Session 2

Relational Model

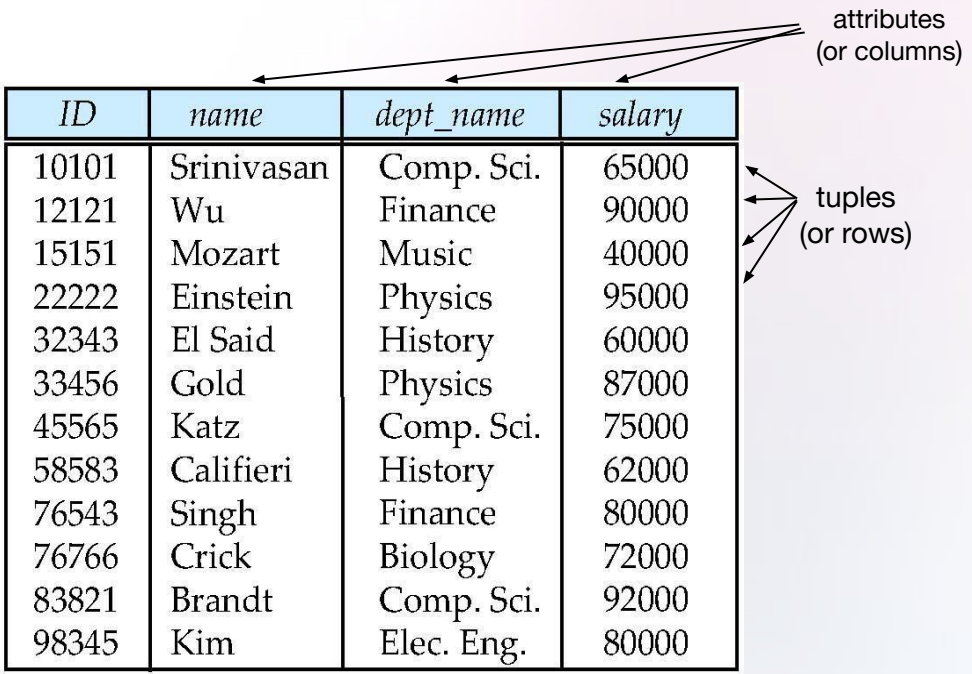
- Relation definition
- Primary and foreign keys
- Constraints
- Relational queries

Relational model

- Relational Model was introduced in 1970 by Edgar Codd (at IBM)
- Simple, uniform data structures, based on relations (as sets)
- Relational Model is basis for most DBMSs, e.g., Oracle, Microsoft SQL Server, IBM DB2, Sybase, PostgreSQL, MySQL etc.
- Typically used in conceptual design
  - either directly (creating tables using SQL DDL)
  - or derived from a given Entity-Relationship schema
- We will see an automatic approach to convert from E-R to relations

Sample relation

- A relation can be visualized as a table
- Attributes of the relation: Columns
- Elements of the relation: Rows



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

Formal definition of relation

- A relation r over collection of sets (**domain values**) $D_1, D_2, \ldots, D_n$ is a subset of the Cartesian Product $D_1 \times D_2 \times \ldots \times D_n$
- A relation thus is a set of n-tuples $(d_1, d_2, \ldots, d_n)$ where $d_i \in D_i$.

Example:

D_1 = studentID = { 412, 307, 540 }

D_2 = studentName = { Smith, Jones }

D_3 = major = { CS, PS, LAW, MBA }

r = { (412, Smith, CS), (307, Smith, CS), (540, Jones, PS),
(412, Jones, MBA), (307, Smith, LAW) }

r is a relation over studentID × studentName × major

Domain values for major: CS, PS, LAW, MBA

Formal definition of relation: Schema

- Let $A_1, A_2, \ldots, A_n$ be attribute names with associated domains $D_1, D_2, \ldots, D_n$, then

  $R (A_1: D_1, A_2: D_2, ..., A_n: D_n)$

  is a relation schema.

Example:

Student (studentID: integer, studentName: string, major: string)

- A relation schema defines name and the structure of the relation
- A collection of relation schemas is called a relational database schema.
- An instance of a relation is denoted as **r(R)**

Relation instance

- A relation instance **r(R)** can be represented as a table with n columns and a number of rows.
- Each row of the table is an element of relation r.
- For simplicity, we say relation instead of relation instance.
- An element **t ∈ r(R)** is called a **tuple** (or row).

Example:

```
r = { (412, Smith, CS), (307, Smith, CS), (540, Jones, PS) }
```

studentID	studentName	major
412	Smith	CS
307	Smith	CS
540	Jones	PS

Relational model concepts

- Domain Values `{412, 307, 540}`
- Attributes `studentID, studentName`
- Relation schema `Student (studentID: number, studentName: string, major: string)`
- Relation instance `r = { (412, Smith, CS), (307, Smith, CS), (540, Jones, PS) }`
- Tuple `(412, Smith, CS)`

Properties of relations

- If we treat the relation schema like a class (in OOP), then a relation instance is like an object
- The order of tuples (rows) in a relation (table) is not important (because it is a set)
- No duplicates are allowed – Think of them as sets
- The order of attributes in a tuple is important
- Attribute values are (normally) required to be atomic; that is, indivisible
- The special value **null** is a member of **every domain**. Indicated that the value is “unknown”
- Integrity Constraints
    - Integrity constraints (ICs): must be true for any instance of a relation schema
    - ICs are specified when the schema is defined
    - ICs are checked by the DBMS when relations (instances) are modified (e.g. when a new tuple is added to a relation instance)

Keys

- Let $K \subseteq R$ (K is a subset of attributes of schema R)
- K is a **superkey** of R if values for K are sufficient to identify a unique tuple of each possible relation r(R)

Example:

Student (studentID, studentName, major)

{studentID}, {studentID, studentName} are both superkeys of Student

- Superkey K is a **candidate key** if K is minimal

Example:

{studentID} is a candidate key of Student.

If we know that student names are unique, another candidate key is {studentName}

- One of the candidate keys is selected to be the **primary key**.

Foreign Key

- **Foreign key** constraint: Value in one relation must appear in another
  - Referencing relation uses the foreign key
  - Referenced relation originally owns the key.
- Given the following relations

  `Student (studentID, studentName, major, `**`departmentID`**`)`

  `Department (`**`departmentID`**`, departmentName, floor)`

  departmentID in Student is a foreign key referencing Department.

