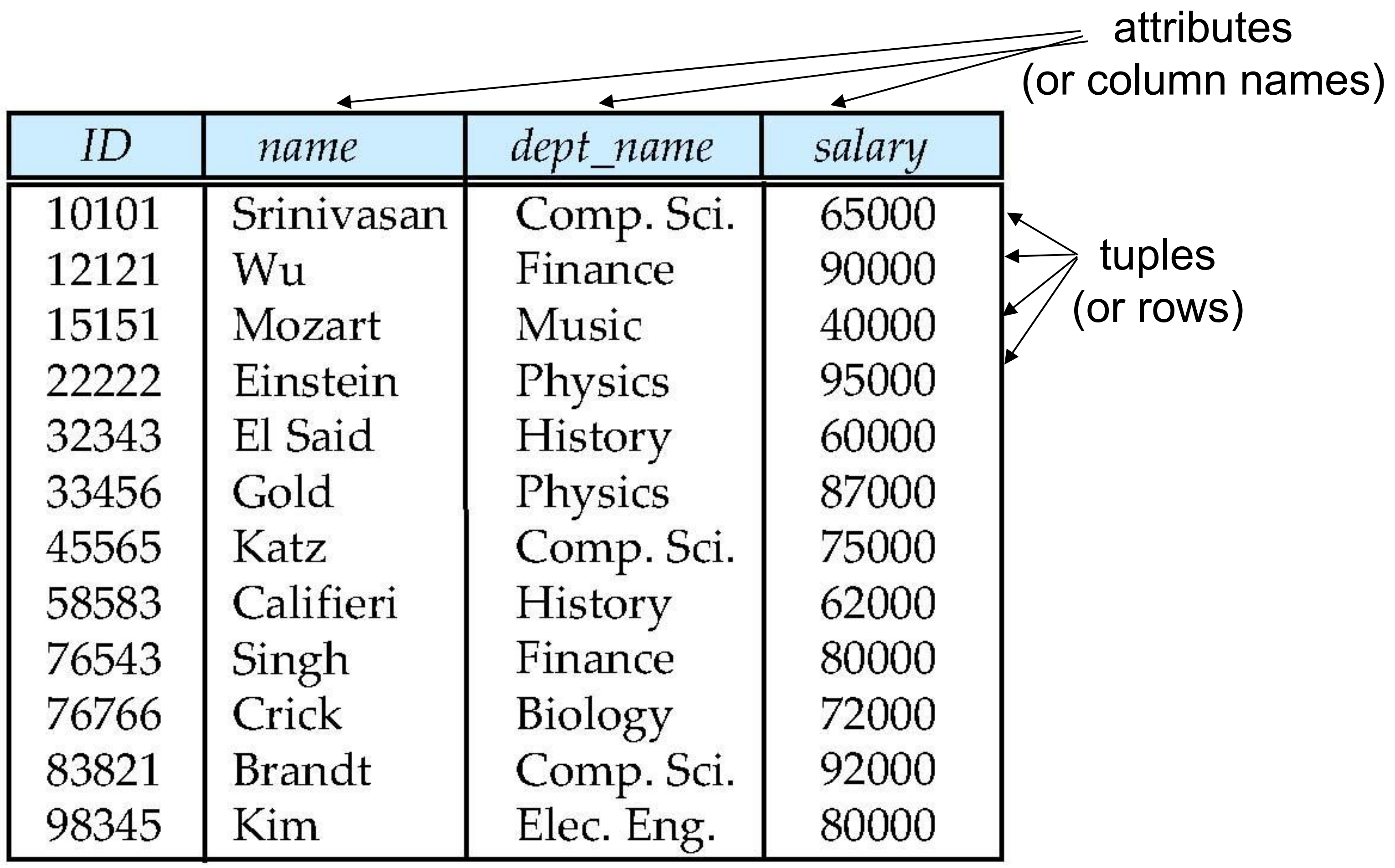
Chapter 2: Intro to Relational Model

Database System Concepts, 7th Ed.

