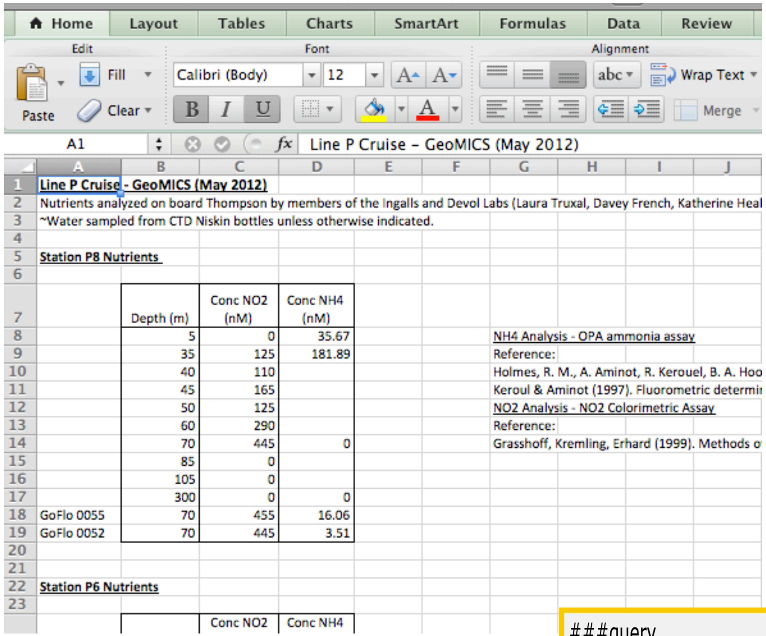
DATA MODELS AND DATABASES

BILL HOWE, PHD
DIRECTOR OF RESEARCH, SCALABLE DATA ANALYTICS
UNIVERSITY OF WASHINGTON ESCIENCE INSTITUTE

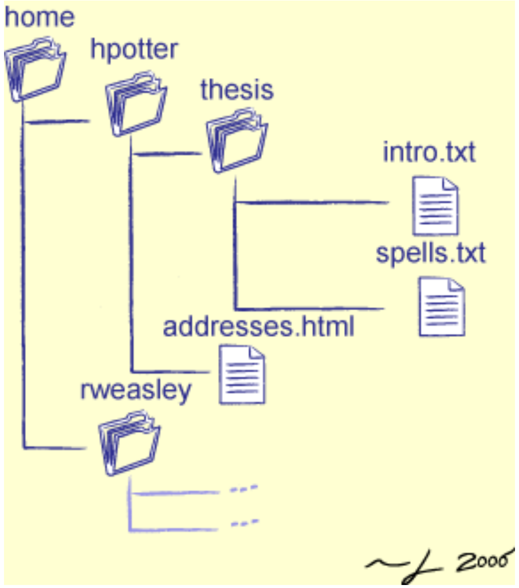
teaching
datascience
.org

HOW DO WE STORE DATA?

HOW DO WE STORE DATA?



	A	B	C	D	E	F	G
1	Line P Cruise - GeoMICS (May 2012)						
2	Nutrients analyzed on board Thompson by members of the Ingalls and Devol Labs (Laura Truxal, Davey French, Katherine Heal						
3	~Water sampled from CTD Niskin bottles unless otherwise indicated.						
4							
5	Station P8 Nutrients						
6							
7		Depth (m)	Conc NO2 (nM)	Conc NH4 (nM)			
8		5	0	35.67			NH4 Analysis - OPA ammonia assay
9		35	125	181.89			Reference:
10		40	110				Holmes, R. M., A. Aminot, R. Kerouel, B. A. Hoo
11		45	165				Keroul & Aminot (1997). Fluorometric determi
12		50	125				NO2 Analysis - NO2 Colorimetric Assay
13		60	290				Reference:
14		70	445	0			Grasshoff, Kremling, Erhard (1999). Methods o
15		85	0				
16		105	0				
17		300	0	0			
18	GoFlo 0055	70	455	16.06			
19	GoFlo 0052	70	445	3.51			
20							
21							
22	Station P6 Nutrients						
23							
			Conc NO2	Conc NH4			



What is the *data model*?

