BLG 317E
DATABASE SYSTEMS

WEEK 2

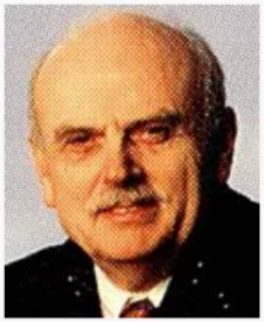
Slides credit:
S. Shivakumar, CS 145, Stanford University
Database System Concepts, Silberschatz, Korth, Sudarshan

ITU

Intro to Relational Model

Relational Model

- Relational model is the most widely used database model.
- It is implemented by many commercial and open source vendors (companies).
- SQL (Structured Query Language) was developed by IBM in 1970s.



1969: Edgar F. "Ted" Codd invents the relational database.

1969: Edgar F. "Ted" Codd invents the relational database.

1973: Cullinane, led by John J. Cullinane, ships IDMS, a network-model database for IBM mainframes.

1976: Honeywell ships Multics Relational Data Store, the first commercial relational database.

1979: Oracle introduces the first commercial SQL relational database management system.

1983: IBM introduces DB2.

1985: The first business intelligence system is designed for Procter & Gamble.

1991: W.H. "Bill" Inmon publishes Building the Data Warehouse.

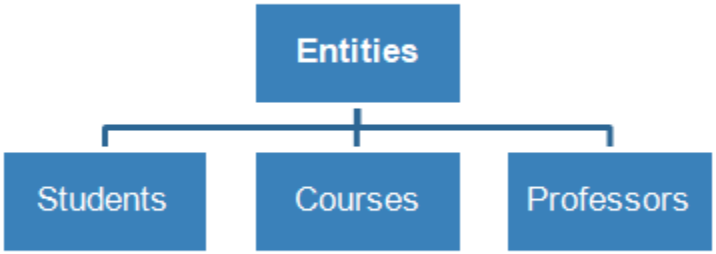
1991: W.H. "Bill" Inmon publishes Building the Data Warehouse.

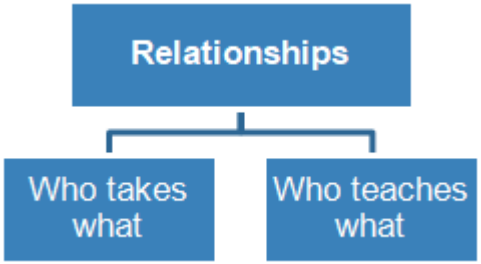
https://www.computerworld.com/article/2588199/the-story-so-far.html