Answer the following question

- In the context of this book, which one of the following is a relation (multiple selection):
  - 1) Your pet dog and you relationship
  - 2) Your bank account and you relationship
  - 3) All the members in a football team relation
  - 4) All the students currently attending this class relation

Example of a Relation


attributes
(or column names)
tuples
(or 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

- Order of tuples is irrelevant (tuples may be stored in an arbitrary order)

Relation Schema and Instance

- $A_1, A_2, \ldots, A_n$ are *attributes*
- $R = (A_1, A_2, \ldots, A_n)$ is a *relation schema*

  Example:

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

- Formally, given sets $D_1, D_2, \ldots. D_n$ a **relation** ***r*** is a subset of

  $D_1 \times D_2 \times \ldots \times D_n \xrightarrow{\qquad}$ 取值范围

  Thus, a relation is a set of *n*-tuples $(a_1, a_2, \ldots, a_n)$ where each $a_i \in D_i$
  - The subset is not always rational, especial temporary relation
- Naming rule: Upper case “R” is a schema, lower case “r” is an instance. r(R) refers to relation instance “r” based on the schema “R”
  - both R and r are sets
  - R is a set of attributes
  - r is a set of tuples
- Table is the an alternative concept to relation instance

Domain of Attribute

- The set of allowed values for each attribute is called the **domain** of the attribute
- Attribute values are (normally) required to be **atomic**; that is, indivisible.
  - Strictly indivisible example: store first name and last name separately, first name: “三”，last name：“张”
  - If do not query by last name or first name frequently, can store the full name in one attribute, name: “张三”
  - Non atomic examples: two telephone numbers stored in one attribute; all the information of a book stored in one attribute, including information of author, title, publisher, etc.
- The special value ***null*** $\xrightarrow{}$ 不知道 is a member of every domain. Null means the values is unknows or does not exist. Example: domain of Semester: Spring, Summer, Autumn, Winter, null; domain of Grade: integer 0 to 100, and null
- The null value causes complications in the definition of many operations

Database

- A database consists of multiple relations
- Information about an enterprise is broken up into parts