ANNOTATIONSUMMARY-COMBINEDORFANNOTATION16_Phaeo_genome

###query	length	COG hit #1	e-value #1	identity #1	score #1	hit length #1	description #1
chr_4[480001-580000].287	4500						
chr_4[560001-660000].1	3556						
chr_9[400001-500000].503	4211	COG4547	2.00E-04	19	44.6	620	Cobalamin biosynthesis protei
chr_9[320001-420000].548	2833	COG5406	2.00E-04	38	43.9	1001	Nucleosome binding factor SP
chr_27[320001-404298].20	3991	COG4547	5.00E-05	18	46.2	620	Cobalamin biosynthesis protei
chr_26[320001-420000].378	3963	COG5099	5.00E-05	17	46.2	777	RNA-binding protein of the Puf
chr_26[400001-441226].196	2949	COG5099	2.00E-04	17	43.9	777	RNA-binding protein of the Puf
chr_24[160001-260000].65	3542						
chr_5[720001-820000].339	3141	COG5099	4.00E-09	20	59.3	777	RNA-binding protein of the Puf
chr_9[160001-260000].243	3002	COG5077	1.00E-25	26	114	1089	Ubiquitin carboxyl-terminal hy
chr_12[720001-820000].86	2895	COG5032	2.00E-09	30	60.5	2105	Phosphatidylinositol kinase and
chr_12[800001-900000].109	1463	COG5032	1.00E-09	30	60.1	2105	Phosphatidylinositol kinase and
chr_11[1-100000].70	2886						
chr_11[80001-180000].100	1523						

WHAT IS A DATA MODEL?

Three components:

1. Structures
2. Constraints
3. Operations

EXAMPLES

1. **Structures**
   - rows and columns?
   - nodes and edges?
   - key-value pairs?
   - a sequence of bytes?
2. **Constraints**
   - all rows must have the same number of columns
   - all values in one column must have the same type
   - a child cannot have two parents
3. **Operations**
   - find the value of key x
   - find the rows where column "lastname" is "Jordan"
   - get the next N bytes

WHAT IS A DATABASE?

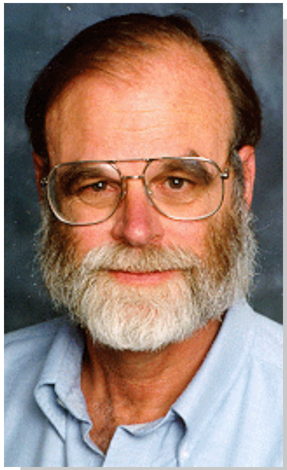
A collection of information organized to afford efficient retrieval

http://www.usg.edu/galileo/skills/unit04/primer04_01.phtml

ANOTHER VIEW

"When people use the word database, fundamentally what they are saying is that the data should be self-describing and it should have a schema. That's really all the word database means."

-- Jim Gray, "The Fourth Paradigm"



WHY WOULD I WANT A DATABASE?

1. Sharing
   Support concurrent access by multiple readers and writers
2. Data Model Enforcement
   Make sure all applications see clean, organized data
3. Scale
   Work with datasets too large to fit in memory
4. Flexibility
   Use the data in new, unanticipated ways

QUESTIONS TO CONSIDER

How is the data physically organized on disk?

What kinds of queries are efficiently supported by this organization, and what kinds are not?

How hard is it update the data, or add new data?

What happens when I encounter new queries that I didn't anticipate? Do I reorganize the data? How hard is that?

MOTIVATING RELATIONAL DATABASES

BILL HOWE, PHD
DIRECTOR OF RESEARCH, SCALABLE DATA ANALYTICS
UNIVERSITY OF WASHINGTON ESCIENCE INSTITUTE

teaching
datascience
.org

QUESTIONS TO CONSIDER

How is the data physically organized on disk?

