UMBC

AN HONORS UNIVERSITY IN MARYLAND

CMSC 461, Database Management Systems
Spring 2018

Lecture 8 - Chapter 7 Entity Relationship Model

These slides are based on "Database System Concepts" 6th edition book and are a modified version of the slides which accompany the book (http://codex.cs.yale.edu/avi/db-book/db6/slide-dir/index.html)

Dr. Jennifer Sleeman

https://www.csee.umbc.edu/~jsleem1/courses/461/spr18

Logistics

- Homework 2 due Monday 2/26/2018
- Project Phase 2 posted due 3/5/2018
  - You need the upcoming lectures to do phase 2
  - You need phase 1 grades to do phase 2
    - Grader plans to finish grading by the weekend

Lecture Outline

- Closing the loop on Views
- Connecting to MySQL using Python
- E-R Modeling
- Entity Sets & Relationship Sets
- Attributes
- Cardinality
- Keys

Lecture Outline

- *Closing the loop on Views*
- Connecting to MySQL using Python
- E-R Modeling
- Entity Sets & Relationship Sets
- Attributes
- Cardinality
- Keys

Views - Closing the Loop

mysql> **create view faculty as select name, dept_name from instructor;**
Query OK, 0 rows affected (0.02 sec)

mysql> select * from faculty;
+------------+------------+
| name | dept_name |
+------------+------------+
Srinivasan	Comp. Sci.
Wu	Finance
Mozart	Music
Einstein	Physics
El Said	History
Gold	Physics
Katz	Comp. Sci.
Califieri	History
Singh	Finance
Crick	Biology
Brandt	Comp. Sci.
Kim	Elec. Eng.
+------------+------------+
12 rows in set (0.00 sec)

mysql> select * from instructor;
+-------+------------+------------+----------+
| ID | name | dept_name | salary |
+-------+------------+------------+----------+
10101	Srinivasan	Comp. Sci.	65000.00
12121	Wu	Finance	90000.00
15151	Mozart	Music	40000.00
22222	Einstein	Physics	95000.00
32343	El Said	History	60000.00
33456	Gold	Physics	87000.00
45565	Katz	Comp. Sci.	75000.00
58583	Califieri	History	62000.00
76543	Singh	Finance	80000.00
76766	Crick	Biology	72000.00
83821	Brandt	Comp. Sci.	92000.00
98345	Kim	Elec. Eng.	80000.00
+-------+------------+------------+----------+
12 rows in set (0.00 sec)

mysql> describe instructor;
+-----------+-------------+------+-----+---------+-------+
| Field | Type | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
ID	varchar(5)	NO	PRI	NULL	
name	varchar(20)	NO		NULL	
dept_name	varchar(20)	YES	MUL	NULL	
salary	decimal(8,2)	YES		NULL	
+-----------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)

Views - Closing the Loop

Knowing there are constraints on the instructor table as it relates to a foreign key on the department table, let's try to insert using our view.

It should give us an error…..

```
mysql> insert into faculty values ('ProfessorFoo', 'DeptFoo');
ERROR 1423 (HY000): Field of view 'lecture_8.faculty' underlying table
doesn't have a default value

mysql> insert into faculty values ('ProfessorFoo', 'History');
ERROR 1423 (HY000): Field of view 'lecture_8.faculty' underlying table
doesn't have a default value
```

Does it reveal anything about the attributes not specified by the view?

Views - Closing the Loop

Knowing there are constraints on the instructor table as it relates to a foreign key on the department table, let's try to insert using our view.

It should give us an error…..

```
mysql> insert into faculty values ('ProfessorFoo', 'DeptFoo');
ERROR 1423 (HY000): Field of view 'lecture_8.faculty' underlying table
doesn't have a default value

mysql> insert into faculty values ('ProfessorFoo', 'History');
ERROR 1423 (HY000): Field of view 'lecture_8.faculty' underlying table
doesn't have a default value
```

Does it reveal anything about the attributes not specified by the view?

