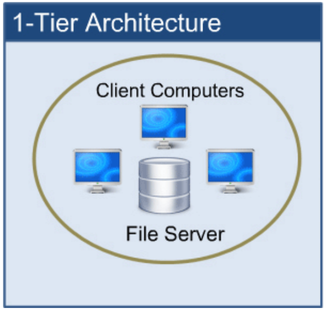
1-tier Architecture

- One-tier architecture involves putting all of the required components for a software application or technology on a single server or platform

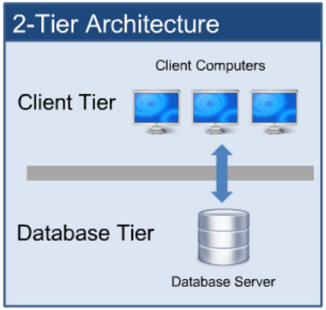


- Basically, a one-tier architecture keeps all of the elements of an application, including the interface, Middleware and back-end data, in one place
- Developers see these types of systems as the simplest and most direct way

Source: *Concepts of Database Architecture*

2-tier Architecture

- The two-tier is based on Client Server architecture
- It is like client server application

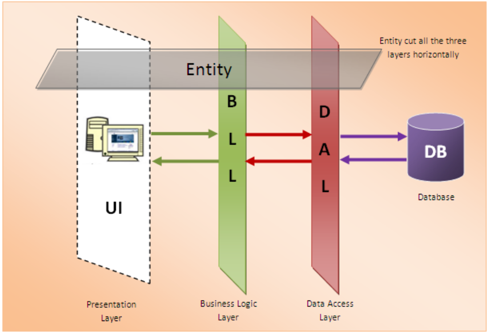


- The direct communication takes place between client and server
- There is no intermediate between client and server

Source: *Concepts of Database Architecture*

3-tier Architecture

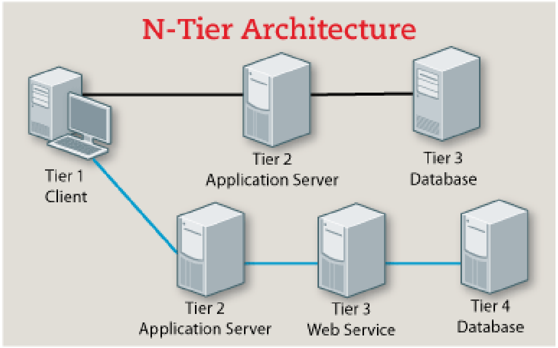
- A 3-tier architecture separates its tiers - *Presentation*, *Logic* and *Data Access* - from each other based on the complexity of the users and how they use the data present in the database
- It is the most widely used architecture to design a DBMS



Source: *Concepts of Database Architecture*

n-tier Architecture

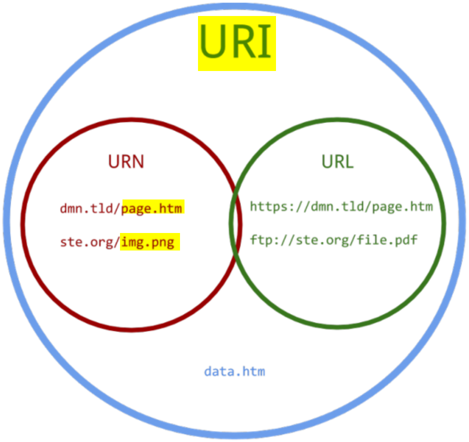
- An n-tier architecture distributes different components of the 3 tiers between different servers and adds interfaces tiers for interactions and workload balancing



Source: *Concepts of Database Architecture*

URI, URL, and URN

- Uniform Resource Identifier (URI)
- Uniform Resource Locator (URL)
- Uniform Resource Name (URN)
- Relationships:
  - URIs can be classified as locators (URLs), or as names (URNs), or as both.
  - URN functions like a person's name
  - URL resembles that person's street address.
  - URN defines an item's identity, while the URL provides a method for finding it



HTTP and Sessions

- The HTTP protocol is connectionless
  - That is, once the server replies to a request, the server closes the connection with the client, and forgets all about the request
  - In contrast, Unix logins, and JDBC/ODBC connections stay connected until the client disconnects
    - ▷ retaining user authentication and other information
  - Motivation: reduces load on server
    - ▷ operating systems have tight limits on number of open connections on a machine
- Information services need session information
  - For example, user authentication should be done only once per session
- Solution: use a cookie

