Database Management Systems
CS430 & CS630
L2

Umass Boston
Summer 2023
Cristina Maier

Part of slides are based on “Database Management System, 3rd ed., by Ramakrishnan and Gehrke

Topics

- Introduction to DBMS
- Relational Data Model
- *Relational Algebra*
- Structured Query Language (SQL)
- Conceptual Design - the Entity-Relationship Model
- Schema Refinement and Normal Forms
- Database Security and Authorization
- Application Development (Java, Python)
- Some NoSQL topics (If time permitted)

Note on Relations in Relational Algebra

- Example of database schema:
  - Sailors(sid:integer, sname:string, rating:integer, age:real)
  - Boats(bid:integer, bname:string,color:string)
  - Reserves(sid:integer,bid:integer, day:date)
- The underlined attributes in a relation's schema represent a primary key for that relation. That means no two records from the same relation can have the same value for all the attributes composing the key!
- Regardless of the presence of primary keys, relations in relation algebra MUST have UNIQUE records(that's the reason some relational algebra operations need to drop duplicates). Note that this does not mean that the values from a column are unique; this means that the entire record is unique!!
- If a primary key is present in a relation, that adds an extra constraint saying that the values for all attributes composing the key are unique within that relation (i.e. no two records from the relation can share the same values for the primary key)
- (Real RDBMS implementations (NOT relational algebra), do not typically enforce the record uniqueness (which means, two records from a relation (i.e table) can be the same). In a RDBMS, to enforce the uniqueness we have to create a primary key, which enforces the uniqueness of the key, and because of that also the uniqueness of the entire record
- For relational algebra exercises we need to remember that records in a table are unique!

Note on Relations in Relational Algebra Schema Example

- Example of database schema:
  - Sailors(<u>sid:integer</u>, sname:string, rating:integer, age:real)
  - Boats(<u>bid:integer</u>, bname:string,color:string)
  - Reserves(<u>sid:integer,bid:integer</u>, day:date)
- Because we are in the world of relational algebra, any instance of these tables has unique records (i.e. tuples, rows)
- Sailors relation: contains information about sailors. This relation has a primary key sid (meaning sailor id), which uniquely identifies a sailor. No two records from this relation could have the same sid.
- Boats relation: contains info about boats. It has a primary key bid, which uniquely identifies a boat. No two records from this relation can have the same bid
- Reserves relation: describes a relationship between sailors and boats. It contains information about reservations a sailor made for a boat. It has a primary key composed of two attributes: sid and bid. This means, no two records from this relation could have the same combined sid,bid values. This basically translates that no sailor can reserve the same boat twice (otherwise, sid,bid would appear twice with a different day field value)
- Note on relational algebra expressions: when applied on these relations, the resulted relation will not have any primary key! But we do have to drop record duplicates (records that have same values on all attributes), because in relational algebra no two records have the same values for all fields

Basic Relational Algebra Operators

- *Projection* π selects only a given subset of columns from the relation
- *Renaming* ρ helper operator. It does not return a new relation.
  - It only renames the relation and/or the fields
- *Selection* σ selects a subset of tuples/rows from the relation
- *Cross-product* $\times$ allows us to combine multiple relations, by performing the cartesian product
- *Join* $\bowtie$ combines multiple relations using conditions
  - condition join, equijoin, natural-join
- *Set-difference* $-$, *Union* $\cup$, *Intersection* $\cap$
- *Division* / used for some special types of queries. It is more complex

Set Operations: Union, Intersection, Set-difference

$R \cup S$ $R \cap S$ $R - S$

- Binary operations
- Input: two relations that must be *union-compatible*:
  - Same number of fields
  - Corresponding fields taken from left to right must have the same domain (data type)
- Note: attribute names are not used in defining union-compatible
- Output: one relation resulted from the union, intersection or difference of the two relations from the input
- Schema of the resulted relation will be equal to the schema of R

Example of two not union-compatible schemas

- Student1(sid: integer, name:string, city:string, age:integer)

- Student2(sid: integer, name: string, city: string, age: string, email: string)

Why these schemas are not union-compatible?

