

Relational Model

Outline

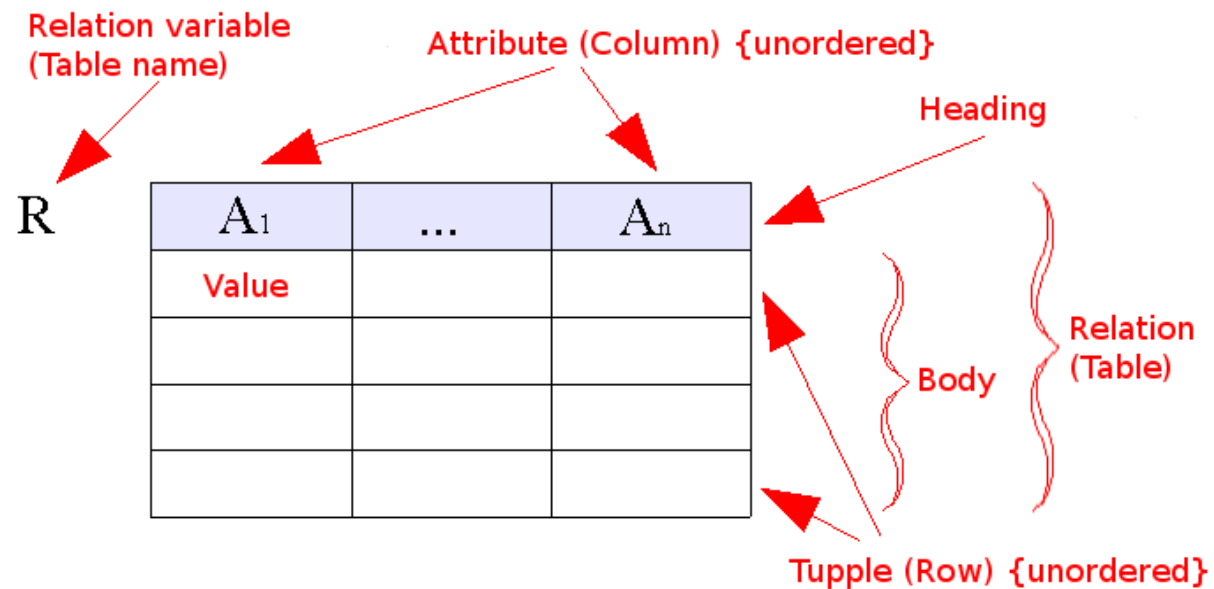
- Structure of Relational Databases
- Database Schema
- Keys
- Schema Diagrams
- Relational Query Languages
- The Relational Algebra

Relational Model Foundation

- Relational Model of data is based on the concept of RELATION
- A Relation is a Mathematical concept based on idea of SETS
- The strength of the relational approach to data management comes from the formal foundation provided by the theory of relations
- The model was first proposed by Dr. E.F. Codd of IBM in 1970

Relational Model (1970)

- Data as set of tables
- Having constraints and relationships
- Oracle, DB2, Sybase



Some Terms

Informal Name

- Table
- Row or Record
- Column or Field
- No. of Rows
- No. of Columns
- Unique Identifier
- Legal Values Set

