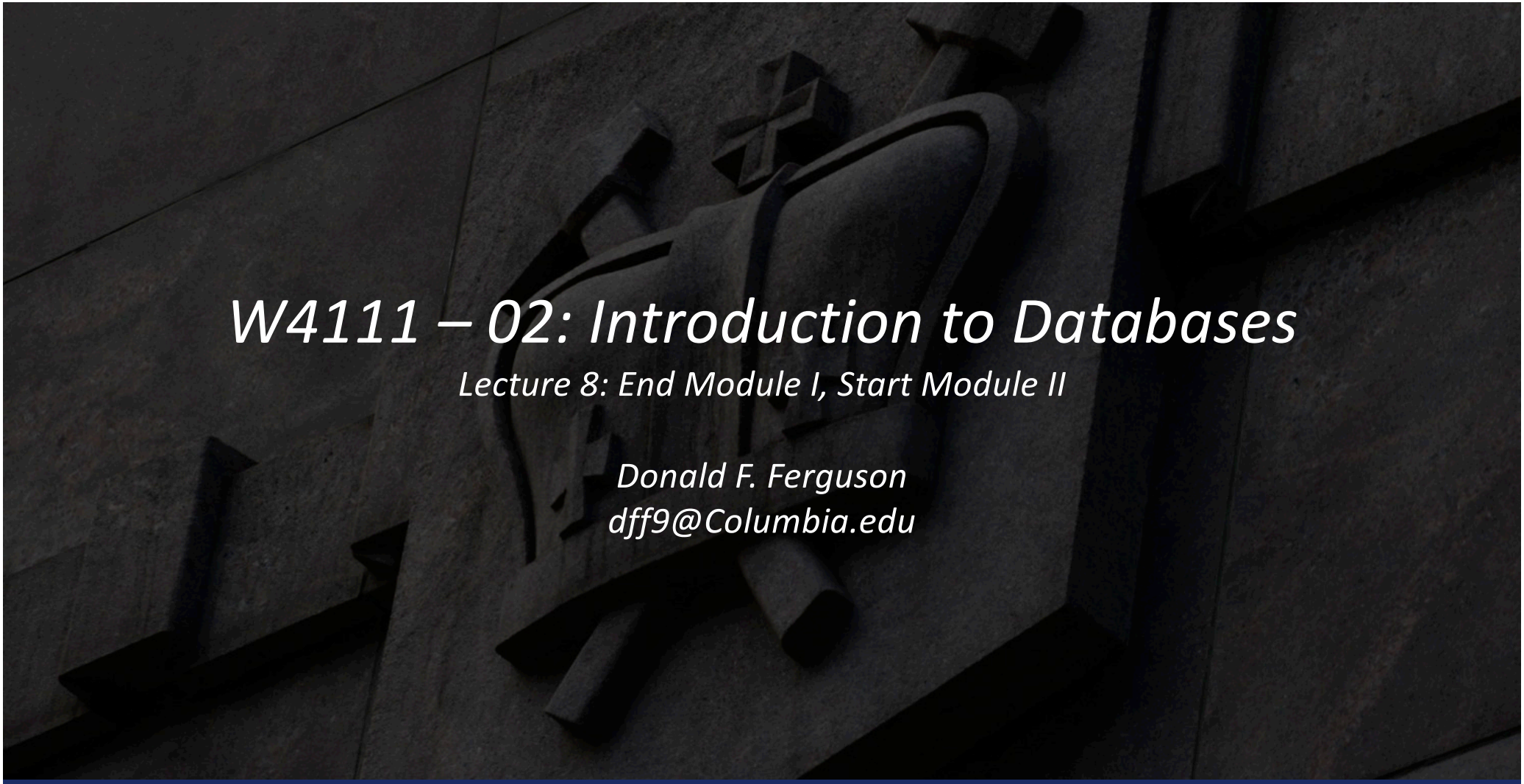


W4111 – 02: Introduction to Databases

Lecture 8: End Module I, Start Module II

Donald F. Ferguson
dff9@Columbia.edu

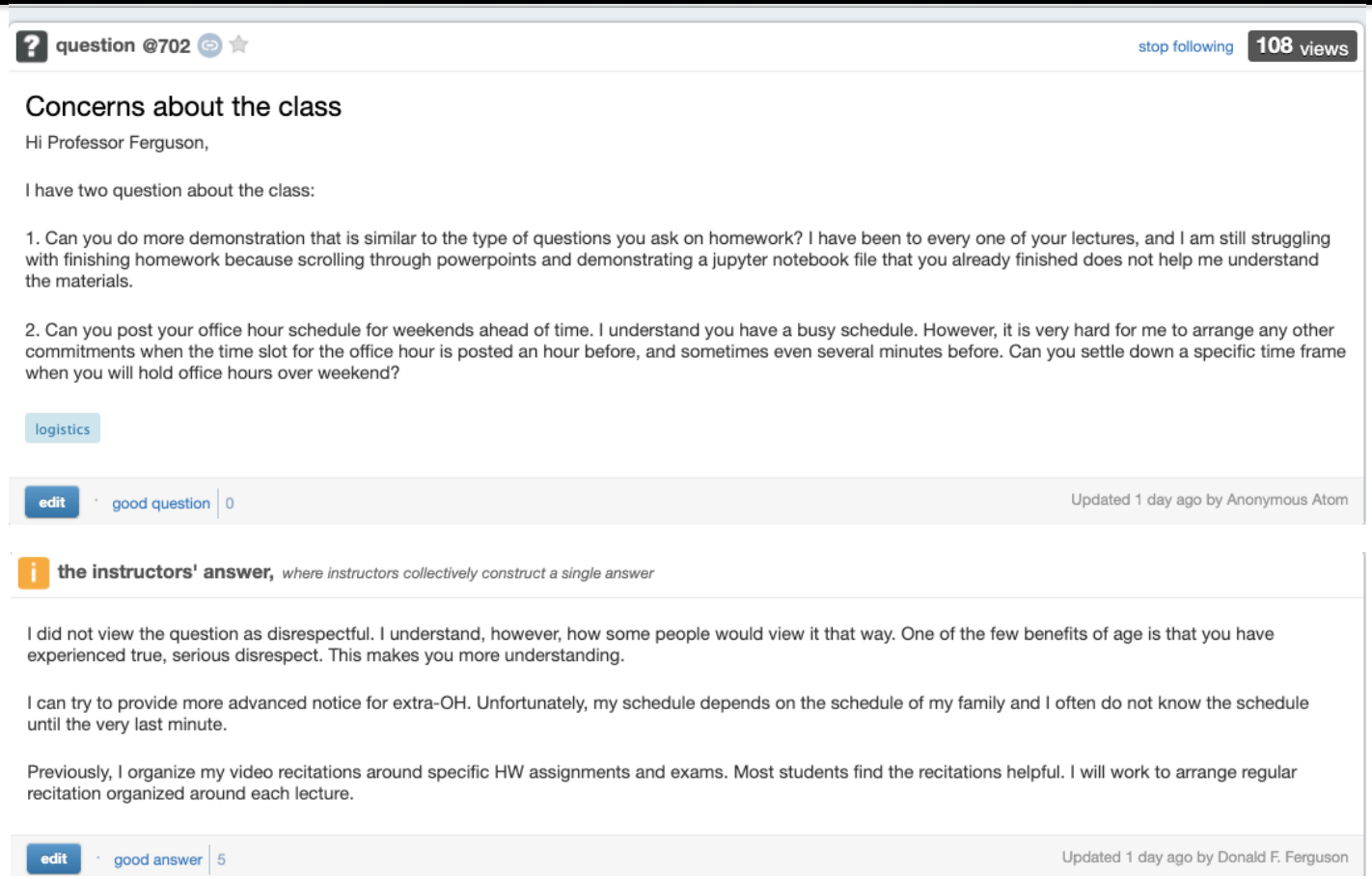
Contents

Content/Agenda

- Note on exams:
  - Will remain take home exams.
  - I will rework homework plan over break.
- Contents/Agenda:
  - Interesting feedback/discussion on Piazza.
  - Course Modules – A reminder because we are transition from I to II.
  - Miscellaneous/wrapping up Module I topics:
    - ER/Data design issues.
    - Accessing SQL from programs.
      - Basic concepts.
      - ORM.
      - Connections, cursors, transactions.
  - Midterm Exam walkthrough.

Interesting Piazza Feedback/Discussion

Interesting Piazza Discussion



question @702 stop following 108 views

Concerns about the class

Hi Professor Ferguson,

I have two question about the class:

1. Can you do more demonstration that is similar to the type of questions you ask on homework? I have been to every one of your lectures, and I am still struggling with finishing homework because scrolling through powerpoints and demonstrating a jupyter notebook file that you already finished does not help me understand the materials.

2. Can you post your office hour schedule for weekends ahead of time. I understand you have a busy schedule. However, it is very hard for me to arrange any other commitments when the time slot for the office hour is posted an hour before, and sometimes even several minutes before. Can you settle down a specific time frame when you will hold office hours over weekend?

logistics

edit · good question | 0 Updated 1 day ago by Anonymous Atom

the instructors' answer, *where instructors collectively construct a single answer*

I did not view the question as disrespectful. I understand, however, how some people would view it that way. One of the few benefits of age is that you have experienced true, serious disrespect. This makes you more understanding.

I can try to provide more advanced notice for extra-OH. Unfortunately, my schedule depends on the schedule of my family and I often do not know the schedule until the very last minute.

Previously, I organize my video recitations around specific HW assignments and exams. Most students find the recitations helpful. I will work to arrange regular recitation organized around each lecture.