Foreign Key Constraint

- Foreign / primary key attributes must have matching domains

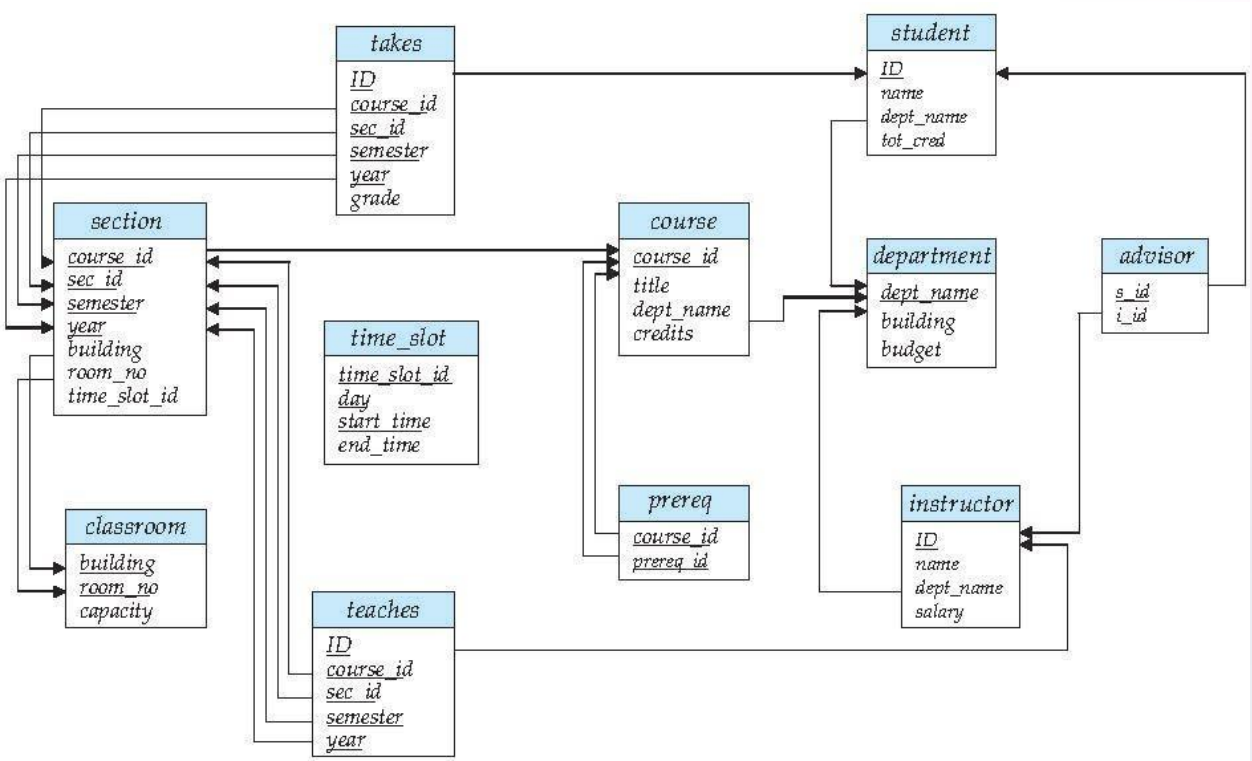
```
Department (name: string, floor: number)
d = { (CS, 1), (Art, 1) (Business, 2), (Music, 3) }

Instructor (ID: integer, name: string, department_name: string)
i = { (203, John, CS), (412, Johan, Music), (413, Amadeus, Music),
      (342, Bill, Physics) }
```

- Any value as a FK (Foreign Key) which does not exist in the set of PK (Primary Key) is not allowed.
- This is called **referential integrity**.

NULL values

- Special value that fits any domain
  - When an attribute is NULL:
  - It either doesn't have a value or the value is unknown
- Primary Keys cannot be null
- Foreign Keys can be null
- NULLs aren't comparable
- As part of ICs, we can exclude NULLs (we can enforce that certain values not be null)

Visualizing relational schemas



A set of relational schemas with the same context is called a database schema.

An example of University database schema.

Relations Examples

- A database of online shop
- Product
  - ID, title, description
  - Price
  - Seller
  - Reviews
- Two relations

Product (<u>ID</u>, title, description, price, seller, reviews)

PK: ID

FK: seller → Seller.name

Seller (<u>ID</u>, name, website)

PK: ID

Product relation

Product (ID, title, description, price, seller, reviews)

ID	Title	Description	Price	Seller	Reviews
123	iPod	NEWEST MODEL Apple Ipod Nano 7th Generation Silver 16 GB Includes Generic Earpods and USB Cable - Non Retail Packaging	100.80 USD	Gadgets World	{ “Mike”: “Doesn’t have WiFi”, “Alice”: “Good quality” }
201	Apple Watch	Size:38 mm | Color: Space Gray Aluminum Case, Black Sport Band	200 EUR	Exclusives Direct	{ “Mike”: “Works fine without problems”, “Bob”: “You need an iPhone to use this”, “Alice”: “Useless, I prefer normal watches” }

sourcemind

Issues?

- Atomicity: Reviews
- Uniform data: Price currencies
- Foreign keys: Seller table

Seller FK

Product

ID	Title	Description	Price	Seller	Reviews
123	iPod	Blah blah	100.80 USD	Gadgets World	NULL
201	Apple Watch	Blah blah	200 EUR	Exclusives Direct	NULL
202	Charger	Blah blah	10 USD	Gadgets World	NULL

Issue: Changing name of a seller

Seller

ID	name	website
1	Gadgets World	Blah blah
2	Exclusives Direct	Blah blah

Seller FK

Product

ID	Title	Description	Price	SellerID	Reviews
123	iPod	Blah blah	100.80 USD	1	NULL
201	Apple Watch	Blah blah	200 EUR	2	NULL
202	Charger	Blah blah	10 USD	1	NULL

Using the ID as the foreign key is better

FK: SellerID → Seller.ID

Seller

ID	name	website
1	Gadgets World	www.gadgets.com
2	Exclusives Direct	www.direct.com

Reviews

Product

ID	Title	Description	Price	SellerID
123	iPod	Blah blah	100.80 USD	1
201	Apple Watch	Blah blah	200 EUR	2
202	Charger	Blah blah	10 USD	1

FK: SellerID → Seller.ID

Seller

ID	name	website
1	Gadgets World	www.gadgets.com
2	Exclusives Direct	www.direct.com

FK: ID → Product.ID
PK: {ID, username}

ID	username	review	rating
123	Mike	Doesn't have WiFi	3
201	Alice	Works only with iPhone	2
123	Bob	Works fine for me	4

Refined database schema

Product (<u>ID</u>, title, description, price, seller, reviews)

PK: ID

FK: seller → Seller.name

Seller (<u>ID</u>, name, website)

PK: ID

Product (<u>ID</u>, title, description, price, seller, reviews)

PK: ID

FK: seller → Seller.ID

Seller (<u>ID</u>, name, website)

PK: ID

Review (<u>ID</u>, <u>username</u>, review, rating)

PK: {ID, username)

