Intro to DB

CHAPTER 3
SQL

Chapter 3: SQL

- Overview
- Data Definition
- Basic Structure
- Additional Basic Operations
- Set Operations
- Null Values
- Aggregate Functions
- Nested Subqueries
- Modification of the Database

History

- IBM ***Sequel*** language developed as part of System R project at the IBM San Jose Research Laboratory
- Renamed ***Structured Query Language (SQL)***
- ANSI and ISO standard SQL:
  - SQL-86, SQL-89, SQL-92
  - SQL:1999 (Y2K!), SQL:2003, SQL:2008
- Commercial systems offer most, if not all, SQL-92 features, plus varying feature sets from later standards and special proprietary features.
  - Not all examples here may work on your particular system.

Create Table Construct

DDL: data definition Language

- An SQL relation is defined using the **create table** command:

  create table *r* (A_1 D_1, A_2 D_2, ..., A_n D_n,
  (integrity-constraint$_1$),
  ...,
  (integrity-constraint$_k$))

  table name

  list of Attrs

  - *r* is the name of the relation
  - each A_i is an attribute name in the schema of relation *r*
  - D_i is the data type of values in the domain of attribute A_i

- Example:

  create table *instructor* (

ID	**char**(5),
name	**varchar**(20) **not null,**
dept_name	**varchar**(20),
salary	**numeric**(8,2))

  Attr

  data type & condition

  5 length char

  Floating with 2 under point

- **insert into** *instructor* **values** ('10211', 'Smith', 'Biology', 66000);

  insert one row(tuple) of instructor

Domain Types in SQL

data type

- **char(n):** Fixed length character string, with user-specified length *n.*
- **varchar(n):** Variable length character strings, with user-specified maximum length *n.*
- **int:** Integer (a finite subset of the integers that is machine-dependent).
- **smallint:** Small integer (a machine-dependent subset of the integer domain type).
- **numeric(p, d):** Fixed point number, with user-specified precision of *p* digits, with *n* digits to the right of decimal point.
- **real, double precision:** Floating point and double-precision floating point numbers, with machine-dependent precision.
- **float(n):** Floating point number, with user-specified precision of at least *n* digits.

Null values are allowed in all the domain types.

무결성

Integrity Constraints in Create Table

- **not null** (null validate) Condition
- **primary key** (A_1, ..., A_n) : set primary (must uniqueness + not null)
- **foreign key** (A_m, ..., A_n) **references** r refer target
  must exist inside refer target

Declare *dept_name* as to be a reference to *department.*

create table *instructor* (
ID **char**(5),
name **varchar**(20) **not null,**
dept_name **varchar**(20),
salary **numeric**(8,2),
primary key (*ID*),
foreign key *(dept_name*) **references** *department)*

- **primary key** declaration on an attribute automatically ensures **not null**

And a Few More Relation Definitions

- **create table** *student* (
  ID **varchar**(5),
  name **varchar**(20) **not null**,
  dept_name **varchar**(20),
  tot_cred **numeric**(3,0),
  primary key (*ID*),
  foreign key *(dept_name*) **references** *department)*);
- **create table** *takes* (
  ID **varchar**(5),
  course_id **varchar**(8),
  sec_id **varchar**(8),
  semester **varchar**(6),
  year **numeric**(4,0),
  grade **varchar**(2),
  primary key *(ID, course_id, sec_id, semester, year),*
  foreign key (*ID*) **references** *student,*
  foreign key (*course_id, sec_id, semester, year*) **references** *section*);
  - Note: *sec_id* can be dropped from primary key above, to ensure a student cannot be registered for two sections of the same course in the same semester

And more still

- **create table** *course* (
  course_id **varchar**(8) **primary key**,
  title **varchar(**50),
  dept_name **varchar**(20),
  credits **numeric**(2,0),
  foreign key *(dept_name*) **references** *department)*);

  - Primary key declaration can be combined with attribute declaration as shown above

Drop and *Alter Table* Constructs

destruct table

- **drop table:** deletes all information about the dropped relation from the database.

target relation

- **alter table**: used to add or drop attributes to an existing relation

  alter table *r* **add** *A D* (*D* → Domain)

  where *A* is the name of the attribute to be added to relation *r* and *D* is the domain of *A.*

  - *null* is assigned to the new attribute for each tuple

  Attr

  alter table *r* **drop** *A*

  where *A* is the name of an attribute of relation *r*

  - (dropping of attributes is not supported by many databases)

Basic Structure of SQL Queries

- SQL is based on set and relational operations with certain modifications and enhancements
- A typical SQL query has the form:

  - A_is represent attributes
  - r_is represent relations
  - P is a predicate.

- The result of an SQL query is a relation.

The *select* Clause

- Find the names of all instructors:

  select *name*
  from *instructor*

  $\Pi_{name}(instructor)$

- An asterisk in the select clause denotes "all attributes"

  select *
  from *instructor*

  instructor

- NOTE:
  - SQL does not permit the '-' character in names (use '_' in a real implementation).
  - SQL names are case insensitive.