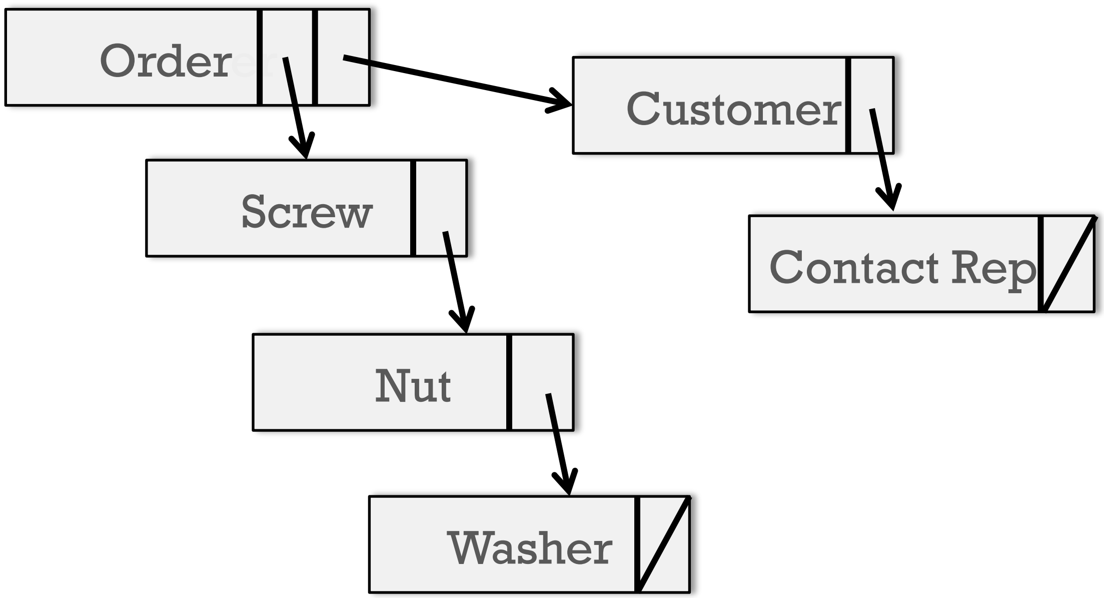
What kinds of queries are efficiently supported by this organization, and what kinds are not?

How hard is it update the data, or add new data?

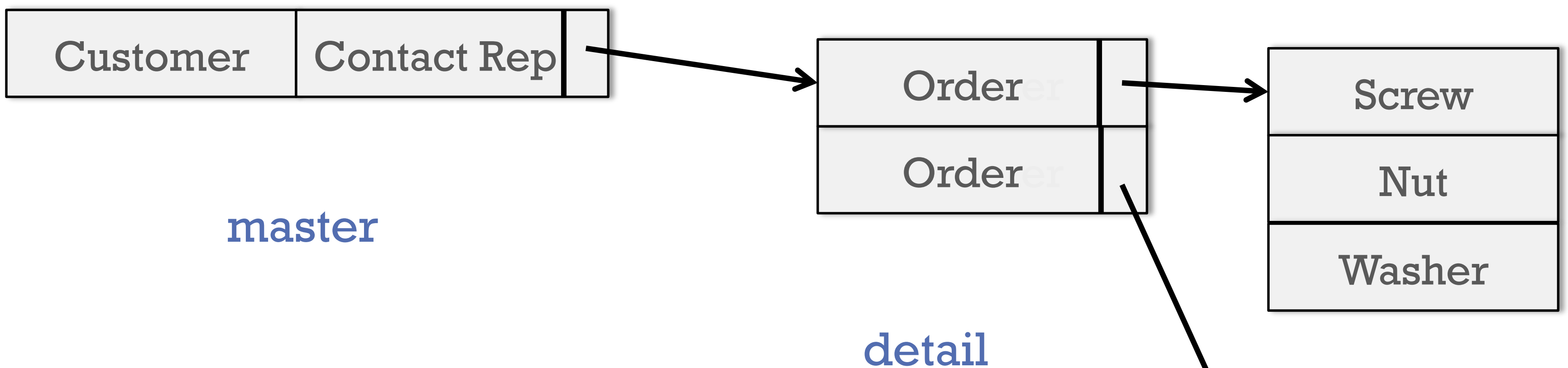
What happens when I encounter new queries that I didn't anticipate? Do I reorganize the data? How hard is that?