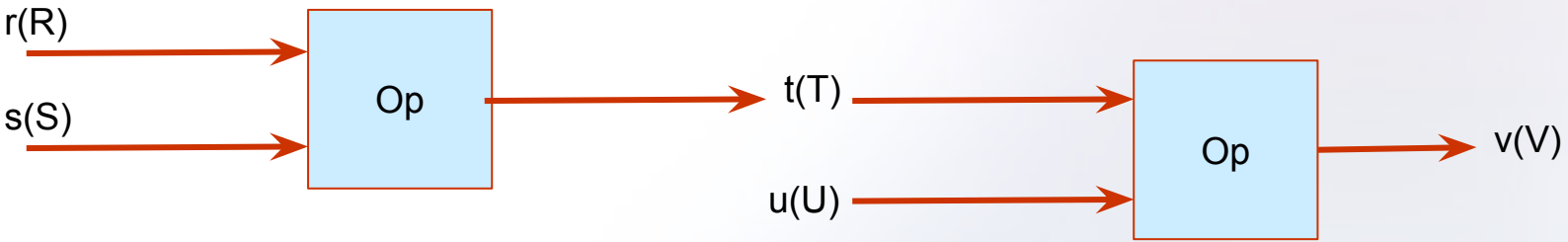
FK: ID → Product.ID

Relational queries

- A query language (QL) is a language that allows users to manipulate and retrieve data from a database.
- SQL is based on Relational Algebra
- Sample queries
  - Find the products sold by “Gadgets World”
  - Find the price of an “Apple Watch”
  - Find the website of the seller of an “Apple Watch”
- Query Language is the syntax that allows us to express these queries in our system and get them executed

Relational operations

- Query languages are composed of operations
- Operations that take one or two relations as input
- They always produce a single operation as the output
- Allow for using the output of an operation as an input of next operation, thus enabling chaining of operations



Relational algebra

- Queries in relational algebra are applied to relation instances, result of a query is again a relation instance
- Six basic operators in relational algebra:

Operator	Symbol	Description
Selecion	σ	selects a subset of tuples from a relation
Projection	π	deletes unwanted columns from a relation
Cartesian product	×	allows to combine two relations
Set difference	−	tuples in relation 1, but not in relation 2
Union	∪	tuples in relation 1 plus tuples in relation 2
Rename	ρ	renames attribute(s) and relation

Selection: $\sigma_P(r)$

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

- Given a relation r with schema R, select only the tuples where the condition **P** holds over **r** (is true)
- P is a simple or complex boolean expression
- Syntax

$$\sigma_{A=B \ \wedge \ D > 5} (r)$$

A	B	C	D
α	α	1	7
β	β	23	10

- General form:

$$\sigma_P(r) := \{t \mid t \in r \text{ and } P(t)\}$$

Declarative queries

- Relational query languages are **declarative**, meaning we don't specify the steps to execute.
- We specify the **properties** of the result.
- The **query engine** (DBMS) must figure out how to execute the queries.

```
List list = new List()
for ( Row t : r ) {
    if (t.A == t.B && t.D > 5) {
        list.add( t )
    }
}
return list
```

$$\sigma_{A=B \wedge D > 5}(r)$$

Exercise

Write queries to:

Select products more expensive than 100 USD

Select products more expensive than 40,000 XOF

* All prices in the Product table are in USD

ID	Title	Description	Price	SellerID
123	iPod	Blah blah	100	1
201	Apple Watch	Blah blah	200	2
202	Charger	Blah blah	10	1
312	Wallet	From leather, black	30	3
124	iPad	iPad mini, white	400	1

Projection: $\Pi_A(r)$

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

A	C
α	1
α	1
β	1
β	2

=

A	C
α	1
β	1
β	2

- Given a relation r with schema R with attributes A1, A2, ..., An, the projection over relation r and attributes A1, A2, ..., Ak is the relation that has k columns obtained by erasing all columns from r that are not listed.
- Duplicate rows are removed from result because relations are sets.
- Syntax

$$\pi_{A1,A2,...,Ak}(r)$$

Exercise

- Selection product title
- Selection seller IDs

ID	Title	Description	Price	SellerID
123	iPod	Blah blah	100	1
201	Apple Watch	Blah blah	200	2
202	Charge r	Blah blah	10	1
312	Wallet	From leather, black	30	3
124	iPad	iPad mini, white	400	1

Cartesian product: r × s

A	B
α	1
β	2

r

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

s

- Defined as: r × s := {tq | t ∈ r and q ∈ s}
- The combination of all rows from both tables
- Assume that attributes of r(R) and s(S) are disjoint, i.e., R ∩ S = ∅.

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

Cartesian product with non-disjoint attributes

- If attributes of r(R) and s(S) are not disjoint, then the rename operation must be applied first.
- Column B in relation r → r.B
- Column B in relation s → s.B

A	B
α	1
β	2

r

B	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

s

A	r.B	s.B	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

Exercise: Compute I × D

Relation I = (inst_id, inst_name, bank_acct, dept_name, salary)

inst_id	inst_name	bank_acct	dept_name	salary
1	Alex	182353	Computer Science	10000
3	Winston	893036	Political Science	10000
5	Sarah	638303	Law	8000

Relation D = (dept_name, dean_id, building, floor)

dept_name	dean_id	building	floor
Computer Science	1	N2	3
Political Science	3	N2	2
Music	NULL	N1	1

I × D

inst_id	inst_name	bank_acct	I.dept_name	salary	D.dept_name	dean_id	building	floor
1	Alex	182353	Computer Science	10000	Computer Science	1	N2	3
1	Alex	182353	Computer Science	10000	Political Science	3	N2	2
1	Alex	182353	Computer Science	10000	Music	NULL	N1	1
3	Winston	893036	Political Science	10000	Computer Science	1	N2	3
3	Winston	893036	Political Science	10000	Political Science	3	N2	2
3	Winston	893036	Political Science	10000	Music	NULL	N1	1
5	Sarah	638303	Law	8000	Computer Science	1	N2	3
5	Sarah	638303	Law	8000	Political Science	3	N2	2
5	Sarah	638303	Law	8000	Music	NULL	N1	1

Exercise: Find Alex's address

1. Compute the cartesian product of I and D (`I × D`)
2. Select only the rows where `I.dept_name` and `D.dept_name` match
3. Select the row by instructor's name `inst_name`
4. Project the result to keep only the instructor's name, building name, and floor number

Solution

$$r = I \times D$$

$$t = \sigma_{I.dept\_name=D.dept\_name \ \wedge \ inst\_name='Alex'}(r)$$

$$\Pi_{inst\_name,building,floor}\ (t)$$

$$\Pi_{inst\_name,building,floor}\ (\sigma_{I.dept\_name=D.dept\_name \ \wedge \ inst\_name='Alex'}(I \times D))$$

Since the result of relational operations is also a relation, we can **chain** the operations and make complex queries.

