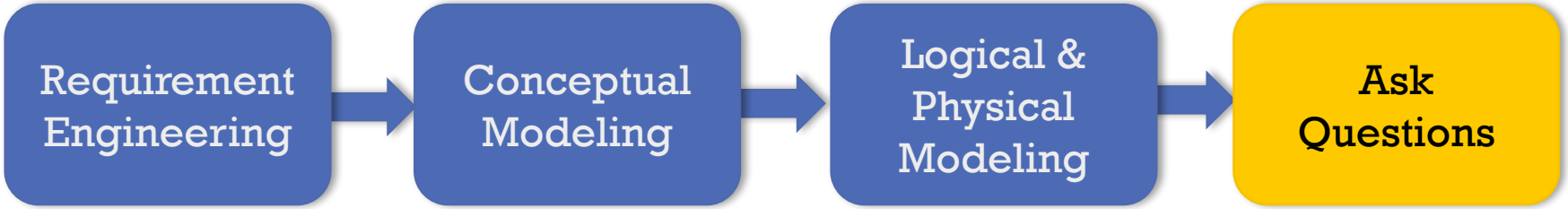
SQL (SEQUEL)

INTRODUCTION TO DATA SCIENCE

CARSTEN BINNIG
BROWN UNIVERSITY

DATABASES FOR DATA SCIENTIST



Book of duty

Conceptual Design (ER)

- Logical design (relational schema)
- Physical design (index, hints)

- Relational Algebra
- SQL

SQL

SQL: Structured Query Language

- **DDL = Data Definition Language**
- **DML = Data Manipulation + Query Language**

SQL is not based on sets (which are duplicate free); it is based on bags (i.e., w duplicates)

Standards defined: SQL 92, …, SQL 2011

- **Core defined since SQL 92**
- **Recent extensions: XML, JSON, Object Relational, …**

SQL (SEQUEL)

DATA DEFINITION LANGUAGE (DDL)

DDL: 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: CREATE TABLE

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

```
CREATE TABLE Student (
  Student-ID INT,
  Name VARCHAR(45));

CREATE TABLE Professor (
  Person-ID INT,
  Name VARCHAR(45));
```

```
CREATE TABLE Lecture(
    Course-ID INT,
    Title VARCHAR(45),
    CP REAL,
    Person-ID INT);
```

DDL: PRIMARY KEYS

primary key (A_1, …, A_n)

Example: Declare *ID* as the primary key for *instructor OR in case of two attributes as keys:*

```
CREATE TABLE Attends (
    Student-ID INT,
    Course-ID INT,
    PRIMARY KEY (Student-ID, Course-ID));
```

primary key declaration on an attribute automatically ensures **not null**

DDL: SINGLE ATTRIBUTE KEY

```
CREATE TABLE course (
        course_id  INTEGER PRIMARY KEY,
        title VARCHAR(50),
        cp INTEGER);
```

Primary key **declaration can be combined with attribute declaration** as shown above

DDL: FOREIGN KEYS

```
CREATE TABLE `Attends` (
  `Student_ID` INT NOT NULL,
  `Course-ID` INT NOT NULL,
  PRIMARY KEY (`Student_ID`, `Course-ID`),
  CONSTRAINT `fk_attend_Student`
    FOREIGN KEY (`Student_ID`)
    REFERENCES `Student` (`ID`);
  CONSTRAINT `fk_attend_lecture`
    FOREIGN KEY (`Course-ID`)
    REFERENCES `Lecture` (`Course-ID`);
```

DDL: FOREIGN KEYS (CNT.)

