Relational Model

MSCI346
Lukasz Golab

Acknowledgement: slides derived from material provided by
Sliberschatz, Korth and Sudarshan, copyright 2019, www.db-book.com,
and by Prof. Wojciech Golab, ECE356

Learning Outcomes

- Relational model:
  - tuples and relations
  - keys
  - schemas
  - schema diagrams
- Reduction of E-R models to relational schemas.
- Textbook sections (6th ed.): 2.1 to 2.4, 7.6

Introduction

- Previously, we modeled data using entities and relationships.
- Relational databases instead "think" in terms of **relations**, which can be used to represent *both* entity sets and relationship sets. (E.g., one relation for *student*, one for *section*, one for *takes*.)
- Conceptually, relations are tables with **rows** and **columns**. The rows may represent entities or relationships, and the columns represent attributes.
- A relation is a mathematical object, and a table is its physical embodiment.
- Later on we will discuss mathematical operators that can be applied to relations.

Attribute Types

- The set of allowed values for each attribute is called the **domain** of the attribute.
- Attribute values are (usually) required to be **atomic**, which means they are indivisible.
  - example: a database designer may insist that the first name and the last name be separated into distinct attributes
- The special value **null** is a member of every domain, and is used to represent missing or unknown data.
- Use null values judiciously because they lead to a number of complications. (More on this later on.)

Relation Schema and Instance

- Let $A_1, A_2, \ldots, A_n$ denote attributes.
- Let $D_1, D_2, \ldots. D_n$ denote their domains.
- $R = (A_1, A_2, \ldots, A_n)$ denotes a **relation schema** over these attributes

  Example:

  instructor = (*ID, name, dept_name, salary*)

- A **relation** ***r*** conforming to schema R, denoted as $r(R)$, is a subset of

  $D_1 \times D_2 \times \ldots \times D_n$

  Thus, a relation is a set of n-tuples $(a_1, a_2, \ldots, a_n)$ where each $a_i \in D_i$.
- An element ***t*** of ***r*** is a **tuple** (specifically an **n-tuple**), and corresponds to a row in a table.
- Note: the order of elements in the tuple does not matter as long as we remember the attribute corresponding to each tuple element.
- A **relation instance** refers to the concrete values of a relation. (E.g., set of Waterloo instructors as of 10am on January 9, 2014.)

Example of a Relation Instance

attributes
(or columns)

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

tuples
(or rows)

Relations are Unordered

- Order of tuples is irrelevant (tuples may be listed in an arbitrary order).
- Example: *instructor* relation with tuples ordered arbitrarily.

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

Database

- A database typically comprises many relations.
- In the design process, information about an enterprise is broken up:

    instructor
    student
    advisor

- Sometimes, database designers make questionable decisions:
    univ (*instructor_ID, name, dept_name, salary, student_ID*, ..)
    - repetition of information (e.g., two students have the same instructor)
    - the need for null values (e.g., represent a student with no advisor)
- Normalization theory (covered later in the course) deals with how to design good relational schemas that satisfy a very precise notion of "goodness".

