SQL

INTRODUCTION TO DATA SCIENCE

TIM KRASKA

teaching datascience .org

THE 1ST PROJECT

- Goal: get started to develop a data product
- Projects can range from gathering and cleaning a data set, over repeating an existing study or web-site to visualize a data set to testing an entirely new hypothesis.
- Groups of 4
  - Piazza helps you to find group members (see special post)

2/16/2016	Pre-Proposal Handin
3/3/2016	Advisor Checkin
3/22/2016	Mid-term report
4/12/2016	Full Project Proposal
4/21/2016	Advisor Checkin
5/3/2016	Advisor Checkin
5/12/2016	Final Project Due

- Mid-term report has to be a public blog post (afterwards you have to write weekly progress reports for the 2nd project)
- 2nd project can be a continuations of the 1st, but doesn't have to be
- 10% of your grade (the final project will be 25%).
- Today, we will publish a list of potential data sources and some project ideas on the web-page
- However, **you are encouraged to find your own data and create your own project**
- Don‘t worry to much about that you do not know all the tools yet

FORMAL DEFINITION OF REL. ALGEBRA

- **Atoms** (basic expressions)
- A **relation** in the database
- A **constant relation**
- **Operators** (composite expressions)
- **Selection**: σ (E1)
- **Projection**: Π (E1)
- **Cartesian Product**: E1 x E2
- **Rename**: $\rho_V(E1)$, $\rho_{A \leftarrow B}(E1)$
- **Union**: E1 $\cup$ E2
- **Minus**: E1 - E2

CODD`S THEOREM

3 Languages:

- **Relational Algebra**
- **Tuple Relational Calculus** (safe expressions only)
- **Domain Relational Calculus** (safe expressions only)

are equivalent.

Impact of Codd`s theorem:

- SQL is based on the **relational calculus**
- SQL implementation is based on **relational algebra**
- **Codd`s theorem shows that SQL implementation is correct and complete.**

NOT COVERED

Set Division

Aggregate Functions

Codd´s Proof

…

IN CLASS TASK

Player

PlayerID	Name	Age	Team
1	Russel	27	Seahawks
...	...	...	...

Team

Team	State
Seahawks	Washington
...	...

Played

PlayerID	Date	Place	Score
1	2/1/15	Phoenix	3
...	...	...	...

In relational algebra:

1) Return all teams, who played at least once in Phoenix
2) Return all Seahawks player, who did not play in the entire season

CLICKER

Player

PlayerID	Name	Age	Team
1	Russel	27	Seahawks
…	…	…	…

Team

Team	State
Seahawks	Washington
…	…

Played

PlayerID	Date	Place	Score
1	2/1/15	Phoenix	3
…	…	…	…

Return all teams, who played at least once in Phoenix

A) Π_{Team} ($\sigma_{\text{Place=`Phoenix`}}$ (Player X Played))

B) Π_{Team} Player $\rtimes$($\sigma_{\text{Place=`Phoenix}}$ (Played))

C) Π_{Team} ($\sigma_{\text{Place=`Phoenix`}}$ (Player $\bowtie$ Played))

CLICKER

1. $\Pi_{\text{Team}}\ (\sigma_{\text{Place=`Phoenix`}}\ (\text{Player} \bowtie \text{Played}))$
2. $\Pi_{\text{Team}}\ \text{Player} \bowtie (\sigma_{\text{Place=`Phoenix}}\ \text{Played})$
3. $\Pi_{\text{Team}}\ (\sigma_{\text{Place=`Phoenix`}}\ (\text{Player} \bowtie \text{Played} \bowtie \text{Team}))$

Which of these expressions are equivalent?

A) All
B) 1 and 2
C) 1 and 3
D) 2 and 3
E) None

CLICKER

Player

PlayerID	Name	Age	Team
1	Russel	27	Seahawks
...	...	...	...

Team

Team	State
Seahawks	Washington
...	...

Played

PlayerID	Date	Place	Score
1	2/1/15	Phoenix	3
...	...	...	...

Return all Seahawks player names, who did not play so far

A) $\Pi_{Name}(\sigma_{Team=`Seahawks`}(Player ⟕ Played))$

B) $\Pi_{Name}(\sigma_{Team=`Seahawks`}(Player)) - \Pi_{Name}(\sigma_{Team=`Seahawks`}(Player \bowtie Played))$

C) $\Pi_{Name}(\sigma_{Team=`Seahawks`}(Player \ltimes Played))$

D) $\Pi_{Name}(\sigma_{Team=`Seahawks` \wedge Date = null}(Player ⟖ Played))$

(SIMPLE) DATA DEFINITION WITH SQL

Data types:

- character (n), char (n)
- character varying (n), varchar (n)
- numeric (p,s), integer
- blob or raw for large binaries
- clob for large string values

Create Tables:

```
create table Professor
        (Person-ID    integer not null,
         Name         varchar (30) not null
         Level        varchar (2) default AP);
```

DDL (CTD.)

