SDSC5003

Database Application

- **RG Chapter 6, Chapter 7**
- GWU Chapter 9

Why this lecture

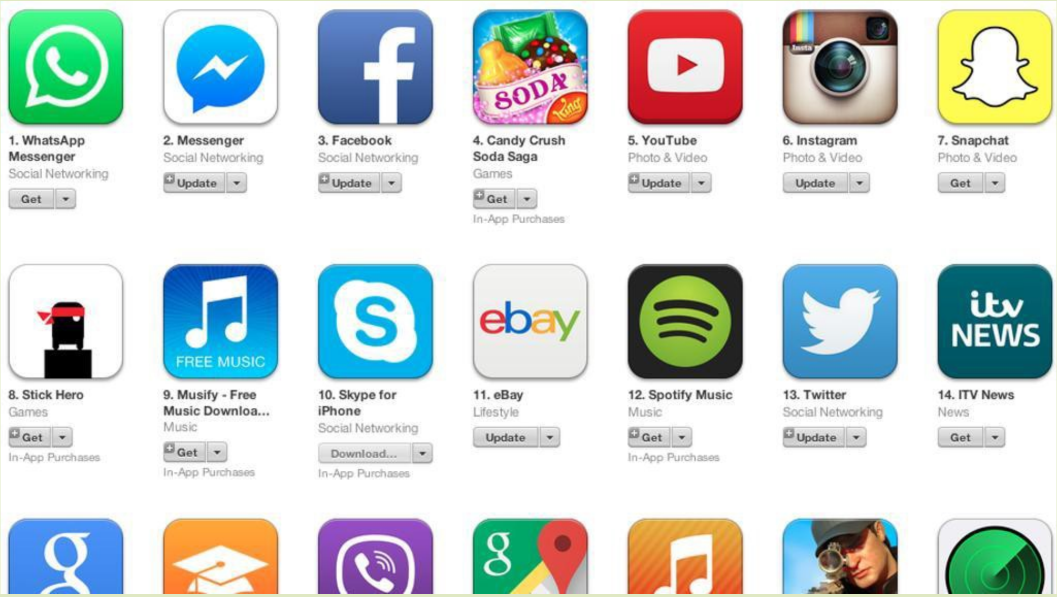
- **DB designer:** establishes schema
- **DB administrator:** tunes systems and keeps whole things running
- **Data scientist:** manipulates data to extract insights
- **Data engineer:** builds a data-processing pipeline

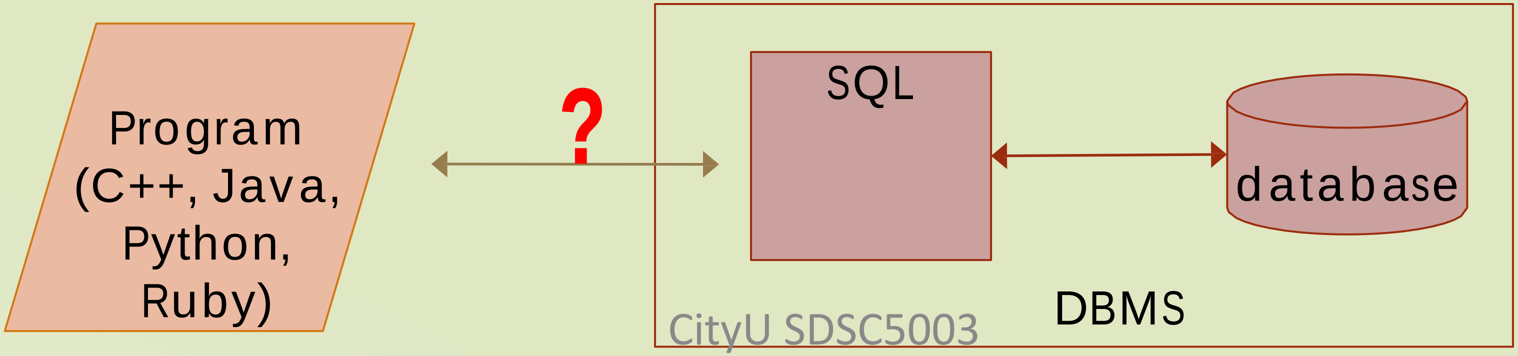
- **DB application developer:** writes programs that query and modify a database