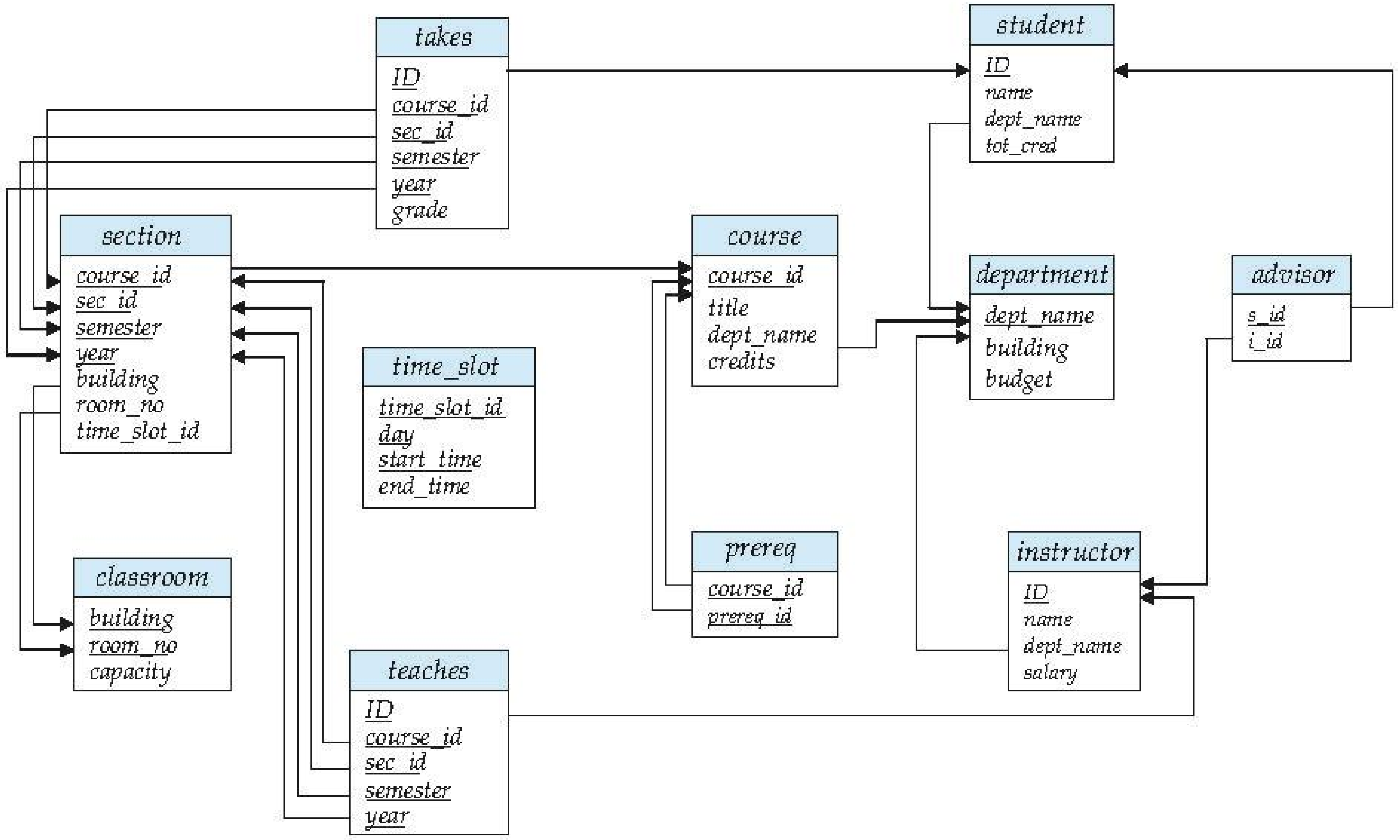
  instructor
  student
  advisor

- Bad design:
  univ (*instructor -ID, name, dept_name, salary, student_Id*, ..)
  results in
  - repetition of information (e.g., two students have the same instructor)
  - the need for null values (e.g., represent an student with no advisor)
- Normalization theory deals with how to design "good" relational schemas
- Database schema -- is the logical structure of the database.
- Database instance -- is a snapshot of the data in the database at a given instant in time.

Keys

- Let $K \subseteq R$ $\longrightarrow$ 超级码
- K is a **superkey** of R if values for K are sufficient to identify a unique tuple of each possible relation r
  - Example: {*ID*} and {ID,name} are both superkeys of *instructor.*
- Superkey K is a **candidate key** if the number of attributes in K is minimal (If we remove one more attribute in K, K will not be a superkey) $\longrightarrow$ 候选码
  - Example: {*ID*} is a candidate key for *Instructor*
- Candidate key can have more than one attribute
  - Example*: grade(student_id, course_id, grade)*
- One of the candidate keys is selected to be the **primary key**.
  - Exampe: student(student_id, name, mobile_phone, email), how many candidate keys, which one will be the primary key
- **Foreign key** constraint: Value in one relation must appear in another
  - **Referencing** relation ；**Referenced** relation
  - Example: *dept_name* in *instructor* is a foreign key from *instructor* referencing *department*

Schema Diagram for University Database



Relational Algebra

- Procedural language, but not a traditional programming language
- Input and output are all relations
- Part One: Relational Algebra – Basic Operators, mainly:
  - For one relation:
    - select: σ 取行
    - project: $\prod$ 取列
    - rename: ρ
  - For two relations:
    - union: $\cup$
    - set difference: –
    - Cartesian product: x 乘法
- Part two: Relational Algebra - Additional Operations 除法:可以用前面的来表达
  - Operations that can be expressed with basic operations, for example: intersection can be expressed with set difference
- Part three: Relational Algebra - Extended Operations 无法用前面的表达
  - Operations that can not be expressed with basic operations. Mainly aggregation functions

Relational Algebra – Basic Operators

- Six basic operators
  - select: σ
  - project: $\prod$
  - union: $\cup$
  - set difference: –
  - Cartesian product: x
  - rename: ρ