Updates on the table to which the FK refers to can have different effects

- **set null:** foreign key values are set to NULL if referred value is updated / deleted
- **cascade:** foreign key are updated / deleted if referred value is updated / deleted
- **restrict:** referred value can not be updated / deleted if a foreign key with the given value exists

DDL: FOREIGN KEYS

```
CREATE TABLE `Attends` (
  `Student_ID` INT NOT NULL,
  `Course-ID` INT NOT NULL,
  PRIMARY KEY (`Student_ID`, `Course-ID`),
  CONSTRAINT `fk_attend_Student`
    FOREIGN KEY (`Student_ID`)
    REFERENCES `Student` (`ID`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `fk_attend_lecture`
    FOREIGN KEY (`Lecture_Course-ID`)
    REFERENCES `Lecture` (`Course-ID`)
    ON DELETE RESTRICT
    ON UPDATE RESTRICT;
```

OTHER INTEGRITY CONSTRAINTS

Not Null: Value must not be set to NULL

Default-Values: If user provides no value

Check-Constraints: e.g., age≥0 and age ≤ 200

Unique-Constraints: any other candidate key

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;
```

SQL (SEQUEL)

DATA MANIPULATOIN LANGUAGE (DML)

Student		
Student-ID	Name	Year
29120	Sally	2
29555	Emily	2

Insert a new tuple?

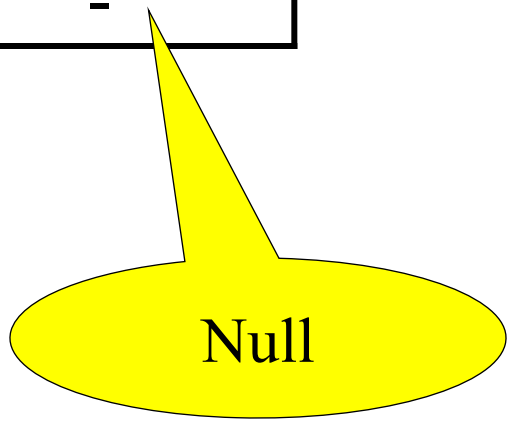
DML: INSERTS

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	Year
29120	Sally	2
29555	Emily	2
28121	Archimedes	-



DETOUR DDL: 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

DML: DELETE AND UPDATES

Delete tuples

