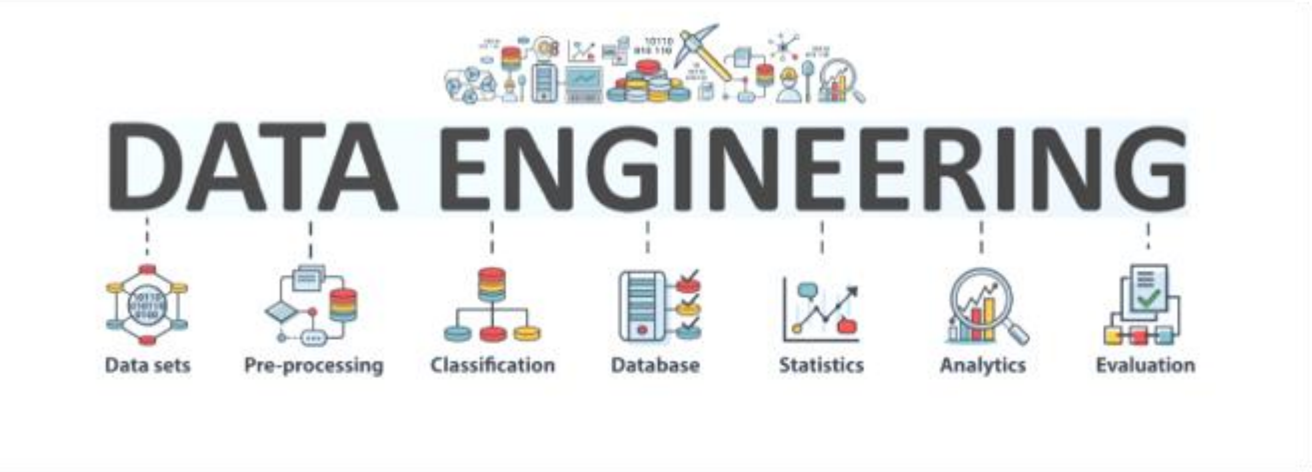


Lecture 6: Data Querying

CS5481 Data Engineering

Instructor: Yifan Zhang

Outline

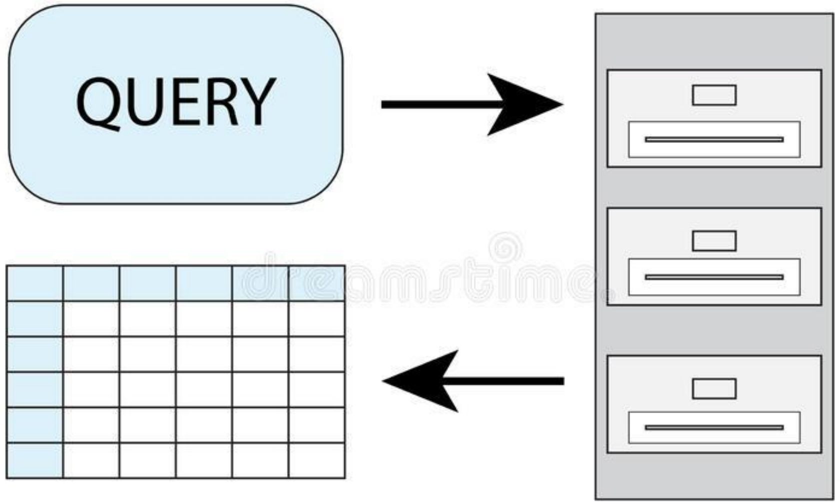
1. Overview of data querying
2. Relational algebra and SQL querying
3. NoSQL querying

Data querying

A data query is either an action query or a select query.

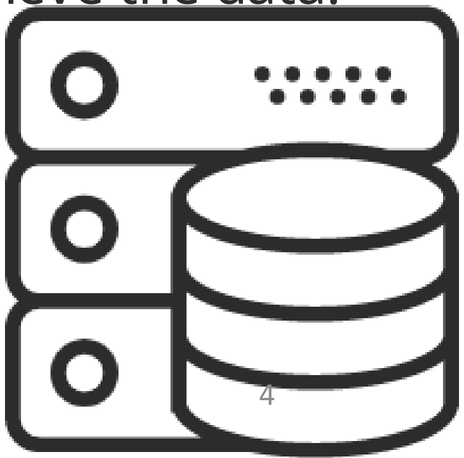
A select query is one that retrieves data from a database.

An action query asks for additional operations on data, such as insertion, updating, deleting or other forms of data manipulation.



How to perform a query in a database?

- **1. Choose your data**
  - Determine the data you want to retrieve or update and consider the method you want to use to perform a query
- **2. Specify data fields**
  - Once you identify the data you want to work with and decide on a method, consider how to perform the query. Choose the data fields you want to include in your query.
- **3. Assign a table (for relational databases) or a data structure component (for NoSql databases)**
  - When performing a query using a query language, it's important to tell the database where to find the information, so the system understands where to retrieve the data.
- **4. Filter data**
  - When a database returns a query result, you can choose to filter the information.