Keys

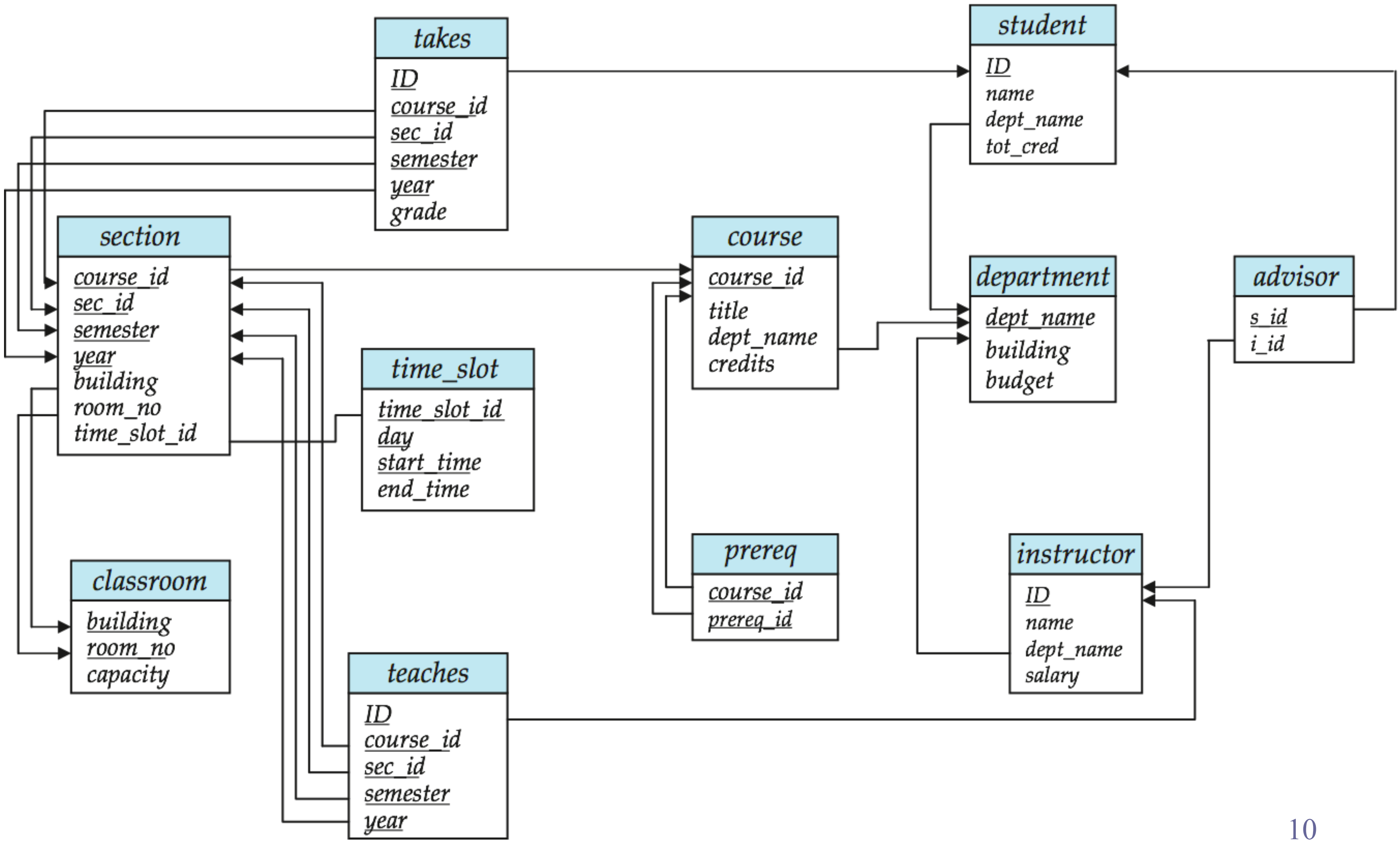
- Let R be a relation schema and let $K \subseteq R$ (K is a subset of R's attributes).
- **Superkey** and **candidate key** are defined as in the E-R model:
  - K is a **superkey** of R if values for K are sufficient to identify a unique tuple of each possible relation $r(R)$
    - Example: {*ID*} and {*ID*, *name*} are both superkeys of *instructor.*
  - Superkey K is a **candidate key** if K is minimal

    Example: {*ID*} is a candidate key for *instructor*
  - One of the candidate keys is selected to be the **primary key**.

    (Which one?)
- **Foreign key** constraint (new idea in the relational model):
  an attribute value in one relation that must appear in another relation.
  - **referencing relation** contains a **foreign key**
  - **referenced relation** contains a **referenced key**
    (usually the primary key)

Relational Schema Diagram for University DB

Example: Attribute *ID* in *takes* (referencing relation) is a foreign key that references *ID* in *student* (referenced relation).



