BLG 317E
DATABASE SYSTEMS

WEEK 6

Slides credit:
S. Shivakumar, CS 145, Stanford University
Database System Concepts, Silberschatz, Korth, Sudarshan

ITU

Structured Query Language (SQL)
View and Index Management

CREATE VIEW Statement

- A view is a virtual table based on the result-set of an SQL statement.
- A view contains rows and columns, just like a real table.
- The fields in a view are fields from one or more real tables in the database.
- A view can be used in **SELECT** statements as if the data were coming from one single table.
- Actually a view does not contain its own data, it just selects data from real tables, every time the view is used in an SQL statement.
- INSERT and UPDATE commands can not be used on views.
- A view always shows up-to-date data.The database engine recreates the view, every time a user queries it.
- A view is created with the CREATE VIEW statement.

CREATE VIEW Statement

The SELECT command is a part of the CREATE VIEW statement.

Syntax

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM   table_names
WHERE  condition;
```

Example: CREATE VIEW Statement

- The following **CREATE VIEW** command creates a view (virtual table) in the database, by using the specified SELECT statament.
- **STU_COURSE** is the name of the view.

```
CREATE VIEW  STU_COURSE  AS
SELECT student.student_id,student_name, student.dept_id,
       course.course_id, course_title, grade
from STUDENT, TAKES, TEACHES, COURSE
where student.student_id  = takes.student_id AND
      takes.CRN           = teaches.CRN AND
      teaches.course_id   = course.course_id
order by student_name;
```

Example1: Query on a View

Query : Get all records from the **STU_COURSE** view.

```
select * from STU_COURSE;
```

Result

student_id	student_name	dept_id	course_id	course_title	grade
98765	Blake	EE	FIN-201	Accounting Methods	C-
98765	Blake	EE	PHY-101	Optics Principles	B
98988	Bradley	BIO	CS-101	Intro. to Algorithms	A
98988	Bradley	BIO	CS-315	Robotics	B
76543	Brown	CS	FIN-201	Accounting Methods	A
76543	Brown	CS	HIS-351	World History	A
19991	Brown	HIS	CS-190	Web Design	B
23121	Chavez	FIN	BIO-301	Molecular Biology	C+
12860	Clark	CS	CS-347	Database System Concepts	A
12860	Clark	CS	CS-319	Image Processing	A-
76653	Hudson	EE	BIO-101	Intro. to Biology	C
12345	Patel	CS	CS-347	Database System Concepts	C
12345	Patel	CS	MU-199	Music Production	A
12345	Patel	CS	PHY-101	Optics Principles	A
12345	Patel	CS	CS-319	Image Processing	A
55739	Sanchez	MU	CS-319	Image Processing	A-
45678	Walker	PHY	CS-347	Database System Concepts	C
45678	Walker	PHY	HIS-351	World History	B
54321	Williams	CS	FIN-201	Accounting Methods	A-
54321	Williams	CS	MU-199	Music Production	B+
44553	Young	PHY	EE-184	Intro. to Digital Systems	B-

Example2: Query on a View

Query : Get department names, student names of the departments, course titles of students, and the grades for each course taken by students. The **STU_COURSE** view will be joined by the DEPARTMENT table.

```
select dept_name, student_name,
       course_title, grade
from   department, STU_COURSE
where department.dept_id = STU_COURSE.dept_id
order by dept_name, STU_COURSE.student_name;
```

Result of query

dept_name	student_name	course_title	grade
Biology	Bradley	Intro. to Algorithms	A
Biology	Bradley	Robotics	B
Computer Science	Brown	Accounting Methods	A
Computer Science	Brown	World History	A
Computer Science	Clark	Database System Concepts	A
Computer Science	Clark	Image Processing	A-
Computer Science	Patel	Database System Concepts	C
Computer Science	Patel	Music Production	A
Computer Science	Patel	Optics Principles	A
Computer Science	Patel	Image Processing	A
Computer Science	Williams	Accounting Methods	A-
Computer Science	Williams	Music Production	B+
Electronics Eng.	Blake	Accounting Methods	C-
Electronics Eng.	Blake	Optics Principles	B
Electronics Eng.	Hudson	Intro. to Biology	C
Finance	Chavez	Molecular Biology	C+
History	Brown	Web Design	B
Music	Sanchez	Image Processing	A-
Physics	Walker	Database System Concepts	C
Physics	Walker	World History	B
Physics	Young	Intro. to Digital Systems	B-

The following statement drops (deletes) the view from database.

```
DROP VIEW  STU_COURSE;
```

CREATE INDEX Statement

- Indexes are not constraints.
- They are used optionally, in order to increase processing speed of the SELECT statements.
- Indexes are used to retrieve data from the database more quickly.
- The CREATE INDEX statement is used to create indexes in tables.
- Updating a table with indexes takes more time than updating a table without. Because the indexes also need an update.
- Only create indexes on columns that will be frequently searched against.

CREATE INDEX Statement

- **CREATE INDEX** : Duplicate values are allowed.
- **CREATE UNIQUE INDEX** : Duplicate values are not allowed.

Syntax

```
CREATE  INDEX  index_name
ON  table_name (column1,
                column2,
                ...);
```

EXAMPLE1: Index on student_name field only.

```
CREATE INDEX name_ix
ON STUDENT (student_name);
```

EXAMPLE2: Delete an existing index in a table.

```
ALTER TABLE STUDENT
DROP INDEX name_ix;
```

EXAMPLE2: Index on both dept_id field and student_name field.

```
CREATE INDEX dept_ve_name_ix
ON STUDENT (dept_id, student_name);
```

- The index helps speed the query below.

```
select dept_id, student_name
from STUDENT
where dept_id = 'CS' and student_name = 'Ali';
```

- It does not help speed the query below.

```
select student_name
from STUDENT
where gpa > 3.0;
```

Structured Query Language (SQL)
Application Development

Application Development with Database Program Libraries

- Database application programs offer user interfaces between users, and the DBMS.
- DBMSs offer built-in (ready-made) program libraries for different programming languages.
- The program libraries are also called as Drivers, Connectors, or API (Application Programming Interface).
- The following database drivers can be used for MySQL databases.
  ODBC, C API, C++, Python, PHP, JDBC, etc.

Steps of Commands for using a Database

- Server-based relational database management systems such as MySQL might contain multiple databases, on the same server.
- To interact with a database, firstly a connection should be established with the server.
- The followings are general steps of a program (any language) that interacts with a relational database server.

  1) Connect to the server.

  2) Connect to an existing database.
  Or, Create a new database, then connect to it.

  3) Execute SQL queries such as INSERT, DELETE, UPDATE, SELECT.

  4) For SELECT commands, fetch the result-set (cursor) by looping through each row in result-set. For DML queries, Commit changes.

  5) Close the connection to the server.

Connection Function Parameters

The connection function takes in the following parameters and returns a MySQLConnection object.

Connection Parameter	Description
Host name : Port number	Host name is name of database server. For testing purposes localhost is used. Also IP number 127.0.0.1 can be used.
	Port number is optional. Default port number in MySQL is 3306. (localhost:3306, or 127.0.0.1:3306)
User name, Password	Default user name in MySQL is root. Password is given during installation.
Database name	Default database name in MySQL is sys.

Python DBAPI

- import driver module
- rename for easier porting to other drivers

Example

```
import mysql.connector as dbapi   # mysql
import psycopg2 as dbapi          # postgresql
import sqlite3 as dbapi           # sqllite
```

Connection

- connection info: username, password, host, port, database name
- data source name (DSN):
  `user=.., password=.., host=.., port=.., dbname=..`
- uniform resource identifier (URI):
  `protocol://user:password@host:port/dbname`

Examples

```
host = "localhost", port = 3306, user = "ali",
    password="12345", database="university"
```

```
postgres://ali:12345@localhost:5432/university
```

Connection Example

```
connection = dbapi.connect(host = "localhost", port = 3306,
                           user = "ali", password="12345",
                           database="University")


