Basic sql- part 3

Natural join to null values in aggregate function

Natural Join

- Natural join matches tuples with the same values for all common attributes, and retains only one copy of each common column
- **select * from** *instructor* **natural join** *teaches*;

ID	*name*	*dept_name*	*salary*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	CS-347	1	Fall	2009
12121	Wu	Finance	90000	FIN-201	1	Spring	2010
15151	Mozart	Music	40000	MU-199	1	Spring	2010
22222	Einstein	Physics	95000	PHY-101	1	Fall	2009
32343	El Said	History	60000	HIS-351	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-101	1	Spring	2010
45565	Katz	Comp. Sci.	75000	CS-319	1	Spring	2010
76766	Crick	Biology	72000	BIO-101	1	Summer	2009
76766	Crick	Biology	72000	BIO-301	1	Summer	2010
83821	Brandt	Comp. Sci.	92000	CS-190	1	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-190	2	Spring	2009
83821	Brandt	Comp. Sci.	92000	CS-319	2	Spring	2010
98345	Kim	Elec. Eng.	80000	EE-181	1	Spring	2009

Natural Join Example

- List the names of instructors along with the course ID of the courses that they taught.
- In two ways we can write the Query:
  - **select** *name*, *course_id* **from** *instructor, teaches* **where** *instructor.ID* = *teaches.ID*;

OR

  - **select** *name*, *course_id* **from** *instructor* **natural join** *teaches*;

- A **from** clause in an SQL query can have multiple relations combined using natural join, as shown here:
- **select** *A*1, *A*2, . . . , *An*
- **from** *r*1 **natural join** *r*2 **natural join** . . . **natural join** *rm*
- **where** *P*;

- Retrieve the names of all instructors, along with their department names and department building name.
- instructor(ID, name, dept_name, salary)
- department(dept_name, building, budget)

- Query:
- Select name, dept_name , building from instructor, department where instructor.dept_name=department.dept_name
- **Using natural join**

- Display the **computer science courses** which are held in **room no**=105 and building AB5
- course(course_id, title, dept_name, credits)
- section(course_id, sec_id, semester, year, building, room_number, time_slot_id)

- select course_id from course, section where course.course_id=section.course_id and building='AB5' and room_number=105 and dept_name='CS';
- **using natural join:**

Natural Join (Cont.)

- Danger in natural join: beware of unrelated attributes with same name which get equated incorrectly
- course(course_id, title, dept_name, credits)
- teaches(ID, course_id, sec_id, semester, year)
- instructor(ID, name, dept_name, salary)
- List the names of instructors along with the titles of courses that they teach
  - **select** *name*, *title*
    from *instructor* **natural join** *teaches* **natural join** *course*;

the natural join of *instructor* and *teaches* contains the attributes (*ID*, *name*, *dept name*, *salary*, *course id*, *sec id, sem, year*),

while the *course* relation contains the attributes (*course id*, *title*, *dept name*, *credits*).

As a result, the natural join of these two would require that the *dept name* attribute values from the two inputs be the same, in addition to requiring that the *course id* values be the same. This query would then omit all (instructor name, course title) pairs where the instructor

teaches a course in a department other than the instructor's own department.

Instructor natural join teaches

- (*ID*, *name*, *dept name*, *salary*, *course id*, *sec id*, *sem*, *year*), ①

Course relation is as follows :-

- (*course id*, *title*, *dept name*, *credits*). ②

If ① natural join ②

common attributes courseid, deptname

- This query would then omit all (instructor name, course title) pairs where the instructor teaches a course in a department other than the instructor's own department.

instructor.dept-name = course.dept-name ✗

List the names of instructors along with the titles of courses that they teach

correct version

```
select name, title
  from (instructor natural join teaches)
        join course using(course_id);
```

- The operation **join** . . . **using** requires a list of attribute names to be specified.
- No other common attribute even if present will be matched, other than the ones indicated in using clause.

- *Display id of students who have taken the course **offered by** physics department*
- student(**SID**, name, dept_name, tot_cred)
- takes(**SID**, course_id, sec_id, semester, year , grade)
- course(course_id, title, dept_name, credits)
-
-
- Select student.SID from student, course, takes where student.SID= takes.SID and course.course_id =takes.course_id and course.dept_name="physics" ;

- **Select student.SID from (student natural join takes), join course using course_id where course.dept_name="physics**

- Display the student' details who scored A or A+ grade in "DBS" course.
- student(**SID**, name, dept_name, tot_cred)
- takes(**SID**, course_id, sec_id, semester, year, **grade**)
- course(course_id, **title**, dept_name, credits)

Additional Basic Operations

The Rename Operation

The names of the attributes in the result are derived from the names of the attributes in the relations in the **from** clause.