Set Difference: r – s

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

A	B
α	1
β	1

- Defined as: r – s := {t | t ∈ r and t ∉ s}
- For r – s to be computable:
  1. r and s must have the same dimension (number of attributes)
  2. Attribute domains must be compatible

Union: r ∪ s

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

A	B
α	1
α	2
β	1
β	3

- Defined as: Defined as r ∪ s := {t | t ∈ r or t ∈ s}
- For r ∪ s to be computable:
  1. r and s must have the same dimension (number of attributes)
  2. Attribute domains must be compatible

Natural Join: r ⋈ s

A	B	C	D
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

r

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	ε

s

A	B	C	D	E
α	1	α	a	α
α	1	α	a	γ
α	1	γ	a	α
α	1	γ	a	γ
δ	2	β	b	δ

- The natural join of relations r and s, r ⋈ s, is a combined operation:
  1. Cartesian product of r and s, r × s
  2. Selection based on **equality** of all **common attributes** of r and s
  3. Projection based on union of attributes of r and s

$$r \bowtie s = \Pi_{A,\, r.B,\, C,\, r.D,\, E}(\sigma_{r.B = s.B \wedge r.D = s.D}(r \times s))$$

sourcemind

Exercise: Compute I ⋈ D

Relation I = (inst_id, inst_name, bank_acct, dept_name, salary)

inst_id	inst_name	bank_acct	dept_name	salary
1	Alex	182353	Computer Science	10000
3	Winston	893036	Political Science	10000
5	Sarah	638303	Law	8000

Relation D = (dept_name, dean_id, building, floor)

dept_name	dean_id	building	floor
Computer Science	1	N2	3
Political Science	3	N2	2
Music	NULL	N1	1

Summary of relational algebra operations

Symbol (Name)	Example of Use
σ (Selection)	$\sigma_{salary >= 85000}(instructor)$
	Return rows of the input relation that satisfy the predicate.
Π (Projection)	$\Pi_{ID, salary}(instructor)$
	Output specified attributes from all rows of the input relation. Remove duplicate tuples from the output.
x (Cartesian Product)	*instructor* x *department*
	Output pairs of rows from the two input relations that have the same value on all attributes that have the same name.
∪ (Union)	$\Pi_{name}(instructor) \cup \Pi_{name}(student)$
	Output the union of tuples from the *two* input relations.
– (Set Difference)	$\Pi_{name}(instructor) - \Pi_{name}(student)$
	Output the set difference of tuples from the two input relations.
⋈ (Natural Join)	*instructor* ⋈ *department*
	Output pairs of rows from the two input relations that have the same value on all attributes that have the same name.

Rename $\rho_x(r)$

- It is sometimes useful to rename a relation
  - To assign a name to a calculated result
  - To assign a new name to an existing relation
- Also known as **copy** operation

Example:

To compute cartesian product of a relation with itself

Relations *r*

A	*B*
α	1
β	2

r

$r \times \rho_s(r)$

r.A	*r.B*	*s.A*	*s.B*
α	1	α	1
α	1	β	2
β	2	α	1
β	2	β	2

Homework 2

1. Using the Instructor and Department relations (tables) in the examples, write relational queries to compute:
   a. List of bank accounts in the Law department
   b. List of departments and their corresponding deans' names
      R (dept_name, inst_name),
      where in each row, the instructor with `inst_name` is the dean of departname with `dept_name`

2. Given a relation over R(value), using only the relational algebra, find the maximum value in the relation.

   Example `r = { (4), (1), (2), (5), (3), (6) }`

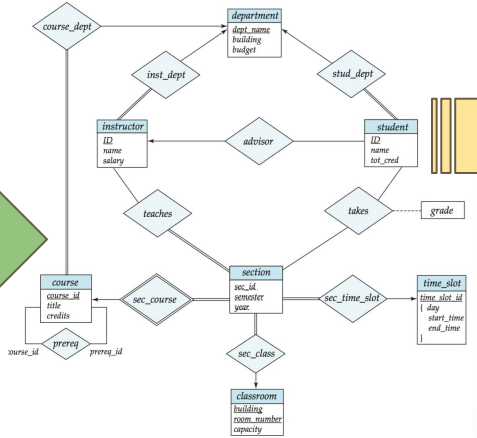
From ERD to Relations

- Conversion rules

Database design process

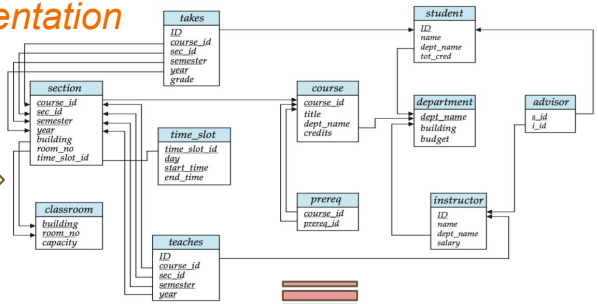
Database Idea

E-R Design

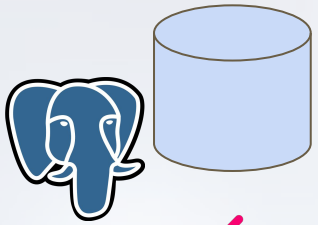


Relational Model

Preparing for implementation



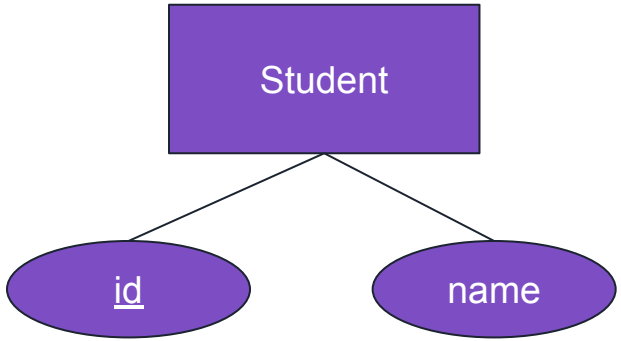
Database Engine



Connect with app

A first example

Student Entity



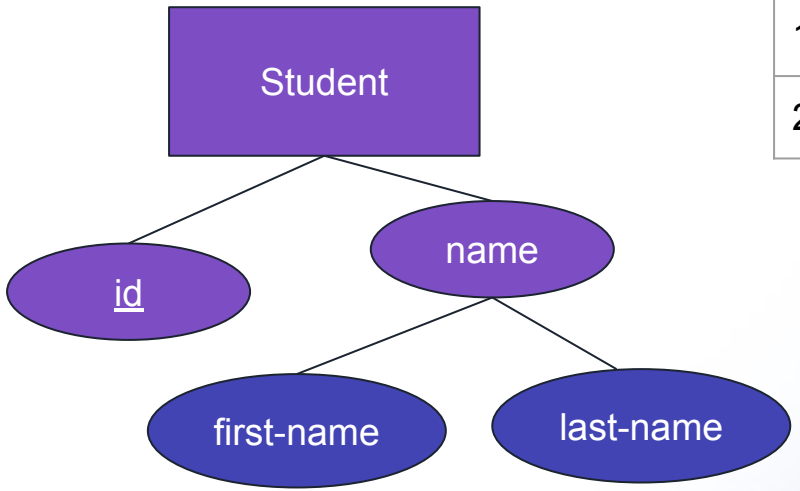
Student Table	
id	**name**
1	Harry
2	Ron

