AALBORG
UNIVERSITY

Database System (SW5)

4. SQL (Part 2)

Tiantian Liu
Department of Computer Science
Aalborg University
Fall 2025

Agenda

- Null Values
- Modification of the Database
- Join Expression
- Nested Subqueries
- Views, transactions and integrity constraints

Null Values

AALBORG UNIVERSITY

- **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;
```

- The predicate **is not null** succeeds if the value on which it is applied is not null.

Null Values (Cont.)

AALBORG UNIVERSITY

```
select name, salary
from instructor
where salary is not null;
```

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	null
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	null

instructor

name	*salary*
Srinivasan	65000
Wu	90000
Einstein	95000
El Said	60000

Null Values (Cont.)

AALBORG UNIVERSITY

- SQL treats as **unknown** the result of any comparison involving a null value (other than predicates **is null** and **is not null**).
  - Example: 5 < **null** or **null** <> **null** or **null** = **null**

ID	*name*	*dept_name*	*salary*	*bonus*
10101	Srinivasan	Comp. Sci.	65000	null
12121	Wu	Finance	90000	null
15151	Mozart	Music	null	null
22222	Einstein	Physics	95000	83000
32343	El Said	History	60000	60000
33456	Gold	Physics	null	null

instructor

salary > bonus
unknown
unknown
unknown
true
false
unknown

Null Values (Cont.)

AALBORG UNIVERSITY

```
select name, salary
from instructor
where salary > 70000;
```

Result of **where** clause predicate is treated as **false** if it evaluates to **unknown**

ID	*name*	*dept_name*	*salary*	*bonus*	***salary > 70000***
10101	Srinivasan	Comp. Sci.	65000	null	false
12121	Wu	Finance	90000	null	true
15151	Mozart	Music	null	null	unknown
22222	Einstein	Physics	95000	83000	true
32343	El Said	History	60000	60000	false
33456	Gold	Physics	null	null	unknown

instructor

Null Values (Cont.)

AALBORG UNIVERSITY

- The predicate in a **where** clause can involve Boolean operations (**and**, **or**, **not**); thus the definitions of the Boolean operations need to be extended to deal with the value **unknown**.
- **and**: (*true* **and** *unknown*) = *unknown*
  (*false* **and** *unknown*) = *false*
  (*unknown* **and** *unknown*) = *unknown*
- **or**: (*unknown* **or** *true*) = *true*
  (*unknown* **or** *false*) = *unknown*
  (*unknown* **or** *unknown*) = *unknown*

Null Values (Cont.)

AALBORG UNIVERSITY

```
select name, salary
from instructor
where salary > 70000 and bonus > 50000;
```

ID	*name*	*dept_name*	*salary*	*bonus*
10101	Srinivasan	Comp. Sci.	65000	null
12121	Wu	Finance	90000	null
15151	Mozart	Music	null	null
22222	Einstein	Physics	95000	83000
32343	EI Said	History	60000	60000
33456	Gold	Physics	null	null

instructor

salary > 70000	***bonus > 50000***	***result***
false	unknown	false
true	unknown	unknown
unknown	unknown	unknown
true	true	true
false	true	false
unknown	unknown	unknown

Exercise 1

AALBORG UNIVERSITY

Evaluate each row using the WHERE conditions and determine the result for that row

```
select name, salary
from instructor
where salary > 70000 or bonus > 50000;
```

ID	*name*	*dept_name*	*salary*	*bonus*
10101	Srinivasan	Comp. Sci.	null	30000
12121	Wu	Finance	90000	40000
15151	Mozart	Music	null	55000
22222	Einstein	Physics	95000	83000
32343	EI Said	History	60000	60000
33456	Gold	Physics	null	null

instructor

Exercise 1--Solution

AALBORG UNIVERSITY

Evaluate each row using the WHERE conditions and determine the result for that row

```
select name, salary
from instructor
where salary > 70000 or bonus > 50000;
```

ID	*name*	*dept_name*	*salary*	*bonus*	***salary > 70000***	***bonus > 50000***	***result***
10101	Srinivasan	Comp. Sci.	null	30000	unknow	false	unknown
12121	Wu	Finance	90000	40000	true	false	true
15151	Mozart	Music	null	55000	unknown	true	true
22222	Einstein	Physics	95000	83000	true	true	true
32343	EI Said	History	60000	60000	false	true	true
33456	Gold	Physics	null	null	unknown	unknown	unknown

instructor

Agenda

- Null Values
- Modification of the Database
- Join Expression
- Nested Subqueries
- Views, transactions and integrity constraints

Modification of the Database

AALBORG UNIVERSITY

- Deletion of tuples from a given relation.
- Insertion of new tuples into a given relation
- Updating of values in some tuples in a given relation

Deletion

AALBORG UNIVERSITY

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
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

instructor

- Delete all instructors

```
delete from instructor;
```

- Delete all instructors from the Finance department

```
delete from instructor
where dept_name = 'Finance';
```

Insertion

AALBORG UNIVERSITY

- Add a new tuple to *course*

course_id	*title*	*dept_name*	*credits*
BIO-101	Intro. to Biology	Biology	4
BIO-301	Genetics	Biology	4
BIO-399	Computational Biology	Biology	3
CS-101	Intro. to Computer Science	Comp. Sci.	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3
CS-319	Image Processing	Comp. Sci.	3
CS-347	Database System Concepts	Comp. Sci.	3
EE-181	Intro. to Digital Systems	Elec. Eng.	3
FIN-201	Investment Banking	Finance	3
HIS-351	World History	History	3
MU-199	Music Video Production	Music	3
PHY-101	Physical Principles	Physics	4

course

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

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

```
insert into course
values ('CS-437', 'Database Systems',
'Comp. Sci.', null);
```

Updates

AALBORG UNIVERSITY

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
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

instructor

- Give a 5% salary raise to all instructors

```
update instructor
set salary = salary * 1.05;
```

- Give a 5% salary raise to those instructors who earn less than 70000

```
update instructor
set salary = salary * 1.05
where salary < 70000;
```

Case Statement

AALBORG UNIVERSITY

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
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

instructor

- Increase salaries of instructors whose salary is over 100,000 by 3%, and all others by a 5%

```
update instructor
set salary = case
        when salary > 100000
        then salary * 1.03
        else salary * 1.05
       end;
```

Exercise 2

AALBORG UNIVERSITY

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
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

instructor

Write SQL statements to achieve the following

- Increase salaries of Computer Science instructors by 10%, Physics instructors by 5%, and all others by a 3%
- Revise the department name "Comp. Sci" to "Computer Science"

Exercise 2--Solution

AALBORG UNIVERSITY

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
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

instructor

- Increase salaries of Computer Science instructors by 10%, Physics instructors by 7%, and all others by a 2%