Delete a Table:

```
drop table Professor;
```

Modify the structure of a Table:

```
alter table Professor add column(age integer);
```

Management of Indexes (Performance tuning):

```
create index myIndex on Professor(name, age);
drop index myIndex;
```

UPDATES (DML)

Insert Tuples

```
insert into Student (Student-ID, Name)
    values (28121, `Archimedes‘);
```

```
insert into attends
    select Student-ID, Course-ID
    from Student, Lecture
    where Title= `Logic‘ ;
```

Student		
Student-ID	Name	Semester
29120	Theophrastos	2
29555	Feuerbach	2
28121	Archimedes	-

Null

SEQUENCE TYPES (AUTOMATIC INCREMENT FOR SURROGATES)

```
create sequence Person-ID_seq increment by 1 start with 1;
insert into Professor(Person-ID, Name)
        values(Person-ID_seq.nextval, „Roscoe“);
```

Syntax is vendor dependent

- E.g., AUTO-INCREMENT Option in MySQL
- Syntax above was standardized in SQL 2003

UPDATES (CTD.)

Delete tuples

delete Student

where Semester > 13;

Update tuples

update Student

set Semester = Semester + 1;

QUERIES

select	Person-ID, Name
from	Professor
where	Level = ´FP´;

Person-ID	Level	Name	Room
2125	FP	Ugur	303
2126	FP	Stan	345
2165	AP	Tim	335
2136	FP	Curie	401
2137	AP	Jeff	507

Person-ID	Name
2125	Ugur
2126	Stan
2136	Curie

$$\Pi_{\text{Person-ID, Name}} (\sigma_{\text{Level=`FP`}} (\text{Professor}))$$

QUERIES: SORTING

```
select Person-ID, Name, Level
from  Professor
order by Level desc, Name asc;
```

Person-ID	Name	Level
2136	Curie	FP
2137	Jeff	FP
2126	Stan	FP
2125	Ugur	FP
2134	Augustinus	AP
2127	Kopernikus	AP
2133	Popper	AP

CLICKER QUESTION: ARE THE FOLLOWING QUERIES EQUIVALENT?

select Level

from Professor

Π_{Level} (Professor)

Person-ID	Name	Level
2136	Curie	FP
2137	Jeff	FP
2126	Stan	FP
2125	Ugur	FP
2134	Augustinus	AP
2127	Kopernikus	AP
2133	Popper	AP

Answer:

(1) Yes

(2) No

DUPLICATE ELIMINATION

select distinct Level

from Professor

Level
AP
FP

QUERIES: JOINS

Who teaches ML?

```
select Name
from Professor, Lecture
where Person-ID = ProfID and Title = `ML‘ ;
```

$$\Pi_{\text{Name}}(\sigma_{\text{Person-Id=Prof-ID} \wedge \text{Title=`ML`}}(\text{Professor} \times \text{Lecture}))$$

Renamed Lecture.Person-ID to Prof-ID
Will show later how this can be done as part of a query.

JOINS

Person-ID	Name	Level	Room
2165	Ugur	FP	226
2166	Stan	FP	232
...	...	...	...
2200	Jeff	FP	7

CID	Title	CP	Prof-ID
5001	Foundation	4	2137
5041	German	4	2125
...	...	...	...
5049	ML	2	2125
...	...	...	...
4630	Vision	4	2137

Professor x Lecture

PID	Name	Level	Room	CID	Title	CP	ProfID
2125	Ugur	FP	226	5001	Foundation	4	2137
1225	Ugur	FP	226	5041	German	4	2125
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

PID	Name	Level	Room	CID	Title	CP	ProfID
2125	Ugur	FP	226	5001	Foundation	4	2137
1225	Ugur	FP	226	5041	German	4	2125
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2125	Ugur	FP	226	5049	ML	2	2125
⋮	⋮		⋮	⋮	⋮	⋮	⋮
2126	Stan	FP	232	5001	ML	4	2137
2126	Stan	FP	232	5041	German	4	2125
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2137	Jeff	FP	7	4630	Vision	4	2137