Views - Closing the Loop

If we delete instructor what happens to the view?

```
mysql> delete from instructor;
Query OK, 12 rows affected (0.01 sec)

mysql> select * from faculty;
Empty set (0.00 sec)
```

The view is left unattached, but it still exists….

```
mysql> insert into faculty values ('ProfessorFoo', 'History');
ERROR 1423 (HY000): Field of view 'lecture_8.faculty' underlying table
doesn't have a default value
mysql>
```

However if we try to insert into it, we receive an error

Lecture Outline

- Closing the loop on Views
- ***Connecting to MySQL using Python***
- E-R Modeling
- Entity Sets & Relationship Sets
- Attributes
- Cardinality
- Keys

Connecting to MySQL from Python

- Multiples ways to connect
- In general, need to use a driver in order for python to be able to access MySQL
  - Evaluation:
    - https://wiki.openstack.org/wiki/PyMySQL_evaluation

MySQL DB Drivers Comparison

Project	PyPi hosted	Eventlet friendly	Python 3 compatibility	Maturity and/or stability	Comment
MySQL-Python	Yes	Partial	No	Yes	Can be monkeypatched by eventlet, but only to enable thread pooling
mysqlclient	Yes	Partial	Yes	Yes	Initial testing shows that this is a promising DBAPI if eventlet requirement can be dropped
OurSQL	Yes	No	Yes, but not Pypi hosted	No	Development halted fairly early on, and has not seen commits/releases in two years
MySQL-Connector-Python	No	Yes	Yes	Yes, though the driver is still fairly new	The official Oracle-supported driver for MySQL
PyMySQL	Yes	Yes	Yes	Yes, however see notes below.	Actively maintained and popular.

Image Credit: https://wiki.openstack.org/wiki/PyMySQL_evaluation

Connecting to MySQL from Python - PyMySQL

- Pure python solution

Run the following command to install:

```
pip install PyMySQL
```


Documentation: https://github.com/PyMySQL/PyMySQL

PyMySQL Code

```
#This is a test of pymysql
import pymysql.cursors

# Connect to the database
connection = pymysql.connect(host='localhost',
                             user='root',
                             password='root',
                             db='lecture5')
try:

    with connection.cursor() as cursor:
        #create the table
        sql="CREATE table users (email varchar(500), password varchar(100))";
        cursor.execute(sql);
        # Create a new record
        sql = "INSERT INTO users (email, password) VALUES (%s, %s)"
        cursor.execute(sql, ('ilovesql@gmail.com', 'mypassword'))

    # connection is not autocommit by default. So you must commit to save
    # your changes.
    connection.commit()
    with connection.cursor() as cursor:
        # Read a single record
        sql = "SELECT password FROM users WHERE email=%s"
        cursor.execute(sql, ('ilovesql@gmail.com'))
        result = cursor.fetchone()
        print("Getting password: " + str(result[0]))
    with connection.cursor() as cursor:
        # Update record
        sql = "Update users set password = %s where email =%s"
        cursor.execute(sql, ('mybadpassword','ilovesql@gmail.com'))

    with connection.cursor() as cursor:
        # Read a single record
        sql = "SELECT password FROM users WHERE email=%s"
        cursor.execute(sql, ('ilovesql@gmail.com'))
        result = cursor.fetchone()
        print("New password is: " + str(result[0]))
    with connection.cursor() as cursor:
        #delete record
        sql = "delete from users where email =%s"
        cursor.execute(sql, ('ilovesql@gmail.com'))

    with connection.cursor() as cursor:
        # Read a single record
        sql = "SELECT password FROM users WHERE email=%s"
        cursor.execute(sql, ('ilovesql@gmail.com'))
        result = cursor.fetchone()
        print("This should be empty: " + str(result))
    with connection.cursor() as cursor:
        #drop the table so we can run this without error
        sql="drop table `users`";
        cursor.execute(sql);
    connection.commit()
finally:
    connection.close()
```


Documentation: https://github.com/PyMySQL/PyMySQL

PyMySQL - Step by Step

Step 1: Connect to the database

```
connection = pymysql.connect(host='localhost',
                             user='root',
                             password='root',
                             db='lecture5')
```


Documentation: https://github.com/PyMySQL/PyMySQL

PyMySQL - Step by Step

Step 2: Get a cursor

```
with connection.cursor() as cursor:
```


Documentation: https://github.com/PyMySQL/PyMySQL

PyMySQL - Step by Step

Step 3: Perform SQL operations - Create

```
    with connection.cursor() as cursor:
        #create the table
        sql="CREATE table users (email
varchar(500), password varchar(100))";
        cursor.execute(sql);
```


Documentation: https://github.com/PyMySQL/PyMySQL

PyMySQL - Step by Step

Step 3: Perform SQL operations - Insert

```
    with connection.cursor() as cursor:
        # Create a new record
        sql = "INSERT INTO users (email,
    password) VALUES (%s, %s)"
        cursor.execute(sql,
    ('ilovesql@gmail.com', 'mypassword'))
```


Documentation: https://github.com/PyMySQL/PyMySQL

PyMySQL - Step by Step

Step 3: Perform SQL operations - Select

```
  with connection.cursor() as cursor:
        # Read a single record
        sql = "SELECT password FROM users WHERE
email=%s"
        cursor.execute(sql,
('ilovesql@gmail.com'))
        result = cursor.fetchone()
        print(result[0])
```


