CS-300: Data-Intensive Systems

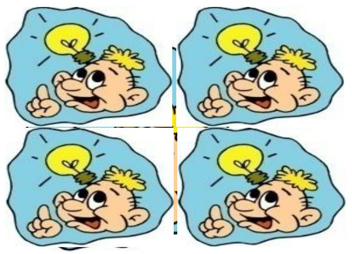
The Relational Model & Relational Algebra

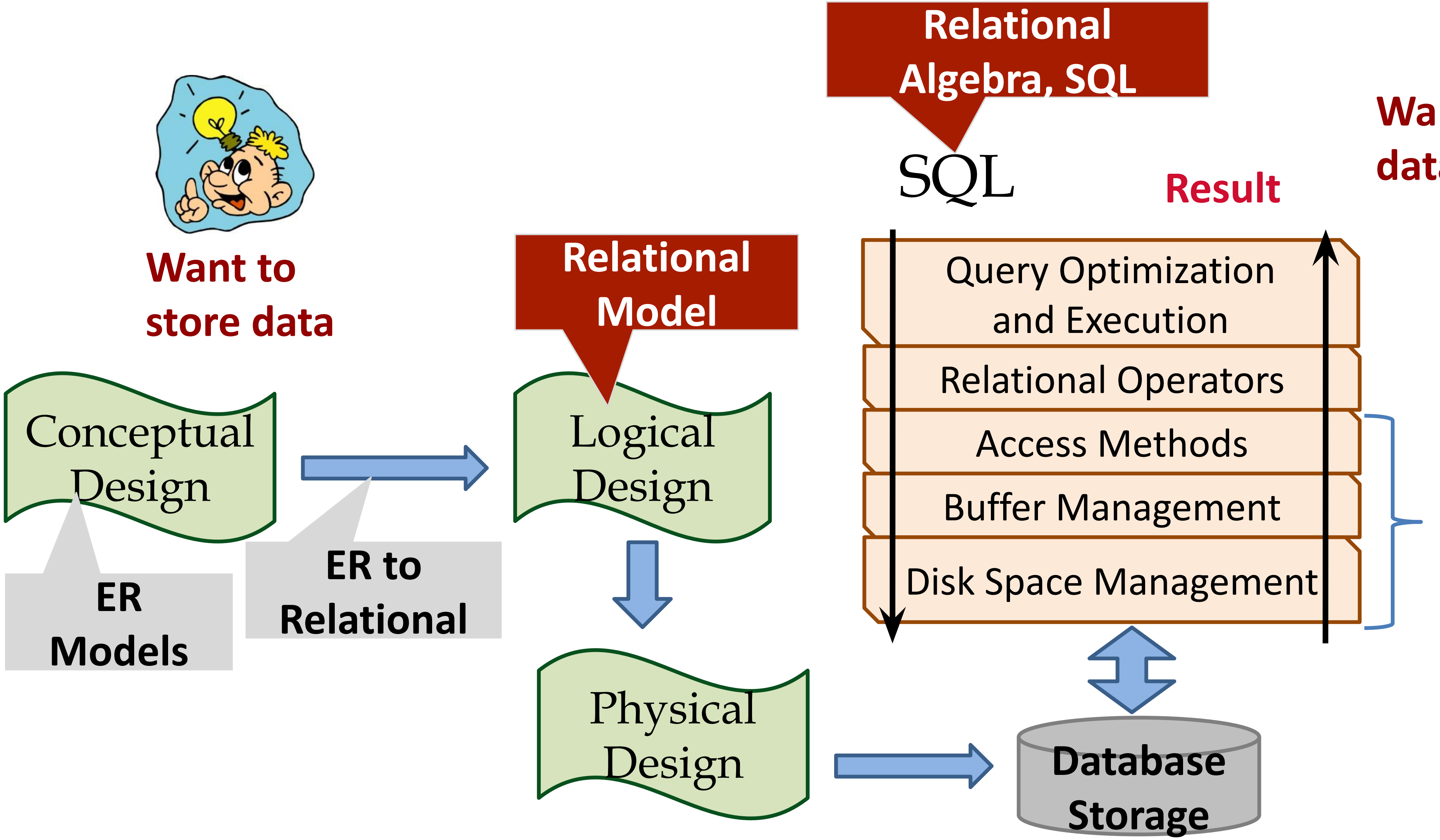
Prof. Anastasia Ailamaki, Prof. Sanidhya Kashyap

Dr. Antonio Boffa

DATA-INTENSIVE APPLICATIONS AND SYSTEMS

Simplified DBMS architecture





Outline

- Relational Model (Chapter 2)
  - Basics
  - SQL overview
  - Keys & Integrity Constraints
- Relational Algebra (Chapter 3.1-3.7)

Background about data models

- Data model: a collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints

- Edgar Codd’s seminal paper “A Relational Model of Data for Large Shared Data Banks” CACM 1970 □ Turing award

- Other models (1960, Legacy): Hierarchical, Network

- Other models (Recent):
  “NoSQL” (Key-value, Document)
  Array (Vector, Matrix, Tensor)

Why should we study the *Relational Model*?

- Relational: **Most widely used model**
  - IBM, Microsoft, Oracle, DuckDB, etc…
- Simple, yet expressive
- Great for use with a high-level query language
- Efficient implementations
- Object-oriented concepts have merged into the
  - Object-relational model: IBM DB2, Oracle 11i

Relational model: basics

- Database = set of named relations (or tables)
- Each relation has a set of named attributes (or columns)
- Each tuple (or row) has a value for each attribute
- Each attribute has a type (or domain)
  - integer, real, string, file formats (jpeg,…), enumerated and many more

Students

sid	name	login	age	gpa
50000	Dave	dave@cs	19	3.3
53666	Jones	jones@cs	18	3.4
53688	Smith	smit@ee	18	3.2
…	…	…	…	…

Colleges

name	location	strength
MIT	USA	10000
Oxford	UK	22000
EPFL	CH	9000
…	…	…

- Can think of a relation as a set of rows or tuples
  - i.e., all rows are distinct, no order among rows

Relational model: basics

- **Schema**: structural description of relations in database
  - Students(*sid*: string, *name*: string, *login*: string, *age*: integer, *gpa*: real)
- **Instance**: actual contents at a given point in time
  - Cardinality: # rows
  - Arity or degree: # attributes

Students

sid	name	login	age	gpa
50000	Dave	dave@cs	19	3.3
53666	Jones	jones@cs	18	3.4
53688	Smith	smit@ee	18	3.2
…	…	…	…	…

Colleges

name	location	strength
MIT	USA	10000
Oxford	UK	22000
EPFL	CH	9000
…	…	…

Relational model: basics

- What if a student does not have any grades yet, what is the value for GPA?
- **Null value**: special value for "unknown" or "undefined"

Students

sid	name	login	age	gpa
50000	Dave	dave@cs	19	3.3
53666	Jones	jones@cs	18	3.4
53688	Smith	smit@ee	18	NULL
…	…	…	…	…

Colleges

name	location	strength
MIT	USA	10000
Oxford	UK	22000
EPFL	CH	9000
…	…	…

Outline

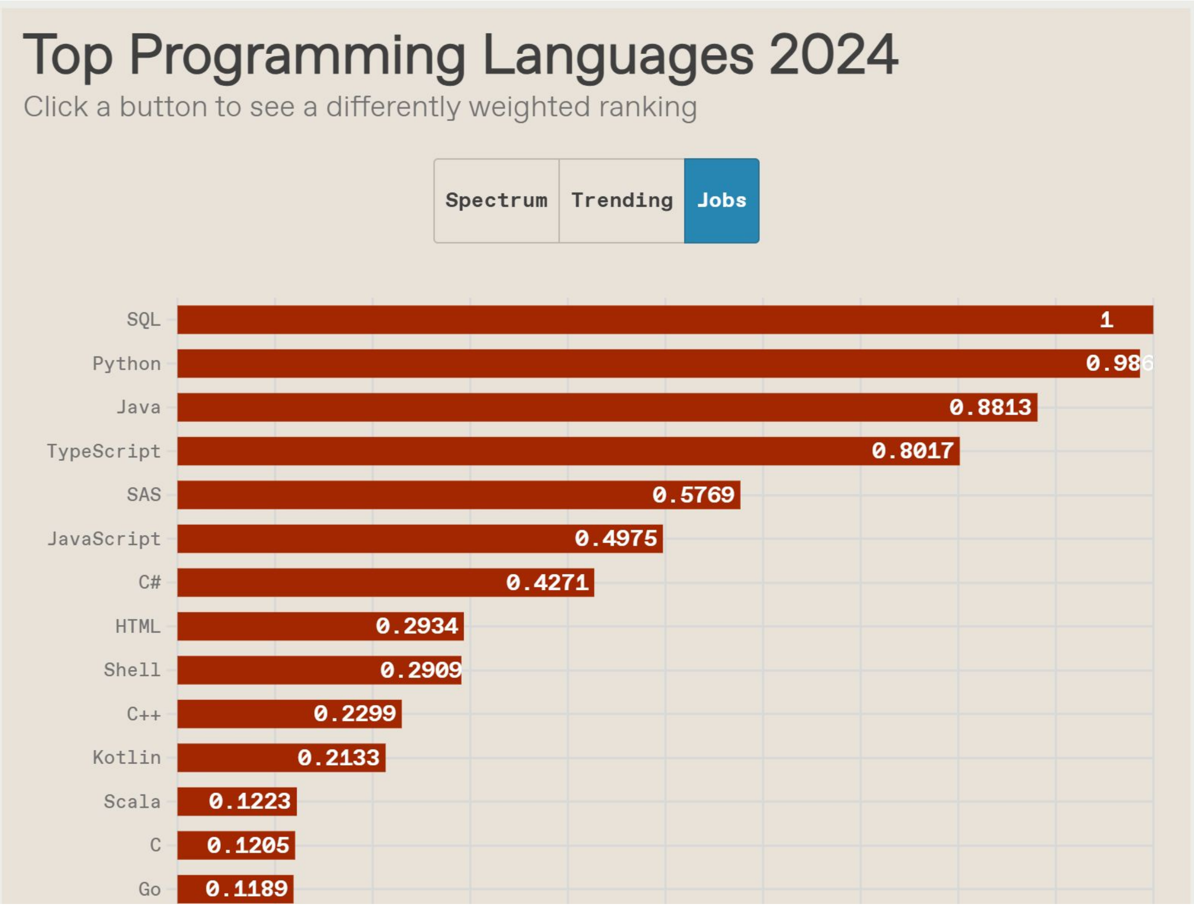
- Relational Model (Chapter 2)
  - Basics
  - SQL overview
  - Keys & Integrity Constraints
- Relational Algebra (Chapter 3.1-3.7)

SQL: A language for relational DBs

- SQL* (a.k.a. "Sequel"), standard language

- Data Definition Language (DDL)
  - Create, modify, delete relations
  - Specify constraints
  - Administer users, security, etc.