```
delete Student
where Year > 13;
```

Update tuples

```
update Student
set Year = Year + 1;
```

DML: QUERIES

```
select    Person-ID, Name
from      Professor
where     Level = 3;
```

Person-ID	Level	Name	Room
2125	4	Ugur	303
2126	3	Stan	345
2165	3	Tim	335
2136	3	Curie	401
2137	4	Stephanie	507

Person-ID	Name
2165	Tim
2126	Stan
2136	Curie

$$\Pi_{\text{Person-ID, Name}} (\sigma_{\text{Level=3}} (\text{Professor}))$$

QUERIES: SORTING

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

Person-ID	Level	Name	Room
2125	4	Ugur	303
2126	3	Stan	345
2165	3	Tim	335
2136	3	Curie	401
2137	4	Stephanie	507

Person-ID	Level	Name
2136	3	Curie
2126	3	Stan
2165	3	Tim
2137	4	Stephanie
2125	4	Ugur

CLICKER QUESTION: ARE THE FOLLOWING QUERIES EQUIVALENT?

select Level

from Professor

Π_{Level} (Professor)

Answer:

(1) Yes

(2) No

Person-ID	Level	Name	Room
2125	4	Ugur	303
2126	3	Stan	345
2165	3	Tim	335
2136	3	Curie	401
2137	4	Stephanie	507

CLICKER QUESTION: ARE THE FOLLOWING QUERIES EQUIVALENT?

select Level

from Professor

Π_{Level} (Professor)

Answer:

(1) Yes

(2) No

… because of distinct values.

Person-ID	Level	Name	Room
2125	4	Ugur	303
2126	3	Stan	345
2165	3	Tim	335
2136	3	Curie	401
2137	4	Stephanie	507

DUPLICATE ELIMINATION

select distinct Level

from Professor

Level
3
4

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
2125	Ugur	4	444
2126	Stan	3	333
…	…	…	…
2137	Stephanie	4	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	4	444	5001	Foundation	4	2137
1225	Ugur	4	444	5041	German	4	2125
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

PID	Name	Level	Room	CID	Title	CP	ProfID
2125	Ugur	4	444	5001	Foundation	4	2137
1225	Ugur	4	444	5041	German	4	2125
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2125	**Ugur**	**4**	**444**	**5049**	**ML**	**2**	**2125**
⋮	⋮		⋮	⋮	⋮	⋮	⋮
2126	Stan	3	333	5001	ML	4	2137
2126	Stan	3	333	5041	German	4	2125
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2137	Stephanie	4	7	4630	Vision	4	2137

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

Person-ID	Name	Level	Room	ID	Title	CP	ProfID
2125	Ugur	4	444	5049	ML	2	2125

↓ Π_{Name}

Name
Ugur

SQL -> RELATIONAL ALGEBRA

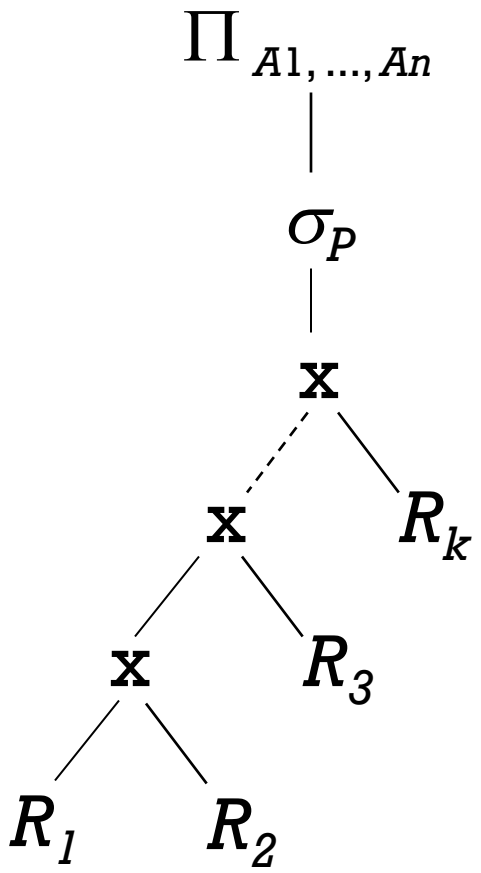
SQL

select $A_1, \ldots, A_n$

from $R_1, \ldots, R_k$

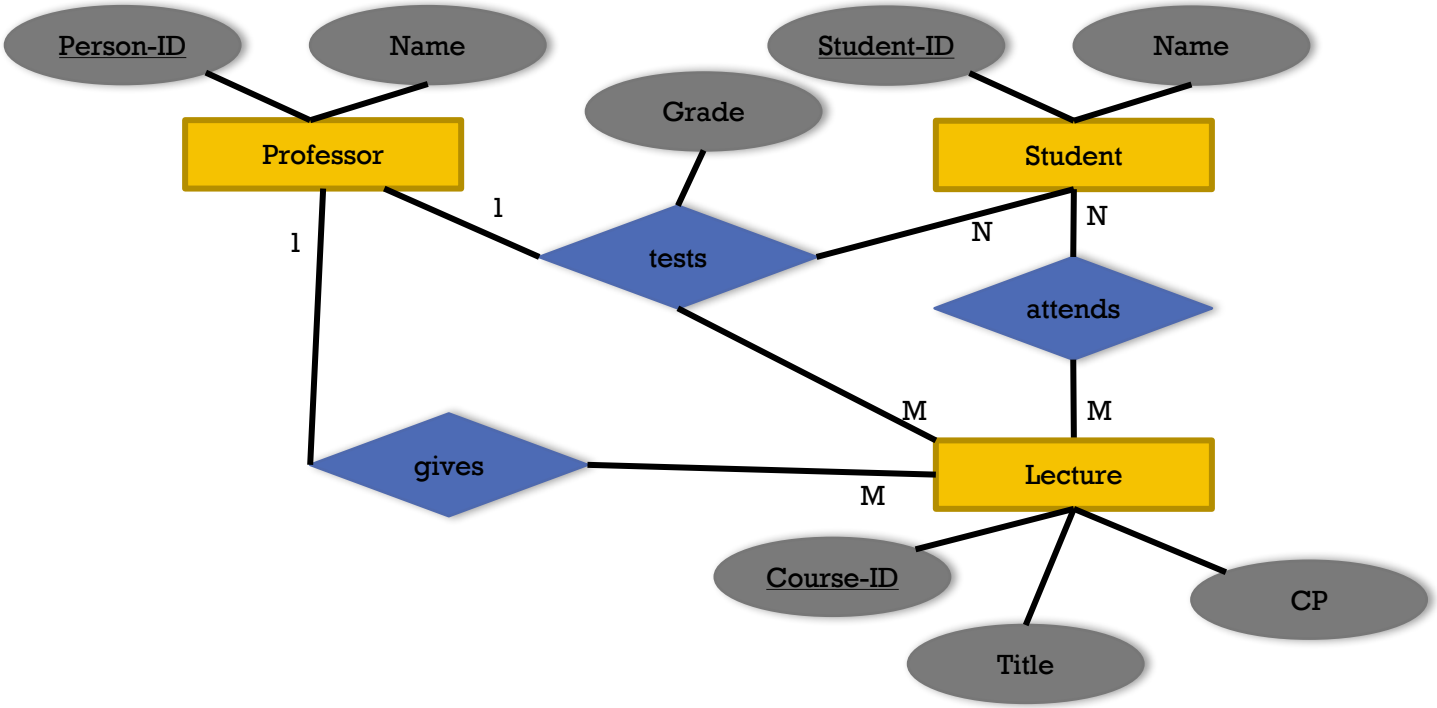
where P;

Relational Algebra



WHO ATTENDS WHICH LECTURE?

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