The *select* Clause (cont.)

Basically, relation algebra is "Set" / no dups

But in real SQL → no dups make extra overhead

∴ default is approve dups

- SQL allows duplicates in relations as well as in query results.
- To force the elimination of duplicates, insert the keyword **distinct** after **select.**

- Find the names of all departments with instructor, and remove duplicates

```
select distinct dept_name
from instructor
```

- The keyword **all** specifies that duplicates not be removed (default = all)

```
select all dept_name
from instructor
```

The *select* Clause (cont.)

- The **select** clause can contain arithmetic expressions
  - +, –, *, /
  - on constants or attributes of tuples.
- The query:

select *ID, name, salary/12*
from *instructor*

would return a relation that is the same as the *instructor* relation, except that the value of the attribute *salary* is divided by 12.

The *where* Clause

- Corresponds to the selection predicate of the relational algebra.
- Predicate involving attributes of the relations that appear in the **from** clause.
- Find all instructors in Comp. Sci. dept with salary > 80000

select *name*
from *instructor*
where *dept_name* = 'Comp. Sci.' **and** *salary* > 80000

where execute first than select

$\sigma_{dept\_name = 'Comp.\ Sci.' \wedge salary > 80000}(instructor)$

- Comparison conditions: >, <, =, <=, >=, !=
- Logical connectives: **and, or, not**
- Comparisons can be applied to results of arithmetic expressions
- SQL includes a **between** comparison operator

where *salary* **between** 90000 **and** 100000

= $90000 \le x \le 10000$

→ can be differ via dbms

The *from* Clause

- Lists the relations to be scanned in the evaluation of the expression.
- Corresponds to the Cartesian product operation of the relational algebra.
- *Instructor* x *teaches*

select *
from *instructor, teaches* (Conceptually execute Cartesian)

⇒ instructor X teaches

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

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

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	Finance	90000	10101	CS-347	1	Fall	2009
12121	Wu	Finance	90000	12121	FIN-201	1	Spring	2010
12121	Wu	Finance	90000	15151	MU-199	1	Spring	2010
12121	Wu	Finance	90000	22222	PHY-101	1	Fall	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

select ∗ **from** *instructor, teaches*

Joins

Execute Sequence : from(X) ⟶ where(σ) ⟶ select(π)

- For all instructors who have taught some course, find their names and the course ID of the courses they taught.

```
select name, course_id
from instructor, teaches
where instructor.ID = teaches.ID
```