Reduction of E-R Models to Relational Schemas

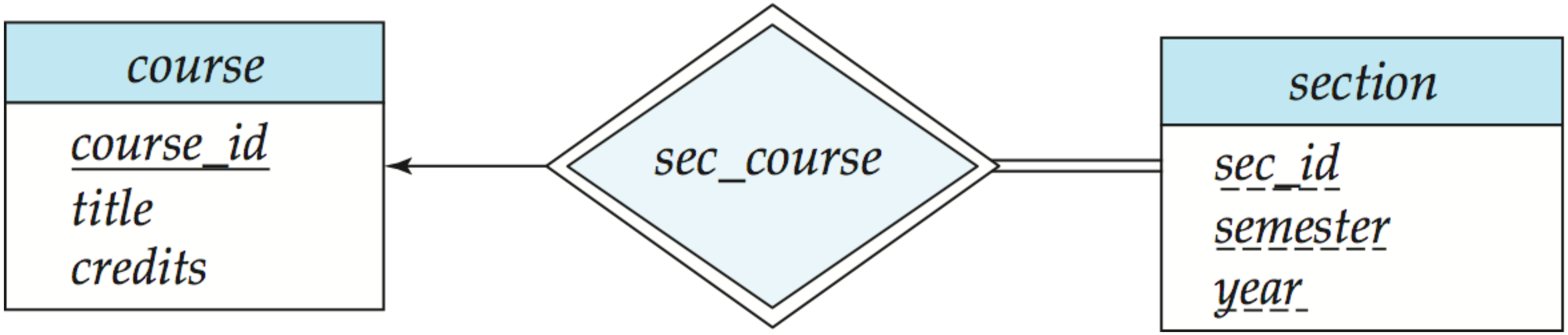
Reduction to Relation Schemas

- Entity sets and relationship sets can be expressed uniformly as *relation schemas*.
- A database that conforms to an E-R diagram can be represented by a collection of schemas.
- For each entity set and relationship set there is (usually) a unique schema that is assigned the name of the corresponding entity set or relationship set.
- In special cases (i.e., total participation) the schema for a relationship set is collapsed with the schema for one of the participating entity sets.
- The attributes of the schema reflects the attributes of the entity set or relationship set reduced to this schema.

Food for thought: how do we translate cardinality and participation constraints in the ER model into equivalent constraints in the relational model?

Entity Sets With Simple Attributes

- **Simple attribute**: not composite or multi-valued.
- An ordinary (i.e., non-weak) entity set with simple attributes reduces to a schema with the same attributes:
  student (*<u>ID</u>, name, tot_cred*)
- A weak entity set with simple attributes reduces to a schema that includes the primary key of the identifying entity set:
  section (*<u>course_id, sec_id, sem, year</u>*)

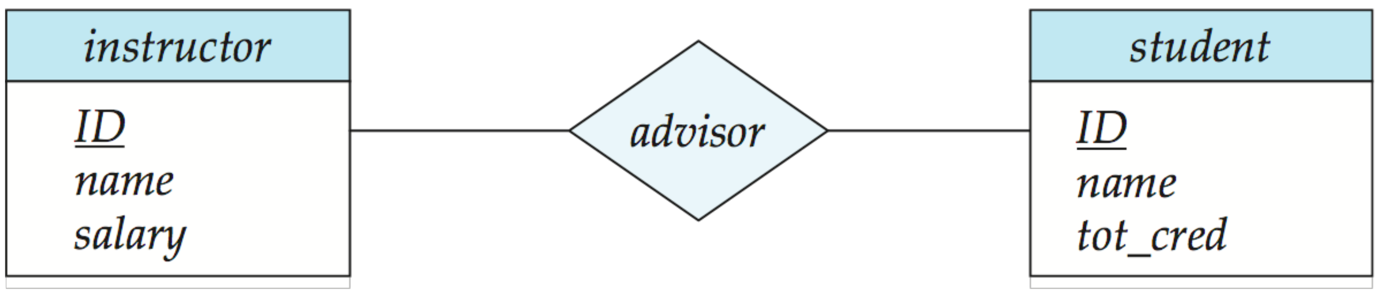


Representing Relationship Sets

- Any relationship set with simple descriptive attributes (or no descriptive attributes) can be represented as a schema with attributes for the primary keys of the participating entity sets, and for all the descriptive attributes.
- Example: schema for relationship set *advisor*

  advisor = (<u>*s_ID, i_ID*</u>)

  Note: we are free to choose new names for the primary keys of the participating entity sets, such as *s_ID* instead of *student.ID*