Outline

- **Database Programming**
- Application Architecture

Programming Environment



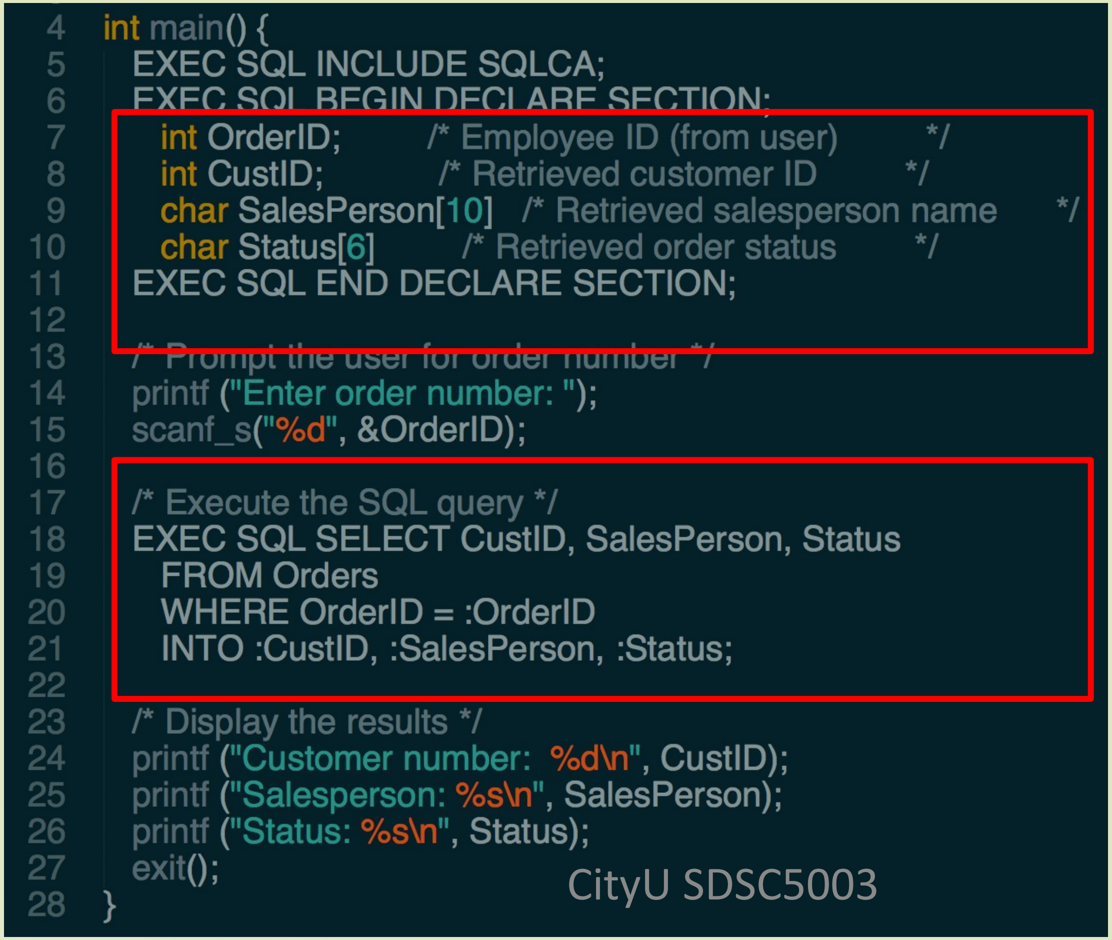


Connecting to DBMS

- **Fully embed into language (embedded SQL)**
- Low-level library with core database calls (DB API)
- Stored Procedures
- Object-relational mapping (ORM)
  - Ruby on rails, django, etc
  - define database-backed classes
  - magically maps between database rows & objects
  - magic is a double-edged sword

Embedded SQL

- **Extend host language (CPP) with SQL syntax**

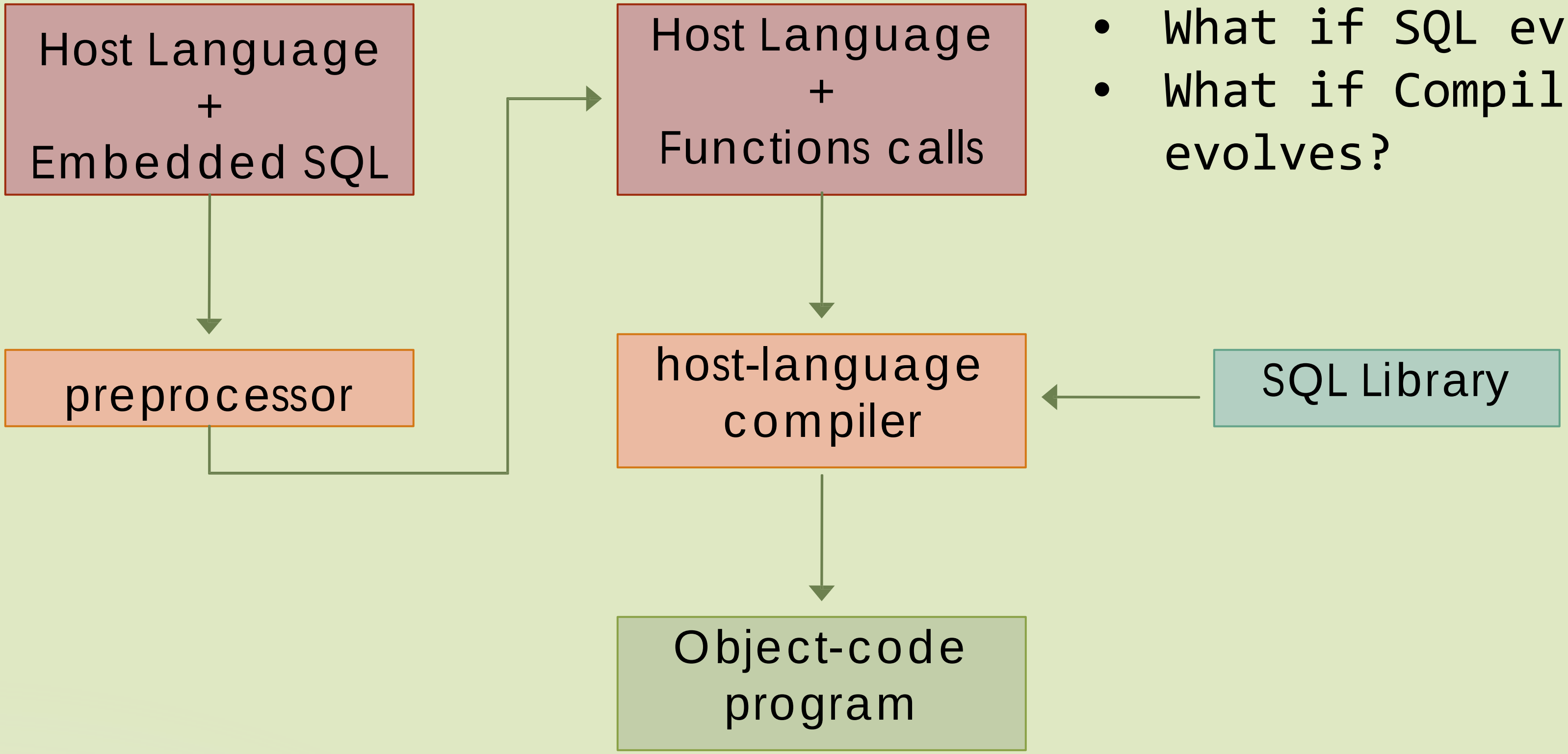


```
int main() {
   EXEC SQL INCLUDE SQLCA;
   EXEC SQL BEGIN DECLARE SECTION;
      int OrderID;         /* Employee ID (from user)         */
      int CustID;            /* Retrieved customer ID         */
      char SalesPerson[10]   /* Retrieved salesperson name      */
      char Status[6]         /* Retrieved order status         */
   EXEC SQL END DECLARE SECTION;

   /* Prompt the user for order number */
   printf ("Enter order number: ");
   scanf_s("%d", &OrderID);

   /* Execute the SQL query */
   EXEC SQL SELECT CustID, SalesPerson, Status
      FROM Orders
      WHERE OrderID = :OrderID
      INTO :CustID, :SalesPerson, :Status;

   /* Display the results */
   printf ("Customer number:  %d\n", CustID);
   printf ("Salesperson: %s\n", SalesPerson);
   printf ("Status: %s\n", Status);
   exit();
}
```