We may need to change this for the following reasons:

- two relations in the **from** clause may have attributes with the same name,

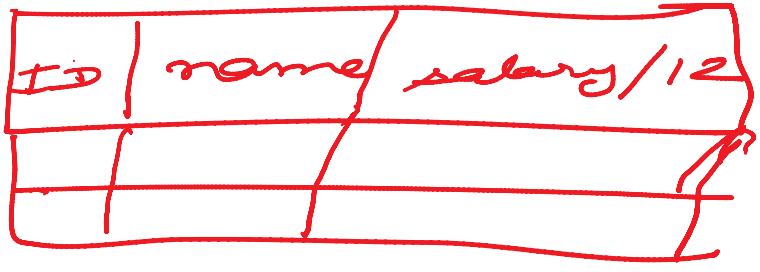
in which case an attribute name is duplicated in the result.

- if we used an arithmetic expression in the **select** clause, the resultant attribute does not have a name.
- even if an attribute name can be derived from the base relations as in the preceding example, we may want to change the attribute name in the result.

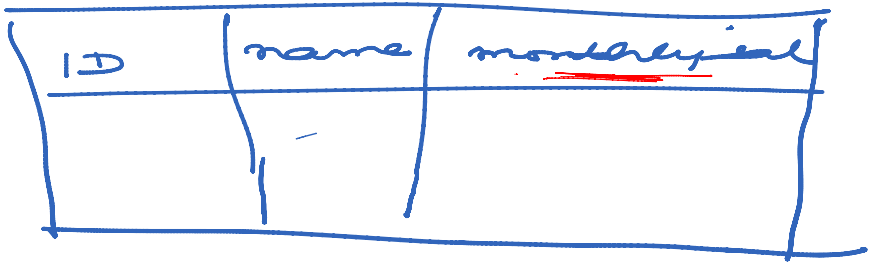
- The SQL allows renaming relations and attributes using the **as** clause:

  old-name **as** *new-name*

- E.g.
  - **select** *ID, name, salary/12* **as** *monthly_salary* **from** *instructor*



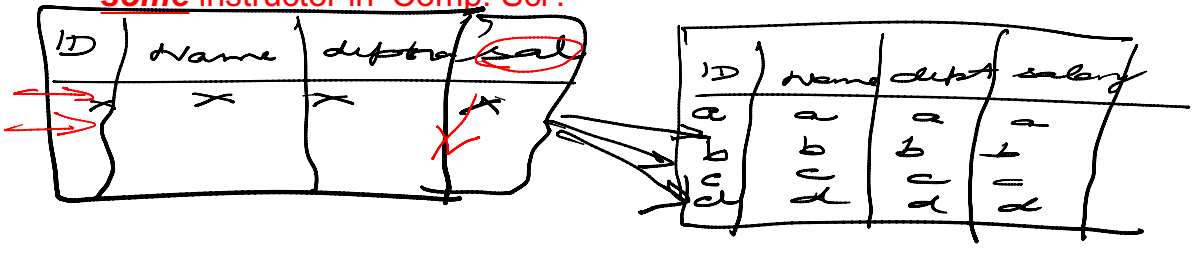
After rename



- One reason to rename a relation is to replace a long relation name with a shortened version that is more convenient to use elsewhere in the query.
- **select** *T.name*, *s.course id*
- **from** *instructor* **as** *T*, *teaches* **as** *s*
- **where** *T.ID*= *s.ID*;

Another reason to rename a relation is a case where we wish to compare tuples in the same relation. We then need to take the Cartesian product of a relation with itself and, without renaming, it becomes impossible to distinguish one tuple from the other.

- Find the names of all instructors who have a higher salary than ***some*** instructor in 'Comp. Sci'.



- **select distinct** *T.name*
  from *instructor* **as** *T*, *instructor* **as** *S*
  where *T.salary > S.salary* **and** *S.dept_name = 'Comp. Sci.'*

An identifier, such as *T* and *S*, that is used to rename a relation is referred to as a **correlation name** in the SQL standard, but is also commonly referred to as a **table alias**, or a **correlation variable**, or a **tuple variable**.