Student (id, name)

student = { (1, Harry), (2, Ron) }

Composite attributes

Student Entity



Student Table		
id	**first-name**	**last-name**
1	Harry	Potter
2	Ron	Weasley

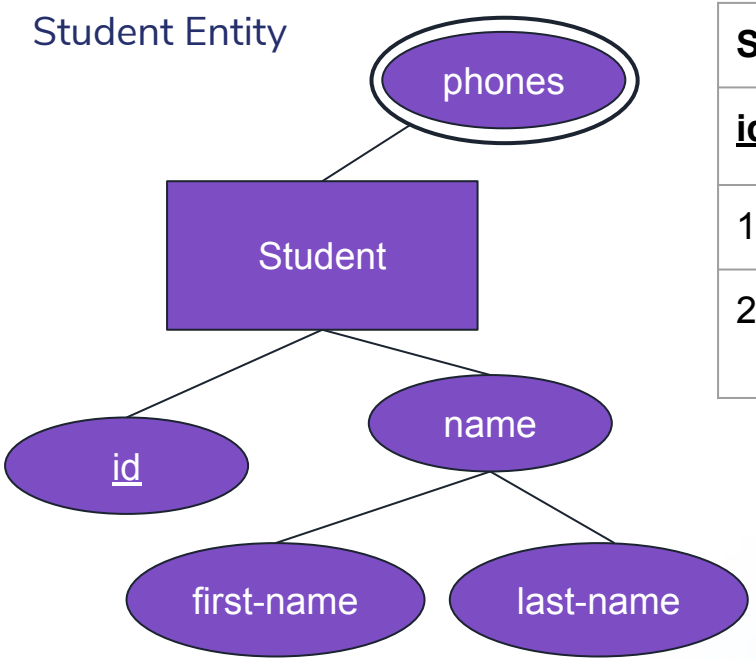
Student (id, first-name, last-name)

student = { (1, Harry, Potter), (2, Ron, Weasly) }

Multi-valued attributes

Wrong!

Student Entity



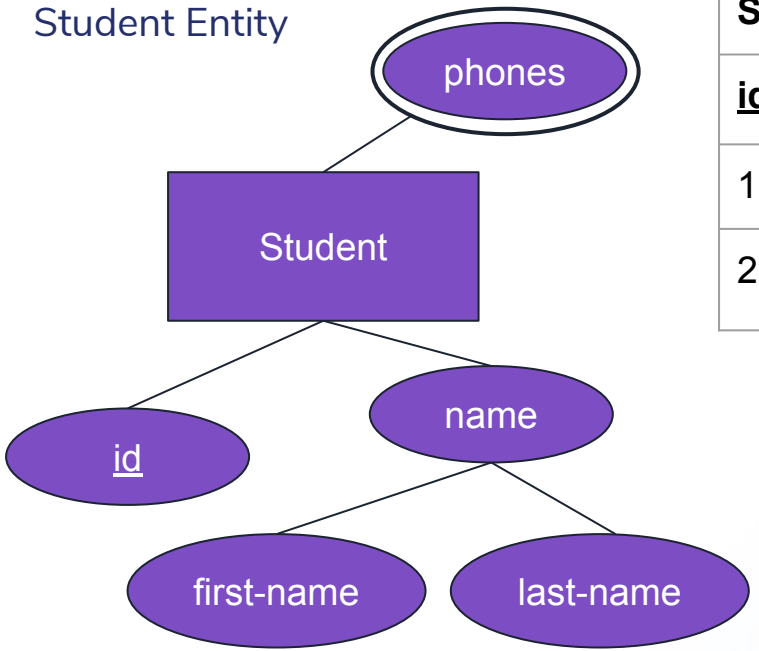
Student Table			
id	**first-name**	**last-name**	**phones**
1	Harry	Potter	093-105736, 055-105736
2	Ron	Weasley	091-123456

Student (id, first-name, last-name, phones)

student = {
(1, Harry, Potter, 093-105736, 055-105736),
(2, Ron, Weasly, 091-123456) }

Multi-valued attributes

Wrong!



Student Table				
id	**first-name**	**last-name**	**phone-1**	**phone-2**
1	Harry	Potter	093-105736	055-105736
2	Ron	Weasley	091-123456	null

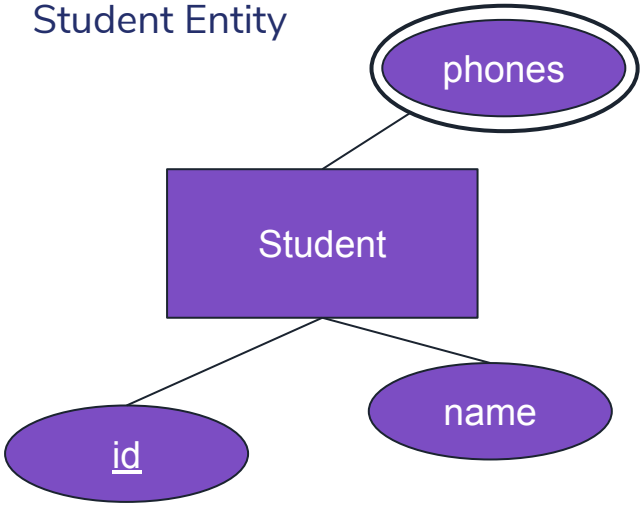
Student (id, first-name, last-name, phone-1, phone-2)

student = {
(1, Harry, Potter, 093-105736, 055-105736),
(2, Ron, Weasly, 091-123456, null) }

Multi-valued attributes

Correct

Student Entity



Student Table	
id	**name**
1	Harry Potter
2	Ron Weasley

Student (id, name)

student = {
(1, Harry Potter),
(2, Ron Weasly}

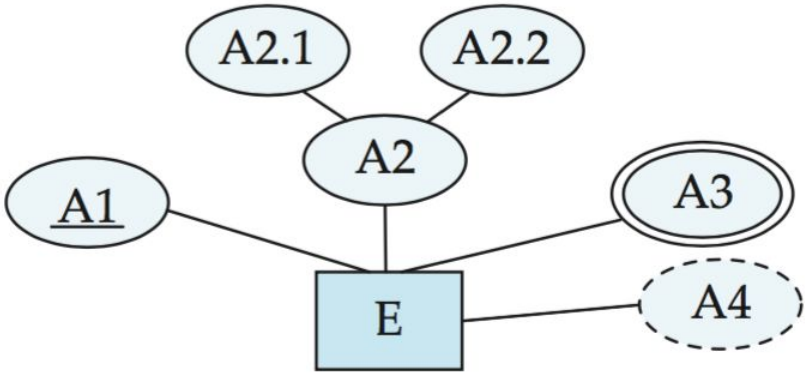
Phones Table	
student-id	**phone**
1	093-105736
1	055-105736
2	091-123456