```
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;
```

In Relational Algebra:
ρ_l (Lecture)

RENAME OF ATTRIBUTES

What is the title and professor of all lectures?

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

SET OPERATIONS

Union (All), Intersect, Minus

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

BUT No Division

GROUPING AGGREGATION

GROUPING, AGGREGATION

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

```
select avg (Year)
    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 OF SQL

Step 1:

from Lecture l, Professor p

where l.Person-ID= p.Person-ID

Step 2:

group by p.Person-ID, Name

Step 3:

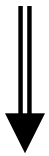
having avg (CP) >= 3;

Step 4:

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

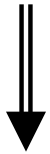
GROUP BY

from Lecture x Professor						
Nr	Title	CP	Person-ID	Person-ID	Name	Room
5001	Foundation	4	2137	2125	Ugur	444
5041	German	4	2125	2125	Ugur	444
...	...	...	...	...	...	...
4630	Vision	4	2137	2137	Stephanie	7



ωηερε l.Person-ID= p.Person-ID

Nr	Title	CP	Person-ID	Person-ID	Name	Room
5001	Foundation	4	2137	2137	Stephanie	7
5041	German	4	2125	2125	Ugur	444
5043	Cyper Stuff	3	2126	2126	Stan	333
5049	ML	2	2125	2125	Ugur	444
4052	Logik	4	2125	2125	Ugur	444
5052	Robotics	3	2126	2126	Stan	333
5216	Adv. German	2	2126	2126	Stan	333
4630	Vision	4	2137	2137	Stephanie	7



group by p.Person-ID, Name

group by p.Person-ID, Name

Nr	Title	CP	Person-ID	Person-ID	Name	Room
5041	German	4	2125	2125	Ugur	444
5049	ML	2	2125	2125	Ugur	444
4052	Logik	4	2125	2125	Ugur	444
5043	Cyper Stuff	3	2126	2126	Stan	333
5052	Robotics.	3	2126	2126	Stan	333
5216	Adv. German	2	2126	2126	Stan	333
5001	Foundation	4	2137	2137	Stephanie	7
4630	Vision	4	2137	2137	Stephanie	7

having AVG(CP) >= 3

Nr	Title	CP	Person-ID	Person-ID	Name	Room
5041	German	4	2125	2125	Ugur	444
5049	ML	2	2125	2125	Ugur	444
4052	Logik	4	2125	2125	Ugur	444
5001	Foundation	4	2137	2137	Stephanie	7
4630	Vision	4	2137	2137	Stephanie	7

