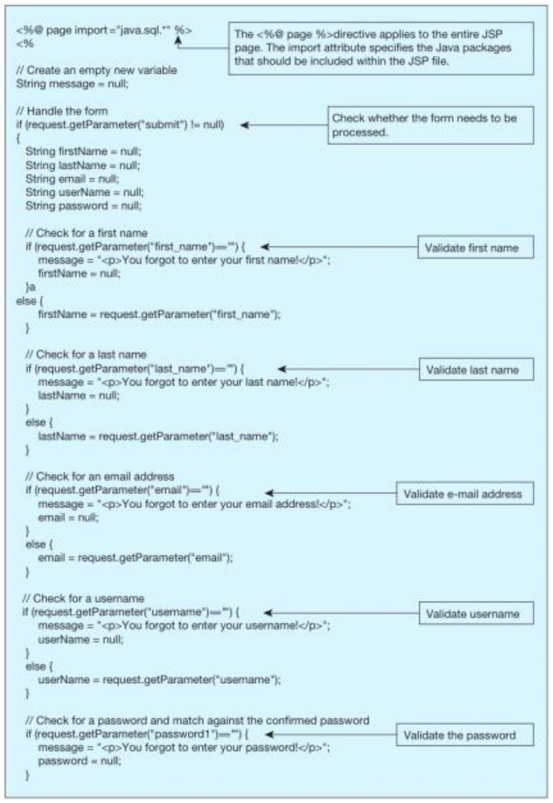
- if the Web server determines that the request from the client can be satisfied without passing the request on to the application server, it will process the request and then return the appropriately formatted information to the client machine.
  - Static
  - This decision is most often based on the file suffix.
- However, if the request has a suffix that requires application server intervention, the information flow shown in Figure 7-5b is invoked.

- In a dynamic page request,
  - The application calls the DBMS, as necessary, using a special software called database-oriented middleware.
  - Middleware is often referred to as the glue that holds together client/server applications.
  - It is a term that is commonly used to describe any software component between the PC client and the relational database in n-tier architectures.
  - middleware is any of several classes of software that allow an application to work with other software without requiring the user to understand and code the low-level operations required to achieve interoperability
    - The database-oriented middleware needed to connect an application to a database consists of two parts: an application programming interface (API) and a database driver to connect to a specific type of database (sql server or oracle)
      - The most common APIs are Open Database Connectivity (ODBC) and ADO.NET for the Microsoft platform (VB.NET and C#) and Java Database Connectivity (JDBC) for use with Java programs.
      - However, no matter which API or language is used, the basic steps for accessing a database from an application remain surprisingly similar:
        - 

1. Identify and register a database driver.
2. Open a connection to a database.
3. Execute a query against the database.
4. Process the results of the query.
5. Repeat steps 3 to 4 as necessary.
6. Close the connection to the database.

A Java Web Application

- This a sample JSP application whose purpose is to capture user registration information and store the data in a database. Let us assume that the name of the page is registration.jsp.



```
<%@ page import="java.sql.*" %>
<%

// Create an empty new variable
String message = null;

// Handle the form
if (request.getParameter("submit") != null)
{
  String firstName = null;
  String lastName = null;
  String email = null;
  String userName = null;
  String password = null;

  // Check for a first name
  if (request.getParameter("first_name")=="") {
     message = "<p>You forgot to enter your first name!</p>";
     firstName = null;
  }a
else {
     firstName = request.getParameter("first_name");
  }

  // Check for a last name
  if (request.getParameter("last_name")=="") {
     message = "<p>You forgot to enter your last name!</p>";
     lastName = null;
  }
  else {
     lastName = request.getParameter("last_name");
  }

  // Check for an email address
  if (request.getParameter("email")=="") {
     message = "<p>You forgot to enter your email address!</p>";
     email = null;
  }
  else {
     email = request.getParameter("email");
  }

 // Check for a username
 if (request.getParameter("username")=="") {
     message = "<p>You forgot to enter your username!</p>";
     userName = null;
  }
  else {
     userName = request.getParameter("username");
  }

  // Check for a password and match against the confirmed password
  if (request.getParameter("password1")=="") {
     message = "<p>You forgot to enter your password!</p>";
     password = null;
  }
```

The <%@ page %>directive applies to the entire JSP page. The import attribute specifies the Java packages that should be included within the JSP file.