- Data Manipulation Language (DML)
  - Specify queries to find tuples that satisfy criteria
  - Add, modify, remove tuples



https://spectrum.ieee.org/top-programming-languages-2024

* Structured Query Language

SQL overview

- ```
  CREATE TABLE <name> ( <field> <domain>, … )
  ```

- ```
  INSERT INTO <name> (<field names>)
       VALUES (<field values>)
  ```

- ```
  DELETE FROM <name>
        WHERE <condition>
  ```

- ```
  UPDATE <name>
     SET <field name> = <value>
   WHERE <condition>
  ```

- ```
  SELECT <fields>
    FROM <name>
   WHERE <condition>
  ```
  ```

Creating relations in SQL

- Creates the ***Students*** *relation*
  - Note: the type (domain) of each field is specified and enforced by the DBMS whenever tuples are added or modified

```
CREATE TABLE Students
   (sid CHAR(20),
    name CHAR(20),
    login CHAR(10),
    age INTEGER,
    gpa FLOAT)
```

- Another example: the Enrolled table holds information about courses students take

```
CREATE TABLE Enrolled
   (sid CHAR(20),
    cid CHAR(20),
    grade CHAR(2))
```

Adding and deleting tuples

- Can insert a single tuple using:

```
INSERT INTO Students (sid, name, login, age, gpa)
 VALUES  ('53688', 'Smith', 'smith@cs', 18, 3.2)
```

- Can delete all tuples satisfying some condition (e.g., name = Smith):

```
DELETE
  FROM Students S
 WHERE S.name = 'Smith'
```

Powerful variants of these commands are available; more later!

Outline

- Relational Model (Chapter 2)
  - Basics
  - SQL overview
  - Keys & Integrity Constraints

- Relational Algebra

Relational models: ***Keys***

- Attribute whose value is unique in each tuple
- Or set of attributes whose combined values are unique
  - Identify tuples by its key
  - Special indexes on key attributes for efficiency
  - One relation referring to tuple of another relation: Foreign Key (more later)

Students

sid	name	login	age	gpa
50000	Dave	dave@cs	19	3.3
53666	Jones	jones@cs	18	3.4
53688	Smith	smit@ee	18	3.2
…	…	…	…	…

Colleges

name	location	strength
MIT	USA	10000
Oxford	UK	22000
EPFL	CH	9000
Oxford	USA	12000
…	….	…

- Integrity Constraint

Relational models: *Keys*

sid	name	login	age	gpa
50000	Dave	dave@cs	19	3.3
53666	Jones	jones@cs	18	3.4
53688	Smith	smit@ee	18	3.2
…	…	…	…	…

- Superkey
  - Set of attributes for which no two distinct tuples can have same values in all superkey fields
- Key
  - Set of attributes for which
    - It is a superkey
    - No subset of the fields is a superkey (minimal superkey)
- Candidate Keys
  - If there are multiple keys each of them is referred to as a candidate key
- Primary Key
  - One of the candidate keys is chosen (by DBA)

•sid, name
•sid, login
•sid, age
•sid, gpa
•sid, name, login
•sid, name, age
•sid, name, gpa
•sid, login, age
•sid, login, gpa
•sid, name, login, age
•…

•sid

Person (ssn, name, age, licence#)
•ssn
•licence#

Person (ssn, name, age, licence#)
•ssn ⬅
•licence#

DBA: Database administrator

Relational models: *Keys*

- Superkey
  - Set of attributes for which no two distinct tuples can have same values in all key fields
- Key
  - Set of attributes for which
    - It is a superkey
    - No subset of the fields is a superkey (minimal superkey)
- Candidate Keys
  - If there are multiple keys each of them is referred to as a candidate key
- Primary Key
  - One of the candidate keys is chosen (by DBA)

```
CREATE TABLE Students
    (sid CHAR(20),
     name CHAR(20),
     login CHAR(10),
     age INTEGER,
     gpa FLOAT,
       PRIMARY KEY(sid))
```

```
CREATE TABLE Person
    (ssn CHAR(9),
     name CHAR(20),
     licence# CHAR(10),
       PRIMARY KEY(ssn),
       UNIQUE(licence#))
```

Primary and candidate keys in SQL

- Possibly many **candidate keys** (specified using UNIQUE), one of which is chosen as the primary key.
- Keys must be used carefully!
  - E.g., “For a given student and course, there is a single grade.”

```
CREATE TABLE Enrolled
  (sid CHAR(20)
   cid  CHAR(20),
   grade CHAR(2),
   PRIMARY KEY (sid,cid))
```

vs.

```
CREATE TABLE Enrolled
   (sid CHAR(20)
    cid  CHAR(20),
    grade CHAR(2),
    PRIMARY KEY  (sid),
    UNIQUE (cid, grade))
```

  - “Students can take only one course, and no two students in a course receive the same grade”

Relational model: Foreign keys

- Set of fields in one relation that is used to `refer’ to a tuple in another relation.
  - Must correspond to the primary key of the other relation
  - Like a ‘*logical pointer*’
- If all foreign key constraints are enforced: achieves **referential integrity** (i.e., no dangling references)

Students

sid	name	login	age	gpa
50000	Dave	dave@cs	19	3.3
53666	Jones	jones@cs	18	3.4
53688	Smith	smit@ee	18	3.2
…	…	…	…	…

Enrolled

cid	sid	grade
Carnatic101	53666	C
Raggae203	50000	B
Topology112	53666	A
…	…	…

Enforcing referential integrity

- Consider Students and Enrolled: sid in Enrolled is a foreign key that references Students
- What should the DBMS do if we insert an Enrolled tuple with a non-existent student id? (Reject it!)
- What should the DBMS do if a Students tuple is deleted?
  - Also delete all Enrolled tuples that refer to it?
  - Disallow deletion of a Students tuple that is referred to?
  - Set sid in Enrolled tuples that refer to it to a default sid?
  - (In SQL, also: Set sid in Enrolled tuples that refer to it to a special value null, denoting *`unknown'* or *`inapplicable'*)
- Similar issues arise if we update primary key of Students tuple

Integrity constraints (IC)

- **IC**: condition that must be true for any instance of the database; e.g., domain constraints
  - ICs are specified when schema is defined
  - ICs are checked when relations are modified
- **A legal instance of a relation is one that satisfies all specified ICs**
  - DBMS should not allow illegal instances
- If the DBMS checks ICs, stored data is more faithful to real-world meaning
  - Avoids data entry errors, too!

Relational model: Summary

- A tabular representation of data
- Simple and intuitive, currently the most widely used
  - Object-relational variant gaining ground
- Integrity constraints can be specified by the DBA, based on application semantics DBMS checks for violations
  - Two important ICs: primary and foreign keys
  - In addition, we always have domain constraints
- Mapping from ER to Relational is (fairly) straightforward

Outline

- Relational Model
- Relational Algebra (Chapter 3.1-3.7)
  - Relational Query Languages
  - Selection & Projection
  - Union, Set Difference & Intersection
  - Cross product & Joins
  - Intro to query optimization
  - Division

Relational query languages

- **Query languages***:* Allow manipulation and **retrieval of data** from a database

- Relational model supports simple, powerful QLs:
  - Strong formal foundation based on logic
  - Allows for much optimization

- Query Languages **!=** Programming Languages!
  - QLs not expected to be “Turing complete”
  - QLs not intended to be used for complex calculations
  - QLs support easy, efficient access to large data sets

Formal relational query languages

Two mathematical Query Languages form the basis for "real" languages (e.g. SQL), and for implementation:

- ***Relational Algebra***: More operational, very useful for representing execution plans
- **Relational Calculus:** Lets users describe what they want, rather than how to compute it
  - Non-procedural, *declarative*

Understanding Algebra & Calculus is key to understanding SQL, query processing!

Importance of relational algebra

- Relational algebra is a simple language
  - 5 operators/language primitives
- Yet captures many queries
- **Codd's Theorem: relational calculus = relational algebra**
  - For every query in relational calculus, there is an equivalent query in relational algebra, and vice versa

- Relational algebra is an imperative language, yet still close to declarative languages like relational calculus and SQL
- Useful as internal representation of queries inside database engines
  - Used as intermediate representation for query optimization

Preliminaries

- A query is applied to ***relation instances***, and the result of a query is also a relation instance
  - ***Schemas* of input** relations for a query are fixed (but query will run over any legal instance)
  - The **schema for the *result*** of a given query is also **fixed**
    - Determined by the definitions of the query language constructs

- Positional vs. named-field notation:
  - Positional notation easier for formal definitions; named-field notation is more readable
  - Both used in SQL

Example schema and instances

classroom(building, room_number, capacity)
department(dept_name, building, budget)
course(course_id, title, dept_name, credits)
instructor(ID, name, dept_name, salary)
section(course_id, sec_id, semester, year, building, room_number, time_slot_id)
teaches(ID, course_id, sec_id, semester, year)
student(ID, name, dept_name, tot_cred)
takes(ID, course_id, sec_id, semester, year, grade)
advisor(s_ID, i_ID)
time_slot(time_slot_id, day, start_time, end_time)
prereq(course_id, prereq_id)

Figure 2.8 Schema of the university database.

instructor

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

course

course_id	*title*	*dept_name*	*credits*
BIO-101	Intro. to Biology	Biology	4
BIO-301	Genetics	Biology	4
BIO-399	Computational Biology	Biology	3
CS-101	Intro. to Computer Science	Comp. Sci.	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3
CS-319	Image Processing	Comp. Sci.	3
CS-347	Database System Concepts	Comp. Sci.	3
EE-181	Intro. to Digital Systems	Elec. Eng.	3
FIN-201	Investment Banking	Finance	3
HIS-351	World History	History	3
MU-199	Music Video Production	Music	3
PHY-101	Physical Principles	Physics	4

teaches

ID	*course_id*	*sec_id*	*semester*	*year*
10101	CS-101	1	Fall	2017
10101	CS-315	1	Spring	2018
10101	CS-347	1	Fall	2017
12121	FIN-201	1	Spring	2018
15151	MU-199	1	Spring	2018
22222	PHY-101	1	Fall	2017
32343	HIS-351	1	Spring	2018
45565	CS-101	1	Spring	2018
45565	CS-319	1	Spring	2018
76766	BIO-101	1	Summer	2017
76766	BIO-301	1	Summer	2018
83821	CS-190	1	Spring	2017
83821	CS-190	2	Spring	2017
83821	CS-319	2	Spring	2018
98345	EE-181	1	Spring	2017

Simplest relational algebra expression

- The name of the relation, without any operator
- No operator is applied