Nr	Title	CP	Person-ID	Person-ID	Name	Room
5041	German	4	2125	2125	Ugur	444
5049	ML	2	2125	2125	Ugur	444
4052	Logik	4	2125	2125	Ugur	444
5001	Foundation	4	2137	2137	Stephanie	7
4630	Vision	4	2137	2137	Stephanie	7

$\Downarrow \Sigma\text{ΕΛΧΕΤ}_{\text{Περσον–ΙΔ, Ναμε, }}$ **sum(CP)**

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

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

CLICKER QUESTIONS

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 canonical 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

CLICKER QUESTIONS

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 canonical 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

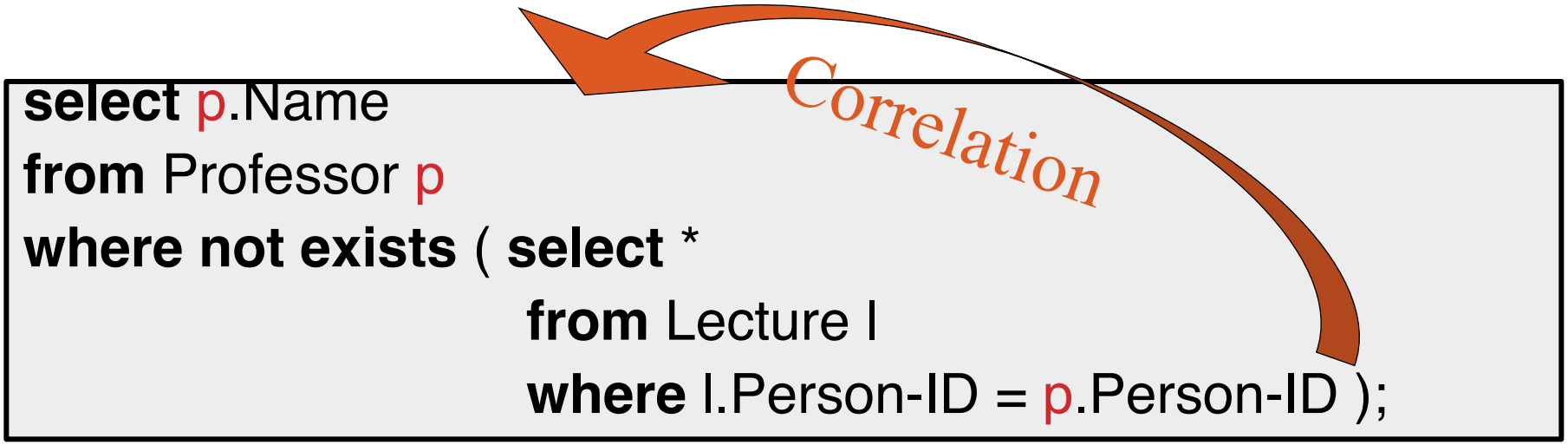
SUBQUERIES

EXISTENTIAL QUANTIFICATION

EXISTS SUBQUERIES

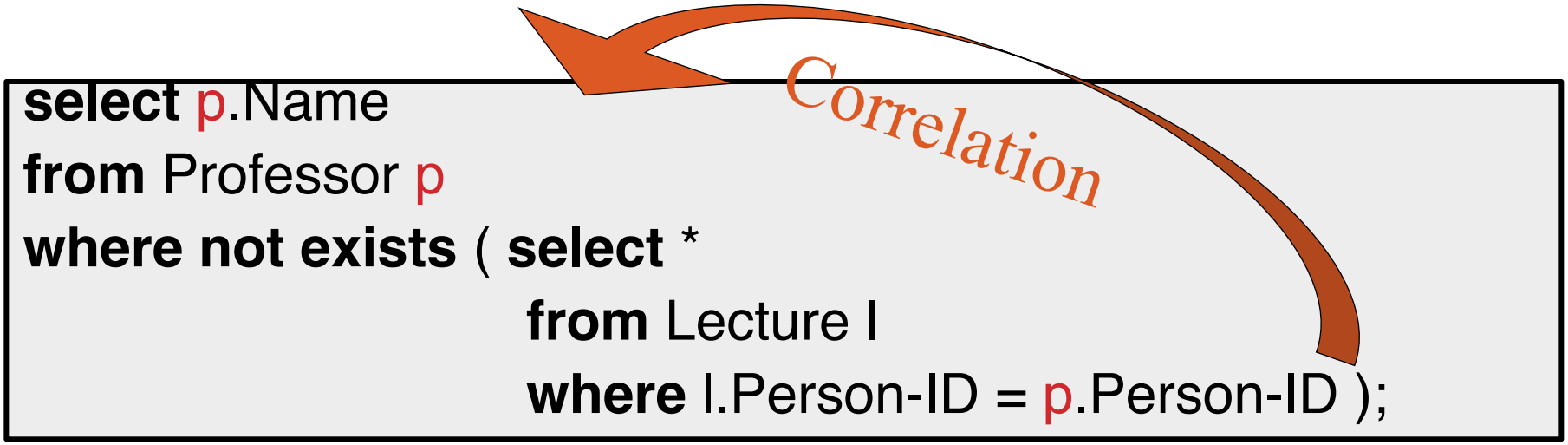
```
select p.Name
from Professor p
where not exists ( select *
                   from Lecture l
                   where l.Person-ID = p.Person-ID );
```