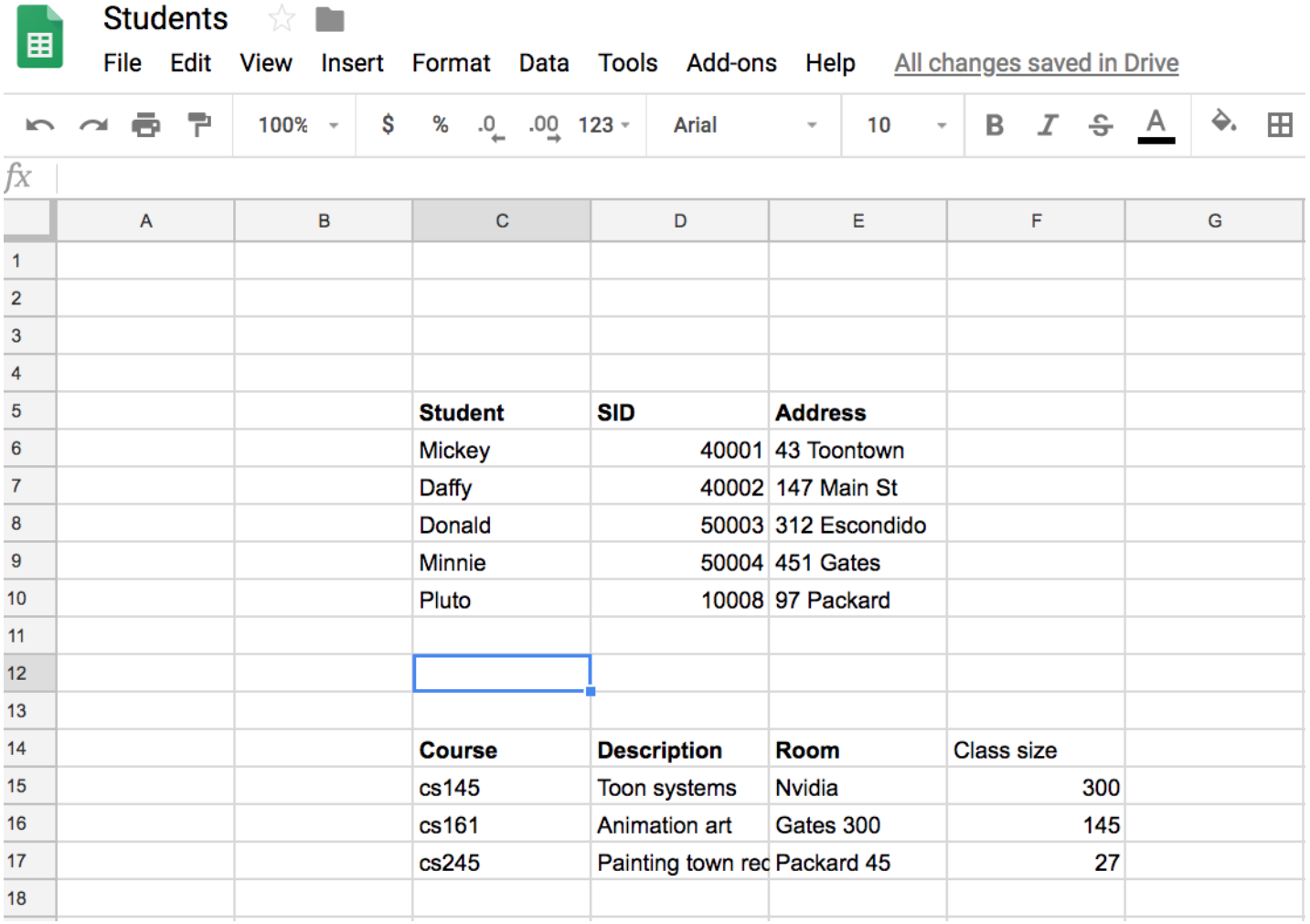
A Motivating, Running Example

Consider building a course management system (CMS):

Entities (e.g., Students, Courses)
Relationships (e.g., Ahmet is enrolled in CS 350)







	A	B	C	D	E	F	G
1							
2							
3							
4							
5			**Student**	**SID**	**Address**		
6			Mickey	40001	43 Toontown		
7			Daffy	40002	147 Main St		
8			Donald	50003	312 Escondido		
9			Minnie	50004	451 Gates		
10			Pluto	10008	97 Packard		
11							
12							
13							
14			**Course**	**Description**	**Room**	Class size	
15			cs145	Toon systems	Nvidia	300	
16			cs161	Animation art	Gates 300	145	
17			cs245	Painting town red	Packard 45	27	
18							

‘Modeling’ the CMS

Logical Schema

Students(sid: *string*, name: *string*, gpa: *float*)

Courses(cid: *string*, cname: *string*, credits: *int*)

Enrolled(sid: *string,* cid*: string,* grade*: string*)

sid	Name	Gpa
101	Bob	3.2
123	Mary	3.8

Students

Corresponding *keys*

sid	cid	Grade
123	564	A

Enrolled

cid	cname	credits
564	564-2	4
308	417	2

Courses

Set algebra (reminder)

List: [1, 1, 2, 3]

Set: {1, 2, 3}

Multiset: {1, 1, 2, 3}

A **multiset** is an unordered list (or: a set with multiple duplicate instances allowed)

UNIONs

Set: {1, 2, 3} U { 2 } = { 1, 2, 3 }

Multiset: {1, 1, 2, 3} U { 2 } = { 1, 1, 2, 2, 3 }

Tables

Product

PName	Price	Manuf
Gizmo	$19.99	GizmoWorks
Powergizmo	$29.99	GizmoWorks
SingleTouch	$149.99	Canon
MultiTouch	$203.99	Hitachi

A **relation** or **table** is a multiset of tuples having the attributes specified by the schema

Let's break this definition down

Tables

Product

PName	Price	Manuf
Gizmo	$19.99	GizmoWorks
Powergizmo	$29.99	GizmoWorks
SingleTouch	$149.99	Canon
MultiTouch	$203.99	Hitachi

An **attribute** (or **column or field**) is a typed data entry present in each tuple in the relation

*Attributes must have an **atomic** type in standard SQL, i.e. not a list, set, etc.*

STUDENT Table

student_id	name	gpa	courses_taken
53410	Jones	3.4	Mat101, Kim101, Fiz101

Standard SQL does not have array datatype.
(PostgreSQL DBMS supports arrays.)

Tables

Product

PName	Price	Manuf
Gizmo	$19.99	GizmoWorks
Powergizmo	$29.99	GizmoWorks
SingleTouch	$149.99	Canon
MultiTouch	$203.99	Hitachi

A **tuple** or **row** is a single entry in the table having the attributes specified by the schema

*Also referred to sometimes as a **Record***

Tables

Product

PName	Price	Manuf
Gizmo	$19.99	GizmoWorks
Powergizmo	$29.99	GizmoWorks
SingleTouch	$149.99	Canon
MultiTouch	$203.99	Hitachi

The number of tuples is the **cardinality** of the relation

The number of attributes is the **arity/degree** of the relation

Tuple order is insignificant.

Attribute order is insignificant.

