Basic SQL Queries (UML diagrams?) ch 3.3-3.4

By Leo Gordon

NOTE

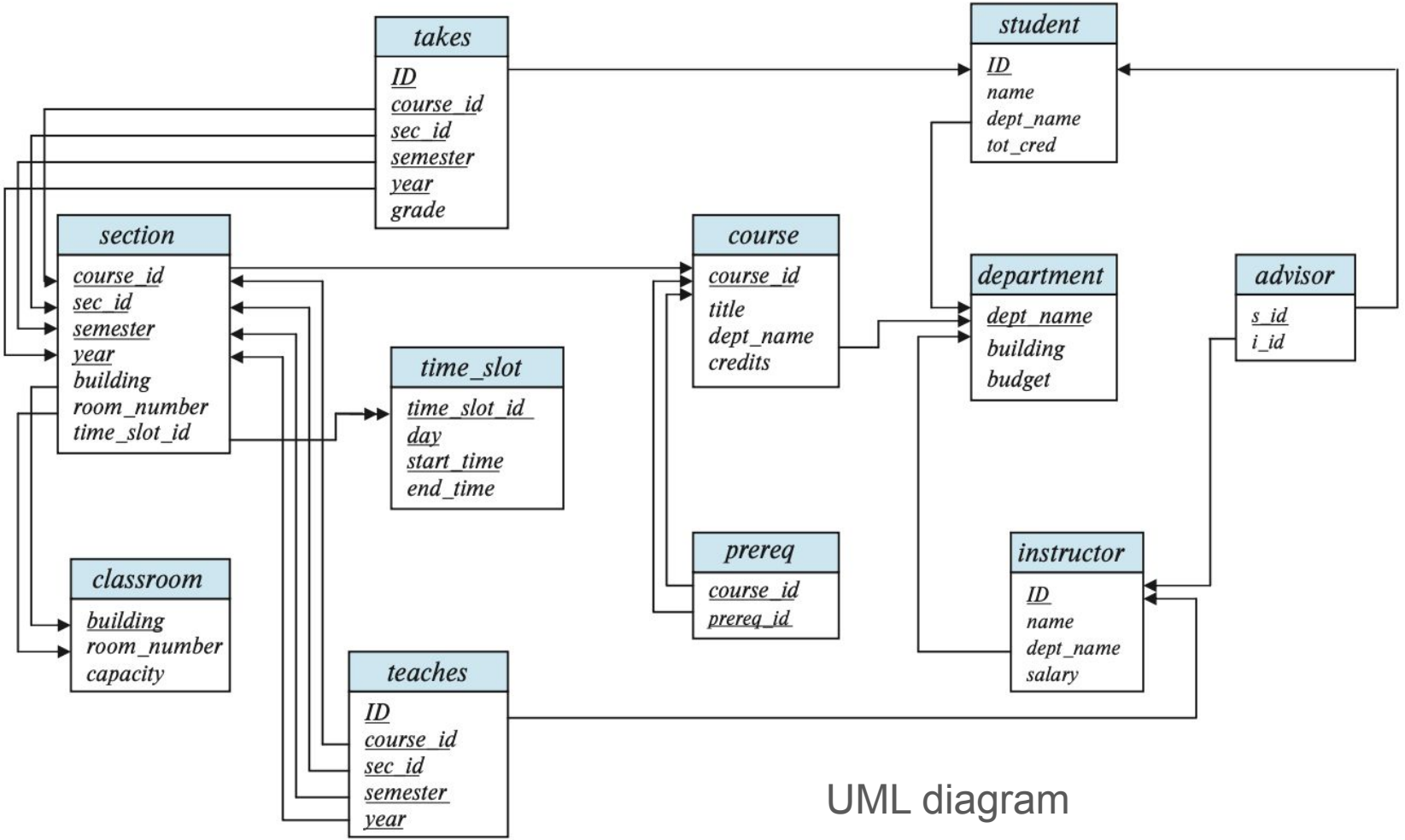
A very good chunk of the examples and text are ripped verbatim from the textbook. All of the images are provided in chapters 3.2-3.4 of the textbook.

Vocabulary

Relation: a table

Tuple: a row in a table

Attribute: a column in a table


takes
ID
course_id
sec_id
semester
year
grade
student
ID
name
dept_name
tot_cred
section
course_id
sec_id
semester
year
building
room_number
time_slot_id
course
course_id
title
dept_name
credits
department
dept_name
building
budget
advisor
s_id
i_id
time_slot
time_slot_id
day
start_time
end_time
prereq
course_id
prereq_id
instructor
ID
name
dept_name
salary
classroom
building
room_number
capacity
teaches
ID
course_id
sec_id
semester
year

UML diagram

A brief note on “DROP TABLE;”

DONT!

DONT!

DONT!

alter table *r* add *A D*;

delete from *r*;

Basic SQL queries (ch 3.3)

"The basic structure of an SQL query consists of three clauses: select, from, and where."