Phone (student-id, value)

phones = {
(1, 093-105736),
(1, 055-105736), (2, 091-123456) }

Rule of thumb: Entities

- Attributes of entity become attributes of the relation
- Primary key is the same attribute (**A1**)
- Only child elements of composite attributes are added to the relation (**A2.1** and **A2.2**)
- If the entity has multivalued attributes, they are transformed into another relation with a foreign key to the original relation (**A3**)
- Derived attributes are omitted (**A4**)
  - because we can retrieve them via queries

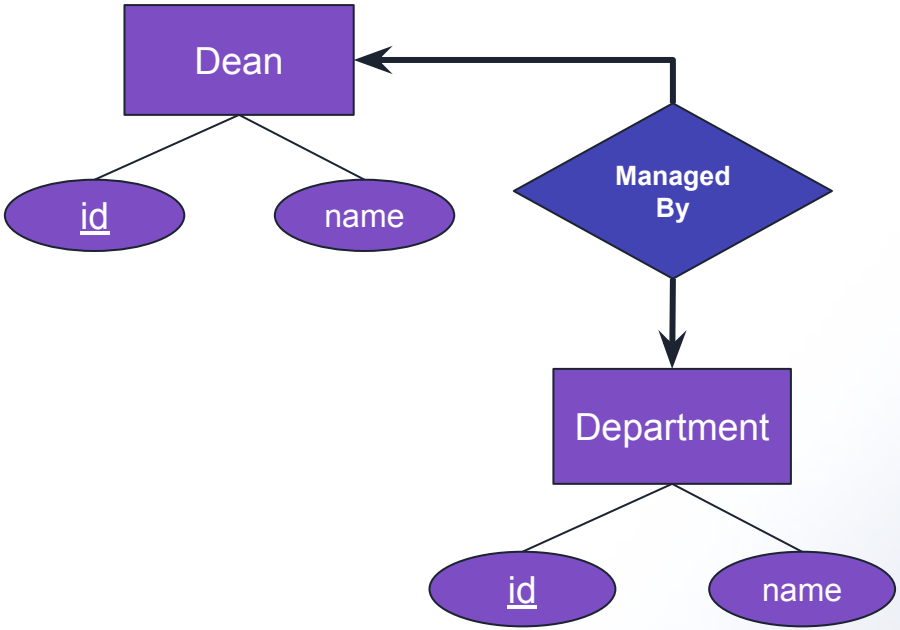


$$E = (\underline{A1}, A2.1, A2.2)$$

$$A3 = (A1, value)$$

1:1 relationship

Department *managed by a* **dean**

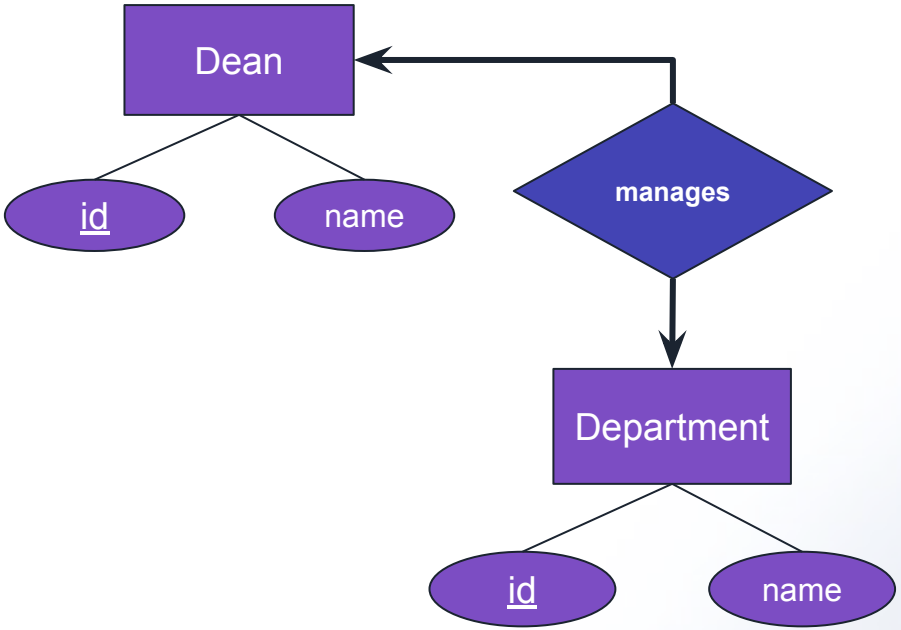


Dean	
dean-id	**name**
1	Harry
2	Ron

Department		
id	**name**	**dean-id**
1	History	1
2	Arts	2
3	Magic Politics	null

1:1 relationship

Dean *manages a* **department**

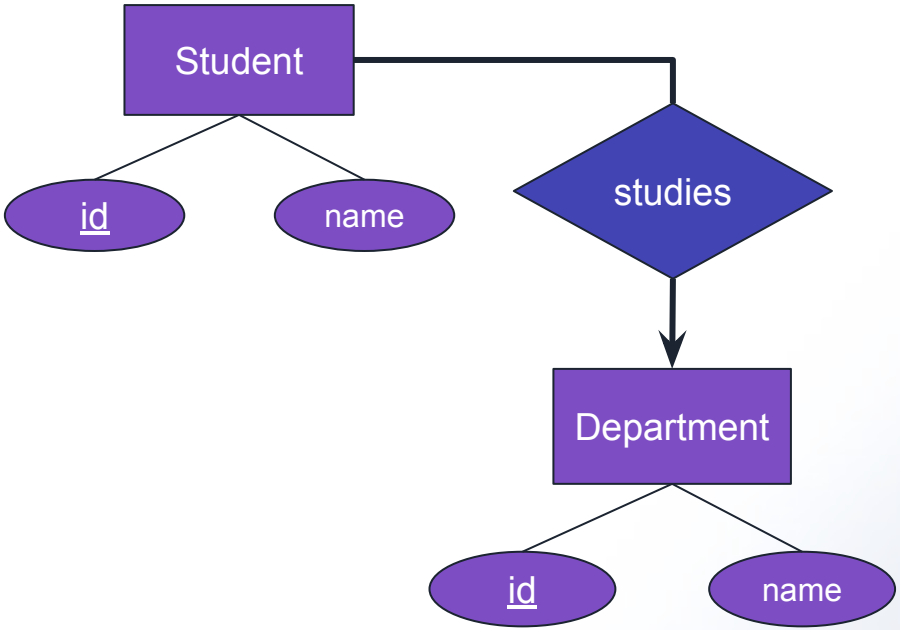


Dean		
id	**name**	**department-id**
1	Harry	1
2	Ron	2

Department	
department-id	**name**
1	History
2	Arts

1:N or N:1 relationship

Students *studies* in a **department**

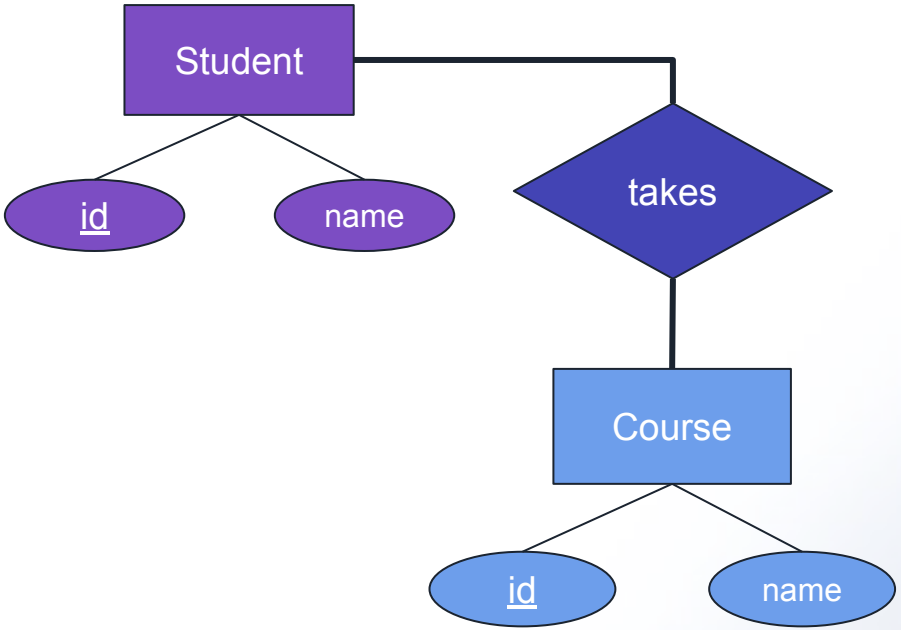


Student		