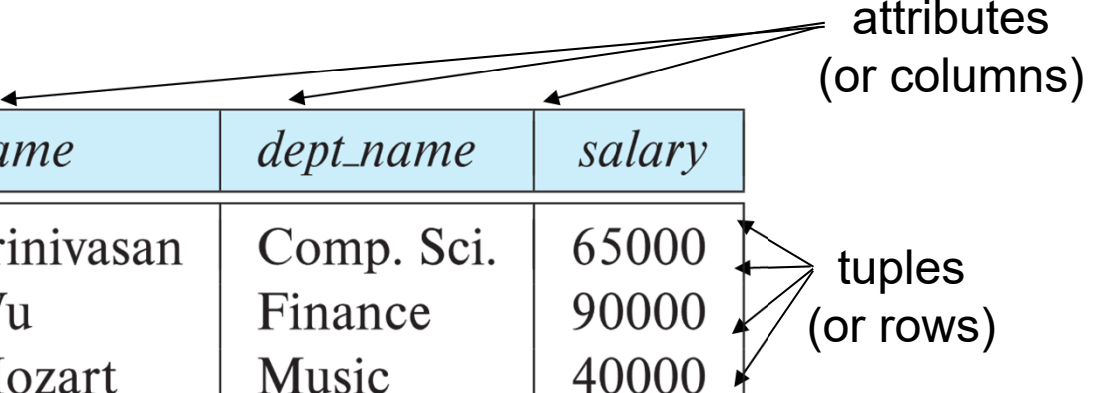
Formal Name

- Relation
- Tuple
- Attribute
- Cardinality
- Degree or Arity
- Primary key
- Domain

Relational Database: Definitions

- Relational database: a set of relations
- Relation: made up of 2 parts:
 - Instance : a table, with rows and columns.
#Rows = cardinality, #fields = degree / arity.
 - Schema : specifies name of relation, plus name and type of each column.
 - E.G. Students(sid: string, name: string, login: string, age: integer, gpa: real).
- Can think of a relation as a set of rows or tuples (i.e., all rows are distinct).

Example of a *Instructor* Relation



<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>
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

Relation Schema and Instance

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

Example:

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

- A relation instance r defined over schema R is denoted by $r(R)$.
- The current values a relation are specified by a table
- An element t of relation r is called a *tuple* and is represented by a *row* in a table

Attributes

- 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
- The special value ***null*** is a member of every domain. Indicated that the value is “unknown”
- The null value causes complications in the definition of many operations

Null Values

- Comparisons with null values return the special truth value *null* (*unknown*)
- Three-valued logic using the truth value *null*:

- **OR:** (*null* **OR** *true*) = *true*
 (*null* **OR** *false*) = *null*
 (*null* **OR** *null*) = *null*
- **AND:** (*true* **AND** *null*) = *null*
 (*false* **AND** *null*) = *false*
 (*null* **AND** *null*) = *null*
- **NOT:** (**NOT** *null*) = *null*

Attributes

- An **attribute** is a characteristic
Property, data element, field, attribute
Ex: Student (name, IDNo, course credit)
- **Domain**
- **Attributes can be**
 - Atomic (simple) vs. Composite
 - Ex: Student ID , Student name
Where student name may consists of (first, middle, last name)
 - Single-valued vs. Multi-valued
 - Ex. number of courses registered, names of registered courses
 - Derived
 - Ex: age of a student

Relations are Unordered

- Order of tuples is irrelevant (tuples may be stored in an arbitrary order)
- Example: *instructor* relation with unordered tuples

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>
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

Database Schema

- 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.
- Example:
 - schema: *instructor (ID, name, dept_name, salary)*
 - Instance:

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>
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

Keys

- **Superkey (SK)**

- Set of one or more attributes which taken collectively identifies uniquely an entity in a relation set
- Ex: Student (student name, IDNo, CPI, program, address)
- Student relation set superkey can be (student name, IDNo)
- Superkey may contain extraneous attributes
- There can be more than one superkeys

- **Candidate Key (CK)**

- A superkey for which no subset is a superkey
- Ex: IDNo is a Candidate Key as it is minimal and uniquely identifies a student from Student relation Set
- There can be more than one candidate keys