Check whether the form needs to be processed.

Validate first name

Validate last name

Validate e-mail address

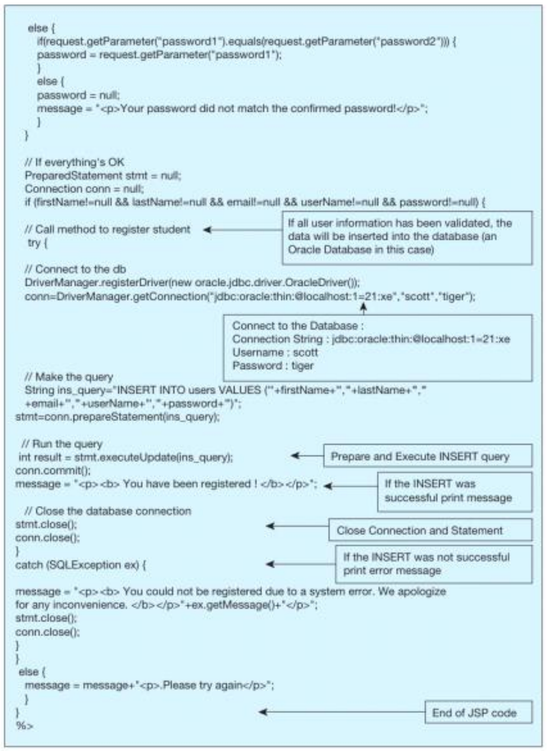
Validate username

Validate the password

This is what this code does:
• Displays the registration form
• Processes a user's filled-in form and checks it for common errors, such as missing items and matching password fields
• If there is an error, redisplays the entire form, with an error message in red
• If there is no error, enters the user's information into a database and sends the user to a "success" screen

Things to note about this java code:

- All Java code is found between <% and %> is not displayed in the browser.
  - This is the part of the code that has nothing to do with the UI it is purely for the logical purposes like validating input, accessing database, etc
- The only items displayed in the browser are the ones enclosed in HTML tags which we will see futher down the program
- The code shows that once the connection is made and stored in the conn variable, the actual SQL query to be issued is constructed as a string variable name ins_query. The conn.prepareStatement and conn.executeQuery commands are then used by the driver to issue the query to the database (in this case insert a record). The conn.commit() statement asks the database to make this change permanent.



```
  else {
    if(request.getParameter("password1").equals(request.getParameter("password2"))) {
    password = request.getParameter("password1");
    }
    else {
    password = null;
    message = "<p>Your password did not match the confirmed password!</p>";
    }
 }

 // If everything's OK
 PreparedStatement stmt = null;
 Connection conn = null;
 if (firstName!=null && lastName!=null && email!=null && userName!=null && password!=null) {

 // Call method to register student
  try {

 // Connect to the db
 DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
 conn=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1=21:xe","scott","tiger");

 // Make the query
 String ins_query="INSERT INTO users VALUES ('"+firstName+"','"+lastName+"','"
 +email+"','"+userName+"','"+password+"')";
stmt=conn.prepareStatement(ins_query);

 // Run the query
 int result = stmt.executeUpdate(ins_query);
conn.commit();
message = "<p><b> You have been registered ! </b></p>";

 // Close the database connection
stmt.close();
conn.close();
}
catch (SQLException ex) {

message = "<p><b> You could not be registered due to a system error. We apologize
for any inconvenience. </b></p>"+ex.getMessage()+"</p>";
stmt.close();
conn.close();
}
}
 else {
  message = message+"<p>.Please try again</p>";
  }
}
%>
```

If all user information has been validated, the data will be inserted into the database (an Oracle Database in this case)

Connect to the Database :
Connection String : jdbc:oracle:thin:@localhost:1=21:xe
Username : scott
Password : tiger