String Operations

Note: The SQL standard specifies that the equality operation on strings is case sensitive; as a result the expression "'comp. sci.' = 'Comp. Sci.'" **evaluates to false.**
However, some database systems, such as **MySQL and SQL Server**, do not distinguish uppercase from lowercase when matching strings; as a result "'comp. sci.'= 'Comp. Sci.'" would evaluate to **true** on these databases. This

- SQL includes a string-matching operator for comparisons on character strings. The operator "like" uses patterns that are described using two special characters:
  - percent (%). The % character matches any substring.
  - underscore (_). The _ character matches any character.
- Find the names of all instructors whose name includes the substring "dar".

  select *name* **from** *instructor* **where** *name* **like** '%dar%'

- Patters are case sensitive.
- Pattern matching examples:
  - ‘Intro%’ matches any string beginning with “Intro”.
  - ‘%Comp%’ matches any string containing “Comp” as a substring.
  - ‘_ _ _’ matches any string of exactly three characters.
  - ‘_ _ _ %’ matches any string of at least three characters.

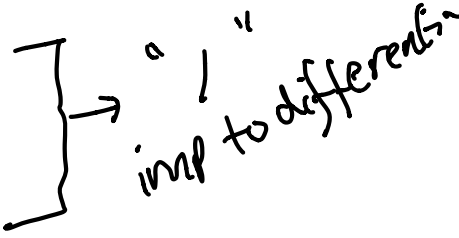
exercises

- display all the records from the CUSTOMERS table, where the SALARY starts with 200.

```
SELECT * FROM table_name WHERE column LIKE 'X%'

SELECT * FROM table_name WHERE column LIKE '%X%'

SELECT * FROM table_name WHERE column LIKE 'X_'

SELECT * FROM table_name WHERE column LIKE '_X'

SELECT * FROM table_name WHERE column LIKE '_X_'
```

String Operations (Cont.)

- Match the string “100 %” // Usage of % in a string
- ‘\’ is used to specify the special character such as % or \.
  (eg. Retrieve whose attendance status is 100%)
  Select names from student where attendance like ‘100\%’



- SQL supports a variety of string operations such as
  - concatenation (using “||”)
  - converting from upper to lower case (and vice versa)
  - finding string length, extracting substrings, etc.

Eg.: Select SUBSTR(‘ABCDEF’, 2, 3) as substring from tablename ; o/p: BCD
select length(name) as length_val from dual ;
select upper(‘hi’) as Upper_val from dual ;

Ordering the Display of Tuples

- List in alphabetic order the names of all instructors

  select distinct *name*
  from *instructor*
  order by *name*

- We may specify **desc** for descending order or **asc** for ascending order, for each attribute; ascending order is the default.
  - Example: **order by** *name* **desc**

- Can sort on multiple attributes
  - Example: **order by** *dept_name, name*

  - *General Form:*
    - *SELSECT* **distinct** *name*
      FROM instructor
    - *WHERE Dept_name='CS'*
    - *OREDER BY name ;*

Where Clause Predicates

- SQL includes a **between** comparison operator
- Example: Find the names of all instructors with salary between $90,000 and $100,000 (that is, ≥ $90,000 and ≤ $100,000)
  - **select** *name*
    from *instructor*
    where *salary* **between** 90000 **and** 100000
- Tuple comparison
  - **select** *name*, *course_id*
    from *instructor*, *teaches*
    where (*instructor.ID*, *dept_name*) = (*teaches.ID*, 'Biology');
- All SQL platforms may not support this syntax

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)

Interpretation is important!

- 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)

- 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)

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**.

Set Operations (Cont.)