Select Operation – Example

- Relation r

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

- $\underset{\text{运算符}}{\underset{\downarrow}{\sigma}}{}_{\underset{\text{条件}}{\underset{\downarrow}{A=B \wedge D > 5}}}\underset{\text{relation instance}}{\underset{\downarrow}{(r)}}$

A	B	C	D
α	α	1	7
β	β	23	10

Select Operation

- The **select** operation selects tuples that satisfy a given predicate.
- Notation: $\sigma_{p}(r)$
- p is called the **selection predicate**
- Defined as:

$$\sigma_p(\mathbf{r}) = \{t \mid t \in r \textbf{ and } p(t)\}$$

Where p is a formula in propositional calculus consisting of **terms** connected by : $\wedge$ (**and**), $\vee$ (**or**), $\neg$ (**not**)
Each **term** is one of:

<attribute> *op* <attribute> or <constant>

where *op* is one of: $=, \neq, >, \geq. <. \leq$

- Example of selection:

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

$$\sigma_{dept\_name=\text{“Physics”} \wedge \text{ salary} > 50000}\ (instructor)$$

Project Operation – Example

- Relation *r*:

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

- $\Pi_{A,C}(r)$

A	C
α	1
α	1
β	1
β	2

=

A	C
α	1
β	1
β	2

消除冗余行

Project Operation

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

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

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
- Duplicate rows removed from result, since relations are sets
- Example: To eliminate the *dept_name* attribute of *instructor*

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

$$\prod_{course\_id} (\sigma_{semester=\text{“Fall”}\ \wedge\ year=2009} (section))$$

Union Operation – Example

- Relations *r, s:*

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- r ∪ s:

A	B
α	1
α	2
β	1
β	3

Union Operation

- The union operation allows us to combine two relations
- Notation: $r \cup s$
- Defined as:

$$r \cup s = \{t \mid t \in r \text{ or } t \in s\}$$

- For $r \cup s$ to be valid.
  1. *r, s* must have the *same* **arity** (same number of attributes)
  2. The attribute domains must be **compatible** (example: 2nd column of *r* deals with the same type of values as does the 2nd column of *s*)

有相同值域

- Duplicate rows removed from result
- Example: to find all courses taught in the Fall 2009 semester, or in the Spring 2010 semester, or in both

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

Set Difference – Example

- Relations *r*, *s*:

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- *r* – *s*:

A	B
α	1
β	1

Set Difference Operation

- The set-difference operation allows us to find tuples that are in one relation but are not in another.
- Notation $r - s$
- Defined as:

$$r - s = \{t \mid t \in r \textbf{ and } t \notin s\}$$

- 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

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

Cartesian-Product Operation – Example

- Relations *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

- *r* x *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 Operation

- The Cartesian-product operation (denoted by X) allows us to combine information from any two relations.
- Notation *r* x *s*
- Defined as:

$$r \times s = \{t\, q \mid t \in r \textbf{ and } q \in s\}$$

- Assume that attributes of r(R) and s(S) are disjoint. (That is, $R \cap S = \varnothing$, or, no attributes with the same name).
- If attributes of *r(R)* and *s(S)* are not disjoint, then renaming must be used, e.g, instructor.ID, teaches.ID.
- The resulting relation may be huge
- In real application, most of the tuples in the resulting relation may be useless. Followed by "σ", "x" may be the most frequently used operation in database applications, as database consists of many interrelated relations.

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:

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

  - 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:

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

		_			_	_		

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

$$\sigma_{\mathit{instructor.id} = \mathit{teaches.id}} \, (\mathit{instructor} \;\mathrm{x}\; \mathit{teaches}\,))$$

- Can equivalently be written as

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

Composition of Operations

- Can build expressions using multiple operations
- Example: $\sigma_{A=C}(r \times s)$
- $r \times s$

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

- $\sigma_{A=C}(r \times s)$

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

$$\Pi_{instructor.name, course\_id}\,(\sigma_{\;instructor.ID=teaches.ID}\;(instructor \times teaches))$$

Composition of Operations – Cont.