Documentation: https://github.com/PyMySQL/PyMySQL

PyMySQL - Step by Step

Step 3: Perform SQL operations - Update

```
  with connection.cursor() as cursor:
        # Update new record
        sql = "Update users set password = %s
      where email =%s"
        cursor.execute(sql,
      ('mybadpassword','ilovesql@gmail.com'))
```


Documentation: https://github.com/PyMySQL/PyMySQL

PyMySQL - Step by Step

Step 3: Perform SQL operations - Delete

```
with connection.cursor() as cursor:
    # Delete record
    sql = "delete from users where email =%s"
    cursor.execute(sql,'ilovesql@gmail.com')
```


Documentation: https://github.com/PyMySQL/PyMySQL

PyMySQL - Step by Step

Step 3: Perform SQL operations - Drop

```
   with connection.cursor() as cursor:
          #drop the table so we can run this
without error
          sql="drop table users";
          cursor.execute(sql);
      connection.commit()
```


Documentation: https://github.com/PyMySQL/PyMySQL

PyMySQL - Step by Step

Step 4: Close the connection

```
try:
…

finally:
    connection.close()
```


Documentation: https://github.com/PyMySQL/PyMySQL

Lecture Outline

- Closing the loop on Views
- Connecting to MySQL using Python
- ***E-R Modeling***
- Entity Sets & Relationship Sets
- Attributes
- Cardinality
- Keys

Design Process

- For ‘simple applications’ may be simple process to define relations, attributes and constraints
- Real world applications are complex
  - They are hard to model
  - Not as easy to define relations, attributes and constraints

Design Phases

- Initial Phase
  - Characterize fully data needs of users
  - Outcome is a specification of user requirements
- Concept Design Phase
  - Choose a data model
  - Translate requirements into conceptual schema (***E-R model***)

Design Phases

- Logical Design Phase
  - Move from abstract data model to implementation
  - Map high level concepts to implementation data model
  - Typically relational model
- Physical Design Phase
  - Physical features such as index structures and file organization

Design Alternatives

- How to represent types of “things” (***entities***)
- How to relate “things” (***relations***)
- Avoid 2 major pitfalls:
  - Redundancy
    - repeated information
    - in the schema
  - Incompleteness
    - Will make certain aspects of enterprise hard if not impossible to model
- Lots of options for design

E-R Modeling

- Useful for mapping meanings and interactions of real-world enterprises to conceptual schema
- 3 basic concepts
  - Entity sets
  - Relationship sets
  - Attributes

Lecture Outline

- Closing the loop on Views
- Connecting to MySQL using Python
- E-R Modeling
- ***Entity Sets & Relationship Sets***
- Attributes
- Cardinality
- Keys

Entities

- An ***entity*** is an object that exists and is distinguishable from other objects.
  - Example: specific person, company, event, plant
  - An entity has a set of **properties**
    - **Values** for some set of properties may uniquely identify an entity
      - Example: a person may have a person_id
  - An entity may be concrete or abstract

Entity Set

- An ***entity set*** is a set of entities of the same type that share the same properties.

  Example:

  “set of all people who are instructors at a given university”

Attributes

- Entities are represented by a set of ***attributes***
  - Example: people have names and addresses
- Attributes are descriptive properties
- For each attribute each entity has a ***value***

Entity Sets

instructor_ID instructor_name

76766	Crick
45565	Katz
10101	Srinivasan
98345	Kim
76543	Singh
22222	Einstein

instructor

student-ID student_name

98988	Tanaka
12345	Shankar
00128	Zhang
76543	Brown
76653	Aoi
23121	Chavez
44553	Peltier

student

Image from "Database System Concepts" book and slides, 6th edition

Relationship

- A **relationship** is an association among several entities

Example:

Dr. Johnson advises Jordan

This defines an *advisor* relationship

Relationship Sets

A **relationship set** is a mathematical relation among $n \geq 2$ entities, each taken from entity sets

$\{(e_1, e_2, \ldots e_n) \mid e_1 \in E_1, e_2 \in E_2, \ldots, e_n \in E_n\}$

where $(e_1, e_2, \ldots, e_n)$ is a relationship

Given the student and section entity sets, a relationship can be defined by takes