#
# database operations
#

connection.close()
```

Update Operations

- For DML/DDL operations (insert, delete, update, create, . . .)
  - create a cursor on the connection
  - execute statement(s) on the cursor
  - commit pending changes on the connection
  - close the cursor

Update Operation Example

```
cursor = connection.cursor()
statement = """CREATE TABLE PERSON (
                    ID PRIMARY KEY AUTO_INCREMENT,
                    NAME VARCHAR(40) UNIQUE NOT NULL)"""

cursor.execute(statement)
connection.commit()

cursor.close()
connection.close()
```

Retrieve Operations

- For retrieval operations (select)
  - create a cursor on the connection
  - execute statement on the cursor
  - iterate over rows on the cursor (every row is a tuple)
  - close the cursor

Retrieve Operation Example

```
cursor = connection.cursor()
statement = """SELECT TITLE, SCORE FROM MOVIE
                WHERE (YR = 1999)"""
cursor.execute(statement)
for title, score in cursor:
    print('%(tt)s: %(sc)s' % {'tt': title, 'sc': score})

cursor.close()
connection.close()
```

Error Handling

- catch database related exceptions
  - rollback operation on error (`except`)
  - close all opened resources (`finally`)

Template

```
try:
    cursor = connection.cursor()
    cursor.execute(statement)
    connection.commit()
except dbapi.DatabaseError:
    connection.rollback()
finally:
    cursor.close()
    connection.close()
```

Connection Context Managers

- in some drivers, connections are context managers: **`with`**
- automatic commit (try), rollback (except), close (finally)
- template:

```
with dbapi.connect(...) as connection:
    cursor = connection.cursor()
    cursor.execute(statement)
    cursor.close()
```

Connection Context Manager Example

```
with dbapi.connect(...) as connection:
    cursor = connection.cursor()
    statement = """CREATE TABLE MOVIE (
        ID SERIAL PRIMARY KEY,
        TITLE VARCHAR(80),
        YR NUMERIC(4),
        SCORE FLOAT,
        VOTES INTEGER DEFAULT 0,
        DIRECTORID INTEGER REFERENCES PERSON (ID)
    )"""
    cursor.execute(statement)
    cursor.close()
```

Cursor Context Managers

- in some drivers, cursors are also context managers
  - automatic close
- template:

```
with dbapi.connect(...) as connection:
    with connection.cursor() as cursor:
        cursor.execute(statement)
```

Cursor Context Manager Example

```
with dbapi.connect(...) as connection:
    with connection.cursor() as cursor:
        statement = """CREATE TABLE CASTING (
            MOVIEID INTEGER REFERENCES MOVIE (ID),
            ACTORID INTEGER REFERENCES PERSON (ID),
            ORD INTEGER,
            PRIMARY KEY (MOVIEID, ACTORID)
        )"""
        cursor.execute(statement)
```

SQL Injection

- SQL injection is a hacking technique that might destroy the database.
- It is the placement of harmful code in SQL statements, usually via a user interface program input, entered by a hacker.
- Usually occurs when a user interface program (such as a web page login screen) asks a user for input.
- For example, program asks user to enter his username.
- Instead of a username, the user enters an SQL statement that will run on the database.

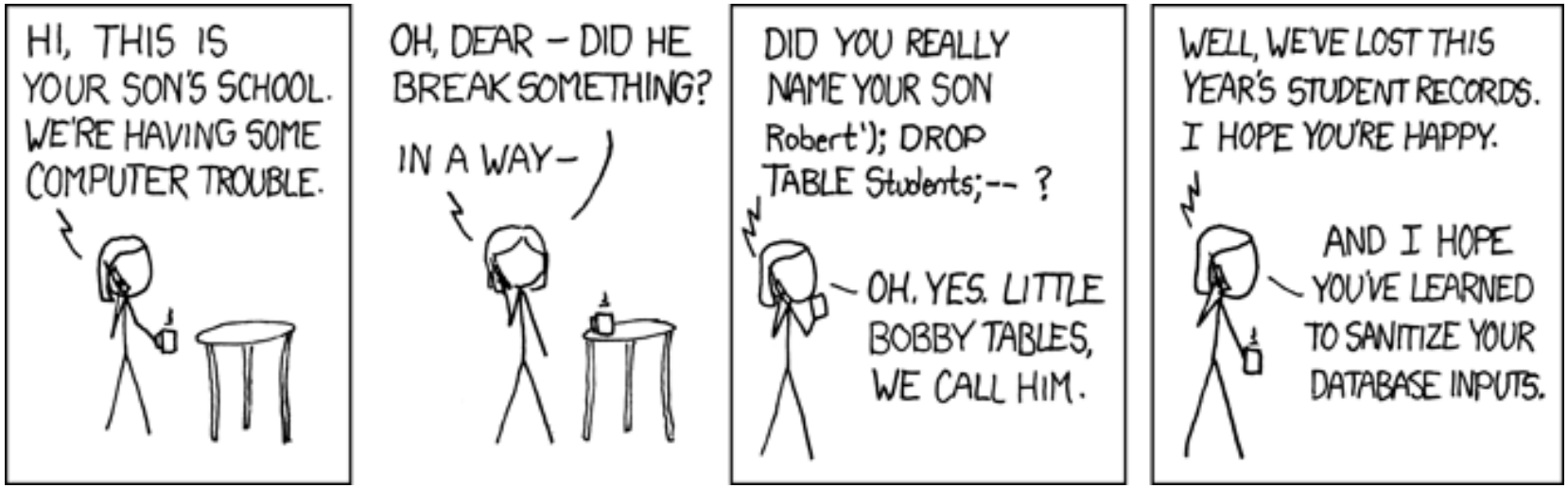
SQL Injection

- unsafe to use string formatting for constructing statements
- never trust inputs from outside sources
  - SQL injection attacks

Bad example

```
name = input('What is your name? ')
statement = """INSERT INTO Students (Name)
                    VALUES ('%s')""" % name
cursor.execute(statement)
```

SQL Injection Example



http://xkcd.com/327/

```
INSERT INTO Students (Name)
  VALUES ('Robert'); DROP TABLE Students;-- ')
```

Placeholders

- placeholders for values
  - different drivers use different formats: `%s`, `?`, ...
  - provide actual parameters as tuples or dictionaries

Placeholder Examples

- using tuples:

```
statement = """INSERT INTO MOVIE (TITLE, YR)
                    VALUES (%s, %s)"""
cursor.execute(statement, (title, year))
```

- using dictionaries:

```
statement = """INSERT INTO MOVIE (TITLE, YR)
                    VALUES (%(title)s, %(yr)s)"""
cursor.execute(statement, {'yr': year, 'title': title})
```

Placeholder Examples

- using tuples:

```
statement = """INSERT INTO MOVIE (TITLE, YR)
                    VALUES (%s, %s)"""
cursor.execute(statement, (title, year))
```

- using dictionaries:

```
statement = """INSERT INTO MOVIE (TITLE, YR)
                    VALUES (%(title)s, %(yr)s)"""
cursor.execute(statement, {'yr': year, 'title': title})
```

Fetching Results At Once

fetching results instead of iterating over cursor:

```
.fetchall()
.fetchone()
```

Fetch Example

- people and movies they directed

```
statement = """SELECT ID, NAME FROM PERSON"""
cursor.execute(statement)
people  =  cursor.fetchall()
for person_id, name in people:
    statement = """SELECT TITLE FROM MOVIE
                   WHERE (DIRECTORID = %s)"""
    cursor.execute(statement, (person_id,))
    directed = cursor.fetchall()
    print('Titles:')
    for (title,) in directed:
        print('  %(tt)s' % {'tt': title})
```

SQL Injection – More Examples
A Sample Database

- Suppose the following existing database is used by a Python program.
- Program asks user to enter an Author ID number from keyboard.
- Then program makes a search on the AUTHOR table, by executing the SQL SELECT statement.
- Finally, result of the search is displayed on screen.

PUBLICATIONS

Database Schema