Sessions and Cookies

- A cookie is a small piece of text containing identifying information
  - Sent by server to browser
    - ▷ Sent on first interaction, to identify session
  - Sent by browser to the server that created the cookie on further interactions
    - ▷ part of the HTTP protocol
  - Server saves information about cookies it issued, and can use it when serving a request
    - ▷ For example, authentication information, and user preferences
- Cookies can be stored permanently or for a limited time

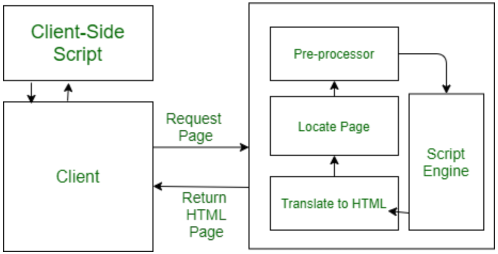
Web Services

- Allow data on Web to be accessed using remote procedure call mechanism
- Two approaches are widely used
  - **Representation State Transfer (REST):** allows use of standard HTTP request to a URL to execute a request and return data
    - returned data is encoded either in XML, or in **JavaScript Object Notation (JSON)**
  - **Big Web Services:**
    - uses XML representation for sending request data, as well as for returning results
    - standard protocol layer built on top of HTTP

Scripting for Web Applications (2)

Scripting of two types

- **Client Side**: Client-side scripting is responsible for interaction within a web page. The client-side scripts are firstly downloaded at the client-end and then interpreted and executed by the browser
- **Server Side**: Server-side scripting is responsible for the completion or carrying out a task at the server-end and then sending the result to the client-end.



Source: *Web Scripting and its Types*

Servlets

- Java Servlet specification defines an API for communication between the Web / application server and application program running in the server
    - For example, methods to get parameter values from Web forms, and to send HTML text back to client
- Application program (also called a servlet) is loaded into the server
    - Each request spawns a new thread in the server
        - ▷ thread is closed once the request is serviced

Servlet (4): Sessions

- Servlet API supports handling of sessions
  - Sets a cookie on first interaction with browser, and uses it to identify session on further interactions
- To check if session is already active:
  - if (request.getSession(false) == true)
    - ▷ .. then existing session
    - ▷ else .. redirect to authentication page
  - authentication page
    - ▷ check login/password
    - ▷ request.getSession(true): creates new session
- Store/retrieve attribute value pairs for a particular session
  - session.setAttribute("userid", userid)
  - session.getAttribute("userid")

USP of JSP

- ***JSP vs. Active Server Pages*** (***ASP***)
  - ASP is a similar technology from Microsoft and is proprietary (uses VB).
  - JSP is platform independent and portable.
- ***JSP vs. Pure Servlets***
  - JSP is a servlet but it is more convenient to write and to modify regular HTML than to have a million println statements that generate the HTML.
  - The Web page design experts can build the HTML, leaving places for the servlet programmers to insert the dynamic content.
- ***JSP vs. JavaScript*** JavaScript can generate HTML dynamically on the client.
  - "Client Side": Java Script code is executed by the browser after the web server sends the HTTP response. With the exception of cookies, HTTP and form submission data is not available to JavaScript.
  - "Server Side": Java Server Pages are executed by the web server before the web server sends the HTTP response. It can access server-side resources like databases, catalogs.
- ***JSP vs. Static HTML*** Regular HTML cannot contain dynamic information.

Working with SQL and Native Language

- Applications use **Application Programming / Program Interface (API)** to interact with a database server
- Applications make calls to
  - Connect with the database server
  - Send SQL commands to the database server
  - Fetch tuples of result one-by-one into program variables
- Frameworks
  - **Connectionist**
    - **Open Database Connectivity (ODBC)** works with C, C++, C#, Visual Basic, and Python. Other data APIs include
      - OLEDB
      - ADO.NET
    - **Java Database Connectivity (JDBC)** works with Java
  - **Embedding**
    - **Embedded SQL** works with C, C++, Java, COBOL, FORTRAN and Pascal

ODBC

- **Open Database Connectivity (ODBC)** is a standard API for accessing DBMS
- It aimed to be independent of database systems and operating systems
- An application written using ODBC can be ported to other platforms, both on the client and server side, with few changes to the data access code
- ODBC is
  - A standard for application program to communicate with a database server
  - An application program interface (API) to
    - ▷ Open a connection with a database
    - ▷ Send queries and updates
    - ▷ Get back results
- Applications such as GUI, Spreadsheets, etc. can use ODBC
- ODBC was originally developed by Microsoft and Simba Technologies during the early 1990s, and became the basis for the Call Level Interface (CLI) standardized by SQL Access Group in the Unix and mainframe field.