Relational databases and relational algebra

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)

Relational databases

Six basic operators

- select: σ
- project: Π
- union: $\cup$
- set difference: –
- Cartesian product: x
- rename: ρ

Additional operations that simplify common queries

- set intersection
- join
- assignment
- outer join

Relational algebra

Relational databases – attributes (columns)

- 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”

Relational databases – tables

- $A_1, A_2, \ldots, A_n$ 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*** (a table) is a subset of

  $D_1 \times D_2 \times \ldots \times D_n$

  where D_i is the domain of attribute A_i. Thus, a relation is a set of *n*-tuples $(a_1, a_2, \ldots, a_n)$ where each $a_i \in D_i$
- The current values (relation instance) of a relation are specified by a table
- An element t of r is a tuple, represented by a row in a table

Relational databases – 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*
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

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

Relational databases – multiple relations

- A relational database consists of multiple relations

- Information about a university 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)

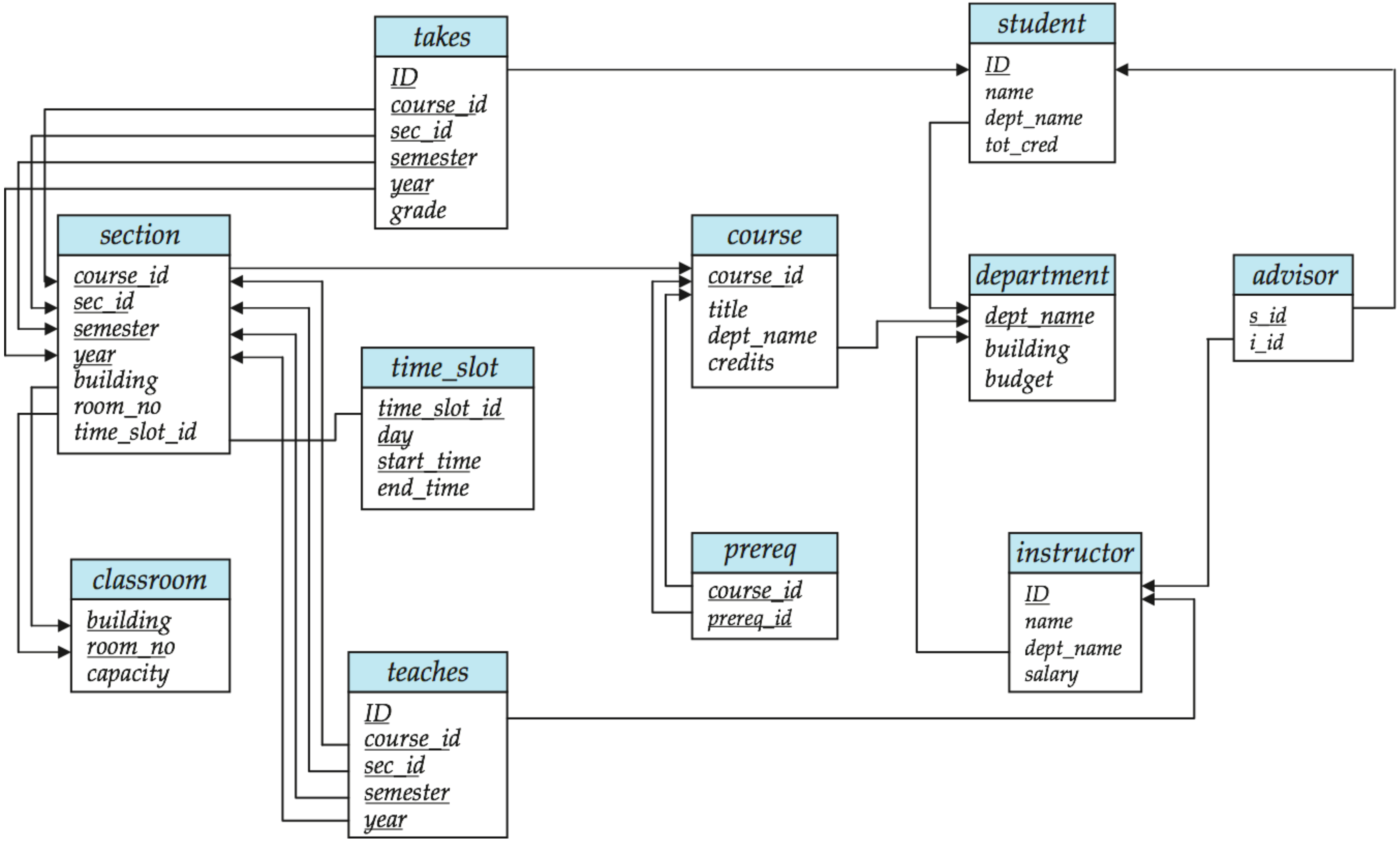
  - the need for null values (e.g., represent an student with no advisor)

Relational databases – 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**.
- **Foreign key:** a set of attributes in a table that refers to the primary key of another table. The foreign key links these two tables.
  - Example – *dept_name* in *instructor* is a foreign key from *instructor* referencing *department*

Relational databases – example



Relational algebra – select (1)

- The **select** operation selects tuples that satisfy a given predicate.
- Notation: $\sigma_p(r)$
- p is called the **selection predicate (condition)**

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*
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

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

Relational algebra – select (2)

- 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:

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

- Then select predicate may include comparisons between two attributes.

Example, find all departments whose name is the same as their building name:

$$\sigma_{dept\_name = building}\ (\text{department})$$

Relational algebra – project

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

Query:

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

Result:

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

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

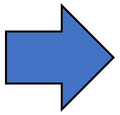
Relational algebra - composition of relational operations

- The result of a relational-algebra operation is relation and therefore 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.

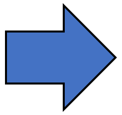
$$\Pi_{name}(\sigma_{dept\_name = \text{“Physics”}}(instructor))$$

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

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



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



name
Einstein
Gold

Relational algebra - union

- The union operation allows us to combine two relations (union of sets)
- Notation: $r \cup s$
- Output the union of tuples from the two input relations
- 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 2009 semester, or in the Spring 2010 semester, or in both

Query:

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

$$\Pi_{course\_id}(\sigma_{semester=\text{“Spring”} \ \land \ year=2010}(section))$$

Result:

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

Relational algebra – set difference

- The set-difference operation allows us to find tuples that are in one relation but are not in another.
- Notation: *r – s*
- Produce a relation containing those tuples in *r* but not in *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

Query:

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

Result:

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

Relational algebra – Cartesian-product (1)

- The Cartesian-product operation (denoted by X) allows us to combine information from any two relations.
- Notation: r x s
- Output all pairs of rows from the two input relations (regardless of whether or not they have the same values on common attributes)
- Assume that attributes of $r(R)$ and $s(S)$ are disjoint. (That is, $R \cap S = \emptyset$).
- If attributes of $r(R)$ and $s(S)$ are not disjoint, then renaming must be used.