```
PUBLISHER (publisher_id, publisher_name, phone)

AUTHOR (author_id, author_name, email)

BOOK (ISBN, publisher_id, author_id, title)
```

Example : Python Query Program

Part1

```
import mysql.connector

conn = mysql.connector.connect(user='root',
           password='', host='localhost',
           database='PUBLICATIONS')

cursor = conn.cursor()

search_id = input("Enter AuthorID to search : ")
print("The entered ID is : ", search_id)

query = "select * from AUTHOR
         where author_id=" + search_id;

print("The query is : ", query)
cursor.execute(query)
```

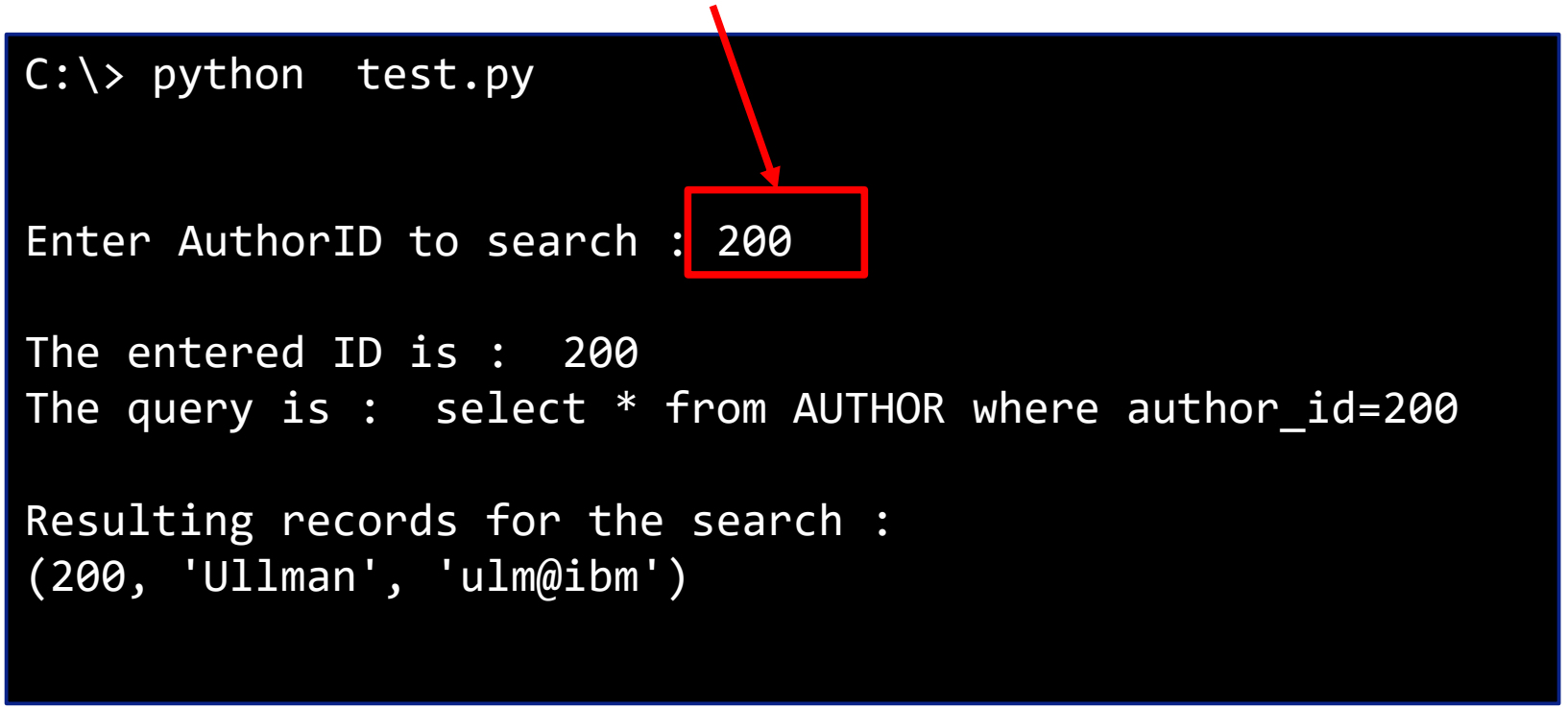
Part2

```
counter=0

print("Resulting records for the search :");

for row in cursor:
  print(row)
  counter = counter + 1


if counter == 0 :
   print("Searched ID not found")

cursor.close()
conn.close()
```

- In the program, the **search_id** is used as a string.
- The user might enter an SQL injection string, as the value of search_id, instead of entering an ID number.
- For security, the Python program should convert the string input to an integer number, or better use a placeholder, before sending the SQL statement to database server.

Example Testing1 : User enters valid input

- The searched ID is found in the AUTHOR table.



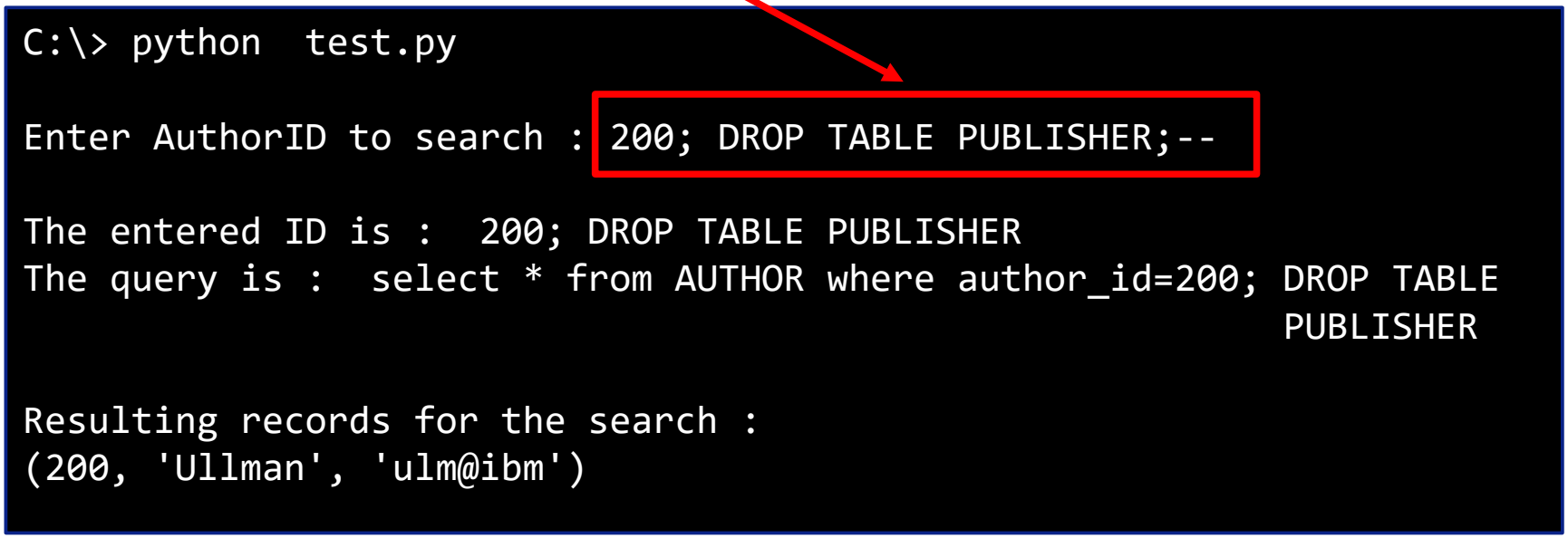
Example Testing2 : User enters injection input

- User enters an SQL injection string : **"200 OR True"** from keyboard, as the value of **author_id**.
- The OR clause always returns True for all author_id numbers.
- In result, database server returns all records in the AUTHOR table.