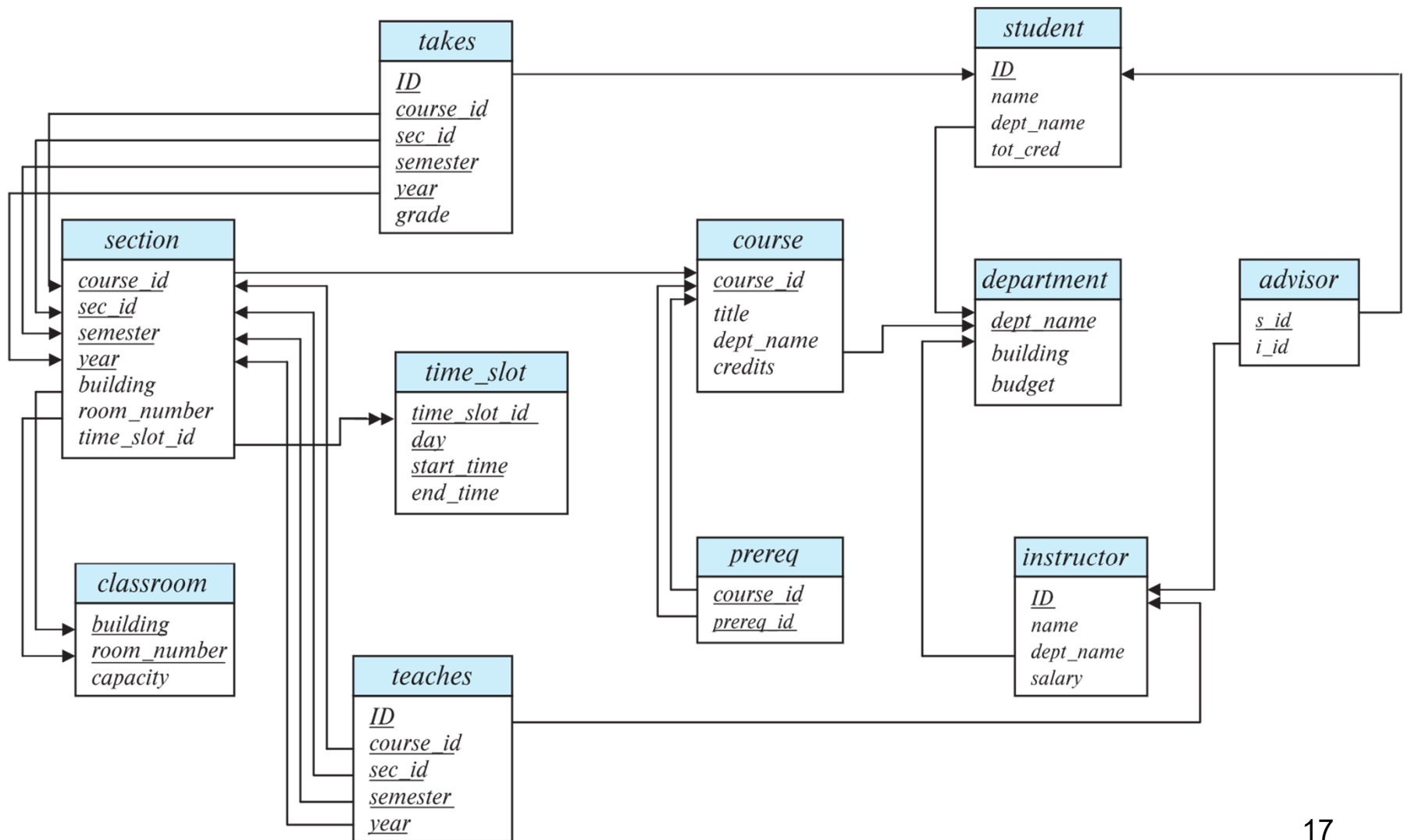
Keys

- **Primary Key (PK)**
 - Is a candidate key
 - One of the candidate key is chosen by database designer as a primary key
- **Example**
 - Telephone Book(STD Code, Telephone number, Name, Address)
 - Composite Key

Keys

- Let $K \subseteq R$
- K is a **superkey** of R if values for K are sufficient to identify a unique tuple of each possible relation $r(R)$
 - Example: $\{ID\}$ 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*
- One of the candidate keys is selected to be the **primary key**.
 - Which one?
- **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



Schema of University Database

- *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)

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 chapter on relational algebra
 - Consists of 6 basic operations

Relational Algebra

- A procedural language consisting of a set of operations that take one or two relations as input and produce a new relation as their result.
- Six basic operators
 - **select:** σ
 - **project:** Π
 - **union:** \cup
 - **set difference:** $-$
 - **Cartesian product:** \times
 - **rename:** ρ

Relational Algebra Operators

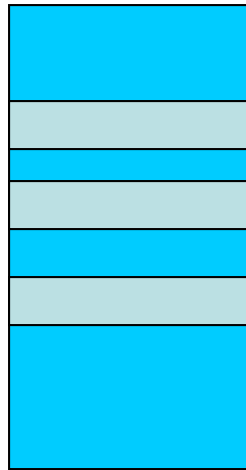
Relational Algebra consists of several groups of operations