Example of two union-compatible schemas

- B1(bid: integer, name:string, brand:string, color:string, age:integer)

- B2(bid:integer, bname: string, brand:string, color:string, ageb: integer)

Union Example

$B1$

bid	name	brand	color	age
10	aloha	yamaha	green	4
20	hello	corsair	white	5
50	cruise	yamaha	white	10
40	vacation	yamaha	tan	6

$B2$

bid	bname	brand	color	ageb
30	autumn	yamaha	green	4
50	cruise	yamaha	white	10

$B1 \cup B2$

bid	name	brand	color	age
10	aloha	yamaha	green	4
20	hello	corsair	white	5
50	cruise	yamaha	white	10
40	vacation	yamaha	tan	6
30	autumn	yamaha	green	4

Result contains all records from B1 and B2 (duplicates are removed)

Intersection Example

$B1$

bid	name	brand	color	age
10	aloha	yamaha	green	4
20	hello	corsair	white	5
50	cruise	yamaha	white	10
40	vacation	yamaha	tan	6

$B2$

bid	bname	brand	color	ageb
100	myboat	corsair	yellow	5
20	hello	corsair	white	5
25	caribbean	yamaha	white	10
40	vacation	yamaha	tan	6

$B1 \cap B2$

bid	name	brand	color	age
20	hello	corsair	white	5
40	vacation	yamaha	tan	6

Result contains records that are present in both B1 and B2

Set Difference Example

$B1$

bid	name	brand	color	age
10	aloha	yamaha	green	4
20	hello	corsair	white	5
50	cruise	yamaha	white	10
40	vacation	yamaha	tan	6

$B2$

bid	bname	brand	color	ageb
100	myboat	corsair	yellow	5
20	hello	corsair	white	5
25	caribbean	yamaha	white	10
40	vacation	yamaha	tan	6

$B1 - B2$

bid	name	brand	color	age
10	aloha	yamaha	green	4
50	cruise	yamaha	white	10

Result contains all records from B1 that are not present in B2

Topics

- Introduction to DBMS
- Relational Data Model
- *Relational Algebra*
- Conceptual Design - the Entity-Relationship Model
- Structured Query Language (SQL)
- Schema Refinement and Normal Forms
- Database Security and Authorization
- Application Development (Java, Python)
- Some NoSQL topics (If time permitted)

Basic Relational Algebra Operators

- *Projection* π selects only a given subset of columns from the relation
- *Renaming* ρ helper operator. It does not return a new relation.
  - It only renames the relation and/or the fields
- *Selection* σ selects a subset of tuples/rows from the relation
- *Cross-product* $\times$ allows us to combine multiple relations, by performing the cartesian product
- *Join* $\bowtie$ combines multiple relations using conditions
  - condition join, equijoin, natural-join
- *Set-difference* $-$, *Union* $\cup$, *Intersection* $\cap$
- *Division /* used for some special types of queries. It is more complex

Basic Relational Algebra Operators

- *Projection* π selects only a given subset of columns from the relation
- *Renaming* ρ helper operator. It does not return a new relation.
  - It only renames the relation and/or the fields
- *Selection* σ selects a subset of tuples/rows from the relation
- *Cross-product* $\times$ allows us to combine multiple relations, by performing the cartesian product
- *Join* $\bowtie$ combines multiple relations using conditions
  - Condition Join (Theta-join), equijoin, natural-join
- *Set-difference* $-$, *Union* $\cup$, *Intersection* $\cap$
- *Division* / used for some special types of queries. It is more complex

Division

$$R/S$$

- Also noted as $R \div S$
- Useful for expressing certain kinds of queries, such as "Find the names of all sailors who have reserved all boats."
- It does not have the same importance as the other operators because it is not needed as often, but it is extremely important for certain queries
  - SQL does not have a specific construct for it. When needed, it is done using a combination of other constructs

Division (cont.)

- A(x:type1, y:type2)
- B(y:type2)
- A/B is the set of all values x (in the form of unary tuples) such that for every y value in B, there is a < x, y> tuple in A
- For each x value in the first column of A, consider the set of values that appear in the y column of A associated with that value of x. If this set contains all values y in B, then x is in the result for A/B
- A value x from A is disqualified from the result if by attaching a value y from B, we have a tuple <x, y> that is not present in A
- $A/B = \pi_x A - \pi_x((\pi_x(A) \times B) - A)$
- To understand the division operation in full generality, we have to consider the case when both x and y are replaced by a set of attributes