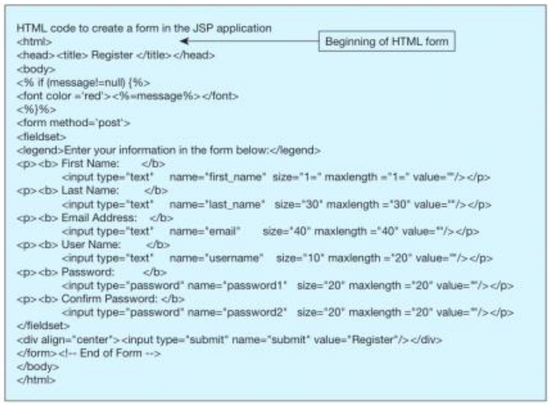
Prepare and Execute INSERT query

If the INSERT was successful print message

Close Connection and Statement

If the INSERT was not successful print error message

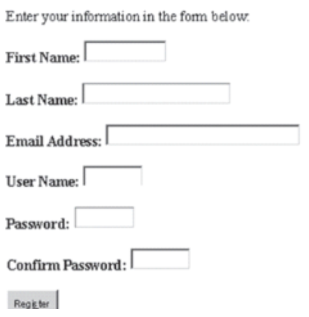
End of JSP code



HTML code to create a form in the JSP application

```
<html>
<head><title> Register </title></head>
<body>
<% if (message!=null) {%>
<font color ='red'><%=message%></font>
<%}%>
<form method='post'>
<fieldset>
<legend>Enter your information in the form below:</legend>
<p><b> First Name:      </b>
          <input type="text"    name="first_name"  size="1=" maxlength ="1=" value=""/></p>
<p><b> Last Name:       </b>
          <input type="text"    name="last_name"   size="30" maxlength ="30" value=""/></p>
<p><b> Email Address:   </b>
          <input type="text"    name="email"       size="40" maxlength ="40" value=""/></p>
<p><b> User Name:       </b>
          <input type="text"    name="username"    size="10" maxlength ="20" value=""/></p>
<p><b> Password:        </b>
          <input type="password" name="password1"  size="20" maxlength ="20" value=""/></p>
<p><b> Confirm Password: </b>
          <input type="password" name="password2"  size="20" maxlength ="20" value=""/></p>
</fieldset>
<div align="center"><input type="submit" name="submit" value="Register"/></div>
</form><!-- End of Form -->
</body>
</html>
```

Beginning of HTML form



Enter your information in the form below:

First Name:

Last Name:

Email Address:

User Name:

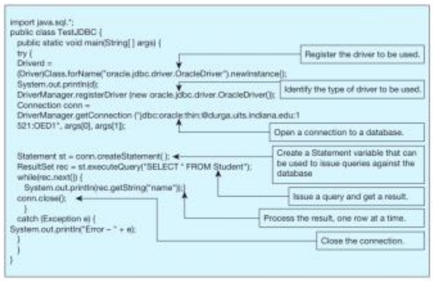
Password:

Confirm Password:

Note: This is html code that lives inside of the .jsp file (java file) which is what displays the actual shit to the screen for the user to click and input into

- This is simple java file that retrieves data from the database:

FIGURE 7-7 Database access from a Java program



```
import java.sql.*;
public class TestJDBC {
  public static void main(String[ ] args) {
  try {
  Driverd =
  (Driver)Class.forName("oracle.jdbc.driver.OracleDriver").newInstance();
  System.out.println(d);
  DriverManager.registerDriver (new oracle.jdbc.driver.OracleDriver());
  Connection conn =
  DriverManager.getConnection ("jdbc:oracle:thin:@durga.uits.indiana.edu:1
  521:OED1", args[0], args[1]);

  Statement st = conn.createStatement( );
  ResultSet rec = st.executeQuery("SELECT * FROM Student");
  while(rec.next()) {
    System.out.println(rec.getString("name"));}
  conn.close();
    }
  catch (Exception e) {
System.out.println("Error – " + e);
  }
  }
}
```

Register the driver to be used.