- **Unary Relational Operations**
 - SELECT (symbol: σ (sigma))
 - PROJECT (symbol: π (pi))
 - RENAME (symbol: ρ (rho))
- **Relational Algebra Operations From Set Theory**
 - UNION (\cup), INTERSECTION (\cap), DIFFERENCE (or MINUS, $-$)
 - CARTESIAN PRODUCT (\times)
- **Binary Relational Operations**
 - JOIN (several variations of JOIN exist)
 - DIVISION
- **Additional Relational Operations**
 - OUTER JOINS, OUTER UNION
 - AGGREGATE FUNCTIONS (These compute summary of information: for example, SUM, COUNT, AVG, MIN, MAX)

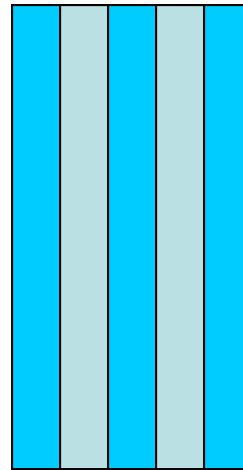
Relational Algebra

- Collection of operations on relations
- 8 operators + Rename operator
- Codd's 8 operators do not form a minimal set
- Some of them are not primitive
- Join, Intersect, & Divide can be defined in terms of other 5
- None of these 5 can be defined in terms of the remaining 4