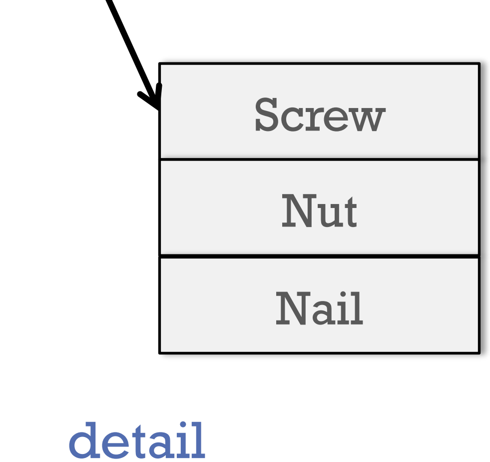
HISTORICAL EXAMPLE: NETWORK DATABASES

Database: A collection of information organized to afford efficient retrieval



HISTORICAL EXAMPLE: HIERARCHICAL DATABASES





Works great if you want to find all orders for a particular customer.

But what if you want to find all Customers who ordered a Nail?

ONE VIEW

"Relational Database Management Systems were invented to let you use one set of data in multiple ways, including ways that are unforeseen at the time the database is built and the 1st applications are written."

- Curt Monash, analyst/blogger

RELATIONAL DATABASES (CODD 1970)

Everything is a table

Every row in a table has the same columns

Relationships are implicit: no pointers

Course	Student Id
CSE 344	223…
CSE 344	244…
CSE 514	255..
CSE 514	244…

Student Id	Student Name
223…	Jane
244…	Joe
255..	Susan

DATABASE PHILOSOPHY

God made the integers;
all else is the work of
man.

- Leopold Kronecker, 19th Century
Mathematician

Codd made relations;
all else is the work of man.

- Raghu Ramakrishnan, DB text book author

slide src: Mike Franklin

RELATIONAL DATABASE HISTORY

Pre-Relational: if your data changed, your application broke.

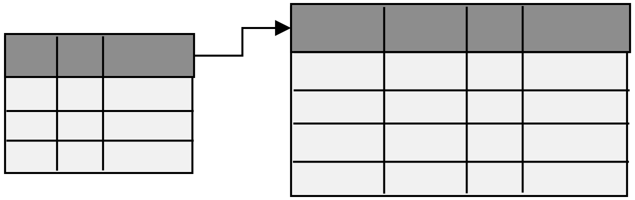
Early RDBMS were buggy and slow (and often reviled), but required only 5% of the application code.

"Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation are changed."

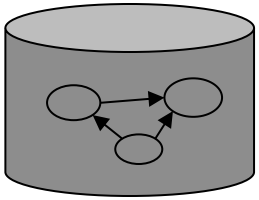
- Codd 1979

Key Ideas: Programs that manipulate tabular data exhibit an <u>algebraic structure</u> allowing reasoning and manipulation independently of physical data representation

KEY IDEA: "PHYSICAL DATA INDEPENDENCE"