Relational algebra – Cartesian-product (2)

- 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*

instructor

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

teaches

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

Inst.ID	*name*	*dept_name*	*salary*	*teaches.ID*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	12121	FIN-201	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	15151	MU-199	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	22222	PHY-101	1	Fall	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
12121	Wu	Finance	90000	10101	CS-101	1	Fall	2009
12121	Wu	Finance	90000	10101	CS-315	1	Spring	2010
12121	Wu	Pinance	90000	10101	CS-347	1	Fall	2009
12121	Wu	Pinance	90000	12121	FIN-201	1	Spring	2010
12121	Wu	Finance	90000	15151	MU-199	1	Spring	2010
12121	Wu	Pinance	90000	22222	PHY-101	1	Fall	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
15151	Mozart	Music	40000	10101	CS-101	1	Fall	2009
15151	Mozart	Music	40000	10101	CS-315	1	Spring	2010
15151	Mozart	Music	40000	10101	CS-347	1	Fall	2009
15151	Mozart	Music	40000	12121	FIN-201	1	Spring	2010
15151	Mozart	Music	40000	15151	MU-199	1	Spring	2010
15151	Mozart	Music	40000	22222	PHY-101	1	Fall	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
22222	Einstein	Physics	95000	10101	CS-101	1	Fall	2009
22222	Einstein	Physics	95000	10101	CS-315	1	Spring	2010
22222	Einstein	Physics	95000	10101	CS-347	1	Fall	2009
22222	Einstein	Physics	95000	12121	FIN-201	1	Spring	2010
22222	Einstein	Physics	95000	15151	MU-199	1	Spring	2010
22222	Einstein	Physics	95000	22222	PHY-101	1	Fall	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

Relational algebra – query examples

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

Query 1

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

inst.ID	*name*	*dept_name*	*salary*	*teaches.ID*	*course_id*	*sec_id*	*semester*	*year*
22222	Einstein	Physics	95000	10101	CS-437	1	Fall	2009
22222	Einstein	Physics	95000	10101	CS-315	1	Spring	2010
22222	Einstein	Physics	95000	12121	FIN-201	1	Spring	2010
22222	Einstein	Physics	95000	15151	MU-199	1	Spring	2010
22222	Einstein	Physics	95000	22222	PHY-101	1	Fall	2009
22222	Einstein	Physics	95000	32343	HIS-351	1	Spring	2010
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
33456	Gold	Physics	87000	10101	CS-437	1	Fall	2009
33456	Gold	Physics	87000	10101	CS-315	1	Spring	2010
33456	Gold	Physics	87000	12121	FIN-201	1	Spring	2010
33456	Gold	Physics	87000	15151	MU-199	1	Spring	2010
33456	Gold	Physics	87000	22222	PHY-101	1	Fall	2009
33456	Gold	Physics	87000	32343	HIS-351	1	Spring	2010
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

name	*course_id*
Einstein	PHY-101

- There is often more than one way to write a query in relational algebra
- The following two queries are equivalent, i.e., they give the same result

Query 1

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

Query 2

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

Query optimization

Query optimization: the process of choosing a suitable execution strategy for processing a query.

A query, e.g., a SQL, first be scanned, parsed and validated

- The scanner identifies the query tokens, e.g., the keywords, attribute names and relation names
- The parser checks the query syntax
- The validation checks that all attributes and relation names are valid

Relational algebra – rename

- Allows us to name, and therefore to refer to, the results of relational-algebra expressions. Allows us to refer to a relation by more than one name.
- The expression $\rho_x\,(E)$ returns the result of 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 $A_1, A_2, \ldots, A_n$.

Example: Find the highest salary in the university

Step 1: find instructor salaries that are less than some other instructor salary (i.e. not the highest). Using a copy of *instructor* under a new name *d*

$$\Pi_{instructor.salary}\,(\sigma_{\,instructor.salary\,<\,d.salary}\,(instructor \times \rho_d\,(instructor)))$$

Step 2: Find the highest salary

$$\Pi_{salary}\,(instructor) - \Pi_{instructor.salary}\,(\sigma_{\,instructor.salary\,<\,d.salary}\,(instructor \times \rho_d\,(instructor)))$$

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

salary
65000
90000
40000
60000
87000
75000
62000
72000
80000
92000

salary
95000

Relational algebra – additional operations (1)

Set-Intersection

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

Relational algebra – additional operations (2)

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*))
- 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* x *teaches*))

can equivalently be written as: *instructor* $\bowtie_{Instructor.id = teaches.id}$ *teaches*.

Relational algebra – additional operations (3)

Join Operation

- The join operation without predicate is called **natural join**.
- Notation: $r \bowtie s$
- Output pairs of rows from the two input relations that have the **same value** on all attributes that have the **same name**
- 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

Relational algebra – additional operations (4)

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 a result of the query.