edit · good answer | 5 Updated 1 day ago by Donald F. Ferguson

Reminder on Modules

Modules

The course logically has four modules/sections:

1. Foundational concepts: This module covers concepts like data models, relational model, relational databases and applications, schema, ~~normalization~~, ... The section focuses on the relational model and relational databases. The concepts are critical or all types of databases and data centric applications.
2. Database management system architecture and implementation: This module covers the software architecture, algorithms and implementation techniques that allow databases management systems to deliver functions. Topics include memory hierarchy, storage systems, caching/buffer pools, indexes, query processing, query optimization, transaction processing, isolation and concurrency control.
3. NoSQL -- "Not Only SQL" databases: This section provides motivation for "NoSQL" data models and databases, and covers examples and use cases. The section also includes cloud databases and databases-as-a-service.
4. Data Enabled Decision Support: This section covers normalization/denormalization, data warehouses, import and cleanse, OLAP, Pivot Tables, Star Schema, reporting and visualization, and provides and overview of analysis techniques, e.g. clustering, classification, analysis, mining.

- We are transitioning from Module I to Module II.

Miscellaneous ER/Data Design Issues

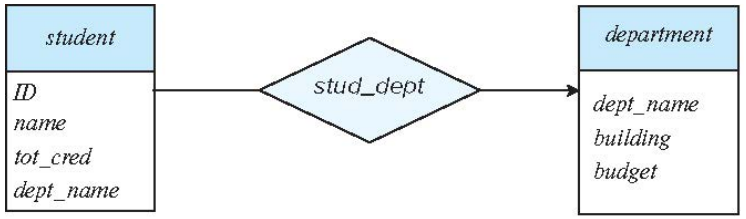
E-R Design Decisions

- The use of an attribute or entity set to represent an object.
- Whether a real-world concept is best expressed by an entity set or a relationship set.
- The use of a ternary relationship versus a pair of binary relationships.
- The use of a strong or weak entity set.
- The use of specialization/generalization – contributes to modularity in the design.
- The use of aggregation – can treat the aggregate entity set as a single unit without concern for the details of its internal structure.

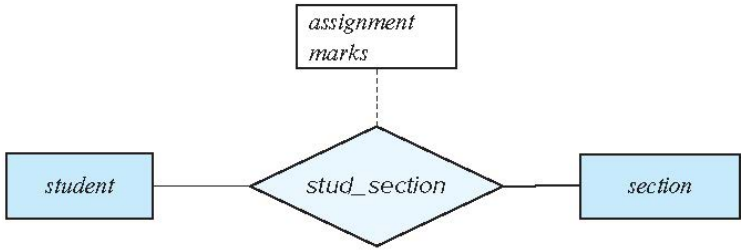
DFF Comments: We will do some walkthroughs in this lecture and recitations.

Common Mistakes in E-R Diagrams

- Example of erroneous E-R diagrams



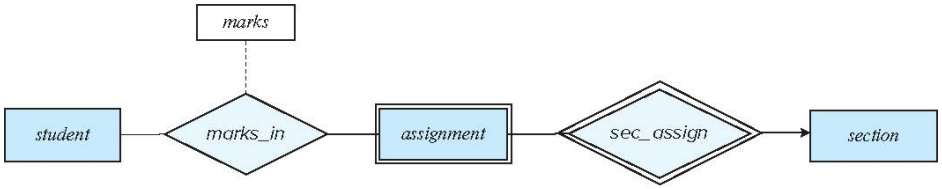
(a) Incorrect use of attribute



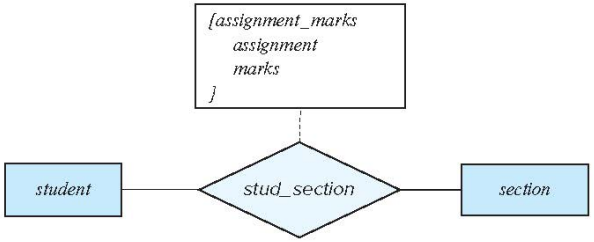
(b) Erroneous use of relationship attributes

Common Mistakes in E-R Diagrams (Cont.)