Declaring Variables

Embedded
SQL Query

Embedded SQL

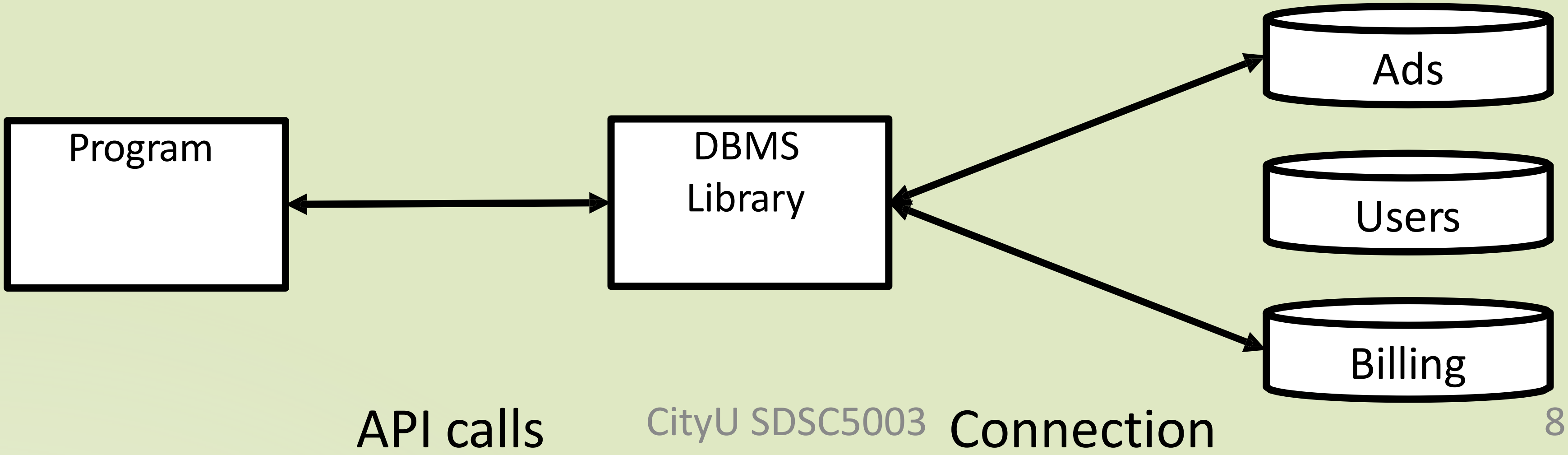


Hard to maintain

- What if SQL evolves?
- What if Compiler evolves?

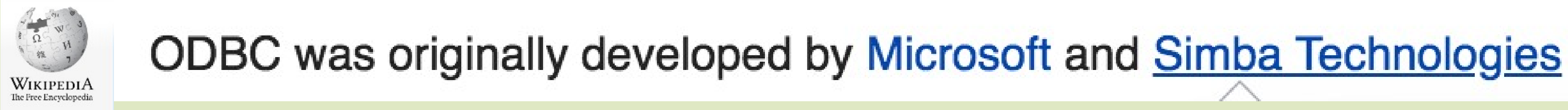
What does a library need to do?

- Single interface to possibly multiple DBMS engines
- Connect to a database
- Map objects between host language and DBMS
- Manage query results



ODBC and JDBC

- ODBC (Open DataBase Connectivity)



ODBC was originally developed by Microsoft and Simba Technologies

- JDBC (Java DataBase Connectivity)
  - Sun developed as set of Java interfaces
    - javax.sql.*

Connections

Create a connection

- Allocate resources for the connection
- Relatively expensive to set up, libraries often cache connections for future use

```
conn = connect(sdsc5003.db)
```

Should close connections when done! Otherwise resource leak.

Query Execution

```
foo = conn.execute("select * from
student")
```

- Challenges
  - Type Mismatch
  - What is the return type of execute()?
  - How to pass data between DBMS and host language?