Relational algebra – additional operations (5)

Outer Join Operation

- An extension of the join operation that avoids loss of information.
- Computes the join and then adds tuples from one relation that does not match tuples in the other relation to the result of the join.
- Uses *null* value to signify that the value is unknown or does not exist
- Three forms of outer join:
  - left outer join
  - right outer join
  - full outer join

Relational algebra – additional operations (6)

Outer Join Operation

- Relation *course*

course_id	*title*	*dept_name*	*credits*
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

- Relation *prereq*

course_id	*prereq_id*
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

- Observe that
- CS-315 is missing in prereq and
- CS-347 is missing in course

Relational algebra – additional operations (7)

Outer Join Operation

- Join
  - *course* ⋈ *prereq*

course_id	title	dept_name	credits	prere_id
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101

- Left Outer Join
  - *course* ⟕ *prereq*

course_id	title	dept_name	credits	prere_id
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	*null*

Outer join can be expressed using basic operations

e.g. r ⟕ s can be written as

$(r \bowtie s) \cup (r - \Pi_R(r \bowtie s)) \times \{(null, \ldots, null)\}$

where the constant relation $\{(null, \ldots, null)\}$ is on the schema $S - R$

- Right Outer Join
  - *course* ⟖ *prereq*

course_id	title	dept_name	credits	prereq_id
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-347	*null*	*null*	*null*	CS-101

- Full Outer Join
  - *course* ⟗ *prereq*

course_id	title	dept_name	credits	prereq_id
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	*null*
CS-347	*null*	*null*	*null*	CS-101

SQL – Select

Select <*List of Columns and expressions (usually involving columns)*>

From <*List of Tables & Join Operators*>

Where <*List of Row conditions joined together by And, Or, Not*>

Group By <*list of grouping columns*>

Having <*list of group conditions connected by And, Or, Not* >

Order By <*list of sorting specifications*>

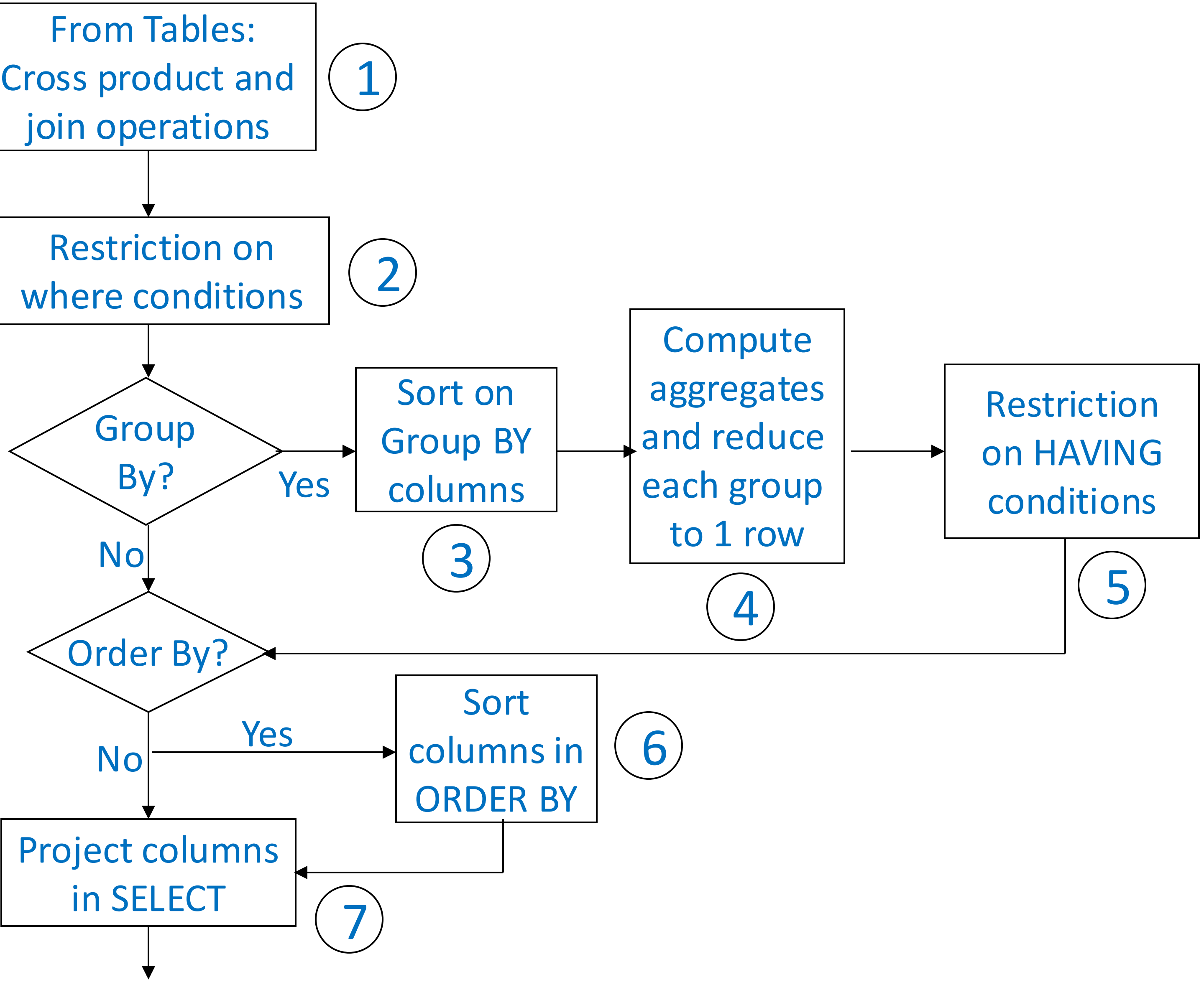
Payroll

UserID	Name	Job	Salary
123	Jack	TA	50000
345	Allison	TA	60000
567	Magda	Prof	120000
789	Dan	Prof	100000

```
SELECT P.Name, P.UserID
  FROM Payroll AS P
 WHERE P.Job = 'TA';
```

Name	UserID
Jack	123
Allison	345

SQL – conceptual evaluation



SQL – Order By

- Used to sort the results based on contents of a column
- Multiple levels of sort can be done by specifying multiple columns
- An expression can be used in Order By clause

Syntax:

```
Select function(column)
    From table1 [, table2 …]
    [Where condition]
    [Order By {Column | alias | position} [ASC | DESC]]
```

SQL – Order By

Example: Sort Movies by profits in Ascending order

Select Movie_title, Gross, Budget, (Gross – Budget) as profits

From movies

Order BY profits

Movie_title	Gross	Budget	Profit
Great Escape	67.5	70	-2.5
Upside Down	54	50	4
Green Warrior	96	80	16
Blue Oranges	28	7	21

SQL – Group By

- Categorizes the query results according to the contents of a column in the database .
- Multiple levels of subgroups can be created by specifying multiple columns.

