CSC7072: Databases, fall 2015

Dr. Kim Bauters

essential SQL: data retrieval

essential SQL

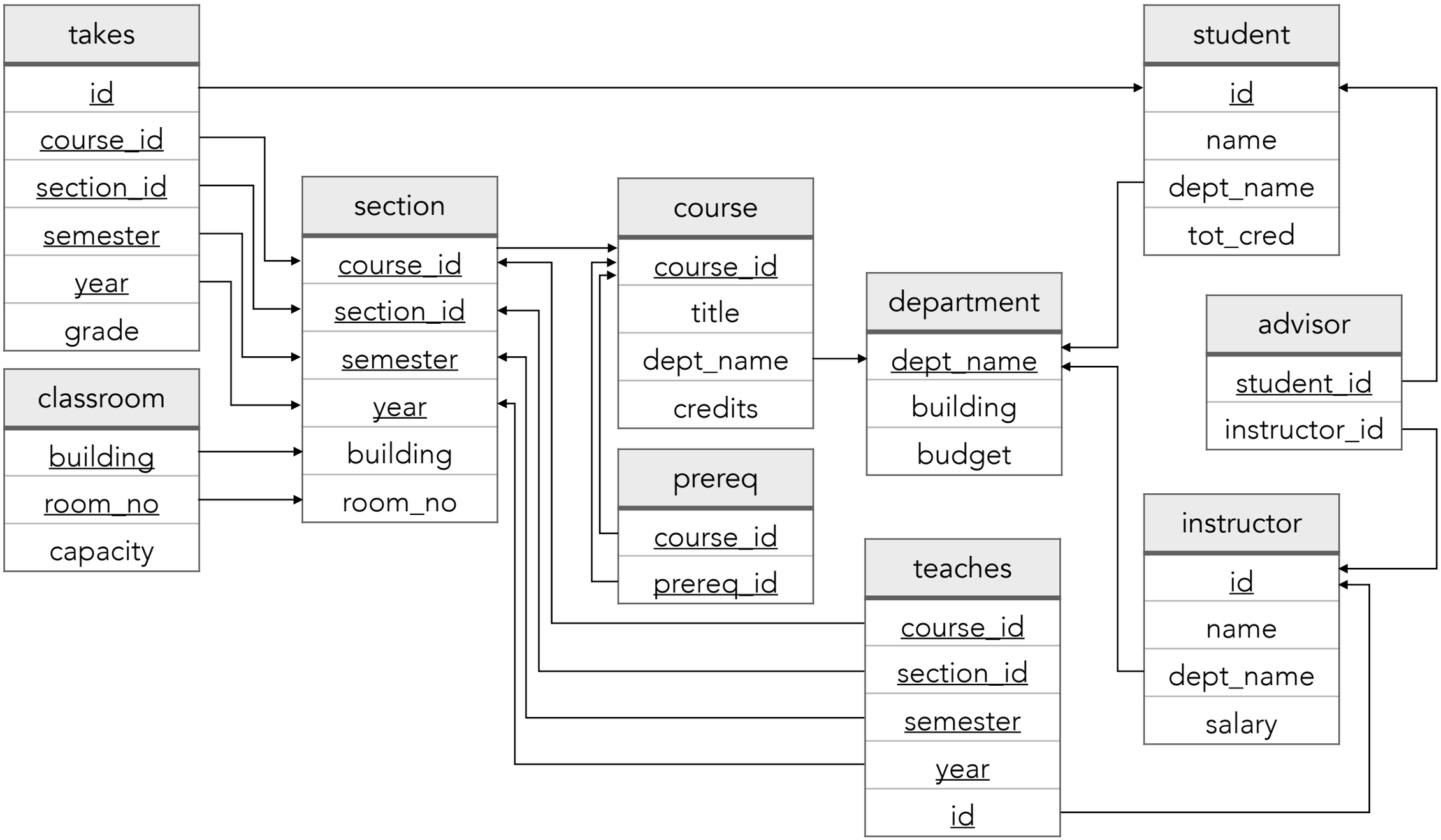
how to retrieve and manipulate data from a DB

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)
- types of joins and views

essential SQL

reminder: schema diagram of our running example



essential SQL

retrieving data using SQL

the SQL language in a nutshell:

```
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

essential SQL

retrieving all data using SQL

selecting everything from a table/relation

```
SELECT *
    FROM student
```

result:

id	name	dept_name	tot_cred
12345	Shankar	Comp. Sci.	32
54321	Williams	Comp. Sci.	32
76653	Aoi	Elec. Eng.	60
98765	Bourikas	Elec. Eng.	98

essential SQL

retrieving all data using SQL

selecting everything from a table/relation

SQL is case-*in*sensitive
but common to type keywords
using capital letters

