Chapter 3: Introduction to SQL (continued)

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

- Any comparison with *null* returns *unknown*
  - Example*: 5 < null   or   null <> null   or   null = null*
- Three-valued logic using the truth 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*

Three-Valued Logic

- To understand how AND, OR, and NOT work in 3-valued logic, think of TRUE = 1, FALSE = 0, and UNKNOWN = ½.
- AND = MIN; OR = MAX, NOT(*x*) = 1-*x*.

Example:

TRUE AND (FALSE OR NOT(UNKNOWN)) =

MIN(1, MAX(0, (1 - ½))) =

MIN(1, MAX(0, ½)) = MIN(1, ½) = ½.

Surprising Example

- From the following Sells relation:

bar	beer	price
Joe’s Bar	Bud	NULL

```
SELECT bar
FROM Sells
WHERE price < 2.00 OR price >= 2.00;
```

UNKNOWN UNKNOWN

UNKNOWN

Reason:
2-Valued Laws != 3-Valued Laws

- Some common laws, like commutativity of AND, hold in 3-valued logic.
  - Unknown AND False AND True = False AND True AND Unknown
- But not others,
- e.g., the *law of the excluded middle* : *p* OR NOT *p* = TRUE.
  - When *p* = UNKNOWN,

    MAX(½, (1 – ½)) = ½ != 1.

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 (Cont.)

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

  -
    ```
    select avg (salary)
    from instructor
    where dept_name= ’Comp. Sci.’;
    ```
- Find the total number of instructors who teach a course in the Spring 2015 semester

  -
    ```
    select count (distinct ID)
    from teaches
    where semester = ’Spring’ and year = 2015
    ```
- Find the number of tuples in the *course* relation

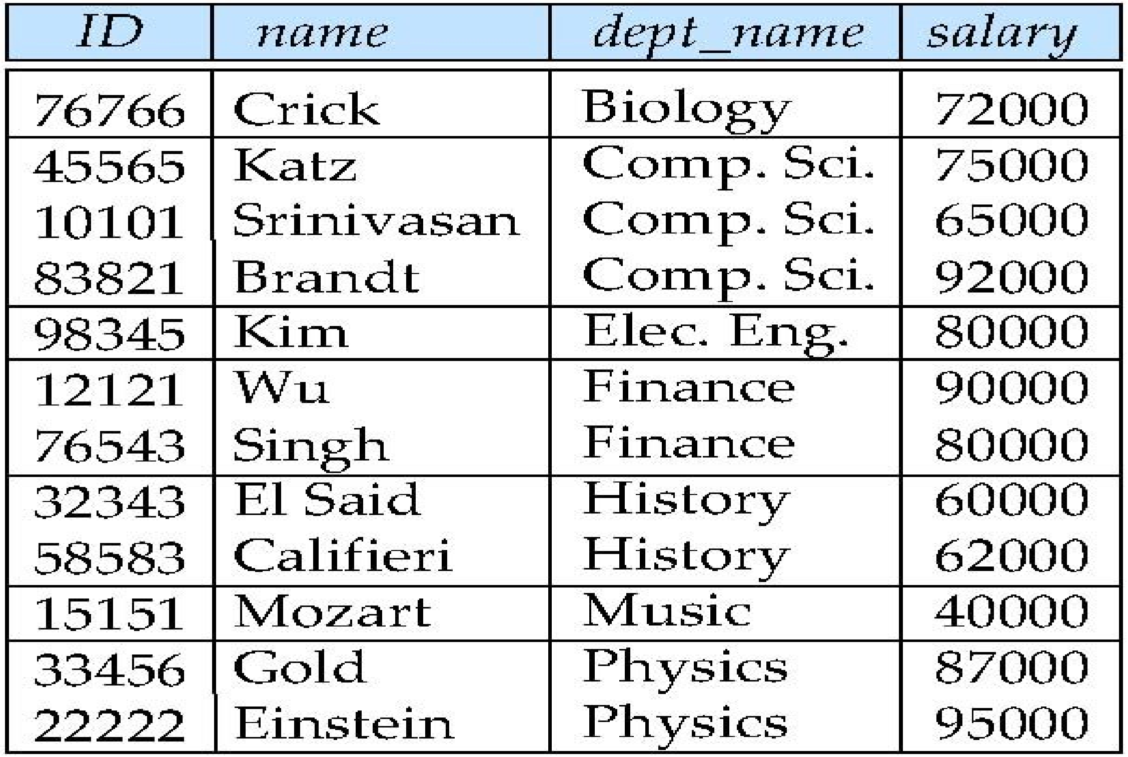
  -
    ```
    select count (*)
    from course;
    ```

Aggregate Functions – Group By

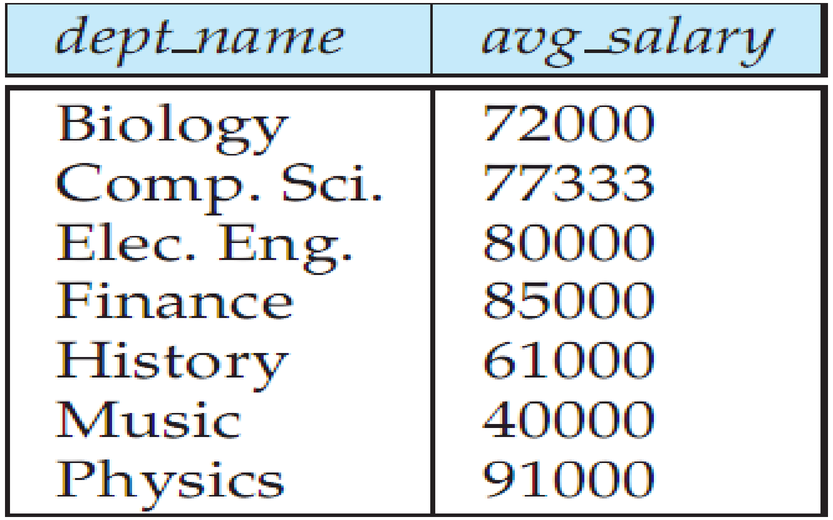
- Find the average salary of instructors in each department

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

  Note: departments with no instructor will not appear in result



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

Queries With GROUP BY and HAVING

SELECT [DISTINCT] *target-list*
FROM *relation-list*
WHERE *qualification*
GROUP BY *grouping-list*
HAVING *group-qualification*

- The *target-list* contains **(i) attribute names** (ii) terms with aggregate operations (e.g., MIN (*S.age*)).

- The **attribute names in (i) above** must be a subset of ***grouping-list***. Intuitively, each answer tuple corresponds to a ***group***, and these attributes **must have a single value per group.**
  (A ***group*** is a set of tuples that have the same value for all attributes in ***grouping-list***.)

Interpretation Semantics (Conceptual Evaluation Strategy)

- ***Interpretation Semantics*** (different than the ***Execution Semantics***):
  - Compute the cross-product of ***relation-list*.**
  - Discard resulting tuples if they fail ***qualifications*.**
  - Delete attributes that are not in ***target-list*.**
  - If **DISTINCT** is specified, eliminate duplicate rows.

- This strategy is the least efficient way to compute a query! An optimizer will find more efficient strategies to compute ***the same answer*.**

Grouping

- We may follow a SELECT-FROM-WHERE expression by GROUP BY and a list of attributes.
- The relation that results from the SELECT-FROM-WHERE is grouped according to the values of all those attributes, and any aggregation is applied only within each group.

Aggregation (Cont.)

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

Example: “Find average salary of instructors for each department. Instructor (ID, name, dept_name, salary)