CORRELATED SUB-QUERIES




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

CORRELATED SUB-QUERIES



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);
```

SUB-QUERIES WITH **ALL**

```
select Name
from Student
where Year >= all ( select Year
                    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;
```

Which query is correlated here?

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;
```

CLICKER QUESTION

Are these two queries equivalent?

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

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

(A) No (B) Yes

CLICKER QUESTION

Are these two queries equivalent?

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

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

WORKING WITH NULL VALUES

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

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

Comparisons: 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):

NULL & 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
FALSE	TRUE	TRUE	FALSE	FALSE
FALSE	FALSE	FALSE	FALSE	TRUE

What if we add unkown?

NULL & 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
FALSE	TRUE	TRUE	FALSE	FALSE
FALSE	FALSE	FALSE	FALSE	TRUE
TRUE	Unknown	TRUE	Unknown	Unknown
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 Year < 13 or Year > =13;
```

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

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

Predicates with null:

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

NULL & AGGREGATION

count(att): NULL is ignored

sum(att): NULL is ignored

avg(att): results from SUM and COUNT

min(att) and max(att): NULL is ignored

Exception (sum, avg, min, max): result is also NULL if NULL is only value in column

CLICKER QUESTION I

SELECT count(*) FROM orders;

Count(*)
100

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

Count(*)
15

Given the above query results, what will be the result of the query below?

SELECT count(*)
FROM orders
WHERE customer_id != '123'

A) 85 B) 100 C) Impossible to say

CLICKER QUESTION I

SELECT count(*) FROM orders;

Count(*)
100

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

Count(*)
15

Given the above query results, what will be the result of the query below?

SELECT count(*)
FROM orders
WHERE customer_id != '123'

A) 85 B) 100 C) Impossible to say

(since we do not know how many are null)

CLICKER QUESTION II

Given the following tables

Runners

id	name
1	John
2	Tim
3	Alice
4	Lisa

Races

Event_id	Event	Winner_id
1	Tough mudder	2
2	500m	3
3	Cross-country	2
4	Triathlon	null

What will be the result of the query:

```
SELECT *
FROM runners
WHERE id NOT IN (SELECT winner_id FROM races)
```

A) 1
B) Empty set
C) (1,4)

CLICKER QUESTION II

Given the following tables

Runners

id	name
1	John
2	Tim
3	Alice
4	Lisa

Races

Event_id	Event	Winner_id
1	Tough mudder	2
2	500m	3
3	Cross-country	2
4	Triathlon	null

What will be the result of the query:

```
SELECT *
FROM runners
WHERE id NOT IN (SELECT winner_id FROM races)
```

A) 1
B) Empty set
C) (1,4)

ID NOT IN (2,3,null) is equal to ID!=2 AND ID!=3 and ID!=NULL

SYNTACTIC SUGAR

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

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