```
SELECT *
  FROM student
```

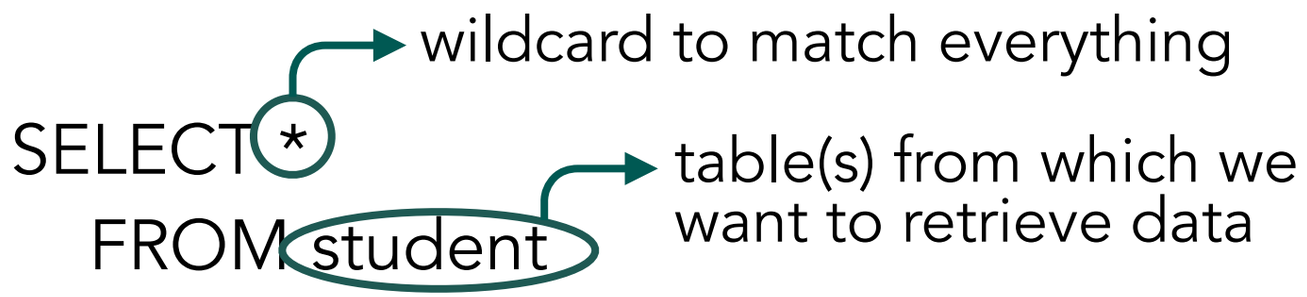
result:

id	name	dept_name	tot_cred
12345	Shankar	Comp. Sci.	32
54321	Williams	Comp. Sci.	32
76653	Aoi	Elec. Eng.	60
98765	Bourikas	Elec. Eng.	98

essential SQL

retrieving all data using SQL

selecting everything from a table/relation



result:

id	name	dept_name	tot_cred
12345	Shankar	Comp. Sci.	32
54321	Williams	Comp. Sci.	32
76653	Aoi	Elec. Eng.	60
98765	Bourikas	Elec. Eng.	98

the result is a new table/relation

essential SQL

retrieving some data using SQL

selecting certain attributes from a table/relation

```
SELECT dept_name
FROM student
```

result:

dept_name
Comp. Sci.
Comp. Sci.
Elec. Eng.
Elec. Eng.

essential SQL

retrieving some data using SQL

selecting certain attributes from a table/relation

result:

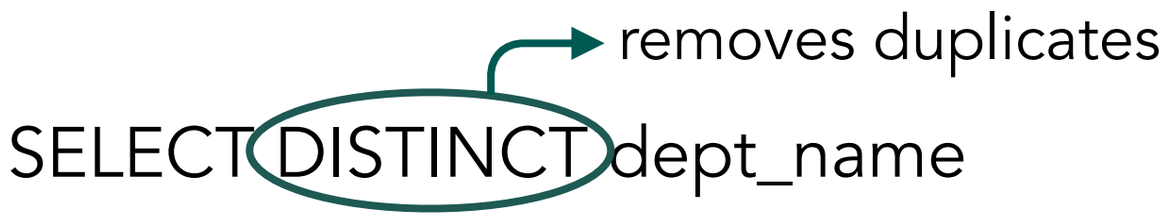
dept_name
Comp. Sci.
Comp. Sci.
Elec. Eng.
Elec. Eng.

SQL *allows duplicates* in relations and query results!

essential SQL

retrieving some data using SQL

selecting certain attributes from a table/relation



result:

dept_name
Comp. Sci.
Elec. Eng.

some implementations do omit duplicates; use ALL to get everything

essential SQL

retrieving some data using SQL

attributes in the resulting table can be renamed

```
SELECT DISTINCT dept_name AS department
    FROM student
```

result:

department
Comp. Sci.
Elec. Eng.

manipulating results using SQL

using arithmetic operators *, \, –, +

```
SELECT id, name, dept_name, tot_cred/3
    FROM student
```

result:

id	name	dept_name	tot_cred/3
12345	Shankar	Comp. Sci.	10
54321	Williams	Comp. Sci.	18
76653	Aoi	Elec. Eng.	20
98765	Bourikas	Elec. Eng.	32

divides the value of the attribute *tot_cred* by 3

essential SQL

manipulating results using SQL

possible to select the same attribute multiple times

```
SELECT tot_cred, tot_cred/3 AS yearly
    FROM student
```

result:

tot_cred	yearly
32	10
54	18
60	20
98	32

retrieves the same column twice, manipulates results of one column

essential SQL

manipulating results using SQL

possible to select the same attribute multiple times

```
SELECT tot_cred, tot_cred/3 AS yearly
    FROM student
```

result:

tot_cred	yearly
32	10
54	18
60	20
98	32

what is happening with results??
tot_cred is intuitively defined as an integer, so also result of division is an integer

retrieves the same column twice, manipulates results of one column

essential SQL

filtering results using SQL

the WHERE clause imposes conditions that the results must satisfy

```
SELECT *
    FROM student
    WHERE tot_cred = 32
```

result:

id	name	dept_name	tot_cred
12345	Shankar	Comp. Sci.	32
54321	Williams	Comp. Sci.	32

only select those tuples where tot_cred is equal to 32

essential SQL

filtering results using SQL

the WHERE clause allows logical connectives **and**, **or**, **not**