```
update instructor
set salary = case
      when dept_name = 'Comp. Sci.'
      then salary * 1.1
      when dept_name = 'Physics'
      then salary * 1.07
      else salary * 1.02
      end;
```

Exercise 2--Solution

AALBORG UNIVERSITY

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
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

instructor

- Revise the department name "Comp. Sci" to "Computer Science"

```
update instructor
set dept_name = 'Computer Science'
where dept_name = 'Comp. Sci. ';
```

Agenda

- Null Values
- Modification of the Database
- Join Expression
- Nested Subqueries
- Views, transactions and integrity constraints

Joined Relations

AALBORG UNIVERSITY

- **Join operations** take two relations and return as a result another relation.
- A join operation is a Cartesian product which requires that tuples in the two relations match (under some condition).
- The join operations are typically used as subquery expressions in the **from** clause

Natural Join in SQL

AALBORG UNIVERSITY

- Natural join matches tuples with the same values for all common attributes and retains only **one** copy of each common column.
- List the names of students along with the course ID of the courses that they take

```
select name, course_id
from  student, takes
where student.ID = takes.ID;
```

```
select name, course_id
from student natural join takes;
```

Natural Join in SQL

AALBORG UNIVERSITY

- Natural join

```
select *
from student natural join takes;
```

ID	*name*	*dept_name*	*total_cred*
1001	Anna	Comp. Sci.	100
1002	Christian	Comp. Sci.	152
1003	David	Comp. Sci.	147

student

ID	*course_id*	*year*	*grade*
1001	PHY-101	2023	12
1003	CS-190	2023	7
1004	CS-315	2023	10

takes

ID	*name*	*dept_name*	*total_cred*	*course_id*	*year*	*grade*
1001	Anna	Comp. Sci.	100	PHY-101	2023	12
1003	David	Comp. Sci.	147	CS-190	2023	7

Dangerous in Natural Join

AALBORG UNIVERSITY

- List the names of students along with the titles of courses that they have taken

ID	*name*	*dept_name*	*total_cred*
1001	Anna	Comp. Sci.	100
1002	Christian	Comp. Sci.	152
1003	David	Comp. Sci.	147

student

course_id	*title*	*dept_name*	*credits*
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3
PHY-101	Physical Principles	Physics	4

courses

ID	*course_id*	*year*	*grade*
1001	PHY-101	2023	12
1003	CS-190	2023	7
1004	CS-315	2023	10

takes

```
select name, title
from student natural join takes, course
where takes.course_id = course.course_id;
```

```
select name, title
from student natural join takes natural join course;
```

Incorrect!

Dangerous in Natural Join (Cont.)

AALBORG UNIVERSITY

- List the names of students along with the titles of courses that they have taken

ID	*name*	*dept_name*	*total_cred*	*course_id*	*year*	*grade*
1001	Anna	Comp. Sci.	100	PHY-101	2023	12
1003	David	Comp. Sci.	147	CS-190	2023	7

student **natural join** *takes*

course_id	*title*	*dept_name*	*credits*
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3
PHY-101	Physical Principles	Physics	4

courses

```
select name, title
from student natural join takes, course
where takes.course_id = course.course_id;
```

```
select name, title
from student natural join takes natural join course;
```

Incorrect!

Dangerous in Natural Join (Cont.)

AALBORG UNIVERSITY

- List the names of students along with the titles of courses that they have taken

```
select name, title
from student natural join takes, course
where takes.course_id = course.course_id;
```

name	*title*
Anna	Physical Principles
David	Game Design

```
select name, title
from student natural join takes natural join course;
```

Incorrect!

name	*title*
David	Game Design

- The second query omits all (student name, course title) pairs where the student takes a course in a department other than the student's own department.

Join with using Clause

AALBORG UNIVERSITY

- To avoid the danger of equating attributes erroneously, we can use the "**using**" construct that allows us to specify exactly which columns should be equated.
- Query example

```
select name, title
from  (student natural join takes) join course using (course_id);
```

Join Condition

AALBORG UNIVERSITY

- The **on** condition allows a general predicate over the relations being joined
- This predicate is written like a **where** clause predicate except for the use of the keyword **on**

```
select name, course_id
from student join takes on student.ID = takes.ID;
```

```
select name, course_id
from student natural join takes;
```

```
select name, course_id
from student join takes using (ID);
```

```
select name, course_id
from student, takes
where student.ID = takes.ID;
```

Outer Join

AALBORG UNIVERSITY

- An extension of the join operation that avoids loss of information.
- Computes the join and then adds tuples form one relation that does not match tuples in the other relation to the result of the join.
- Uses *null* values.
- Three forms of outer join:
  - left outer join
  - right outer join
  - full outer join

Left Outer Join

AALBORG UNIVERSITY

```
select *
from student natural left outer join takes;
```

ID	*name*	*dept_name*	*total_cred*
1001	Anna	Comp. Sci.	100
1002	Christian	Comp. Sci.	152
1003	David	Comp. Sci.	147

student

ID	*course_id*	*year*	*grade*
1001	PHY-101	2023	12
1003	CS-190	2023	7
1004	CS-315	2023	10

takes

ID	*name*	*dept_name*	*total_cred*	*course_id*	*year*	*grade*
1001	Anna	Comp. Sci.	100	PHY-101	2023	12
1002	Christian	Comp. Sci.	152	null	null	null
1003	David	Comp. Sci.	147	CS-190	2023	7

Right Outer Join

AALBORG UNIVERSITY

```
select *
from student natural right outer join takes;
```

ID	*name*	*dept_name*	*total_cred*
1001	Anna	Comp. Sci.	100
1002	Christian	Comp. Sci.	152
1003	David	Comp. Sci.	147

student

ID	*course_id*	*year*	*grade*
1001	PHY-101	2023	12
1003	CS-190	2023	7
1004	CS-315	2023	10

takes

ID	*name*	*dept_name*	*total_cred*	*course_id*	*year*	*grade*
1001	Anna	Comp. Sci.	100	PHY-101	2023	12
1003	David	Comp. Sci.	147	CS-190	2023	7
1004	null	null	null	CS-315	2023	10

Full Outer Join

AALBORG UNIVERSITY