Identify the type of driver to be used.

Open a connection to a database.

Create a Statement variable that can be used to issue queries against the database

Issue a query and get a result.

Process the result, one row at a time.

Close the connection.

Note:
- Notice that after the connection is opened—unlike the INSERT query shown above—running a SQL SELECT query requires us to capture the data inside an object that can appropriately handle the tabular data returned.
  - JDBC provides two key mechanisms for this: the ResultSet and RowSet objects.
  - The ResultSet object has a mechanism, called the cursor, that points to its current row of data. When the ResultSet object is first initialized, the cursor is positioned before the first row. This is why we need to first call the next() method before retrieving data.
    - The ResultSet object is used to loop through and process each row of data and retrieve the column values that we want to access.
      - In this case, we access the value in the name column using the rec.getString method, which is a part of the JDBC API.

Turn data from database into java variable

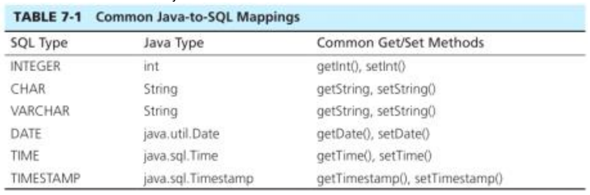


TABLE 7-1 Common Java-to-SQL Mappings

SQL Type	Java Type	Common Get/Set Methods
INTEGER	int	getInt(), setInt()
CHAR	String	getString, setString()
VARCHAR	String	getString, setString()
DATE	java.util.Date	getDate(), setDate()
TIME	java.sql.Time	getTime(), setTime()
TIMESTAMP	java.sql.Timestamp	getTimestamp(), setTimestamp()

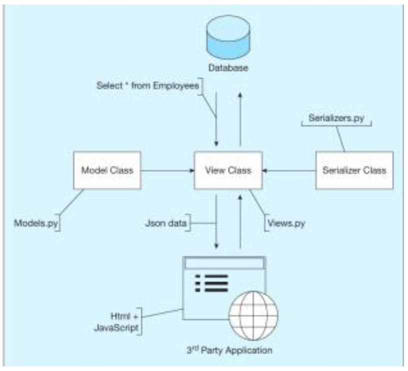
- The JSP example presented above has several drawbacks associated with it. First, the HTML code, Java code, and SQL code are all mixed in together. Because the same person is unlikely to possess expertise in all three areas, creating large applications using this paradigm will be challenging.
  - To overcome this problem, most Web applications are designed using a concept known as the Model-View-Controller (MVC).
  - Using this architecture, the presentation logic (view), the business logic (controller/model), and the database logic (model) are separated.

- We will use MVC architecture and frameworks in the Python example below.

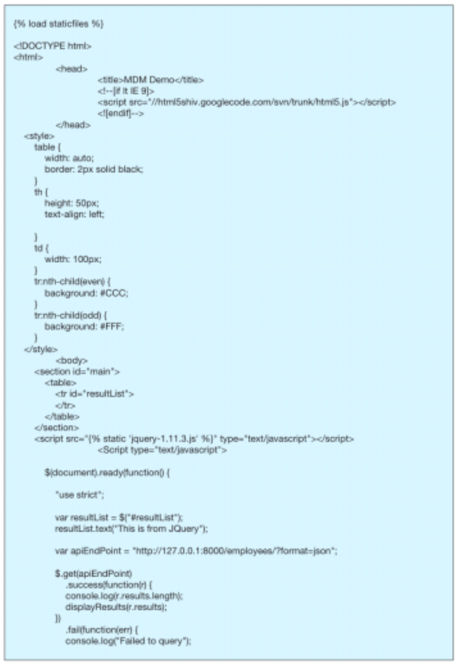
A Python Web Application (also showcasing MVC framework)

- Python has gained more traction and popularity than any other languages in the developer community due to the availability of many excellent frameworks, such as Django, Flask, Pyramid, Tornado, Bottle, Diesel, Pecan, and Falcon, to name a few.
- These frameworks provide a collection of packages or modules that allow developers to write Web applications or services without having to handle low-level details.
- This example uses Django, which is a Python Web framework.

