CSC7072: Databases, fall 2015

Dr. Kim Bauters

nested queries in SQL, derived relations, and views

nested queries, derived relations, and views

how to retrieve and manipulate data from a DB

we will be looking at:

- basic queries (*e.g.* selecting data, linking tables, sorting)
- set operations (*e.g.* joins, union, difference, intersection)
- aggregate functions (*e.g.* average, minimum, sum)
- null values (*e.g.* handling missing information)
- complex queries (*i.e.* putting it all together)
- modifying data
- nested subqueries (*i.e.* a query as part of a query)
- joins and views

nested queries, derived relations, and views

retrieving data using SQL

don't forget:

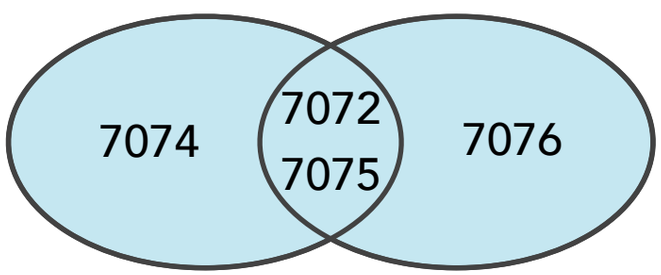
```
SELECT {attribute [AS new_attribute_name]}
FROM {table [AS new_table_name]}
[{JOIN table ON attribute = attribute}]
[WHERE {condition}]
[GROUP BY {attribute}]
    [HAVING {condition}]
[ORDER BY {attribute}]
```

where {argument} denotes you need to have at least one, and
where [argument] denotes a part that is optional and can be omitted

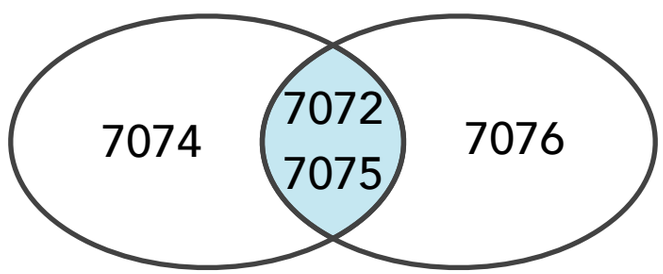
nested queries, derived relations, and views

retrieving data using SQL

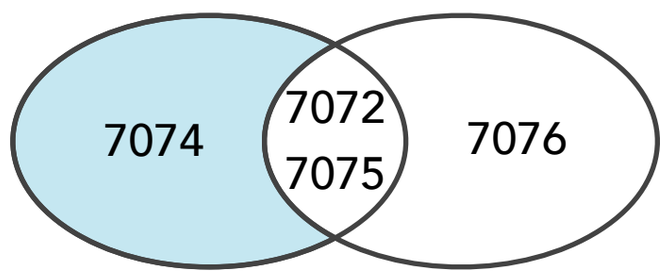
remember these set operations?



courses thought in either 2014 or 2015



courses thought in both 2014 and 2015



courses only thought in 2014, not in 2015

all courses thought in 2014:

```
(SELECT course_id FROM section WHERE year = 2014)
```

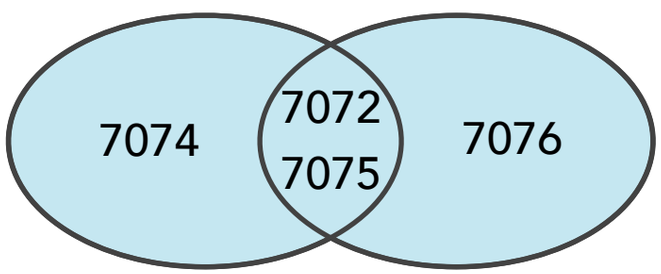
all courses thought in 2015:

```
(SELECT course_id FROM section WHERE year = 2015)
```