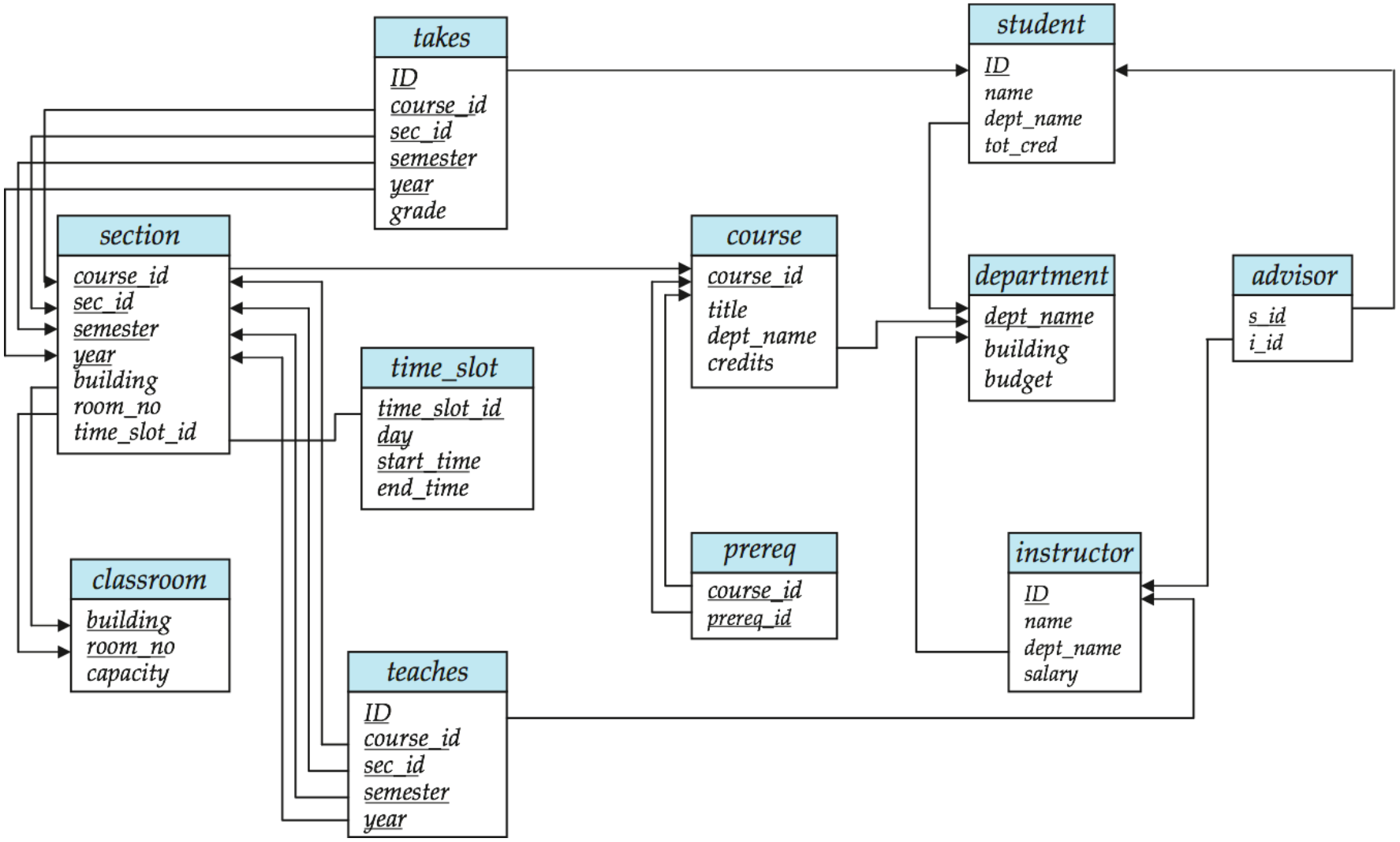
⇒ instructor $\bowtie_{ID}$ teaches

- Find the course ID, semester, year and title of each course offered by the Comp. Sci. department

```
select section.course_id, semester, year, title
from section, course
where section.course_id = course.course_id and
      dept_name = 'Comp. Sci.'
```

$\Pi_{\sim}(\sigma_{dept\_name='Comp.\ Sci.'}(section \bowtie_{id} course))$

Schema Diagram for the University



SQL Examples

- Find the IDs of students advised by an instructor named Einstein.

```
Select s_id
from advisor, instructor
where advisor.i_id = instructor.ID and name = 'Einstein'
```

- Find the IDs of all students who were taught by an instructor named Einstein; make sure there are no duplicates in the result.

```
Select distinct takes.ID  ~> prevent ambiguous referenced
from takes, sections, teaches, instructor
where takes.course_ID = sections.course_ID and (year, sec, semester also)
      sections.course_ID = teaches.course_ID and teaches.ID = Instructor.ID and
      Instructor.name = 'Einstein'
```

The *Rename* Operation

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

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

  select *ID, name, salary/12* **as** *monthly_salary*
  from *instructor*

- 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* : important ⟶ inst x inst
  where *T.salary* > *S.salary* **and** *S.dept_name* = *'Comp. Sci.'*

- *S*, *T* are called ***tuple variables***
- Keyword **as** is optional and may be omitted

  instructor **as** *T* ≡ *instructor T*

String Operations

- For comparisons on character strings
- Patterns are described using special characters:
  - percent (%): matches any substring.
  - underscore (_). matches any character.
- Find all courses whose title includes the substring “data”.

```
select *
from course
where title like '%data%'
```

ex) like '% Comp. Sci %'
↳ Wild Card of string
⇒ ∀ string

- Escape character to specify % and \ within string

```
like ‘100\%’
```

↳ % character

like '___' : matching with ∀ character

- A variety of string operations such as
  - concatenation (using “||”)
  - case conversion, string length, substrings, etc.

Ordering the Display of Tuples

- List in alphabetic order all instructors

  select *
  from *instructor*
  order by *name*

- **desc** for descending order or **asc** for ascending order (default)
  - **order by** *name* **desc**

- Can sort on multiple attributes
  - **order by** *dept_name, name*
  - **order by** *dept-name* **desc**, *name* **asc**

Set Operations

Same as relational algebra

- The set operations: **union, intersect, except** correspond to the relational algebra operations $\cup$, $\cap$, $-$.
- *Find all names that appear in instructor, student, or both:*

  (select *name* **from** *instructor*)
  union
  (select *name* **from** *student)*

  $i \cup s$

- *Find instructor names that are also student names.*

  (**select** *name* **from** *instructor*)
  intersect
  (select *name* **from** *student)*

  $i \cap s$

- *Find instructor names that are not student names.*

  (**select** *name* **from** *instructor*)
  except
  (select *name* **from** *student)*

  $i - s$

Set Operations (cont.)

- Each set operation *automatically eliminates duplicates*
- To retain all duplicates use 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*

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 predicate **is null** can be used to check for null values.

Find all instructors whose salary is null.

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

- Why not the following?

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

☆ hope!