$\downarrow \sigma_{\text{Person-Id=Prof-ID} \wedge \text{Title=`ML}}$

Person -ID	Name	Level	Room	ID	Title	CP	ProfID
2125	Ugur	FP	226	5049	ML	2	2125

$\downarrow \Pi_{\text{Name}}$

Name
Ugur

SQL -> RELATIONAL ALGEBRA

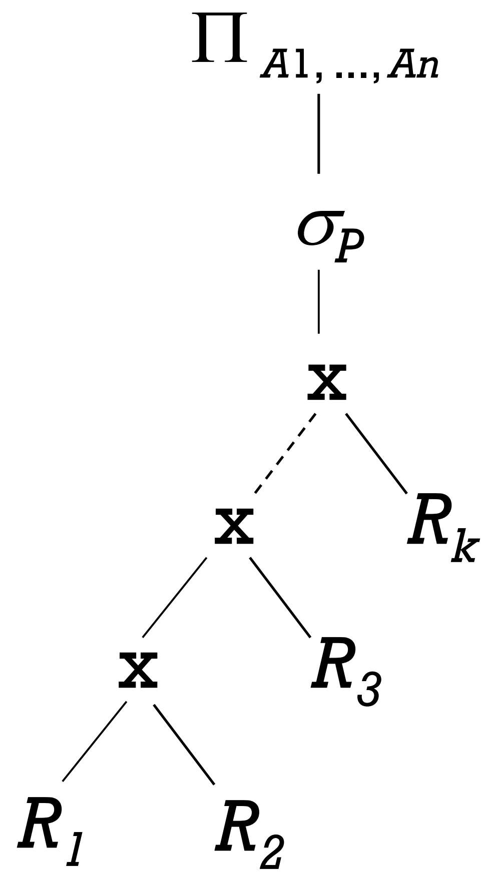
SQL

select $A_1, \ldots, A_n$

from $R_1, \ldots, R_k$

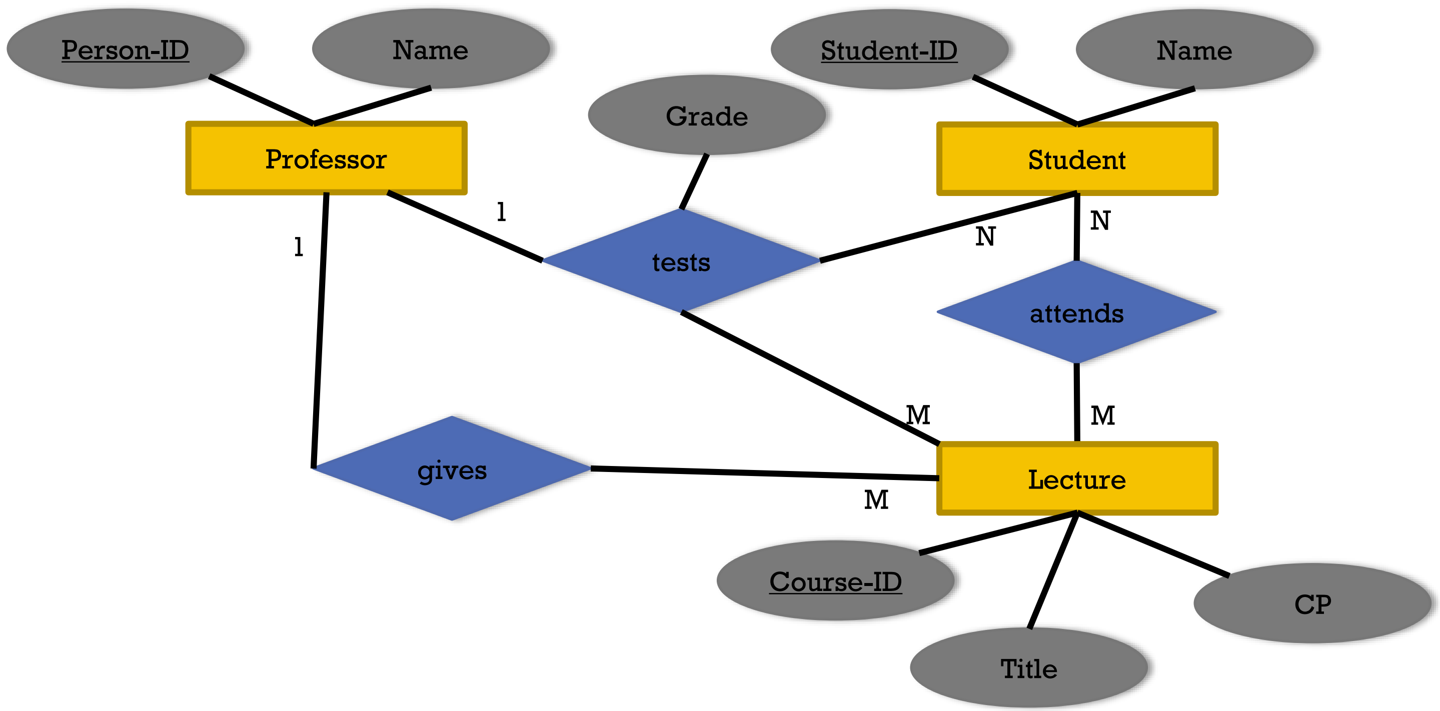
where P;