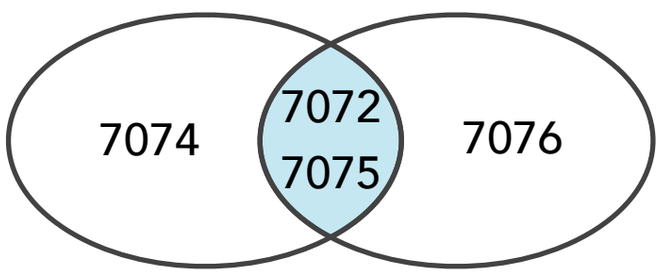
nested queries, derived relations, and views

retrieving data using SQL

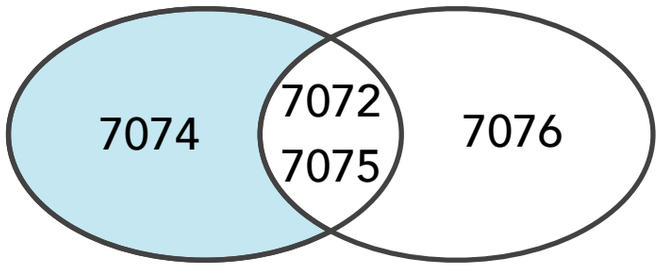
remember these set operations?



(SELECT course_id FROM section WHERE year = 2014)
UNION
(SELECT course_id FROM section WHERE year = 2015)



(SELECT course_id FROM section WHERE year = 2014)
INTERSECT
(SELECT course_id FROM section WHERE year = 2015)



(SELECT course_id FROM section WHERE year = 2014)
EXCEPT
(SELECT course_id FROM section WHERE year = 2015)

nested queries, derived relations, and views

retrieving data using SQL

remember these set operations?

UNION, INTERSECT, EXCEPT will remove all duplicates

can be overridden by adding ALL at the end:

UNION ALL
INTERSECT ALL
EXCEPT ALL

how do you write a query to find all instructors working in both the computer science and the psychology department?

nested queries, derived relations, and views

retrieving data using SQL

many other set operations possible:

problem
find the instructors earning more than some/all other instructors in the computer science department (CS due to space constraints)

```
SELECT name
    FROM instructor
        WHERE salary > SOME(
            SELECT salary
                FROM instructor
                    WHERE dept_name = 'CS')
```

```
SELECT name
    FROM instructor
        WHERE salary > ALL(
            SELECT salary
                FROM instructor
                    WHERE dept_name = 'CS')
```

nested queries, derived relations, and views

retrieving data using SQL

how does >SOME(…) work?

5 > SOME(| 2 / 5 / 6 |) — *true*, because 5 > 2

5 > SOME(| 9 / 5 / 6 |) — *false*, because 5 ⊁ 5, 5 ⊁ 6, 5 ⊁ 9

5 = SOME(| 0 / 5 |) — *true*, because 5 = 5, equal to "5 in(…)"

5 <> SOME(| 0 / 5 |) — *true*, because 5 <> 0, **not** equal to "5 not in(…)"

nested queries, derived relations, and views

retrieving data using SQL

still many other set operations possible:

```
SELECT name
    FROM instructor
        WHERE salary IN(
SELECT salary
    FROM instructor
        WHERE dept_name = 'CS')
```

select those instructors having the same salary as someone in the computer science department

nested queries, derived relations, and views

retrieving data using SQL

still many other set operations possible:

```
SELECT name
    FROM instructor
        WHERE EXISTS(
            SELECT name
                FROM student
                    JOIN advisor ON advisor.student_id = student.id,
                                    advisor.instructor_id = instructor.id
                WHERE tot_cred < 10)
```

select those instructors that are advisors to students with less than 10 credits in total

EXISTS returns **true** if the subquery is not empty

nested queries, derived relations, and views

retrieving data using SQL

many set operations are possible, but most people will spend years working with databases before they find one "in the wild"