Data Types (Domains)

Atomic types:

- Characters: CHAR(20), VARCHAR(50)
- Numbers: INT, BIGINT, SMALLINT, FLOAT
- Others: MONEY, DATETIME…

Every attribute must have an atomic type

- Hence tables are flat

Table Schemas

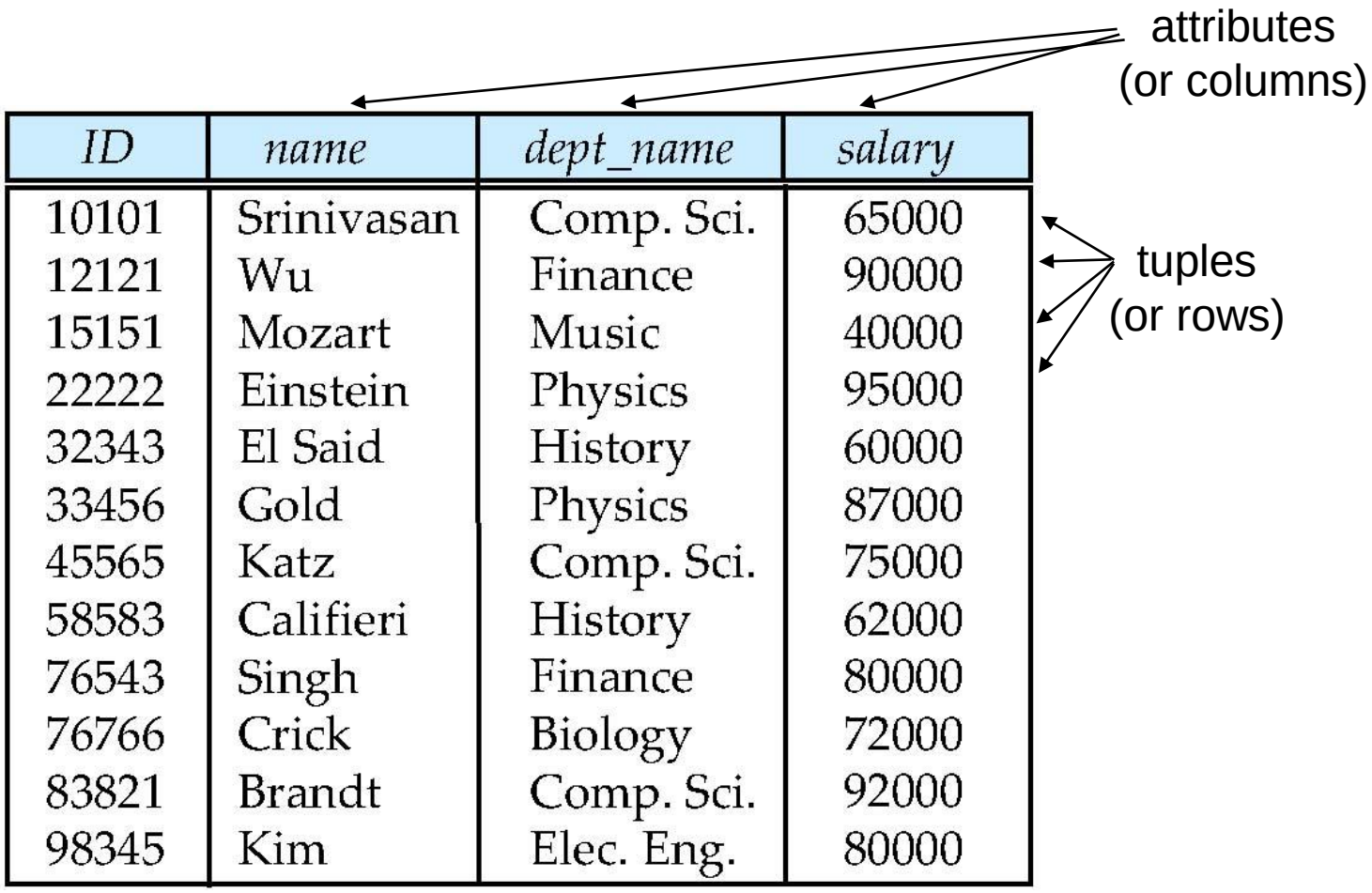
The **schema** of a table is the table name, its attributes, and their types:

Product(Pname: *string*, Price: *float*, Category: *string*, Manufacturer: *string*)

A **key** is an attribute or a set of attributes whose values are unique; we underline a key.

Product(<u>Pname</u>: *string*, Price: *float*, Category: *string*, <u>Manufacturer</u>: *string*)

Example of a Relation



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

Relational Data Model - Summary

- Relational database contains a set of relations (tables).
- A relation is a table with rows and columns.
- Every relation has a schema definition, which describes the columns (fields).
- **Schema :** Specifies name of relation (table), also names and data types of each column in table.
- **Instance :** A table with rows (records, tuples) and columns (fields, attributes), containing the actual data.
  - Number of rows : cardinality
  - Number of fields : arity/degree

Example: PUBLICATIONS Database Schema

- The following is a conceptual schema definition.
  (They are not SQL commands, only informal definitions.)
- Table names and field names are written.
- Data types of fields may be omitted.
- Primary key fields are underlined.