A simple query (ch. 3.3.1)

Let us consider a simple query using our university example, “Find the names of all instructors.”

```
select name
from instructor;
```

name
Srinivasan
Wu
Mozart
Einstein
El Said
Gold
Katz
Califieri
Singh
Crick
Brandt
Kim

Figure 3.2

Instructor names are found in the instructor relation, so we put that relation in the **from** clause. The instructor’s name appears in the name attribute, so we put that in the **select** clause.

Queries and duplicates

Consider the following query:

select *dept_name*
from *instructor*;

dept_name
Comp. Sci.
Finance
Music
Physics
History
Physics
Comp. Sci.
History
Finance
Biology
Comp. Sci.
Elec. Eng.

Duplicates!!!

Figure 3.3 Result of "**select** *dept_name* **from** *instructor*".

select distinct *dept_name*
from *instructor*;

Removes duplicates

select all *dept_name*
from *instructor*;

Explicitly keep duplicates

Arithmetic operations

“The select clause may also contain arithmetic expressions involving the operators +, −, ∗, and / operating on constants or attributes of tuples.”

```
select ID, name, dept_name, salary * 1.1
from instructor;
```

“This shows what would result if we gave a 10% raise to each instructor; note, however, that it does not result in any change to the instructor relation”

Where

Consider an example where we want to find all the computer science professors who have a salary greater than 70k:

```
select name
from instructor
where dept_name = 'Comp. Sci.' and salary > 70000;
```

Where lets us filter data in the from clause when a specific condition is met.

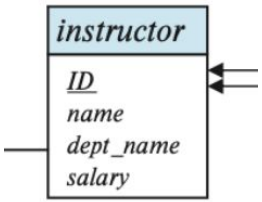
SQL allows the use of the logical connectives **and**, **or**, and **not** in the where clause.

SQL also allows the use of the comparison operators: **<, <=**, **>**, **>=**, **=**, and **<>**.

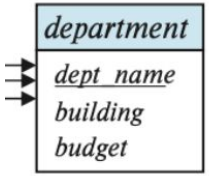
(ch 3.3.2) Query on multiple relations

‘Suppose we want to answer the query “Retrieve the names of all instructors, along with their department names and department building name.”’

In our instructor relation, we have an attribute *dept_name* but no *building*



name	*dept_name*	*building*
Srinivasan	Comp. Sci.	Taylor
Wu	Finance	Painter
Mozart	Music	Packard
Einstein	Physics	Watson
El Said	History	Painter
Gold	Physics	Watson
Katz	Comp. Sci.	Taylor
Califieri	History	Painter
Singh	Finance	Painter
Crick	Biology	Watson
Brandt	Comp. Sci.	Taylor
Kim	Elec. Eng.	Taylor



However, we do have this attribute in our department relation

select *name*, *instructor.dept_name*, *building*
from *instructor*, *department*
where *instructor.dept_name*= *department.dept_name*;

Where instructor.dept_name = department.dept_name is a join condition, will talk about why

Cartesian products using from

"The from clause by itself defines a Cartesian product of the relations listed in the clause."

Result of:
SELECT instructor.ID, … ,teaches.year
FROM instructor, teaches;

instructor.ID	*name*	*dept_name*	*salary*	*teaches.ID*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2017
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2017
10101	Srinivasan	Comp. Sci.	65000	12121	FIN-201	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	15151	MU-199	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	22222	PHY-101	1	Fall	2017
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
12121	Wu	Finance	90000	10101	CS-101	1	Fall	2017
12121	Wu	Finance	90000	10101	CS-315	1	Spring	2018
12121	Wu	Finance	90000	10101	CS-347	1	Fall	2017
12121	Wu	Finance	90000	12121	FIN-201	1	Spring	2018
12121	Wu	Finance	90000	15151	MU-199	1	Spring	2018
12121	Wu	Finance	90000	22222	PHY-101	1	Fall	2017
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
15151	Mozart	Music	40000	10101	CS-101	1	Fall	2017
15151	Mozart	Music	40000	10101	CS-315	1	Spring	2018
15151	Mozart	Music	40000	10101	CS-347	1	Fall	2017
15151	Mozart	Music	40000	12121	FIN-201	1	Spring	2018
15151	Mozart	Music	40000	15151	MU-199	1	Spring	2018
15151	Mozart	Music	40000	22222	PHY-101	1	Fall	2017
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
22222	Einstein	Physics	95000	10101	CS-101	1	Fall	2017
22222	Einstein	Physics	95000	10101	CS-315	1	Spring	2018
22222	Einstein	Physics	95000	10101	CS-347	1	Fall	2017
22222	Einstein	Physics	95000	12121	FIN-201	1	Spring	2018
22222	Einstein	Physics	95000	15151	MU-199	1	Spring	2018
22222	Einstein	Physics	95000	22222	PHY-101	1	Fall	2017
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

Figure 3.6 The Cartesian product of the *instructor* relation with the *teaches* relation.

Using where to filter the cartesian product

Recall:

```
select name, instructor.dept_name, building
from instructor, department
where instructor.dept_name= department.dept_name;
```

What this is really saying is make the Cartesian product and only keep the examples where the dept_names are equal!!

