Database System Concepts, 7th Ed.

Chapter 2: Intro to Relational Model

Morteza Zakeri, Ph.D.
(m-zakeri@live.com)

Fall 2023

Outline

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

Example of a *Instructor* Relation

attributes
(or columns)

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

tuples
(or rows)

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*)
- 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

Relations are Unordered

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

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

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:

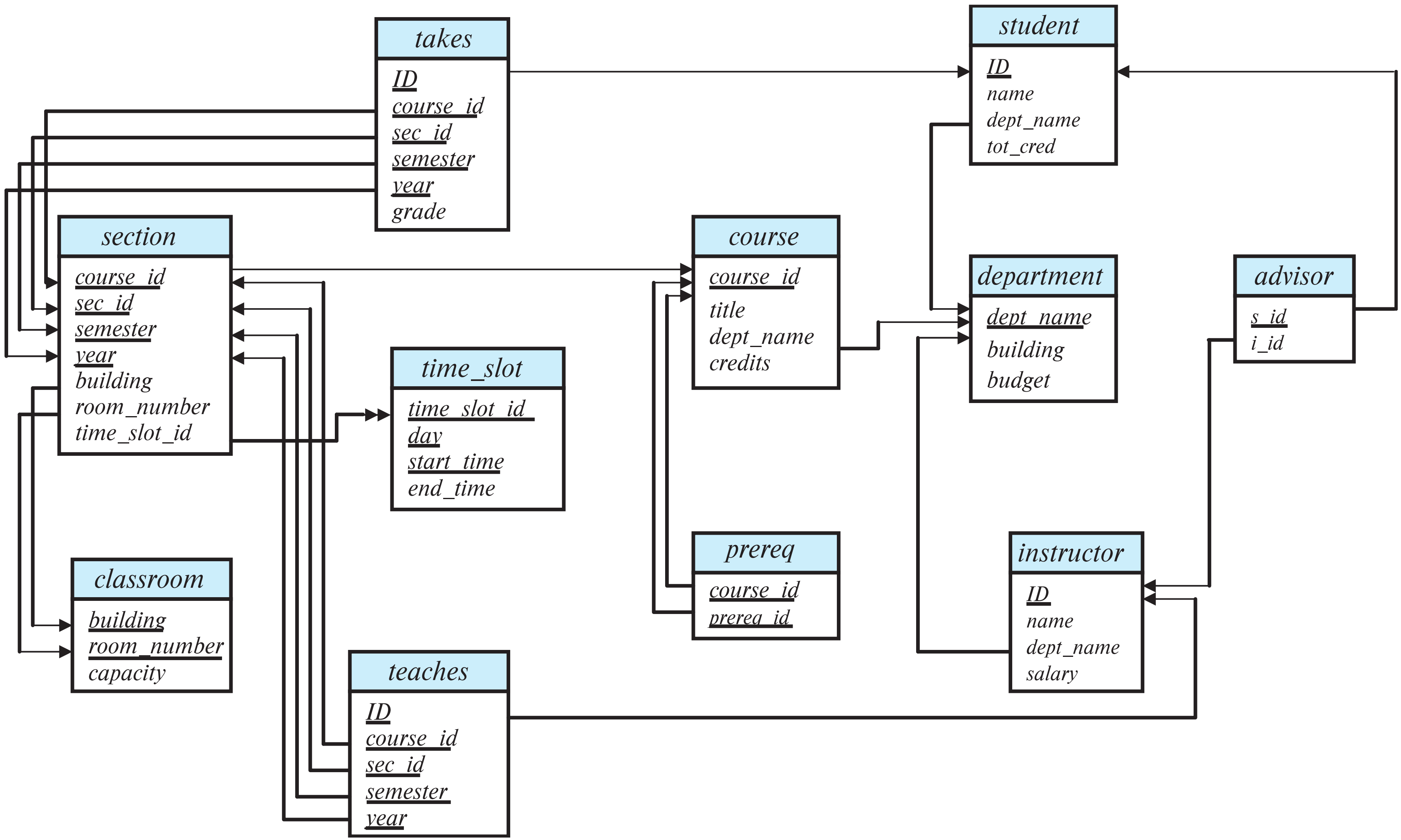
ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music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