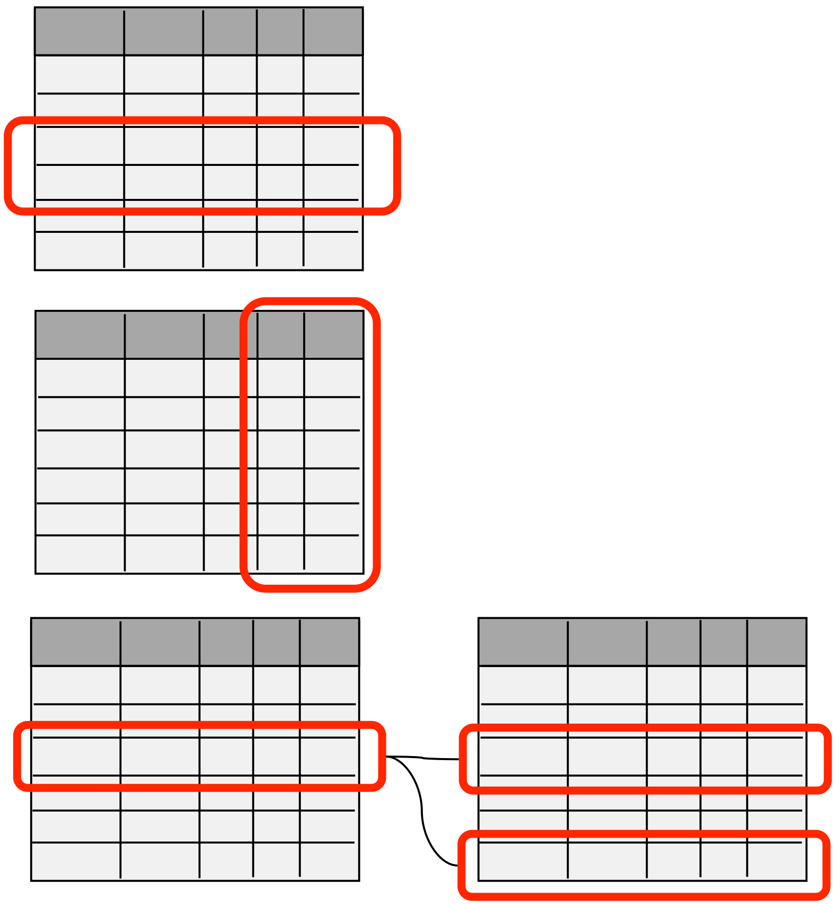
physical data independence




```
SELECT seq
  FROM ncbi_sequences
 WHERE seq =
 'GATTACGATATTA' ;
```

```
f = fopen( 'table_file' );
fseek(10030440);
while (True) {
  fread(&buf, 1, 8192, f);
  if (buf == GATTACGATATTA) {
    . . .
```

KEY IDEA: AN *ALGEBRA OF TABLES*



select

project

join

Other operators: aggregate, union, difference, cross product

KEY IDEA: ALGEBRAIC OPTIMIZATION

N = ((z*2)+((z*3)+0))/1

Algebraic Laws:

1. **(+) identity:** **x+0 = x**
2. **(/) identity:** **x/1 = x**
3. **(*) distributes:** **(n*x+n*y) = n*(x+y)**
4. **(*) commutes:** **x*y = y*x**

Apply rules 1, 3, 4, 2:

N = (2+3)*z

two operations instead of five, no division operator

Same idea works with the Relational Algebra!

EQUIVALENT LOGICAL EXPRESSIONS; DIFFERENT COSTS

$$\sigma_{p=knows}(R) \bowtie_{o=s} (\sigma_{p=holdsAccount}(R) \bowtie_{o=s} \sigma_{p=accountHomepage}(R))$$

right associative

$$(\sigma_{p=knows}(R) \bowtie_{o=s} \sigma_{p=holdsAccount}(R)) \bowtie_{o=s} \sigma_{p=accountHomepage}(R)$$

left associative

$$\sigma_{p1=knows\ \&\ p2=holdsAccount\ \&\ p3=accountHomepage}(R \times R \times R)$$

cross product

SAME LOGICAL EXPRESSION, DIFFERENT PHYSICAL ALGORITHMS

A = select(p=knows)
B = select(p=holdsAccount)
C = select(p=accountWebpage)