Division Example

R

sid	bid
12	101
10	102
12	102
11	101
10	101

B

bid
101
102

Find the ids of sailors who have reserved all boats

R/B

sid
12
10

Division Example

- Schemas:
  - Sailors(<u>sid:integer</u>, sname:string, rating:integer, age:real)
  - Boats(<u>bid:integer</u>, bname:string,color:string)
  - Reserves(<u>sid:integer,bid:integer</u>, day:date)
- Find the names of sailors who reserved all boats
  - We first apply projections to prepare Boats and Reserve to have the same format as in the definition of Division (in this case from Reserves we want to get a relation that has 2 attributes, and from Boats a relation that has 1 attribute).
  - After that, we apply the division operator to get the sid of all sailors that reserved all boats
  - Then we join this with Sailors, and get the name of the sailors

Division Example

- Schemas:
  - Sailors(<u>sid:integer</u>, sname:string, rating:integer, age:real)
  - Boats(<u>bid:integer</u>, bname:string,color:string)
  - Reserves(<u>sid:integer,bid:integer</u>, day:date)
- Find the names of sailors who reserved all boats

$$\rho(TempIds, \pi_{sid,bid}Reserves / \pi_{bid}Boats)$$

$$\pi_{sname}(TempIds \bowtie Sailors)$$

Example of result

Sailors

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.0
58	rusty	10	35.0

Reserves

sid	bid	day
22	101	10/10/22
58	101	10/11/22
22	102	10/20/22

Boats

bid	name	color
101	interlake	red
102	clipper	green

$\pi_{sid,bid} Reserves$

sid	bid
22	101
58	101
22	102

$\pi_{bid} Boats$

bid
101
102

Example Division

Sailors

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.0
58	rusty	10	35.0

Reserves

sid	bid	day
22	101	10/10/22
58	101	10/11/22
22	102	10/20/22

Boats

bid	name	color
101	interlake	red
102	clipper	green

$\pi_{sid,bid} Reserves$

sid	bid
22	101
58	101
22	102

$\pi_{bid} Boats$

bid
101
102

$TempIds$

sid
22

$TempIds \bowtie Sailors$

sid	sname	rating	age
22	dustin	7	45.0

$\pi_{sname}(TempIds \bowtie Sailors)$

sname
dustin

Basic Relational Algebra Operators

- *Projection* π selects only a given subset of columns from the relation
- *Renaming* ρ helper operator. It does not return a new relation.
  - It only renames the relation and/or the fields
- *Selection* σ selects a subset of tuples/rows from the relation
- *Cross-product* $\times$ allows us to combine multiple relations, by performing the cartesian product
- *Join* $\bowtie$ combines multiple relations using conditions
  - Condition Join (Theta-join), equijoin, natural-join
- *Set-difference* $-$, *Union* $\cup$, *Intersection* $\cap$
- *Division* $\div$ used for some special types of queries. It is more complex

Next: some practice queries

Some practice queries

- Two types of problems:
  - Given some relations instances and a relational algebra expression, provide the resulted relation (we did many of these exercises during our previous lecture)
  - Given the schemas of some relations, and a requirement describing what we want to extract, write a relational algebra expression (query) to extract such data from those relations
- We use parentheses in the queries to make the relational algebra expression unambiguous!!!

Operators Precedence

- When combined in an expression
- In decreasing order of priority
  - Selection σ, Projection π
  - Cross-product $\times$, Joins $\bowtie$, Division /
  - Set-Difference $-$, Intersection $\cap$
  - Union $\cup$
- Parenthesis can be used to change the order of operations

Example Query 1

- Given this schemas
  - Sailors(<u>sid: integer</u>, sname: string, rating: integer, age: real)
  - Boats(<u>bid: integer</u>, <u>name</u>: string, color: string)
  - Reserves(<u>sid: integer, bid:integer,</u> day: date)
- Find the name of the sailors that reserved boat # 102
- ?

Example Query 1 (cont.)

- Given this db schema:
  - Sailors(<u>sid: integer</u>, sname: string, rating: integer, age: real)
  - Boats(<u>bid: integer</u>, name: string, color: string)
  - Reserves(<u>sid: integer, bid:integer</u>, day: date)
- Find the name of the sailors who reserved boat id # 102

$$\pi_{sname}((\sigma_{bid=102}Reserves) \bowtie Sailors)$$

Example Query 2

- Given this db schema:
  - Sailors(<u>sid: integer</u>, sname: string, rating: integer, age: real)
  - Boats(<u>bid: integer</u>, name: string, color: string)
  - Reserves(<u>sid: integer, bid:integer</u>, day: date)
- Find the name and ratings of the sailors who reserved a 'red' boat

$$\pi_{sname,rating}(((\sigma_{color='red'}Boats) \bowtie Reserves) \bowtie Sailors)$$