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	*null*
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	*null*

Null Values and Three Valued Logic

- Any arithmetic expression involving *null* is *null*
  - Example: 5 + *null* returns null
- Any comparison with *null* returns *unknown* (True, False, 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*
- Result of **where** clause predicate is treated as ***false*** if it evaluates to ***unknown***
  - "*P* **is unknown**" evaluates to true if predicate *P* evaluates to *unknown*

Aggregate Functions : Statistic function

- 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.';
```

after 'salary' Querying then avg run with that result

- Find the total number of instructors who teach a course in the Spring 2021 semester

```
select count (distinct ID)
from teaches
where semester = 'Spring' and year = 2021
```

Counting distinct ID

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

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

instructor

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

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

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

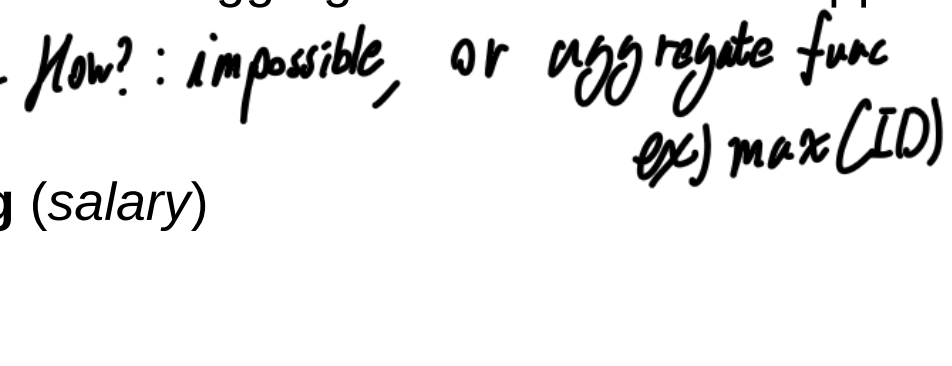
group by

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

avg result

Group By (Cont.)

- 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*;



Having : group version where

- Find the names and average salaries of all departments whose average salary is greater than 50000

```
select dept_name, avg (salary)
from instructor
group by dept_name
having avg (salary) > 50000;
```

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

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

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	*null*
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	*null*

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

Nested Subqueries

- A ***subquery*** is

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

  (select *name* **from** *instructor*)
  union : set operation between subqueries
  (select *name* **from** *student)*

  Subqueries

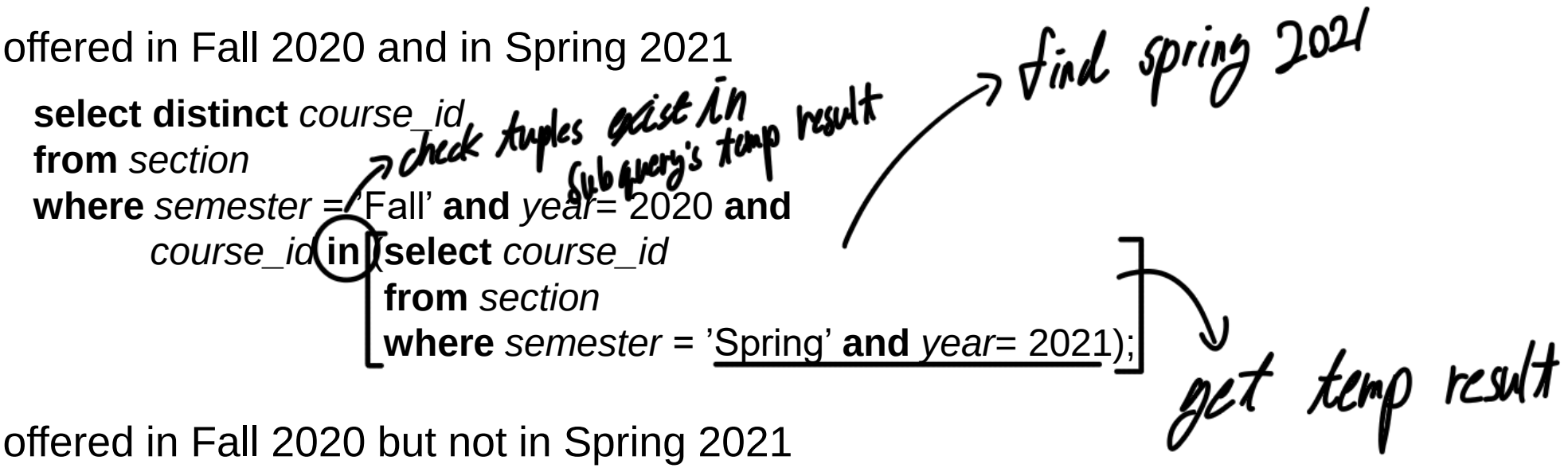
- Common use of subqueries:

  perform tests for set membership, set comparisons, and set cardinality.

Set Membership

- Find courses offered in Fall 2020 and in Spring 2021

