Chapter 3: Introduction to SQL
Nested SubQueries, and more

Last Lecture:

- Null Values
- Aggregate Functions

Today:

- Nested Subqueries
- More complex queries
- More SQL updates
- Relational Algebra (maybe)

Nested Subqueries

- SQL provides a mechanism for the nesting of subqueries.

- A **subquery** is a **select-from-where** expression that is nested within another query.

- A common use of subqueries is to perform tests for set membership, set comparisons, and set cardinality.

Example Database Schema

Instructor (ID, name, dept_name, salary)

Course (course_id, title, dept_name, credits)

Section (course_id, section_id, semester, year)

Teaches (ID, course_id, sec_id, semester, year)

Department(dept_name, building, budget)

Takes (ID, course_id, sec_id, semester, year, grade)

Examples

Find courses offered in Fall 2015 and in Spring 2015

```
select distinct course_id
from section
where semester = 'Fall' and year= 2015 and
          course_id in (select course_id
                        from section
                        where semester = 'Spring' and year= 2015);
```

Find courses offered in Fall 2015 but not in Spring 2015

```
select distinct course_id
from section
where semester = 'Fall' and year= 2015 and
          course_id  not in (select course_id
                             from section
                             where semester = 'Spring' and year= 2015);
```

Example Query

- Find the total number of (distinct) students who have taken course sections taught by the instructor with *ID* 10101

```
select count (distinct ID)
from takes
where (course_id, sec_id, semester, year) in
                        (select course_id, sec_id, semester, year
                         from teaches
                         where teaches.ID= 10101);
```

Note: Above query can be written in a much simpler manner. The formulation above is simply to illustrate SQL features.

How??

Set Comparison

Find names of instructors with salary greater than that of some (at least one) instructor in the Biology department.

select distinct *T.name*
from *instructor T*, *instructor S*
where *T.salary* > *S.salary* **and** *S.dept_name* = 'Biology';

- Same query using > **some** clause

select *name*
from *instructor*
where *salary* > **some** (**select** *salary*
from *instructor*
where *dept_name* = 'Biology');

Definition of Some Clause

F <comp> **some** $r \Leftrightarrow \exists\, t \in r$ such that (F <comp> t)
where <comp> can be: $<,\ \leq,\ >,\ =,\ \neq$

(5 < **some** $\begin{array}{|c|}\hline 0 \\ \hline 5 \\ \hline 6 \\ \hline \end{array}$) = true   (read: 5 < some tuple in the relation)

(5 < **some** $\begin{array}{|c|}\hline 0 \\ \hline 5 \\ \hline \end{array}$) = false

(5 = **some** $\begin{array}{|c|}\hline 0 \\ \hline 5 \\ \hline \end{array}$) = true

(5 ≠ **some** $\begin{array}{|c|}\hline 0 \\ \hline 5 \\ \hline \end{array}$) = true (since 0 ≠ 5)

(= some) ≡ in   However, (≠ some) $\not\equiv$ not in

Example Query

Find the names of all instructors whose salary is greater than the salary of all instructors in the Biology department.

```
select name
from instructor
where salary > all (select salary
                    from instructor
                    where dept_name = 'Biology');
```

Definition of all Clause

F <comp> **all** $r \Leftrightarrow \forall\, t \in r$ (F <comp> t)

$$(5 < \textbf{all} \begin{array}{|c|}\hline 0 \\ \hline 5 \\ \hline 6 \\ \hline \end{array}) = \text{false}$$

$$(5 < \textbf{all} \begin{array}{|c|}\hline 6 \\ \hline 10 \\ \hline \end{array}) = \text{true}$$

$$(5 = \textbf{all} \begin{array}{|c|}\hline 4 \\ \hline 5 \\ \hline \end{array}) = \text{false}$$

$$(5 \neq \textbf{all} \begin{array}{|c|}\hline 4 \\ \hline 6 \\ \hline \end{array}) = \text{true (since } 5 \neq 4 \text{ and } 5 \neq 6)$$

($\neq$ all) $\equiv$ not in   **However, (= all) $\not\equiv$ in**

Test for Empty Relations

- The **exists** construct returns the value **true** if the argument subquery is nonempty.