- 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")
```

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;**

ID	Student	Email1	Email2	Phone	Father
1	Stan Marsh	NULL	smarsh@gmail.com	740-097-0951	Randy Marsh
2	Butters Stotch	bstotch@spelementary.com	bstotch@gmail.com	NULL	Stephen Stotch
3	Kenny McCormick	NULL	NULL		Stuart McCormick
4	Kyle Broflovski	kbroflovski@spelementary.com	kbroflovski@gmail.com	619-722-2491	Gerald Broflovski
666	Eric Cartman	ecartman@spelementary.com	ecartman@gmail.com	740-6733-5671	NULL

ID	Student	Email1	Email2
3	Kenny McCormick	NULL	NULL

- **SELECT** ID, Student,Email1,Email2
  FROM tblSouthPark **WHERE** Email1 **IS NULL** AND Email2 **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 P is not know

Aggregate Functions

Aggregate functions are functions that take a collection (a set or multiset) of values as input and return a single value. SQL offers five built-in aggregate functions:

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 2010 semester
  - **select count** (**distinct** *ID*) [Distinct: Since same teacher teaching more than one subject in Spring 10]
    from *teaches*
    where *semester* = 'Spring' **and** *year* = 2010
- Find the number of tuples in the *course* relation
  - **select count** (*)
    from *course*;

“group by” clause

- There are circumstances where we would like to apply the aggregate
- function not only to a single set of tuples, but also to a group of sets of
- tuples; we specify this wish in SQL using the **group by** clause.

- The attribute or attributes given in the **group by** clause are used to form groups.
- Tuples with the same value on all attributes in the **group by** clause are placed in one group.

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

- Find the number of instructors in each department who teach a course in the Spring 2

- **select** *dept name*, **count** (**distinct** *ID*) **as** *instr_count*
- **from** *instructor* **natural**
- **where** *semester* = 'Spri
- **group by** *dept name*; 0

dept_name	*instr_count*
Comp. Sci.	3
Finance	1
History	1
Music	1

Aggregation (Cont.)

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

In other words, any attribute that is not present in the **group by** clause must appear only inside an aggregate function if it appears in the **select** clause, otherwise the query is treated as erroneous.

- /* erroneous query */

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

Each instructor in a particular group (defined by *dept name*) can have a different *ID*, and since only one tuple is output for each group, there is no unique way of choosing which *ID* value to output. As a result, such cases are disallowed by SQL.

exercise

ord_no	purch_amt	ord_date	customer_id	salesman_id
70001	150.5	2012-10-05	3005	5002
70009	270.65	2012-09-10	3001	5005
70002	65.26	2012-10-05	3002	5001
70004	110.5	2012-08-17	3009	5003
70007	948.5	2012-09-10	3005	5002
70005	2400.6	2012-07-27	3007	5001
70008	5760	2012-09-10	3002	5001

- Write a SQL statement to find the total purchase amount of all orders
  Select sum(purch_amt) from orders;

- Write a SQL statement to find the number of salesmen currently listing for all of their customers.
  Select count(distinct salesman_id) from orders;

ord_no	purch_amt	ord_date	customer_id	salesman_id
70001	150.5	2012-10-05	3005	5002
70009	270.65	2012-09-10	3001	5005
70002	65.26	2012-10-05	3002	5001
70004	110.5	2012-08-17	3009	5003
70007	948.5	2012-09-10	3005	5002
70005	2400.6	2012-07-27	3007	5001
70008	5760	2012-09-10	3002	5001