```
SELECT *
FROM instructor
```

instructor

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

instructor →

Output

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

Relational algebra: Five basic operations

1. **Selection (σ):** Selects a subset of **rows** from relation (horizontal)
2. **Project (π):** Retains only wanted **columns** from relation (vertical)
3. **Cross-product(×):** Allows us to combine two relations
4. **Set-difference (–):** Tuples in R, but not in S
5. **Union(∪):** Tuples in R and/or in S

Unary operators (1–2)

Binary operators (3–5)

Since each operation retains a relation, **operations can be composed!**

Outline

- Relational Model
- Relational Algebra (Chapter 3.1-3.7)
  - Relational Query Languages
  - Selection & Projection
  - Union, Set Difference & Intersection
  - Cross product & Joins
  - Intro to query optimization
  - Division

Selection operator (σ) – Examples 1

- Selects rows that satisfy *selection condition*
- ***Output schema*** of result is same as that of the input relation

instructor

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~~

$$\sigma_{dept\_name = \text{“Physics”}}(instructor)$$

```
SELECT *
FROM instructor
WHERE dept_name = “Physics”
```

Output

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
33456	Gold	Physics	87000

Selection operator (σ) – Examples 2

- Selects rows that satisfy *selection condition*
- ***Output schema*** of result is same as that of the input relation

`instructor`

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~~

$$\sigma_{dept\_name = \text{“Physics”} \wedge salary > 90000}(instructor)$$

```
SELECT *
FROM instructor
WHERE dept_name = “Physics”
    AND
    salary > 90000
```

Output

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000

Projection operator (π)

- Retains only attributes that are in the *projection list*
- **Output schema** is exactly the fields in the projection list, with the same names that they had in the input relation

instructor

	ID	*name*	*dept_name*	*salary*
0	22222	Einstein	Physics	95000
1	12121	Wu	Finance	90000
2	32343	El Said	History	60000
3	45565	Katz	Comp. Sci.	75000
4	98345	Kim	Elec. Eng.	80000
5	76766	Crick	Biology	72000
6	10101	Srinivasan	Comp. Sci.	65000
7	58583	Califieri	History	62000
8	83821	Brandt	Comp. Sci.	92000
9	15151	Mozart	Music	40000
10	33456	Gold	Physics	87000
11	76543	Singh	Finance	80000

$$\Pi_{ID,\ name,\ salary}(instructor)$$

```
SELECT ID, name, salary
FROM instructor
```

Output

	ID	*name*	*salary*
0	10101	Srinivasan	65000
1	12121	Wu	90000
2	15151	Mozart	40000
3	22222	Einstein	95000
4	32343	El Said	60000
5	33456	Gold	87000
6	45565	Katz	75000
7	58583	Califieri	62000
8	76543	Singh	80000
9	76766	Crick	72000
10	83821	Brandt	92000
11	98345	Kim	80000

Projection operator (π)

- Projection operator has to *eliminate duplicates*
- *Relation □ SET of tuples, dept_name contains duplicate*
  - Why remove them?
- Set semantics and multiset ("bag") semantics (like SQL)

	ID	name	dept_name	salary
0	22222	Einstein	Physics	95000
1	12121	Wu	Finance	90000
2	32343	El Said	History	60000
3	45565	Katz	Comp. Sci.	75000
4	98345	Kim	Elec. Eng.	80000
5	76766	Crick	Biology	72000
6	10101	Srinivasan	Comp. Sci.	65000
7	58583	Califieri	History	62000
8	83821	Brandt	Comp. Sci.	92000
9	15151	Mozart	Music	40000
10	33456	Gold	Physics	87000
11	76543	Singh	Finance	80000

$\Pi_{dept\_name}(instructor)$

```
SELECT DISTINCT dept_name
FROM instructor
```

Output

	dept_name
0	Comp. Sci.
1	Finance
2	Music
3	Physics
4	History
5	Biology
6	Elec. Eng.

Why 7 rows and not 12???

```
SELECT dept_name
FROM instructor
```

	dept_name
0	Comp. Sci.
1	Finance
2	Music
3	Physics
4	History
5	Physics
6	Comp. Sci.
7	History
8	Finance
9	Biology
10	Comp. Sci.
11	Elec. Eng.

Composing multiple operators

- Output of one operator can become input to another operator

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

$$\Pi_{name}\ (\sigma_{dept\_name\,=\,\text{“Physics”}}\ (instructor))$$

```
SELECT name
FROM instructor
WHERE dept_name = “Physics”
```

Output

name
Einstein
Gold

Outline

- Relational Model
- Relational Algebra
  - Relational Query Languages
  - Selection & Projection
  - Union, Set Difference & Intersection
  - Cross product & Joins
  - Intro to query optimization
  - Division

Rename operator (ρ)

- Renames the list of attributes specified in the form of oldname → newname or position → newname
- Can also be used to rename the name of the output relation
- **Output schema** is same as input except for the renamed attributes
- Returns same tuples as input

```
SELECT i.name
FROM instructor AS i
WHERE i.ID = …
```

instructor

ID	name	dept_name	salary
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000

$\rho_i(instructor)$ →

Output

i

ID	name	dept_name	salary
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000

$\rho_{ID \rightarrow instructor.ID}(instructor)$ →

Output

instructor

instructor.ID	name	dept_name	salary
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000

Union operator (∪)

- All these operations take two input relations, which must be *union-compatible*:
  - Same number of fields
  - "Corresponding" fields have the same type
- Is duplicate elimination required?

```
(SELECT *  FROM c1)
      UNION
(SELECT *  FROM c2)
```

c1 =

course_id
CS-101
CS-347
PHY-101

c2 =

course_id
CS-101
CS-315
CS-319
FIN-201
HIS-351
MU-199

c1 ∪ *c2* →

Output

course_id
CS-101
CS-315
CS-319
CS-347
FIN-201
HIS-351
MU-199
PHY-101

Set-difference operator (–)

- Two input relations, which must be *union-compatible*
- Set difference is not commutative

```
(SELECT *  FROM c1)
      EXCEPT
(SELECT *  FROM c2)
```

c1 =

course_id
CS-101
CS-347
PHY-101

c2 =

course_id
CS-101
CS-315
CS-319
FIN-201
HIS-351
MU-199

c1 – *c2*

Output

course_id
CS-347
PHY-101

c2 – *c1*

Output

course_id
CS-315
CS-319
FIN-201
HIS-351
MU-199

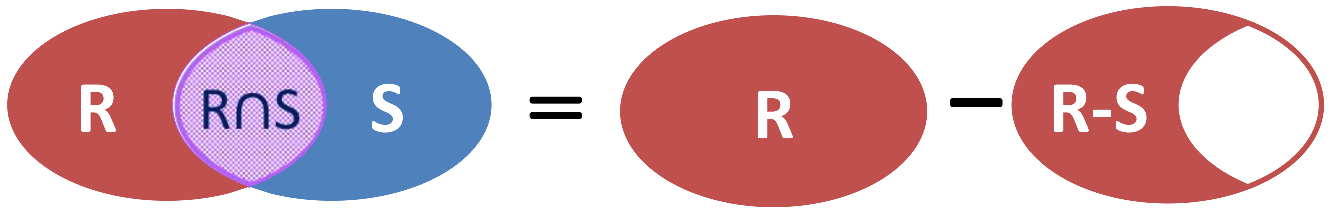
Compound operator: Intersection

- Alongside the **five** basic operators, there are several additional **Compound operators**
  - These add no computational power to the language, but are useful shorthands
  - Can be expressed solely with the basic operations

- Intersection takes two input relations, which must be union-compatible

Q: How to express it using basic operators?

$$R \cap S = R - (R - S)$$



Intersection operator (∩)