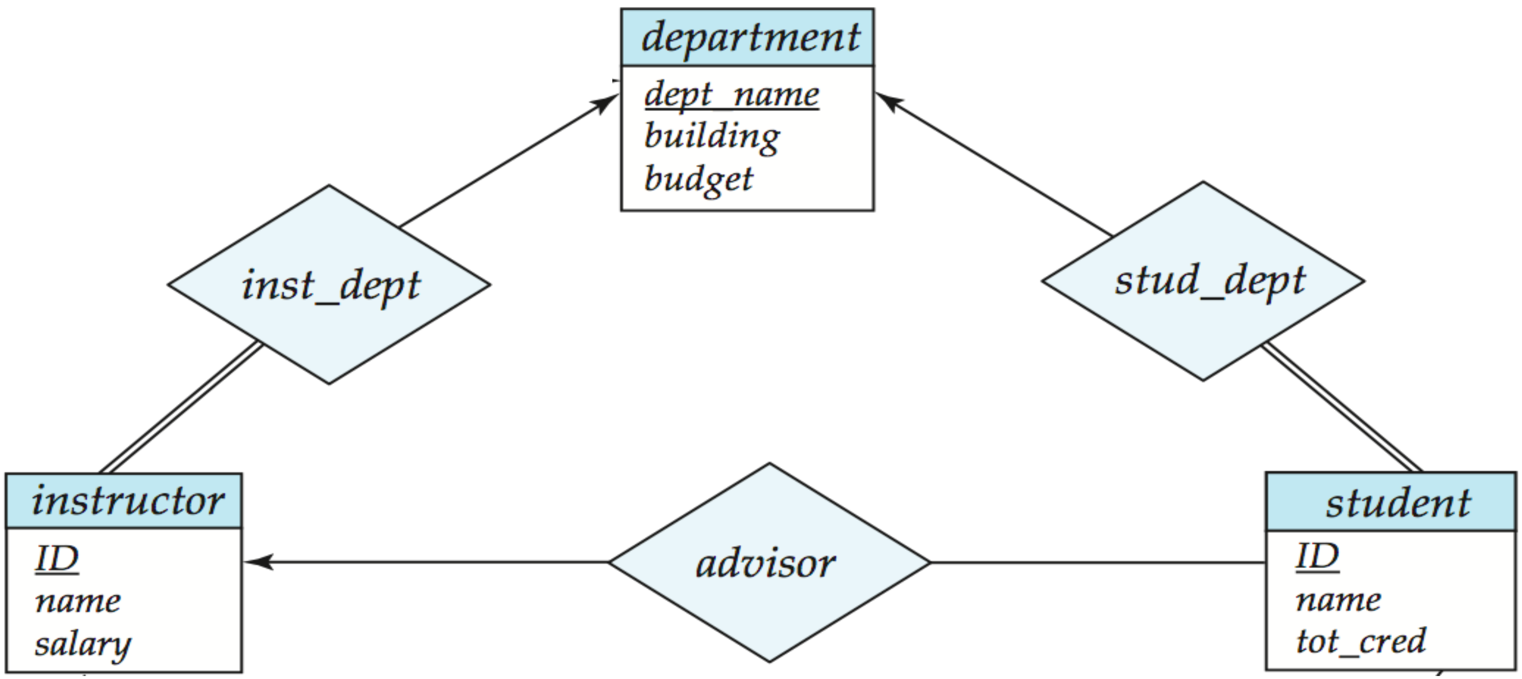
- Question: how do we correctly choose the primary key of the relation for the relationship?

Representing Relationship Sets

- Choosing primary keys for relations derived from binary relationship sets:
  - **many-to-many** – include attributes for the primary keys of both participating entity sets
    - E.g.: primary key is {*s_ID*, *i_ID*} for *advisor* relation on last slide
  - **one-to-many** – include only the attributes for the primary key of the "many" side
    - E.g.: one instructor to many students, primary key is {*s_ID*}
  - **many-to-one** – analogous to one-to-many
  - **one-to-one** – choose one of the participating entity sets (arbitrarily), and include only the attributes of that entity set
    - E.g.: one instructor to one student, primary key is {*s_ID*} or {*i_ID*}

Redundancy in Schemas

- Many-to-one and one-to-many relationship sets that are total on the many-side can also be represented by adding an extra attribute to the "many" side, containing the primary key of the "one" side.
- Example: Instead of creating a separate schema for relationship set *inst_dept*, add an attribute *dept_name* to the schema for *instructor*.
- <u>Always do this for the identifying relationship of weak entity!</u>



Composite and Multivalued Attributes

instructor

<u>ID</u>
name
  first_name
  middle_initial
  last_name
address
  street
    street_number
    street_name
    apt_number
  city
  state
  zip
{ phone_number }
date_of_birth
age ( )