Type Mismatch

- SQL standard defines mappings between SQL and several languages

SQL types	C types	Python types
CHAR(20)	char[20]	str
INTEGER	int	int
SMALLINT	short	int
REAL	float	float

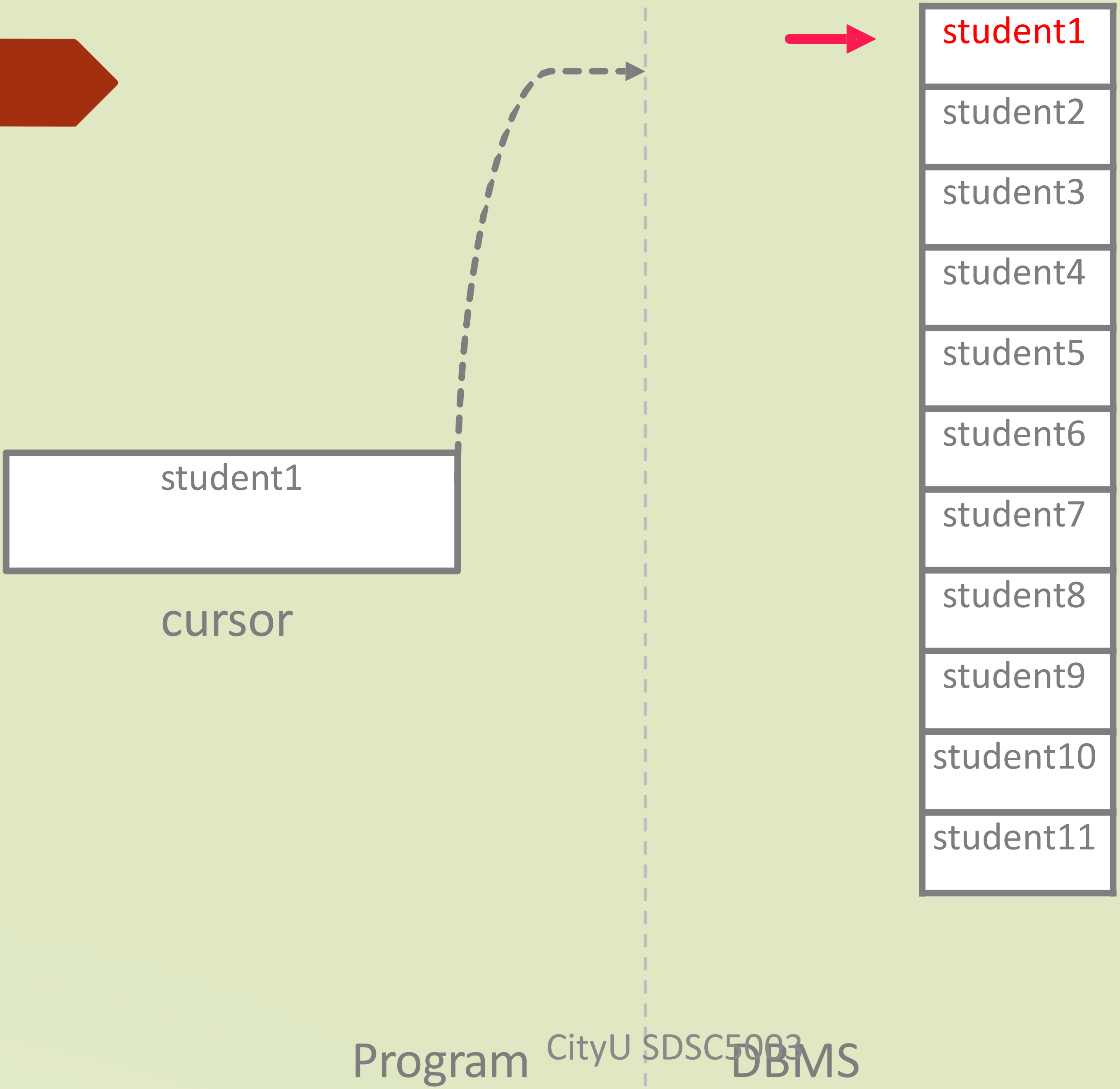
Cursor

- SQL relations and results are sets of records

- What is the type of foo?

```
foo = conn1.execute(“select * from student”)
```

- Cursor over the Result Set
  - similar to an iterator interface
  - Note: relations are unordered!
    - Cursors have no ordering guarantees
    - Use ORDER BY to ensure an ordering