ODBC (2): Python Example

- The code uses a data source named "SQLS" from the odbc.ini file to connect and issue a query.
- It creates a table, inserts data using literal and parameterized statements and fetches the data

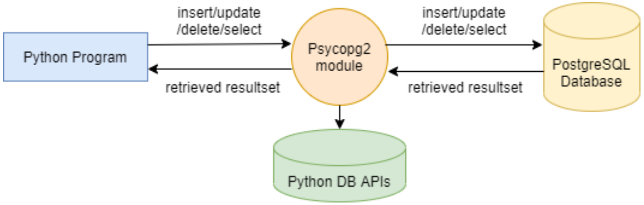
```
import pyodbc

conn = pyodbc.connect('DSN=SQLS;UID=test01;PWD=test01')
cursor=conn.cursor()
cursor.execute("create table rvtest (col1 int, col2 float,
col3 varchar(10))")
cursor.execute("insert into rvtest values(1, 10.0,
\"ABC\")")
cursor.execute("select * from rvtest")

while True:
      row=cursor.fetchone()
      if not row:
            break
      print(row)


cursor.execute("delete from rvtest")
cursor.execute("insert into rvtest values (?, ?, ?)", 2,
20.0, 'XYZ')
cursor.execute("select * from rvtest")

while True:
      row=cursor.fetchone()
      if not row:
            break
      print(row)
```

Source: *https://dzone.com/articles/tutorial-connecting-to-odbc-data-sources-with-pyth*

Steps to access PostgresSQL from Python

Steps to access PostgresSQL from Python using Psycopg

a) Create connection
b) Create cursor
c) Execute the query
d) Commit/rollback
e) Close the cursor
f) Close the connection



Source: *https://pynative.com/python-postgresql-tutorial/*

Python psycopg2 Module APIs: connection objects

- `psycopg2.connect(database="mydb", user="myuser", password="mypass" host="127.0.0.1", port="5432")`
  This API opens a connection to the PostgreSQL database. If database is opened successfully, it returns a connection object.
- `connection.close()`
  This method closes the database connection.

Important **psycopg2** module routines for managing cursor object:

- `connection.cursor()`
  This routine creates a cursor which will be used throughout the program.
- `cursor.close()`
  This method closes the cursor.

Source: *https://www.tutorialspoint.com/postgresql/postgresql_python.htm*

Python psycopg2 Module APIs: insert, delete, update & stored procedures

- `cursor.execute(sql [, optional parameters])`
  This routine executes an SQL statement. The SQL statement may be parameterized (i.e., placeholders instead of SQL literals). The **psycopg2** module supports placeholder using %s sign. For example: `cursor.execute("insert into people values (%s, %s)", (who, age))`
- `cursor.executemany(sql, seq_of_parameters)`
  This routine executes an SQL command against all parameter sequences or mappings found in the sequence SQL.
- `cursor.callproc(procname[, parameters])`
  This routine executes a stored database procedure with the given name. The sequence of parameters must contain one entry for each argument that the procedure expects.
- `cursor.rowcount`
  This is a read-only attribute which returns the total number of database rows that have been modified, inserted, or deleted by the last `execute()`.

Source: *https://www.tutorialspoint.com/postgresql/postgresql_python.htm*

Python psycopg2 Module APIs: select

returns tuple

- `cursor.fetchone()`
  This method fetches the next row of a query result set, returning a single sequence, or None when no more data is available.

- `cursor.fetchmany([size=cursor.arraysize])` returns list of tuples
  This routine fetches the next set of rows of a query result, returning a list. An empty list is returned when no more rows are available. The method tries to fetch as many rows as indicated by the size parameter.