```
C:\> python  test.py

Enter AuthorID to search : 200 OR True
The entered ID is :  200 OR True
The query is :  select * from AUTHOR where author_id=200 OR True

Resulting records for the search :
(100, 'Elmasri', 'elm@uta')
(200, 'Ullman', 'ulm@ibm')
(300, 'Weinberg', None)
(400, 'Silberschatz', 'slb@ms')
(500, 'Ramakrishnan', 'rma@wisc')
(600, 'Campbell', 'aaa@bbb')
```

Example Testing3 : User enters injection input

- User enters the following SQL injection string from keyboard. **“200; DROP TABLE PUBLISHER;--”** , as the value of **author_id**.
- In result, database server might drop (delete completely) the PUBLISHER table.



References

Supplementary Reading

- Python Database API Specification: https://www.python.org/dev/peps/pep-0249/

Problem

- mismatch between data model and software model
  - data is relational: relation, tuple, foreign key, ...
  - software is object-oriented: object, reference, ...

Mismatch Example

- adding an actor to a movie: SQL definitions

```
CREATE TABLE MOVIE (ID INTEGER PRIMARY KEY,
    TITLE VARCHAR(80) NOT NULL)

CREATE TABLE PERSON (ID INTEGER PRIMARY KEY,
    NAME VARCHAR(40) NOT NULL)

CREATE TABLE CASTING (
    MOVIEID INTEGER REFERENCES MOVIE (ID),
    ACTORID INTEGER REFERENCES PERSON (ID),
    PRIMARY KEY (MOVIEID, ACTORID)
)
```

Mismatch Example

- adding an actor to a movie: SQL operations

```
INSERT INTO MOVIE (ID, TITLE)
  VALUES (110, 'Sleepy Hollow')

INSERT INTO PERSON (ID, NAME)
  VALUES (26, 'Johnny Depp')

INSERT INTO CASTING (MOVIEID, ACTORID)
  VALUES (110, 26)
```

Mismatch Example

- adding an actor to a movie: Python definitions

```
class Person:
    def __init__(self, name):
        self.name = name

class Movie:
    def __init__(self, title):
        self.title = title  self.cast = []

    def add_actor(self, person):
        self.cast.append(person)
```

Mismatch Example

- adding an actor to a movie: Python operations

```
movie = Movie('Sleepy Hollow')
actor = Person('Johnny Depp')
movie.add_actor(actor)
```

Object/Relational Mapping

- map software components to database components
- translate the object interface into SQL statements

model	SQL	software
relation	table	class
tuple	row	object (instance)
attribute	column	attribute

SQLAlchemy

- abstraction over SQL expressions
- object-relational mapper
  - regular Python class
  - SQL table description
  - mapper maps class to table

Connection Example

```
from sqlalchemy import create_engine

uri = 'mysql+mysqlconnector://ali:12345@localhost/Sandbox'

engine = create_engine(uri, echo=True)

from sqlalchemy import MetaData

metadata = MetaData()
```

Connection Example

```
from sqlalchemy import create_engine

uri = 'mysql+mysqlconnector://ali:12345@localhost/Sandbox'

engine = create_engine(uri, echo=True)

from sqlalchemy import MetaData

metadata = MetaData()
```

Class Example

```
class Movie:
    def __init__(self, title, year=None,
                 score=None, votes=None):
        self.title = title
        self.yr = year
        self.score = score
        self.votes = votes
```

Table Example

```
from sqlalchemy import Column, Table
from sqlalchemy import Float, Integer, String

movie_table = Table(
    'Movie', metadata,
    Column('id', Integer, primary_key=True),
    Column('title', String(80), nullable=False),
    Column('yr', Integer),
    Column('score', Float),
    Column('votes', Integer)
)
```

Mapper Example

```
from sqlalchemy.orm import mapper

mapper(Movie, movie_table)
```

Creating Tables

```
metadata.create_all(bind=engine)
```

```
CREATE TABLE `Movie` (
        id INTEGER NOT NULL AUTO_INCREMENT,
        title VARCHAR(80) NOT NULL,
        yr INTEGER,
        score FLOAT,
        votes INTEGER,
        PRIMARY KEY (id)
)
```

Sessions

- data operations are handled in sessions
- end with commit or rollback
- session keeps track of modified and new objects

Session Example

```
from sqlalchemy.orm import sessionmaker

Session = sessionmaker(bind=engine)
session = Session()
```

Session Example: Insert

```
movie = Movie('Casablanca', year=1942)
session.add(movie)
session.commit()
```

```
INSERT INTO `Movie` (title, yr, score, votes)
  VALUES (%(title)s, %(yr)s, %(score)s, %(votes)s)
{'yr': 1942, 'title': 'Casablanca', 'score': None,
 'votes': None}

# autogenerated id is assumed to be 1
```

Session Example: Update

```
movie.votes = 23283
session.commit()
```

```
UPDATE "Movie" SET votes=%(votes)s
 WHERE "Movie".id = %(Movie_id)s

{'Movie_id': 1, 'votes': 23283}
```

Session Example: Delete

```
session.delete(movie)
session.commit()
```

```
DELETE FROM "Movie"
 WHERE "Movie".id = %(id)s

{'id': 1}
```

Query Examples

```
session.query(Movie)
```

```
SELECT "Movie".id AS "Movie_id",
       "Movie".title AS "Movie_title",
       "Movie".yr AS "Movie_yr",
       "Movie".score AS "Movie_score",
       "Movie".votes AS "Movie_votes"
  FROM "Movie"
```

Query Examples: Selecting Columns

```
session.query(Movie.title, Movie.score)
```

```
SELECT "Movie".title AS "Movie_title",
       "Movie".score AS "Movie_score"
  FROM "Movie"
```

SQLAlchemy Example: Ordering

```
session.query(Movie).order_by(Movie.yr)
```

```
SELECT "Movie".id AS "Movie_id",
       "Movie".title AS "Movie_title",
       "Movie".yr AS "Movie_yr",
       "Movie".score AS "Movie_score",
       "Movie".votes AS "Movie_votes"
  FROM "Movie"
  ORDER BY "Movie".yr
```

SQLAlchemy Example: Selecting Rows

```
session.query(Movie).filter_by(yr=1999)
```

```
SELECT "Movie".id AS "Movie_id",
       "Movie".title AS "Movie_title",
       "Movie".yr AS "Movie_yr",
       "Movie".score AS "Movie_score",
       "Movie".votes AS "Movie_votes"
  FROM "Movie"
 WHERE "Movie".yr = %(yr_1)s

{'yr_1': 1999}
```

SQLAlchemy Example: Selecting Rows by Predicate

```
session.query(Movie).filter(Movie.yr < 1999)
```

```
SELECT "Movie".id AS "Movie_id",
       "Movie".title AS "Movie_title",
       "Movie".yr AS "Movie_yr",
       "Movie".score AS "Movie_score",
       "Movie".votes AS "Movie_votes"
  FROM "Movie"
 WHERE "Movie".yr < %(yr_1)s

{'yr_1': 1999}
```

Foreign Keys

- add foreign key columns to table definitions
- add a“relationship”property to the mapper
  - property name becomes attribute from source to target
  - backref parameter becomes attribute from target to source

Foreign Key Example

```
class Person:
    def __init__(self, name):
        self.name = name

person_table = Table(
    'Person', metadata,
    Column('id', Integer, primary_key=True),
    Column('name', String(40), nullable=False,
           unique=True)
)

mapper(Person, person_table)
```

Foreign Key Example

```
from sqlalchemy import ForeignKey

movie_table = Table(
    'Movie', metadata,
    Column('id', Integer, primary_key=True),
    Column('title',  String(80)),
    Column('yr', Integer),
    Column('score', Float),
    Column('votes', Integer),
    Column('directorid', Integer, ForeignKey('Person.id'))
)
```

Foreign Key Example