Database Schema

```
PUBLISHER (publisher_id, publisher_name, phone)
AUTHOR    (author_id, author_name, email)
BOOK      (ISBN, publisher_id, author_id, title)
```

Key Attributes

Key Attribute:

- must uniquely identify a row in a table.
- No duplicates, no null values.
- Example: **author_id** column

Candidate Key:

- Each alternative key is called a candidate key.
- One of the candidate keys is arbitrarily designated to be the primary key.
- Example: **email column** is a candidate key.

Attributes (columns / fields)

author_id	author_name	email
100	Elmasri	elm@uta
200	Ullman	ulm@ibm
300	Weinberg	null
400	Silberschatz	slb@ms
500	Ramakrishnan	rma@wisc

Tuples (rows / records)

Primary Key

- A Primary Key is a unique attribute, or a group of multiple attributes, that uniquely identify a record in a table without duplications.
- If there are more than one **candidate** key in a table, one is selected as the primary key, others are alternate keys.
- Any attribute that is part of the primary key can not be empty in any tuple.
- Every relation (table) should have a primary key.

Surrogate (Substitute) Key

- If a natural key can not be found, a surrogate (substitute) key can be defined.
- It can be generated by the DBMS system as an auto-incremented counter.

AUTHOR Table

author_id	author_name	email
1	Elmasri	elm@uta
2	Ullman	ulm@ibm
3	Weinberg	null
4	Silberschatz	slb@ms
5	Ramakrishnan	rma@wisc

Surrogate Key:
Sequential record numbers are (record_number) automatically assigned by DBMS, every time a new record is added to table.

Super Key

- Let K $\subseteq$ R
- *K* is a **super key** of *R* if values for *K* are sufficient to identify a unique tuple of each possible relation *r(R)*
  - Example: {*ID*} and {ID,name} are both superkeys of *instructor.*
- Superkey *K* is a **candidate key** if *K* is minimal

  Example: {*ID*} is a candidate key for *Instructor*

Foreign Key

- **Foreign Key:** An attribute of a table that is a Candidate key of another table.
- Foreign keys are for implementation of relations (associations) between tables.
- Must correspond (reference) to the primary/candidate key of the other table.
- DBMS enforces all foreign key constraints, to maintain **referential integrity**.

Primary Key

PUBLISHER table

publisher_id	publisher_name	phone
10	Addison-Wesley	212-303-8421
20	Pearson Prentice Hall	null
30	McGraw-Hil	216-404-02354

BOOK table

Foreign Key Foreign Key

ISBN	publisher_id	author_id	title
A0012345	10	100	Fundamentals of Databases
B0033224	20	200	Database Systems
C0092831	30	300	SQL Reference
D0027354	30	400	Database Concepts
E0091237	30	500	Database Management
F0072261	20	100	Database Design

Foreign Keys

- *dept_name* in *instructor* is a foreign key from *instructor* referencing *department*

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

(a) The *instructor* table

dept_name	*building*	*budget*
Comp. Sci.	Taylor	100000
Biology	Watson	90000
Elec. Eng.	Taylor	85000
Music	Packard	80000
Finance	Painter	120000
History	Painter	50000
Physics	Watson	70000

(b) The *department* table

Referential Integrity

- **Referential Integrity:** All values of a **foreign key** attribute in a table must be present, among the values of the referenced table's **primary key** or candidate key attribute.
- If there is no corresponding Primary Key for a Foreign Key, a runtime error occurs.
- If a request (Add/Delete/Update) would break referential integrity, the DBMS can use one of the following actions:
  - Don't allow, and return an error (default action of DBMS)
  - Reflect the change to affected tuples
  - Assign null value to foreign key
  - Assign default value to foreign key

Referential Integrity

Example:

- Insert operation for the following row is not allowed by DBMS.
  - Because **publisher_id** and **author_id** must exist in the related tables (PUBLISHER and AUTHOR).

BOOK Table

ISBN	publisher_id	author_id	title
A0036721	0	0	Digital Systems

Relational Query Languages

- Procedural versus non-procedural, or declarative
- “Pure” languages:
  - Relational algebra
  - Tuple relational calculus
  - Domain relational calculus
- The above 3 pure languages are equivalent in computing power
- We will concentrate in this semester only on relational algebra
  - Procedural language consisting of a set of operations that take one or two relations as input and produce a new relation as their result.
  - Not turing-machine equivalent
    - (not equal to a programming language in expressive power)

Relational Algebra
Basic Operations

Operation	Symbol	Description
Selection	σ	Selects a subset of rows from relation (horizontal)
Projection	π	Retains only wanted columns from relation (vertical).
Cross-product	$\times$	Allows to combine each row of the left relation with the each row of the right relations (Cartesian product).
Join	$\bowtie$	A compound operator involving cross product, selection, and projection.