```
select *
from Student
where Year 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.Course-ID = 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**

```
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 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	Stephanie	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	Stephanie	2137	2	27550	27550	Schopen-hauer
-	-	-	-	-	26120	Fichte
⋮	⋮	⋮	⋮	⋮	⋮	⋮

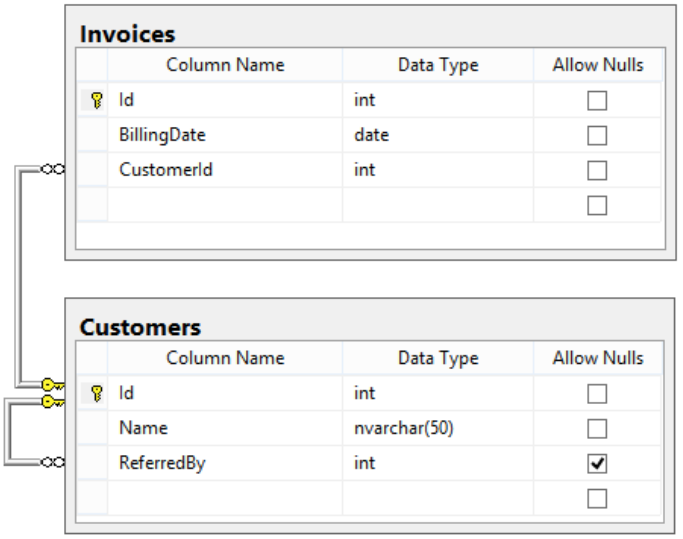
FULL OUTER JOINS

```
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	Stephanie	2137	2	27550	27550	Schopen-hauer
-	-	-	-	-	26120	Fichte
2136	Curie	-	-	-	-	-

CLICKER QUESTION III

You have your first day at TheShop LLC. Your first task is to write a SQL query to return a list of all the invoices. For each invoice, you want the Invoice ID, the billing date, the customer's name, and the name of the customer who referred that customer (if any). The list should be ordered by billing date.



What is the correct SQL statement?

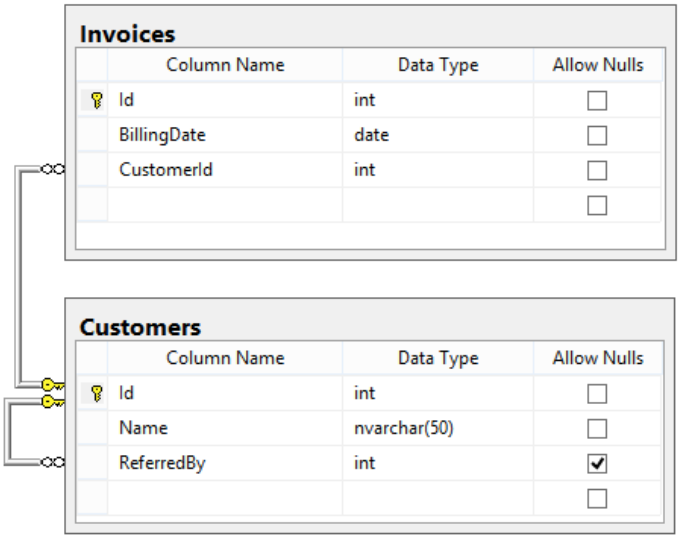
A)
```
SELECT i.Id, i.BillingDate, c.Name, r.Name AS ReferredByName
FROM (Invoices i JOIN Customers c ON i.CustomerId = c.Id)
        JOIN Customers r ON c.ReferredBy = r.Id
ORDER BY i.BillingDate;
```

B)
```
SELECT i.Id, i.BillingDate, c.Name, r.Name AS ReferredByName
FROM (Invoices I JOIN Customers c ON i.CustomerId = c.Id)
        LEFT JOIN Customers r ON c.ReferredBy = r.Id
ORDER BY i.BillingDate;
```

C)
```
SELECT i.Id, i.BillingDate, c.Name, r.Name AS ReferredByName
FROM (Invoices i LEFT JOIN Customers c ON i.CustomerId = c.Id)
        JOIN Customers r ON c.ReferredBy = r.Id
ORDER BY i.BillingDate;
```

CLICKER QUESTION III

You have your first day at TheShop LLC. Your first task is to write a SQL query to return a list of all the invoices. For each invoice, you want the Invoice ID, the billing date, the customer's name, and the name of the customer who referred that customer (if any). The list should be ordered by billing date.