Example Query 2 Solution 2 - small change (reduce the number of fields in the join)

- Given this db schema:
  - Sailors(sid: integer, sname: string, rating: integer, age: real)
  - Boats(bid: integer, name: string, color: string)
  - Reserves(sid: integer, bid:integer, day: date)
- Find the name and ratings of the sailors who reserved a 'red' boat

$$\pi_{sname,rating}(\pi_{sid}(\pi_{bid}(\sigma_{color='red'}Boats) \bowtie Reserves) \bowtie Sailors)$$

Example Query 3

- Given this db schema:
  - Sailors(sid: integer, sname: string, rating: integer, age: real)
  - Boats(bid: integer, name: string, color: string)
  - Reserves(sid: integer, bid:integer, day: date)
- Find the name of the sailors who reserved a red or a green boat
- ?

Example Query 3

- Given this db schema:
  - Sailors(sid: integer, sname: string, rating: integer, age: real)
  - Boats(bid: integer, name: string, color: string)
  - Reserves(sid: integer, bid:integer, day: date)
- Find the name of the sailors who reserved a red or a green boat

$$\pi_{sname}(((\sigma_{color='green' \vee color='red'} Boats) \bowtie Reserves) \bowtie Sailors)$$

Example Query 3
(Solution is written in two steps)

- Given this db schema:
  - Sailors(<u>sid: integer</u>, sname: string, rating: integer, age: real)
  - Boats(<u>bid: integer</u>, name: string, color: string)
  - Reserves(<u>sid: integer, bid:integer</u>, day: date)

- Find the name of the sailors who reserved a red or a green boat

$$\rho(TempBoats, \sigma_{color='green' \vee color='red'} Boats)$$

$$\pi_{sname}(TempBoats \bowtie Reserves \bowtie Sailors)$$

Example Query 4

- Given this db schema:
  - Sailors(sid: integer, sname: string, rating: integer, age: real)
  - Boats(bid: integer, name: string, color: string)
  - Reserves(sid: integer, bid:integer, day: date)
- Find the name of the sailors who reserved a red and a green boat
- ?

Example Query 4

- Given this db schema:
  - Sailors(<u>sid: integer</u>, sname: string, rating: integer, age: real)
  - Boats(<u>bid: integer</u>, name: string, color: string)
  - Reserves(<u>sid: integer</u>, <u>bid</u>:integer, day: date)
- Find the name of the sailors who reserved a red and a green boat
  - First we get the ids of the sailors who reserved a green boat; Then the ids of the sailors who reserved a red boat. Then we have to intersect them to get the ones that reserved both colors. We join with Sailors to get the name.

$$\rho(TempGreen, \pi_{sid}((\sigma_{color='green'}Boats) \bowtie Reserves))$$

$$\rho(TempRed, \pi_{sid}((\sigma_{color='red'}Boats) \bowtie Reserves))$$

$$\pi_{sname}((TempGreen \cap TempRed) \bowtie Sailors)$$

Example Query 5

- Given this db schema:
  - Sailors(<u>sid: integer</u>, sname: string, rating: integer, age: real)
  - Boats(<u>bid: integer</u>, name: string, color: string)
  - Reserves(<u>sid: integer, bid:integer</u>, day: date)
- Find the name of the sailors who reserved only red boats
- It means those sailors reserved red boats and they did not reserve any other boats of different color
- $\rho(TempRed, \pi_{sid}(\sigma_{color='red'} Boats \bowtie Reserves))$
- $\rho(TempNotRed, \pi_{sid}((\sigma_{color<>'red'} Boats) \bowtie Reserves))$
- $\pi_{sname}((TempRed - TempNotRed) \bowtie Sailors)$

Example Query 6
An example of Self Join

- Given this db schema:
  - Sailors(<u>sid: integer</u>, sname: string, rating: integer, age: real)
  - Boats(<u>bid: integer</u>, name: string, color: string)
  - Reserves(<u>sid: integer, bid:integer</u>, day: date)
- Find the oldest sailors (this means sailors with max age)
- We only need Sailors relation. We need to join it with itself.
  - If a sailor's age is lower than any other sailor's age, then this sailor will not be in the results (We get those sailors in TempLeft relation)
  - We use set-difference. Only sailors who are not younger than any sailor could be in the result
- $\rho(S1, Sailors)$
- $\rho(S2, Sailors)$
- $\rho(Temp(1 \rightarrow fsid, 2 \rightarrow fsname, 3 \rightarrow frating, 4 \rightarrow fage), S1 \bowtie_{S1.age<S2.age} S2)$
- $\rho(TempLeft, \pi_{fsid, fsname, frating, fage} Temp)$
- $Sailors - TempLeft$