id	**name**	**department-id**
1	Harry	1
2	Ron	1
3	Ginny	2
4	Nevill	2

Department	
department-id	**name**
1	History
2	Arts

N:M relationship

Students *take* **courses**



Student (id, name) **Course (id, name)**

Takes (s-id, c-id)

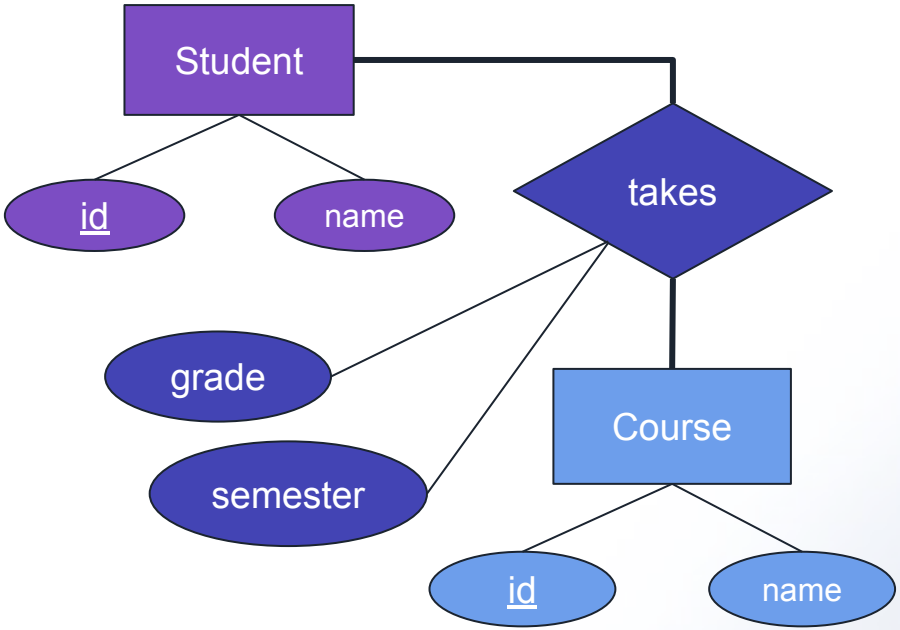
s-id	c-id
1	CS-101
2	CS-101
3	CS-101
1	CS-222

Relationship attributes

Student (id, name) Course (id, name)

Takes (s-id, c-id, grade, semester)

Students *take* **courses**



s-id	c-id	grade	semester
1	CS-101	A	Fall 2014
2	CS-101	A-	Spring 2015
3	CS-101	A-	Spring 2015
1	CS-222	B+	Fall 2016

Rule of thumb: Relationships

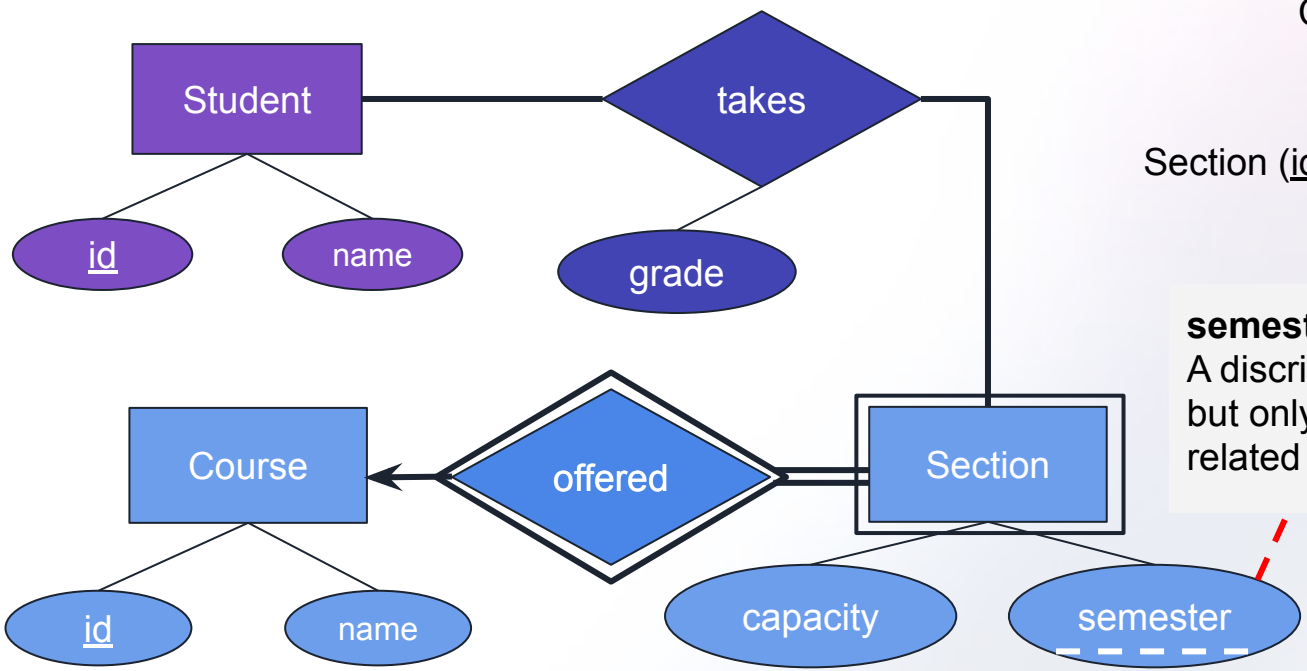
- For 1:1 relationship between E and F, the PK(E) can be added to F, or the PK(F) can be added to E
- For 1:N relationship from E to F (one instance of E with many instances of F), the PK(E) can be placed on F. On the “many” side
- For N:M relationship from E to F, a new relation R is added with primary key constructed from PK(E) and PK(F)
- When the relationship has attributes, they are also added to R

$$R = PK(E) \cup PK(F) \cup \{ A_1, A_2, \dots, A_n)$$
$$PK(R) = PK(E) \cup PK(F)$$

What if a student takes the same course again, next year?