- **exists** $r \Leftrightarrow r \neq \emptyset$

- **not exists** $r \Leftrightarrow r = \emptyset$

Correlation Variables

- Yet another way of specifying the query “Find all courses taught in both the Fall 2015 semester and in the Spring 2015 semester”

    select *course_id*
    from *section* **as** *S*
    where *semester* = ’Fall’ **and** *year* = ‘2015’ **and**
    exists (**select** *
    from *section* **as** *T*
    where *semester* = ’Spring’ **and** *year* = ‘2015’
    and *S*.*course_id*= *T*.*course_id*);

- **Correlated subquery**
- **Correlation name** or **correlation variable**

Not Exists

Find all students who have taken all courses offered in the Biology department.

```
select distinct S.ID, S.name
from student as S
where not exists ( (select course_id
                    from course
                    where dept_name = 'Biology')
                  except
                   (select T.course_id
                    from takes as T
                    where S.ID = T.ID));
```

- Note that $X - Y = \emptyset \iff X \subseteq Y$
- *Note:* Cannot write this query using = **all** and its variants

Test for Absence of Duplicate Tuples

- The **unique** construct tests whether a subquery has any duplicate tuples in its result.
  - (Evaluates to “true” on an empty set)

- Find all courses that were offered at most once in 2017

```
  select T.course_id
from course as T
where unique (select R.course_id
              from section as R
              where T.course_id= R.course_id
                  and R.year = 2017);
```

SQL allows a subquery expression to be used in the FROM clause

- Find the average instructors' salaries of those departments where the average salary is greater than $60,000.

```
select dept_name, avg_salary
from (select dept_name, avg (salary) as avg_salary
        from instructor
        group by dept_name)
where avg_salary > 60000;
```

- Note that we do not need to use the **having** clause. Another way to write above query

```
select dept_name, avg_salary
from (select dept_name, avg (salary)
        from instructor
        group by dept_name)  as dept_avg (dept_name,  avg_salary)
where avg_salary > 60000;
```

Same Query using Having Clause

- Find the names and average salaries of all departments whose average salary is greater than 60000

select *dept_name*, **avg** (*salary*)
from *instructor*
group by *dept_name*
having avg (*salary*) > 60000;

Note: predicates in the **having** clause are applied **after** the formation of groups whereas predicates in the **where** clause are applied **before** forming groups

With Clause

- The **with** clause provides a way of defining a temporary view whose definition is available only to the query in which the **with** clause occurs.
- Find all departments with the maximum budget

```
with max_budget (value) as
    (select max(budget)
     from department)
select budget
from department, max_budget
where department.budget = max_budget.value;
```

Complex Queries using With Clause

- With clause is very useful for writing complex queries
- Supported by most database systems, with minor syntax variations
- Find all departments where the total salary is greater than the average of the total salary at all departments

```
with dept _total (dept_name, value) as
        (select dept_name, sum(salary)
         from instructor
         group by dept_name),
dept_total_avg(value) as
       (select avg(value)
       from dept_total)
select dept_name
from dept_total, dept_total_avg
where dept_total.value >= dept_total_avg.value;
```

Scalar Subquery

- Scalar subquery is one which is used where a single value is expected
- E.g.

```
select dept_name,
    (select count(*)
      from instructor
     where department.dept_name = instructor.dept_name)
    as num_instructors
from department;
```

- E.g.

```
select name
from instructor
where  salary * 10 >
    (select budget  from department
      where department.dept_name = instructor.dept_name)
```

- Runtime error if subquery returns more than one result tuple

Modification of the Database

- Deletion of tuples from a given relation
- Insertion of new tuples into a given relation
- Updating values in some tuples in a given relation

Modification of the Database – Deletion

- Delete all instructors

  delete from *instructor*

- Delete all instructors from the Finance department

  delete from *instructor*
  where *dept_name*= 'Finance';

- Delete all tuples in the *instructor* relation for those instructors associated with a department located in the Watson building.

  delete from *instructor*
  where *dept_name* **in** (**select** *dept_name*
  from *department*
  where *building* = 'Watson');

Deletion (Cont.)

- Delete all instructors whose salary is less than the average salary of instructors

  delete from *instructor*
  where *salary* < (**select avg** (*salary*) **from** *instructor*);

- **Problem:** as we delete tuples from deposit, the average salary changes
- Solution used in SQL:
  1. First, compute **avg** salary and find all tuples to delete
  2. Next, delete all tuples found above (without recomputing **avg** or retesting the tuples)

Modification of the Database – Insertion

- Add a new tuple to *course*

  insert into *course*
  values (’CS-437’, ’Database Systems’, ’Comp. Sci.’, 4);

- or equivalently
  insert into *course* (*course_id*, *title*, *dept_name*, *credits*)
  values (’CS-437’, ’Database Systems’, ’Comp. Sci.’, 4);

- Add a new tuple to *student* with *tot_creds* set to null

  insert into *student*
  values (’3003’, ’Green’, ’Finance’, *null*);

Insertion (Cont.)