```
(SELECT *  FROM c1)
      INTERSECT
(SELECT *  FROM c2)
```

c1 =

course_id
CS-101
CS-347
PHY-101

c2 =

course_id
CS-101
CS-315
CS-319
FIN-201
HIS-351
MU-199

$c1 \cap c2$

$c1 - (c1 - c2)$

Output

course_id
CS-101

Outline

- Relational Model
- Relational Algebra (Chapter 3.1-3.7)
    - Relational Query Languages
    - Selection & Projection
    - Union, Set Difference & Intersection
    - Cross product & Joins
    - Intro to query optimization
    - Division

Cross-product operator (X)

- S x R: Each row of S paired with each row of R

Q: How many rows in the result?

- **Result schema** has one field per field of S and R, with field names “inherited” if possible
  - *May have a naming conflict*: Both S and R have a field with the same name
  - In this case, can use the *renaming operator (ρ)*

Cross-product example

Output

Rename operator (ρ) applied, not shown in the formula

instructor

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

teaches

ID	*course_id*	*sec_id*	*semester*	*year*
10101	CS-101	1	Fall	2017
10101	CS-315	1	Spring	2018
10101	CS-347	1	Fall	2017
12121	FIN-201	1	Spring	2018
15151	MU-199	1	Spring	2018
22222	PHY-101	1	Fall	2017
32343	HIS-351	1	Spring	2018
45565	CS-101	1	Spring	2018
45565	CS-319	1	Spring	2018
76766	BIO-101	1	Summer	2017
76766	BIO-301	1	Summer	2018
83821	CS-190	1	Spring	2017
83821	CS-190	2	Spring	2017
83821	CS-319	2	Spring	2018
98345	EE-181	1	Spring	2017

$instructor \times teaches$

instructor.ID	*name*	*dept_name*	*salary*	*teaches.ID*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2017
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2017
10101	Srinivasan	Comp. Sci.	65000	12121	FIN-201	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	15151	MU-199	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	22222	PHY-101	1	Fall	2017
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
12121	Wu	Finance	90000	10101	CS-101	1	Fall	2017
12121	Wu	Finance	90000	10101	CS-315	1	Spring	2018
12121	Wu	Finance	90000	10101	CS-347	1	Fall	2017
12121	Wu	Finance	90000	12121	FIN-201	1	Spring	2018
12121	Wu	Finance	90000	15151	MU-199	1	Spring	2018
12121	Wu	Finance	90000	22222	PHY-101	1	Fall	2017
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
15151	Mozart	Music	40000	10101	CS-101	1	Fall	2017
15151	Mozart	Music	40000	10101	CS-315	1	Spring	2018
15151	Mozart	Music	40000	10101	CS-347	1	Fall	2017
15151	Mozart	Music	40000	12121	FIN-201	1	Spring	2018
15151	Mozart	Music	40000	15151	MU-199	1	Spring	2018
15151	Mozart	Music	40000	22222	PHY-101	1	Fall	2017
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
22222	Einstein	Physics	95000	10101	CS-101	1	Fall	2017
22222	Einstein	Physics	95000	10101	CS-315	1	Spring	2018
22222	Einstein	Physics	95000	10101	CS-347	1	Fall	2017
22222	Einstein	Physics	95000	12121	FIN-201	1	Spring	2018
22222	Einstein	Physics	95000	15151	MU-199	1	Spring	2018
22222	Einstein	Physics	95000	22222	PHY-101	1	Fall	2017
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

Compound operator: Join ⋈

- The Cartesian-Product $instructor \times teaches$ associates every tuple of instructor with every tuple of teaches

- Most of the resulting rows have information about instructors who did NOT teach a particular course.

- To get only those tuples of instructors and the courses that they taught

$$\sigma_{instructor.id = teaches.id}\,(instructor \times teaches)$$

- JOIN OPERATOR:

$$instructor \bowtie_{Instructor.id = teaches.id} teaches$$

$$instructor \bowtie teaches$$

If the columns have the same name, condition can be omitted (NATURAL JOIN)

Join example

instructor

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

