Database Systems

Lecture 6: Introduction to SQL (part2)

Dr. Momtazi
momtazi@aut.ac.ir

based on the slides of the course book

Outline

- Overview of The SQL Query Language
- Data Definition
- Basic Query Structure
- Additional Basic Operations
- **Set Operations**
- Null Values
- Aggregate Functions
- Nested Subqueries
- Modification of the Database

Set Operations

- Find courses that ran in Fall 2009 or in Spring 2010

(**select** *course_id*

from *section*

where *sem* = 'Fall' **and** *year* = 2009)
 union
(**select** *course_id*

from *section*

where *sem* = 'Spring' **and** *year* = 2010)

Set Operations

- Find courses that ran in Fall 2009 and in Spring 2010

```
(select course_id
from section
where sem = ‘Fall’ and year = 2009)
 intersect
(select course_id
from section
where sem = ‘Spring’ and year = 2010)
```

Set Operations

- Find courses that ran in Fall 2009 but not in Spring 2010

  (**select** *course_id*

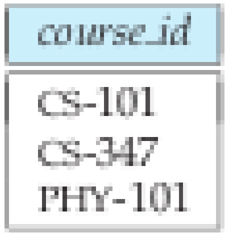
  from *section*

  where *sem* = 'Fall' **and** *year* = 2009)
  except
  (**select** *course_id*

  from *section*

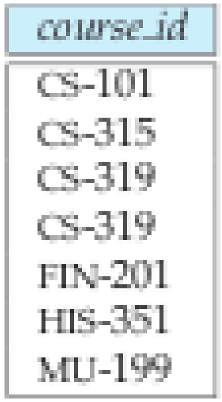
  where *sem* = 'Spring' **and** *year* = 2010)

Example: set operations



course_id
CS-101
CS-347
PHY-101

Figure 3.9 The *c1* relation, listing courses taught in Fall 2009.



course_id
CS-101
CS-315
CS-319
CS-319
FIN-201
HIS-351
MU-199

Figure 3.10 The *c2* relation, listing courses taught in Spring 2010.

Example: set operations

course_id
CS-101

Figure 3.12 The result relation for *c1* intersect *c2*.

course_id
CS-101
CS-315
CS-319
CS-347
FIN-201
HIS-351
MU-199
PHY-101

Figure 3.11 The result relation for *c1* union *c2*.

course_id
CS-347
PHY-101

Figure 3.13 The result relation for *c1* except *c2*.

Set Operations

- Find the salaries of all instructors that are less than the largest salary.
  - **select distinct** *T.salary*
    from *instructor* **as** *T, instructor* **as** *S*
    where *T.salary < S.salary*

- Find all the salaries of all instructors
  - **select distinct** *salary*
    from *instructor*

- Find the largest salary of all instructors.
  - (**select** "second query")
    except
    (**select** "first query")

Set Operations

- Set operations **union**, **intersect**, and **except**
  - Each of the above operations automatically eliminates duplicates

- To retain all duplicates use the corresponding multiset versions **union all, intersect all** and **except all.**

- Suppose a tuple occurs *m* times in *r* and *n* times in *s, then*, it occurs:
  - $m + n$ times in *r* **union all** *s*
  - min(*m*,*n*) times in *r* **intersect all** *s*
  - max(0, *m* – *n*) times in *r* **except all** *s*

Outline

- Overview of The SQL Query Language
- Data Definition
- Basic Query Structure
- Additional Basic Operations
- Set Operations
- **Null Values**
- Aggregate Functions
- Nested Subqueries
- Modification of the Database

Null Values

- It is possible for tuples to have a null value, denoted by *null*, for some of their attributes
- *null* signifies an unknown value or that a value does not exist.
- The result of any arithmetic expression involving *null* is *null*
  - Example: 5 + *null* returns null
- The predicate **is null** can be used to check for null values.
  - Example: Find all instructors whose salary is null.

select *name*
from *instructor*
where *salary* **is null**

Null Values and Three Valued Logic

- Three values – *true*, *false*, *unknown*
- Any comparison with *null* returns *unknown*
  - Example*: 5 < null   or   null <> null   or   null = null*
- Three-valued logic using the value *unknown*:
  - OR: (*unknown* **or** *true*)   = *true*,
    (*unknown* **or** *false*) = *unknown*
    (*unknown* **or** *unknown) = unknown*
  - AND: *(true* **and** *unknown)* *= unknown,*
    (false **and** *unknown) = false,*
    (unknown **and** *unknown) = unknown*
  - NOT*:* (**not** *unknown) = unknown*
  - "*P* **is unknown**" evaluates to true if predicate *P* evaluates to *unknown*
- Result of **where** clause predicate is treated as *false* if it evaluates to *unknown*

Outline

- Overview of The SQL Query Language
- Data Definition
- Basic Query Structure
- Additional Basic Operations
- Set Operations
- Null Values
- **Aggregate Functions**
- Nested Subqueries
- Modification of the Database

Aggregate Functions

- These functions operate on the multiset of values of a column of a relation, and return a value

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

Aggregate Functions

- Find the average salary of instructors in the Computer Science department

select avg (*salary*)
from *instructor*
where *dept_name*= ’Comp. Sci.’;

Aggregate Functions

- Find the total number of instructors who teach a course in the Spring 2010 semester