- Correct versions of the E-R diagram of previous slide



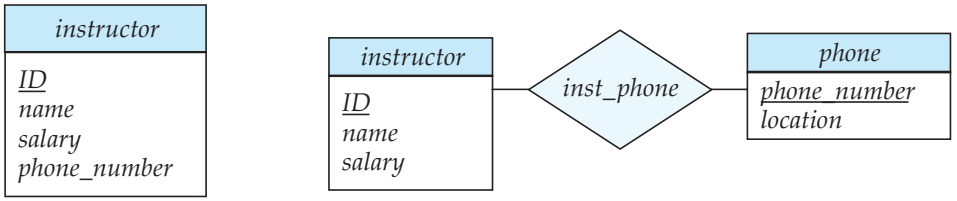
(c) Correct alternative to erroneous E-R diagram (b)



(d) Correct alternative to erroneous E-R diagram (b)

Entities vs. Attributes

- Use of entity sets vs. attributes



- Use of phone as an entity allows extra information about phone numbers (plus multiple phone numbers)

DFF Comments: Let's walk through this one from start to finish.

- Instructor: Where does ID come from? Factoring salary and phone number.
- Phone number structure.
- Approach to contacts.

Switch to various development tools.

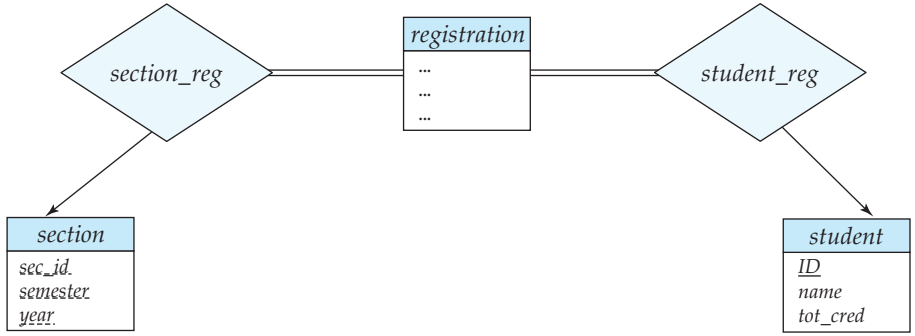
Contact (Phone, Email, …) – Worked Example End-to-End

- Switch to tools.

Entities vs. Relationship sets

- **Use of entity sets vs. relationship sets**

  Possible guideline is to designate a relationship set to describe an action that occurs between entities



- **Placement of relationship attributes**

  For example, attribute date as attribute of advisor (sic?) or as attribute of student

- How might this look in a relational model?
- Why is there a difference between ER diagrams and relational capabilities? (Neo4j)

Switch to various development tools.

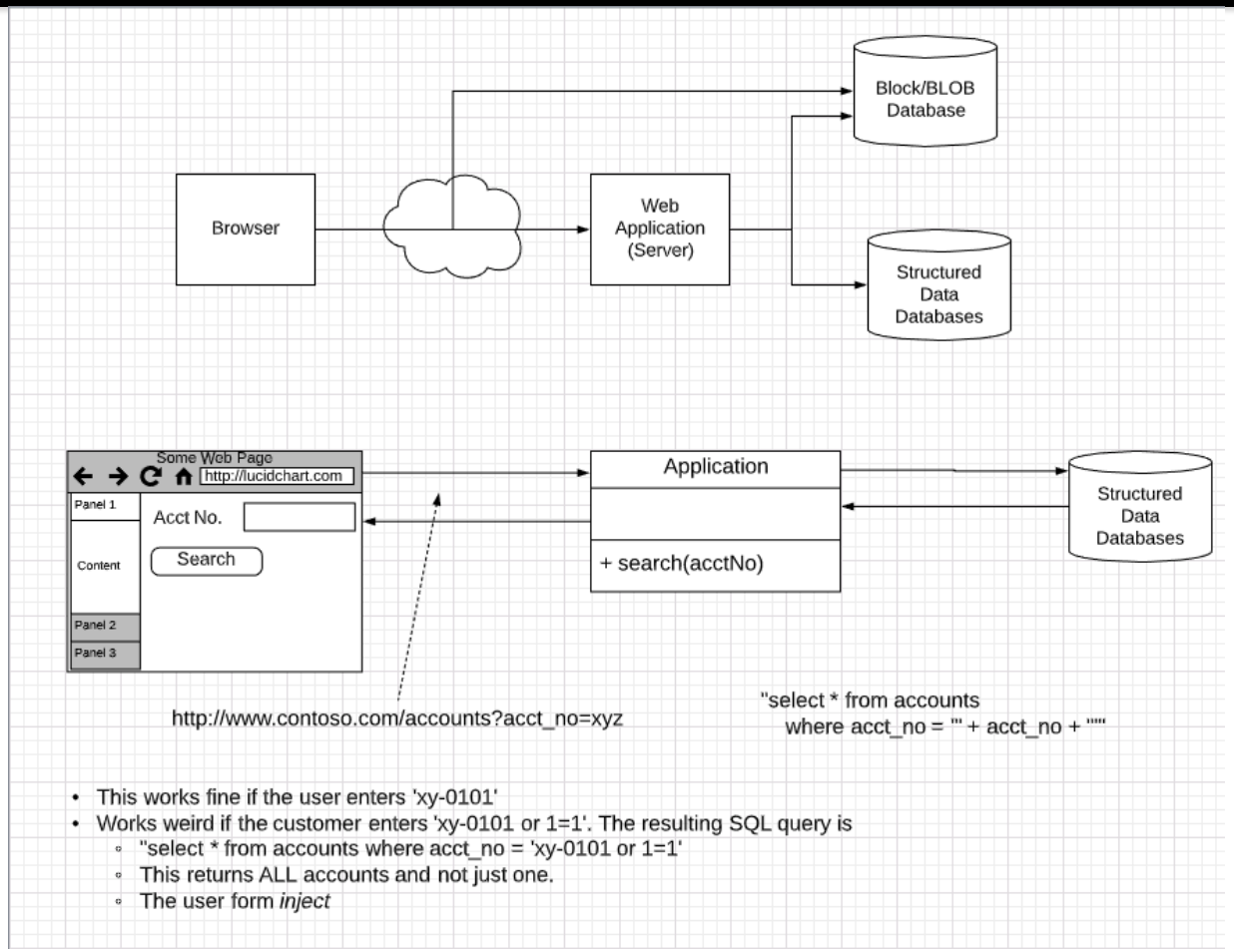
Instructor, Person, Student, Course, Section, Grade, … …

- Switch to tools.