- Find the names of all instructors in the Physics department, along with the *course_id* of all courses they have taught
  - Query 1

    $$\Pi_{instructor.name, course\_id}(\sigma_{dept\_name=\text{“Physics”}}(\sigma_{instructor.ID=teaches.ID}(instructor \times teaches)))$$

  - Query 2

    $$\Pi_{instructor.name, course\_id}(\sigma_{instructor.ID=teaches.ID}(\sigma_{dept\_name=\text{“Physics”}}(instructor) \times teaches))$$

Rename Operation

- Allows us to refer to a relation by more than one name.
- Example:

$$\rho_x(E)$$

returns the expression *E* under the name *X*

- If a relational-algebra expression *E* has arity *n*, then

$$\rho_{x(A_1, A_2, \ldots, A_n)}(E)$$

returns the result of expression *E* under the name *X*, and with the attributes renamed to $A_1, A_2, \ldots, A_n$.

Rename Operation-Example

- Find instructors whose salary is greater than some instructors (at least one instructor) in physics department
  - $\Pi_{Instructor.ID}(\sigma_{instructor.salary > d.salary}$ $(instructor \times \rho_d (\sigma_{department = \text{“physics”}}(instructor))))$ 两张相同的表合在一起时

Formal Definition

- A basic expression in the relational algebra consists of either one of the following:
  - A relation in the database
  - A constant relation
- Let E_1 and E_2 be relational-algebra expressions; the following are all relational-algebra expressions:
  - $E_1 \cup E_2$
  - $E_1 – E_2$
  - E_1 x E_2
  - $\sigma_p (E_1)$, P is a predicate on attributes in E_1
  - $\prod_s(E_1)$, S is a list consisting of some of the attributes in E_1
  - $\rho_x (E_1)$, x is the new name for the result of E_1

Relational Algebra - Additional Operations

We define additional operations that do not add any power to the relational algebra, but that simplify common queries.

- Set intersection
- Natural join
- Assignment
- Outer join
- Division

Set-Intersection Operation – Example

- Relation *r, s*:

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

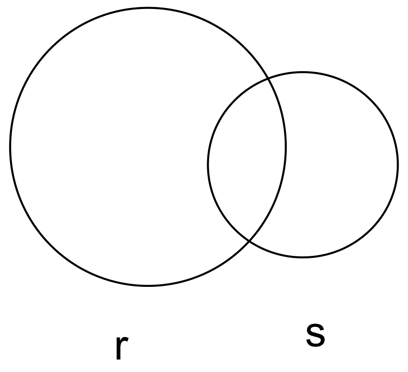
s

- $r \cap s$

A	B
α	2

Set-Intersection Operation

- The set-intersection operation allows us to find tuples that are in both the input relations.
- Notation: $r \cap s$
- Defined as:
- $r \cap s = \{ t \mid t \in r \textbf{ and } t \in s \}$
- Assume:
  - *r*, *s* have the *same arity*
  - attributes of *r* and *s* are compatible
- Note: $r \cap s = r - (r - s)$



Natural-Join Operation

- Notation: $r \bowtie s$
- Let *r* and *s* be relations on schemas *R* and *S* respectively. Then, $r \bowtie s$ is a relation on schema $R \cup S$ obtained as follows:
  - Consider each pair of tuples t_r from *r* and t_s from *s*.
  - If t_r and t_s have the same value on each of the attributes in $R \cap S$, add a tuple *t* to the result, where
    - *t* has the same value as t_r on *r*
    - *t* has the same value as t_s on *s*
- Example:

  $R = (A, B, C, D)$

  $S = (E, B, D)$

  - Result schema = (A, B, C, D, E)
  - $r \bowtie s$ is defined as:

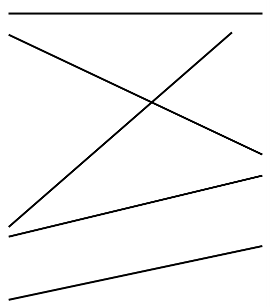
    $$\Pi_{r.A,\ r.B,\ r.C,\ r.D,\ s.E}\ (\sigma_{r.B = s.B \wedge r.D = s.D}\ (r \times s))$$