Syntax:

Select column1, [column2, …]

From table [, table …]

[Where condition]

Group By column1, [column2, ….]

Having [Condition]

Table: Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT country, COUNT(*) AS number
FROM Customers
GROUP BY country;
```

country	number
UAE	1
UK	2
USA	2

SQL – Join

- A Join is a Query that combines data from multiple tables
  - Multiple tables are specified in the From Clause
  - For two tables to be joined in a sensible manner, they need to have data in common

Example:

Schema: Movies (movie_title, director_id, release_date)

People(person_fname, person_lname, person_id)

Query:

```
Select movie_title, person_fname, person_lname
From Movies, People
Where director_id = person_id
```

SQL – Joining Condition

For a useful Join query a joining condition is required

- Defined in where clause as relationships between columns
- Multiple conditions may be defined if multiple columns shared
- More than two tables can be joined in a query

Example: Find people who live in same state as studio

Schema:

Studios(studio_id, studio_state, studio_name, studio_city)

People(person_fname, person_lname, person_id, person_state, person_city)

Query:

```
Select person_fname, person_lname, studio_name
From Movies, People
Where studio_city = person_city
AND studio_state = person_state
```

SQL – Self Join

Required to compare values within a single column

- Need to define aliases for the table names

Example: Find actors living in the same state

Schema:

People(person_fname, person_lname, person_id, person_state, person_city)

Query:

```
Select p1.person_id, p1.person_fname, p1.person_lname, p1.person_state
From People p1, People p2
Where p1.person_state = p2.person_state
AND p1.person_id != p2.person_id
```

Note:

Distinct operator is critical because if there are more than two people from any state each person will appear as many times as there are people from the state

SQL – Left/Right Join

Schema:

People(person_fname, person_lname, person_id, person_state, person_city)

Movies(movie_id, movie_title, director_id, studio_id)

Location(movie_id, city, state)

Query:

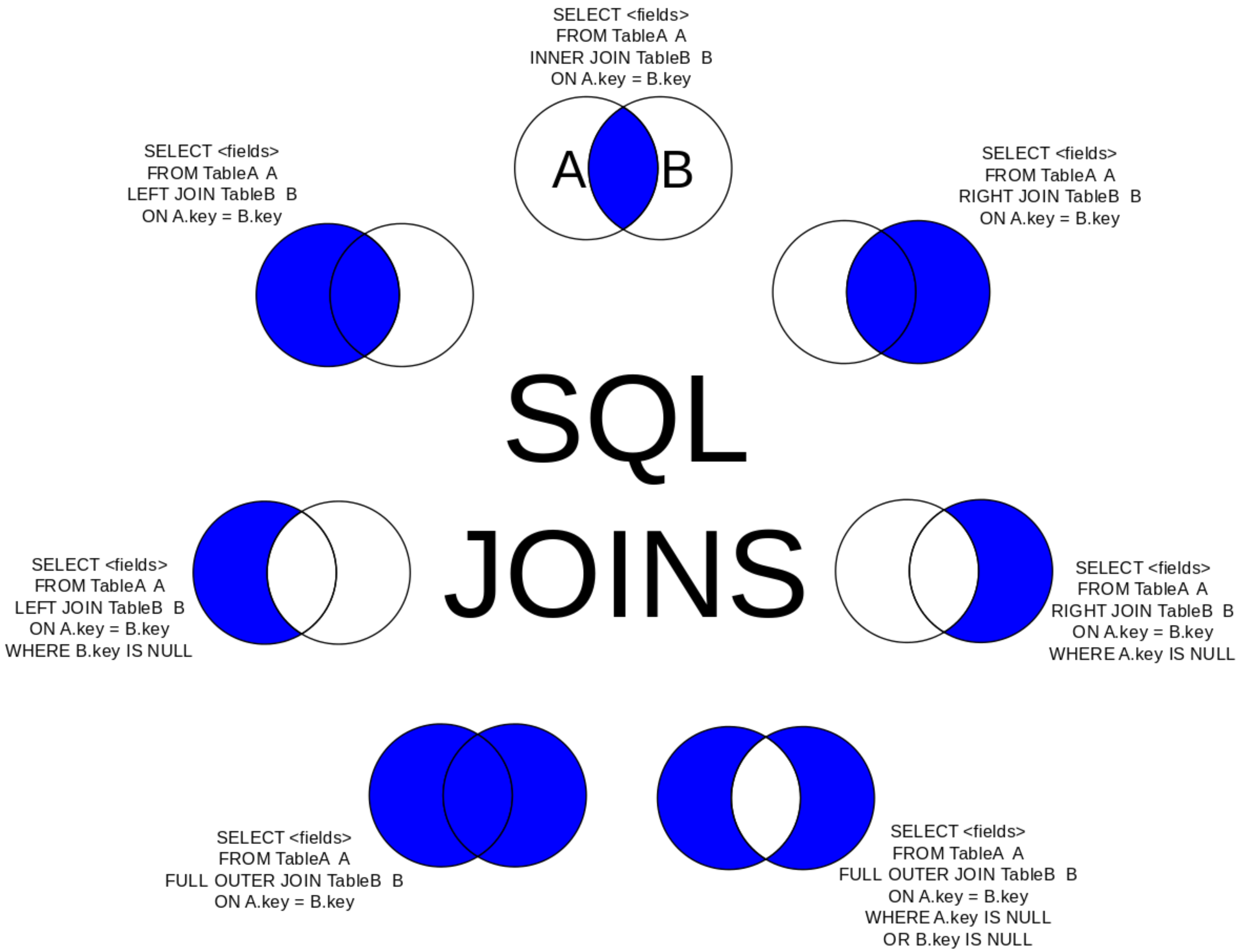
```
Select movie_title, city, state
From Movies Left Join Locations
    On Movies.movie_id = Locations.movie_id
```

Includes all non matched movie titles