Relational Algebra



WHO ATTENDS WHICH LECTURE?

Professor(Person-ID:integer, Name:string)
Student(Student-ID:integer, Name:string)
Lecture(Course-ID:string, Title:string, CP:float)
Gives(Person-ID:integer, Course-ID:string)
Attends(Student-ID:integer, Course-ID:string)
Tests(Student-ID:integer, Course-ID:string, Person-ID:integer, , Grade:string)



JOINS AND TUPLE VARIABLES

Equivalent queries: Who attends which lecture?

```
select Name, Title
from Student, attends, Lecture
where Student.Student-ID = attends.Student-ID and
     attends.Course-ID = Lecture. Course-ID;
```

In Relational Algebra: ρ_l (Lecture)

```
select s.Name, l.Title
from Student s, attends a, Lecture l
where s.Student-ID = a.Student-ID and
    a.Course-ID = l.Course-ID;
```

RENAME OF ATTRIBUTES

Give title and professor of all lectures?

```
select Title, Person-ID as ProfID
from Lecture;
```

SET OPERATIONS

Union, Intersect, Minus

```
( select Name
  from Assistant )
union
( select Name
  from Professor);
```

GROUPING AGGREGATION

GROUPING, AGGREGATION

Aggregate functions: avg, max, min, count, sum

```
select avg (Semester)
    from Student;
```

```
select Person-ID, sum (CP) as load
    from Lecture
    group by Person-ID;
```

```
select p.Person-ID, Name, sum (CP)
    from Lecture l, Professor p
    where l.Person-ID= p.Person-ID and level = ´FP´
    group by p.Person-ID, Name
    having avg (CP) >= 3;
```

IMPERATIVE PROCESSING IN SQL

Step 1:

from Lecture l, Professor p

where l.Person-ID= p.Person-ID **and** level = ´FP

Step 2:

group by p.Person-ID, Name

Step 3:

having avg (CP) >= 3;

Step 4:

select p.Person-ID, Name, **sum** (CP)

GROUP BY

Lecture x Professor							
Nr	Title	CP	Person-ID	Perso n-ID	Name	Level	Room
5001	Foundation	4	2137	2125	Ugur	FP	226
5041	German	4	2125	2125	Ugur	FP	226
...	...	...	...	...	...	...	...
4630	Vision	4	2137	2137	Jeff	AP	7

$\Downarrow$ σ

(where l.Person-ID= p.Person-ID and level = ´FP`)

Nr	Title	CP	Person-ID	Person-ID	Name	Level	Room
5001	Foundation	4	2137	2137	Jeff	FP	7
5041	German	4	2125	2125	Ugur	FP	226
5043	Cyper Stuff	3	2126	2126	Stan	FP	232
5049	ML	2	2125	2125	Ugur	FP	226
4052	Logik	4	2125	2125	Ugur	FP	226
5052	Robotics	3	2126	2126	Stan	FP	232
5216	Adv. German	2	2126	2126	Stan	FP	232
4630	Vision	4	2137	2137	Jeff	FP	7

⇓ **group by** p.Person-ID, Name

⇓ **group by** p.Person-ID, Name

Nr	Title	CP	Person-ID	Person-ID	Name	Level	Room
5041	German	4	2125	2125	Ugur	FP	226
5049	ML	2	2125	2125	Ugur	FP	226
4052	Logik	4	2125	2125	Ugur	FP	226
5043	Cyper Stuff	3	2126	2126	Stan	FP	232
5052	Robotics.	3	2126	2126	Stan	FP	232
5216	Adv. German	2	2126	2126	Stan	FP	232
5001	Foundation	4	2137	2137	Jeff	FP	7
4630	Vision	4	2137	2137	Jeff	FP	7