Relationship Sets - Advisors

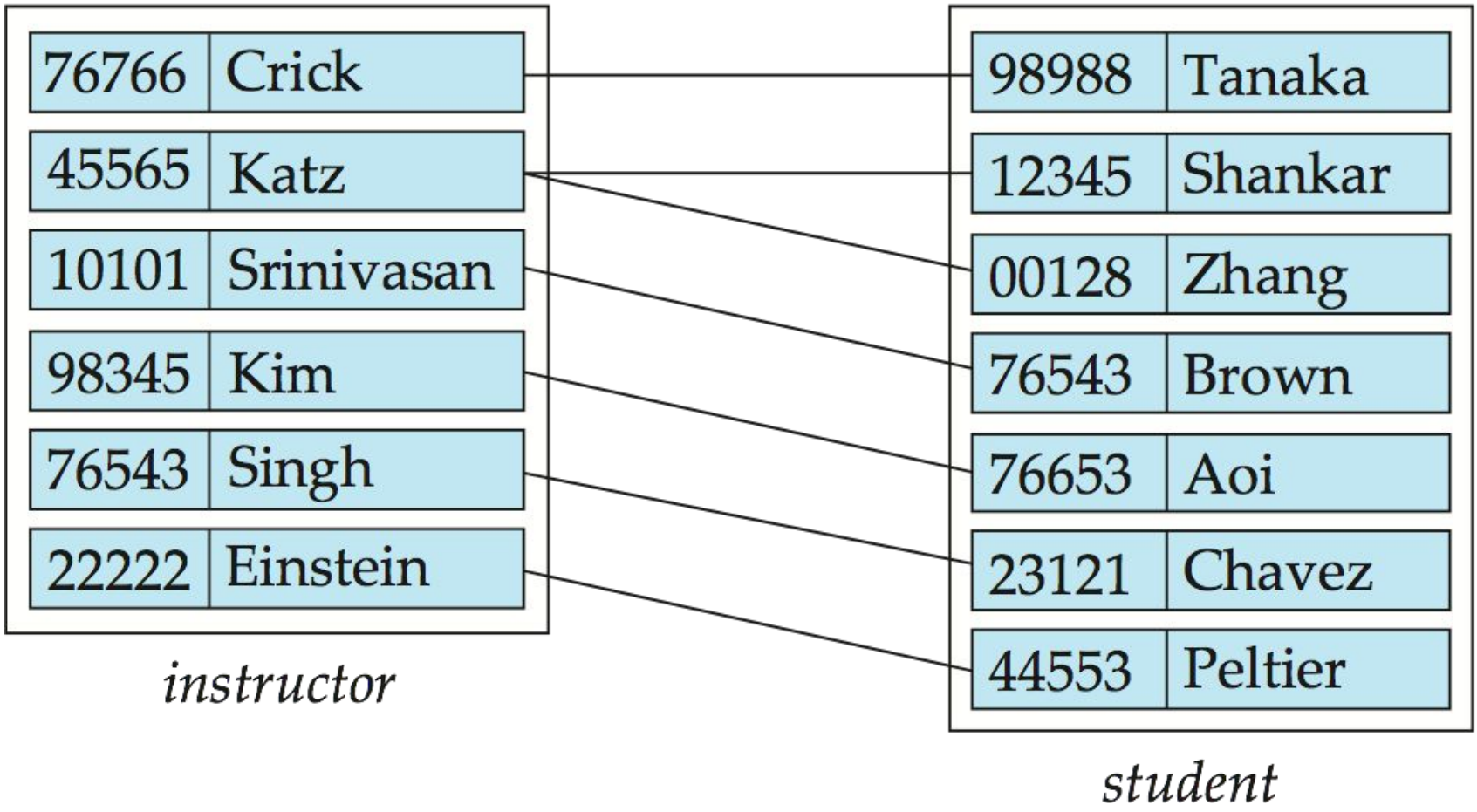
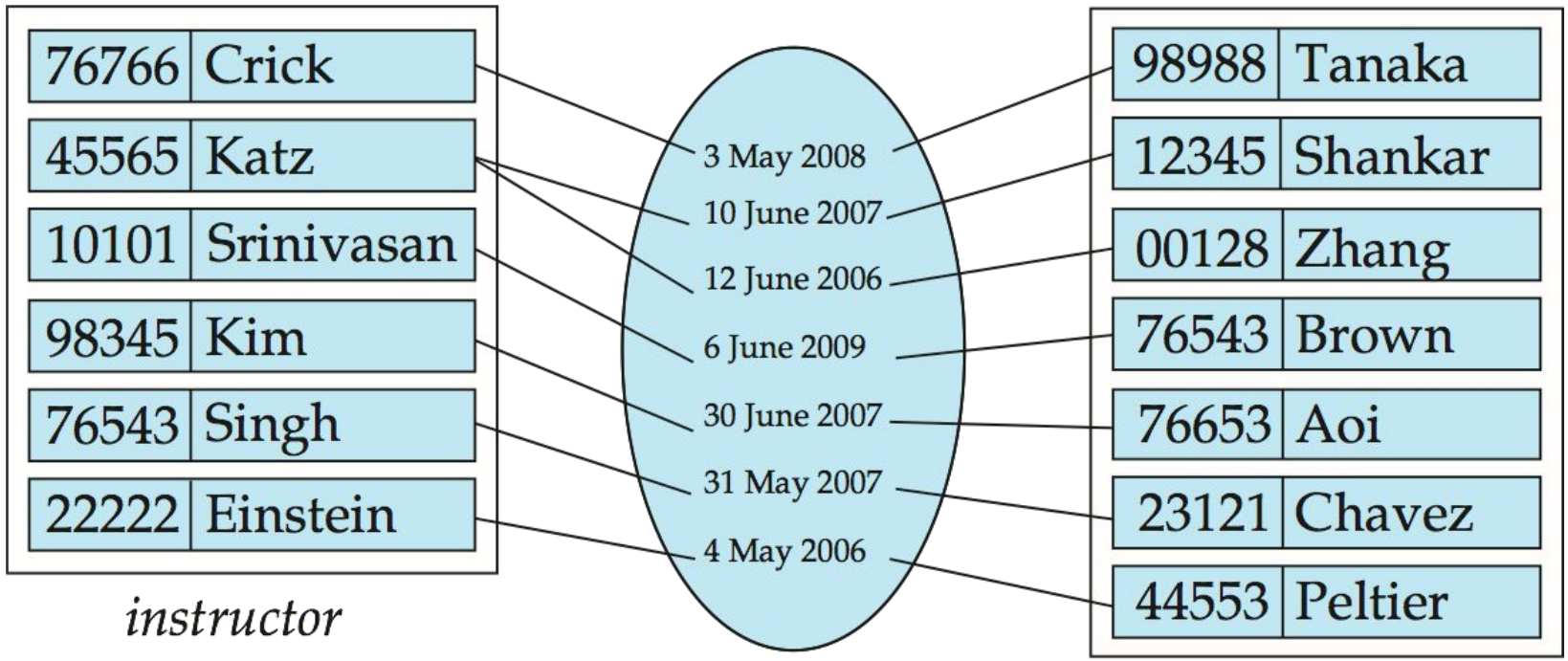