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 i*nstructor* is a foreign key from *instructor* referencing *department*

Schema Diagram for University Database



Relational Query Languages

- Procedural versus non-procedural, or declarative
- "Pure" languages:
  - **Relational algebra**
  - Tuple relational calculus (Chapter 27)
  - Domain relational calculus (Chapter 27)
- The above 3 pure languages are equivalent in computing power
- We will concentrate in this chapter on **relational algebra**
  - Not Turing-machine equivalent
  - 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: $\prod$
  - union: $\cup$
  - set difference: –
  - Cartesian product: x
  - rename: ρ

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
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:

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

- 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:

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

  where A_1, A_2, …, 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*
- Query*:*

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

- Result:

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 and therefore of relational-algebra operations can be composed together into a **relational-algebra expression**.
- Consider the query:
  - *Find the names of all instructors in the Physics department.*

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

- 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 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 next slide)
- Since the **instructor *ID*** appears in both relations we distinguish between these attribute 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

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

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:

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

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
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

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

- Can equivalently be written as

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

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

$$\Pi_{course\_id}\,(\sigma_{semester=\text{“Fall”} \wedge year=2017}\,(section)) \cup$$
$$\Pi_{course\_id}\,(\sigma_{semester=\text{“Spring”} \wedge year=2018}\,(section))$$

Union Operation (Cont.)

- Result of:

$$\Pi_{course\_id}\,(\sigma_{\,semester=\text{“Fall”}\ \wedge\ year=2017}\,(section))\ \cup$$
$$\Pi_{course\_id}\,(\sigma_{\,semester=\text{“Spring”}\ \wedge\ year=2018}\,(section))$$

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 2017 and the Spring 2018 semesters.

$$\Pi_{course\_id} (\sigma_{semester=\text{"Fall"} \land year=2017} (section)) \cap$$
$$\Pi_{course\_id} (\sigma_{semester=\text{"Spring"} \land year=2018} (section))$$

  - Result

course_id
CS-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.

$$\Pi_{course\_id} (\sigma_{semester=\text{“Fall”} \land year=2017}(section)) \cap$$
$$\Pi_{course\_id} (\sigma_{semester=\text{“Spring”} \land year=2018}(section))$$

  - Result

course_id
CS-101

  - Note: $r \cap s = r - (r - s)$

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

$$\Pi_{course\_id}\ (\sigma_{\ semester=\text{“Fall”}\ \wedge\ year=2017}\ (section))\ -$$
$$\Pi_{course\_id}\ (\sigma_{\ semester=\text{“Spring”}\ \wedge\ year=2018}\ (section))$$

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.

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

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

$$Physics \cup Music$$

- 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:

$$\rho_{x(A1,A2, .. An)} (E)$$

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

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

- Query 2

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

- 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

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

- 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.

ADVANCED RELATIONAL ALGEBRA OPERATIONS

Natural join (⋈)

- **R ⋈ S**
- The result of the **natural join** is the set of all combinations of tuples in *R* and *S* that are **equal** on their **common attribute names**.

Employee

Name	EmpId	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales
Mary	1257	Human Resources

Dept

DeptName	Manager
Finance	George
Sales	Harriet
Production	Charles

Employee ⋈ Dept

Name	EmpId	DeptName	Manager
Harry	3415	Finance	George
Sally	2241	Sales	Harriet
George	3401	Finance	George
Harriet	2202	Sales	Harriet

Semijoin ($\ltimes$ and $\rtimes$)

- **Left semijoin $\ltimes$**
  - **R** $\ltimes$ S
  - The difference from a natural join is that columns of **S** do not appear.

Employee

Name	EmpId	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Production

Dept

DeptName	Manager
Sales	Sally
Production	Harriet

Employee $\ltimes$ Dept

Name	EmpId	DeptName
Sally	2241	Sales
Harriet	2202	Production

Outer joins

- Whereas the result of a join (or inner join) consists of tuples formed by combining matching tuples in the two operands, an outer join contains those tuples and additionally some tuples formed by extending an unmatched tuple in one of the operands by "fill" values for each of the attributes of the other operand.