Accessing SQL From Programming Languages

Basic Concepts

SQL Injection Attack



- This works fine if the user enters 'xy-0101'
- Works weird if the customer enters 'xy-0101 or 1=1'. The resulting SQL query is
  - "select * from accounts where acct_no = 'xy-0101 or 1=1'
  - This returns ALL accounts and not just one.
  - The user form *inject*

SQL Injection

- How you use prepared statements varies from language to language, library to library, …
- I will not ask programming/SQL questions but may ask written questions.

- Suppose query is constructed using
  - "select * from instructor where name = '" + name + "'"
- Suppose the user, instead of entering a name, enters:
  - X' or 'Y' = 'Y
- then the resulting statement becomes:
  - "select * from instructor where name = '" + "X' or 'Y' = 'Y" + "'"
  - which is:
    - select * from instructor where name = 'X' or 'Y' = 'Y'
  - User could have even used
    - X'; update instructor set salary = salary + 10000; --
- Prepared statement internally uses:
  "select * from instructor where name = 'X\' or \'Y\' = \'Y'
  - Always use prepared statements, with user inputs as parameters

Prepared Statement

- PreparedStatement pStmt = conn.prepareStatement(
  "insert into instructor values(?,?,?,?)");
  pStmt.setString(1, "88877");
  pStmt.setString(2, "Perry");
  pStmt.setString(3, "Finance");
  pStmt.setInt(4, 125000);
  pStmt.executeUpdate();
  pStmt.setString(1, "88878");
  pStmt.executeUpdate();

  - This is Java code.
  - Do not worry about this in Python for this class, but worry in general if you build apps.

- WARNING: always use prepared statements when taking an input from the user and adding it to a query
  - NEVER create a query by concatenating strings
  - "insert into instructor values(' " + ID + " ', ' " + name + " ', " + " ' + dept name + " ', " ' balance + ')“
  - What if name is “D'Souza”?

Embedded SQL

- The SQL standard defines embeddings of SQL in a variety of programming languages such as C, C++, Java, Fortran, and PL/1,
- A language to which SQL queries are embedded is referred to as a **host language**, and the SQL structures permitted in the host language comprise *embedded* SQL.
- The basic form of these languages follows that of the System R embedding of SQL into PL/1.
- **EXEC SQL** statement is used in the host language to identify embedded SQL request to the preprocessor

  EXEC SQL <embedded SQL statement >;

  Note: this varies by language:
  - In some languages, like COBOL, the semicolon is replaced with END-EXEC
  - In Java embedding uses # SQL { …. };

DFF Comments:

- Do not worry about this. I will not ask query or programming questions.
- You should be aware. I may ask a short written question.

External Language Routines

- SQL allows us to define functions in a programming language such as Java, C#, C or C++.
  - Can be more efficient than functions defined in SQL, and computations that cannot be carried out in SQL\can be executed by these functions.
- Declaring external language procedures and functions

```
create procedure dept_count_proc(in dept_name varchar(20),
                                 out count integer)
language C
external name   '/usr/avi/bin/dept_count_proc'

create function dept_count(dept_name varchar(20))
returns integer
language C
external name '/usr/avi/bin/dept_count'
```

DFF Comments:

- Do not worry about this. I will not ask query or programming questions.
- You should be aware. I may ask a short written question.

Object-Relational Mapping

Object "Relational" Mapping

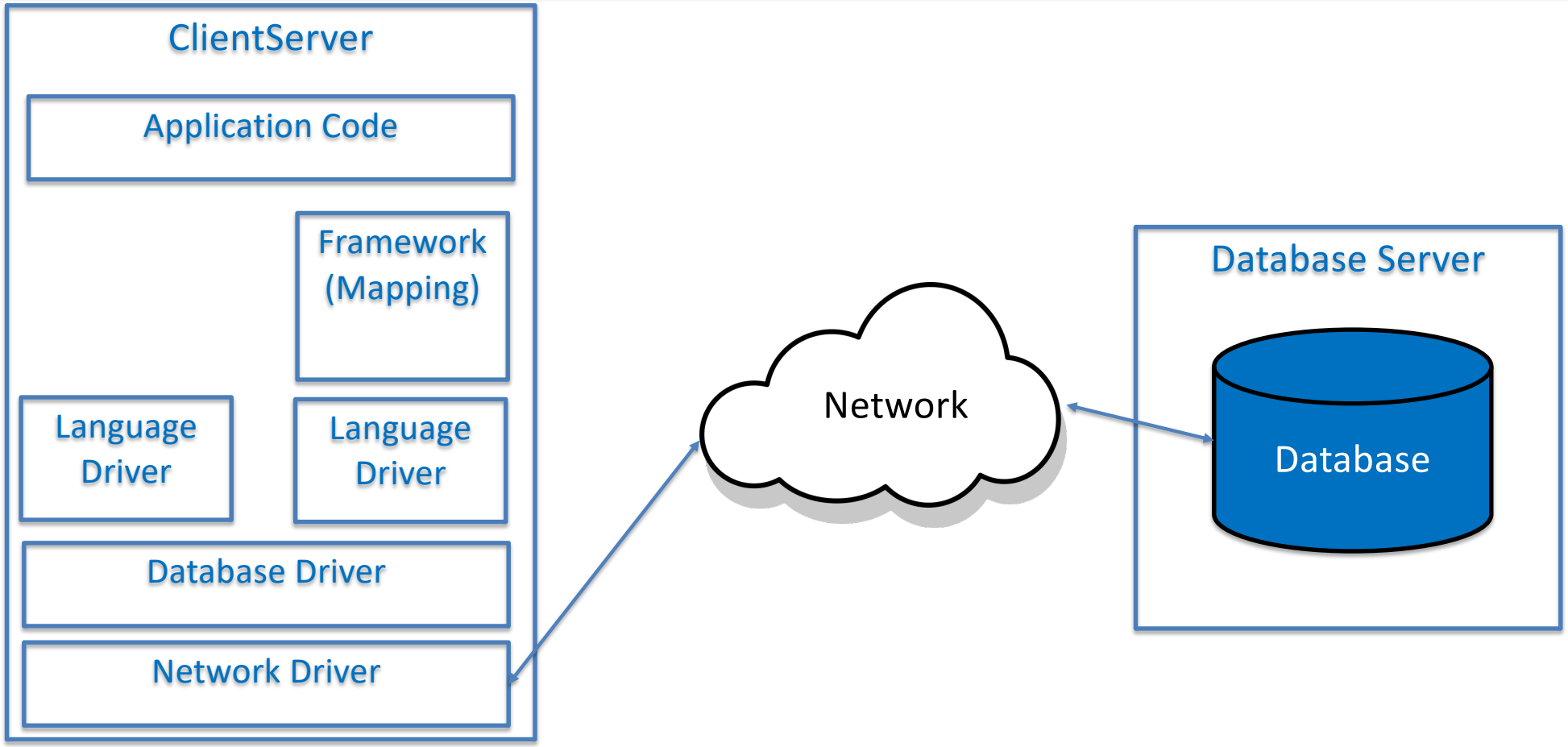
Object-relational mapping (**ORM**, **O/RM**, and **O/R mapping tool**) in computer science is a programming technique for converting data between incompatible type systems using object-oriented programming languages. This creates, in effect, a "virtual object database" that can be used from within the programming language. There are both free and commercial packages available that perform object-relational mapping, although some programmers opt to construct their own ORM tools.