Relational Algebra
Set Operations

(R1 and R2 are tables.)

Operation	Symbol	Description
Union	$\cup$	Tuples in R1 or in R2.
Intersection	$\cap$	Tuples in R1 and in R2.
Difference	—	Tuples in R1, but not in R2.

Select Operation

- The **select** operation selects tuples that satisfy a given predicate.
- Notation: $\sigma_p(r)$
- p is called the **selection predicate**
- Example: select those tuples of the *instructor* relation where the instructor is in the "Physics" department.
  - Query

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

  - Result

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

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

Select Operation (Cont.)

- We allow comparisons using

  $=, \neq, >, \geq, <, \leq$

  in the selection predicate.
- We can combine several predicates into a larger predicate by using the connectives:

  $\wedge$ (**and**), $\vee$ (**or**), $\neg$ (**not**)

- Example: Find the instructors in Physics with a salary greater $90,000, we write:

$$\sigma_{dept\_name = \text{“Physics”} \wedge salary > 90,000}(instructor)$$

- Besides, select predicate may include comparisons between two attributes.
  - Example, find all departments whose name is the same as their building name:
  - $\sigma_{dept\_name = building}(department)$

Project Operation

- A unary operation that returns its argument relation, with certain attributes left out.
- Notation:

$$\Pi_{A_1,A_2,A_3 \ldots A_k} (r)$$

where A_1, A_2 are attribute names and r is a relation name.

- The result is defined as the relation of k columns obtained by erasing the columns that are not listed

Project Operation (Cont.)

- Example: eliminate the *dept_name* attribute of *instructor*
- Query*:*

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

- Result:

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

ID	*name*	*salary*
10101	Srinivasan	65000
12121	Wu	90000
15151	Mozart	40000
22222	Einstein	95000
32343	El Said	60000
33456	Gold	87000
45565	Katz	75000
58583	Califieri	62000
76543	Singh	80000
76766	Crick	72000
83821	Brandt	92000
98345	Kim	80000

Composition of Relational Operations

- The result of a relational-algebra operation is relation.
- Therefore, relational-algebra operations can be composed together into a **relational-algebra expression**.

QUERY : Get author names and their email address, whose ID number is less than 300.

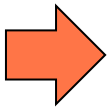
RA Query

$$\pi_{\text{author\_name , email}}(\sigma_{\text{author\_id} < 300}(\text{AUTHOR}))$$

Projection operation is applied second.

Selection operation is applied first.

author_id	author_name	email
100	Elmasri	elm@uta
200	Ullman	ulm@ibm
300	Weinberg	null
400	Silberschatz	slb@ms
500	Ramakrishnan	rma@wisc



author_name	email
Elmasri	elm@uta
Ullman	ulm@ibm

Cartesian-Product Operation

- The Cartesian-product operation (denoted by X) allows us to combine information from any two relations.
  - Example: the Cartesian product of the relations *instructor* and t*eaches* is written as:

    instructor X *teaches*

- We construct a tuple of the result out of each possible pair of tuples:
  - one from the *instructor* relation and one from the *teaches* relation (see the next slides)

- Since the instructor *ID* appears in both relations we distinguish between these attributes by attaching to the attribute the name of the relation from which the attribute originally came.
  - *instructor.ID*
  - *teaches.ID*

ID	name	dept_name	salary

ID	course_id	sec_id	semester	year

The *instructor* x *teaches* table

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

x

ID	*course_id*	*sec_id*	*semester*	*year*
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009
32343	HIS-351	1	Spring	2010
45565	CS-101	1	Spring	2010
45565	CS-319	1	Spring	2010
76766	BIO-101	1	Summer	2009
76766	BIO-301	1	Summer	2010
83821	CS-190	1	Spring	2009
83821	CS-190	2	Spring	2009
83821	CS-319	2	Spring	2010
98345	EE-181	1	Spring	2009

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
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

Cartesian-Product Operation

Student

S_id	Name	Class	Age
1	Ali	5	25
2	Veli	10	30
3	Ayse	8	35

Course

C_id	C_name
11	Database Sys
21	Operating Sys

Student X Course

S_id	Name	Class	Age	C_id	C_name
1	Ali	5	25	11	Database Sys
1	Ali	5	25	21	Operating Sys
2	Veli	10	30	11	Database Sys
2	Veli	10	30	21	Operating Sys
3	Ayse	8	35	11	Database Sys
3	Ayse	8	35	21	Operating Sys

Join Operation

- The Cartesian-Product

  instructor X *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 "*instructor* X *teaches* " that pertain to instructors and the courses that they taught, we write:

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

  - We get only those tuples of "*instructor* X *teaches*" that pertain to instructors and the courses that they taught.
- The result of this expression, shown in the next slide

Join Operation (Cont.)