```
SELECT *
    FROM student
    WHERE tot_cred = 60 or dept_name = 'Comp. Sci.'
```

result:

id	name	dept_name	tot_cred
12345	Shankar	Comp. Sci.	32
54321	Williams	Comp. Sci.	32
76653	Aoi	Elec. Eng.	60

select students in Computer Science or with a total credit of 60

essential SQL

filtering results using SQL

the WHERE clause also allows **BETWEEN**

problem

select those instructors with a salary between 60000 and 75000

```
SELECT *
    FROM instructor
        WHERE salary BETWEEN 60000 and 75000
```

essential SQL

filtering results using SQL

back to basics: selecting from multiple tables

```
SELECT *
    FROM instructor, course
```

id	name	dept_name	course_ID	credits
3412	Sophia	CS	CSC7072	3
0657	Singh	CS	CSC7075	2
3412	Sophia	CS	CSC7076	3
0657	Singh	CS	CSC7072	3
3412	Sophia	CS	CSC7075	2
0657	Singh	CS	CSC7076	3

gives us the Cartesian product: every possible pair of instructor and course
not very useful by itself, *very* useful when combined with WHERE

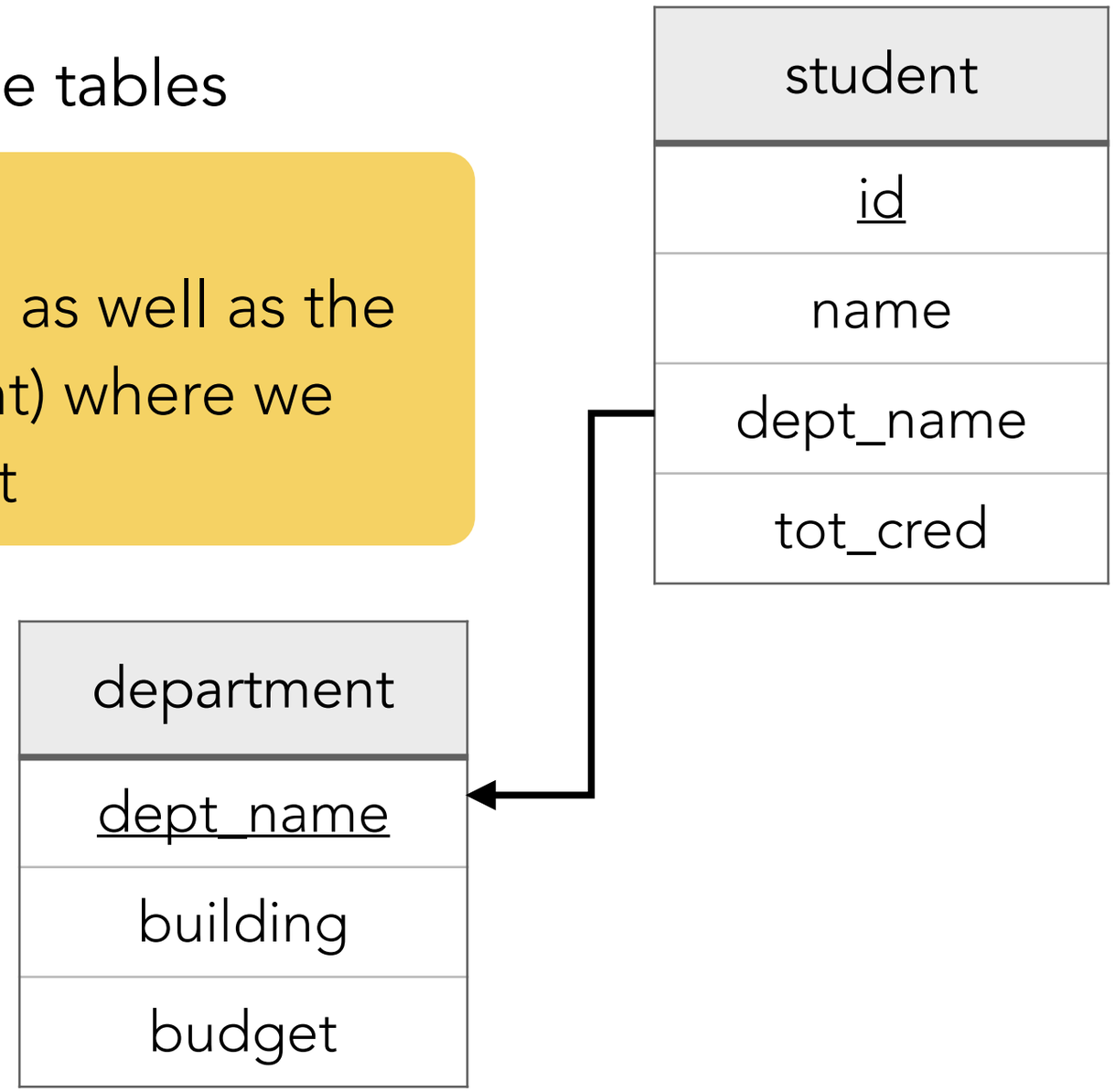
essential SQL

filtering results using SQL

joining: selecting from multiple tables

problem

get the student information, as well as the building (not the department) where we can typically find the student



essential SQL

filtering results using SQL

joining: selecting from multiple tables