In object-oriented programming, data-management tasks act on objects that are almost always non-scalar values. For example, an address book entry that represents a single person along with zero or more phone numbers and zero or more addresses. This could be modeled in an object-oriented implementation by a "Person object" with attributes/fields to hold each data item that the entry comprises: the person's name, a list of phone numbers, and a list of addresses. The list of phone numbers would itself contain "PhoneNumber objects" and so on. The address-book entry is treated as a single object by the programming language (it can be referenced by a single variable containing a pointer to the object, for instance). Various methods can be associated with the object, such as a method to return the preferred phone number, the home address, and so on.

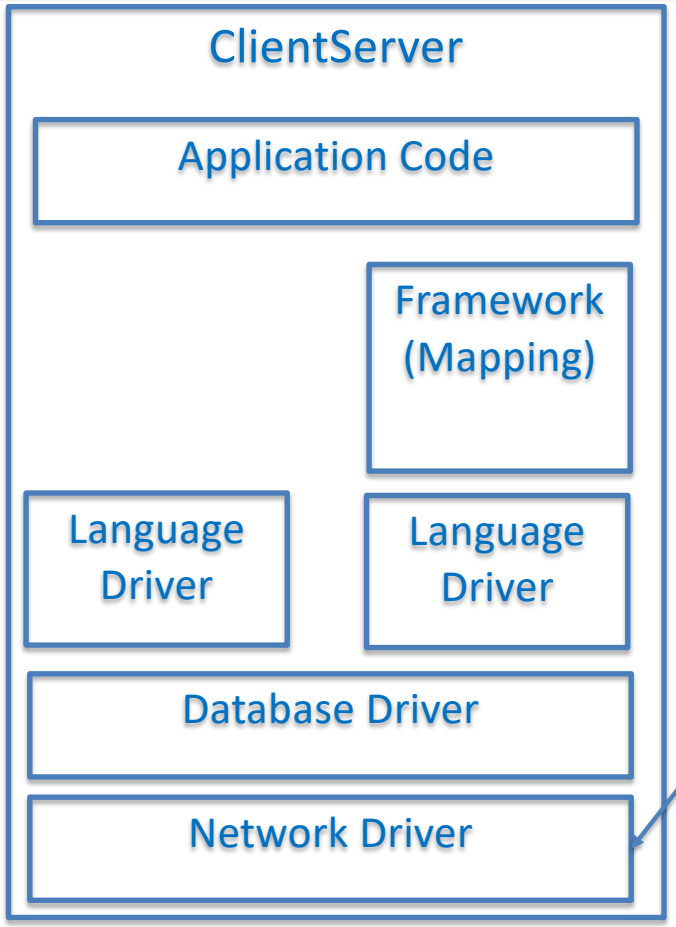
However, many popular database products such as SQL database management systems (DBMS) can only store and manipulate scalar values such as integers and strings organized within tables. The programmer must either convert the object values into groups of simpler values for storage in the database (and convert them back upon retrieval), or only use simple scalar values within the program. Object-relational mapping implements the first approach.[1]

The heart of the problem involves translating the logical representation of the objects into an atomized form that is capable of being stored in the database while preserving the properties of the objects and their relationships so that they can be reloaded as objects when needed. If this storage and retrieval functionality is implemented, the objects are said to be persistent.