```
select *
from student natural full outer join takes;
```

ID	*name*	*dept_name*	*total_cred*
1001	Anna	Comp. Sci.	100
1002	Christian	Comp. Sci.	152
1003	David	Comp. Sci.	147

student

ID	*course_id*	*year*	*grade*
1001	PHY-101	2023	12
1003	CS-190	2023	7
1004	CS-315	2023	10

takes

ID	*name*	*dept_name*	*total_cred*	*course_id*	*year*	*grade*
1001	Anna	Comp. Sci.	100	PHY-101	2023	12
1002	Christian	Comp. Sci.	152	null	null	null
1003	David	Comp. Sci.	147	CS-190	2023	7
1004	null	null	null	CS-315	2023	10

Agenda

- Null Values
- Modification of the Database
- Join Expression
- Nested Subqueries
- Views, transactions and integrity constraints

Nested Subqueries

AALBORG UNIVERSITY

- SQL provides a mechanism for the nesting of subqueries.
- A **subquery** is a **select-from-where** expression that is nested within another query.
- The nesting can be done as follows:
  - **From clause:** r_i can be replaced by any valid subquery
  - **Where clause:** *P* can be replaced with an expression of the form:

    B <operation> (subquery)

    B is an attribute and <operation> to be defined later.
  - **Select clause:** A_i can be replaced be a subquery that generates a single value.

Nested Subqueries

AALBORG UNIVERSITY

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

```
select name
from instructor
where salary > (select avg(salary) from instructor);
```

```
select name, (select avg(salary) from instructor) as avg_salary
from instructor;
```

Set Membership

AALBORG UNIVERSITY

```
select distinct name
from instructor
where name not in ('Mozart', 'Einstein');
```

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
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

- Find all instructors whose name is neither “Mozart” nor “Einstein”

Set Membership--in

AALBORG UNIVERSITY

```
select distinct course_id
from section
where semester = 'Fall' and year = 2017 and
      course_id in (select course_id
                    from section
                    where semester = 'Spring' and year = 2018);
```

course_id	sec_id	semester	year	building	room_number	time_slot_id
BIO-101	1	Summer	2017	Painter	514	B
BIO-301	1	Summer	2018	Painter	514	A
CS-101	1	Fall	2017	Packard	101	H
CS-101	1	Spring	2018	Packard	101	F
CS-190	1	Spring	2017	Taylor	3128	E
CS-190	2	Spring	2017	Taylor	3128	A
CS-315	1	Spring	2018	Watson	120	D
CS-319	1	Spring	2018	Watson	100	B
CS-319	2	Spring	2018	Taylor	3128	C
CS-347	1	Fall	2017	Taylor	3128	A
EE-181	1	Spring	2017	Taylor	3128	C
FIN-201	1	Spring	2018	Packard	101	B
HIS-351	1	Spring	2018	Painter	514	C
MU-199	1	Spring	2018	Packard	101	D
PHY-101	1	Fall	2017	Watson	100	A

- Find course_ids of courses offered in Fall 2017 and in Spring 2018

Set Membership--not in

AALBORG UNIVERSITY

```
select distinct course_id
from section
where semester = 'Fall' and year = 2017 and
        course_id not in (select course_id
                          from section
                          where semester = 'Spring' and year = 2018);
```

course_id	sec_id	semester	year	building	room_number	time_slot_id
BIO-101	1	Summer	2017	Painter	514	B
BIO-301	1	Summer	2018	Painter	514	A
CS-101	1	Fall	2017	Packard	101	H
CS-101	1	Spring	2018	Packard	101	F
CS-190	1	Spring	2017	Taylor	3128	E
CS-190	2	Spring	2017	Taylor	3128	A
CS-315	1	Spring	2018	Watson	120	D
CS-319	1	Spring	2018	Watson	100	B
CS-319	2	Spring	2018	Taylor	3128	C
CS-347	1	Fall	2017	Taylor	3128	A
EE-181	1	Spring	2017	Taylor	3128	C
FIN-201	1	Spring	2018	Packard	101	B
HIS-351	1	Spring	2018	Painter	514	C
MU-199	1	Spring	2018	Packard	101	D
PHY-101	1	Fall	2017	Watson	100	A

- Find course_ids of courses offered in Fall 2017 but not in Spring 2018

Set Comparison-- some Clause

AALBORG UNIVERSITY

```
select name
from instructor
where salary > some (select salary
                     from instructor
                     where dept name = ' Comp. Sci.');
```

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
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

- Find names of instructors with salary greater than that of some (at least one) instructor in the computer science department.

Definition of some Clause

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

(5 < **some** | 0 / 5 / 6 |) = true (read: 5 < some tuple in the relation)

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

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

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

Set Comparison-- all Clause

AALBORG UNIVERSITY

```
select name
from instructor
where salary > all (select salary
                    from instructor
                    where dept name = 'Comp. Sci.');
```

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
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

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

Definition of all Clause

AALBORG UNIVERSITY

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

(5 < **all** | 0 / 5 / 6 |) = false

(5 < **all** | 6 / 10 |) = true

(5 = **all** | 4 / 5 |) = false

(5 ≠ **all** | 4 / 6 |) = true (since 5 ≠ 4 and 5 ≠ 6)

Test for Empty Relations

AALBORG UNIVERSITY

- The **exists** construct returns the value **true** if the argument subquery is nonempty.
- **exists** $r \Leftrightarrow r \neq \emptyset$
- **not exists** $r \Leftrightarrow r = \emptyset$

exists--Example

AALBORG UNIVERSITY

```
select i.name
from instructor i
where exists (select *
              from teaches t
              where t.ID = i.ID);
```

- Find names of instructors who teach at least one course.
- **Correlation name** – variable in the outer query
- **Correlated subquery** – the inner query

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

instructor

ID	*course_id*	*semester*	*year*
10101	CS-101	Fall	2017
22222	PHY-101	Fall	2017
76766	BIO-101	Summer	2017
83821	CS-190	Spring	2017
83821	CS-190	Spring	2018

teaches

exists--Example

AALBORG UNIVERSITY

```
select i.name
from instructor i
where exists (select *
              from teaches t
              where t.ID = i.ID);
```

ID	*course_id*	*semester*	*year*
10101	CS-101	Fall	2017

ID	*course_id*	*semester*	*year*

ID	*course_id*	*semester*	*year*

ID	*course_id*	*semester*	*year*
83821	CS-190	Spring	2017
83821	CS-190	Spring	2018

ID	*course_id*	*semester*	*year*

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

instructor

ID	*course_id*	*semester*	*year*
10101	CS-101	Fall	2017
22222	PHY-101	Fall	2017
76766	BIO-101	Summer	2017
83821	CS-190	Spring	2017
83821	CS-190	Spring	2018

teaches

not exists--Example

AALBORG UNIVERSITY