Restrict & Project

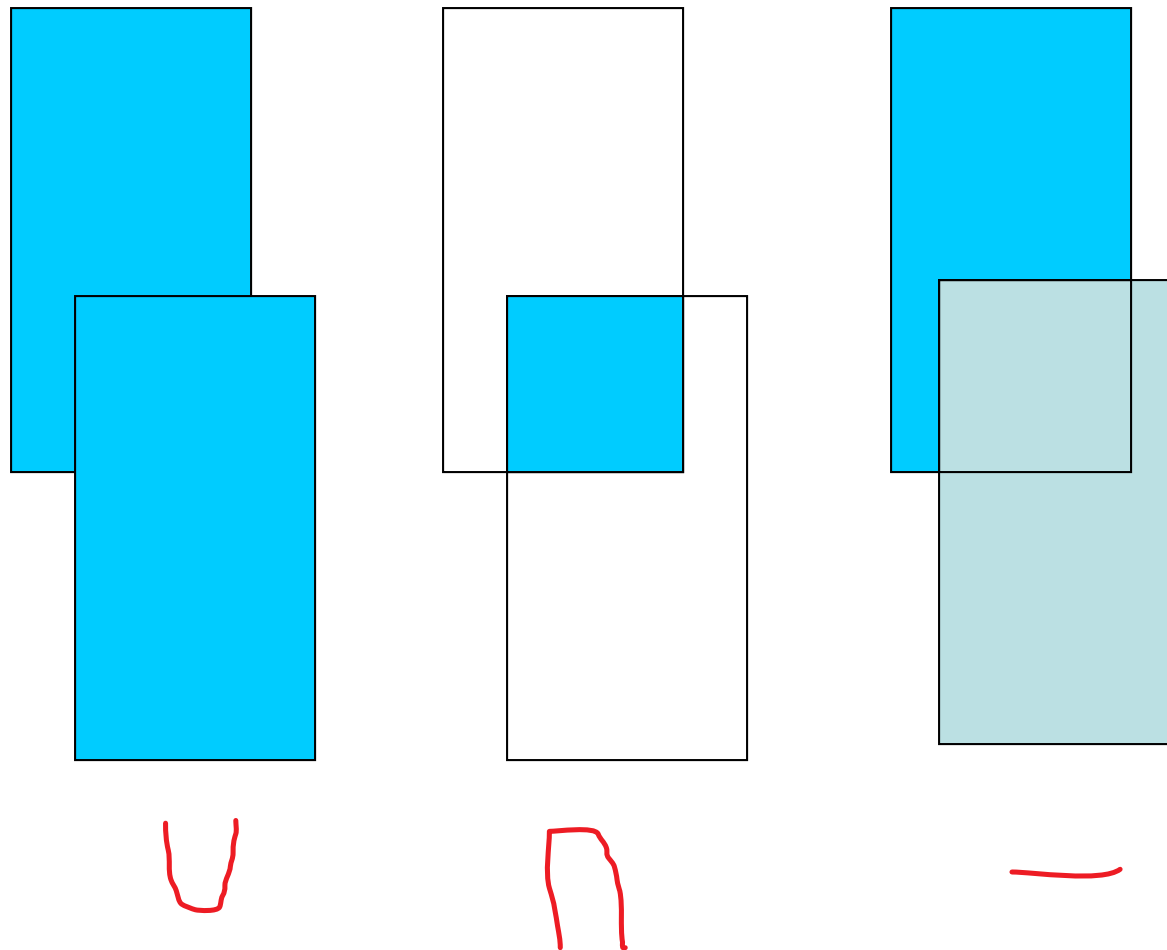


select: σ



project: Π

Union, Intersection & Difference



Instructor

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>
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 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

$$\sigma_{dept_name="Physics"}(instructor)$$

- **Result**

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>
22222	Einstein	Physics	95000
33456	Gold	Physics	87000

Select Operation

- 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:

$\sigma_{dept_name="Physics" \wedge salary > 90,000} (instructor)$

- The 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:

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

where A_1, A_2, \dots, A_k 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

Project Operation Example

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

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

- Result:**

<i>ID</i>	<i>name</i>	<i>salary</i>
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 a relation and therefore relational-algebra operations can be composed together into a **relational-algebra expression**.
- *Find the names of all instructors in the Physics department.*

$$\Pi_{name}(\sigma_{dept_name = "Physics"}(instructor))$$

- Instead of giving the name of a relation as the argument of the projection operation, we give an expression that evaluates to a relation.

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 *teaches* 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
- 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*

The *instructor X teaches* table

<i>instructor.ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>	<i>teaches.ID</i>	<i>course_id</i>	<i>sec_id</i>	<i>semester</i>	<i>year</i>
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
...
...

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.

Join Operation (Cont.)

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

<i>instructor.ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>	<i>teaches.ID</i>	<i>course_id</i>	<i>sec_id</i>	<i>semester</i>	<i>year</i>
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
12121	Wu	Finance	90000	12121	FIN-201	1	Spring	2018
15151	Mozart	Music	40000	15151	MU-199	1	Spring	2018
22222	Einstein	Physics	95000	22222	PHY-101	1	Fall	2017
32343	El Said	History	60000	32343	HIS-351	1	Spring	2018
45565	Katz	Comp. Sci.	75000	45565	CS-101	1	Spring	2018
45565	Katz	Comp. Sci.	75000	45565	CS-319	1	Spring	2018
76766	Crick	Biology	72000	76766	BIO-101	1	Summer	2017
76766	Crick	Biology	72000	76766	BIO-301	1	Summer	2018
83821	Brandt	Comp. Sci.	92000	83821	CS-190	1	Spring	2017
83821	Brandt	Comp. Sci.	92000	83821	CS-190	2	Spring	2017
83821	Brandt	Comp. Sci.	92000	83821	CS-319	2	Spring	2018
98345	Kim	Elec. Eng.	80000	98345	EE-181	1	Spring	2017

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

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

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 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 2017 semester, or in the Spring 2018 semester, or in both

$$\begin{aligned} & \Pi_{course_id} (\sigma_{semester = \text{"Fall"} \wedge year = 2017} (section)) \\ & \cup \\ & \Pi_{course_id} (\sigma_{semester = \text{"Spring"} \wedge year = 2018} (section)) \end{aligned}$$

Union Operation

- Result of:

$$\Pi_{course_id} (\sigma_{semester="Fall" \wedge year=2017} (section))$$
$$\cup$$
$$\Pi_{course_id} (\sigma_{semester="Spring" \wedge year=2018} (section))$$

<i>course_id</i>
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 2017 and the Spring 2018 semesters.

$$\begin{aligned} & \Pi_{\text{course_id}} (\sigma_{\text{semester} = \text{"Fall"} \wedge \text{year} = 2017} \\ & (\text{section})) \cap \\ & \Pi_{\text{course_id}} (\sigma_{\text{semester} = \text{"Spring"} \wedge \text{year} = 2018} \\ & (\text{section})) \end{aligned}$$

Result

<i>course_id</i>
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 2017 semester, but not in the Spring 2018 semester

$$\Pi_{course_id} (\sigma_{semester="Fall" \wedge year=2017}(section)) - \Pi_{course_id} (\sigma_{semester="Spring" \wedge year=2018}(section))$$

<i>course_id</i>
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.