- The table corresponding to:

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

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

ID	course_id	sec_id	semester	year
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009
32343	HIS-351	1	Spring	2010
45565	CS-101	1	Spring	2010
45565	CS-319	1	Spring	2010
76766	BIO-101	1	Summer	2009
76766	BIO-301	1	Summer	2010
83821	CS-190	1	Spring	2009
83821	CS-190	2	Spring	2009
83821	CS-319	2	Spring	2010
98345	EE-181	1	Spring	2009

ID	name	dept_name	salary	course_id	sec_id	semester	year
10101	Srinivasan	Comp. Sci.	65000	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	CS-347	1	Fall	2009
12121	Wu	Finance	90000	FIN-201	1	Spring	2010
15151	Mozart	Music	40000	MU-199	1	Spring	2010
22222	Einstein	Physics	95000	PHY-101	1	Fall	2009
32343	El Said	History	60000	HIS-351	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-101	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-319	1	Spring	2010
76766	Crick	Biology	72000	BIO-101	1	Summer	2009
76766	Crick	Biology	72000	BIO-301	1	Summer	2010
83821	Brandt	Comp. Sci.	92000	CS-190	1	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-190	2	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-319	2	Spring	2010
98345	Kim	Elec. Eng.	80000	EE-181	1	Spring	2009

Join Operation (Cont.)

- The **join** operation allows us to combine a select operation and a Cartesian-Product operation into a single operation.
- Consider relations *r* (*R*) and *s* (*S*)
- Let "theta" be a predicate on attributes in the schema R "union" S. The join operation r $\bowtie_\theta$ s is defined as follows:

$$r \bowtie_\theta s = \sigma_\theta(r \times s)$$

- Thus

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

- Can equivalently be written as

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

Join Example

$$\text{Student} \bowtie_{\text{Student.Std} = \text{Course.Class}} (\text{Course})$$

Student

S_id	Name	Std	Age
1	Ali	5	25
2	Veli	10	30
3	Ayse	8	35

Course

Class	C_name
10	Database Sys
5	Operating Sys

$$\text{Student} \bowtie_{\text{Student.Std} = \text{Course.Class}} (\text{Course})$$

S_id	Name	Std	Age	Class	C_name
1	Ali	5	25	5	Operating Sys
2	Veli	10	30	10	Database Sys

Join Types

- **Natural Join :** Uses attributes with the same name, that exists in both tables. Ex: ***author*** $\bowtie$ ***book***
  - The followings are intermediate steps.
    - Compute cross product R1 X R2
    - Select rows where attributes that appear in both relations have equal key values
    - Project all unique atttributes and one copy of each of the common ones.
- **Equi-Join :** Special case of Natural join, where an equality condition operator is written on common columns (names may be different). Ex: ***instructor*** $\bowtie_{Instructor.id = teaches.id}$ ***teaches***
- **Theta-Join :** Any join condition may be used. Ex: $r \bowtie_{\theta} s$

Join operation (Natural-Join)

QUERY : Join all author records with corresponding book records, over the common key field (author ID).

AUTHOR

author_id	author_name	email

BOOK

ISBN	publisher_id	author_id	title

RA Query $AUTHOR \bowtie BOOK$

Result

author_id	author_name	email	ISBN	publisher_id	author_id	title
100	Elmasri	elm@uta	A0012345	10	100	Fundamentals of Databases
100	Elmasri	elm@uta	F0072261	20	100	Database Design
200	Ullman	ulm@ibm	B0033224	20	200	Database Systems
300	Weinberg	null	C0092831	30	300	SQL Reference
400	Silberschatz	slb@ms	D0027354	30	400	Database Concepts
500	Ramakrishnan	rma@wisc	E0091237	30	500	Database Management

Join operation (Natural-Join)

QUERY : Join all author records with corresponding book records, over the common key field (author ID).

AUTHOR

author_id	author_name	email

BOOK

ISBN	publisher_id	author_id	title

RA Query $AUTHOR \bowtie BOOK$

Result

author_id	author_name	email	ISBN	publisher_id	title
100	Elmasri	elm@uta	A0012345	10	Fundamentals of Databases
100	Elmasri	elm@uta	F0072261	20	Database Design
200	Ullman	ulm@ibm	B0033224	20	Database Systems
300	Weinberg	null	C0092831	30	SQL Reference
400	Silberschatz	slb@ms	D0027354	30	Database Concepts
500	Ramakrishnan	rma@wisc	E0091237	30	Database Management

Duplicate common column removed

Union Operation

- The union operation allows us to combine two relations
  - Notation: $r \cup s$
- For $r \cup s$ to be valid.
  1. *r, s* must have the *same* **arity** (same number of attributes)
  2. The attribute domains of r and s must be **compatible**

     (example: 2nd column of *r* deals with the same type of values as does the 2nd column of *s*)
- Example: to find all courses taught in the Fall 2009 semester, or in the Spring 2010 semester, or in both

$$\Pi_{course\_id}(\sigma_{semester=\text{“Fall”} \wedge year=2009}(section)) \cup \Pi_{course\_id}(\sigma_{semester=\text{“Spring”} \wedge year=2010}(section))$$

course_id	*sec_id*	*semester*	*year*	*building*	*room_number*	*time_slot_id*

Union Operation (Cont.)