```
SELECT *
FROM instructor JOIN teaches
WHERE
instructor.ID = teaches.ID
```

Output

instructor.ID	*name*	*dept_name*	*salary*	*teaches.ID*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2017
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2017
~~10101~~	~~Srinivasan~~	~~Comp. Sci.~~	~~65000~~	~~12121~~	~~FIN-201~~	~~1~~	~~Spring~~	~~2018~~
~~10101~~	~~Srinivasan~~	~~Comp. Sci.~~	~~65000~~	~~15151~~	~~MU-199~~	~~1~~	~~Spring~~	~~2018~~
~~10101~~	~~Srinivasan~~	~~Comp. Sci.~~	~~65000~~	~~22222~~	~~PHY-101~~	~~1~~	~~Fall~~	~~2017~~
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
~~12121~~	~~Wu~~	~~Finance~~	~~90000~~	~~10101~~	~~CS-101~~	~~1~~	~~Fall~~	~~2017~~
~~12121~~	~~Wu~~	~~Finance~~	~~90000~~	~~10101~~	~~CS-315~~	~~1~~	~~Spring~~	~~2018~~
~~12121~~	~~Wu~~	~~Finance~~	~~90000~~	~~10101~~	~~CS-347~~	~~1~~	~~Fall~~	~~2017~~
12121	Wu	Finance	90000	12121	FIN-201	1	Spring	2018
~~12121~~	~~Wu~~	~~Finance~~	~~90000~~	~~15151~~	~~MU-199~~	~~1~~	~~Spring~~	~~2018~~
~~12121~~	~~Wu~~	~~Finance~~	~~90000~~	~~22222~~	~~PHY-101~~	~~1~~	~~Fall~~	~~2017~~
...	...	...	...	...	...	...	...	...

teaches

ID	*course_id*	*sec_id*	*semester*	*year*
10101	CS-101	1	Fall	2017
10101	CS-315	1	Spring	2018
10101	CS-347	1	Fall	2017
12121	FIN-201	1	Spring	2018
15151	MU-199	1	Spring	2018
22222	PHY-101	1	Fall	2017
32343	HIS-351	1	Spring	2018
45565	CS-101	1	Spring	2018
45565	CS-319	1	Spring	2018
76766	BIO-101	1	Summer	2017
76766	BIO-301	1	Summer	2018
83821	CS-190	1	Spring	2017
83821	CS-190	2	Spring	2017
83821	CS-319	2	Spring	2018
98345	EE-181	1	Spring	2017

$$instructor \bowtie teaches$$

$$\sigma_{instructor.id\ =\ teaches.id}\ (instructor\ \times\ teaches\ )$$

Outline

- Relational Model
- Relational Algebra (Chapter 3.1-3.7)
  - Relational Query Languages
  - Selection & Projection
  - Union, Set Difference & Intersection
  - Cross product & Joins
  - Intro to query optimization
  - Division

Complex queries

"Find the names of all instructors in the Music department together with the course title of all the courses that the instructors teach."

instructor

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

course

course_id	title	dept_name	credits
BIO-101	Intro. to Biology	Biology	4
BIO-301	Genetics	Biology	4
BIO-399	Computational Biology	Biology	3
CS-101	Intro. to Computer Science	Comp. Sci.	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3
CS-319	Image Processing	Comp. Sci.	3
CS-347	Database System Concepts	Comp. Sci.	3
EE-181	Intro. to Digital Systems	Elec. Eng.	3
FIN-201	Investment Banking	Finance	3
HIS-351	World History	History	3
MU-199	Music Video Production	Music	3
PHY-101	Physical Principles	Physics	4

teaches

ID	course_id	sec_id	semester	year
10101	CS-101	1	Fall	2017
10101	CS-315	1	Spring	2018
10101	CS-347	1	Fall	2017
12121	FIN-201	1	Spring	2018
15151	MU-199	1	Spring	2018
22222	PHY-101	1	Fall	2017
32343	HIS-351	1	Spring	2018
45565	CS-101	1	Spring	2018
45565	CS-319	1	Spring	2018
76766	BIO-101	1	Summer	2017
76766	BIO-301	1	Summer	2018
83821	CS-190	1	Spring	2017
83821	CS-190	2	Spring	2017
83821	CS-319	2	Spring	2018
98345	EE-181	1	Spring	2017

1. $\Pi_{name,title} (\sigma_{dept\_name = \text{"Music"}} (instructor \bowtie (teaches \bowtie \Pi_{course\_id,title}(course))))$

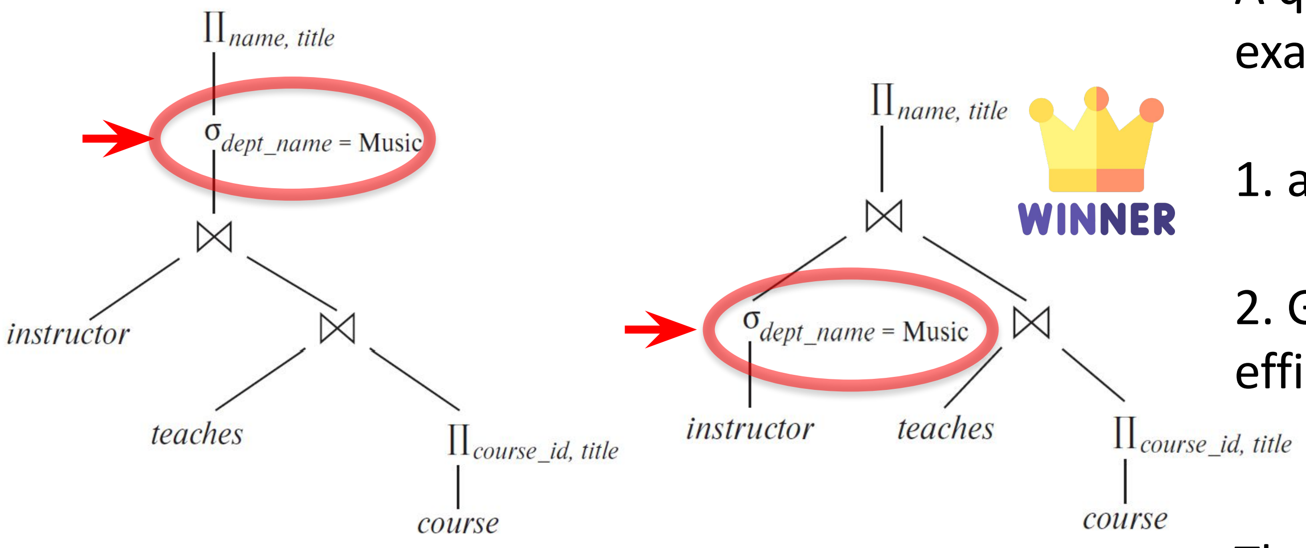
2. $\Pi_{name,title} ((\sigma_{dept\_name = \text{"Music"}} (instructor)) \bowtie (teaches \bowtie \Pi_{course\_id,title}(course)))$

Equivalent in relational algebra! But…

Intro to query optimization

1. $\Pi_{name,title}\,(\sigma_{dept\_name = \text{"Music"}}\,(instructor \bowtie (teaches \bowtie \Pi_{course\_id,title}(course))))$

2. $\Pi_{name,title}\,((\sigma_{dept\_name = \text{"Music"}}\,(instructor)) \bowtie (teaches \bowtie \Pi_{course\_id,title}(course)))$



(a) Initial expression tree

(b) Transformed expression tree

A query engine applying these operators following the exact order of the expression

1. and 2. will have very different performance!

2. Generates way fewer intermediate results, so more efficient! (Due to selection pushdown)

The **Query Optimizer** select the best way to execute a query! (Next lessons)

Outline

- Relational Model
- Relational Algebra
  - Relational Query Languages
  - Selection & Projection
  - Union, Set Difference & Intersection
  - Cross product & Joins
  - Intro to query optimization
  - Division

<u>Last</u> compound operator: Division

- Useful for expressing "for all" queries like:

  Find the instructors teaching courses FOR ALL number of credits

For A/B attributes of B are subset of attributes of A.

  - May need to "project" to make this happen.

- **Example**: Let *A* have 2 fields, *x and y*; let *B* have only field *y*

$$A/B = \{\langle x\rangle \mid \forall \langle y\rangle \in B(\exists \langle x, y\rangle \in A)\}$$

_A/B_ contains all _x_ tuples such that

for <u>_every_</u> _y_ tuple in _B_, there is an _xy_ tuple in _A_

course

course_id	*title*	*dept_name*	*credits*
BIO-101	Intro. to Biology	Biology	4
BIO-301	Genetics	Biology	4
BIO-399	Computational Biology	Biology	3
CS-101	Intro. to Computer Science	Comp. Sci.	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3
CS-319	Image Processing	Comp. Sci.	3
CS-347	Database System Concepts	Comp. Sci.	3
EE-181	Intro. to Digital Systems	Elec. Eng.	3
FIN-201	Investment Banking	Finance	3
HIS-351	World History	History	3
MU-199	Music Video Production	Music	3
PHY-101	Physical Principles	Physics	4

instructor

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

teaches

ID	*course_id*	*sec_id*	*semester*	*year*
10101	CS-101	1	Fall	2017
10101	CS-315	1	Spring	2018
10101	CS-347	1	Fall	2017
12121	FIN-201	1	Spring	2018
15151	MU-199	1	Spring	2018
22222	PHY-101	1	Fall	2017
32343	HIS-351	1	Spring	2018
45565	CS-101	1	Spring	2018
45565	CS-319	1	Spring	2018
76766	BIO-101	1	Summer	2017
76766	BIO-301	1	Summer	2018
83821	CS-190	1	Spring	2017
83821	CS-190	2	Spring	2017
83821	CS-319	2	Spring	2018
98345	EE-181	1	Spring	2017

Examples of division A/B

sno	pno
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

A

pno
p2

$B1$

pno
p2
p4

$B2$

pno
p1
p2
p4

$B3$

sno
s1
s2
s3
s4

$A/B1$

Examples of division A/B

sno	pno
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

A

pno
p2

$B1$

pno
p2
p4

$B2$

pno
p1
p2
p4

$B3$

sno
s1
s2
s3
s4

$A/B1$

sno
s1
s4

$A/B2$

Examples of division A/B

sno	pno
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

A

pno
p2

$B1$

pno
p2
p4

$B2$

pno
p1
p2
p4

$B3$

sno
s1
s2
s3
s4

$A/B1$

sno
s1
s4

$A/B2$

sno
s1

$A/B3$

Expressing A/B using basic operators

- Division is not essential op; just a useful shorthand
  - (Also true for joins, but joins are so common that systems implement joins specially)
- *Idea*: For *A/B*, compute all *x* values that are not "disqualified" by some *y* value in *B*
  - *x* value is *disqualified* if by attaching *y* value from *B*, we obtain an *xy* tuple that is not in *A*

Disqualified x values: $\pi_x((\pi_x(A) \times B) - A)$

A/B: $\pi_x(A)$ − Disqualified x values

Expressing A/B

$$\pi_{sno}(A) - \pi_{sno}((\pi_{sno}(A) \times B) - A)$$

sno	pno
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

A

sno	pno
s1	p1
s1	p2
s1	p4
s2	p1
s2	p2
s2	p4
s3	p1
s3	p2
s3	p4
s4	p1
s4	p2
s4	p4

$T1 = \pi_{sno}(A) \times B$

$\Leftarrow$

sno
s1
s2
s3
s4

$\pi_{sno}(A)$

$\times$

pno
p1
p2
p4

B

Expressing A/B

$$\pi_{sno}(A) - \pi_{sno}(T1 - A)$$

Subtraction

sno	pno
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

A

sno	pno
~~s1~~	~~p1~~
~~s1~~	~~p2~~
~~s1~~	~~p4~~
~~s2~~	~~p1~~
~~s2~~	~~p2~~
s2	p4
s3	p1
~~s3~~	~~p2~~
s3	p4
s4	p1
~~s4~~	~~p2~~
~~s4~~	~~p4~~

$T1 = \pi_{sno}(A) \times B$

sno	pno
s2	p4
~~s3~~	~~p1~~
s3	p4
s4	p1

$T1 - A$

Projection

Duplicate elimination

pno
p1
p2
p4

B

sno
s2
s3
s4

$T2 = \pi_{sno}(T1 - A)$

Expressing A/B

sno	pno
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

A

sno	pno
~~s1~~	~~p1~~
~~s1~~	~~p2~~
~~s1~~	~~p4~~
~~s2~~	~~p1~~
~~s2~~	~~p2~~
s2	p4
s3	p1
~~s3~~	~~p2~~
s3	p4
s4	p1
~~s4~~	~~p2~~
~~s4~~	~~p4~~

$T1 = \pi_{sno}(A) \times B$

$\pi_{sno}(A) - T2$

sno	pno
s2	p4
~~s3~~	~~p1~~
s3	p4
s4	p1

$T1 - A$

pno
p1
p2
p4

B

$\pi_{sno}(A)$

sno
s1
s2
s3
s4

$-$

sno
s2
s3
s4

$T2 = \pi_{sno}(T1 - A)$

$=$

sno
s1

$A/B = \pi_{sno}(A) - T2$

Summary

- Relational model is ubiquitous
  - Reasoning about information in tables was not always the case!
  - ...but it can be restrictive for specific applications

- Formal foundation for real query languages
  - Helps represent and reason about execution plans
- Five basic operators forming a robust, well-balanced language
  - Selection, projection, cross-product, union, set difference

- Compound operators
  - Useful shorthands like join and division
  - Can be expressed with basic operators
  - But enable faster query execution

Backup Slides

DATA-INTENSIVE APPLICATIONS AND SYSTEMS

Relational Database	Key/Value Database
Defined table schema (database contains tables, tables contain rows, and rows are made up of column values)	No defined domain schema (A domain is basically a bucket with items that can have differing schemas)
Strongly typed schema with constraints and relationships that enforce **data integrity**	Items are identified by keys, and a given item can have a dynamic set of attributes attached to it.
The data model is based on a **"natural" representation** of the data it contains, not on an application's functionality.	In some implementations, attributes are all of a *string* type. In other implementations, attributes have simple types that reflect code types, such as *ints*, *string arrays*, and *lists*.
Normalization of the data model: • Remove data duplication. • Establish table relationships to associate data between tables.	No relationships are explicitly defined between domains or within a given domain.

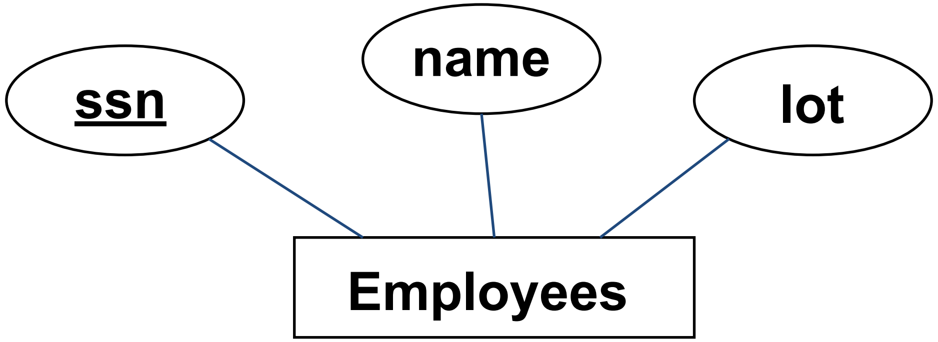
table from http://readwrite.com

Data models

- Basics
- SQL overview
- Keys & Integrity Constraints
- ER to Relational
- ISA to Relational

Logical DB design: ER to relational model

- Entity sets to tables

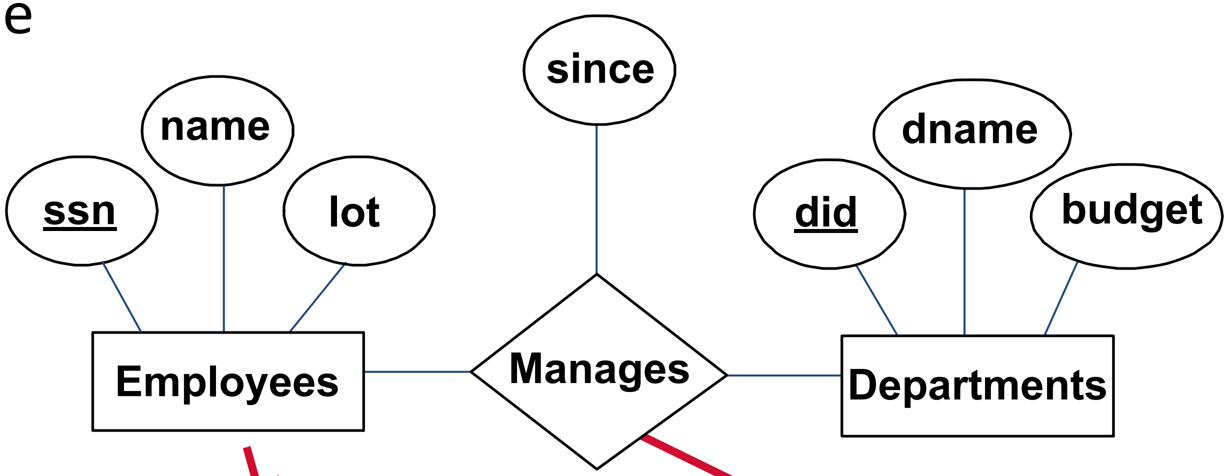