- often (always?) you can write a query in a different way that does not involve set operations
- operations such as **IN**(…), **EXISTS**(…) are important, as they can be non-trivial to replace
- operations such as **>SOME**(…), **<>ALL**(…) are seldom used; often easier to solve in alternative ways

you need to be able to write queries with **IN**/**EXISTS**, but only need to be able to explain what a query with >some(…), <>all(…) does

nested queries, derived relations, and views

retrieving data using SQL

all set operations so far are examples of nested queries

what is a nested query?

a *nested query*, or *subquery*, or *inner query*, is a query embedded in ***parentheses*** in the ***WHERE*** clause of ***another query***

outer query

```
SELECT name
  FROM instructor
    WHERE salary IN (
      SELECT salary
        FROM instructor
          WHERE dept_name = 'CS')
```

inner query

nested queries, derived relations, and views

retrieving data using SQL

nesting queries with SQL

possible to have a query in a query

the inner query only allows a limited syntax

but SELECT … FROM … WHERE is generally fine

typically used for set membership/comparison/cardinality

```
SELECT DISTINCT course_id
    FROM section
        WHERE year = 2014 and
            course_id IN (SELECT course_id
                              FROM section
                                  WHERE year = 2015)
```

nested queries, derived relations, and views

retrieving data using SQL

nesting queries allow us to solve some very hard problems

problem revisited

what are the names of the students who scored more than the average total credits among the students in the computer science department?

```
SELECT name, tot_cred
    FROM student
        WHERE tot_cred > (SELECT AVG(tot_cred)
                            FROM student AS inner
                                WHERE inner.dept_name = 'CS')
```

nested queries, derived relations, and views

retrieving data using SQL

nested queries have their own terminology

```
SELECT name
    FROM instructor
        WHERE EXISTS (
            SELECT name
                FROM student
                    JOIN advisor ON advisor.student_id = student.id,
                                    advisor.instructor_id = instructor.id
                WHERE tot_cred < 10)
```

correlation name/variable

the subquery uses values from the outer query

called a ***correlated*** subquery

more on subqueries in next lessons

nested queries, derived relations, and views

types of nested queries

not all nested queries are created equal!

- (simple) nested subqueries: the inner query can be fully evaluated, and passes results on to outer query
  - ↳ often heavily optimised, and thus very fast
    can be as fast, or faster, than joins

- correlated nested subqueries: inner query uses names/variables from outer query and is therefore reliant on the (results of the) outer query
  - ↳ not a lot of ways to optimise
    typically slow, *sometimes* easiest for complex problems

nested queries, derived relations, and views

types of nested queries

how are nested subqueries executed?

- (simple) nested subqueries
  - ↳ the innermost query is executed first (and is fully executed)
    the result is passed on to the outer query

- correlated nested subqueries
  - ↳ innermost query is executed *once for each row* in outer query
    needed because inner query references row of outer query

nested queries, derived relations, and views

types of nested queries

not all nested queries are created equal!

- single row subquery SELECT AVG(tot_cred) …
  returns a single row, consisting of one column, to the outer query
- multiple row subquery SELECT name FROM student …
  returns multiple rows to the outer query
- multiple column subquery
  returns multiple columns to the outer query

nested queries, derived relations, and views

single row subquery: use with arithmetic operators

single row subquery

↳ the inner query returns only one result to the outer query

```
SELECT name, tot_cred
  FROM student
    WHERE tot_cred > (SELECT AVG(tot_cred)
                        FROM student AS inner
                          WHERE inner.dept_name = 'CS')
```

can be used with the arithmetic operators =, >, <, >=, <=, < >

nested queries, derived relations, and views

multiple row subquery: use with multiple value operators

multiple row subquery

↳ the inner query returns one or more results to the outer query

```
SELECT DISTINCT course_id
    FROM section
        WHERE year = 2014 and
            course_id IN (SELECT course_id
                            FROM section
                                WHERE year = 2015)
```

can only be used with multiple value operators,
such as IN, ANY, ALL, SOME, EXISTS

