**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);***

instructor: relation

ID: attribute

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

**delete from** *student*

removes all tuples from the student relation

**drop table** *r*

**alter table** *r* **add** *A D*

A is the name of the attribute to be added to relation r and D is the domain (type) of A

Tablodaki var olan verilerin bu sütun için değeri null olur

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

The result of SQL query is relation

**select distinct** *dept\_name* **from** *instructor*

removes duplicates

**select all** *dept\_name* **from** *instructor*

*duplicates are not removed*

A close-up of a computer screen

Description automatically generated

A computer screen shot of a computer

Description automatically generated

A computer screen shot of a computer

Description automatically generated

for every tuple in table

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

A screenshot of a computer

Description automatically generated

where: corresponds to selection predicate of the relational algebra

from: corresponds to cartesian product operation of the relational algebra

**select** \* **from** *instructor, teaches*

**select** *name* **from** *instructor* **where** *dept\_name =* *'*Comp. Sci.'

and, or, not

<, <=, >, >=, =, and <>

**select** *name* **from** *instructor* **where** *dept\_name =* *'*Comp. Sci.'**and** *salary* > 70000

Find the names of all instructors who have taught some course and the course\_id

**select** *name, course\_id***from** *instructor , teaches***where** *instructor.ID = teaches.ID*

Find the names of all instructors in the Art department who have taught some course and the course\_id

**select** *name, course\_id***from** *instructor , teaches***where** *instructor.ID = teaches.ID* ***and*** *instructor. dept\_name =* 'Art'

**select distinct** *T.name***from** *instructor* **as** *T, instructor* **as** *S***where** *T.salary > S.salary* **and** *S.dept\_name = 'Comp. Sci.’*

maaşı bilgisayar mühendisliğindeki maaştan yüksek olan instructorları listeler

as demesen de olur -----> instructor as T = instructor T

**se**le**ct** *name***from** *instructor***where***name* **like '**%dar%'

içinde dar geçen isimler

**like '**100 \%' **escape '**\'

100% ile eşleşen string ararken

‘Intro%’ ----> Intro ile başlayan stringler

‘%Comp%’ ----> içinde Comp substringini içerenler

‘\_\_\_’ ------> tam 3 karakter içeren stringler

‘\_\_\_%’ ------------> en az 3 karakter içeren stringler

|| ---------> concatenation

**select distinct** *name***from** *instructor***order by** *name*

name’den sonra desc diyebilirsin (asc is default)

name, dept\_name… ----> pek çok attribute kullanabilirsin

**select** *name*, *course\_id***from** *instructor*, *teaches***where** (*instructor*.*ID*, *dept\_name*) = (*teaches*.*ID*, 'Biology');

**select** *name***from** *instructor*  
**where** *salary* **between** 90000 **and** 100000

A white paper with black text

Description automatically generated

* To retain all duplicates use the
  + **union all**,
  + **intersect all**
  + **except all**

**Aggregate Functions**

**avg:** average value  
 **min:** minimum value  
 **max:** maximum value  
 **sum:** sum of values  
 **count:** number of values

**select avg** (*salary*)  
**from** *instructor***where** *dept\_name*= 'Comp. Sci.';

**select count** (**distinct** *ID*)  
**from** *teaches***where** *semester* = 'Spring' **and** *year* = 2018;

Get number of tuples:

**select count** (\*)  
**from** *course*;

**GROUP BY**

Find the average salary of instructors in each department

**select** *dept\_name*, **avg** (*salary*) **as** *avg\_salary*  
**from** *instructor***group by** *dept\_name*;

In standard SQL, when using a GROUP BY clause, any columns in the SELECT list that are not included in an aggregate function must be included in the GROUP BY clause.

/\* erroneous query \*/  
**select** *dept\_name*, *ID*, **avg** (*salary*)  
**from** *instructor***group by** *dept\_name*;

Find the names and average salaries of all departments whose average salary is greater than 42000

**select** *dept\_name*, **avg** (*salary*) **as** *avg\_salary*

**from** *instructor*

**group by** *dept\_name*

**having avg** (*salary*) > 42000;

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

**select** *A*1, *A*2, ..., *An*  
**from** *r*1, *r*2, ..., *rm*  
**where** *P*

r 🡪 herhangi bir subquery eklenebilir

P 🡪 expression olabilir:

B <operation> (subquery)

B: attribute

A 🡪 single value üreten bir subquery olabilir

Find courses offered in Fall 2017 and in Spring 2018

**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);

Find courses offered in Fall 2017 but not in Spring 2018

**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);

Names other than Mozart and Einstein

**select distinct** *name*

**from** *instructor*

**where**  *name* **not in** ('Mozart', 'Einstein')

Find the total number of (distinct) students who have taken course sections taught by the instructor with *ID* 10101