```
CREATE TABLE Employees
 (ssn CHAR(11),
  name CHAR(20),
  lot  INTEGER,
  PRIMARY KEY  (ssn))
```

ssn	name	lot
123-22-3666	Attishoo	48
231-31-5368	Smiley	22
131-24-3650	Smethurst	35

Relation sets to tables

Our favorite example



ssn	name	lot
123-22-3666	Attishoo	48
231-31-5368	Smiley	22
131-24-3650	Smethurst	35

ssn	did	since
123-22-3666	51	1/1/91
123-22-3666	56	3/3/93
231-31-5368	51	2/2/92

Relation sets to tables

- In translating a **many-to-many** relationship set to a relation, attributes of the relation must include:

  1) Keys for each participating entity set (as foreign keys): Such a set of attributes forms a ***superkey*** for the relation

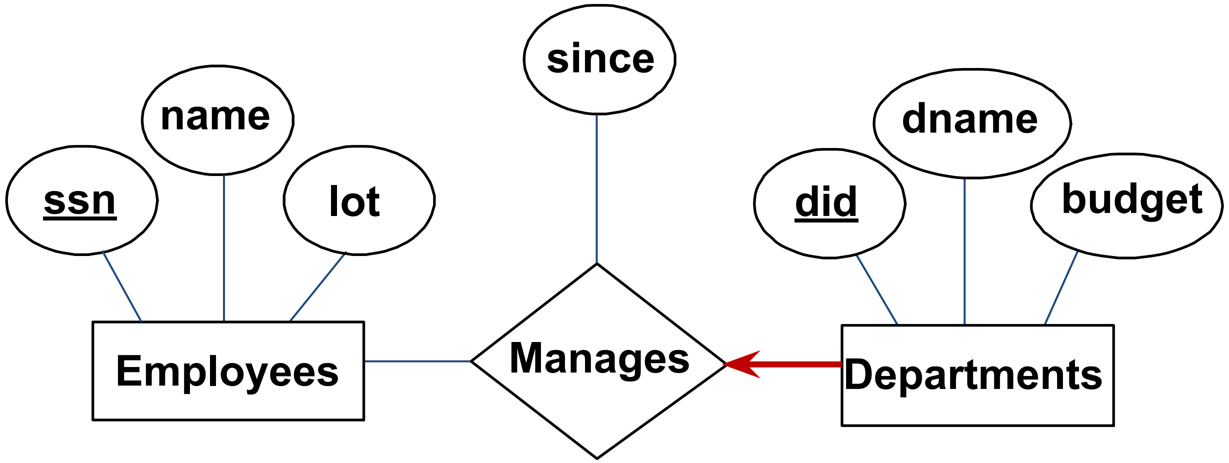
  2) All descriptive attributes

```
CREATE TABLE Manages(
  ssn  CHAR(1),
  did  INTEGER,
  since  DATE,
 PRIMARY KEY (ssn, did),
 FOREIGN KEY (ssn)
    REFERENCES Employees,
 FOREIGN KEY (did)
  REFERENCES Departments)
```

ssn	did	since
123-22-3666	51	1/1/91
123-22-3666	56	3/3/93
231-31-5368	51	2/2/92

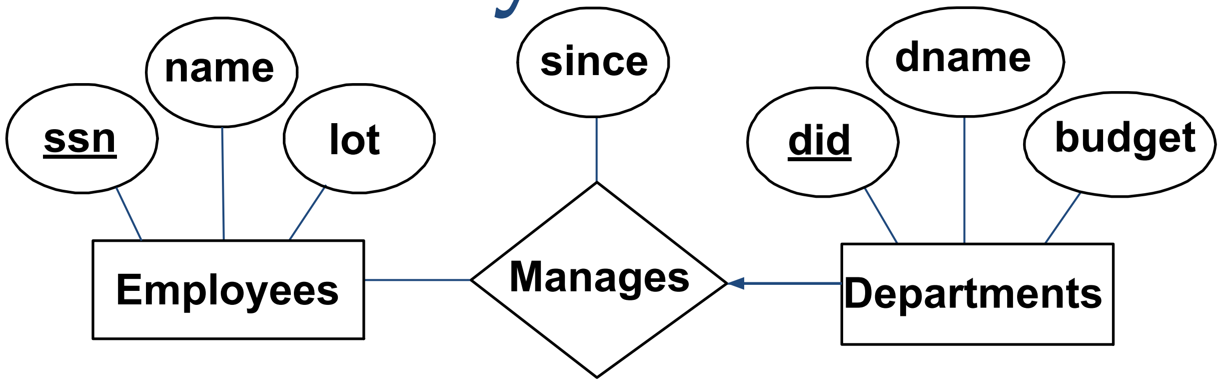
Review: Key constraints in ER

- Each dept has at most one manager, according to the key constraint on Manages




```
CREATE TABLE Manages(
  ssn  CHAR(1),
  did  INTEGER,
  since  DATE,
 PRIMARY KEY (did),
 FOREIGN KEY (ssn)REFERENCES Employees,
 FOREIGN KEY (did)REFERENCES Departments)
```

Translating ER with key constraints



- Since each department has a unique manager, could combine Manages and Departments

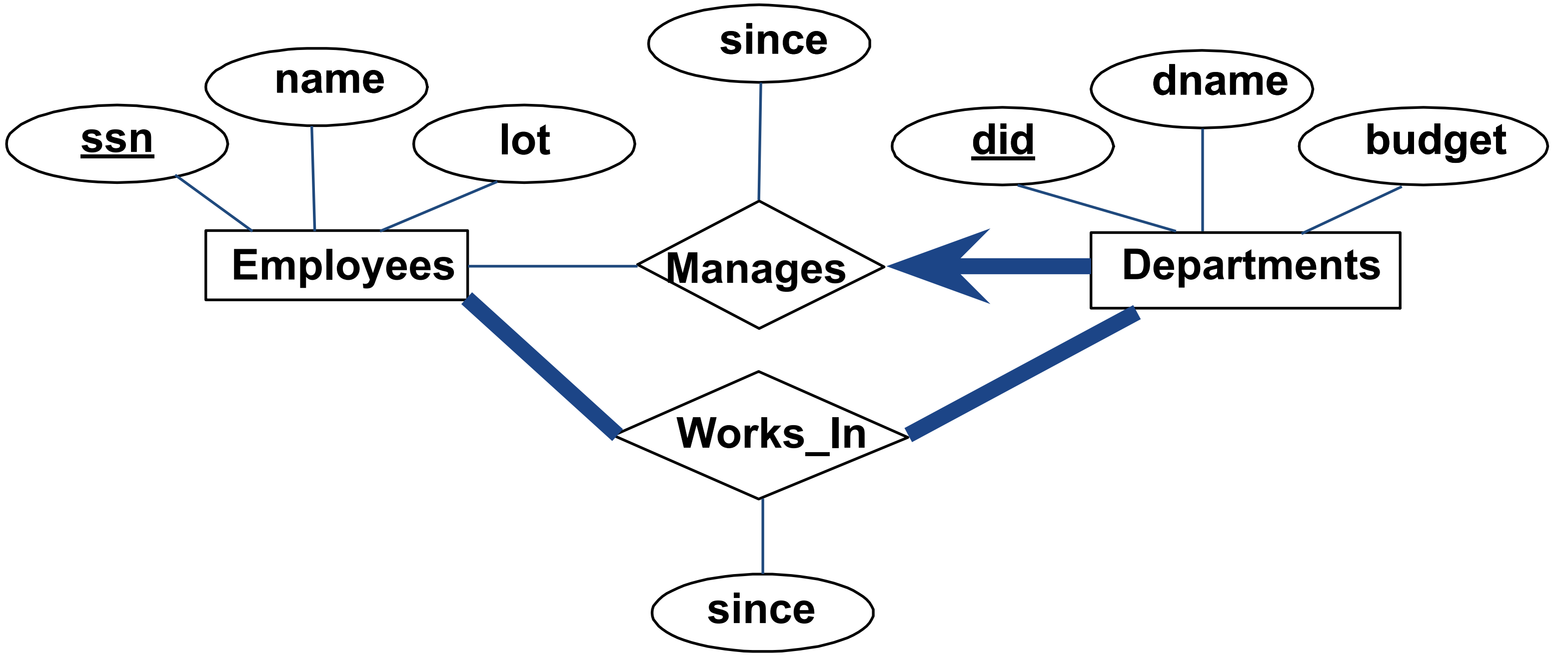
```
CREATE TABLE  Manages(
 ssn  CHAR(11),
 did  INTEGER,
 since  DATE,
 PRIMARY KEY  (did),
 FOREIGN KEY (ssn)
REFERENCES Employees,
 FOREIGN KEY (did) REFERENCES
Departments)
```

vs.