Image from "Database System Concepts" book and slides, 6th edition

Relationship Sets

- An attribute can also be property of a relationship set (***descriptive attribute***)
- For instance, the advisor relationship set between entity sets instructor and student may have the attribute date which tracks when the student started being associated with the advisor



Based on and image from "Database System Concepts" book and slides, 6th edition

Design of a Relationship

- ***Binary relationship*** involves two entity sets (or degree two)
  - most relationship sets in a database system are binary
- Relationships between more than two entity sets are rare
  - example: students work on research projects under the guidance of an instructor
  - relationship proj_guide is a ternary relationship between instructor, student, and project

Lecture Outline

- Closing the loop on Views
- Connecting to MySQL using Python
- E-R Modeling
- Entity Sets & Relationship Sets
- ***Attributes***
- Cardinality
- Keys

Attributes

- An entity is represented by a set of attributes, that is descriptive properties possessed by all members of an entity set.

Example:

instructor = (ID, name, street, city, salary)

course= (course_id, title, credits)

Attributes

- ***Domain*** – the set of permitted values for each attribute
- Attribute types:
  - ***Simple*** and ***composite*** attributes
    - Example: Name (first, middle, last)
  - ***Single-valued*** and ***multivalued*** attributes
    - Example: multivalued attribute: phone_numbers
- ***Derived*** attributes
  - Can be computed from other attributes
  - Example: age, given date_of_birth