Natural Join Example

- Relations 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

- r ⋈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

$$\Pi_{instructor.name,course\_id}(\,instructor \bowtie teaches)$$

Natural Join Properties

- Find the names of all instructors in the Comp. Sci. department together with the course titles of all the courses that the instructors teach
  - $\Pi_{name,\ title}\ (\sigma_{dept\_name=\text{“Comp. Sci.”}}\ (instructor \bowtie teaches \bowtie course))$
- Natural join is associative
  - $(instructor \bowtie teaches) \bowtie course$ is equivalent to $instructor \bowtie (teaches \bowtie course)$
- Natural join is commutative
  - $instruct \bowtie teaches$ is equivalent to $teaches \bowtie instructor$

Assignment Operation

- The assignment operation ($\leftarrow$) provides a convenient way to express complex queries.
  - Can assign temporary result to a temporary relation variable
  - A complex query consists of a series of assignments followed by an expression whose value is displayed as a result of the query.
- Example:

$$d \leftarrow \sigma_{department\ =\ \text{“}physics\text{”}}(\ instructor)$$

先得出一个临时表

$$result = \Pi_{\text{instructor.}ID}(\sigma_{instructor.salary\ >d.salary}(\text{instructor x d}))$$

Outer Join

- An extension of the join operation that avoids loss of information.
- Computes the join and then adds tuples form one relation that does not match tuples in the other relation to the result of the join.
- Uses *null* values:
  - *null* signifies that the value is unknown or does not exist

Outer Join – Example

- Relation *instructor1*

ID	*name*	*dept_name*
10101	Srinivasan	Comp. Sci.
12121	Wu	Finance
15151	Mozart	Music

- Relation *teaches1*

ID	*course_id*
10101	CS-101
12121	FIN-201
76766	BIO-101

Outer Join – Example

- Natural Join

  instructor ⋈ *teaches*

ID	*name*	*dept_name*	*course_id*
10101	Srinivasan	Comp. Sci.	CS-101
12121	Wu	Finance	FIN-201

- Left Outer Join

  instructor ⟕ *teaches*

ID	*name*	*dept_name*	*course_id*
10101	Srinivasan	Comp. Sci.	CS-101
12121	Wu	Finance	FIN-201
15151	Mozart	Music	*null*

Outer Join – Example

- Right Outer Join

  instructor ⟖ *teaches*

ID	*name*	*dept_name*	*course_id*
10101	Srinivasan	Comp. Sci.	CS-101
12121	Wu	Finance	FIN-201
76766	null	null	BIO-101

- Full Outer Join

  instructor ⟗ *teaches*

ID	*name*	*dept_name*	*course_id*
10101	Srinivasan	Comp. Sci.	CS-101
12121	Wu	Finance	FIN-201
15151	Mozart	Music	*null*
76766	null	null	BIO-101

Outer Join using Joins

- Outer join can be expressed using basic operations
  - e.g. $r ⟕ s$ can be written as

    $(r \bowtie s)\ \text{U}\ (r - \prod_R(r \bowtie s)\ \text{x}\ \{(null, \ldots, null)\}$

Division Operator

- Given relations r(R) and s(S), such that $S \subset R$, $r \div s$ is the largest relation t(R-S) such that

  $$t \times s \subseteq r$$

- E.g. let $r(ID, course\_id) = \Pi_{ID,\ course\_id}(takes)$ and

  $s(course\_id) = \Pi_{course\_id}(\sigma_{dept\_name=\text{“Biology”}}(course))$

  then $r \div s$ gives us students who have taken all courses in the Biology department

Division Using Basic Operations

- Can write $r \div s$ as

$$temp1 \leftarrow \Pi_{R-S}(r)$$
$$temp2 \leftarrow \Pi_{R-S}((temp1 \times s) - \Pi_{R-S,S}(r))$$
$$result = temp1 - temp2$$

  - The result to the right of the $\leftarrow$ is assigned to the relation variable on the left of the $\leftarrow$.