```
from sqlalchemy.orm import relationship

mapper(Movie, movie_table,
       properties={
           'director':
               relationship(Person,
                            backref='directed')
       })
```

Foreign Key Example

```
movie = session.query(Movie) \
                .filter_by(title='Ed Wood').first()
```

```
SELECT "Movie".id AS "Movie_id",
       "Movie".title AS "Movie_title",
       ...
       "Movie".directorid AS "Movie_directorid"
  FROM "Movie"
 WHERE "Movie".title = %(title_1)s

{'title_1': 'Ed Wood'}

# returned directorid is assumed to be 8
```

Foreign Key Example

```
person = movie.director
print(person.name)
```

```
SELECT "Person".id AS "Person_id",
       "Person".name AS "Person_name"
  FROM "Person"
 WHERE "Person".id = %(param_1)s

{'param_1': 8}
```

Backref Example

```
for movie in person.directed:
    print(movie.title)
```

```
SELECT "Movie".id AS "Movie_id",
       "Movie".title AS "Movie_title",
       ...
       "Movie".directorid AS "Movie_directorid"
  FROM "Movie"
 WHERE %(param_1)s = "Movie".directorid

{'param_1': 8}
```

Foreign Key Example

- over a secondary table

```
casting_table = Table(
    'Casting', metadata,
    Column('movieid', Integer, ForeignKey('Movie.id'),
           primary_key=True),
    Column('actorid', Integer, ForeignKey('Person.id'),
           primary_key=True),
    Column('ord', Integer)
)
```

```
CREATE TABLE `Casting` (
        movieid INTEGER NOT NULL,
        actorid INTEGER NOT NULL,
        ord INTEGER,
        PRIMARY KEY (movieid, actorid),
        FOREIGN KEY(movieid) REFERENCES `Movie` (id),
        FOREIGN KEY(actorid) REFERENCES `Person` (id)
)
```

Foreign Key Example

```
mapper(Movie, movie_table,
       properties={
           'director':
               relationship(Person,
                            backref='directed'),
           'cast':
               relationship(Person,
                            backref='acted',
                            secondary=casting_table)
       })
```

Foreign Key Example

```
for movie in session.query(Movie):
    print('%(tt)s' % {'tt': movie.title})
    for person in movie.cast:
        print('  %(nm)s:' % {'nm': person.name})

for person in session.query(Person):
    print('%(nm)s:' % {'nm': person.name})
    for movie in person.acted:
        print('  %(tt)s' % {'tt': movie.title})
```

Foreign Key Example

```
for movie in session.query(Movie):
    print('%(tt)s' % {'tt': movie.title})
    for person in movie.cast:
        print('  %(nm)s:' % {'nm': person.name})

for person in session.query(Person):
    print('%(nm)s:' % {'nm': person.name})
    for movie in person.acted:
        print('  %(tt)s' % {'tt': movie.title})
```

References

Supplementary Reading

- SQLAlchemy Documentation:
  http://docs.sqlalchemy.org/

Structured Query Language (SQL)
App Development – Extra Examples

MySQL Drivers (Libraries) for Python

- The officially recommended MySQL library for Python is **MySQL Connector/Python**, which is written in Python. It can be downloaded from MySQL website.

- Other alternative MySQL libraries for Python:
  - **Mysqlclient :** Written in C, and is faster.
  - **MySQLdb :** Legacy software, written in C.
  - **PyMySQL :** Written in Python.

- There are also high-level abstraction libraries such as SQLAlchemy-Object-Relational mapping (ORM). ORM library depends on a driver library.

Topics

- Application Development
- C Example
- Python - Desktop example
- Python - Web example
- Python - SQLAlchemy examples
- SQL Injection

MySQL Drivers (Libraries) for C

- MySQL DBMS is used for all example programs in the lecture.
- The following files are required for C example program.
- The C API (libmysqlclient) is a client library for C programming.
- When you install the MySQL Community Edition, these files are also installed.

Phase	Required MySQL Files	
	File name	File type
Compile time	`mysql.h`	Header file
Link time	`libmysql.lib` `mysqlclient.lib`	Static library files
Run time	`libmysql.dll`	Dynamic link library

Example: C program

Write the following C program with any text editor such as Notepad, or Visual Studio Code editor.

Part1

```
#include <mysql.h>    // Header file located in MySQL installation folder
#include <stdio.h>

int main()
{
   MYSQL       *conn;    // Pointer to struct
   MYSQL_RES   *results;    // Pointer to struct
   MYSQL_ROW   row;        // Array of struct

   char *sunucu     = "localhost";
                      // Alternative is IP number: "127.0.0.1"

   char *kullanici  = "root";          // Database user name
   char *sifre      = "****";          // User password
   char *veritabani = "PUBLICATIONS";  // Database name
```

Part2

```
conn = mysql_init(NULL);        // Initialize MySQL

// Connect to database server
if (! mysql_real_connect(conn, sunucu,
                         kullanici, sifre,
                         veritabani, 0, NULL, 0))
{
   printf("conn hatasi : %s \n",
           mysql_error(conn) );
   return 0;
}
```

Part3

```
// Insert records to tables:

strcpy(komut, "insert into PUBLISHER
values(40, 'Kodlab Yayinevi', '212-123-4567');");
mysql_query(conn, komut);

strcpy(komut, "insert into AUTHOR
values(600, 'Campbell', 'aaa@bbb');");
mysql_query(conn, komut);

strcpy(komut, "insert into BOOK
       values('Z0012345', 40, 600, 'Data Structures');");
mysql_query(conn, komut);
```

Part4

```
// Select query:
strcpy(aranan_kelime, "base");
// Alternative: scanf from keyboard.

strcpy(komut, "select title, author_name,
      publisher_name from BOOK, AUTHOR, PUBLISHER
      where BOOK.author_id = AUTHOR.author_id and
            BOOK.publisher_id = PUBLISHER.publisher_id
            and  title LIKE '%");
strcat(komut, aranan_kelime);
strcat(komut, "%' ");
strcat(komut, "order by title;");
// Send SQL query to server
if (mysql_query(conn, komut))
{
   printf("Select komutu hatasi : %s \n",
           mysql_error(conn) );
   return 0;
}
```

Part5

```
// Retrieve the resulting records
results = mysql_use_result(conn);

printf("C PROGRAM \n");
printf("Row#  Book Title              Author Name      Publisher Name \n");
printf("====  ====================  ==============  ====================== \n");

counter = 1;
// Print the fields of each row
 while ((alan = mysql_fetch_row(results)) != NULL)
 {
    printf("%4d  %-22s  %-15s  %-22s \n",
           counter, alan[0], alan[1], alan[2]);
    counter++;
 }

 // Close the connection
 mysql_free_result(results);
 mysql_close(conn);
 return 0;
} //End of program
```

Using GNU C Compiler and Linker

- **Compile** the C source program with following command-line switches.
  The **`-c`** switch tells the GNU C Compiler (**gcc.exe**) to compile the file only.
  The **`-I`** switch tells the path of include directory.
  Compiler generates an object file (example.o)

```
C:\> gcc.exe  -c  example.c  -o  example.o
     -I"C:/Program Files/MySQL/MySQL Server 8.0/include"
```

- **Link** the object program file, and the MySQL static library files with following command-line switch.
  The **`-o`** switch tells the GNU C Linker (**gcc.exe**) the name of executable file (example.exe)

```
C:\> gcc.exe  example.o  -o  example.exe
     "C:/Program Files/MySQL/MySQL Server 8.0/lib/libmysql.lib"
     "C:/Program Files/MySQL/MySQL Server 8.0/lib/mysqlclient.lib"
```

Running the Program

- First, by using the windows command **SET**, add the search path of the **libmysql.dll** file (Dynamic Link Library file), to the existing windows environment variable **PATH**.
- Then, **run** the executable program.