- Result of:

$$\Pi_{course\_id}(\sigma_{semester=\text{“Fall”} \wedge year=2009}(section)) \cup \Pi_{course\_id}(\sigma_{semester=\text{“Spring”} \wedge year=2010}(section))$$

course_id	*sec_id*	*semester*	*year*	*building*	*room_number*	*time_slot_id*
BIO-101	1	Summer	2009	Painter	514	B
BIO-301	1	Summer	2010	Painter	514	A
CS-101	1	Fall	2009	Packard	101	H
CS-101	1	Spring	2010	Packard	101	F
CS-190	1	Spring	2009	Taylor	3128	E
CS-190	2	Spring	2009	Taylor	3128	A
CS-315	1	Spring	2010	Watson	120	D
CS-319	1	Spring	2010	Watson	100	B
CS-319	2	Spring	2010	Taylor	3128	C
CS-347	1	Fall	2009	Taylor	3128	A
EE-181	1	Spring	2009	Taylor	3128	C
FIN-201	1	Spring	2010	Packard	101	B
HIS-351	1	Spring	2010	Painter	514	C
MU-199	1	Spring	2010	Packard	101	D
PHY-101	1	Fall	2009	Watson	100	A

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

Set-Intersection Operation

- The set-intersection operation allows us to find tuples that are in both the input relations.
  - Notation: $r \cap s$
- Assume:
  - r, s have the *same arity*
  - attributes of r and s are compatible
- Example: Find the set of all courses taught in both the Fall 2009 and the Spring 2010 semesters.

$$\Pi_{course\_id}\,(\sigma_{semester=\text{“Fall”}\ \wedge\ year=2009}\,(section)) \cap \Pi_{course\_id}\,(\sigma_{semester=\text{“Spring”}\ \wedge\ year=2010}\,(section))$$

course_id	*sec_id*	*semester*	*year*	*building*	*room_number*	*time_slot_id*
BIO-101	1	Summer	2009	Painter	514	B
BIO-301	1	Summer	2010	Painter	514	A
CS-101	1	Fall	2009	Packard	101	H
CS-101	1	Spring	2010	Packard	101	F
CS-190	1	Spring	2009	Taylor	3128	E
CS-190	2	Spring	2009	Taylor	3128	A
CS-315	1	Spring	2010	Watson	120	D
CS-319	1	Spring	2010	Watson	100	B
CS-319	2	Spring	2010	Taylor	3128	C
CS-347	1	Fall	2009	Taylor	3128	A
EE-181	1	Spring	2009	Taylor	3128	C
FIN-201	1	Spring	2010	Packard	101	B
HIS-351	1	Spring	2010	Painter	514	C
MU-199	1	Spring	2010	Packard	101	D
PHY-101	1	Fall	2009	Watson	100	A

course_id
CS-101

Set Difference Operation

- The set-difference operation allows us to find tuples that are in one relation but are not in another.
- Notation $r - s$
- Set differences must be taken between **compatible** relations.
  - r and s must have the same arity
  - attribute domains of r and s must be compatible
- Example: to find all courses taught in the Fall 2009 semester, but not in the Spring 2010 semester

$$\Pi_{course\_id}(\sigma_{semester=\text{“Fall”} \wedge year=2009}(section)) - \Pi_{course\_id}(\sigma_{semester=\text{“Spring”} \wedge year=2010}(section))$$

course_id	*sec_id*	*semester*	*year*	*building*	*room_number*	*time_slot_id*
BIO-101	1	Summer	2009	Painter	514	B
BIO-301	1	Summer	2010	Painter	514	A
CS-101	1	Fall	2009	Packard	101	H
CS-101	1	Spring	2010	Packard	101	F
CS-190	1	Spring	2009	Taylor	3128	E
CS-190	2	Spring	2009	Taylor	3128	A
CS-315	1	Spring	2010	Watson	120	D
CS-319	1	Spring	2010	Watson	100	B
CS-319	2	Spring	2010	Taylor	3128	C
CS-347	1	Fall	2009	Taylor	3128	A
EE-181	1	Spring	2009	Taylor	3128	C
FIN-201	1	Spring	2010	Packard	101	B
HIS-351	1	Spring	2010	Painter	514	C
MU-199	1	Spring	2010	Packard	101	D
PHY-101	1	Fall	2009	Watson	100	A

course_id
CS-347
PHY-101

The Assignment Operation

- It is convenient at times to write a relational-algebra expression by assigning parts of it to temporary relation variables.
- The assignment operation is denoted by $\leftarrow$ and works like assignment in a programming language.
- Example: Find all instructor in the "Physics" and Music department.

$$Physics \leftarrow \sigma_{dept\_name="Physics"}(instructor)$$

$$Music \leftarrow \sigma_{dept\_name="Music"}(instructor)$$

$$Physics \cup Music$$

- With the assignment operation, a query can be written as a sequential program consisting of a series of assignments followed by an expression whose value is displayed as the result of the query.