- Composite attributes are flattened out by creating a separate attribute for each component attribute.
  - Example: given entity set *instructor* with composite attribute *name* with component attributes *first_name* and *last_name* the schema corresponding to the entity set has two attributes *name_first_name* and *name_last_name*
    - note: omit the prefix (in this case "*name_*") if there is no ambiguity
- Ignoring multivalued attributes, the flattened instructor schema is
  - *instructor*(*ID, first_name, middle_initial, last_name, street_number, street_name, apt_number, city, state, zip_code, date_of_birth*)

Composite and Multivalued Attributes

- A multivalued attribute *M* of an entity *E* is represented by a separate schema *EM.*
  - Schema *EM* has attributes corresponding to the primary key of *E* and an attribute corresponding to multivalued attribute *M.*
  - The primary key of *EM* comprises both attributes.
  - Example: Multivalued attribute *phone_number* of *instructor* is represented by a schema:
    inst_phone = (*ID*, *phone_number*)
  - Each value of the multivalued attribute maps to a separate tuple of the relation on schema *EM*
    - For example, an *instructor* entity with primary key 22222 and phone numbers 456-7890 and 123-4567 maps to two tuples:
      (22222, 456-7890) and (22222, 123-4567)

Specialization via Schemas

- Method 1:
  - Form a schema for the higher-level entity first.
  - Then form a schema for each lower-level entity set. Include the primary key of the higher-level entity set and local attributes.

schema	attributes
person	*ID, name, street, city*
student	*ID, tot_cred*
employee	*ID, salary*

  - Drawback: getting information about, an *employee* requires accessing two relations, the one corresponding to the low-level schema and the one corresponding to the high-level schema.

Specialization via Schemas (Cont.)

- Method 2:
  - Form a schema for each entity set with all local and inherited attributes

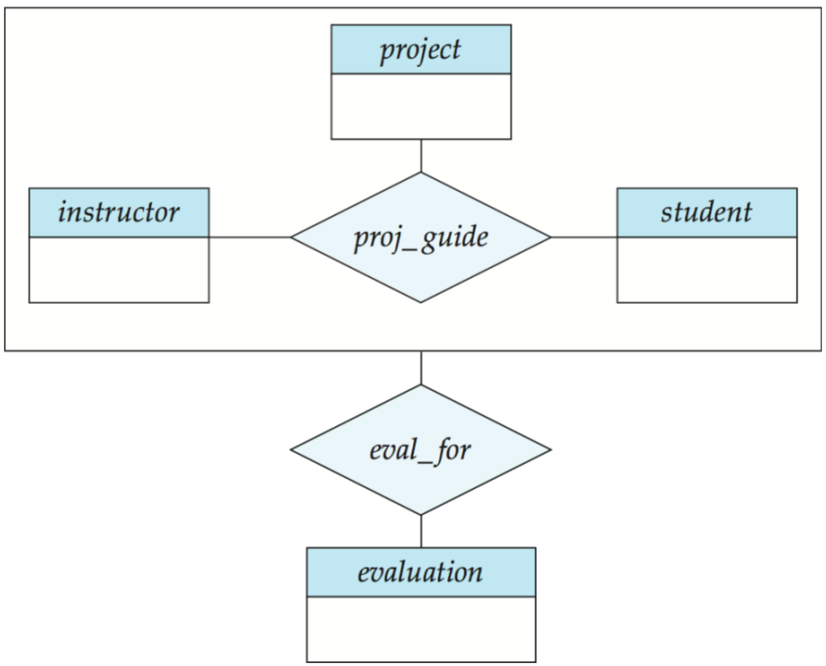
schema	attributes
person	*ID*, *name*, *street*, *city*
student	*ID*, *name*, *street*, *city*, *tot_cred*
employee	*ID*, *name*, *street*, *city*, *salary*

  - If specialization is total, the schema for the generalized entity set (*person*) is not required to store information.
    - Relation *person* can be defined as a "view" relation containing the union of student and employee. (More on views later in the course.)
    - An explicit schema may still be needed for foreign key constraints
  - Drawback: *name, street* and *city* may be stored redundantly for people who are both students and employees.

Schemas Corresponding to Aggregation

- To represent the aggregation of a relationship, simply create a schema for that relationship using the technique described earlier.
- The schema for *eval_for* comprises the primary key of the schema for *proj_guide*, the primary key for *evaluation*, as well as any descriptive attributes of *eval_for*.

  eval_for (*<u>i_ID, p_ID, s_ID, evaluation_ID</u>*)



Food for thought:
Does this make the relation for *proj_guide* redundant?