```
SELECT id, name, building, tot_cred
    FROM student, department
        WHERE student.dept_name = department.dept_name
```

oldest disliked, but still in use

result:

id	name	building	tot_cred
128	Zhang	Taylor	102
12345	Shankar	Taylor	32
23121	Chavez	Painter	110
45678	Levy	Watson	46

retrieved the building name associated with the department

essential SQL

filtering results using SQL

joining alternative: selecting from multiple tables

best!
very explicit,
which is good!

```
SELECT id, name, building, tot_cred
    FROM student
        JOIN department
            ON department.dept_name = student.dept_name
```

result:

id	name	building	tot_cred
128	Zhang	Taylor	102
12345	Shankar	Taylor	32
23121	Chavez	Painter	110
45678	Levy	Watson	46

makes explicit that we are joining two tables on dept_name

essential SQL

filtering results using SQL

joining alternative: selecting from multiple tables

best! very explicit, which is good!

```
SELECT id, name, building, tot_cred
    FROM student
        JOIN department
            USING(dept_name)
```

→ easier notation when same name

result:

id	name	building	tot_cred
128	Zhang	Taylor	102
12345	Shankar	Taylor	32
23121	Chavez	Painter	110
45678	Levy	Watson	46

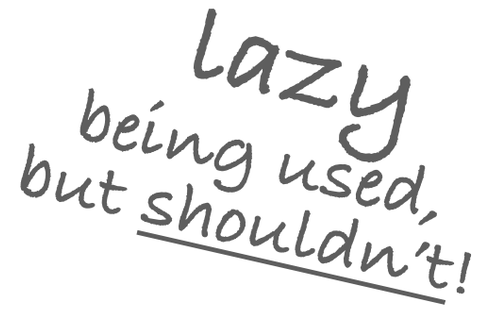
makes explicit that we are joining two tables on dept_name

essential SQL

filtering results using SQL

joining alternative: selecting from multiple tables

```
SELECT id, name, building, tot_cred
    FROM student
        NATURAL JOIN department
```



result:

id	name	building	tot_cred
128	Zhang	Taylor	102
12345	Shankar	Taylor	32
23121	Chavez	Painter	110
45678	Levy	Watson	46

natural join links both tables based on attributed in common: dept_name

essential SQL

danger of natural join

beware of natural joins

they can be very convenient, but:

- some attributes in two tables might have the same name, while they are unrelated; natural join does not know this
- sometimes *too* easy for your own good: guesswork!

example: list name of instructor along with title of the course thought

```
SELECT name, title
    FROM instructor
        NATURAL JOIN teaches NATURAL JOIN course
```

does not work as it links course.dept_name with instructor.dept_name

essential SQL

danger of natural join

beware of natural joins

they can be very convenient, but:

- some attributes in two tables might have the same name, while they are unrelated; natural join does not know this
- sometimes *too* easy for your own good: guesswork!

example: list name of instructor along with title of the course thought

```
SELECT name, title
    FROM instructor
        JOIN teaches ON instructor.id = teaches.id
            JOIN course ON teaches.course_id = course.course_id
```

essential SQL

danger of natural join

beware of natural joins

they can be very convenient, but:

- some attributes in two tables might have the same name, while they are unrelated; natural join does not know this
- sometimes *too* easy for your own good: guesswork!

example: list name of instructor along with title of the course thought

```
SELECT name, title
    FROM instructor
        JOIN teaches USING(id)
            JOIN course USING(course_id)
```

essential SQL

danger of natural join

we can also rename tables as needed:

```
SELECT name, title
    FROM instructor AS i
        JOIN teaches AS t ON i.id = t.id
            JOIN course AS c ON t.course_id = c.course_id
```

can be handy to reduce typing, but can affect readability

essential SQL

retrieving data using SQL

the SQL language recap:

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

many, *many* queries need nothing more than this simple syntax

essential SQL

retrieving data using SQL

the SQL language recap:

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

select the attributes you really need; use DISTINCT as needed
renaming is good if meaningful

many, *many* queries need nothing more than this simple syntax

essential SQL

retrieving data using SQL

the SQL language recap:

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

renaming is often unnecessary (some edge cases exist, see next)

many, *many* queries need nothing more than this simple syntax

essential SQL

retrieving data using SQL

the SQL language recap:

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

join explicitly using JOIN … ON … or JOIN … USING(…)

many, *many* queries need nothing more than this simple syntax

essential SQL

retrieving data using SQL

the SQL language recap:

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

use WHERE to check for conditions, *not* for joining

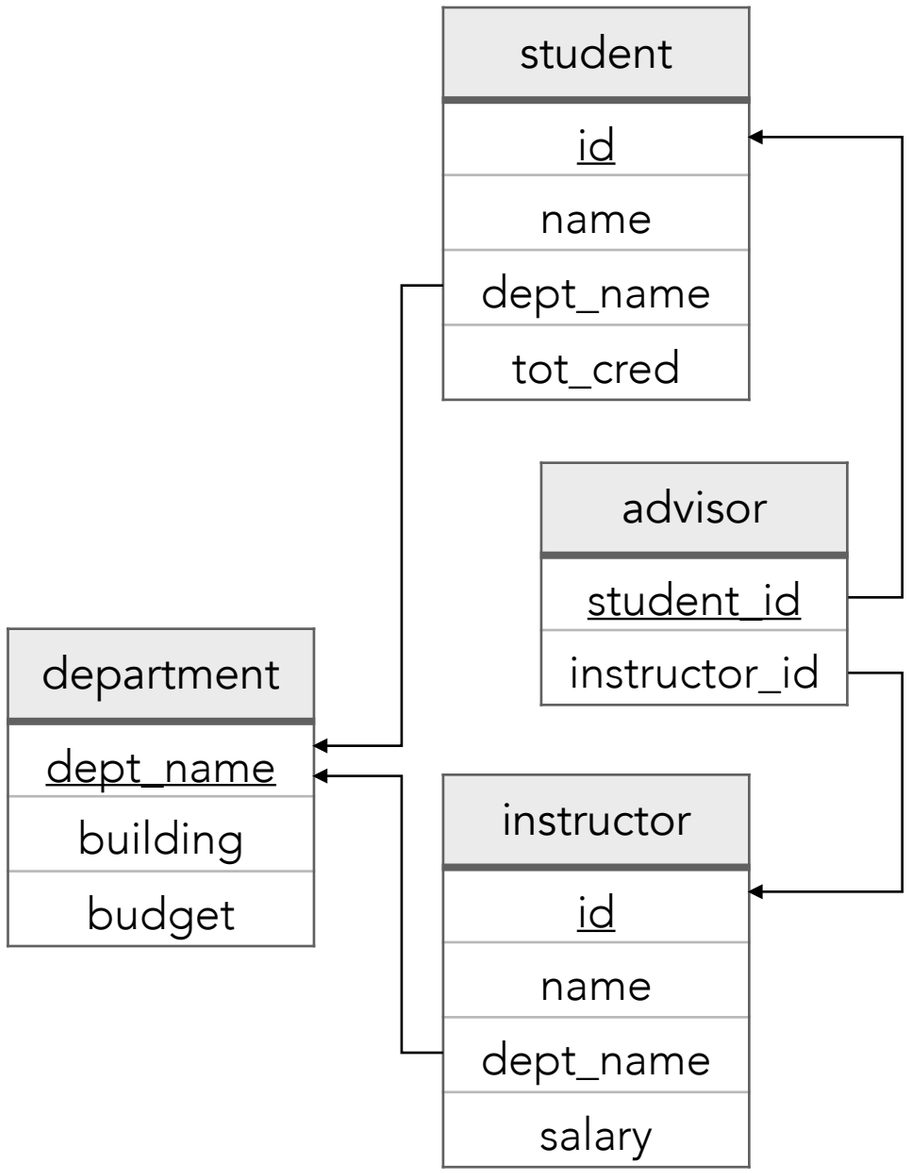
many, *many* queries need nothing more than this simple syntax

essential SQL

your turn

try some queries yourself!

- retrieve all attributes from students
- retrieve all names from students
- retrieve all distinct names from students
- retrieve the id of all students with a total credit of more than 30
- retrieve the id and name of the instructor, and the budget of the department he is working in
- retrieve the id and name of each student, along with the id and name of his/her advisor



essential SQL

renaming tables: very handy when used wisely

when renaming a table makes sense:

problem
find all courses that are a prerequisite for the course CSC7052

```
SELECT course.course_id, prereq_course.title as requires
    FROM course
        JOIN prereq USING(course_ID)
            JOIN course AS prereq_course
                ON prereq_course.course_id = prereq.prereq_id
    WHERE course.course_ID = 'CSC7052'
```

essential SQL

renaming tables: very handy when used wisely

when renaming a table makes sense:

```
SELECT course.course_id, prereq_course.title as requires
    FROM course
        JOIN prereq USING(course_ID)
            JOIN course AS prereq_course
                ON prereq_course.course_id = prereq.prereq_id
WHERE course.course_ID = 'CSC7052'
```

table *'course'* used both for course and title of its prerequirements

result:

course_id	requires
CSC7052	Java
CSC7052	logical reasoning

essential SQL

pattern matching in strings

operations on strings

a string is just a sequence of characters

if we know the exact string, use single or double quotation marks:

```
WHERE course_id = 'CSC7052'
```

if we only know a part, then use LIKE and wildcards:

```
WHERE course_id LIKE 'CSC%'
```

→ any number of characters (or 0)