- **Left outer join (⟕)**
- **Right outer join (⟖)**
- **Full outer join (⟗)**

Left outer join (⟕)

- The result of the left outer join is the set of all combinations of tuples in *R* and *S* that are equal on their common attribute names, in addition (loosely speaking) to tuples in *R* that have no matching tuples in *S*.

Employee

Name	EmpId	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales
Tim	1123	Executive

Dept

DeptName	Manager
Sales	Harriet
Production	Charles

Employee ⟕ Dept

Name	EmpId	DeptName	Manager
Harry	3415	Finance	ω
Sally	2241	Sales	Harriet
George	3401	Finance	ω
Harriet	2202	Sales	Harriet
Tim	1123	Executive	ω

Right outer join (⟖)

- The result of the right outer join is the set of all combinations of tuples in *R* and *S* that are equal on their common attribute names, in addition to tuples in *S* that have no matching tuples in *R*.

Employee

Name	EmpId	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales
Tim	1123	Executive

Dept

DeptName	Manager
Sales	Harriet
Production	Charles

Employee ⟖ Dept

Name	EmpId	DeptName	Manager
Sally	2241	Sales	Harriet
Harriet	2202	Sales	Harriet
ω	ω	Production	Charles

Full outer join (⟗)

- The **outer join** or **full outer join** in effect combines the results of the left and right outer joins.
- The result of the full outer join is the set of all combinations of tuples in *R* and *S* that are equal on their common attribute names, in addition to tuples in *S* that have no matching tuples in *R* and tuples in *R* that have no matching tuples in *S* in their common attribute names.

Employee

Name	EmpId	DeptName
Harry	3415	Finance
Sally	2241	Sales
George	3401	Finance
Harriet	2202	Sales
Tim	1123	Executive

Dept

DeptName	Manager
Sales	Harriet
Production	Charles

Employee* ⟗ *Dept

Name	EmpId	DeptName	Manager
Harry	3415	Finance	ω
Sally	2241	Sales	Harriet
George	3401	Finance	ω
Harriet	2202	Sales	Harriet
Tim	1123	Executive	ω
ω	ω	Production	Charles

Division (÷)

- Division is not implemented directly in SQL.
- The result consists of the restrictions of tuples in *R* to the attribute names unique to *R*, i.e., in the header of *R* but not in the header of *S*, for which it holds that all their combinations with tuples in *S* are present in *R*.

Completed

Student	Task
Fred	Database1
Fred	Database2
Fred	Compiler1
Eugene	Database1
Eugene	Compiler1
Sarah	Database1
Sarah	Database2

DBProject

Task
Database1
Database2

Completed* ÷ *DBProject

Student
Fred
Sarah

Division (÷)

- **Tuple calculus:**
  - $R \div S = \{\ t[a_1,\ldots,a_n] : t \in R \wedge \forall s \in S\ (\ (t[a_1,\ldots,a_n] \cup s) \in R)\ \}$

- **Relational algebra:**
  - $R \div S = \pi_{A-B}(R) - \pi_{A-B}((\pi_{A-B}(R) \times S) - R)$

QUIZE

- Suppliers (***sid****:* integer, ***sname****:* string, *address:* string)
- Parts (***pid****:* integer, ***pname****:* string, *color:* string)
- Catalog (***sid****:* integer, ***pid****:* integer, ***cost****:*real)

- **Find** the names of *suppliers* who supply some *red* part
- **Find** the *sid*s of suppliers who supply some red part and some green part

QUIZE

- Suppliers (***sid**:* integer, ***sname**:* string, *address:* string)
- Parts (***pid**:* integer, ***pname**:* string, *color:* string)
- Catalog(***sid**:* integer, ***pid**:* integer, ***cost***:real)

- **Find** the names of *suppliers* who supply some *red* part
- **Find** the *sid*s of suppliers who supply some red part and some green part.

$$\pi_{sname}(\pi_{sid}(\pi_{pid}(\sigma_{color=red}(P)) \bowtie C) \bowtie S)$$

$$\rho(R1, \pi_{sid}(\pi_{pid}(\sigma_{color=red}(P)) \bowtie C))$$
$$\rho(R2, \pi_{sid}(\pi_{pid}(\sigma_{color=green}(P)) \bowtie C)$$

$$R1 \cap R2$$

End of Chapter 2