What is the correct SQL statement?

A)

```
SELECT i.Id, i.BillingDate, c.Name, r.Name AS ReferredByName
FROM (Invoices i JOIN Customers c ON i.CustomerId = c.Id)
            JOIN Customers r ON c.ReferredBy = r.Id
ORDER BY i.BillingDate;
```

B)

```
SELECT i.Id, i.BillingDate, c.Name, r.Name AS ReferredByName
FROM (Invoices I JOIN Customers c ON i.CustomerId = c.Id)
            LEFT JOIN Customers r ON c.ReferredBy = r.Id
ORDER BY i.BillingDate;
```

C)

```
SELECT i.Id, i.BillingDate, c.Name, r.Name AS ReferredByName
FROM (Invoices i LEFT JOIN Customers c ON i.CustomerId = c.Id)
            JOIN Customers r ON c.ReferredBy = r.Id
ORDER BY i.BillingDate;
```

HOW DOES THE DB PROCESS A SQL QUERY?

SQL is the "WHAT" not the "HOW"

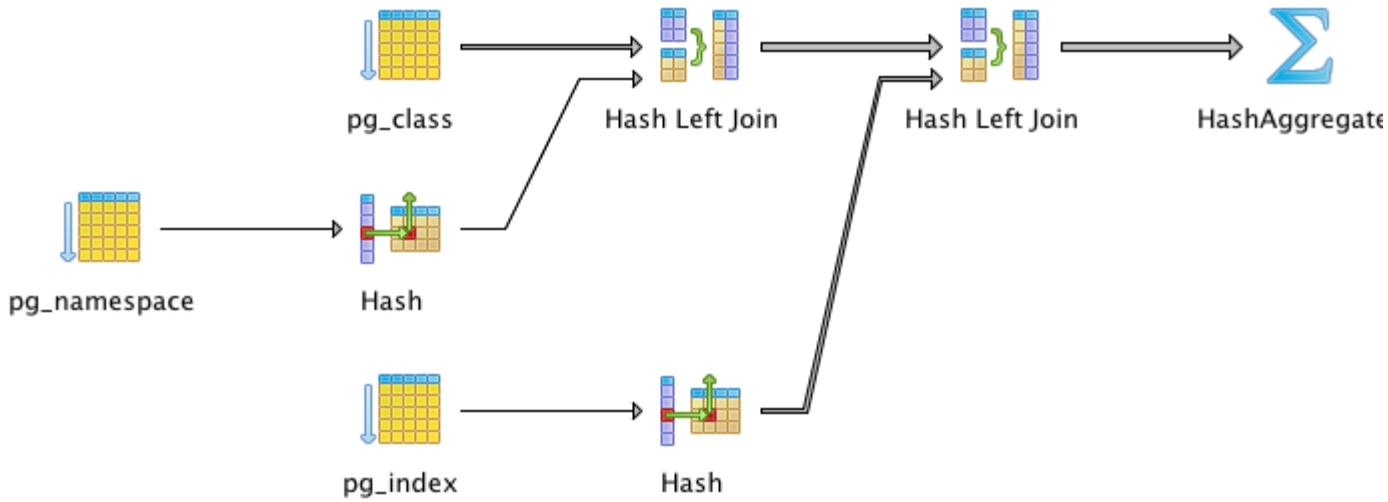
Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)

```
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = y.cid and
      x.price > 100 and z.city = 'Seattle'
```

It's clear WHAT we want, unclear HOW to get it

Exposing the Algebra: PostgreSQL

EXPLAIN SELECT ….



screenshot from pgAdmin3

SUMMARY

Data Definition Language

- **CREATE / DROP**
- **ALTER**

Data Manipulation Language

- **UPDATE / INSERT / DELETE**
- **SELECT**
  - Basic Select – From- Where
  - Joins
  - Group-by
  - Sub-Queries
  - NULL-handling