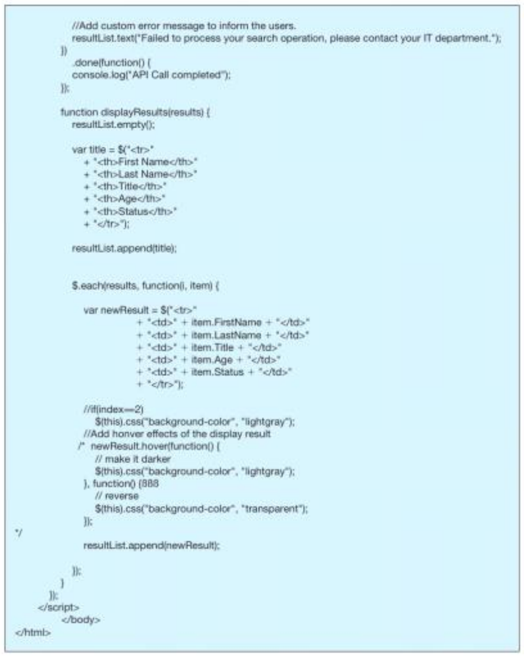
- Figure 7-8 shows the system architecture for a three-tier application built using Python. ------------>
  - The third-party application that resides on the client is likely going to be HTML based.
  - The Model, View, and Serializer classes are written using Python and reside on the application server.
  - The database resides on the database server.
  - The figure also shows that the data exchange between View classes and the application are done using a standard format, in our case JavaScript Object Notation (JSON).
  - Notice that the view class used the model class and serialization class to be able to run its operation
    - The operations being grabbing from database, turning data into json data for the front end 3rd party application to read from, calling functions from the model class, etc.



- Figure 7-10 shows the HTML/Javascript code that can be used to retrieve and display data from the sample database on a Web page

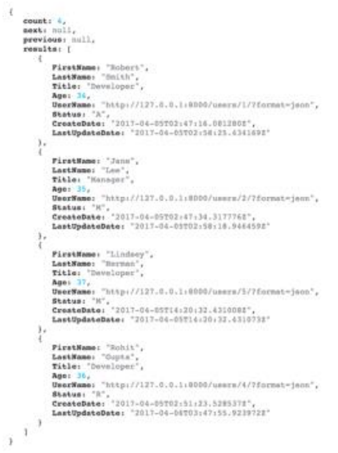


```
{% load staticfiles %}

<!DOCTYPE html>
<html>
            <head>
                        <title>MDM Demo</title>
                        <!--[if lt IE 9]>
                        <script src="//html5shiv.googlecode.com/svn/trunk/html5.js"></script>
                        <![endif]-->
            </head>
   <style>
      table {
         width: auto;
         border: 2px solid black;
      }
      th {
         height: 50px;
         text-align: left;

      }
      td {
         width: 100px;
      }
      tr:nth-child(even) {
         background: #CCC;
      }
      tr:nth-child(odd) {
         background: #FFF;
      }
   </style>
            <body>
      <section id="main">
         <table>
            <tr id="resultList">
            </tr>
         </table>
      </section>
      <script src="{% static 'jquery-1.11.3.js' %}" type="text/javascript"></script>
                        <Script type="text/javascript">

         $(document).ready(function() {

            "use strict";

            var resultList = $("#resultList");
            resultList.text("This is from JQuery");

            var apiEndPoint = "http://127.0.0.1:8000/employees/?format=json";

            $.get(apiEndPoint)
               .success(function(r) {
               console.log(r.results.length);
               displayResults(r.results);
            })
               .fail(function(err) {
               console.log("Failed to query");
```