Example Query 6
An example of Self Join
Instances Example

Sailors

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.0
58	rusty	10	35.0

Reserves

sid	bid	day
22	101	10/10/22
58	101	10/11/22
22	102	10/20/22

Boats

bid	name	color
101	interlake	red
102	clipper	green

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.0
58	rusty	10	35.0

S2

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.0
58	rusty	10	35.0

Example Query 6
An example of Self Join
Instances Example (cont.)

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.0
58	rusty	10	35.0

S2

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.0
58	rusty	10	35.0

$$S1 \bowtie_{S1.age<S2.age} S2$$

(sid)	(sname)	(rating)	(age)	(sid)	(sname)	(rating)	(age)
22	dustin	7	45.0	31	lubber	8	55.0
58	rusty	10	35.0	22	dustin	7	45.0
58	rusty	10	35.0	31	lubber	8	55.0

Columns get unnamed

Temp

fsid	fsname	frating	fage	(sid)	(sname)	(rating)	(age)
22	dustin	7	45.0	31	lubber	8	55.0
58	rusty	10	35.0	22	dustin	7	45.0
58	rusty	10	35.0	31	lubber	8	55.0

Example Query 6
An example of Self Join
Instances Example (cont.)

Temp

fsid	fsname	frating	fage	(sid)	(sname)	(rating)	(age)
22	dustin	7	45.0	31	lubber	8	55.0
58	rusty	10	35.0	22	dustin	7	45.0
58	rusty	10	35.0	31	lubber	8	55.0

TempLeft

fsid	fsname	frating	fage
22	dustin	7	45.0
58	rusty	10	35.0

Sailors – TempLeft

sid	sname	rating	age
31	lubber	8	55.0

Example Query 7

- Given this db schema:
  - Sailors(<u>sid: integer</u>, sname: string, rating: integer, age: real)
  - Boats(<u>bid: integer</u>, name: string, color: string)
  - Reserves(<u>sid: integer, bid:integer</u>, day: date)
- Find the name of the sailors who are 40 years old and who reserved a white or a green boat
- ?

Example Query 7 (cont.)

- Given this schemas
  - Sailors(<u>sid: integer</u>, sname: string, rating: integer, age: real)
  - Boats(<u>bid: integer</u>, name: string, color: string)
  - Reserves(<u>sid: integer, bid:integer</u>, day: date)
- Find the name of the sailors who are 40 years old and who reserved a white or a green boat

$$\pi_{sname}((\sigma_{age=40} Sailors) \bowtie Reserves \bowtie (\sigma_{color='white' \vee color='green'} Boats))$$

Topics

- Introduction to DBMS
- Relational Data Model
- *Relational Algebra*
- Conceptual Design: the Entity-Relationship Model
- Structured Query Language (SQL)
- Schema Refinement and Normal Forms
- Database Security and Authorization
- Application Development (Java, Python)
- Some NoSQL topics (If time permitted)

Database Design

- Step1: Requirement Analysis
  - What data we want to store, what apps use it. Usually, an informal process
- Step2: Conceptual Design
  - The information gathered in requirement analysis is used to present a high level description of what data we want to store in db
  - It is usually done using the Entity Relationship (ER) model
  - ER diagram is an approximation of the db
  - Logical Design translates the ER model into a relational model

Database Design

- Step 3: Logical Design
  - Convert the ER model into a relational database schema
  - Relational data model
  - Logical Design can be further refined during Schema Refinement
- Step 4: Schema Refinement
  - Normalization
- Step 5: Physical Design
  - Considers performance criteria
  - Database tuning
  - Storage and indexing
- Step 6: Application And Security Design

Database Design

- Today we'll talk about the Conceptual Design: The Entity-Relationship Model
- We will learn how to create ER diagrams