student1

cursor

Program

DBMS

student1

student2

student3

student4

student5

student6

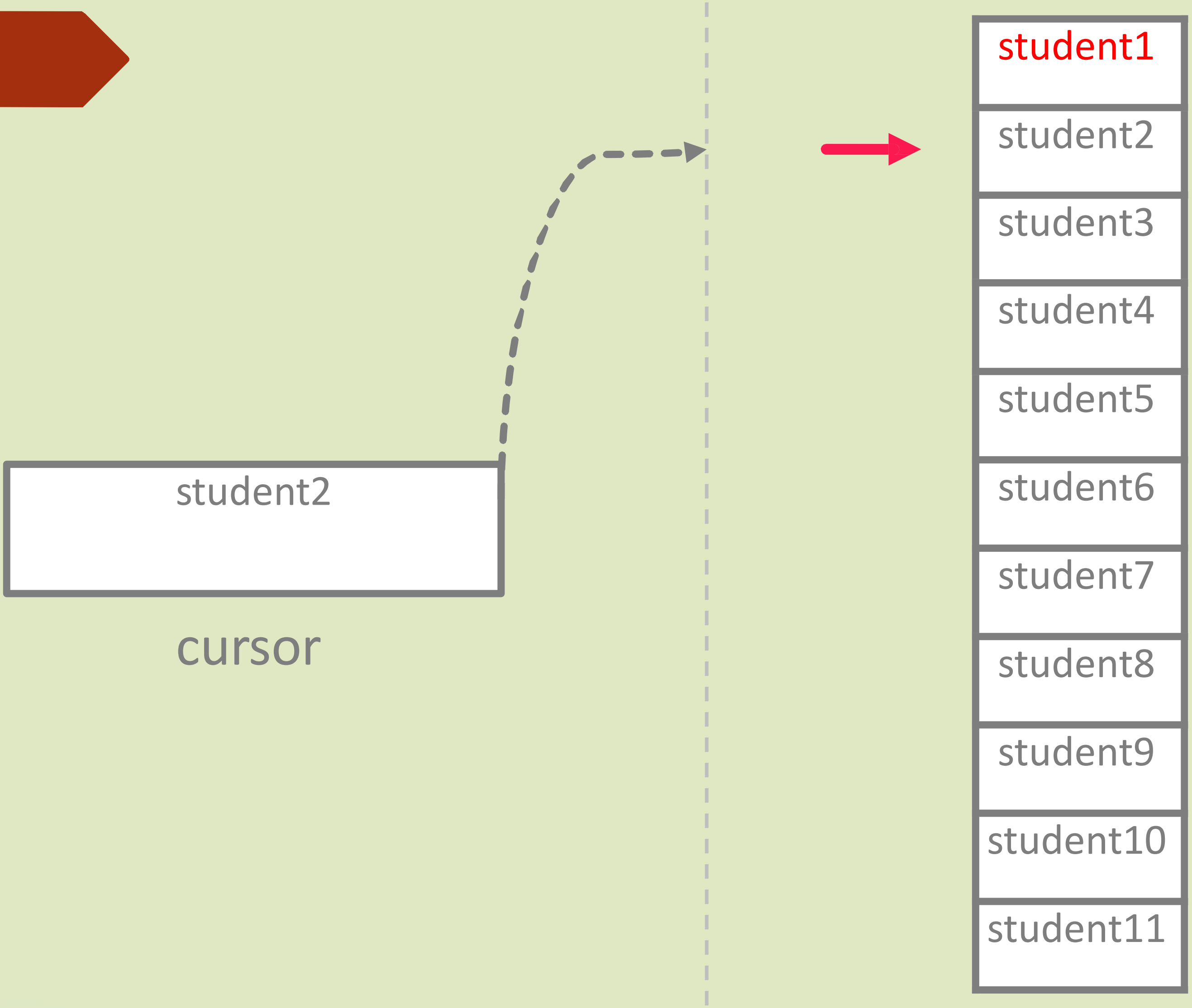
student7

student8

student9

student10

student11



student2

cursor

student1

student2

student3

student4

student5

student6

student7

student8

student9

student10

student11

Program

DBMS

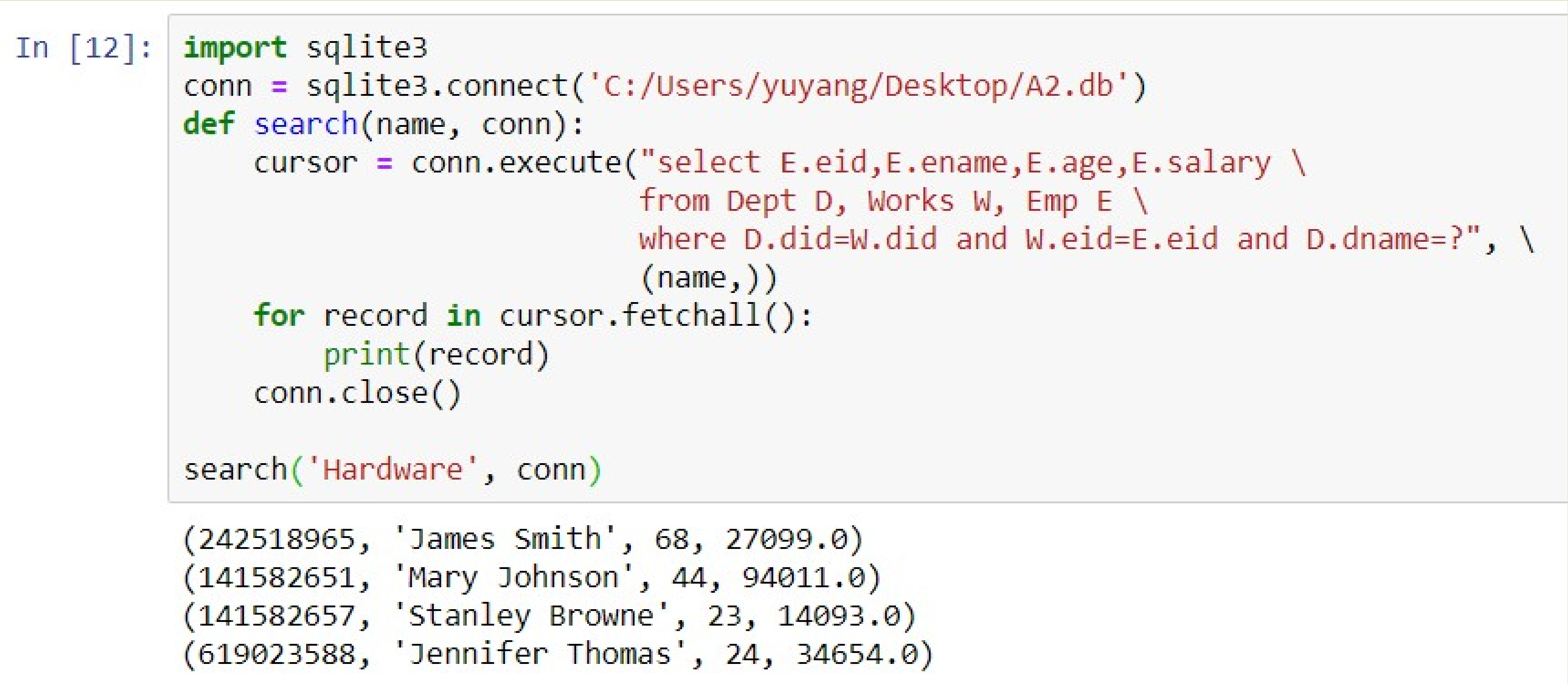
Cursor

- Cursor similar to an iterator
  - cursor = conn.execute(“SELECT * FROM student”)

- Cursor methods
  - fetchone()
  - fetchall()

Cursor

- Cursor similar to an iterator



In [12]:

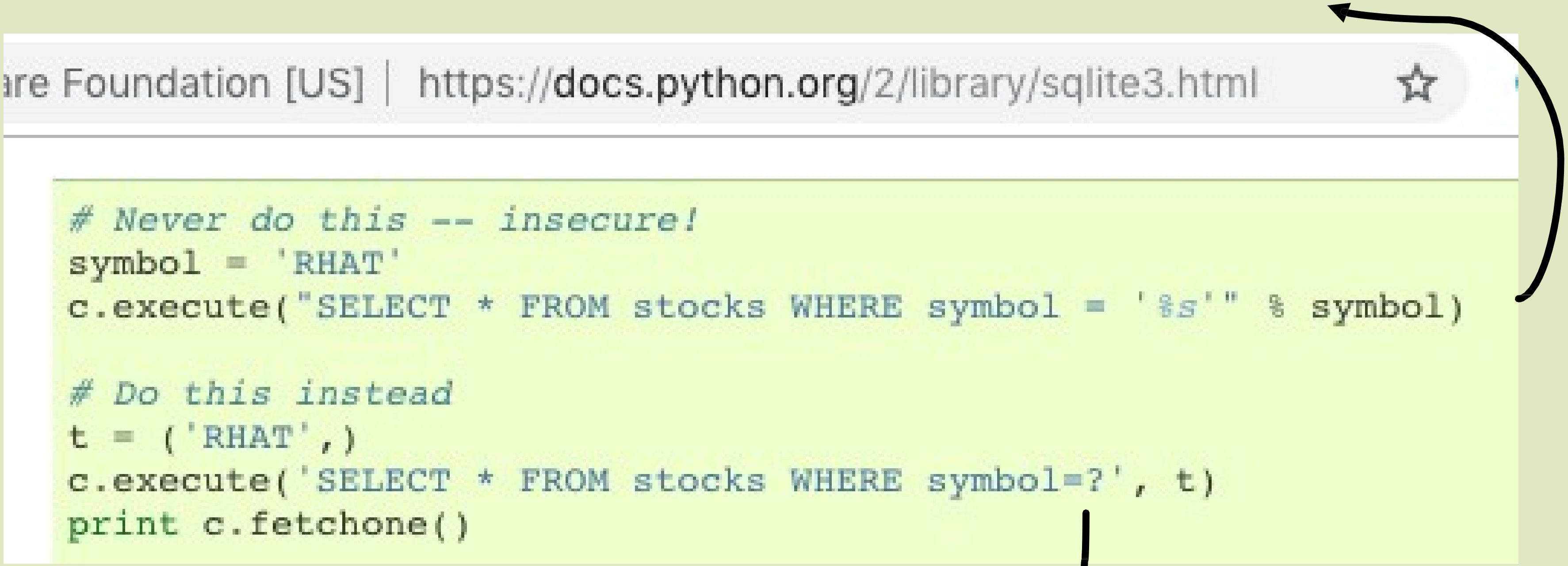
```
import sqlite3
conn = sqlite3.connect('C:/Users/yuyang/Desktop/A2.db')
def search(name, conn):
    cursor = conn.execute("select E.eid,E.ename,E.age,E.salary \
                          from Dept D, Works W, Emp E \
                          where D.did=W.did and W.eid=E.eid and D.dname=?", \
                          (name,))
    for record in cursor.fetchall():
        print(record)
    conn.close()

search('Hardware', conn)
```

```
(242518965, 'James Smith', 68, 27099.0)
(141582651, 'Mary Johnson', 44, 94011.0)
(141582657, 'Stanley Browne', 23, 14093.0)
(619023588, 'Jennifer Thomas', 24, 34654.0)
```

SQL Injection!!

symbol = “RHAT’ OR True -- ”

SELECT * **FROM** stocks **WHERE** symbol = 'RHAT' OR True -- '



are Foundation [US] | https://docs.python.org/2/library/sqlite3.html ☆

```
# Never do this -- insecure!
symbol = 'RHAT'
c.execute("SELECT * FROM stocks WHERE symbol = '%s'" % symbol)

# Do this instead
t = ('RHAT',)
c.execute('SELECT * FROM stocks WHERE symbol=?', t)
print c.fetchone()
```

SELECT * **FROM** stocks **WHERE** symbol = 'RHAT'' OR True -- '

Stored Procedures

- A *stored procedure* is a function / procedure written in a general-purpose programming language that is executed within the DBS.
- Performs computations that cannot be expressed in SQL.
- Procedure executed through a single SQL statement.
- Executed in the process space of the DB server.
- SQL standard: *PSM* (Persistent Stored Modules). Extends SQL by basic concepts of a general-purpose programming language.

Benefits of Stored Procedures

- Stored procedures are modular
  - It is easier to change a stored procedure than to edit an embedded query
  - This makes it easier to maintain stored procedures, and to
  - Change the procedure to increase its efficiency
- Stored procedures are registered with the DB server
  - They can be used by multiple applications and
  - Avoid tuple-at-a-time return of records through cursors
  - Separate server-side functions from client-side functions

Stored Procedures: Examples

```
CREATE PROCEDURE ShowNumReservations
    SELECT S.sid, S.sname, COUNT(*)
    FROM Sailors S, Reserves R
    WHERE S.sid = R.sid
    GROUP BY S.sid, S.sname
```

Stored procedures can have parameters:

- Three different modes: IN, OUT, INOUT

```
CREATE PROCEDURE IncreaseRating(
    IN sailor_sid INTEGER, IN increase INTEGER)
    UPDATE Sailors
    SET rating = rating + increase
    WHERE sid = sailor_sid
```

Stored Procedures: Examples (Contd.)

Stored procedure do not have to be written in SQL:

```
CREATE PROCEDURE TopSailors(
    IN num INTEGER)
LANGUAGE JAVA
EXTERNAL NAME “file:///c:/storedProcs/rank.jar”
```

Main SQL/PSM Constructs (Contd.)