Simplistic Overview



Simplistic Overview

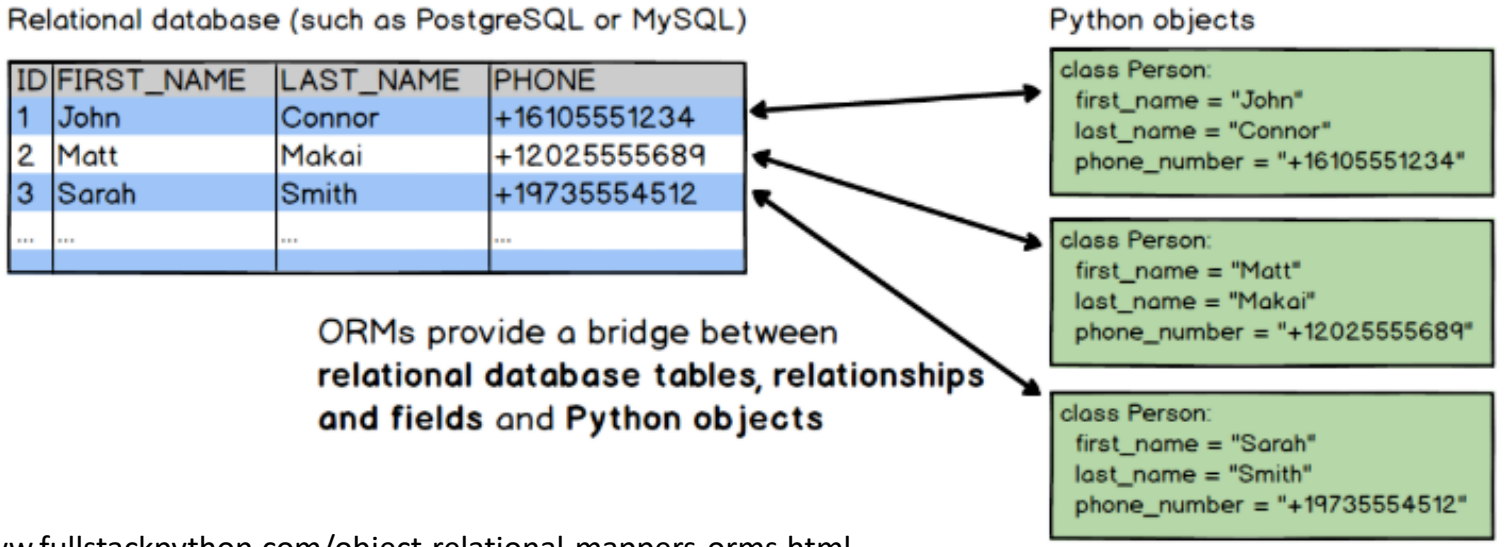


- The application code is what you write to implement your homework (project)
- Network driver is part of the operating system and sends/receives message using HTTP, TCP/IP, etc.
- Database driver uses network driver to send commands and receive response, e.g.
  - SELECT * FROM …
  - Match (n:Node) return …
- The commands and responses are in string/byte format.
- The language driver connects to database driver to language concepts in a library, e.g. functions, object.
- Framework/Mapping "reduces" the mismatch between
  - Language model.
  - Database model.

Object "Relational" Mapping – **Simple** Python Example

Object-relational mappers (ORMs)

An object-relational mapper (ORM) is a code library that automates the transfer of data stored in relational databases tables into objects that are more commonly used in application code.



https://www.fullstackpython.com/object-relational-mappers-orms.html

Object "Relational" Mapping – **Simple** Python Example

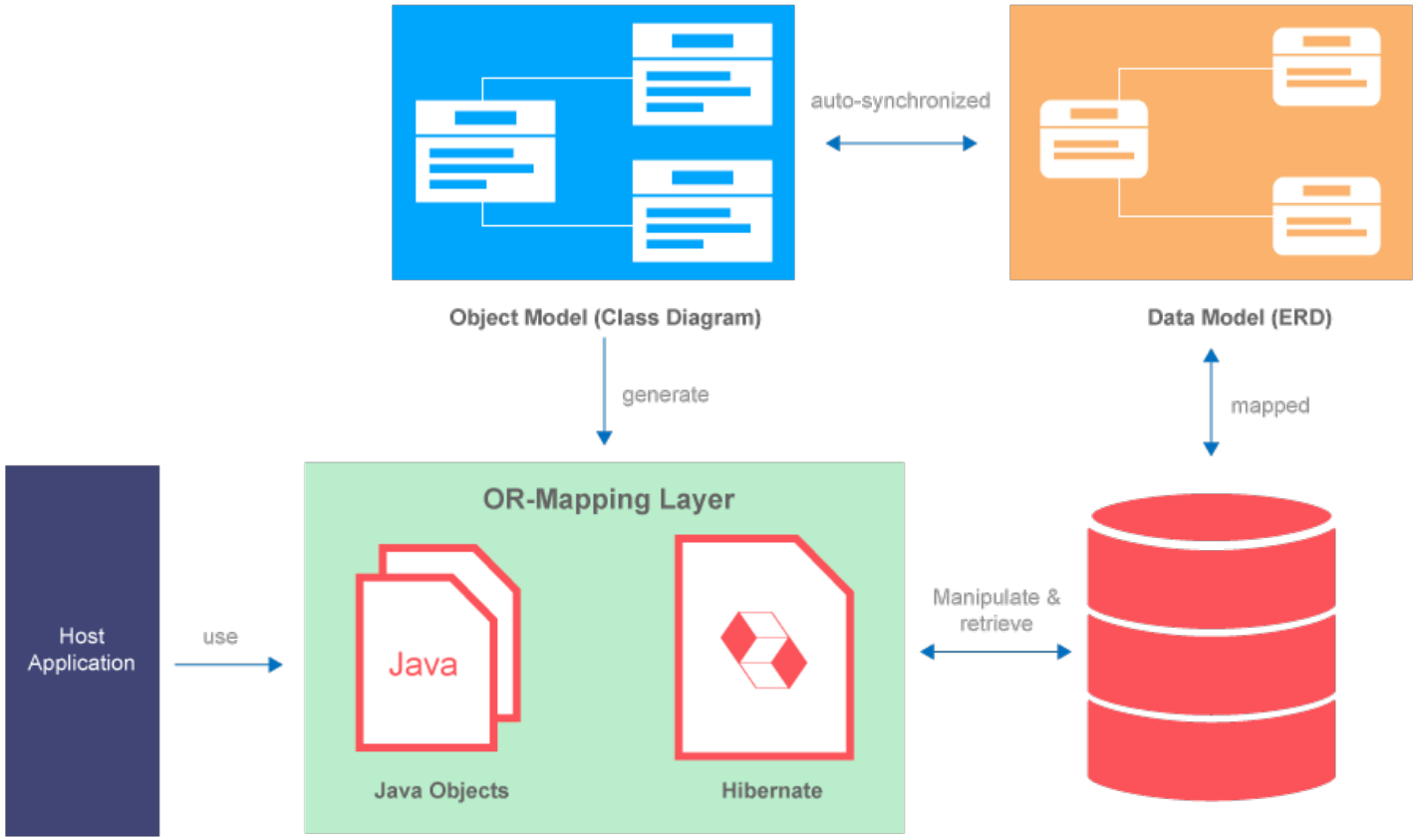
```
from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy import Column, Integer, String
from sqlalchemy.orm import sessionmaker

engine = create_engine('mysql://dbuser:dbuser@localhost:3306/University')
Session = sessionmaker(bind=engine)
session = Session()

Base = declarative_base()


class Student(Base):
    __tablename__ = 'students2'

    uni = Column(String, primary_key=True)
    last_name = Column(String)
    first_name = Column(String)
    year = Column(String)
    school = Column(String)

    def __repr__(self):
        return "<User(uni='%s', last_name='%s', first_name='%s', year='%s', school='%s')>" % (
            self.uni, self.last_name, self.first_name, self.year, self.school)
```