$Physics \leftarrow \sigma_{dept_name = \text{“Physics”}}(instructor)$
 $Music \leftarrow \sigma_{dept_name = \text{“Music”}}(instructor)$
 $Physics \cup Music$

- 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.

The Rename Operation

- The results of relational-algebra expressions do not have a name that we can use to refer to them. The rename operator, ρ , is provided for that purpose
- The expression:

$$\rho_x (E)$$

returns the result of expression E under the name x

- Another form of the rename operation:

$$\rho_{x(A1,A2, \dots A_n)} (E)$$

Equivalent Queries

- There is more than one way to write a query in relational algebra.
- Example: Find information about courses taught by 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 are, however, equivalent -- they give the same result on any database.

Equivalent Queries

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

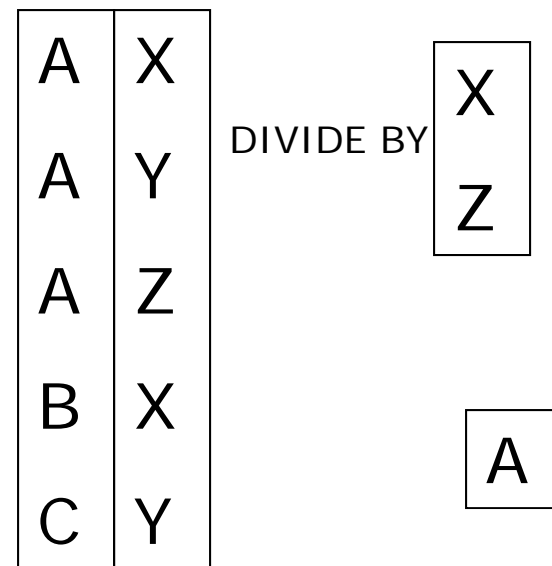
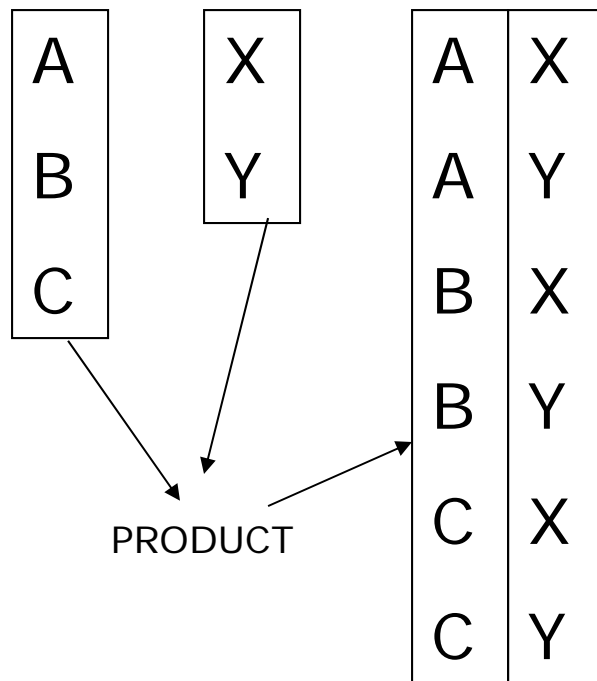
$\sigma_{dept_name="Physics"} (instructor \bowtie instructor.ID = teaches.ID \text{ teaches})$

- Query 2

$(\sigma_{dept_name="Physics"} (instructor)) \bowtie instructor.ID = teaches.ID \text{ teaches}$

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

Product & Divide



Division

S1	P1
S1	P2
S2	P3
S2	P4
S2	P1
S2	P2
S3	P2
S4	P2
S4	P4

P2	S1
	S2
	S3
	S4
P2	S2
P4	S4
P1	S2
P2	
P4	