- Local variables (DECLARE)
- RETURN values for FUNCTION
- Assign variables with SET
- Branches and loops:
  - IF (condition) THEN statements;
    ELSEIF (condition) statements;
    … ELSE statements; END IF;
  - LOOP statements; END LOOP
- Queries can be parts of expressions
- Can use cursors without “EXEC SQL”

Calling Stored Procedures

```
EXEC SQL BEGIN DECLARE SECTION
Int sid;
Int rating;
EXEC SQL END DECLARE SECTION

// now increase the rating of this sailor
EXEC SQL CALL IncreaseRating(:sid,:rating);
```

SQL/PSM

Most DBMSs allow users to write stored procedures in a simple, general-purpose language (close to SQL) → SQL/PSM standard is a representative

Declare a stored procedure:

```
CREATE PROCEDURE name(p1, p2, …, pn)
   local variable declarations
   procedure code;
```

Declare a function:

```
CREATE FUNCTION name (p1, …, pn) RETURNS
   sqlDataType
   local variable declarations
   function code;
```

Main SQL/PSM Constructs

```
CREATE FUNCTION rate Sailor
     (IN sailorId INTEGER)
     RETURNS INTEGER
DECLARE rating INTEGER
DECLARE numRes INTEGER
SET numRes = (SELECT COUNT(*)
               FROM Reserves R
               WHERE R.sid = sailorId)
IF (numRes > 10) THEN rating =1;
ELSE rating = 0;
END IF;
RETURN rating;
```

SQL Server Version

```
CREATE FUNCTION rateSailor (@sailorId INT)
  RETURNS INT
AS
BEGIN
  DECLARE @numRes INT
  DECLARE @rating INT
        SET @numRes = (SELECT COUNT(*)
                   FROM Reserves R
                WHERE R.sid = @sailorId)
  IF @numRes > 10
     SET @rating = 1
  ELSE
     SET @rating = 0
  RETURN @rating
END
GO;
SELECT dbo.rateSailor(22); go
```

Outline

- Database Programming
- **Application Architecture**

Application Architectures

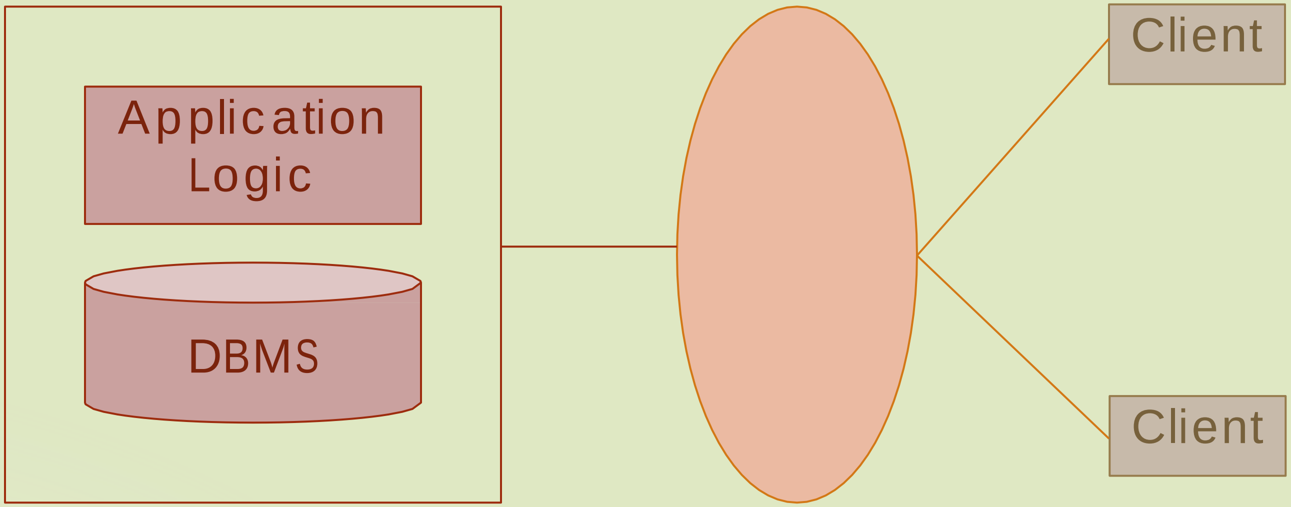
- Single tier
  - How things used to be …
- Two tier
  - Client-server architecture
- Three tier (and multi-tier)
  - Used for many web systems
  - Very scalable

Single Tier Architecture

- Historically, data intensive applications ran on a single tier which contained
  - The DBMS,
  - Application logic and business rules, and
  - User interface

Two-Tier Architecture

- Client/ server architecture
  - The server implements the business logic and data management
- Separate presentation from the rest of the application