nested queries, derived relations, and views

combinations with arithmetic operators

remember: can combine ALL/ANY/SOME with arithmetic operators

5 > ALL(| 2 |) — *false*, because 5 ≯ 6
| 5 |
| 6 |

5 < ALL(| 9 |) — *true*, because 5 < 6, 5 < 7, 5 < 9
| 7 |
| 6 |

5 = ALL(| 0 |) — *false*, because 5 ≠ 0
| 5 |

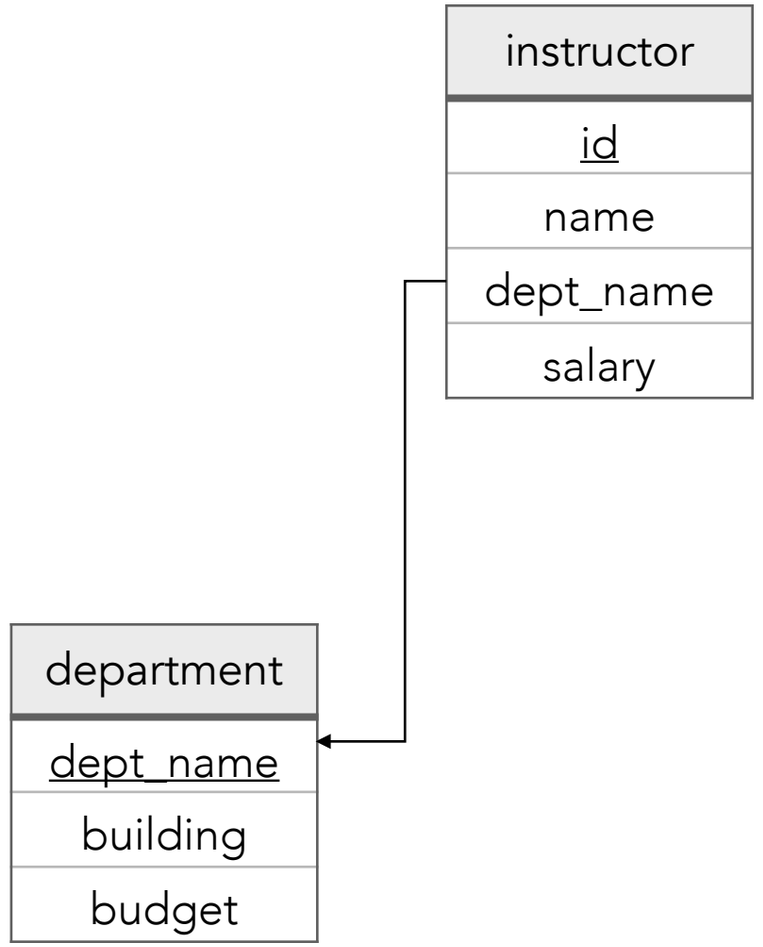
5 <> ALL(| 0 |) — *false*, because 5 <> 5
| 5 |

nested queries, derived relations, and views

your turn

try some queries yourself!