```
select distinct course_id
from section
where semester = 'Fall' and year= 2020 and
        course_id in (select course_id
                      from section
                      where semester = 'Spring' and year= 2021);
```



- Find courses offered in Fall 2020 but not in Spring 2021

```
select distinct course_id
from section
where semester = 'Fall' and year= 2020 and
        course_id  not in (select course_id
                           from section
                           where semester = 'Spring' and year= 2021);
```

Set Membership (cont.)

- 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);
```

Compare tuple

Compare takes with teaches using partial Attrs

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

Set Comparison – "some" = Not a min

- 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 as T, instructor as S
where T.salary > S.salary and S.dept_name = 'Biology';
```

Not min value

- Same query using > **some** clause

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

=> matching with any tuple of subquery

same

Set Comparison – "some" (cont.) : at least one of

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

(5 < **some** | 0 | 5 | 6 |) = true *(5 is less than some tuple in the relation)*

(5 < **some** | 0 | 5 |) = false

(5 = **some** | 0 | 5 |) = true

(5 ≠ **some** | 0 | 5 |) = true (since 0 ≠ 5)

(= **some**) ≡ **in**

However,
(≠ **some**) ≢ **not in**

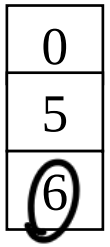
Set Comparison – "all" : match with ≠ value of subquery

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

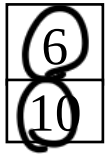
```
select name
from instructor
where salary > all (select salary
                    from instructor
                    where dept_name = 'Biology');
```

Set Comparison – "all" (cont.)

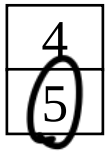
- A <comp> **all** *r* $\Leftrightarrow$ $\forall\ t \in r$ (A <comp> *t)*

(5 < **all** 

) = false

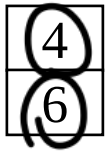
(5 < **all** 

) = true

(5 = **all** 

) = false

(= **all**) $\neq$ **in**

(5 ≠ **all** 

) = true

(≠ **all**) $\equiv$ **not in**

(since 5 ≠ 4 and 5 ≠ 6)

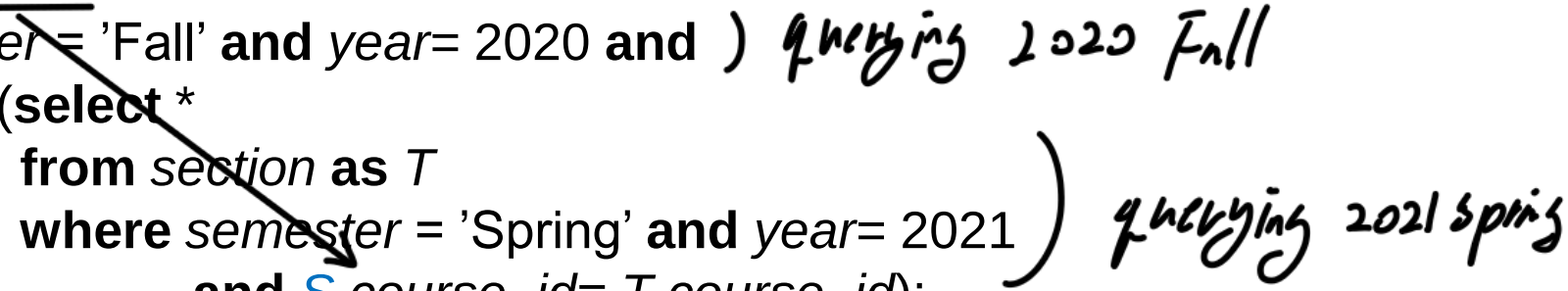
Test for Empty Relations

- 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 2020 semester and in the Spring 2021 semester”

```
select course_id
from section as S
where semester = 'Fall' and year= 2020 and
        exists (select *
                from section as T
                where semester = 'Spring' and year= 2021
                        and S.course_id= T.course_id);
```



- Correlated subquery
- *S*: 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));
```

repeat for ∀ student !

B

A

Biology dept offer

Students takes

- Note that $B - A = \emptyset \Leftrightarrow B \supseteq A$
- *Note:* Cannot write this query using "= **all**" and its variants

Subqueries in the From Clause

- SQL allows a subquery expression to be used in the **from** clause
- Find the average instructors' salaries of departments where the average is greater than $42,000.

```
select dept_name, avg_salary
from (select dept_name, avg (salary) as avg_salary
      from instructor
      group by dept_name)
where avg_salary > 42000;
```

result of select used for from's object.

- Note that we do not need to use the **having** clause

Subqueries in the From Clause (cont.)