Equivalent Queries

- There is more than one way to write a query in relational algebra.
- Example: Find instructors in the Physics department with salary greater than 90,000
  - Query 1

$$\sigma_{dept\_name=\text{“Physics”} \wedge salary > 90{,}000}(instructor)$$

  - Query 2

$$\sigma_{dept\_name=\text{“Physics”}}(\sigma_{salary > 90.000}(instructor))$$

- The two queries are not identical;
- they arehowever, equivalent –
  - they give the same result on any database.

ID	*course_id*	*sec_id*	*semester*	*year*
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009
32343	HIS-351	1	Spring	2010
45565	CS-101	1	Spring	2010
45565	CS-319	1	Spring	2010
76766	BIO-101	1	Summer	2009
76766	BIO-301	1	Summer	2010
83821	CS-190	1	Spring	2009
83821	CS-190	2	Spring	2009
83821	CS-319	2	Spring	2010
98345	EE-181	1	Spring	2009

Equivalent Queries

- There is more than one way to write a query in relational algebra.
- Another Example: Find information about courses taught by instructors in the Physics department
  - Query 1

$$\sigma_{dept\_name=\text{“Physics”}}(instructor \bowtie_{instructor.ID = teaches.ID} teaches)$$

  - Query 2

$$(\sigma_{dept\_name=\text{“Physics”}}(instructor)) \bowtie_{instructor.ID = teaches.ID} teaches$$

- The two queries are not identical; they are, however, equivalent -- they give the same result on any database.

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

ID	*course_id*	*sec_id*	*semester*	*year*
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009
32343	HIS-351	1	Spring	2010
45565	CS-101	1	Spring	2010
45565	CS-319	1	Spring	2010
76766	BIO-101	1	Summer	2009
76766	BIO-301	1	Summer	2010
83821	CS-190	1	Spring	2009
83821	CS-190	2	Spring	2009
83821	CS-319	2	Spring	2010
98345	EE-181	1	Spring	2009

Relational Algebra Exercises

employee (*person_name*, *street*, *city*)
works (*person_name*, *company_name*, *salary*)
company (*company_name*, *city*)

- Q1: Find the names of all employees who live in city “Miami”.

$$\Pi_{person\_name}\ (\sigma_{city = \text{“Miami”}}\ (employee))$$

- Q2: Find the names of all employees whose salary is greater than $100,000.

$$\Pi_{person\_name}\ (\sigma_{salary > 100000}\ (employee \bowtie works))$$

- Q3: Find the names of all employees who live in “Miami” and whose salary is greater than $100,000.

$$\Pi_{person\_name}\ (\sigma_{city = \text{“Miami”} \wedge salary > 100000}\ (employee \bowtie works))$$

Relational Algebra Exercises

branch(*branch_name*, *branch_city*, *assets*)
customer (*customer_name*, *customer_street*, *customer_city*)
loan (*loan_number*, *branch_name*, *amount*)
borrower (*customer_name*, *loan_number*)
account (*account_number*, *branch_name*, *balance*)
depositor (*customer_name*, *account_number*)

- Q4: Find the names of all branches located in “Chicago”.

$$\Pi_{branch\_name} (\sigma_{branch\_city = \text{“Chicago”}} (branch))$$

- Q5: Find the names of all borrowers who have a loan in branch “Downtown”.

$$\Pi_{ID} (\sigma_{branch\_name = \text{“Downtown”}} (borrower \bowtie loan))$$

Relational Algebra Exercises

PUBLISHER

publisher_id	publisher_name	phone

AUTHOR

author_id	author_name	email

BOOK

ISBN	publisher_id	author_id	title

QUERY : Find titles of books that were published by the publisher id#20.

RA Query: Selection; Projection

$$\pi_{\text{title}} (\sigma_{\text{publisher\_id} = 20} (\text{BOOK}))$$

Relational Algebra Exercises

PUBLISHER

publisher_id	publisher_name	phone

AUTHOR

author_id	author_name	email

BOOK

ISBN	publisher_id	author_id	title

QUERY : Find all book titles, along with their author name and publisher name.

RA Query

$$\pi \text{ title, author\_name, publisher\_name } (\text{BOOK} \bowtie \text{AUTHOR} \bowtie \text{PUBLISHER})$$

Summary of Relational Algebra Operators

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 (regardless of whether they 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.