```
1  SELECT customer_id,ord_date,MAX(purch_amt)
2  FROM orders
3  GROUP BY customer_id,ord_date;
```

Output of the Query:

customer_id	ord_date	max
3002	2012-10-05	65.26
3003	2012-08-17	75.29
3005	2012-10-05	150.50
3007	2012-07-27	2400.60
3009	2012-08-17	110.50
3001	2012-09-10	270.65
3002	2012-09-10	5760.00
3005	2012-09-10	948.50
3009	2012-10-10	2480.40
3008	2012-06-27	250.45
3004	2012-10-10	1983.43
3002	2012-04-25	3045.60

customer_id	cust_name	city	grade	salesman_id
3002	Nick Rimando	New York	100	5001
3007	Brad Davis	New York	200	5001
3005	Graham Zusi	California	200	5002
3008	Julian Green	London	300	5002
3004	Fabian Johnson	Paris	300	5006
3009	Geoff Cameron	Berlin	100	5003
3003	Jozy Altidor	Moscow	200	5007

- Write a SQL statement to know how many customer have listed their names.
- find the number of customers who gets at least a gradation for his/her performance.
- find the highest purchase amount by the each customer with their ID and highest purchase amount
- SELECT customer_id, MAX(purch_amt) FROM orders GROUP BY customer_id;

“Group by… having “ clause

- At times, it is useful to state a condition that applies to groups rather than to tuples.
- For example, we might be interested in only those departments where the average salary of the instructors is more than $42,000.
- This condition does not apply to a single tuple; rather, it applies to each group constructed by the **group by** clause.
- To express such a query, we use the **having** clause of SQL. SQL applies
- predicates in the **having** clause after groups have been formed, so aggregate functions may be used

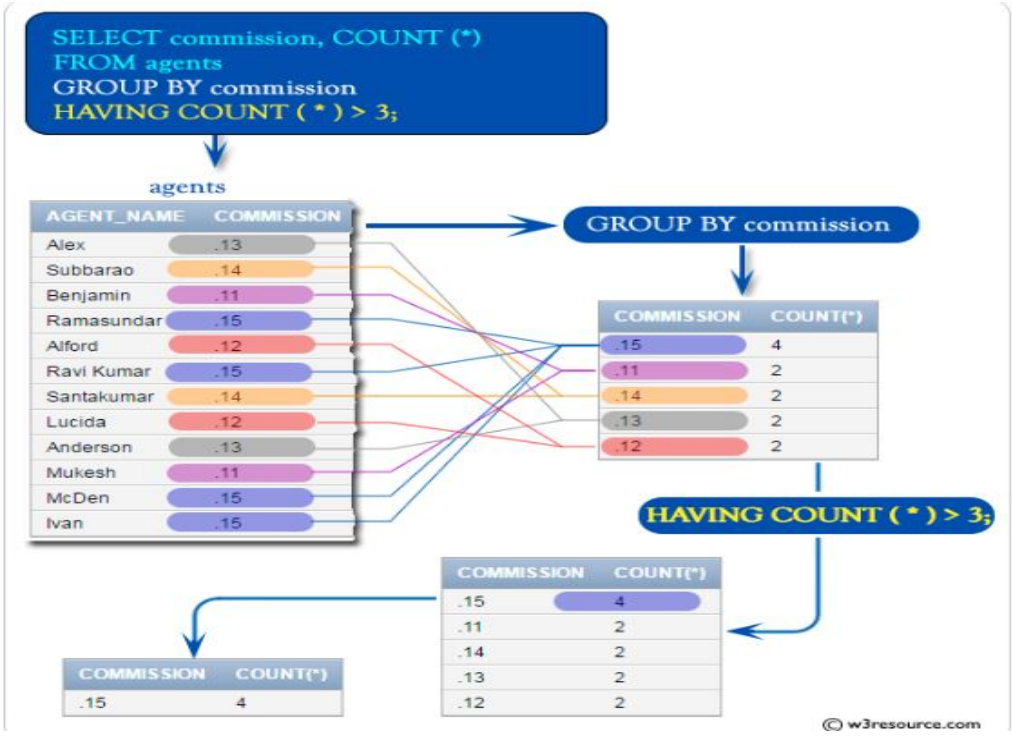
Aggregate Functions – Having Clause

- 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;
```

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

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


SELECT commission, COUNT (*)
FROM agents
GROUP BY commission
HAVING COUNT (*) > 3;
agents
AGENT_NAME
COMMISSION
Alex .13
Subbarao .14
Benjamin .11
Ramasundar .15
Alford .12
Ravi Kumar .15
Santakumar .14
Lucida .12
Anderson .13
Mukesh .11
McDen .15
Ivan .15
GROUP BY commission
COMMISSION COUNT(*)
.15 4
.11 2
.14 2
.13 2
.12 2
HAVING COUNT (*) > 3;
COMMISSION COUNT(*)
.15 4
.11 2
.14 2
.13 2
.12 2
COMMISSION COUNT(*)
.15 4
© w3resource.com

- To illustrate the use of both a **having** clause and a **where** clause in the same query, we consider the query “For each course section offered in 2009, find the average total credits (*tot cred*) of all students enrolled in the section, if the section had at least 2 students.”

- **select** *course _id*, *semester*, *year*, *sec_id*, **avg** (*tot cred*)
- **from** *takes* **natural join** *student*
- **where** *year* = 2009
- **group by** *course id*, *semester*, *year*, *sec _id*
- **having count** (*ID*) >= 2;

Null Values and Aggregates

- Total all salaries

  select sum (*salary*)
  from *instructor*

  - Above statement ignores null amounts

- 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

Table1:

```
Field1    Field2    Field3
-------------------------
  1         1         1
 NULL      NULL      NULL
  2         2         NULL
  1         3         1
```

Then

```
SELECT COUNT(*), COUNT(Field1), COUNT(Field2), COUNT(DISTINCT Field3)
FROM Table1
```

Output Is:

```
COUNT(*) = 4; -- count all rows even null/duplicates

-- count only rows without null values on that field
COUNT(Field1) = COUNT(Field2) = 3

COUNT(Field3) = 2
COUNT(DISTINCT Field3) = 1 -- Ignore duplicates
```