- Another way to write above query;
  Find the average instructors' salaries of departments where the average is greater than $42,000.

```
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 > 42000;
```

rename result of subquering

Modification of the Database (Insert, delete, update)

- Deletion

  delete from <*table*> [**where** <*condition*>]

- Insertion

  insert into <*table*> **values** <*values*>

  insert into <*table*> <*select subquery*>

- Update

  update <*table*>
  set <*assignment_statements*>
  [**where** <*condition*>]

Deletion

- Delete all courses of Appl. Math department

  select

  delete from *course*
  where *dept_name* = ‘Appl. Math’

- 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’);

  where clause

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 *instructor,* 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)

Insertion

- Add a new tuple to *course*

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

need to follow origin's sequence

- or equivalently

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

not need to follow origin but match each others.

- 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 tot_creds
  from *instructor*

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

    (if *table1* did not have any primary key defined)

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

Updates

- Give a 5% salary raise to all instructors

  update *instructor*
  set *salary* = *salary* * 1.05

- Give a 5% salary raise to instructors whose salary is less than average

  update *instructor*
  set *salary* = *salary* * 1.05
  where *salary* < (**select avg** (salary)
  from *instructor*);

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

DISCUSSIONS – CHAPTER 3

Discussion 3-1

- What is an *integrity constraint*?

Discussion 3-2

- Why are integrity constraint declarations, such as *primary key* and *foreign key*, part of the **create table** statement instead of, say, the **select** or **insert** statement?

Discussion 2-5

person (*pname*, *street*, *city*)
works (*pname*, *cname*, *salary*)
company (*cname*, *city*) /* keys are underlined

- Using the above database schema, represent the following queries in *relational algebra*.
  A. *Find all names of persons.*
  B. *Find the names of persons who live in "Seoul"*
  C. *Find the names of persons who work in "SNU"*

Discussion 2-7

person (*pname*, *street*, *city*)
works (*pname*, *cname*, *salary*)
company (*cname*, *city*)

/* keys are underlined

A.
```
select person.pname
from person, works
where person.pname = works.pname and
      works.cname = 'SNU'
```

- Using the above database schema, represent the following queries in *relational algebra*.
  - A. *Find names and addresses of persons who work for "SNU".*
  - B. *Find company names located in the city where "SNU" is located.*
  - C. *Find names and addresses of persons who work for companies located in "Seoul".*

B.
```
select person.pname
from works, company
where person.pname = works.pname and
      works.cname = company.cname
```

C.
```
select person.pname
from person, works, company
where person.pname = works.pname and
      works.cname = company.cname and
      company.city = 'Seoul'
```

Discussion 2-8

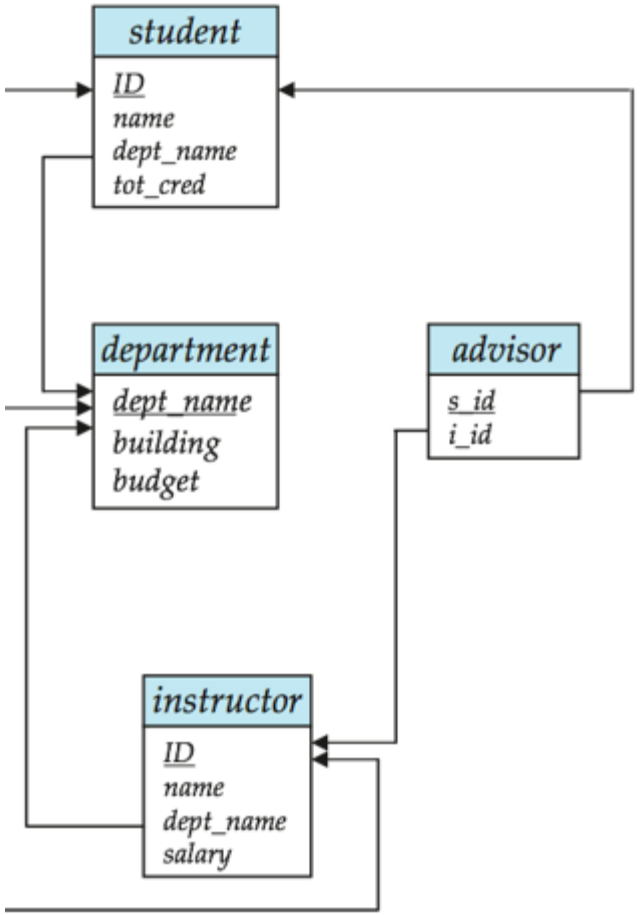
person (*pname*, *street*, *city*)
works (*pname*, *cname*, *salary*)
company (*cname*, *city*) /* keys are underlined

- Using the above database schema, represent the following queries in *relational algebra*.
  A. *Find pairs of person names who live in the same city.*
  B. *Find pairs of person names who live in the same city, without duplicate pairs.*

Discussion 3-3

Represent the following query in SQL.