Object "Relational" Mapping – **Simple** Python Example

```python
def test1():
    q = session.query(Student)
    result = q.all()
    print("All students:")
    for s in result:
        print(s)


def test2():
    q = session.query(Student)
    result =\
        q.filter_by(last_name='Ferguson').all()
    print("Some students:")
    for s in result:
        print(s)


def test3():
    q = session.query(Student)
    result = q.all()
    print("Some student names:")
    for s in result:
        print(s.first_name, s.last_name)
```

```
All students:
<User(uni='FED00', last_name='Ferguson', first_name='Donald', year='2018', school='C')>
<User(uni='FED02', last_name='Ferguson', first_name='Donald', year='2019', school='C')>
<User(uni='GAWI1', last_name='Gandalf', first_name='Wizard', year='2018', school='E')>
<User(uni='GRHE1', last_name='Grainger', first_name='Hermione', year='2020', school='C')>
<User(uni='HAP01', last_name='Potter', first_name='Harry', year='2021', school='C')>
<User(uni='SMJO1', last_name='Smith', first_name='John', year='0', school='C')>


Some students:
<User(uni='FED00', last_name='Ferguson', first_name='Donald', year='2018', school='C')>
<User(uni='FED02', last_name='Ferguson', first_name='Donald', year='2019', school='C')>


Some student names:
Donald Ferguson
Donald Ferguson
Wizard Gandalf
Hermione Grainger
Harry Potter
John Smith
```

Java Hibernate



Some Questions

- Why did I take this diversion?
  - I have alluded to various frameworks, but never covered.
    You should be aware of the option of using the frameworks.
  - py2neo is an example of one of these frameworks.
- There are pros and cons to using the frameworks:
  - Pros: significantly improved productivity for simple applications and mappings.
  - Cons: Anything complex can be virtually impossible if you use a framework.
  - The Wikipedia article is a good start on pros and cons.
- Why didn't I tell you about/let you use these frameworks?
  - You would not have learned the fundamental concepts in databases and models.
  - Frameworks are not helpful for many application scenarios.
  - This is not what the cool kids do.

Connections, Sessions, Cursors, Transactions

Database Connection

A **Database connection** is a facility in computer science that allows client software to talk to database server software, whether on the same machine or not. A **connection** is required to send commands and receive answers, usually in the form of a result set.

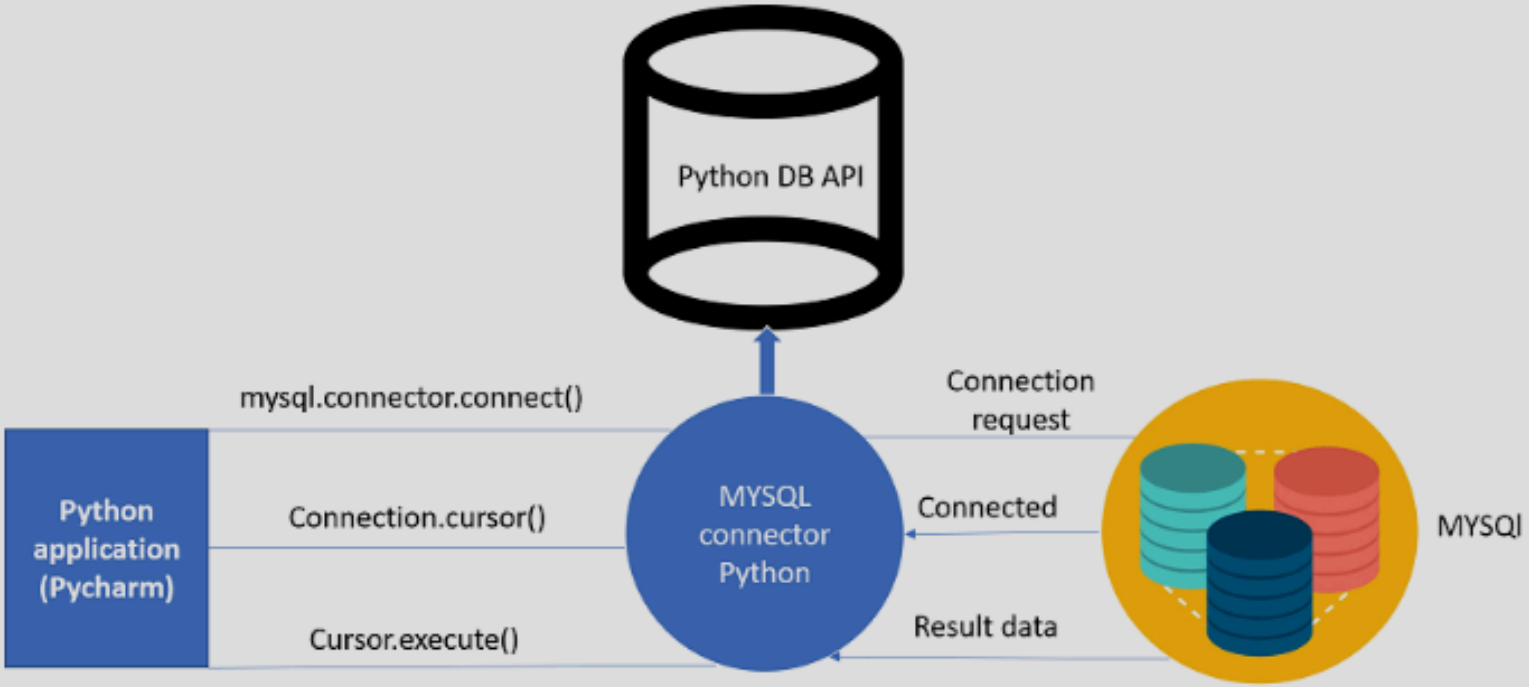
Connections are a key concept in data-centric programming. Since some DBMS engines require considerable time to connect, connection pooling was invented to improve performance. No command can be performed against a database without an "open and available" connection to it.

Connections are built by supplying an underlying driver or provider with a connection string, which is a way of addressing a specific database or server and instance as well as user authentication credentials (for example, ***Server=****sql_box;****Database=****Common;****User ID=****uid;****Pwd=****password;*). Once a connection has been built it can be opened and closed at will, and properties (such as the command time-out length, or transaction, if one exists) can be set. The Connection String is composed of a set of key/value pairs as dictated by the data access interface and data provider being used.

This is what those weird statements are doing, e.g. *pymysql.connect*

- *pymysql* is the driver library.
- *connect* returns the connection object used to send SQL commands and receive responses.
- Something very similar is happening when you use “connection” is MySQL Workbench or PyCharm.