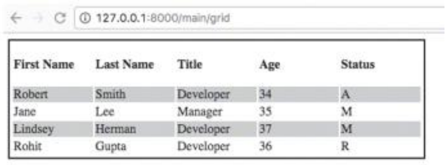
```
               //Add custom error message to inform the users.
               resultList.text("Failed to process your search operation, please contact your IT department.");
            })
               .done(function() {
               console.log("API Call completed");
            });

            function displayResults(results) {
               resultList.empty();

               var title = $("<tr>"
                  + "<th>First Name</th>"
                  + "<th>Last Name</th>"
                  + "<th>Title</th>"
                  + "<th>Age</th>"
                  + "<th>Status</th>"
                  + "</tr>");

               resultList.append(title);


               $.each(results, function(i, item) {

                  var newResult = $("<tr>"
                              + "<td>" + item.FirstName + "</td>"
                              + "<td>" + item.LastName + "</td>"
                              + "<td>" + item.Title + "</td>"
                              + "<td>" + item.Age + "</td>"
                              + "<td>" + item.Status + "</td>"
                              + "</tr>");

                  //if(index==2)
                     $(this).css("background-color", "lightgray");
                  //Add honver effects of the display result
                 /*  newResult.hover(function() {
                     // make it darker
                     $(this).css("background-color", "lightgray");
                  }, function() {888
                     // reverse
                     $(this).css("background-color", "transparent");
                  });
*/
                  resultList.append(newResult);

               });
            }
         });
      </script>
            </body>
</html>
```

The Web server routes this URL to the application server (running Django/Python), which in turn returns a JSON formatted result as shown.
- This would be the json file that is returned back to the frontend (3rd party application)



```
{
   count: 4,
   next: null,
   previous: null,
   results: [
      {
         FirstName: "Robert",
         LastName: "Smith",
         Title: "Developer",
         Age: 34,
         UserName: "http://127.0.0.1:8000/users/1/?format=json",
         Status: "A",
         CreateDate: "2017-04-05T02:47:16.081280Z",
         LastUpdateDate: "2017-04-05T02:58:25.434169Z"
      },
      {
         FirstName: "Jane",
         LastName: "Lee",
         Title: "Manager",
         Age: 35,
         UserName: "http://127.0.0.1:8000/users/2/?format=json",
         Status: "M",
         CreateDate: "2017-04-05T02:47:34.317776Z",
         LastUpdateDate: "2017-04-05T02:58:18.946459Z"
      },
      {
         FirstName: "Lindsey",
         LastName: "Herman",
         Title: "Developer",
         Age: 37,
         UserName: "http://127.0.0.1:8000/users/5/?format=json",
         Status: "M",
         CreateDate: "2017-04-05T14:20:32.431008Z",
         LastUpdateDate: "2017-04-05T14:20:32.431073Z"
      },
      {
         FirstName: "Rohit",
         LastName: "Gupta",
         Title: "Developer",
         Age: 36,
         UserName: "http://127.0.0.1:8000/users/4/?format=json",
         Status: "R",
         CreateDate: "2017-04-05T02:51:23.528537Z",
         LastUpdateDate: "2017-04-06T03:47:55.923972Z"
      }
   ]
}
```

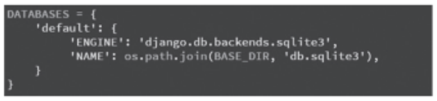
This would be the ouput of the 3rd party application once it retrieves data from the backend and all is said and done



127.0.0.1:8000/main/grid

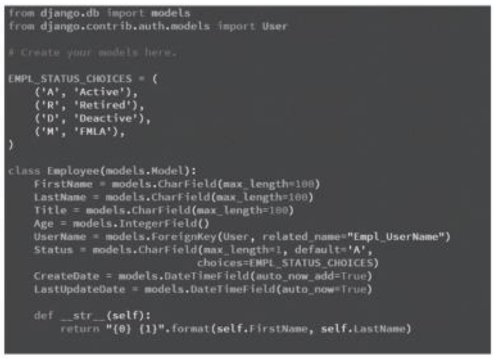
First Name	Last Name	Title	Age	Status
Robert	Smith	Developer	34	A
Jane	Lee	Manager	35	M
Lindsey	Herman	Developer	37	M
Rohit	Gupta	Developer	36	R

- We will now examine how the above call is processed in the application server and how it retrieves data from the database.
  - The first step is to specify which database to use. This is specified in Django in the settings.py file (Figure 7-13). Similar to the JSP application, The 'ENGINE' specifies the database driver/middleware to use, and the 'NAME' specifies a path to the actual database.