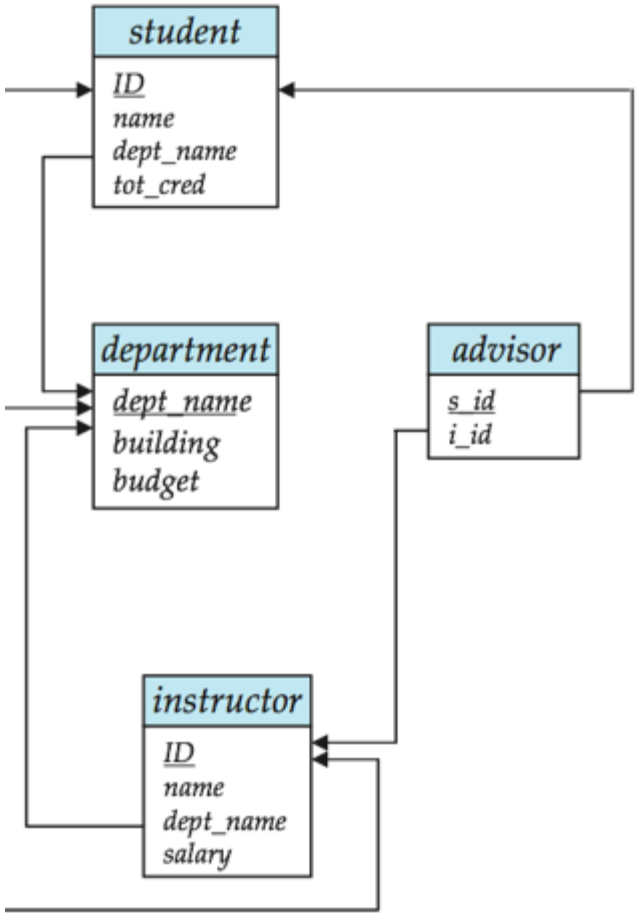
- Find *instructors* whose *name* starts with ‘E’ and ends with an ‘n’.
- Find *departments* whose *building* ends with the character ‘%’



Discussion 3-4

Represent the following query in SQL.

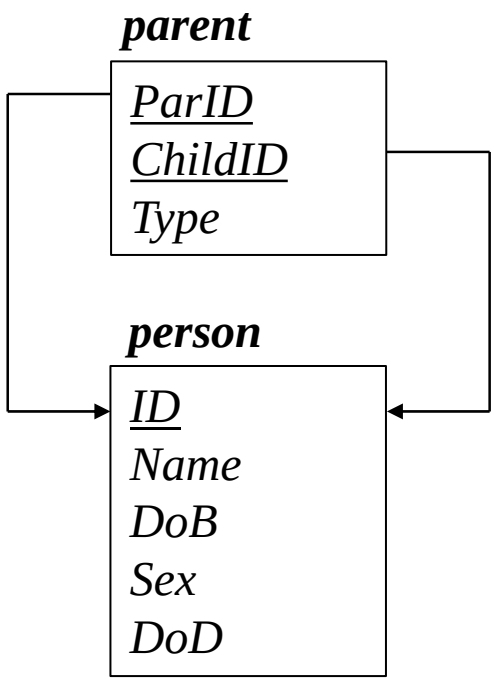
A. Find ID of students who do not have an advisor
B. Find name of students who has an advisor



Discussion 3-5

Represent the following queries in SQL.

A. Find the grand children of James Kim.
B. Find the descendants of James Kim.

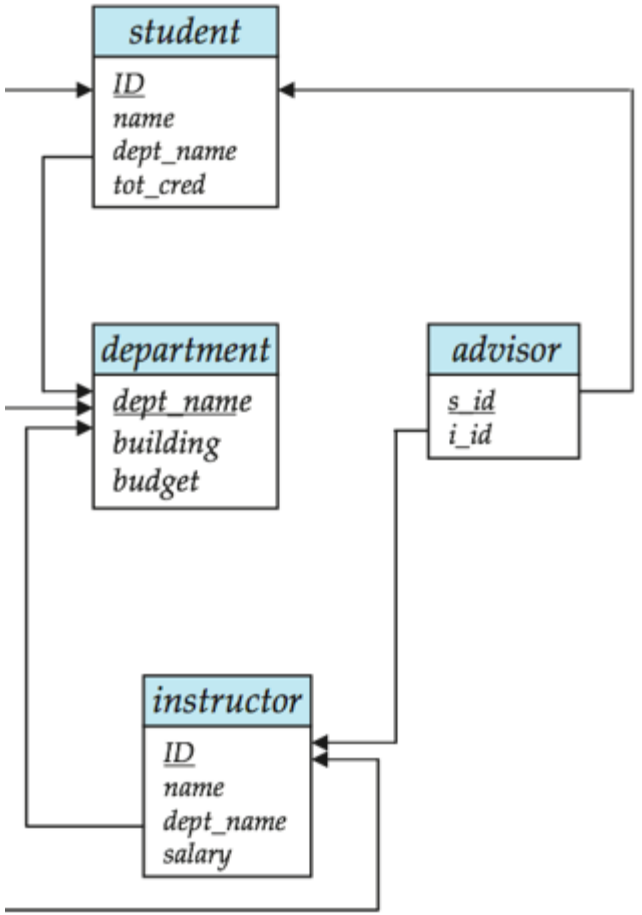


Discussion 3-6

Represent the following queries in SQL.