Connection, Cursor, … …



Before connecting to the MySQL database, make sure you have MySQL installer installed on your computer. It provides a comprehensive set of tools which helps in installing MySQL with the following components:

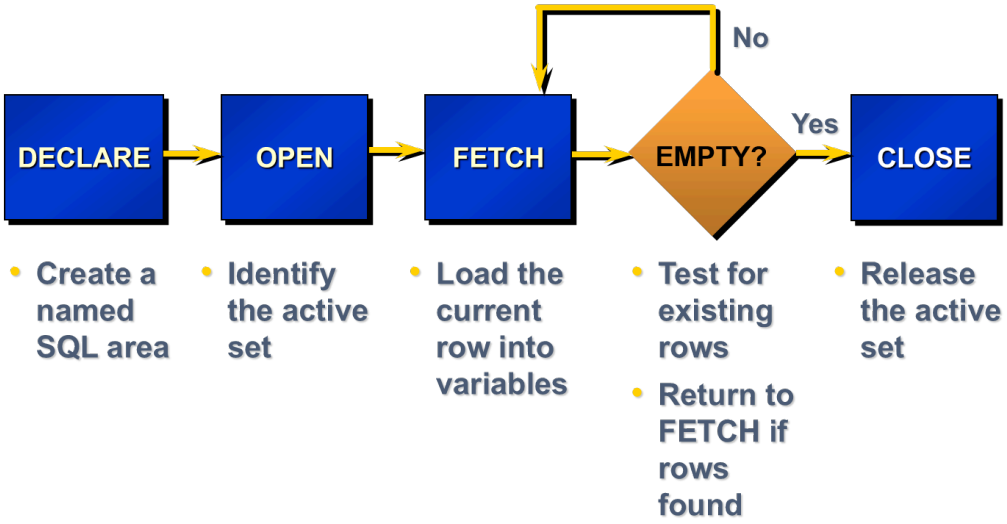
Database Connection

In computer science, a database **cursor** is a control structure that enables traversal over the records in a database. Cursors facilitate subsequent processing in conjunction with the traversal, such as retrieval, addition and removal of database records. The database cursor characteristic of traversal makes cursors akin to the programming language concept of iterator.

Cursors are used by database programmers to process individual rows returned by database system queries. Cursors enable manipulation of whole result sets at once. In this scenario, a cursor enables the sequential processing of rows in a result set.

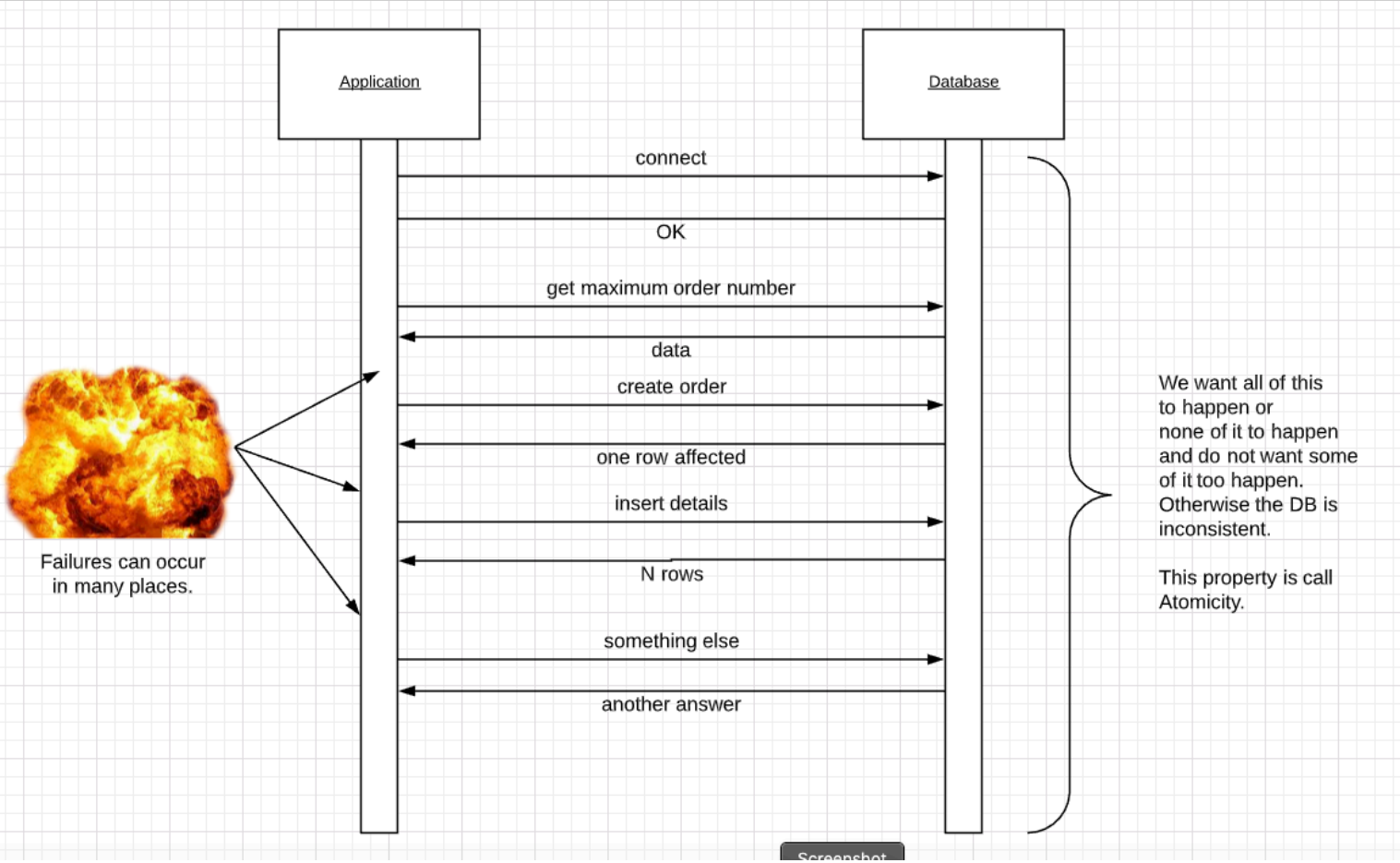
In SQL procedures, a cursor makes it possible to define a result set (a set of data rows) and perform complex logic on a row by row basis. By using the same mechanics, a SQL procedure can also define a result set and return it directly to the caller of the SQL procedure or to a client application.

A cursor can be viewed as a pointer to one row in a set of rows. The cursor can only reference one row at a time, but can move to other rows of the result set as needed.



- SQL/RDBMS have commands for cursors:
  - DECLARE
  - OPEN
  - FETCH
  - CLOSE
- The driver/libraries encapsulate/use these basic primitives.
- These are similar to things to concepts in programming languages, e.g. Java Iterators

Transactions



Switch to Examples

- Connection
  - MySQL Workbench
  - pymysql in Jupyter Notebook
- pymysql Cursors
  - Execute
  - Various forms of fetch
- MySQL/SQL cursors only work inside stored procedures.
- Transactions (topic in Module II)
  - Commit
  - Rollback
  - Autocommit

Show in Python

- Connect
- Autocomit
- Forms of fetch
- Abort
- Rollback

- Show interactions with Workbench

Midterm Walkthrough