```
WHERE course_id LIKE 'CSC_'
```

→ exactly one character

essential SQL

pattern matching in strings

examples for operations on strings

WHERE word LIKE '%dar'	matches *ra*dar, *ched*dar, dar
WHERE word LIKE 'dar%'	matches dar*win*, dar*e*, dar
WHERE word LIKE '%dar%'	matches *man*dar*in*, dar
WHERE word LIKE '%d_r%'	matches *o*d*o*r*ant*, dar does *not* match dr

known as pattern matching

use \% to match % itself

essential SQL

ordering results of queries

results can be ordered using ORDER BY

```
SELECT DISTINCT name
    FROM instructor
        ORDER BY name
```

by default, ordered in ascending order

specify DESC to order in descending order

```
SELECT DISTINCT name
    FROM instructor
        ORDER BY name DESC
```

essential SQL

grouping values together

aggregate functions: used to group values of multiple rows together

most common aggregates:

- avg: average value of a group
- min: minimum value in a group
- max: maximum value in a group
- sum: sum of values in a group
- count: number of values in a group

essential SQL

grouping values together

aggregate functions: used to group values of multiple rows together

problem
how many instructors are there?

```
SELECT COUNT(*) AS number_of_instructors
    FROM instructor
```

result:

number_of_instructors
12

essential SQL

grouping values together

aggregate functions: used to group values of multiple rows together

problem

what is the minimum salary of an instructor?

```
SELECT MIN(salary) AS minimum_salary
    FROM instructor
```

problem

what is the average salary of an instructor?

```
SELECT AVG(salary) AS average_salary
    FROM instructor
```

essential SQL

grouping values together

aggregate functions: used to group values of multiple rows together

problem
what is the average salary of an instructor in each department?

```
SELECT AVG(salary) AS average, dept_name
    FROM instructor
```

result:

average	dept_name
74833.33	Elec. Eng

oops!
not what we want

essential SQL

grouping values together

aggregate functions: used to group values of multiple rows together

problem

what is the average salary of an instructor in each department?

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

we need to explicitly say which groups to form

result:

average	dept_name
72000	Biology
77333.33	Comp. Sci.
80000	Elec. Eng.

essential SQL

grouping values together

aggregate functions: used to group values of multiple rows together

problem

what is the average salary of an instructor in each department?

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

result:

average	dept_name
72000	Biology
77333.33	Comp. Sci.
80000	Elec. Eng.

all attributes that aren't aggregate functions should be listed in the GROUP BY list

essential SQL

grouping values together

conditions on aggregate functions

problem

what are the departments with an average salary over 3000?

```
SELECT AVG(salary) AS average, dept_name
    FROM instructor
        GROUP BY dept_name
            HAVING average > 78000
```

conditions of HAVING are applied *after* the formation of groups

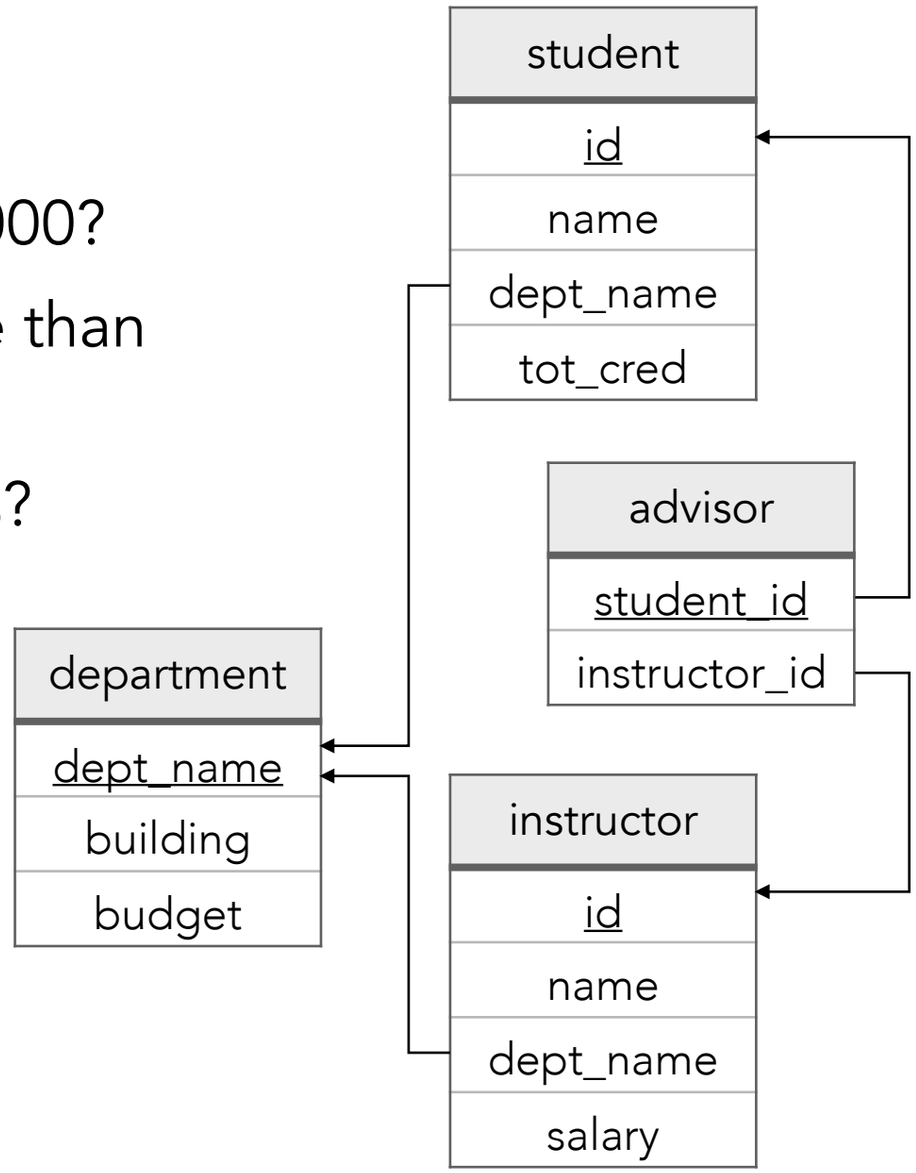
conditions of WHERE are applied *before* the formation of groups

essential SQL

your turn

try some queries yourself!

- how many instructors earn more than 75000?
- what are the departments that have more than 3 instructors? 50 instructors?
- what is the average credit of the students?
- what is the average credit of the students in the computer science department?
- what are the names of the students who scored more than the average total credits among the students in the computer science department?



essential SQL

retrieving data using SQL

we have come full circle:

```
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

essential SQL

odds and ends

some odds and ends we need to deal with:

- how can we handle *null* values?
- how do we insert data into a DB?
- how do we delete data from a DB?
- how do we update data in a DB?

instructor

id	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000
30594	Selma	Comp. Sci.	*null*
12121	Wu	Finance	90000
22222	Einstein	Physics	95000

essential SQL

dealing with missing data in SQL

null values: how to deal with the odd one out

they can mess up arithmetic/aggregate operators

special option available to find fields that are *null*:

```
SELECT *
    FROM instructor
        WHERE salary IS NULL
```

result:

id	name	dept_name	salary
30594	Selma	Comp. Sci.	*null*

essential SQL

dealing with missing data in SQL

null values: how to deal with the odd one out

signifies a missing value, *a.k.a.* unknown:

- what is 7 + *null*?
- what is (*null* or true)? (*null* or false)?
- what is (*null* and true)? (*null* and false?)
- what is (not *null*)?
- what is SELECT AVG(salary) FROM instructor when some salaries are *null*?

essential SQL

a family of joins

what is a join?

definition

a join operation take two relations and returns another relation

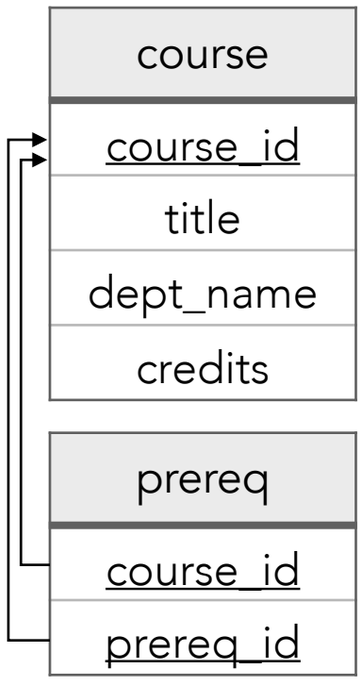
or more verbose:

definition

a join is a Cartesian product where tuples in the two relations match, alongside some condition, and a specification of the attributes that are present in the result of the join

essential SQL

running example



course

course_id	title	dept_name	credits
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

prereq

course_id	**prereq_id**
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

note: prerequirements missing for CS-315;
course information missing for CS-347

essential SQL

outer join

what is an outer join?

- extends the normal join operation to avoid loss of information
- computes the normal join; then adds tuples from one relation that do not match tuples in the other relation to the result of the join using *null* values
- can be both a *left*, *right* or *full* join

returns *all rows from left table*, with matching rows from right table and padded with nulls if necessary

returns *all rows from right table*, with matching rows from left table and padded with nulls if necessary

essential SQL

outer join examples

course

course_id	title	dept_name	credits
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

prereq

course_id	prereq_id
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

*SELECT * FROM course LEFT OUTER JOIN prereq USING(course_id)*

course_id	title	dept_name	credits	prereq_id
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	*null*

essential SQL

outer join examples

course

course_id	title	dept_name	credits
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

prereq

course_id	prereq_id
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

*SELECT * FROM course RIGHT OUTER JOIN prereq USING(course_id)*

course_id	title	dept_name	credits	prereq_id
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-347	*null*	*null*	*null*	*CS-101*

essential SQL

outer join examples

course

course_id	title	dept_name	credits
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