```
C:\> SET  PATH=%PATH%;"C:\Program Files\MySQL\MySQL Server 8.0\lib"


C:\> example.exe


C PROGRAM
Row#  Book Title                Author Name      Publisher Name
====  ======================    ===============  ======================
   1  Database Concepts         Silberschatz     McGraw-Hill
   2  Database Design           Elmasri          Pearson Prentice Hall
   3  Database Fundamentals     Elmasri          Addison-Wesley
   4  Database Management       Ramakrishnan     McGraw-Hill
   5  Database Systems          Ullman           Pearson Prentice Hall
```

SQL SELECT statement gets the book titles that contain “base” word.

Topics

- Application Development
- C Example
- Python - Desktop example
- Python - Web example
- Python - SQLAlchemy examples
- SQL Injection

Example: Python Desktop Program

- The following program uses MySQL Connector/Python library.
- It also uses the Graphical User Interface (GUI) library tkinter. (Tkinter : Tool Kit Interface)
- Program connects to the PUBLICATIONS database.
- It retrieves the book titles, author name and publisher name of each book.
- Output is written both to the console screen, and to the GUI (tkinter) screen.

Example: Python Desktop Program

Part1

```
import mysql.connector
from tkinter import *
# function
def read_from_database():
    conn = mysql.connector.connect(user='root',
                        password='', host='localhost',
                        database='PUBLICATIONS')
    cursor = conn.cursor()
    cursor.reset()

    query = '''select title, author_name, publisher_name
           from BOOK, AUTHOR, PUBLISHER
           where BOOK.author_id = AUTHOR.author_id and
                 BOOK.publisher_id = PUBLISHER.publisher_id
           order by title;'''
    cursor.execute(query)
```

Part2

```
formatted_row = '{0:>4}  {1:<22}  {2:<15}  {3:<22}'.
     format("ROW", "BOOK TITLE", "AUTHOR NAME",
            "PUBLISHER NAME")
print(formatted_row)  #Console screen
list_box.insert(END, formatted_row) #GUI screen

counter = 1
for (field1, field2, field3) in cursor:
   formatted_row = '{0:>4}  {1:<22}  {2:<15}
         {3:<22}'.format(counter, field1, field2, field3)

   print(formatted_row)  #Console screen
   list_box.insert(END, formatted_row) #GUI screen
   counter = counter + 1

cursor.close()
conn.close()
```

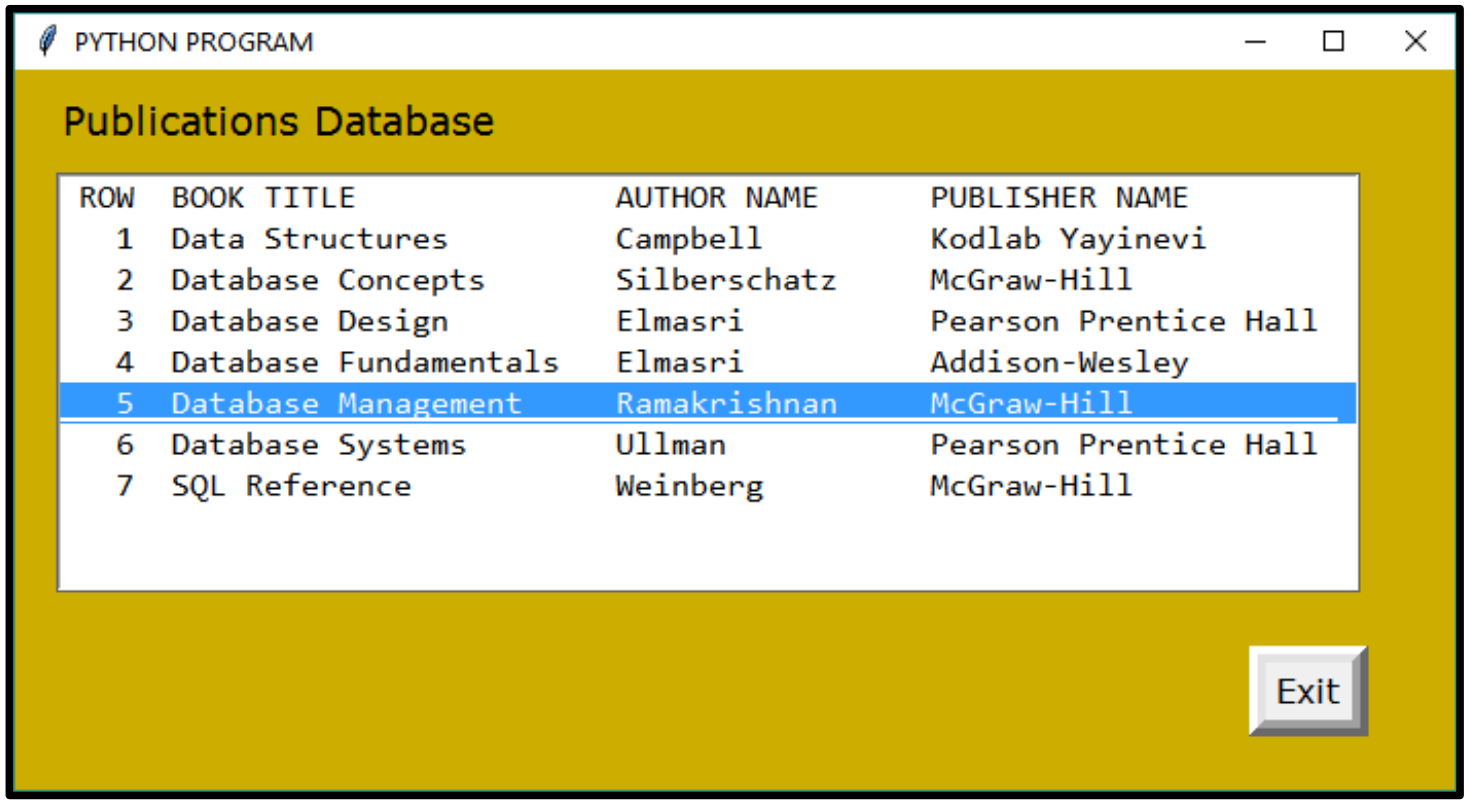
Part3

```
# MAIN PROGRAM
COLOR = 'gold3'
window = Tk()
window.title('PYTHON PROGRAM')
window.configure(background=COLOR)
window.geometry("700x350")
label1 = Label(window,text='Publications Database',
               font=('Verdana', 14),background=COLOR,
               justify='left')
label1.place(x=20, y=10)

list_box = Listbox(
    window,
    height=10,
    width=70,
    selectmode=SINGLE,
    font=('consolas', 12)
)
list_box.place(x=20, y=50)
```

Part4

```
button1 = Button(window, text='Exit',
                 font=('Verdana', 12),
                 command=window.destroy, borderwidth=8)

button1.place(x=600, y=280)

read_from_database()