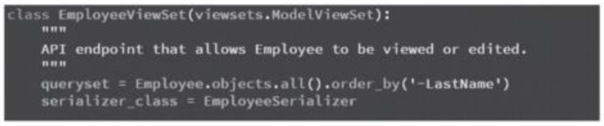
```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}
```

  - The next step is for us to specify a model class for each table in the database we are likely to use in the application. Each row retrieved from the database will be stored in an instance of this model class.
    - This is essentially like creating a replication of a table in our database as a class so that we can create objects out of them. The models we create will of course have attributes that are the same as the columns in the table alongside their type.
    - Figure 7-14 shows the model class Employee (stored in models.py), which corresponds to an employee table in the database. Notice how each attribute in the model class, such as FirstName, corresponds to the name of a column in the table.
      - This mapping is what allows the Django framework to identify which fields to retrieve from the Employee table.

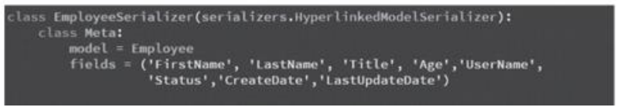
- 

```python
from django.db import models
from django.contrib.auth.models import User

# Create your models here.

EMPL_STATUS_CHOICES = (
    ('A', 'Active'),
    ('R', 'Retired'),
    ('D', 'Deactive'),
    ('M', 'FMLA'),
)

class Employee(models.Model):
    FirstName = models.CharField(max_length=100)
    LastName = models.CharField(max_length=100)
    Title = models.CharField(max_length=100)
    Age = models.IntegerField()
    UserName = models.ForeignKey(User, related_name="Empl_UserName")
    Status = models.CharField(max_length=1, default='A',
                              choices=EMPL_STATUS_CHOICES)
    CreateDate = models.DateTimeField(auto_now_add=True)
    LastUpdateDate = models.DateTimeField(auto_now=True)

    def __str__(self):
        return "{0} {1}".format(self.FirstName, self.LastName)
```

- Once the model class has been defined, it can be then be used in a View class as a surrogate for the data in the database.

- The code in the View class is what is called from the client application. Thus, each View class is designed to perform a specific function and has a well-defined input and output.
    - This is the "point man" as it is what runs the show on the backend. It uses both the model class and serialization class to do the actual work that needs to be done
    - The Python code for the View class—EmployeeViewSet (stored in the views.py file)—to perform this function is shown in Figure 7-15
    - 

      ```python
      class EmployeeViewSet(viewsets.ModelViewSet):
          """
          API endpoint that allows Employee to be viewed or edited.
          """
          queryset = Employee.objects.all().order_by('-LastName')
          serializer_class = EmployeeSerializer
      ```

        - Essentially, it says to retrieve all objects in the table that correspond to the Employee model class and return the data sorted by LastName in descending order.
        - The Django framework takes care of opening the database connection, issuing the appropriate SQL query and populating the results into a set of instances (objects) of type Employee model class.
- The second line in the EmployeeViewSet class is used to serialize (using the EmployeeSerializer; Figure 7-16) the instances in the variable queryset so that it can be sent over in a format that the client browser can process, in our case in JSON format.
    - 

      ```python
      class EmployeeSerializer(serializers.HyperlinkedModelSerializer):
          class Meta:
              model = Employee
              fields = ('FirstName', 'LastName', 'Title', 'Age','UserName',
                        'Status','CreateDate','LastUpdateDate')
      ```

    - The value of the model variable indicates to the serializer that each object that is being serialized is of type Employee.
    - The value of the fields variable indicates which fields from the model you want to serialize. The end result of this serialization is the JSON-formatted data