**select count** (**distinct** *ID*)

**from** *takes*

**where** (*course\_id*, *sec\_id*, *semester*, *year*) **in** (**select** *course\_id*, *sec\_id*, *semester*, *year*

**from** *teaches*

**where** *teaches*.*ID*= 10101);

**some vs all**

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

**select distinct** *T*.*name*

**from** *instructor* **as** *T*, *instructor* **as** *S*

**where** *T.salary* > *S.salary* **and** *S.dept name* = 'Biology';

Same query using > **some** clause

**select** *name*

**from** *instructor*

**where** *salary* > **some** (**select** *salary*

**from** *instructor*

**where** *dept name* = 'Biology');

---------------🡪 salary > some tuple in the relation

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

**select** *name*

**from** *instructor*

**where** *salary* > **all** (**select** *salary*

**from** *instructor*

**where** *dept name* = 'Biology');

exists

* **exists**  *r* ⇔ *r* ≠ *Ø*
* **not exists** *r* ⇔ *r* = *Ø*

Find all courses taught in both the Fall 2017 semester and in the Spring 2018 semester

**select** *course\_id* **from** *section* **as** *S* **where** *semester* = 'Fall' **and** *year* = 2017 **and   
 exists** (**select** \*  
 **from** *section* **as** *T* **where** *semester* = 'Spring' **and** *year*= 2018   
 **and** *S*.*course\_id* = *T*.*course\_id*)

Find all students who have taken all courses offered in the Biology department.

**select distinct** *S*.*ID*, *S*.*name*

**from** *student* **as** *S*

**where not exists** ( (**select** *course\_id*

**from** *course*

**where** *dept\_name* = 'Biology')

**except**

(**select** *T*.*course\_id*

**from** *takes* **as** *T*

**where** *S*.*ID* = *T*.*ID*));

**unique**

subquery’de duplicate tuple var mı yok mu bakar

duplicate yoksa true döner

Find all courses that were offered at most once in 2017

**select** *T*.*course\_id***from** *course* **as** *T***where unique** ( **select** *R*.*course\_id* **from** *section* **as** *R* **where** *T*.*course\_id*= *R*.*course\_id* **and** *R*.*year* = 2017);

**subqueries in form**

Find the average instructors’ salaries of those departments where the average salary is greater than $42,000.”

**select** *dept\_name*, *avg\_salary***from** ( **select** *dept\_name*, **avg** (*salary*) **as** *avg\_salary* **from** *instructor* **group by** *dept\_name*)  
**where** *avg\_salary* > 42000;

**select** *dept\_name*, *avg\_salary***from** ( **select** *dept\_name*, **avg** (*salary*) **from** *instructor* **group by** *dept\_name*)

**as** *dept\_avg* (*dept\_name*, *avg\_salary*)

**where** *avg\_salary* > 42000;

select dept\_name, avg(salary) as avg\_salary from instructor group by dept\_name having avg\_salary > 42000

**with**

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*;

Find all departments where the total salary is greater than the average of the total salary at all departments

**with** *dept \_total* (*dept\_name*, *value*) **as**

(**select** *dept\_name*, **sum**(*salary*)

**from** *instructor*

**group by** *dept\_name*),

*dept\_total\_avg*(*value*) **as**

(**select avg**(*value*)

**from** *dept\_total*)

**select** *dept\_name*

**from** *dept\_total*, *dept\_total\_avg*

**where** *dept\_total.value* > *dept\_total\_avg.value*;

scalar subquery: single value is expected

List all departments along with the number of instructors in each department

**select** *dept\_name*,   
 ( **select count**(\*)   
 **from** *instructor* **where** *department*.*dept\_name* = *instructor*.*dept\_name*)  
 **as** *num\_instructors***from** *department*;

Runtime error if subquery returns more than one result tuple

**DELETION**

delete all instructors

delete from instructor

delete all instructors from finance department

delete from instructor where dept\_name = ‘Finance’

delete all tuples in the instructor relation for those instructors associated with a department located in the Watson building

delete from instructor

where dept\_name in (select dept\_name from department where building = ‘Watson’)

Delete all instructors whose salary is less than the average salary of instructors

**delete from** *instructor*

**where** *salary* < (**select avg** (*salary*)

**from** *instructor*);

**INSERTION**

**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** *student* **values** ('3003', 'Green', 'Finance', *null*);

Make each student in the Music department who has earned more than 144 credit hours an instructor in the Music department with a salary of $18,000.

**insert into** *instructor* **select** *ID, name, dept\_name, 18000* **from**  *student* **where**  *dept\_name = '*Music' **and** *total\_cred*>144;

**UPDATES**

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;

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

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

* Increase salaries of instructors whose salary is over $100,000 by 3%, and all others by a 5%
  + Write two **update** statements:

**update** *instructor* **set** *salary* = *salary* \* 1.03  
 **where** *salary* > 100000;  
 **update** *instructor* **set** *salary* = *salary* \* 1.05  
 **where** *salary* <= 100000;

* + The order is important
  + Can be done better using the **case** statement (NEXT)

**CASE**

**update** *instructor* **set** *salary* = **case  
 when** *salary* <= 100000 **then** *salary* \* 1.05  
 **else** *salary* \* 1.03  
 **end**

update with scalar subqueries

**update** *student S* **set** *tot\_cred* = (**select sum**(*credits*)  
 **from** *takes, course* **where** *takes.course\_id* = *course.course\_id* **and** *S*.*ID*= *takes*.*ID* **and** *takes*.*grade* <> 'F' **and** *takes*.*grade* **is not null**);

SELECT

FROM

WHERE

GROUP BY

HAVING

A screenshot of a computer

Description automatically generated

avg(unitPrice) as avg\_price

sum(unitPrice)

select customername from customers where customerid in (select customerid from orders groupBy customerId having avg(totalAmount) > 300)