```
Select movie_title, person_fname, person_lname
From Movies Right Join People
    On Movies.director_id = Person.person_id
```

Includes all people not matching to directors

SQL – all Joins



SQL – Nested Queries

- A sub query is a query nested within another query
  - The enclosing query also called outer query
  - Nested query is called inner query
- There can be multiple levels of nesting

Example:

```
Select movie_title
From movies
Where director_id IN (
        Select person_id
        From People
        Where person_state = 'TX')
```

NoSQL querying – key-value data store

- There is no schema and the value of the data can be just about anything. Values are identified and accessed via a key, and values can be numbers, strings, counters, JSON, XML, HTML, binaries, images, videos, and more.
  - E.g., shopping carts, session logs.
- Keys can be queried. Values cannot be queried.
  - To query using some attribute of the value is not possible (in general). We need to read the value to test any query condition
- No query language for key-value stores.
- Solution:
  - Sorted keys: keys can be indexed first, and then searched.
  - Secondary keys/indices can be used, e.g., email addresses, telephone numbers, names, or contents from the value.

Key	Value
K1	AAA,BBB,CCC
K2	AAA,BBB
K3	AAA,DDD
K4	AAA,2,01/01/2015
K5	3,ZZZ,5623

Example of key value database with keys mapping to different values and data types.

Key	Value
user-123	"John Doe"
image-123.jpg	<binary image file>
http://webpage-123.html	<web page html>
file-123.pdf	<pdf document>

NoSQL querying – document databases (1)

- Document databases
  - Structured text data - Hierarchical tree data structures
  - Typically JSON, XML, BSON (Binary JSON)
- XML: Extensible Markup Language
  - Tags: XML tags are not predefined. You need to define your customized tags.
  - XML was designed to carry data, not to display that data
- JSON: JavaScript Object Notation
  - object: an unordered set of name+value pairs,

syntax: { name: value, name: value, name: value, ...}

  - array: an ordered collection of values (elements),

syntax [comma-separated values]

JSON Example

```
{"employees":[
  { "firstName":"John", "lastName":"Doe" },
  { "firstName":"Anna", "lastName":"Smith" },
  { "firstName":"Peter", "lastName":"Jones" }
]}
```

XML Example

```
<employees>
  <employee>
    <firstName>John</firstName> <lastName>Doe</lastName>
  </employee>
  <employee>
    <firstName>Anna</firstName> <lastName>Smith</lastName>
  </employee>
  <employee>
    <firstName>Peter</firstName> <lastName>Jones</lastName>
  </employee>
</employees>
```

```
{
  "conferences":
  [
    {
      "name": "XML Prague 2015",
      "start": "2015-02-13",
      "end": "2015-02-15",
      "web": "http://xmlprague.cz/",
      "price": 120,
      "currency": "EUR",
      "topics": ["XML", "XSLT", "XQuery", "Big Data"],
      "venue": {
        "name": "VŠE Praha",
        "location": {
          "lat": 50.084291,
          "lon": 14.441185
        }
      }
    },
    {
      "name": "DATAKON 2014",
      "start": "2014-09-25",
      "end": "2014-09-29",
      "web": "http://www.datakon.cz/",
      "price": 290,
      "currency": "EUR",
      "topics": ["Big Data", "Linked Data", "Open Data"]
    }
  ]
}
```

NoSQL querying – document databases (2)

- E.g., MongoDB querying: Mongo query language
  - Targets a specific collection of documents
  - Specifies criteria that identify the returned documents
  - May include a projection to specify returned fields
  - May impose limits, sort, orders, …
- Basic query - all documents in the collection:
  - db.users.find()
  - db.users.find({})
- E.g.,

```
db.inventory.find({ type: "snacks" })
```

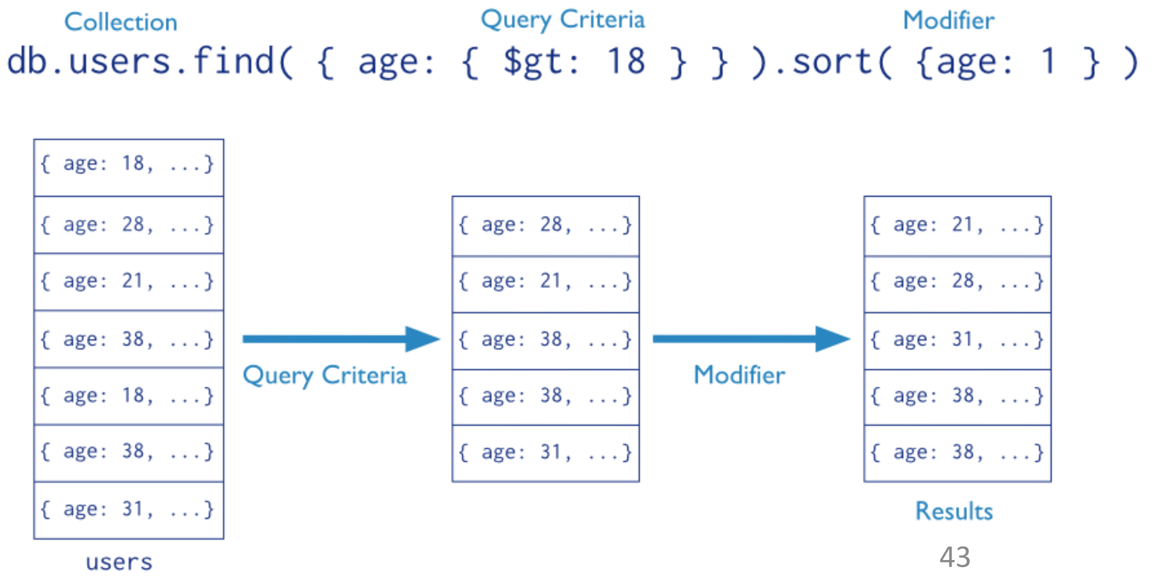
All documents from collection inventory where the type field has the value snacks