```
CREATE TABLE  Dept_Mgr(
 did  INTEGER,
 dname  CHAR(20),
 budget  REAL,
 ssn  CHAR(11),
 since  DATE,
 PRIMARY KEY  (did),
 FOREIGN KEY (ssn)
  REFERENCES Employees)
```

Review: Participation constraints

- Does every department have a manager?
  - If so, this is a participation constraint: the participation of Departments in Manages is said to be total (vs. partial)
    - Every did value in Departments table must appear in a row of the Manages table (with a non-null ssn value!)



Participation constraints in SQL

- Can capture participation constraints involving one entity set in a binary relationship
  - But little else (without resorting to CHECK constraints)

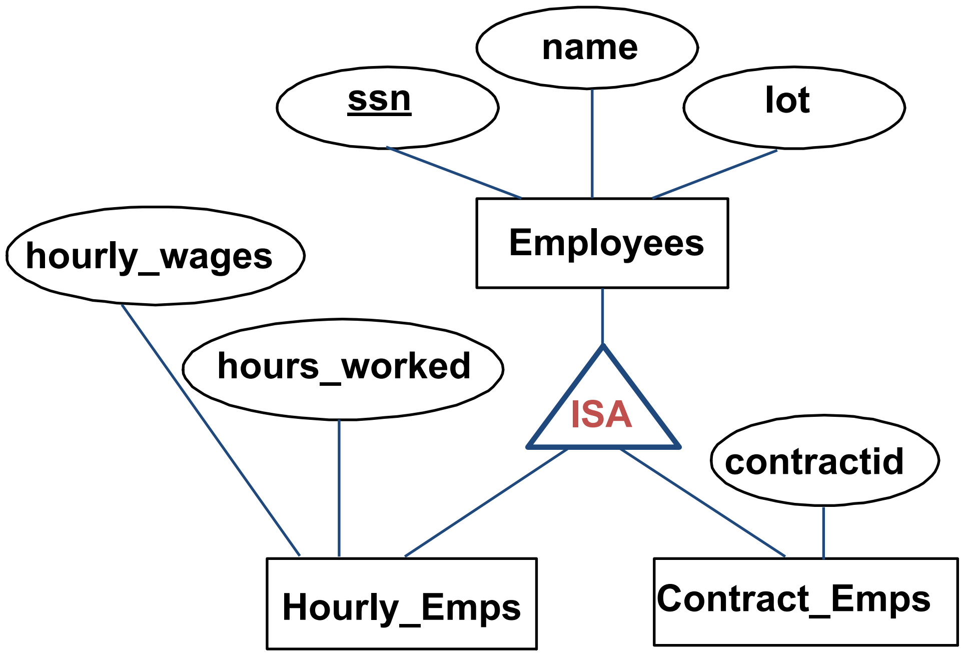
```
CREATE TABLE  Dept_Mgr(
   did  INTEGER,
   dname  CHAR(20),
   budget  REAL,
   ssn  CHAR(11) NOT NULL,
   since  DATE,
   PRIMARY KEY  (did),
   FOREIGN KEY  (ssn) REFERENCES Employees,
      ON DELETE NO ACTION)
```

Data Models

- Basics
- SQL overview
- Keys & Integrity Constraints
- ER to Relational
- ISA to Relational

Translating ISA hierarchy to relations

- *General approach:*
  - 3 relations: Employees, Hourly_Emps and Contract_Emps.
    - *Hourly_Emps*: Every employee is recorded in Employees. For hourly emps, extra info recorded in Hourly_Emps (*hourly_wages*, *hours_worked*, *<u>ssn</u>*); must delete Hourly_Emps tuple if referenced Employees tuple is deleted)
    - Queries involving all employees easy, those involving just Hourly_Emps require a join to get some attributes



- Alternative: Just Hourly_Emps and Contract_Emps
  - *Hourly_Emps*: *<u>ssn</u>, name, lot, hourly_wages, hours_worked*
  - Each employee must be in one of these two subclasses

Relational model: Foreign keys – SQL

- Example: Only students listed in the Students relation should be allowed to enroll for courses.
  - sid is a foreign key referring to Students

```
CREATE TABLE Enrolled
(cid CHAR(20),sid CHAR(20),grade CHAR(2),
 PRIMARY KEY (sid,cid),
 FOREIGN KEY (sid) REFERENCES Students(sid))
```

Students

sid	name	login	age	gpa
50000	Dave	dave@cs	19	3.3
53666	Jones	jones@cs	18	3.4
53688	Smith	smit@ee	18	3.2
…	…	…	…	…

Enrolled

cid	sid	grade
Carnatic101	53666	C
Raggae203	50000	B
Topology112	53666	A
…	…	…

Where do ICs come from?

- ICs are based upon the semantics of the real-world that is being described in the database relations

- We can check a database instance to see if an IC is violated, but we can NEVER infer that an IC is true by looking at an instance
  - An IC is a statement about all possible instances
  - From example, we know name is not a key, but the assertion that sid is a key is given to us

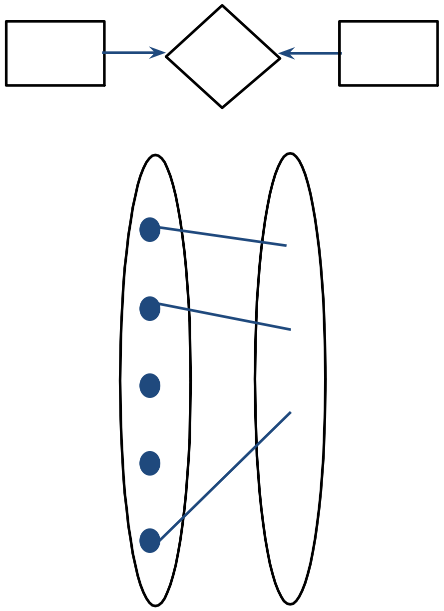
- Key and foreign key ICs are the most common; more general ICs supported too

Wake-up question

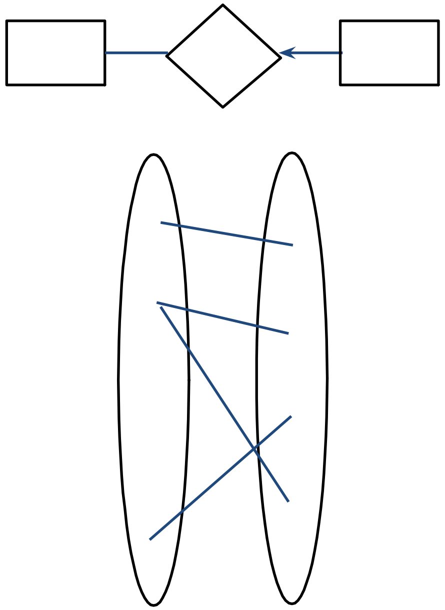
- What if the toy department has no manager (yet) ?