⇓ **Having** (CP) >= 3

Nr	Title	CP	Person-ID	Person-ID	Name	Level	Room
5041	German	4	2125	2125	Ugur	FP	226
5049	ML	2	2125	2125	Ugur	FP	226
4052	Logik	4	2125	2125	Ugur	FP	226
5001	Foundation	4	2137	2137	Jeff	FP	7
4630	Vision	4	2137	2137	Jeff	FP	7

Nr	Title	CP	Person-ID	Person-ID	Name	Level	Room
5041	German	4	2125	2125	Ugur	FP	226
5049	ML	2	2125	2125	Ugur	FP	226
4052	Logik	4	2125	2125	Ugur	FP	226
5001	Foundation	4	2137	2137	Jeff	FP	7
4630	Vision	4	2137	2137	Jeff	FP	7

$\Downarrow$ Π⚐♏❒⬧□■📫✋👎📭 ☠♋❍♏ 🕮
sum(select)

Person-ID	Name	**sum** (CP)
2125	Ugur	10
2137	Jeff	8

Question: Why do we need to group-by on Person-ID and Name?

CLICKER

Which of the following is the correct order of keywords for SQL SELECT statements?

A) From, Where, Select, Group-By, Having
B) Select, From, Where, Group-by, Having
C) Having, Select, From, Where, Group-by
D) From, Where, Group-By, Having, Select

What is the execution order?

A) From, Where, Select, Group-By, Having
B) Select, From, Where, Group-by, Having
C) Having, Select, From, Where, Group-by
D) From, Where, Group-By, Having, Select

SUBQUERIES

EXISTENTIAL QUANTIFICATION

EXISTS SUBQUERIES

```
select p.Name
from Professor p
where not exists ( select *
                   from Lecture l
                   where l.Person-ID = p.Person-ID );
```

CORRELATED SUB-QUERIES

Correlation

```
select p.Name
from Professor p
where not exists ( select *
                   from Lecture l
                   where l.Person-ID = p.Person-ID );
```

CORRELATED SUB-QUERIES

Correlation

```
select p.Name
from Professor p
where not exists ( select *
                   from Lecture l
                   where l.Person-ID = p.Person-ID );
```

```
For every p in Professor
    where not exist ( // like a check for empty set
        select * from lecture where l.Person-ID = p.Person-ID
        )
    emit p.Name
```

UNCORRELATED SUB-QUERY

```
select Name
from Professor
where Person-ID not in ( select Person-ID
                         from Lecture);
```

What is better? Correlated or uncorrelated?

SUB-QUERIES WITH **ALL**

```
select Name
from Student
where Semester >= all ( select Semester
                        from Student);
```

SUBQUERIES IN SELECT, FROM

```
select Person-ID, Name, ( select sum (CP) as load
                          from Lecture l
                          where p.Person-ID=l.Person-ID )
from Professor p;
```

```
select p.Person-ID, Name, pl.load

from Professor p, (   select Person-ID, sum (CP) as load
                      from Lecture
                      group by Person-ID ) pl

where p.Person-ID = pl.Person-ID;
```

Is this better than the simple Group By Query from before?

QUERY REWRITE

Two equivalent join queries: Which is better?

```
select *
from Assistant a
where exists
        ( select *
          from Professor p
          where a.Boss = p.Person-ID and p.age < a.age);
```

```
select a.*
from Assistant a, Professor p
where a.Boss=p.Person-ID and p.age < a.age;
```

ARE THESE TWO QUERIES EQUIVALENT?

```
select count (*)
from Student
where Semester < 13 or
Semester > =13;
```

```
select count (*)
from Student;
```

(A) No (B) Yes

NULL VALUES

NULL VALUES (NULL = UNKNOWN)

Are these two queries equivalent?

```
select count (*)
from Student
where Semester < 13 or
Semester > =13;
```

```
select count (*)
from Student;
```

WORKING WITH NULL VALUES

Arithmetics: Propagate null: If an operand is null, the result is null.

- null + 1 -> null
- null * 0 -> null

Comparisons: New Boolean value unknown. All comparisons that involve a null value, evaluate to unknown.