Additional basic operators (Ch 3.4.1) –rename

Question: why would we want to rename the relations in a query?

Textbook quote

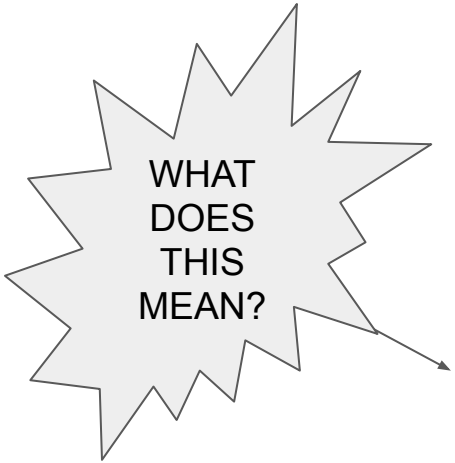
“First, two relations in the from clause may have attributes with the same name, in which case an attribute name is duplicated in the result. Second, if we use an arithmetic expression in the select clause, the resultant attribute does not have a name. Third, even if an attribute name can be derived from the base relations as in the preceding example, we may want to change the attribute name in the result.”

rename applications:

We can use rename to clarify our attributes

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

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



We can use rename to shorten names

```
select T.name, S.course_id
from instructor as T, teaches as S
where T.ID= S.ID;
```

We can use rename to compare tuples of a relation

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

Strings in SQL (ch 3.4.2)

- SQL specifies strings by enclosing them in single quotes
- A single quote character that is part of a string can be specified by using two single quote characters

“It’s right” ⟶ ‘It’‘s right’

In the SQL standard, string comparisons are case sensitive, but aren’t in some implementations

---

SQL Standard

‘Hello’ ≠ ‘hello’

MySQL, SQL Server

‘Hello’ = ‘hello’

More Strings

- upper(s) to make a string uppercase
- lower(s) to make it lower
- “||” to concatenate
- trim(s) to remove spaces at the end of a string

~~“See your database system’s manual for more details on exactly what string functions it supports”~~

These are all supported!

Documentation

https://www.postgresql.org/docs/current/functions-string.html

Pattern-Matching (3.4.2)

Let's say we want to select all the classes that start with 'Intro' (Intro classes)

We can use the following command and the like keyword:

SELECT title FROM course WHERE title like 'Intro%';

We can use % to match substrings

Intro% returns all classes that start with intro

%Intro% returns all classes with Intro in the name

We can use _ to match any character

___ returns all 3 character strings

___% returns all strings of 3 or more characters

Escape Sequences

We can use _ to match any character

___ returns all 3 character strings

___ returns all strings of 3 or more characters

Let's say we store full names in a format of *Firstname_Lastname* in some attribute *Fullname*, and we want to find all the 'Leo_G's at this college

SELECT fullname FROM r WHERE fullname like 'Leo_G'; is invalid!

We can use an escape character to solve the issue

SELECT fullname FROM r WHERE fullname like 'Leo_G';

3.4.3 & 3.4.4

We can use the asterisk (*) to select all the attributes of a relation

We can use order by to sort their tuples alphabetically

```
select name
from instructor
where dept_name = 'Physics'
order by name;
```

And we can use desc and asc to specify direction

```
select *
from instructor
order by salary desc, name asc;
```


“Suppose that we wish to list the entire instructor relation in descending order of salary. If several instructors have the same salary, we order them in ascending order by name.”

between keyword (3.4.5)

```
select name
from instructor
where salary <= 100000 and salary >= 90000;
```

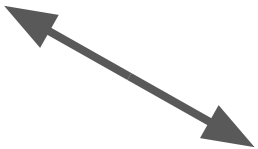
```
select name
from instructor
where salary between 90000 and 100000;
```

We can also use not between as opposed to between

Combining equals statements

Instead of writing individual equals statements and chaining them together, we can construct two tuples and equate them

```
select name, course_id
from instructor, teaches
where instructor.ID= teaches.ID and dept_name = 'Biology';
```



```
select name, course_id
from instructor, teaches
where (instructor.ID, dept_name) = (teaches.ID, 'Biology');
```

Relational Algebra and SQL

- select: σ
- project: ∏
- ~~union: ∪~~
- ~~set difference: –~~
- Cartesian product: x
- ~~rename: ρ~~

- SQL is defined around a relational algebra called the multiset relational algebra, which can contain duplicates
- SELECT is not equal to the relational algebra select, it is more implicit

$$\Pi_{A_1, A_2, \ldots, A_n}(\sigma_P(r_1 \times r_2 \times \cdots \times r_m))$$

WHERE

FROM

Handout

Handout answers

1. SELECT, WHERE, FROM

2. By listing multiple relations in the ‘from’ clause

3.

```
select name, instructor.dept_name, building
from instructor, department
where instructor.dept_name= department.dept_name;
```

4.

```
SELECT name, department.dept_name, building
FROM student, department
WHERE student.dept_name =department.dept_name
AND student.name = ‘John’
ORDER BY department.dept_name asc;
```