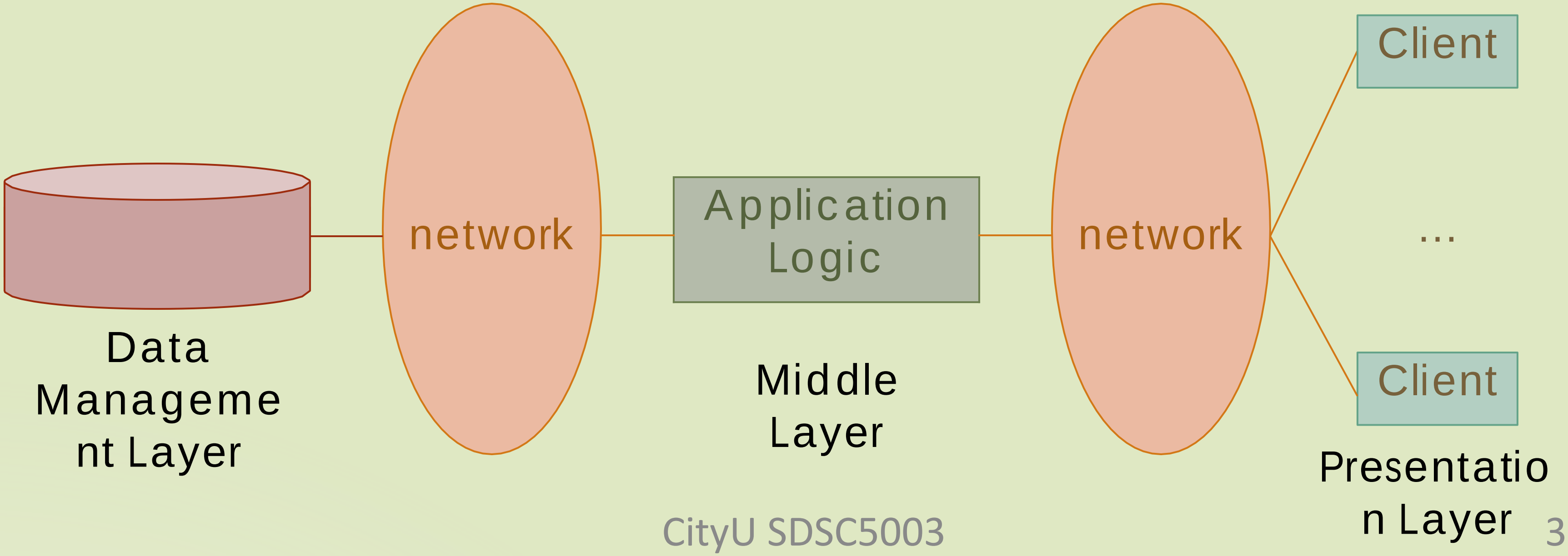
Presentation Layer

- Responsible for handling the user's interaction with the middle tier

- One application may have multiple versions that correspond to different interfaces
  - Web browsers, mobile phones, ...
  - Style sheets can assist in controlling versions

Three-Tier Architecture

- Separate presentation from the rest of the application
- Separate the application logic from the data management



Business logic Layer

- The middle layer is responsible for running the business logic of the application which controls
  - What data is required before an action is performed
  - The control flow of multi-stage actions
  - Access to the database layer

- Multi-stage actions performed by the middle tier may require database access
  - But will not usually make permanent changes until the end of the process
    - e.g. adding items to a shopping basket in an Internet shopping site

Data Management Layer

- The data management tier contains one, or more databases
  - Which may be running on different DBMSs
- Data needs to be exchanged between the middle tier and the database servers
  - This task is not required if a single data source is used but,
  - May be required if multiple data sources are to be integrated
  - *XML* is a language which can be used as a data exchange format between database servers and the middle tier

Example: Airline reservations

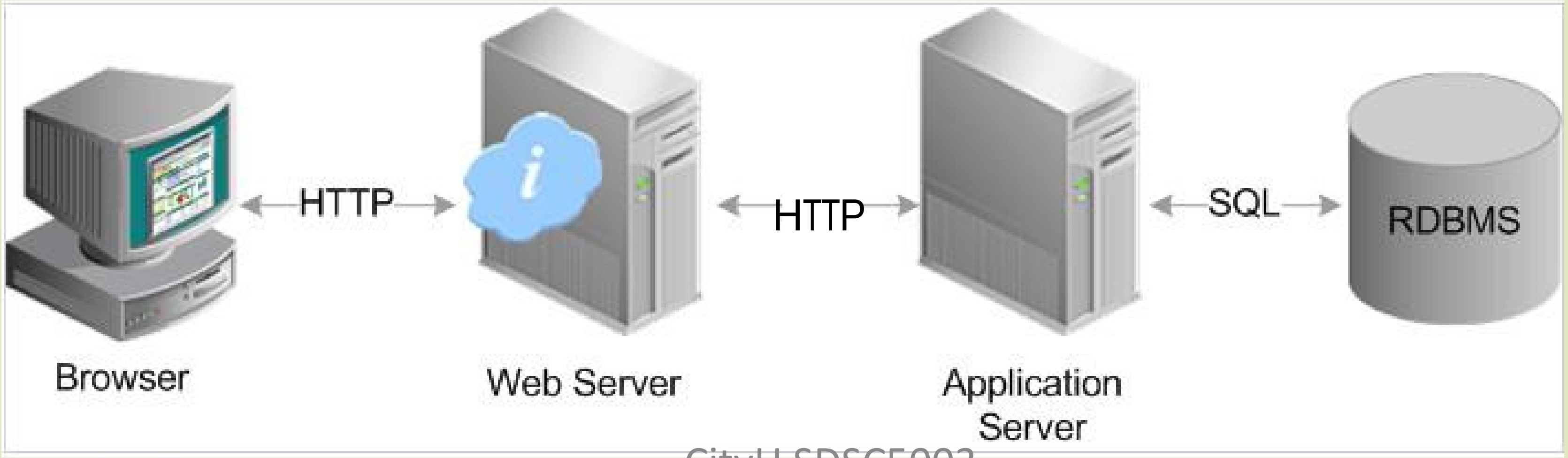
- Consider the three tiers in a system for airline reservations
- Database System
  - Airline info, available seats, customer info, etc.
- Application Server
  - Logic to make reservations, cancel reservations, add new airlines, etc.
- Client Program
  - Log in different users, display forms and human-readable output

Example: Course Enrollment

- Student enrollment system tiers
- Database System
  - Student information, course information, instructor information, course availability, pre-requisites, etc.
- Application Server
  - Logic to add a course, drop a course, create a new course, etc.
- Client Program
  - Log in different users (students, staff, faculty), display forms and human-readable output

3 Tier Architecture and the Web

- In the domain of web applications three tier architecture usually refers to
  - Web server
  - Application server
  - Database server



Summary

- Database Programming
  - Embedded SQL
  - DB API
- Application Architecture
  - Three Tier Architecture