```
select i.name
from instructor i
where not exists (select *
                  from teaches t
                  where t.ID= i.ID);
```

- Find names of instructors who do not teach any course.

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

instructor

ID	*course_id*	*semester*	*year*
10101	CS-101	Fall	2017
22222	PHY-101	Fall	2017
76766	BIO-101	Summer	2017
83821	CS-190	Spring	2017
83821	CS-190	Spring	2018

teaches

not exists--Example

AALBORG UNIVERSITY

```
select i.name
from instructor i
where not exists (select *
                  from teaches t
                  where t.ID= i.ID);
```

ID	course_id	semester	year
10101	CS-101	Fall	2017

ID	course_id	semester	year

ID	course_id	semester	year

ID	course_id	semester	year
83821	CS-190	Spring	2017
83821	CS-190	Spring	2018

ID	course_id	semester	year

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

instructor

ID	course_id	semester	year
10101	CS-101	Fall	2017
22222	PHY-101	Fall	2017
76766	BIO-101	Summer	2017
83821	CS-190	Spring	2017
83821	CS-190	Spring	2018

teaches

In vs. exists

AALBORG UNIVERSITY

```
select i.name
from instructor i
where i.ID in (select t.ID
               from teaches t );
```

Is the “left tuple” in the “right set”

```
select i.name
from instructor i
where exists (select *
              from teaches t
              where t.ID= i.ID);
```

Is the “right set” nonempty?

Test for Absence of Duplicate Tuples

AALBORG UNIVERSITY

- The **unique** construct tests whether a subquery has any duplicate tuples in its result.
- The **unique** construct evaluates to “true” if a given subquery contains no duplicates .

unique--Example

AALBORG UNIVERSITY

```
select name
from instructor i
where unique (select course_id
                from teaches t
                where t.ID= i.ID);
```

- Find names of instructors who do not teach duplicate courses.

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

instructor

ID	*course_id*	*semester*	*year*
10101	CS-101	Fall	2017
22222	PHY-101	Fall	2017
76766	BIO-101	Summer	2017
83821	CS-190	Spring	2017
83821	CS-190	Spring	2018

teaches

unique--Example

AALBORG UNIVERSITY

```
select name
from instructor i
where unique (select course_id
              from teaches t
              where t.ID= i.ID);
```

course_id
CS-101

course_id

course_id

course_id
CS-190
CS-190

course_id

The **unique** predicate would evaluate to true on the empty set

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

instructor

ID	course_id	semester	year
10101	CS-101	Fall	2017
22222	PHY-101	Fall	2017
76766	BIO-101	Summer	2017
83821	CS-190	Spring	2017
83821	CS-190	Spring	2018

teaches

not unique--Example

AALBORG UNIVERSITY

```
select name
from instructor i
where not unique (select course_id
                  from teaches t
                  where t.ID= i.ID);
```

- Find names of instructors who teach duplicate courses.

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

instructor

ID	*course_id*	*semester*	*year*
10101	CS-101	Fall	2017
22222	PHY-101	Fall	2017
76766	BIO-101	Summer	2017
83821	CS-190	Spring	2017
83821	CS-190	Spring	2018

teaches

not unique--Example

AALBORG UNIVERSITY

```
select name
from instructor i
where not unique (select course_id
                  from teaches t
                  where t.ID= i.ID);
```

course_id
CS-101

course_id

course_id

course_id
CS-190
CS-190

course_id

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

instructor

ID	course_id	semester	year
10101	CS-101	Fall	2017
22222	PHY-101	Fall	2017
76766	BIO-101	Summer	2017
83821	CS-190	Spring	2017
83821	CS-190	Spring	2018

teaches

Subqueries in the from Clause

AALBORG UNIVERSITY

- SQL allows a subquery expression to be used in the **from** clause

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

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

instructor

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

with Clause

AALBORG UNIVERSITY

- The **with** clause provides a way of defining a temporary relation whose definition is available only to the query in which the **with** clause occurs.

dept_name	*building*	*budget*
Biology	Watson	90000
Comp. Sci.	Taylor	100000
Elec. Eng.	Taylor	85000
Finance	Painter	120000
History	Painter	50000
Music	Packard	80000
Physics	Watson	70000

department

- Find all departments with the maximum budget

```
with max_budget (value) as
        (select max(budget)
         from department)
select department.name
from department, max_budget
where department.budget = max_budget.value;
```

Agenda

- Null Values
- Modification of the Database
- Join Expression
- Nested Subqueries
- Views, transactions and integrity constraints

Views

AALBORG UNIVERSITY

- In some cases, it is not desirable for all users to see the entire logical model (that is, all the actual relations stored in the database.)
  - Consider a person who needs to know an instructor's name and department, but not the salary.
- A **view** provides a mechanism to hide certain data from the view of certain users.

View Definition

AALBORG UNIVERSITY

- A view is defined using the **create view** statement which has the form

create view *v* **as** < query expression >;

where <query expression> is any legal SQL expression. The view name is represented by *v*

- Once a view is defined, the view name can be used to refer to the virtual relation that the view generates

View--Example

AALBORG UNIVERSITY

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
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

instructor

```
select name
from faculty
where dept_name = 'Biology';
```

- A view of instructors without their salary

```
create view faculty as
select ID, name, dept_name
from instructor;
```

- Create a view of department salary totals

```
create view
departments_avg_salary(dept
_name, avg_salary) as
select dept_name, avg (salary)
from instructor
group by dept_name;
```

Transactions

AALBORG UNIVERSITY

- A **transaction** consists of a sequence of query and/or update statements and is a “unit” of work
- **Begin** a transaction

  begin;
- End a transaction and make all the changes made by the transaction **visible** to others and become **permanent**

  commit;
- Reset a transaction and **discard** all the changes made by the transaction

  rollback;

Integrity Constraints

AALBORG UNIVERSITY

- Integrity constraints guard against accidental damage to the database, by ensuring that authorized changes to the database do not result in a loss of data consistency.
  - A salary of a bank employee must be at least 200 an hour
  - A customer must have a (non-null) phone number

Constraints on a Single Relation

AALBORG UNIVERSITY

- **not null**
- **primary key**
- **unique**
- **check** (P), where P is a predicate

The check Clause

AALBORG UNIVERSITY

- The **check** (P) clause specifies a predicate P that must be satisfied by every tuple in a relation.
- Example: ensure that semester is one of fall, winter, spring or summer