Weak Entity

- Courses are recurring in some semesters: Course, Section



Course (id, name)

Section (id, semester, capacity)

semester is the discriminator
A discriminator is **like** a primary key, but only amongst the weak entities related to the strong entity.

Weak Entity

Students *take* **sections of a course**

PK(Student) = { id }

PK(Course) = { id }

PK(Section) = PK(Course) U { semester }
= { id, semester }

PK(Takes) = PK(Student) U PK(Section)
= { s-id, c-id, semester }

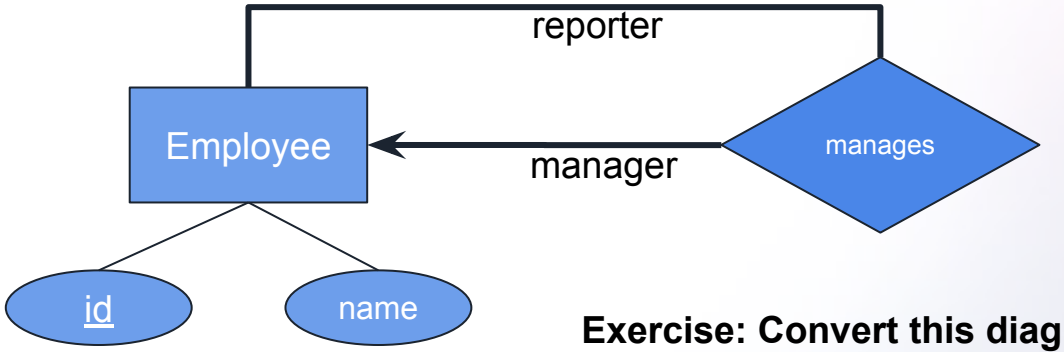
Table: Takes

s-id	**c-id**	**semester**	**grade**
1	CS-101	Fall 2014	F
1	CS-101	Spring 2015	A
3	CS-101	Spring 2015	A-
1	CS-222	Fall 2016	B+

*We renamed Student's id to **s-id** and Course's id to **c-id** to avoid repetition*

Unary relationships

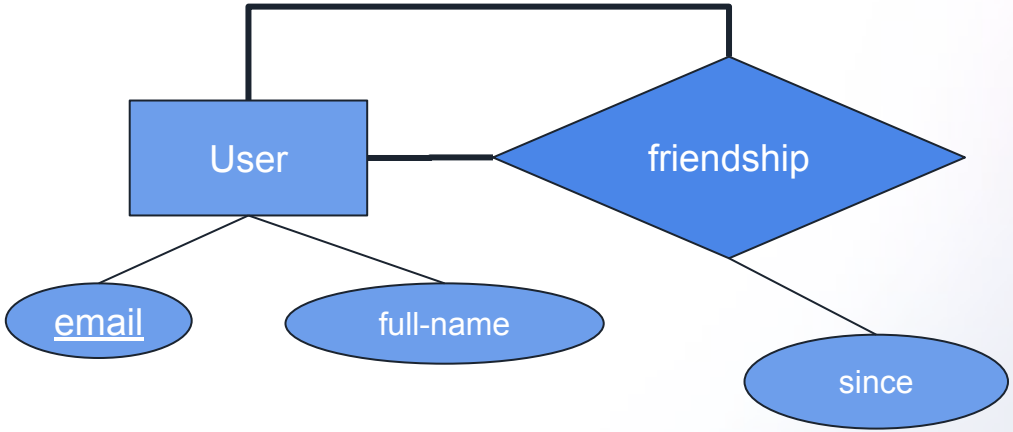
A manager is an employee, but manages other employees



Exercise: Convert this diagram to a relation

Unary relationships

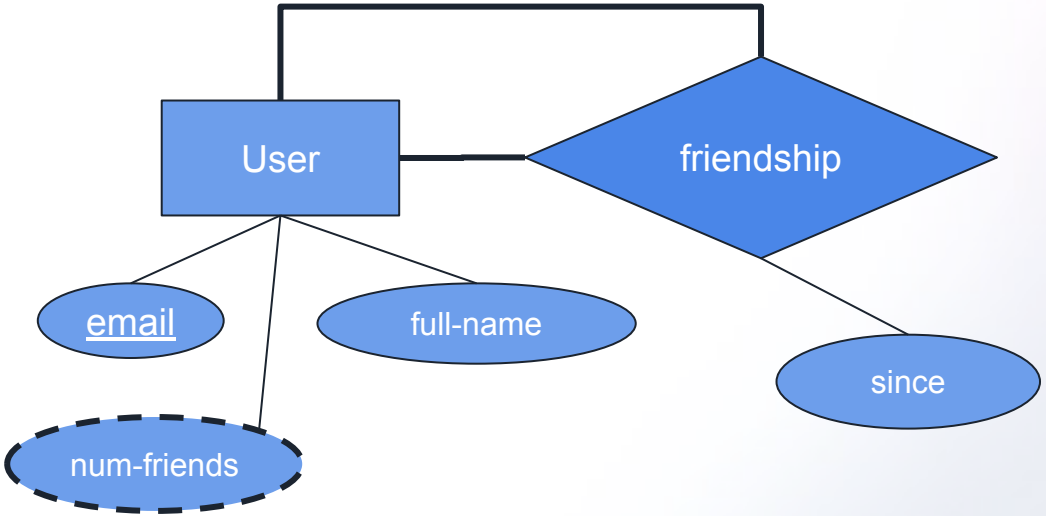
A Facebook user is friend with other facebook users



Exercise: Convert this diagram to a relation (or relations)

Derived attributes: # of friends on Facebook

num-friends is a derived attribute.



Question: Do we need to store the “num-friends” attribute?

What about:
– Performance
– Consistency

Homework 3

- Use the ERD you created in Homework 1 and the ERD-to-relations conversion rules to create the relational model of the football match.
- For each relation, identify the primary key as well as all the foreign keys