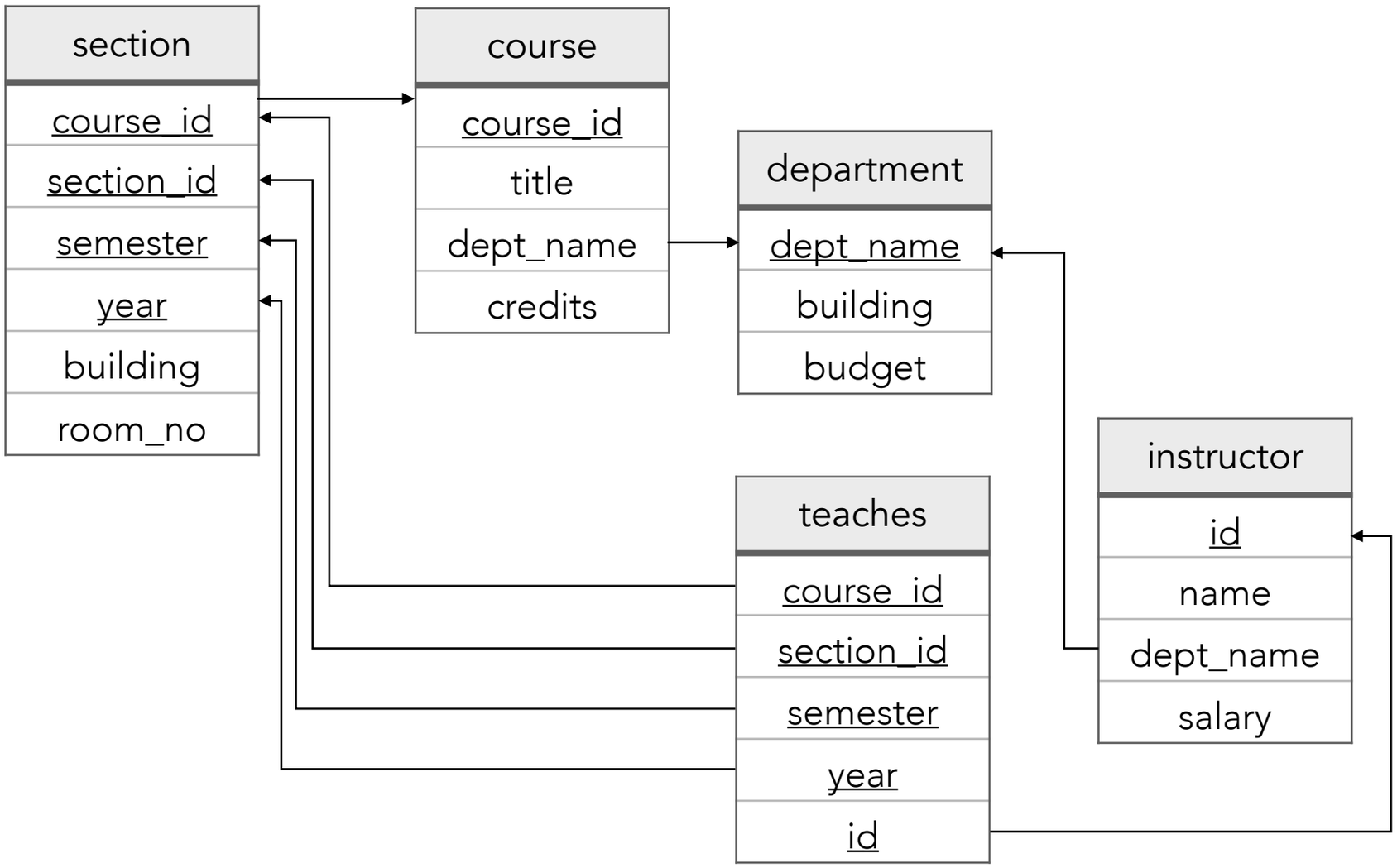
- who are the instructors earning more than the average?
- who are the instructors in CS earning more than average for the instructors in PS?
- who are the instructors in CS earning more than any/some/at least one of the instructors in PS?
- who are the instructors working in both PS and CS? (use EXISTS)



nested queries, derived relations, and views

your turn

who are the instructors in CS not teaching any course?



nested queries, derived relations, and views

multiple column subquery: the rare one

multiple column subquery

↳ the inner query returns one or more tuples to the outer query, each of which consists of more than one column

```
SELECT *
  FROM section
    WHERE
      (building, room_number) IN
      (SELECT building, room_number
        FROM classroom
          WHERE capacity > 100)
```

specify the names of the columns using the ({name, …}) IN syntax

nested queries, derived relations, and views

multiple column subquery: the rare one

how about retrieving the actual titles of these courses?

```
SELECT title
    FROM section
        JOIN course USING(course_ID)
    WHERE
        (building, room_number) IN
        (SELECT building, room_number
            FROM classroom
                WHERE capacity > 100)
```

multiple column subqueries are not universally supported;
safer (and easier) to stick to one column

nested queries, derived relations, and views

subqueries in other parts of queries

also possible to have subqueries in HAVING
recall that HAVING is used to select results from a GROUP BY

problem
which departments have an average salary higher than the average salary of the department of computer science?

```
SELECT dept_name, AVG(salary) AS average
    FROM instructor
        GROUP BY dept_name
            HAVING average > (SELECT AVG(salary)
                                FROM instructor
                                    WHERE dept_name = 'Comp. Sci.')
```

nested queries, derived relations, and views

derived relations

also possible to have subqueries in FROM

↳ called a *derived relation*

these create temporary tables,
and are especially useful when dealing with groupings!