```
create table section
        (course_id varchar (8),
         sec_id varchar (8),
         semester varchar (6),
         year numeric (4,0),
         building varchar (15),
         room_number varchar (7),
         time_slot_id varchar (4),
         primary key (course_id, sec_id, semester, year),
         check (semester in ('Fall', 'Winter', 'Spring', 'Summer')));
```

Referential Integrity

AALBORG UNIVERSITY

- Foreign keys can be specified as part of the SQL **create table** statement

  foreign key (*dept_name*) **references** *department*

- By default, a foreign key references the primary-key attributes of the referenced table.
- SQL allows a list of attributes of the referenced relation to be specified explicitly.

  foreign key (*dept_name*) **references** *department* (*dept_name*)

Referential Integrity

AALBORG UNIVERSITY

dept_name	*building*	*budget*
Comp. Sci.	Taylor	100000
Finance	Painter	120000
Music	Packard	80000
Elec. Eng.	Taylor	85000

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(dept_name));
```

```
insert into instructor
values (22222, 'Einstein', 'Physics', 95000);
```

```
insert into instructor
values (22222, 'Einstein', null, 95000);
```

Foreign keys must always reference existing tuples or be **null**

Exercise 3

AALBORG UNIVERSITY

- Create a *new_student* table
  - Five attributes: ID char(5), name varchar(20), gender char(1), dept_name varchar(20), credit numeric(5, 2)
  - The primary key is ID
  - The dept_name references department(dept_name)
  - The name should not be empty
  - The default value of credit is 0.00
  - The credit should not less than 0.00
  - The gender should only contain values “F” and “M”

Exercise 3--Solution

AALBORG UNIVERSITY

- Create a new_student table
  - Five attributes: ID char(5), name varchar(20), gender char(1), dept_name varchar(20), credit numeric(5, 2)
  - The primary key is ID
  - The dept_name references department(dept_name)
  - The name should not be empty
  - The default value of credit is 0.00
  - The credit should not less than 0.00
  - The gender should only contain values “F” and “M”

```
create table new_student (
ID          char(5),
name        varchar(20)   not null,
dept_name   varchar(20),
credit      numeric(5,2) default 0.00,
primary key (ID),
foreign key (dept_name) references
department(dept_name)
check (credit >= 0.00)
check (gender in ('F', 'M'))
);
```

Handling Updates

AALBORG UNIVERSITY

- Dynamic integrity constraints need to be fulfilled by each change of a database
- Possible responses to changes of referenced data
  - Rejection of updates (**default** behavior)
  - Propagation of updates (**cascade**)
  - Set references to “unknown” (**set null**)

Example

AALBORG UNIVERSITY

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

instructor

dept_name	*building*	*budget*
Comp. Sci.	Taylor	100000
Finance	Painter	120000
Music	Packard	80000
Elec. Eng.	Taylor	85000

department

```
update department
set dept_name = 'Computer Science'
where dept_name = 'Comp. Sci.';
```

```
delete department
where dept_name = 'Comp. Sci.';
```

What to do now

Option 1: reject the update

AALBORG UNIVERSITY

```
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(dept_name));
```

```
update department
set dept_name = 'Computer Science'
where dept_name = 'Comp. Sci.';
```

```
delete department
where dept_name = 'Comp. Sci.';
```

Result of the operation:

DB remains unchanged

Option 2: cascading the update

AALBORG UNIVERSITY

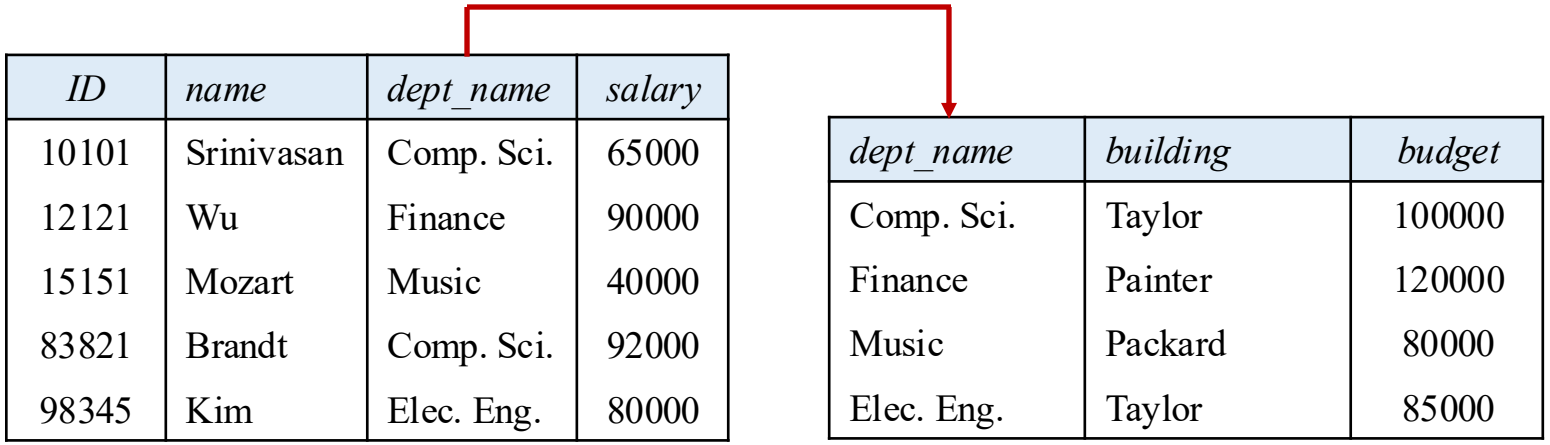
```
create table instructor (
ID          char(5),
name        varchar(20)   not null,
dept_name   varchar(20),
salary          numeric(8,2),
primary key (ID),
foreign key (dept_name) references department(dept_name)
on update cascade on delete cascade);
```

```
update department
set dept_name = 'Computer Science'
where dept_name = 'Comp. Sci.';
```

```
delete department
where dept_name = 'Comp. Sci.';
```

Option 2: cascading the update

AALBORG UNIVERSITY



ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

instructor

dept_name	*building*	*budget*
Comp. Sci.	Taylor	100000
Finance	Painter	120000
Music	Packard	80000
Elec. Eng.	Taylor	85000