- null = null -> unknown
- null < 13 -> unknown
- null > null -> unknown

Logic: Boolean operators are evaluated using the following tables (next slide):

LOGICAL OPERATIONS

p	NOT p
TRUE	FALSE
FALSE	TRUE
Unknown	Unknown

p	q	p OR q	p AND q	p = q
TRUE	TRUE	TRUE	TRUE	TRUE
TRUE	FALSE	TRUE	FALSE	FALSE
TRUE	Unknown	TRUE	Unknown	Unknown
FALSE	TRUE	TRUE	FALSE	FALSE
FALSE	FALSE	FALSE	FALSE	TRUE
FALSE	Unknown	Unknown	FALSE	Unknown
Unknown	TRUE	TRUE	Unknown	Unknown
Unknown	FALSE	Unknown	FALSE	Unknown
Unknown	Unknown	Unknown	Unknown	Unknown

where: Only tuples which evaluate to **true** are part of the query result. (**unknown** and **false** are equivalent here):

```
select count (*)
from Student
where Semester < 13 or Semester > =13;
```

group by: If exists, then there is a group for **null**.

```
select count (*)
from Student
group by Semester;
```

Predicates with null:

```
select count (*) from Student
where Semester is null;
```

CLICKER

SELECT count(*) AS total FROM orders;

Count(*)
100

SELECT count(*) AS cust_123_total FROM orders
WHERE customer_id = '123';

Count(*)
15

Given the above query results, what will be the result of the query below?
SELECT count(*) AS cust_not_123_total
FROM orders
WHERE customer_id <> '123'

A) 85 B) 100 C) Impossible to say

SYNTACTIC SUGAR

```
select *
from Student
where Semester > = 1 and Semester < = 6;
```

```
select *
from Student
where Semester between 1 and 6;
```

```
select *
from Student
where Semester in (2,4,6);
```

COMPARISONS WITH **LIKE**

"%„ represents any sequence of characters (0 to n)

"_„ represents exactly one character

N.B.: For comparisons with = , % and _ are normal chars.

select *

from Student

where Name **like** ´Tim%´;

select distinct Name

from Lecture l, attends a, Student s

where s.Student-ID = a.Student-ID **and** a.CID = l.CID
and l.Title like ´%science%´;

JOINS IN SQL-92

- **cross join**: Cartesian product
- **natural join**
- **join** or **inner join**
- **left, right** or **full outer join**
- (union join: not discussed here)

```
select *
from R1, R2
where R1.A = R2.B;
```

```
select *
from R1 join R2 on R1.A = R2.B;
```

LEFT OUTER JOINS

```
select p.Person-ID, p.Name, t.Person-ID, t.Grade, t.Student-ID,
s.Student-ID, s.Name
from Professor p left outer join
          (tests t left outer join Student s
          on t.Student-ID= s.Student-ID)
          on p.Person-ID=t.Person-ID;
```

Person-ID	p.Name	t.Person-ID	t.Grade	t.Student-ID	s.Student-ID	s.Name
2126	Stan	2126	1	28106	28106	Carnap
2125	Ugur	2125	2	25403	25403	Jonas
2137	Jeff	2137	2	27550	27550	Schopen-hauer
2136	Curie	-	-	-	-	-
⋮	⋮	⋮	⋮	⋮	⋮	⋮

RIGHT OUTER JOINS

select p.Person-ID, p.Name, t.Person-ID, t.Grade, t.Student-ID, s.Student-ID, s.Name
from Professor p **right outer join**
(tests t **right outer join** Student s **on**
t.Student-ID= s.Student-ID)
on p.Person-ID=t.Person-ID;

Person-ID	p.Name	t.Person-ID	t.Grade	t.Student-ID	s.Student-ID	s.Name
2126	Stan	2126	1	28106	28106	Carnap
2125	Ugur	2125	2	25403	25403	Jonas
2137	Jeff	2137	2	27550	27550	Schopen-hauer
-	-	-	-	-	26120	Fichte
⋮	⋮	⋮	⋮	⋮	⋮	⋮

FULL OUTER JOINS

```
select p.Person-ID, p.Name, t.Person-ID, t.Grade, t.Student-ID,
s.Student-ID, s.Name
from Professor p full outer join
        (tests t full outer join Student s on
        t.Student-ID= s.Student-ID)
        on p.Person-ID=t.Person-ID;
```

p.Person-ID	p.Name	t.Person-ID	t.Grade	t.Student-ID	s.Student-ID	s.Name
2126	Stan	2126	1	28106	28106	Carnap
2125	Ugur	2125	2	25403	25403	Jonas
2137	Jeff	2137	2	27550	27550	Schopen-hauer
-	-	-	-	-	26120	Fichte
2136	Curie	-	-	-	-	-