prereq

course_id	prereq_id
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

*SELECT * FROM course FULL OUTER JOIN prereq USING(course_id)*

course_id	title	dept_name	credits	prereq_id
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-347	*null*	*null*	*null*	*CS-101*
CS-315	Robotics	Comp. Sci.	3	*null*

essential SQL

join summary

(NATURAL) [INNER] JOIN	
(NATURAL) LEFT [OUTER] JOIN	ON {attribute = attribute}
(NATURAL) RIGHT [OUTER] JOIN	USING({attribute})
(NATURAL) FULL [OUTER] JOIN	

join type

defines how tuples that do not match in the other relation are treated

join condition

defines which tuples in both relations match up

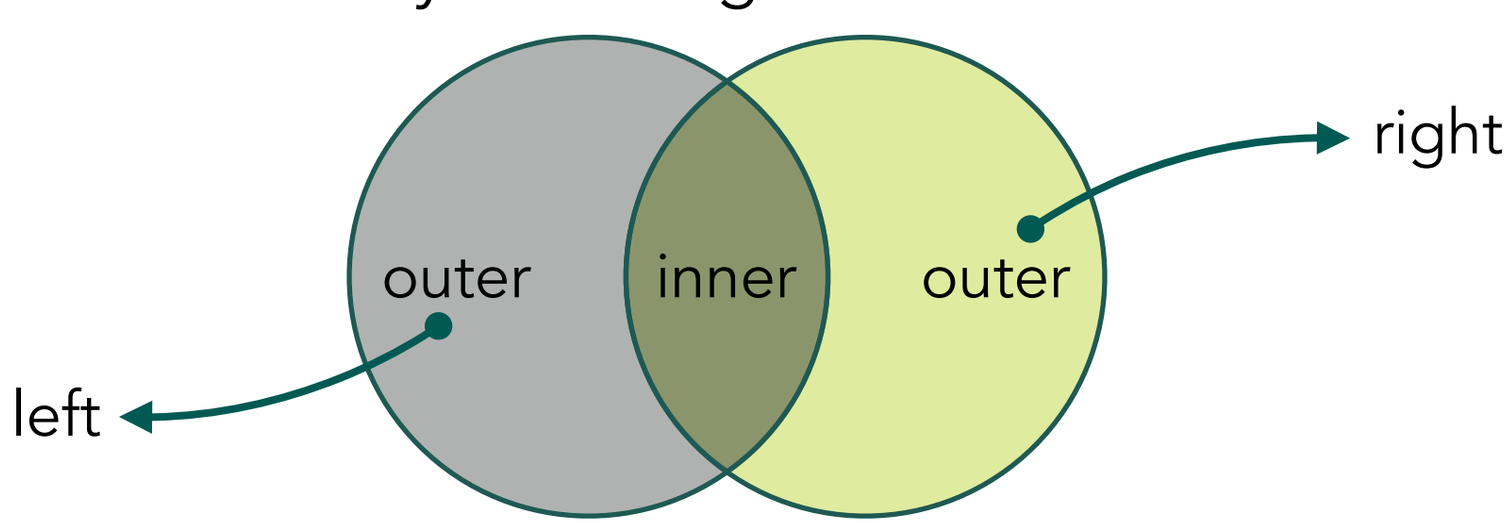
essential SQL

join summary

what is this inner join?

an inner join, or join, is the default type of joining

only considers exactly matching attributes



*SELECT * FROM course JOIN prereq USING(course_id)*

course_id	title	dept_name	credits	prereq_id
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101

essential SQL

data manipulation: adding/deleting/updating

inserting values in a database

```
INSERT INTO student
    VALUES (45603, “Thomas”, “Comp. Sci.”, 30)
INSERT INTO student (name, dept_name, tot_cred)
    VALUES (“Paul”, “Music”, 4)
```

omit/automatically generate value

deleting values in a database

```
DELETE FROM student
```

delete all tuples (!!)

```
DELETE FROM student WHERE id = 45603
```

delete previously inserted student

essential SQL

data manipulation: adding/deleting/updating

updating values in a database

```
UPDATE instructor
    SET salary = salary * 1.03
        WHERE salary > 75000
```

```
UPDATE instructor
    SET salary = CASE
        WHEN salary > 75000
        THEN salary * 1.03
        ELSE salary * 1.05
    END
```

```
UPDATE instructor
    SET salary = salary * 1.05
        WHERE salary <= 75000
```

overall: Keep It Simple and Stupid (KISS)

getting a SELECT wrong messes up your results

getting an INSERT/DELETE/UPDATE wrong messes up a DB!

essential SQL

data manipulation: adding/deleting/updating

the four pillars of the Data Manipulation Language (DML):

```
SELECT {attribute}
  FROM {table}
 WHERE {condition}
```

```
UPDATE table
   SET {attribute = value}
 WHERE {condition}
```

```
INSERT INTO table [({attribute})]
  VALUES ({value})
```

```
DELETE FROM table
 WHERE {condition}
```