problem

give an overview of the departments, as well as the average salary in each department

nested queries, derived relations, and views

derived relations

also possible to have subqueries in FROM

↳ called a *derived relation*

these create temporary tables,
and are especially useful when dealing with groupings!

problem

give an overview of the departments, as well as the average salary in each department

nested queries, derived relations, and views

derived relations

also possible to have subqueries in FROM

problem

give an overview of the departments, as well as the average salary in each department

```
SELECT dept_name, avg_salary
    FROM (SELECT dept_name, AVG(salary) AS avg_salary
            FROM instructor
              GROUP BY dept_name)
```

can be convenient, as we did not have to use the HAVING clause

nested queries, derived relations, and views

derived relations

also possible to have subqueries in FROM

problem

give an overview of the departments and their average salary for those departments with an average salary greater than 80000.

```
SELECT dept_name, avg_salary
    FROM (SELECT dept_name, AVG(salary) AS avg_salary
            FROM instructor
                GROUP BY dept_name)
        WHERE avg_salary > 80000
```

can be convenient, as we did not have to use the HAVING clause

nested queries, derived relations, and views

derived relations

but it isn't always needed …

problem

give an overview of the departments and their average salary for those departments with an average salary greater than 80000.

```
SELECT dept_name, avg(salary) AS avg_salary
    FROM instructor
        GROUP BY dept_name
            HAVING avg_salary > 80000
```

often more ways than one to achieve your goal in SQL!

nested queries, derived relations, and views

retrieving data using SQL

don't forget:

```
SELECT {attribute [AS new_attribute_name]}
FROM {table [AS new_table_name]}
[{JOIN table ON {attribute = attribute}}]
[WHERE {condition}]
[GROUP BY {attribute}]
    [HAVING {condition}]
[ORDER BY {attribute}]
```

where {argument} denotes you need to have at least one, and
where [argument] denotes a part that is optional and can be omitted

nested queries, derived relations, and views

why do we need views?

remember that security is a reason for using DB

how do we deal with limited access?

access to all instructors, ***but not*** to their salary

remember that efficiency is a reason for using DB

what if someone always looks at average department salaries?

what if someone always joins multiple tables?

duplication of table is a bad idea!

views are stored queries that can be used like any other table

like a variable in programming!

nested queries, derived relations, and views

creating views

a view is any relation visible to a user that is not a conceptual model

CREATE VIEW view_name AS sql_expression

name you choose

any valid SQL expression

once a view is created, *view_name* can be used as any other table

crucially, a view does ***not*** create a new table

defining a view saves the SQL expression,
and substitutes that expression as needed in other queries

nested queries, derived relations, and views

example

create a view that excludes salary:

```
CREATE VIEW faculty AS
    SELECT id, name, dept_name
    FROM instructor
```

SELECT * FROM faculty

result:

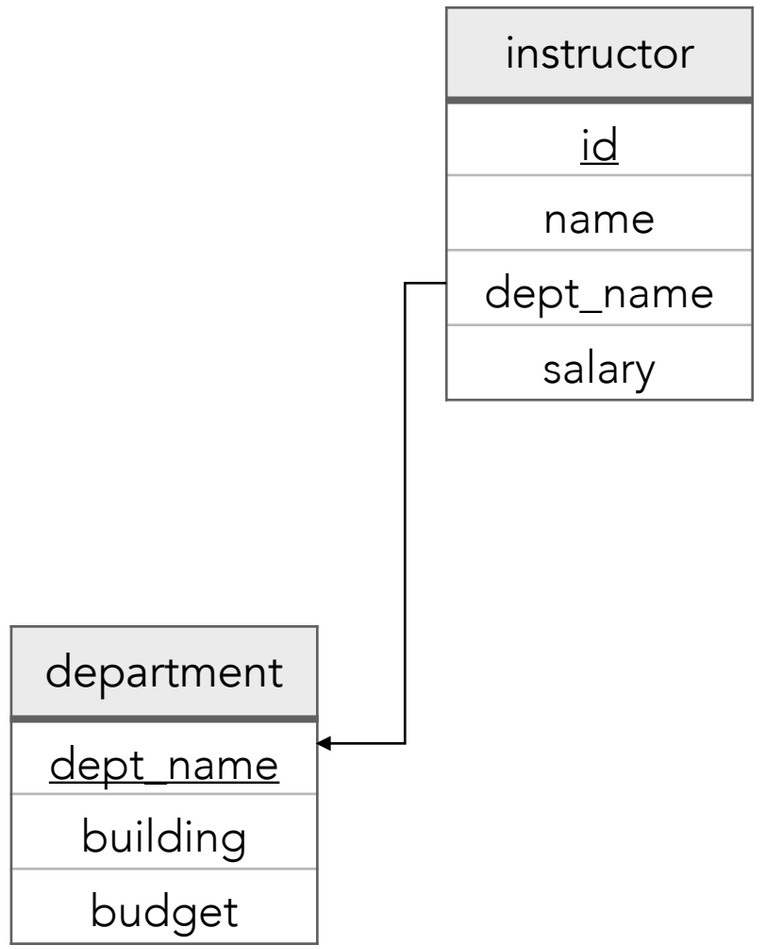
id	name	dept_name
45039	John	Comp. Sci.
30594	Selma	Comp. Sci.
30492	Paul	Elec. Eng.
20996	Tisan	Elec. Eng.

nested queries, derived relations, and views

your turn

try creating some views yourself!

- create a view of all instructors on the Biology department
- create a view of department salary totals