```
db.inventory.find({ type: { $in: [ 'food', 'snacks' ] } } )
```

All inventory docs where the type field is either food or snacks

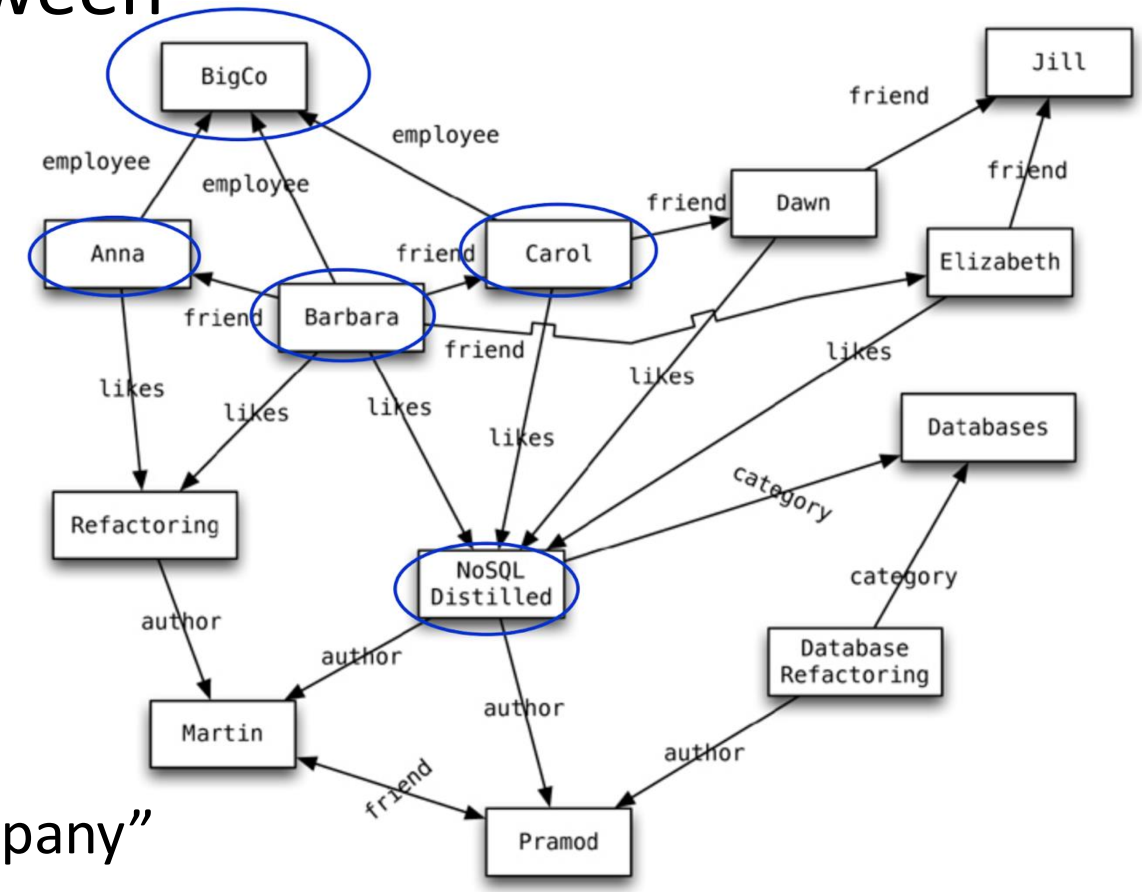
```
db.inventory.find( { type: 'food', price: { $lt: 9.95 } } )
```

All … where the type field is food and the price is less than 9.95



NoSQL querying – graph databases (1)

- To store entities and relationships between them
  - Nodes are instances of objects
  - Nodes have properties, e.g., name
  - Edges connect nodes and have directional significance
  - Edges have types e.g., likes, friend, …

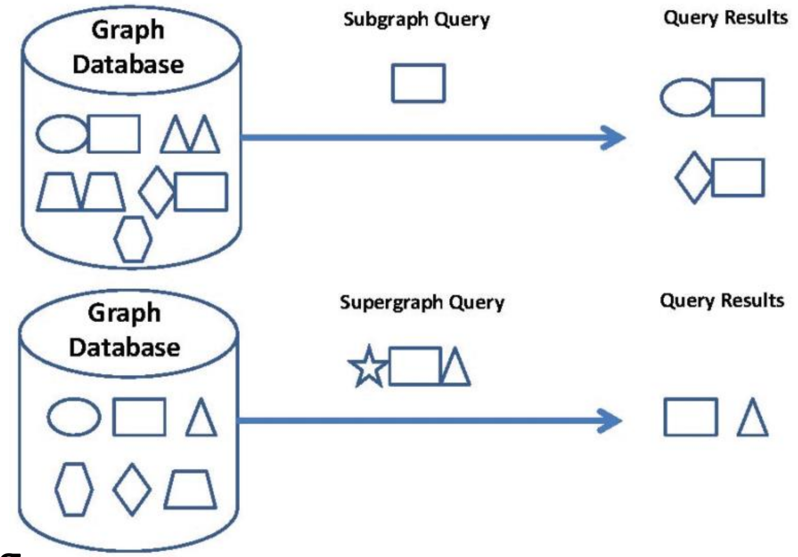


- Nodes are organized by relationships
  - Allow to find interesting patterns

- **E.g.,** Get all nodes that are "employee" of "Big Company" and that "likes" "NoSQL Distilled"

NoSQL querying – graph databases (2)

- Subgraph queries
  - Searches for a specific pattern in the graph database
  - Query = a small graph
    - or a graph, where some parts are uncertain, e.g., vertices with wildcard labels
  - More general type: allow sub-graph isomorphism
- Super-graph queries
  - Search for graphs whose whole structure is contained in the query graph
- Similarity (approximate matching) queries
  - Finds graphs which are similar to a given query graph
    - but not necessarily isomorphic
  - Key question: how to measure the similarity



- Cypher Language: Neo4j graph query language for querying and updating
  - MATCH: The graph pattern to match
  - WHERE: Filtering criteria
  - RETURN: What to return
  - START: Starting points in the graph by explicit index lookups or by node IDs (both deprecated)
  - WITH: Divides a query into multiple parts
  - CREATE: Creates nodes and relationships.
  - DELETE: Remove nodes, relationships, properties
  - SET: Set values to properties