window.mainloop()
```

Output of GUI Screen



Topics

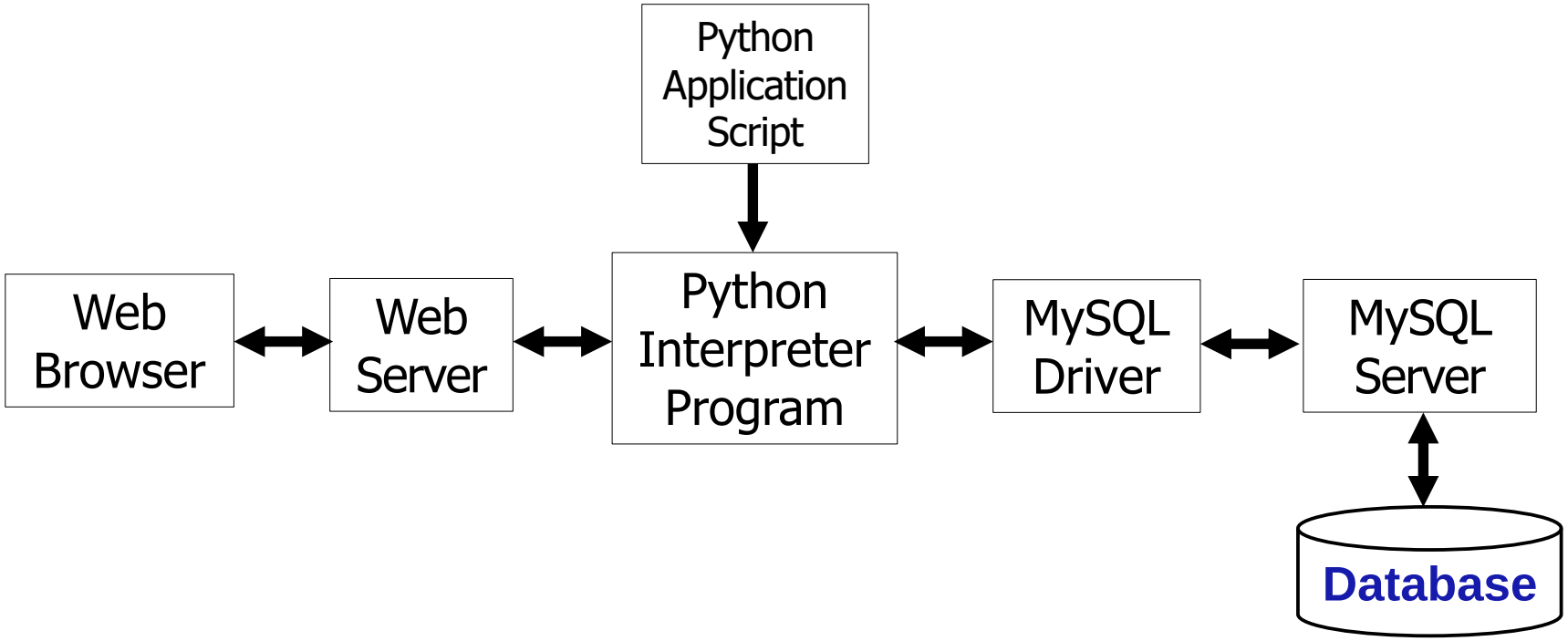
- Application Development
- C Example
- Python - Desktop example
- Python - Web example
- Python - SQLAlchemy examples
- SQL Injection

Web Application Components

- The following components should be installed.
- The web-server and Python interpreter program must be on same computer.
- The database server can be on same computer, or on a separate computer at a connected network.
- **Database server:** Responds to database requests made by the web application program. (MySQL will be used.)
- **Web server:** Any web-server such as Microsoft Internet Information Server (IIS), Apache, Flask, etc. can be used.
  The web-server should be configured to run Python scripts.
- **Python interpreter:** Runs the application program script, and generates an HTML output.
  (Also any other language such as PHP, Java, Javascript, C#, etc. can be used for web-server-side programming.)
- **Web browser:** Makes web requests from web server.

Web Application Components

For testing purpose, all of the components can be on the same computer.



Invoking the Application Program

- By using a web-browser, user enters the URL address of the Python script, which is located on the root directory of web-server.
- Example URL (Uniform Resource Locator):
  `http://localhost/example_web.py`
- The web server calls the Python interpreter to execute the application program script file.
- The Python script file calls MySQL library functions.
- Python interpreter generates an HTML output to web-server.
- Web-server sends the HTML file as respond to the web browser.

Example: Python Web Program

Part1

```
# Example web-based Python program for MySQL.

import mysql.connector

conn = mysql.connector.connect(user='root',
               password='', host='localhost',
               database='PUBLICATIONS')
cursor = conn.cursor()
cursor.reset()
query = '''select title, author_name, publisher_name
           from BOOK, AUTHOR, PUBLISHER
           where BOOK.author_id = AUTHOR.author_id and
                 BOOK.publisher_id = PUBLISHER.publisher_id
           order by title;'''

cursor.execute(query)
```

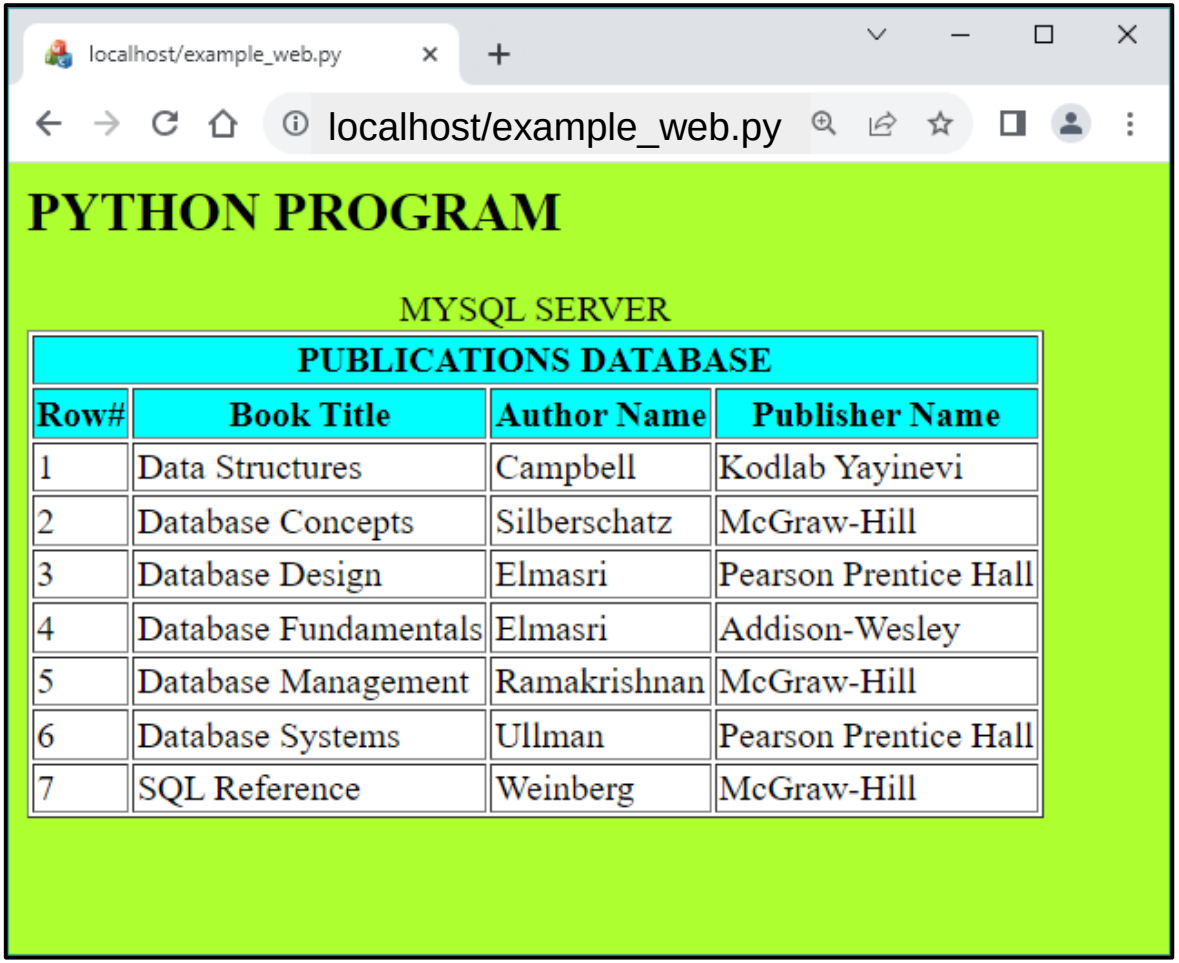
Part2

```
print("Content-Type: text/html\n")
print("<body bgcolor=greenyellow>");
print("<h2>PYTHON PROGRAM </h2>")
print("<table border=1 bgcolor=white >")
print("<caption>MYSQL SERVER</caption>")
print("<tr>")
print("<th colspan=4 bgcolor=aqua >PUBLICATIONS
                                    DATABASE</th>")
print("</tr>")
print("<tr  bgcolor=aqua>")
print("<th>Row#</th>")
print("<th>Book Title</th>")
print("<th>Author Name</th>")
print("<th>Publisher Name</th>")
print("</tr>")
```

Part3