- `cursor.fetchall()` returns list of tuples
  This routine fetches all (remaining) rows of a query result, returning a list. An empty list is returned when no rows are available.

Source: *https://www.tutorialspoint.com/postgresql/postgresql_python.htm*

Python psycopg2 Module APIs: commit & rollback

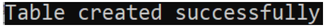
- `connection.commit()`
  This method commits the current transaction. If you do not call this method, anything you did since the last call to `commit()` is not visible to other database connections.
- `connection.rollback()`
  This method rolls back any changes to the database since the last call to `commit()`.

Source: *https://www.tutorialspoint.com/postgresql/postgresql_python.htm*

CREATE new PostgreSQL tables

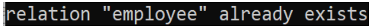
```
import psycopg2
def createTable():
    conn = None
    try:
        conn = psycopg2.connect(database = "mydb", user = "myuser", \
            password = "mypass", host = "127.0.0.1", port = "5432") # connect to the database
        cur = conn.cursor()  # create a new cursor
        cur.execute('''CREATE TABLE EMPLOYEE \
            (emp_num INT PRIMARY KEY     NOT NULL, \
            emp_name VARCHAR(40)    NOT NULL, \
            department VARCHAR(40)     NOT NULL)''') # execute the CREATE TABLE statement
        conn.commit()       # commit the changes to the database
        print ("Table created successfully")
        cur.close()         # close the cursor
    except (Exception, psycopg2.DatabaseError) as error:
        print(error)
    finally:
        if conn is not None:
            conn.close()    # close the connection
createTable()    #function call
```

Output (if table EMPLOYEE does not exist):



```
Table created successfully
```

Output (if table EMPLOYEE already exists):

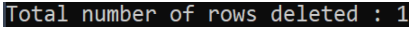


```
relation "employee" already exists
```

Executing DELETE statement from Python

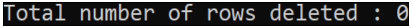
```python
import psycopg2
def deleteRecord(num):
    conn = None
    try:
        # connect to the PostgreSQL database
        conn = psycopg2.connect(database = "mydb", user = "myuser", \
            password = "mypass", host = "127.0.0.1", port = "5432")
        cur = conn.cursor()      # create a new cursor
        # execute the DELETE statement
        cur.execute("DELETE FROM EMPLOYEE WHERE emp_num = %s", (num,))
        conn.commit()            # commit the changes to the database
        print ("Total number of rows deleted :", cur.rowcount)
        cur.close()              # close the cursor
    except (Exception, psycopg2.DatabaseError) as error:
        print(error)
    finally:
        conn.close()             # close the connection
deleteRecord(110)                #function call
```

Output:



```
Total number of rows deleted : 1
```

Output:



```
Total number of rows deleted : 0
```

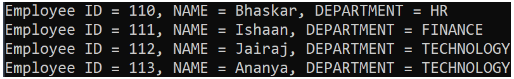
If the row does not exist

Executing SELECT statement from Python

```
import psycopg2
def selectAll():
    conn = None
    try:
        # connect to the PostgreSQL database
        conn = psycopg2.connect(database = "mydb", user = "myuser", \
            password = "mypass", host = "127.0.0.1", port = "5432")
        cur = conn.cursor()       # create a new cursor
        # execute the SELECT statement
        cur.execute("SELECT emp_num, emp_name, department FROM EMPLOYEE")
        rows = cur.fetchall()     # fetches all rows of the query result set
        for row in rows:
            print (print ("Employee ID = ", row[0], ", NAME = ", \
                row[1], ", DEPARTMENT = ", row[2]))
        cur.close()               # close the cursor
    except (Exception, psycopg2.DatabaseError) as error:
        print(error)
    finally:
        conn.close()              # close the connection
selectAll()                       # function call
```

no commit since you have not made any changes

Output:



```
Employee ID = 110, NAME = Bhaskar, DEPARTMENT = HR
Employee ID = 111, NAME = Ishaan, DEPARTMENT = FINANCE
Employee ID = 112, NAME = Jairaj, DEPARTMENT = TECHNOLOGY
Employee ID = 113, NAME = Ananya, DEPARTMENT = TECHNOLOGY
```

Flask Web Application Framework

- Flask is a lightweight **WSGI** (Web Server Gateway Interface) web application framework. It is designed to make *getting started* quick and easy, with the ability to scale up to complex applications.
- It began as a simple wrapper around **Werkzeug** (Werkzeug WSGI toolkit) and **Jinja** (Jinja template engine) and has since then become one of the most popular Python web application frameworks.
- Flask offers suggestions, but does not enforce any dependencies or project layouts. It is up to the developer to choose the tools and libraries they want to use.
- There are many extensions provided by the community that make adding new functionality easy.