Composite Attributes

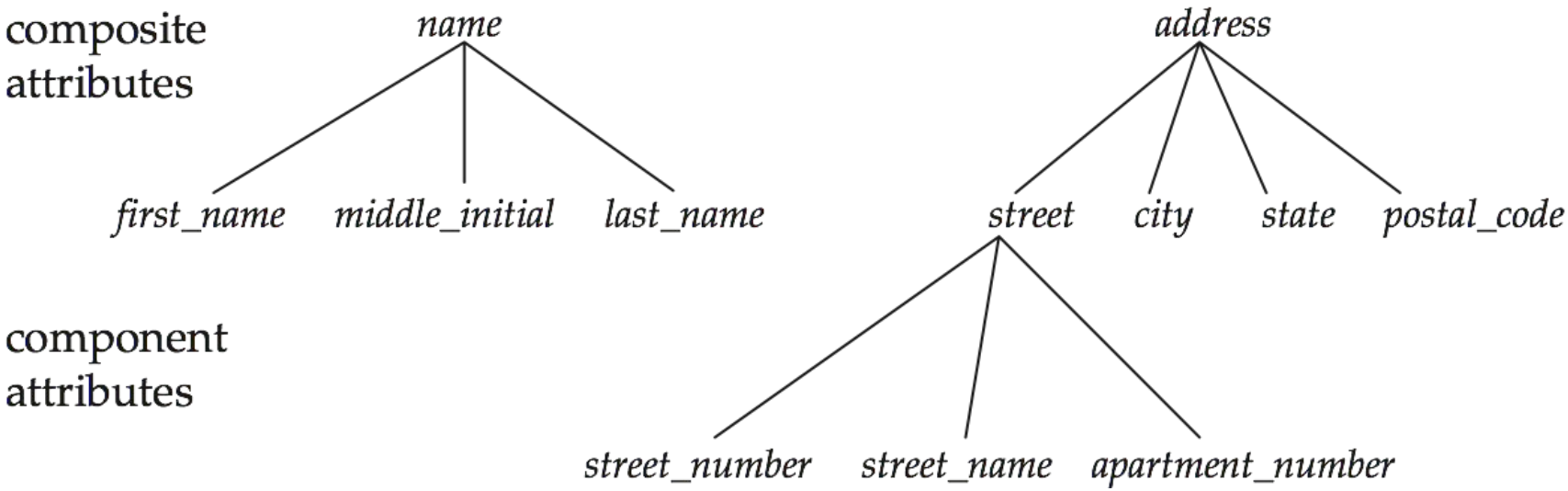


Image from “Database System Concepts” book and slides, 6th edition

Lecture Outline

- Closing the loop on Views
- Connecting to MySQL using Python
- E-R Modeling
- Entity Sets & Relationship Sets
- Attributes
- ***Cardinality***
- Keys

Mapping Cardinality Constraints

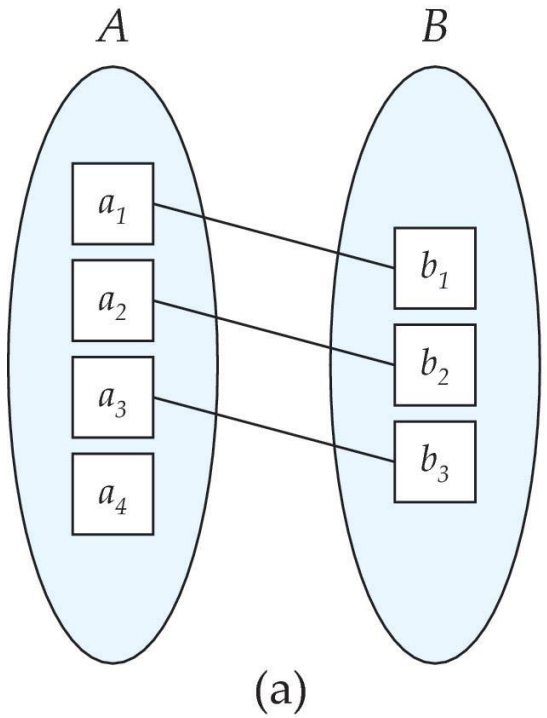
- Express the number of entities to which another entity can be associated via a relationship set.
- Most useful in describing binary relationship sets.

Mapping Cardinality Constraints

- For a binary relationship set the mapping cardinality must be one of the following types:
  - One to one
  - One to many
  - Many to one
  - Many to many

Mapping Cardinality

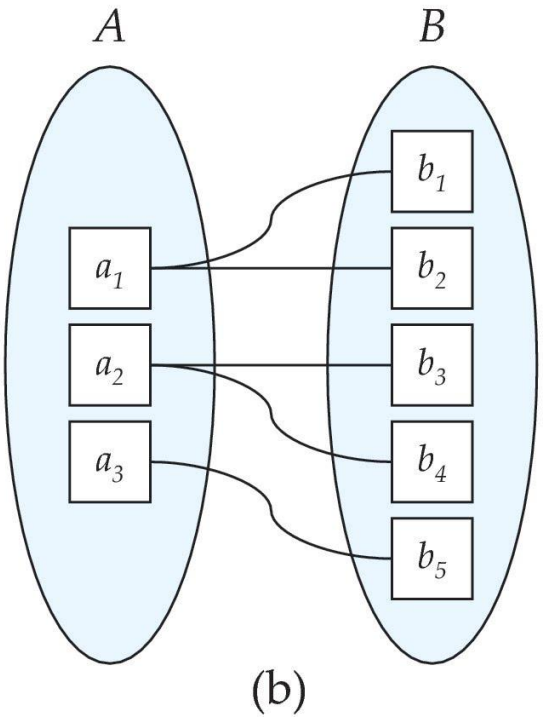
Entity in A is associated with at most 1 entity in B and entity in B is associated with at most 1 entity in A