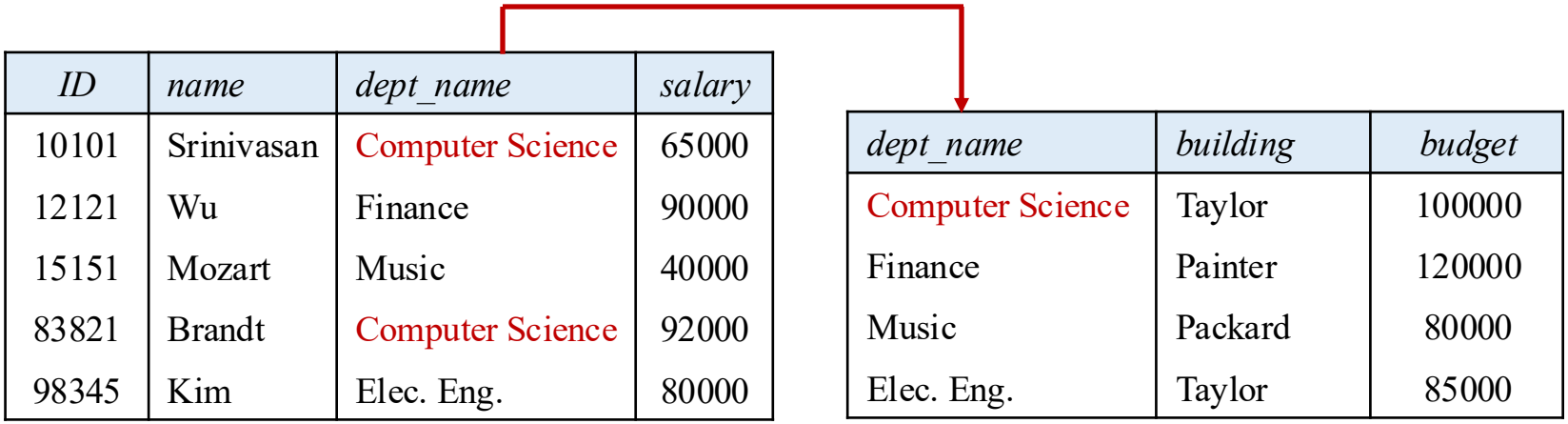
department

```
update department
set dept_name = 'Computer Science'
where dept_name = 'Comp. Sci.';
```

Result of the **update** operation: Keys in *department* and *instructor* updated

Option 2: cascading the update

AALBORG UNIVERSITY



ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Computer Science	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
83821	Brandt	Computer Science	92000
98345	Kim	Elec. Eng.	80000

instructor

dept_name	*building*	*budget*
Computer Science	Taylor	100000
Finance	Painter	120000
Music	Packard	80000
Elec. Eng.	Taylor	85000

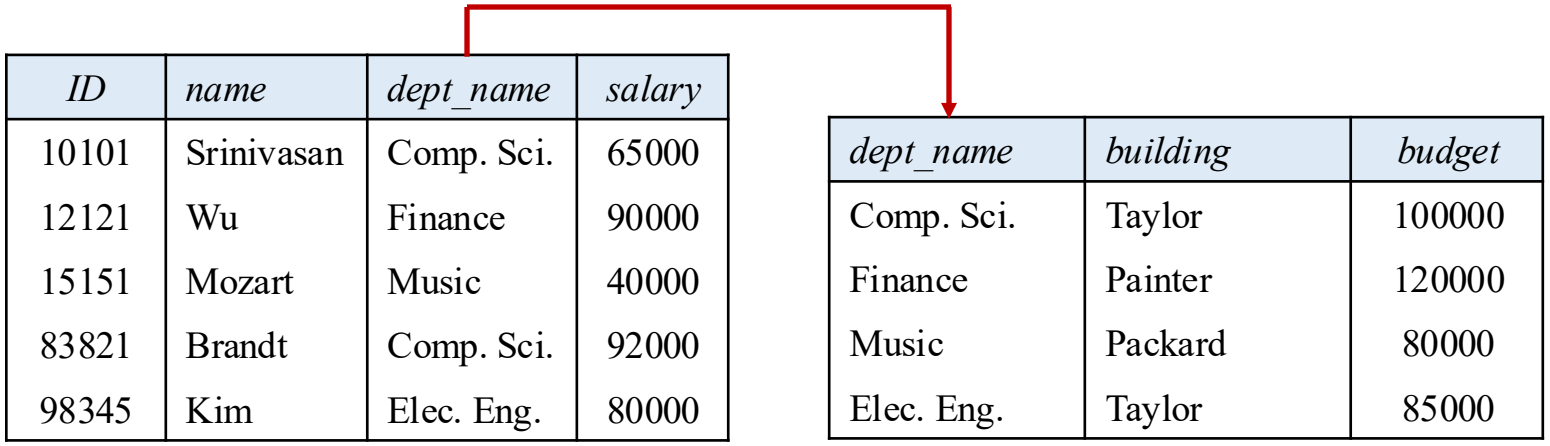
department

```
update department
set dept_name = 'Computer Science'
where dept_name = 'Comp. Sci.';
```

Result of the **update** operation: Keys in *department* and *instructor* updated

Option 2: cascading the update

AALBORG UNIVERSITY



ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

instructor

dept_name	*building*	*budget*
Comp. Sci.	Taylor	100000
Finance	Painter	120000
Music	Packard	80000
Elec. Eng.	Taylor	85000

department

delete *department*
where *dept_name* = 'Comp. Sci.';

Result of the **delete** operation: Tuples in *department* and *instructor* deleted

Option 2: cascading the update

ID	*name*	*dept_name*	*salary*
12121	Wu	Finance	90000
15151	Mozart	Music	40000
98345	Kim	Elec. Eng.	80000

instructor

dept_name	*building*	*budget*
Finance	Painter	120000
Music	Packard	80000
Elec. Eng.	Taylor	85000

department

delete *department*
where *dept_name* = 'Comp. Sci.';

Result of the **delete** operation: Tuples in *department* and *instructor* deleted