```
/* erroneous query */

select dept_name, ID, avg (salary)
from instructor
group by dept_name;
```

What is the correct SQL query?

Remove ID

Query: "Find the number of instructors in each department who teach a course in the Fall 2009 semester."

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)

In SQL?

Query: “Find the number of instructors in each department who teach a course in the Fall 2009 semester.”

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)

```
SELECT I.dept_name, Count (distinct T.ID)
FROM Instructor I, Teaches T
WHERE I.ID= T.ID
    AND  T.semester=“Fall”
    AND   T.year= “2009”
GROUP BY   I.dept_name
```

Query: “For each department, find the number of instructors who teach a course offered by that department in the Fall 2009 semester.”

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)

```
SELECT D.dept_name, Count (T.ID)
FROM Department D, Teaches T, COURSE C
WHERE D.dept_name = C.dept_name // A course offered by the department
    AND  C.course_id = T.course_id
    AND  T.semester=‘Fall’
    AND   T.year= ‘2009’
GROUP BY   I.dept_name
```

Is this SQL query correct?

Missing distinct!

Query: “For each department, find the number of instructors who teach a course offered by that department in the Fall 2009 semester.”

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)

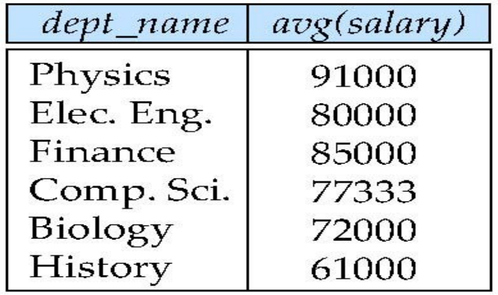
```
SELECT I.dept_name, Count (distinct T.ID)
FROM Instructor I, Teaches T, COURSE C
WHERE I.dept_name = C.dept_name //A course offered by the department
     AND  C.course_id = T.course_id
     AND  T.semester=‘Fall’
     AND   T.year= ‘2009’
GROUP BY   I.dept_name
```

Aggregate Functions – 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



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

The result relation for the query “Find the average salary of instructors in those departments where the average salary is more than $60,000.”

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

Next:

More SQL, Nested Subqueries, Correlated subqueries, ...