- Add all instructors to the *student* relation with tot_creds set to 0

```
insert into student
select ID, name, dept_name, 0
from instructor
```

Use 0 instead of NULL as default value

- The **select from where** statement is evaluated **fully** before any of its results are inserted into the relation (otherwise queries like

insert into *table*1 **select** * **from** *table*1

would cause problems, if *table1* did not have any primary key defined.

Modification of the Database – Updates

Increase salaries of instructors whose salary is over $100,000 by 3%, and all others receive a 5% raise

- Write two **update** statements:

update *instructor*
  set *salary* = *salary* * 1.03
  where *salary* > 100000;
update *instructor*
  set *salary* = *salary* * 1.05
  where *salary* <= 100000;

- The order is important
- Can be done better using the **case** statement (next slide)

Case Statement for Conditional Updates

- Same query as before but with case statement

update *instructor*
set *salary* = **case**
when *salary* <= 100000 **then** *salary* * 1.05
else *salary* * 1.03
end

Chapter 6:
Formal Relational Query Languages

Relational Algebra
Relational Calculus

Relational Algebra

- Procedural language
- Six basic operators
  - select: σ (Greek sigma)
  - project: $\prod$ (Greek Pi)
  - union: $\cup$
  - set difference: –
  - Cartesian product: x
  - rename: ρ (Greek rho)
- The operators take one or two relations as inputs and produce a new relation as a result.

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

- $\sigma_{A=B \wedge D > 5}(r)$

A	B	C	D
α	α	1	7
β	β	23	10

Select Operation

- 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:

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

The *instructor* 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

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

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

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

Project Operation

- Notation: $$\Pi_{A_1, A_2, \ldots, A_k}(r)$$

  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*

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

Instructor 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*	*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

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

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

- 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*)

- **Example: to find all courses taught in the Fall 2015 semester, or in the Spring 2016 semester, or in both**

$$\Pi_{course\_id}\,(\sigma_{semester=\text{“Fall”}\ \land\ year=2015}\,(section))\ \cup$$
$$\Pi_{course\_id}\,(\sigma_{semester=\text{“Spring”}\ \land\ year=2016}\,(section))$$

Set difference of two relations

Relations *r*, *s*:

r

A	B
α	1
α	2
β	1

s

A	B
α	2
β	3

r – *s*:

A	B
α	1
β	1

Set Difference Operation

- Notation **r – s**
- Defined as:

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

- 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 2015 semester, but not in the Spring 2016 semester**

$$\Pi_{course\_id}\ (\sigma_{semester=\text{"Fall"}\ \land\ year=2015}\ (section))\ -$$

$$\Pi_{course\_id}\ (\sigma_{semester=\text{"Spring"}\ \land\ year=2016}(section))$$

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

- Notation $r \times s$
- Defined as:

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

- 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.

The *instructor* 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

The *teaches* relation

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

Result of
instructor x teaches

Composition of Operations

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

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

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

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

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

Result of $\sigma_{dept\_name=\text{"Physics"}}(instructor \times teaches)$

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
…	…	…	…	…	…	…	…	…
…	…	…	…	…	…	…	…	…

Rename Operation

- 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.
- Example:

  $\rho_{x}(E)$

  returns the expression *E* under the name *X*
- If a relational-algebra expression *E* has arity *n*, then

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

returns the result of expression *E* under the name *X*, and with the

attributes renamed to $A_1, A_2, ..., A_n$.

Example Query

Find the largest salary in the university

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

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

- **Step 2:** Find the largest salary

$$\Pi_{salary}\ (instructor)\ –\ \Pi_{instructor.salary}\ (\sigma_{\ instructor.salary\ <\ d.salary}\ (instructor\ x\ \rho_d\ (instructor)))$$

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

Result of

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

salary
65000
90000
40000
60000
87000
75000
62000
72000
80000
92000

Largest salary

salary
95000

Result of

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

Example Queries

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

Query 1

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

Query 2

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

name	*course_id*
Einstein	PHY-101

Result of

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