Option 3: update and set null

AALBORG UNIVERSITY

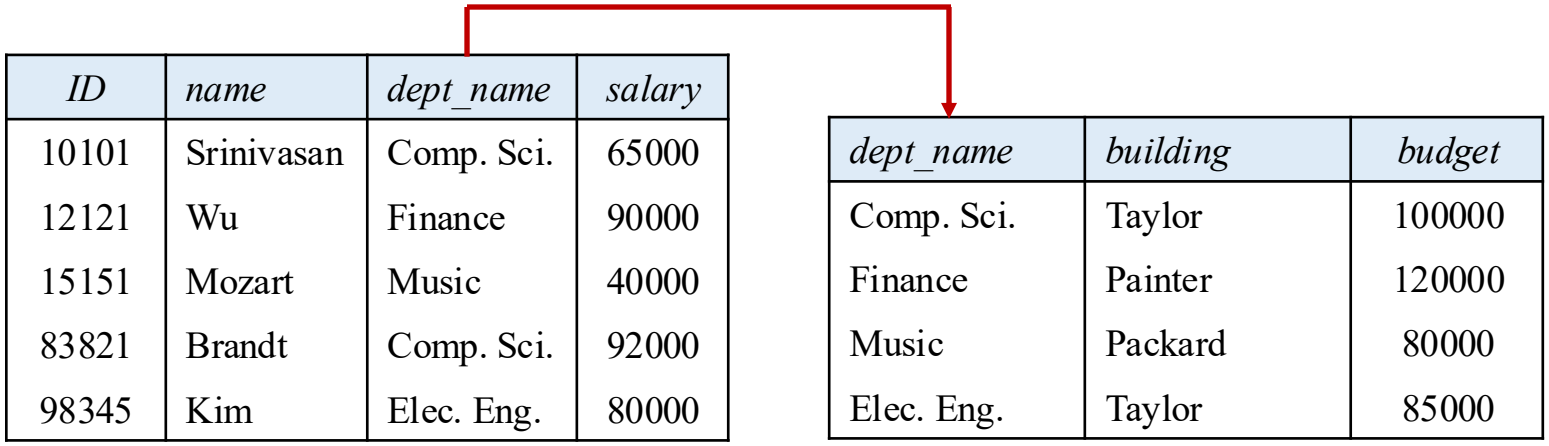
```
create table instructor (
ID          char(5),
name        varchar(20)   not null,
dept_name   varchar(20),
salary          numeric(8,2),
primary key (ID),
foreign key (dept_name) references department(dept_name)
on update set null on delete set null);
```

```
update department
set dept_name = 'Computer Science'
where dept_name = 'Comp. Sci.';
```

```
delete department
where dept_name = 'Comp. Sci.';
```

Option 3: update and set null

AALBORG UNIVERSITY



ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

instructor

dept_name	*building*	*budget*
Comp. Sci.	Taylor	100000
Finance	Painter	120000
Music	Packard	80000
Elec. Eng.	Taylor	85000

department

```
update department
set dept_name = 'Computer Science'
where dept_name = 'Comp. Sci.';
```

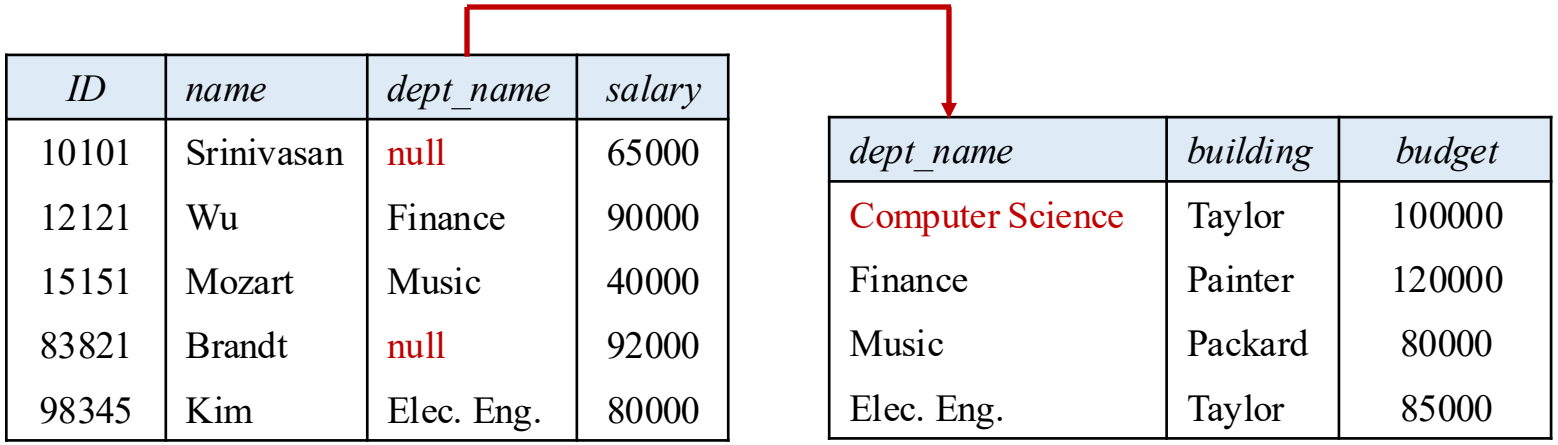
Result of the **update** operation:

dept_name in *department* set to 'Computer Science',

dept_name in *instructor* set to null

Option 3: update and set null

AALBORG UNIVERSITY



ID	*name*	*dept_name*	*salary*
10101	Srinivasan	null	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
83821	Brandt	null	92000
98345	Kim	Elec. Eng.	80000

instructor

dept_name	*building*	*budget*
Computer Science	Taylor	100000
Finance	Painter	120000
Music	Packard	80000
Elec. Eng.	Taylor	85000

department

```
update department
set dept_name = 'Computer Science'
where dept_name = 'Comp. Sci.';
```