A. Minimum, maximum, and average budget of departments.

B. Number of advisees of Prof. ‘Kim’.



Discussion 3-7

Represent the following queries in SQL using *nested subqueries*.

A. Title of courses that were offered in room number ‘301’

B. Title of courses that were not offered in ‘spring’ semester of 2020.

Discussion 3-8

Represent the following queries in SQL using *nested subqueries*.

A. Title of course(s) with the largest room number.

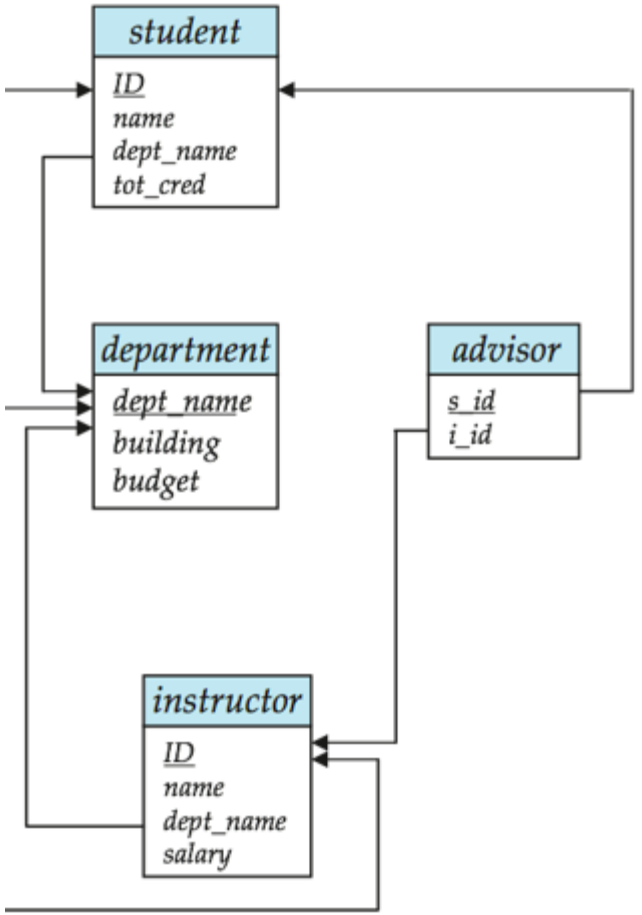
B. Title of course whose credit is larger than the average of credits of all courses in its department.

Discussion 3-9

Represent the following query in SQL.

A. Raise by 20% the salary of the instructors in the ‘CS’ department.

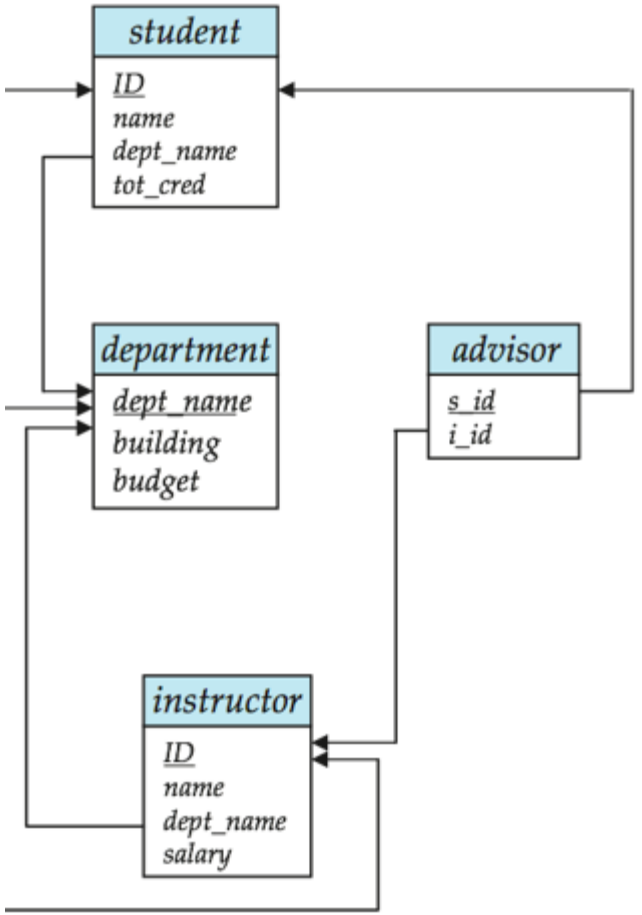
B. Raise by 20% the salary of the instructors whose department is in building ‘301’.



Discussion 3-10

Represent the following query in SQL.

- For each student without an advisor, assign instructor ‘10101’ as their advisor.



END OF CHAPTER 3