Relational Algebra - Extended Operations

- Generalized Projection
- Aggregate Functions

Generalized Projection

- Extends the projection operation by allowing arithmetic functions to be used in the projection list.

$$\Pi_{F_1, F_2, \ldots, F_n}(E)$$

- *E* is any relational-algebra expression
- Each of F_1, F_2, …, F_n are arithmetic expressions involving constants and attributes in the schema of *E*.
- Given relation *instructor(ID, name, dept_name,* salary) where salary is annual salary, get the same information but with monthly salary

$$\Pi_{ID,\ name,\ dept\_name,\ salary/12}\ (instructor)$$

Aggregate Functions and Operations

- **Aggregation function** takes a collection of values and returns a single value as a result.

  avg: average value
  min: minimum value
  max: maximum value
  sum: sum of values
  count: number of values

- **Aggregate operation** in relational algebra

$$_{G_1, G_2, \ldots, G_n}\ \mathcal{G}_{\ F_1(A_1), F_2(A_2, \ldots, F_n(A_n)}(E)$$

E is any relational-algebra expression

- G_1, G_2 …, G_n is a list of attributes on which to group (can be empty)
  - On which to group: to do the aggregation based on a subset of E. For any two tuples t1 and t2 in the subset, t1.G1 = t2.G1, t1.G2 = t2.G2, ..., t1.Gn=t2.Gn
- Each F_i is an aggregate function
- Each A_i is an attribute name

Aggregate Operation – Example

- Relation *r*:

A	*B*	*C*
α	α	7
α	β	7
β	β	3
β	β	10

- $\mathcal{G}_{\mathbf{sum(c)}}(r)$

sum(*c*)
27

- $_{A}\mathcal{G}_{\mathbf{sum(c)}}(r)$; $_{A,B}\mathcal{G}_{\mathrm{sum(c)}}(r)$

A	*sum(c)*
α	14
β	13

A	*B*	*sum(c)*
α	α	7
α	β	7
β	β	13

Aggregate Operation – Example

- Find the average salary in each department

  $_{dept\_name}\,\mathcal{G}\,_{\mathbf{avg}(salary)}\,(instructor)$

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

dept_name	*avg_salary*
Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000

Aggregate Functions (Cont.)

- Result of aggregation does not have a name
  - Can use rename operation to give it a name
  - For convenience, we permit renaming as part of aggregate operation

$$_{dept\_name}\ \mathcal{G}\ _{\textbf{avg}(salary)\ \textbf{as}\ avg\_sal}\ (instructor)$$

Modification of the Database

- The content of the database may be modified using the following operations:
  - Deletion
  - Insertion
  - Updating
- All these operations can be expressed using the assignment operator

Null Values

- It is possible for tuples to have a null value, denoted by *null*, for some of their attributes
- *null* signifies an unknown value or that a value does not exist.
- The result of any arithmetic expression involving *null* is *null.*
  - *5 + null = null*
- Aggregate functions simply ignore null values (as in SQL)
  - avg, min, max, sum, count(c): ignore null
  - count(*): count the number of records, if there are null values in some records, they will still be counted
- For duplicate elimination and grouping, null is treated like any other value, and two nulls are assumed to be the same (as in SQL)

Null Values

- Comparisons with null values return the special truth value: *unknown*
- Three-valued logic using the truth value *unknown*:
  - OR: (*unknown* **or** *true*) = *true*,
    (*unknown* **or** *false*) = *unknown*
    (*unknown* **or** *unknown*) = *unknown*
  - AND: (*true* **and** *unknown*) = *unknown,*
    (*false* **and** *unknown*) = *false,*
    (*unknown* **and** *unknown*) = *unknown*
  - NOT*:* (**not** *unknown*) = *unknown*
  - In SQL "*P* **is unknown**" evaluates to true if predicate *P* evaluates to *unknown*
- Result of select predicate is treated as *false* if it evaluates to *unknown*

End of Chapter 2

Exercise 7,12,13,15