nested queries, derived relations, and views

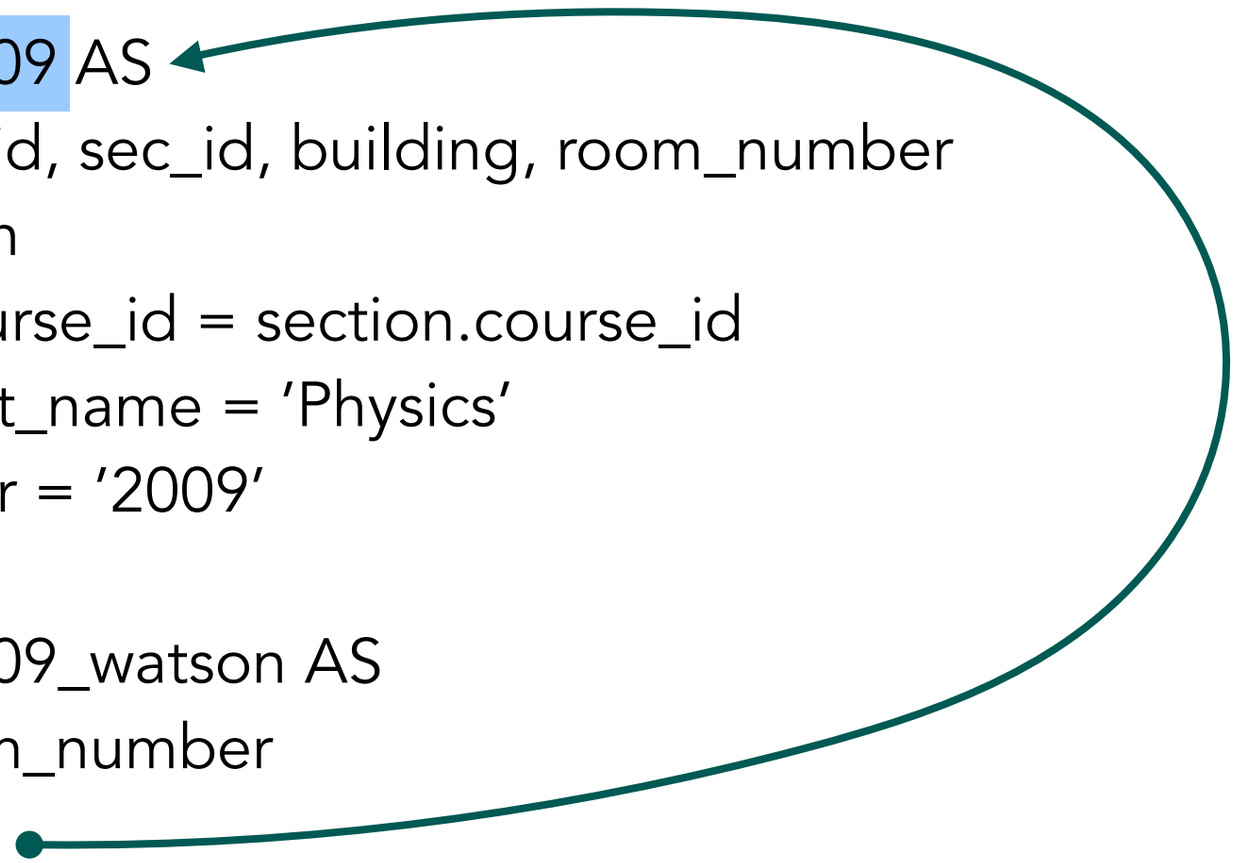
views defined on views

views defined on views

remember: once a view is created, it acts like a table

```
CREATE VIEW physics_2009 AS
    SELECT course.course_id, sec_id, building, room_number
        FROM course, section
            WHERE course.course_id = section.course_id
              AND course.dept_name = 'Physics'
              AND section.year = '2009'

CREATE VIEW physics_2009_watson AS
    SELECT course_id, room_number
        FROM physics_2009
            WHERE building= 'Watson'
```



nested queries, derived relations, and views

views defined on views: terminology

views defined on views

some terminology:

a view *v* ***depends directly*** on *w* when *w* is used in the definition of the view *v*

```
CREATE VIEW w …

CREATE VIEW v …
   SELECT …
      FROM w
```

a view v ***depends*** on w if v depends directly on w, or some view used (indirectly) by v depends directly on w

a view *v* is ***recursive*** if it depends on itself, *i.e.* view *v*

nested queries, derived relations, and views

views as *"variables"*

using *with* to define a temporary view

when creating a difficult query, we can temporary save and use a result as a view, accessible only by that query

```
WITH max_budget (value) AS
    (SELECT max(budget)
        FROM department)
    SELECT budget
        FROM department, max_budget
            WHERE department.budget = max_budget.value
```

nested queries, derived relations, and views

updating views

since views are virtual tables, we can update them

```
CREATE VIEW faculty AS
    SELECT id, name, dept_name
    FROM instructor
```

```
INSERT INTO faculty VALUES ('30765', 'Green', 'Music')
```

corresponds to:

```
INSERT INTO instructor VALUES ('30765', 'Green', 'Music', null)
```

nested queries, derived relations, and views

updating views

since views are virtual tables, we can update them ***to some extent***

```
CREATE VIEW instructor_info AS
    SELECT ID, name, building
    FROM instructor
        JOIN department USING(dept_name)

INSERT INTO instructor_info VALUES ('69987', 'White', 'Taylor')
```

some unresolved ambiguity:

- what if there are no departments located in building Taylor?
- what if there are multiple departments in Taylor?

nested queries, derived relations, and views

updating views

updating generally only allowed on *updatable* views:

1. The FROM clause has only one database relation.
2. The SELECT clause contains only attribute names of the relation, and does not have any expressions, aggregates, or DISTINCT specification.
3. Any attribute not listed in the SELECT clause can be set to *null*.
4. The query does not have a GROUP BY or HAVING clause.