Installing Flask using `pip` command

- `pip install -U Flask`

Source: *https://pypi.org/project/Flask/*

Python: Flask (5)

- `savaDetails()` function for `add.html` (in Python):

```
@app.route("/savedetails",methods = ["POST"])
def saveDetails():
    cno = request.form["cno"]
    name = request.form["name"]
    email = request.form["email"]
    conn = None
    try:
        conn = psycopg2.connect(database = "mydb", user = "myuser", \
            password = "mypass", host = "127.0.0.1", port = "5432") # connect to the PostgreSQL database
        cur = conn.cursor()     # create a new cursor
        cur.execute("INSERT INTO Candidate (cno, name, email) \
            VALUES (%s, %s, %s)", (cno, name, email)) # execute the INSERT statement
        conn.commit() # commit the changes to the database
        cur.close() # close the cursor
    except (Exception, psycopg2.DatabaseError) as error:
        render_template("fail.html")
    finally:
        if conn is not None:
            conn.close()    # close the connection
    return render_template("success.html")
```

Rapid Application Development

- A lot of effort is required to develop Web application interfaces, especially the rich interaction functionality associated with Web 2.0 applications
- Several approaches to speed up application development
  - Function library to generate user-interface elements
  - Drag-and-drop features in an IDE to create user-interface elements
  - Automatically generate code for user interface from a declarative specification
- Used as part of **Rapid Application Development (RAD)** tools even before Web
- RAD Software is an agile model that focuses on fast prototyping and quick feedback in app development to ensure speedier delivery and an efficient result
  - App development has 4 phases: business modeling, data modeling, process modeling, and testing & turnover: Defining the requirements, Prototyping, Receiving feedback and Finalizing the software
  - With RAD, the time between prototypes and iterations is short, and integration occurs since inception.

Application Performance

- Performance is an issue for popular Web sites
  - May be accessed by millions of users every day, thousands of requests per second at peak time
- Caching techniques used to reduce cost of serving pages by exploiting commonalities between requests
  - At the server site:
    - ▷ Caching of JDBC connections between servlet requests
      - a.k.a. connection pooling
    - ▷ Caching results of database queries
      - Cached results must be updated if underlying database changes
    - ▷ Caching of generated HTML
  - At the client's network
    - ▷ Caching of pages by Web proxy

Application Security (4): Application-Level Authorization

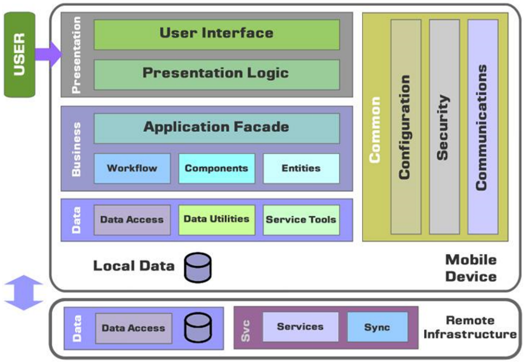
- Current SQL standard does not allow fine-grained authorization such as "students can see their own grades, but not other's grades"
  - Problem 1: Database has no idea who are application users
  - Problem 2: SQL authorization is at the level of tables, or columns of tables, but not to specific rows of a table
- One workaround: use views such as

  create view *studentTakes as*
  select *
  from *takes*
  where *takes.ID* = *syscontext.user_id()*

  - where syscontext.user_id() provides end user identity
    - ▷ end user identity must be provided to the database by the application
  - Having multiple such views is cumbersome

Architecture of Mobile App

- Typically 3 tier
  - Presentation
  - Business
  - Data
- Data Layer is often split between:
  - Local Data
  - Remote Data
- Needs customization for platform
  - Android
  - iOS
  - Windows



Source: https://www.slideshare.net/hassandar18/architecture-of-mobile-software-applications?from_action=save