```
MATCH (p: Person)
WHERE p.age >= 18 AND p.age < 30
RETURN p.name
```

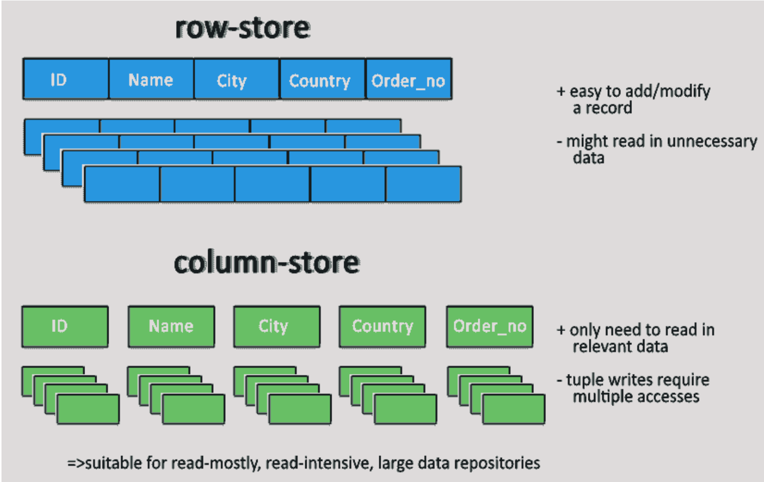
(return names of all adult people under 30)

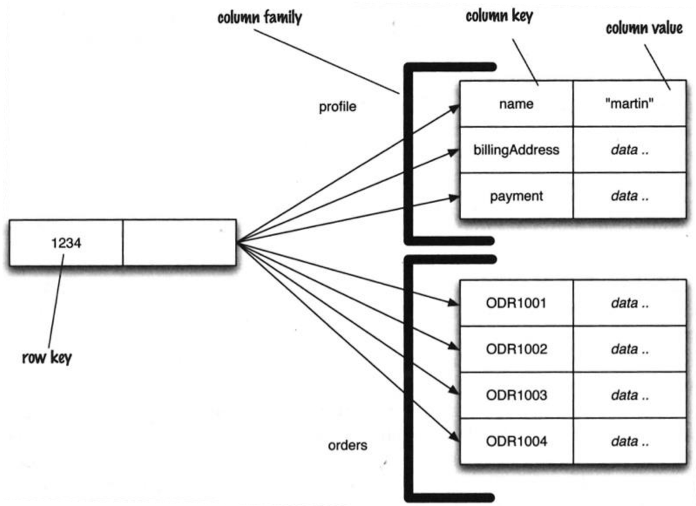
```
MATCH (user: Person {name: 'Andres'})-[:Friend]->(follower)
RETURN user.name, follower.name
```

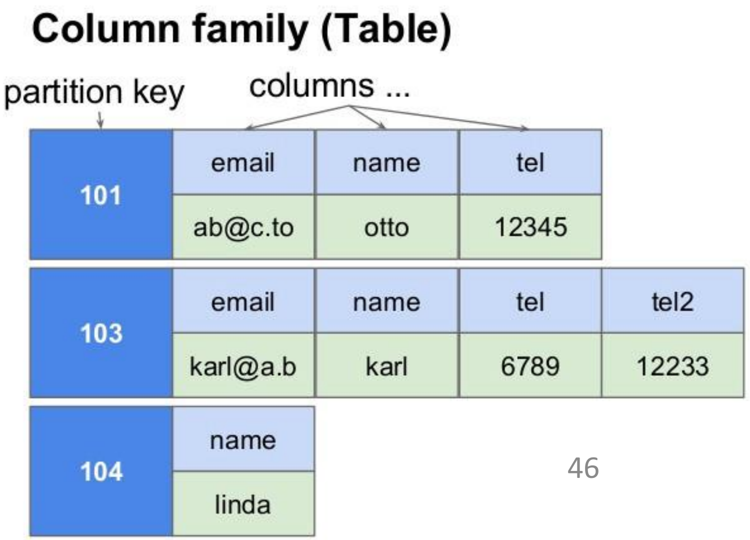
(find all 'Friends' of 'Andres')

NoSQL querying – columnar databases

- **Column:** the basic data item, column key-value pairs
- **Row:** a collection of columns attached to row key
  - Columns can be added to any row at any time
    - without having to add it to other rows
- E.g., **Cassandra Query Language (CQL)**.
  The syntax of CQL is similar to SQL
  - But search just in one table (no joins)
  - `SELECT <selectExpr>`
  - `FROM [<keyspace>.]<table>`
  - `[WHERE <clause>]`
  - `[ORDER BY <clustering_colname> [DESC]]`
  - `[LIMIT m];`
  - e.g.,
  - `SELECT column_name, column_value`
  - `FROM mytable`
  - `WHERE row_id=3`
  - `ORDER BY column_value;`







Thanks for your attention!

Appendix

1. https://www.sqltutorial.org/

2. http://disa.fi.muni.cz/david-novak/teaching/nosql-databases-2018/lectures/

3. https://www.oreilly.com/library/view/nosql-for-mere/9780134029894/