> **select count** (**distinct** *ID*)
> **from** *teaches*
> **where** *semester* = ’Spring’ **and** *year* = 2010;

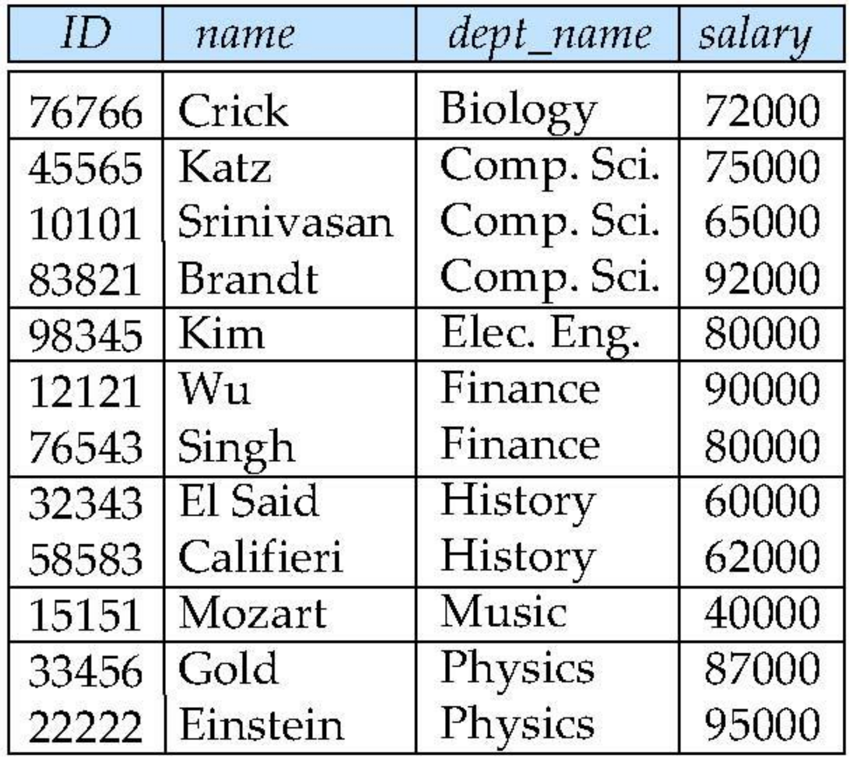
Aggregate Functions

- Find the number of tuples in the *course* relation

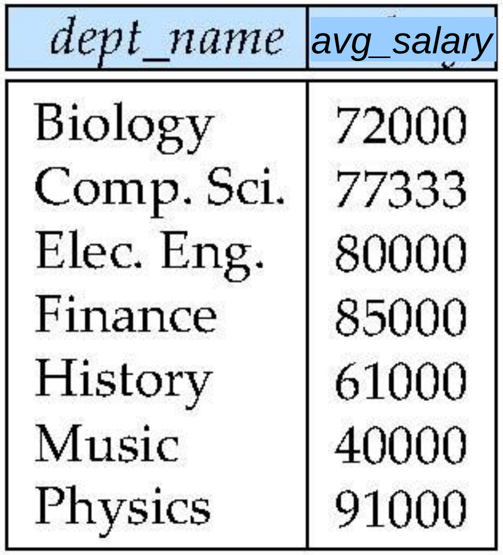
```
select count (*)
from course;
```

Aggregate Functions – Group By

- Find the average salary of instructors in each department
  - **select** *dept_name*, **avg** (*salary*) **as** *avg_salary*
    from *instructor*
    group by *dept_name*;



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



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

Aggregation

- Attributes in **select** clause outside of aggregate functions must appear in **group by** list

/* erroneous query */
select *dept_name*, *ID*, **avg** (*salary*)
from *instructor*
group by *dept_name*;

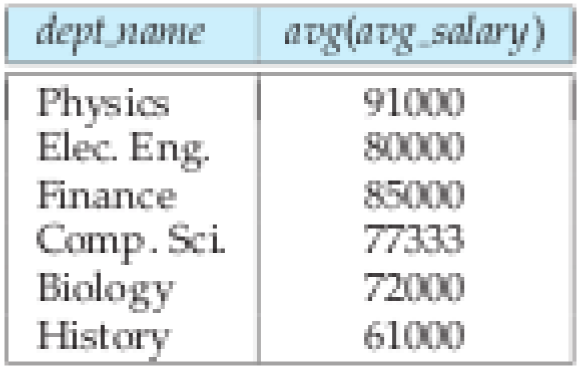
Aggregate Functions – Having Clause

- Find the names and average salaries of all departments whose average salary is greater than 42000

select *dept_name*, **avg** (*salary*)
from *instructor*
group by *dept_name*
having avg (*salary*) > 42000;

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

Example: having clause



dept_name	*avg(avg_salary)*
Physics	91000
Elec. Eng.	80000
Finance	85000
Comp. Sci.	77333
Biology	72000
History	61000

Figure 3.17 The result relation for the query "Find the average salary of instructors in those departments where the average salary is more than $42,000."

Null Values and Aggregates

- Total all salaries

  select sum (*salary*)
  from *instructor*

  - Above statement ignores null amounts
  - Result is *null* if there is no non-null amount

- All aggregate operations except **count(*)** ignore tuples with null values on the aggregated attributes

- What if collection has only null values?
  - count returns 0
  - all other aggregates return null

The picture can't be displayed.

Questions?