Entities, Attributes and Entity Sets

- An Entity represents a real-world object
  - e.g.: employee entity; student entity
  - Is characterized by a set of attributes (i.e. properties)
  - Each attribute has a domain (same as a variable has a data type)
- Entity set represents a collection of similar entities
  - All entities in a set share the same set of attributes
  - E.g. All employees from an organization; all students from a school

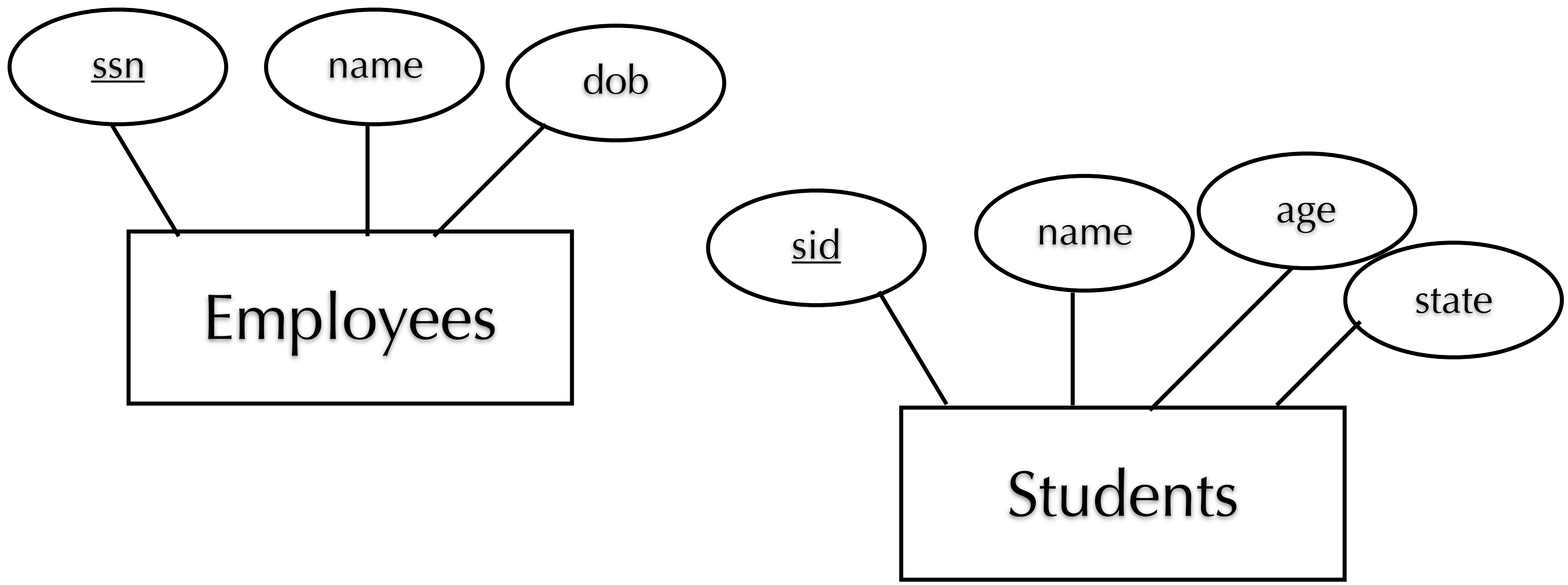
Keys

- For each entity set , we might be able choose a key
- A key is a minimum set of attributes in an entity set that uniquely identifies an entity
- There could be more candidate keys. From those, during the design, we'll choose one as a primary key
- For today's lecture, we assume that each entity set contains a key

Representation

- Entity set: represented as rectangle
- An attribute: represented as an oval
- A key: underlined attribute(s) name
- Edges: used to connect an entity set to its attributes

Example of Employees and Students Entity Sets



Relationship and Relationship Set

- Relationship: association among two or more entities
    - A relationship can have descriptive attributes
        - But relationships must be determined only by entities
    - E.g.: Mary is a student at Umass Boston
    - Since 09/06/2022
- Relationship set: collection of similar relationships
- A relationship set can be seen as a set of n-tuples where $\{(e_1, e_2, \ldots, e_n) \mid e_1 \in E_1, e_2 \in E_2, \ldots, e_n \in E_n\}$, where $E_i, i \in [1,n]$ is an entity set
- Each n-tuple denotes a relationship involving n Entities Sets
- If a relationship between two entity sets, we have a binary relationship(2-tuples); three entity sets: ternary relationship (3-tuple)
- Several relationship sets might involve the same entity set

Questions?