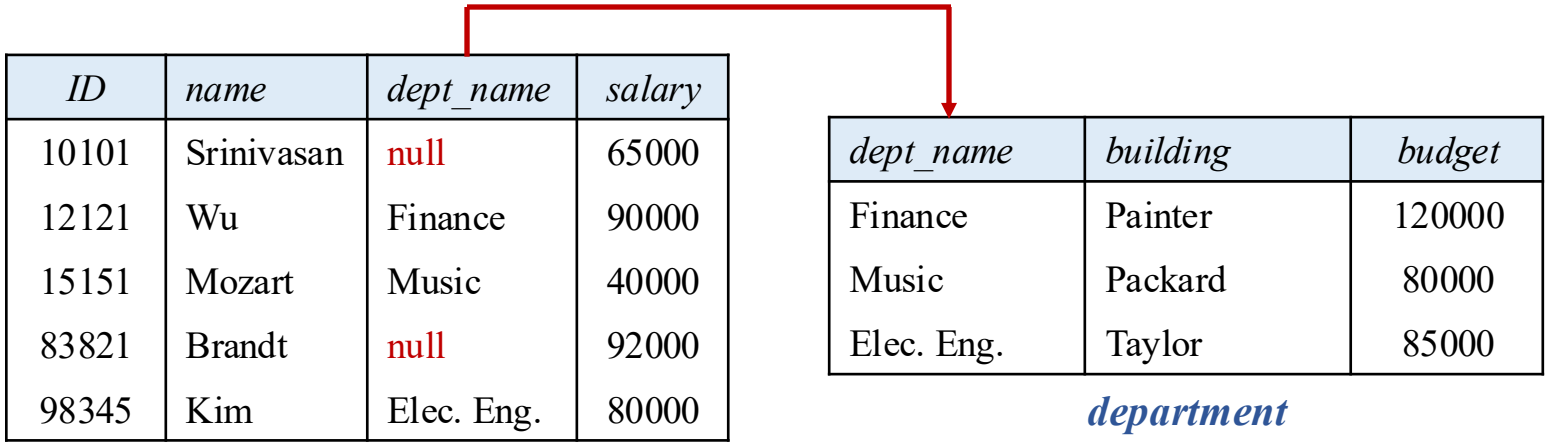
Result of the **update** operation:

dept_name in *department* set to 'Computer Science',

dept_name in *instructor* set to null

Option 3: update and set null

AALBORG UNIVERSITY



ID	*name*	*dept_name*	*salary*
10101	Srinivasan	null	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
83821	Brandt	null	92000
98345	Kim	Elec. Eng.	80000

instructor

dept_name	*building*	*budget*
Finance	Painter	120000
Music	Packard	80000
Elec. Eng.	Taylor	85000

department

delete *department*
where *dept_name* = 'Comp. Sci.';

Result of the **delete** operation:

Tuple in *department* deleted

dept_name in *instructor* set to NULL

Summary

AALBORG UNIVERSITY

- Null Values
- Modification of the Database
- Join Expression
- Nested Subqueries
- Views, transactions and integrity constraints
- Authorization and recursive queries (optional)

Next Lecture

- The Entity-Relationship Model
  - Create non-trivial ER diagrams
  - Analyze if an ER diagram is good or bad
  - Create and explain the mapping of ER diagrams to relations
  - Use a particular ER notation properly

Lecture time

AALBORG UNIVERSITY

- Current
  - 8:15 – 10:00 Lecture 1
  - 10:15 –12:00 Exercise session 1
  - 12:30 – 14:15 Lecture 2
  - 14:30 – 16:15 Exercise session 2
- How about
  - 9:00 – 10:45 Lecture 1
  - 11:00 –12:00 Exercise session 1
  - 12:30 – 14:15 Lecture 2
  - 14:30 – 17:00 Exercise session 1&2

Authorization

AALBORG UNIVERSITY

- We may assign a user several forms of authorizations on parts of the database.
  - **read** - allows reading, but not modification of data.
  - **insert** - allows insertion of new data, but not modification of existing data.
  - **update** - allows modification, but not deletion of data.
  - **delete** - allows deletion of data.
- Each of these types of authorizations is called a **privilege**.
- We may authorize the user all, none, or a combination of these types of privileges on specified parts of a database, such as a relation or a view.

Authorization Specification in SQL

AALBORG UNIVERSITY

- The **grant** statement is used to confer authorization

  grant <privilege list> **on** <relation or view > **to** <user list>
- <user list> is:
  - a user-id
  - **public**, which allows all valid users the privilege granted
  - A role (more on this later)
- Example:

  grant select on *department* **to** Amit, Satoshi
- Granting a privilege on a view does not imply granting any privileges on the underlying relations.
- The grantor of the privilege must already hold the privilege on the specified item (or be the database administrator).

Privileges in SQL

AALBORG UNIVERSITY

- **select**: allows read access to relation, or the ability to query using the view
  - Example: grant users U_1, U_2, and U_3 **select** authorization on the *instructor* relation:

    grant select on *instructor* **to** U_1, U_2, U_3
- **insert**: the ability to insert tuples
- **update**: the ability to update using the SQL update statement
- **delete**: the ability to delete tuples.
- **all privileges**: used as a short form for all the allowable privileges

Revoking Authorization in SQL

AALBORG UNIVERSITY

- The **revoke** statement is used to revoke authorization.

  revoke <privilege list> **on** <relation or view> **from** <user list>

- Example:

  revoke select on *student* **from** U_1, U_2, U_3

- <privilege-list> may be **all** to revoke all privileges the revokee may hold.
- If <revokee-list> includes **public,** all users lose the privilege except those granted it explicitly.
- If the same privilege was granted twice to the same user by different grantees, the user may retain the privilege after the revocation.
- All privileges that depend on the privilege being revoked are also revoked.

Role

AALBORG UNIVERSITY

- A **role** is a way to distinguish among various users as far as what these users can access/update in the database.
- To create a role we use:

  create role <name>

  - Example:

    create role *instructor*

- Once a role is created, we can assign "users" to the role using:

  grant <role> **to** <users>

- Example:

  grant *instructor* **to** Amit

Recursion in SQL

AALBORG UNIVERSITY

course_id	*title*	*dept_name*	*credits*
SW-101	ALG	Software	5
SW-105	DBS	Software	5
SW-107	DIS	Software	5
SW-108	MDLS	Software	5

course

course_id	*prereq_id*
SW-108	SW-107
SW-107	SW-105
SW-105	SW-101

prereq

Which courses need to be taken before taking the course “MDLS”?

selet *course_id, title*
From *course, prereq*
where *prereq.prereq_id = course.course.id* **and** *title =* ’MDLS’;

But this query only finds the direct predecessors

Recursion in SQL

AALBORG UNIVERSITY

course_id	*title*	*dept_name*	*credits*
SW-101	ALG	Software	5
SW-105	DBS	Software	5
SW-107	DIS	Software	5
SW-108	MDLS	Software	5

course

course_id	*prereq_id*
SW-108	SW-107
SW-107	SW-105
SW-105	SW-101

prereq

Which courses need to be taken before taking the course "MDLS"?

```
with recursive rec_prereq(course_id, prereq_id) as (
        select course_id, prereq_id
        from prereq
    union
        select rec_prereq.course_id, prereq.prereq_id
        from rec_rereq, prereq
        where rec_prereq.prereq_id = prereq.course_id
    )
select *
from rec_prereq, course
Where rec_ prereq.prereq_id = course.course.id and title
= 'MDLS;
```

Recursion in SQL

AALBORG UNIVERSITY