One to one

Mapping Cardinality

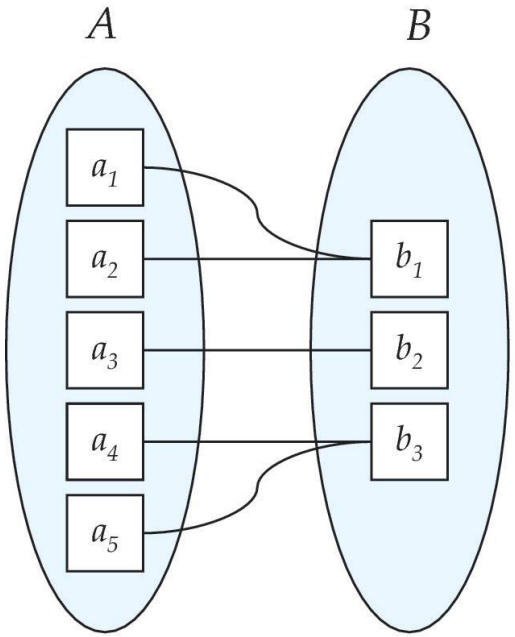
An entity in A is associated with any number (zero or more) of entities in B.
An entity in B is associated with AT MOST one entity in A.



One to many

Mapping Cardinality

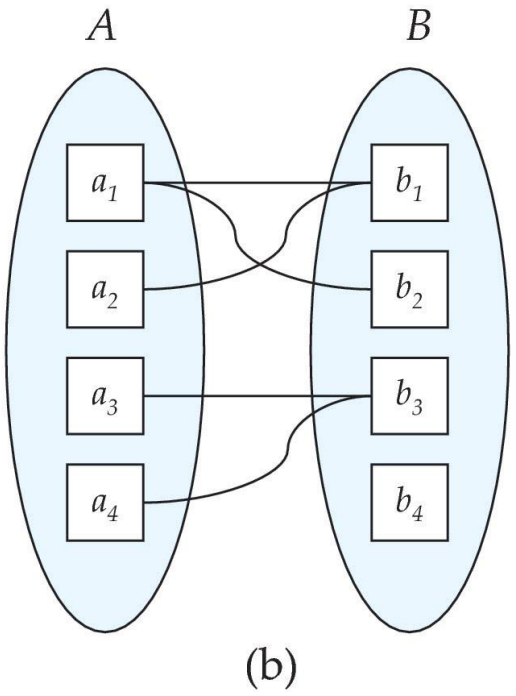
An entity in A is associated with AT MOST one entity in B.
An entity in B is associated with any number (zero or more) entities in A.



(a)
Many to one

Mapping Cardinality

An entity in A is associated with any number of entities in B, and B in A

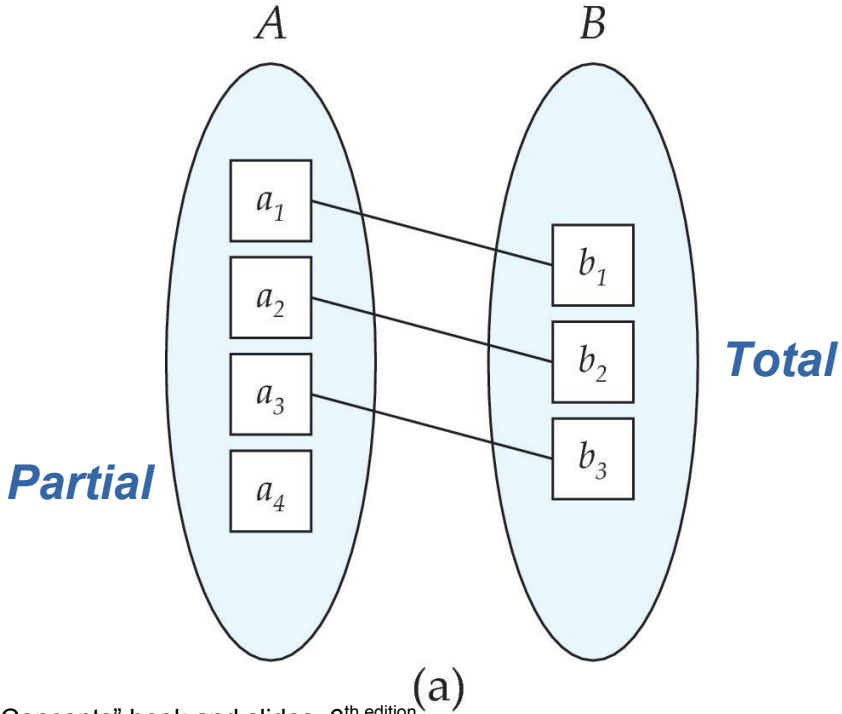


(b)