```
CREATE TABLE  Dept_Mgr(
 did  INTEGER,
 dname  CHAR(20),
 budget  REAL,
 ssn  CHAR(11),
 since  DATE,
 PRIMARY KEY  (did),
 FOREIGN KEY (ssn)
  REFERENCES Employees)
```

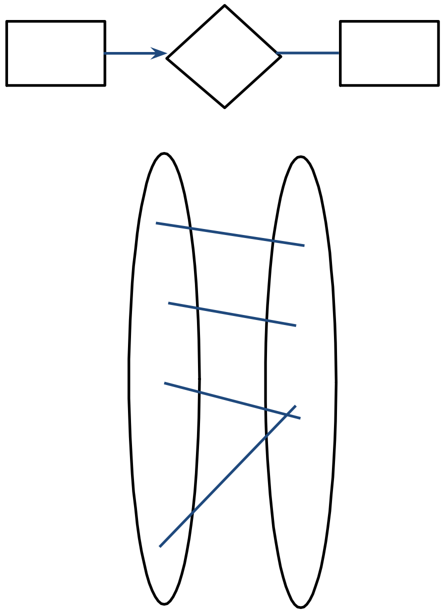
Review: Key Constraints in ER



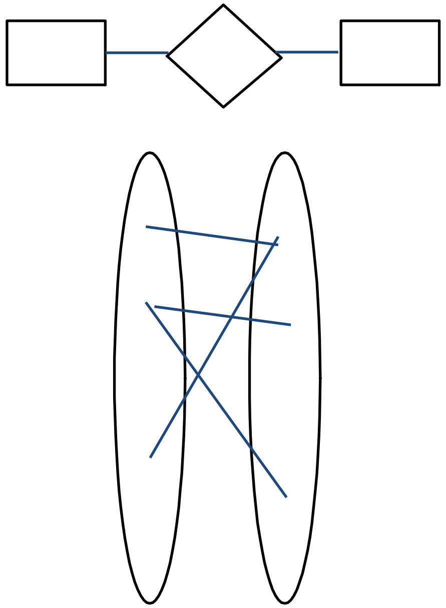
1-to-1



1-to Many



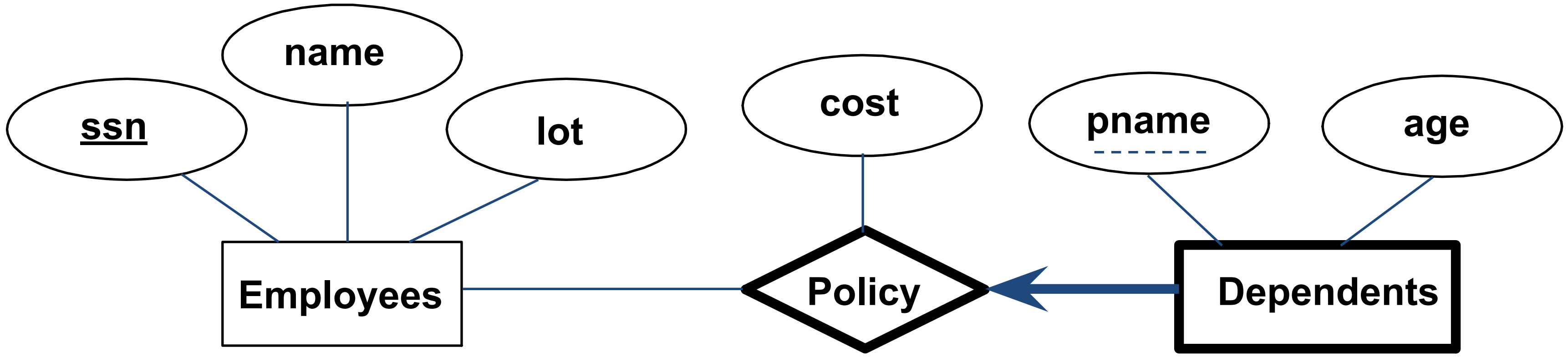
Many-to-1



Many-to-Many

Review: Weak entities

- A weak entity can be identified uniquely only by considering the primary key of another (owner) entity.
  - Owner entity set and weak entity set must participate in a one-to-many relationship set (1 owner, many weak entities).
  - Weak entity set must have total participation in this identifying relationship set.



Translating Weak Entity Sets

- Weak entity set and identifying relationship set are translated into a single table.
  - When the owner entity is deleted, all owned weak entities must also be deleted.

```
CREATE TABLE  Dep_Policy (
  pname  CHAR(20),
  age  INTEGER,
  cost  REAL,
  ssn  CHAR(11) NOT NULL,
  PRIMARY KEY  (pname, ssn),
  FOREIGN KEY  (ssn) REFERENCES Employees,
    ON DELETE CASCADE)
```

Data Models

- Basics
- SQL overview
- Keys & Integrity Constraints
- ER to Relational
- ISA to Relational
- noSQL data models

Not all data fits in tables naturally

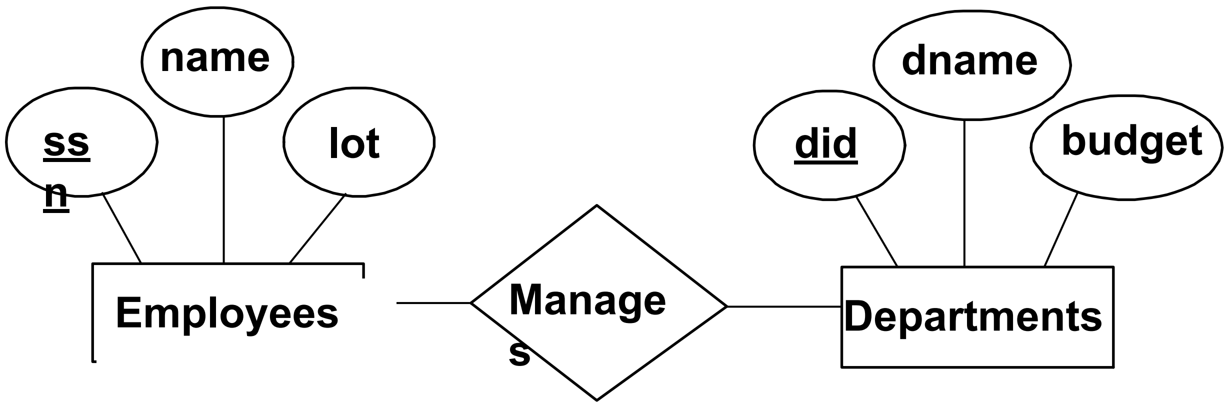
- The rise of noSQL!

1. Key-Value data model
   - Object vs. Table

2. Hierarchies, Arrays

How different are SQL and noSQL?

Object-centric representation



- Represent as a single data block
- Center all information around a core entity
  - In this case: “Employee, and all the info about him/her”

```
class Employee
{
 int ssn;
 string name;
 int lot;
 list<Department> managedDepts;
}
```

```
private class Department
{
 int did;
 string dname;
 int budget;
}
```

Anchor all information on an object type!

Key-value pairs

123-22-3666 →	Attishoo	48	[(51,IT,1000),(56,Accounting,3000)]
231-31-5368 →	Smiley	22	[(51,IT,1000)]
131-24-3650 →	Smethurst	35	[]

123-22-3666 →	Binary Object 1
231-31-5368 →	Binary Object 2
131-24-3650 →	Binary Object 3

- Many applications prefer the 2nd option!
- Pros: Schema Flexibility / less rigid constraints
- Cons: Queries less expressive
  - put()
  - get()

cassandra

Support for Hierarchies & Arrays?

- K-V model supports storing hierarchies & arrays
- But it is agnostic to them!
  - "Associate any value with a key", but that's it!
- Do we need this support? Are there use cases?
  - XML!
  - JSON!

Same example, JSON representation

```
{
  "id": 123223666, "name": "Attishoo",
  "manages": [{"did": 51, "name": "IT",
               "budget": 1000},
                 {"did": 56, "name": "Accounting",
               "budget": 3000}]
}
```

Many systems implement it as K-V!

Example 3

- Find sailors who have reserved a red or a green boat.
- **Hint:** Can identify all red or green boats, then find sailors who have reserved one of these boats.

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

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

Example 4

- Find names of sailors who've reserved a red <u>and</u> a green boat.
- **Hint:** Previous approach won't work! Must identify sailors who've reserved red boats, sailors who've reserved green boats, then find the intersection (note that *sid* is a key for Sailors).

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

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

$$\pi_{sname}((Tempred \cap Tempgreen) \bowtie Sailors)$$

Relational Algebra

- Relational Query Languages
- Selection & Projection
- Union, Set Difference & Intersection
- Cross product & Joins
- Examples
- Division

Your turn…

1. Find (the name of) all sailors whose rating is above 9.

2. Find all sailors who reserved a boat prior to November 1, 1996.

3. Find (the names of) all boats that have been reserved at least once.

4. Find all pairs of sailors with the same rating.

5. Find all pairs of sailors in which the older sailor has a lower rating.

Answers…

1. Find (the name of) all sailors whose rating is above 9.

$$\pi_{sname}(\sigma_{rating>9}(Sailors))$$

Answers…

2. Find all sailors who reserved a boat prior to November 1, 1996.

$$\pi_{sname}(Sailors \bowtie \sigma_{day<'11/1/96'}(\text{Reserves}))$$

Answers…

3. Find (the names of) all boats that have been reserved at least once.

$$\pi_{bname}(Boats \bowtie Reserves)$$

Answers…

4. Find all pairs of sailors with the same rating.

$$\rho(S1(1 \rightarrow sid1, 2 \rightarrow sname1, 3 \rightarrow rating1, 4 \rightarrow age1), Sailors)$$

$$\rho(S2(1 \rightarrow sid2, 2 \rightarrow sname2, 3 \rightarrow rating2, 4 \rightarrow age2), Sailors)$$

$$\pi_{sname1, sname2}(S1 \bowtie_{rating1 = rating2 \wedge sid1 \neq sid2} S2)$$

Answers…

5. Find all pairs of sailors in which the older sailor has a lower rating.

$$\pi_{sname1, sname2}(S1 \bowtie_{age1 > age2 \wedge rating1 < rating2} S2)$$

Set semantics vs. multiset ("bag") semantics

- Both versions of relational algebra exist.
- Database systems use bag semantics.
- Set semantics simpler and cleaner.
- Some operations require set semantics.
- Some operations ”force” bag semantics, unless we eliminated duplicates.
- Under bag semantics, set-shaped databases become bag-shaped. (example?)

Example: Find the names of sailors who have reserved all boats

- Uses division; schemas of the input relations to / must be carefully chosen:

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

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

- To find sailors who have reserved all 'Interlake' boats:

$$\ldots \quad / \ \pi_{bid}(\sigma_{bname='Interlake'} Boats)$$