```
counter = 1
for (field1, field2, field3) in cursor:
  print("<tr>")
  print('<td>{0:>4}</td>
         <td>{1:<22}</td>
         <td>{2:<15}</td>
         <td>{3:<22}</td>'.format(
                  counter, field1, field2, field3)
  )
  print("</tr>\n")
  counter = counter + 1

print("</table>")
print("</body>")
print("</html>")

cursor.close()
conn.close()
```

Browser Output of Program



PUBLICATIONS DATABASE

Row#	Book Title	Author Name	Publisher Name
1	Data Structures	Campbell	Kodlab Yayinevi
2	Database Concepts	Silberschatz	McGraw-Hill
3	Database Design	Elmasri	Pearson Prentice Hall
4	Database Fundamentals	Elmasri	Addison-Wesley
5	Database Management	Ramakrishnan	McGraw-Hill
6	Database Systems	Ullman	Pearson Prentice Hall
7	SQL Reference	Weinberg	McGraw-Hill

Topics

- Application Development
- C Example
- Python - Desktop example
- Python - Web example
- Python - SQLAlchemy examples
- SQL Injection

SQLAlchemy Library for Python

- SQLAlchemy is a high-level abstraction library for Python.
- It is used for object-oriented programming (classes and objects) and relational databases together.
- SQLAlchemy has dialects (versions) for many RDBMSs such as SQLite, Postgresql, MySQL, Oracle, MS-SQL, Sybase, etc.
- It consists of two components: the Core and the ORM.

Components of SQLAlchemy Library

Component	Description
SQLAlchemy Core	• It is a low-level SQL toolkit that provides a SQL abstraction layer and allows to work directly with relational databases using Python.
SQLAlchemy ORM	• It is a higher-level API built on top of SQLAlchemy Core. • It provides an Object Relational Mapper (ORM) for working with databases. • ORM allows to map Python classes to database tables, and to interact with the data in those tables. • ORM can simplify database operations.

Installing SQLAlchemy Library

From windows command-line, enter the following commands for online installation.
(PIP : Pip Installs Packages)

```
pip  install  sqlalchemy

pip  install  mysql-connector-python
```

Object-Relational Mapping (ORM)

- In object-oriented programming, data-management tasks use objects of classes.
- Relational database uses tables and records.
- An object-relational-mapper maps a relational database tables to Python class objects.
- The mapping is as follows:
  - **Python Class = SQLTable**
  - **Instance of the Class (object) = Row in theTable**

SQLAlchemy Connection String for MySQL

- The following is the syntax of connection string for MySQL.
- Driver name can be any MySQL driver, such as mysqlconnector.

```
mysql+<drivername>://<username>:<password>@<server>:<port>/dbname
```

Example1: Using SQLAlchemy Core
(Text SQL command)

- The example below uses a raw text SQL statement.
- Firstly, program connects to an existing database named Test.
- Then, SQL SELECT command is sent.
- Lastly, records (rows) in result-set are printed on screen in a loop.

```
from sqlalchemy import create_engine, text

# Connect to the database

engine = create_engine("mysql+mysqlconnector://root:@localhost/Test")

conn = engine.connect()

results = conn.execute( text("SELECT * FROM publisher2") )


for row in results:

   print(row)
```

Output of Program

```
C:\> python  example_sql.py


(101, 'Nobel Yayinevi', '212-12345')
(102, 'Pandora Yayinevi', '216-5678')
```

Example2: Using SQLAlchemy Core
(Built-in select() function of class)

- Program defines a Python object named **publishers_tablosu** (from the **Table** built-in class).
- The object is mapped to the database table named **publisher2**.
- The built-in select() function of the class is called.
- SQLAlchemy generates the necessary SQL commands automatically.

```
engine = create_engine("mysql+mysqlconnector:
         //root:@localhost/Test", echo=True);
meta = MetaData()

publishers_table = Table(
   'publisher2', meta,
   Column('publisher_id', Integer, primary_key = True),
   Column('publisher_name', String),
   Column('phone', String),
)
conn = engine.connect()
select_command = publishers_tablosu.select()

results = conn.execute(select_command)
for row in results:
   print(row)
```

Output of Program

- In the create_engine() function call, echo=True parameter is written.
- The echo is an optional parameter, used for debugging purposes.
- SQLAlchemy prints the SQL statement that it generated automatically, based on the Python commands.

```
C:\> python  example_core.py

664 INFO sqlalchemy.engine.Engine SELECT DATABASE()
664 INFO sqlalchemy.engine.Engine [raw sql] {}
664 INFO sqlalchemy.engine.Engine SELECT @@sql_mode
664 INFO sqlalchemy.engine.Engine [raw sql] {}
664 INFO sqlalchemy.engine.Engine SELECT @@lower_case_table_names
664 INFO sqlalchemy.engine.Engine [raw sql] {}

679 INFO sqlalchemy.engine.Engine BEGIN (implicit)
679 INFO sqlalchemy.engine.Engine SELECT publisher2.publisher_id,
                                       publisher2.publisher_name,
                                       publisher2.phone
                                  FROM publisher2
679 INFO sqlalchemy.engine.Engine [generated in 0.00545s] {}


(101, 'Nobel Yayinevi', '212-12345')
(102, 'Pandora Yayinevi', '216-5678')
```

Example3: Using SQLAlchemy ORM

- SQLAlchemy ORM library functions map the Python classes to database tables.
- Programmer performs SQL operations, without using any SQL command syntax.
- The example program below uses the ORM library functions.
  - Uses an existing database named Test.
  - Creates a new table named publisher2, adds records, then retrieves records.

Part1

```
from sqlalchemy      import Column, Integer, String
from sqlalchemy      import create_engine
from sqlalchemy.orm  import declarative_base
from sqlalchemy.orm  import sessionmaker

engine = create_engine("mysql+mysqlconnector:
                       //root:@localhost/Test");

base_class = declarative_base()

#Declaration of mapping from database table to Python class
class PublisherClass(base_class):
    __tablename__   = 'publisher2'
    publisher_id    = Column(Integer, primary_key=True)
    publisher_name = Column(String(50))
    phone           = Column(String(15))
```

Part2

```
#This command generates a CREATE TABLE sql command.
base_class.metadata.create_all(engine)
print("publisher2 table is created.")

#-------------------------------------------------------------
#Add records
Session = sessionmaker(bind = engine)
session = Session()

#This command generates INSERT INTO sql command.
session.add_all([
   PublisherClass(publisher_id = '101',
      publisher_name = 'Nobel Yayinevi', phone = '212-12345'),
   PublisherClass(publisher_id = '102',
     publisher_name = 'Pandora Yayinevi', phone = '216-5678')]
)

session.commit()
print("Rows are added to the table.")
```

Part3

```
#Select query returns an array of class objects
results = session.query(PublisherClass).all()
print("PUBLISHER2 Table \n");
print("ID No  Publisher Name          Phone");
print("=====  ======================  ==============");

#row is an object of PublisherClass class
#It corresponds to records in database table.

for row in results:
  field1 = row.publisher_id
  field2 = row.publisher_name
  if row.phone is None:
     field3 = "N/A"
  else:
     field3 = row.phone

  print('{0:>5}  {1:<22}  {2:<15}'.format(field1, field2, field3)
)
```

Output of Program

```
C:\> python  example_orm.py


publisher2 table is created.

Rows are added to the table.

PUBLISHER2 Table

ID No  Publisher Name          Phone
=====  ======================  ==============
  101  Nobel Yayinevi          212-12345
  102  Pandora Yayinevi        216-5678
```