Many to many

Participation Constraints

- If every entity in entity set E participates in at least 1 relationship in the relationship set R then R is said to be ***total***
- The relationship set R is said to be ***partial*** if only some entities in entity set E participate



Based on and image from "Database System Concepts" book and slides, 6th edition

Lecture Outline

- Closing the loop on Views
- Connecting to MySQL using Python
- E-R Modeling
- Entity Sets & Relationship Sets
- Attributes
- Cardinality
- ***Keys***

How can we identify entities in a given entity set?

Keys

- A super key of an entity set is a set of one or more attributes whose values uniquely determine each entity
- A candidate key of an entity set is a minimal super key
  - ID is candidate key of instructor
  - course_id is candidate key of course
- Although several candidate keys may exist, one of the candidate keys is selected to be the primary key

Keys For Relationship Sets

- Keys can be used to uniquely identify relationships
- The combination of primary keys of the participating entity sets forms a super key of a relationship set.
  - (s_id, i_id) is the super key of advisor
  - NOTE: this means a pair of entity sets can have at most one relationship in a particular relationship set.

Keys For Relationship Sets

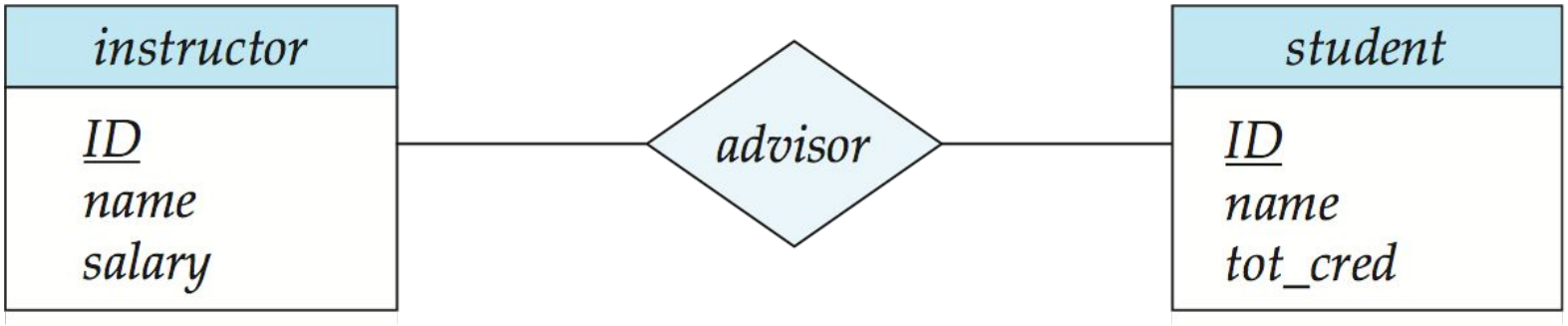
- Must consider the mapping cardinality of the relationship set when deciding what are the candidate keys
- Need to consider semantics of relationship set in selecting the primary key in case of more than one candidate key

Redundant Attributes

- Start design by identifying entity sets
- Choose identifying attributes
- Then choose relationship sets among entities
- Could result in redundant attributes across entity sets
- Try to remove
  - Only during the ER modeling phase

Up Next: E-R Diagram

- Rectangles represent entity sets.
- Diamonds represent relationship sets.
- Attributes listed inside entity rectangle
- Underline indicates primary key attributes



In-Class Exercise

Client
Name
Address
Telephone
Hobby
HouseType

Dog
Name
Breed
Size
Color

I have a list of clients who want to adopt a dog. I have dogs I want to get adopted. I will suggest dogs to clients based on their hobbies and housing. I will suggest large dogs for people who have houses. I will suggest small dogs for people who live in apartments. If someone's hobby includes an outdoor sport I will suggest a Beagle, Labrador Retriever or German Shepard. If someone's hobby includes watching television or reading books I will suggest a poodle, pug or terrier.

How would you model this further? Are the current tables sufficient? What might you add?