hA = hash(A)
AB = probe hA with B

hC = hash(C)
ABC = probe hC with AB

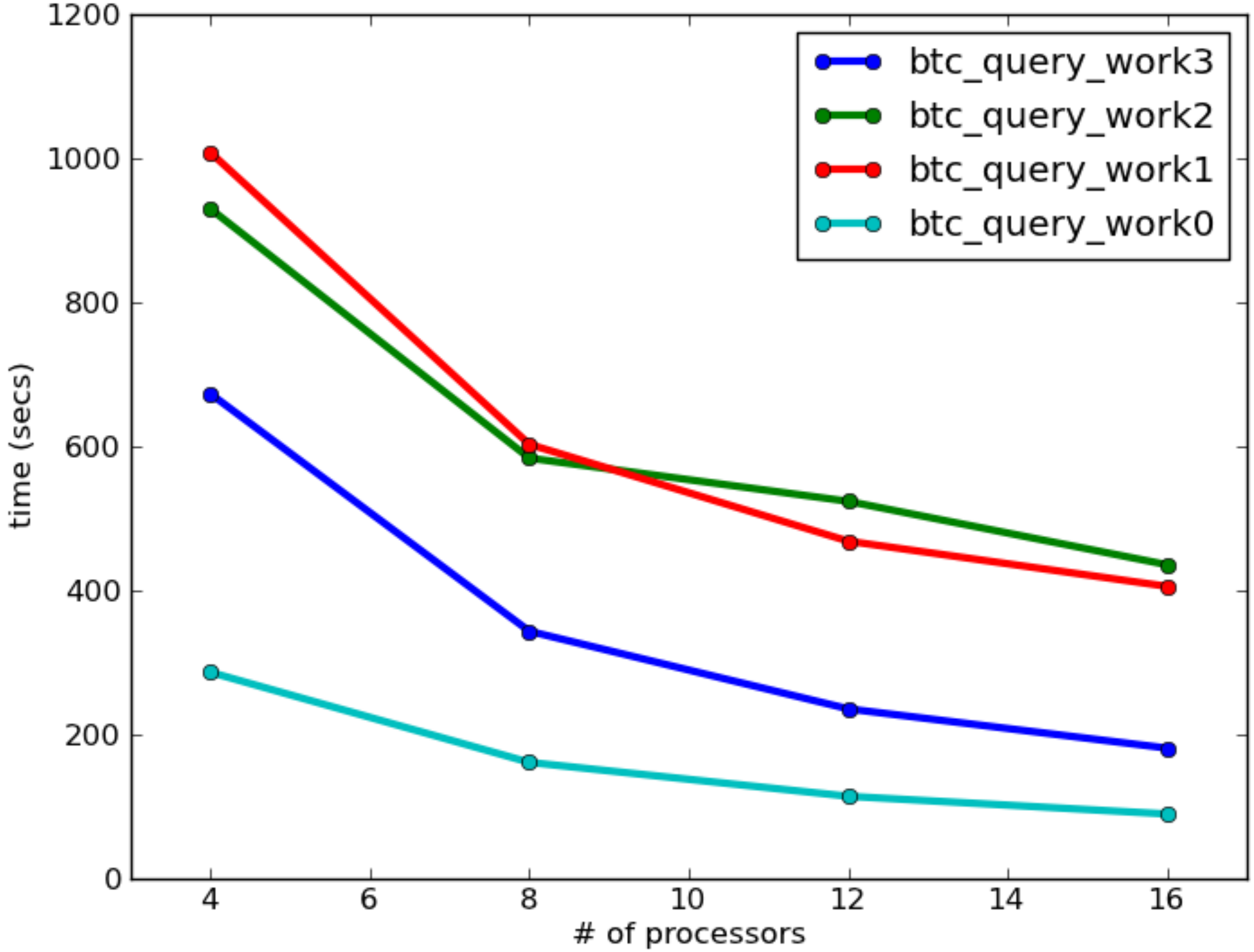
A = select(p=knows)
B = select(p=holdsAccount)
C = select(p=accountWebpage)

hB = hash(B)
AB = probe hB with A

hC = hash(C)
ABC = probe hC with AB

Which is faster?

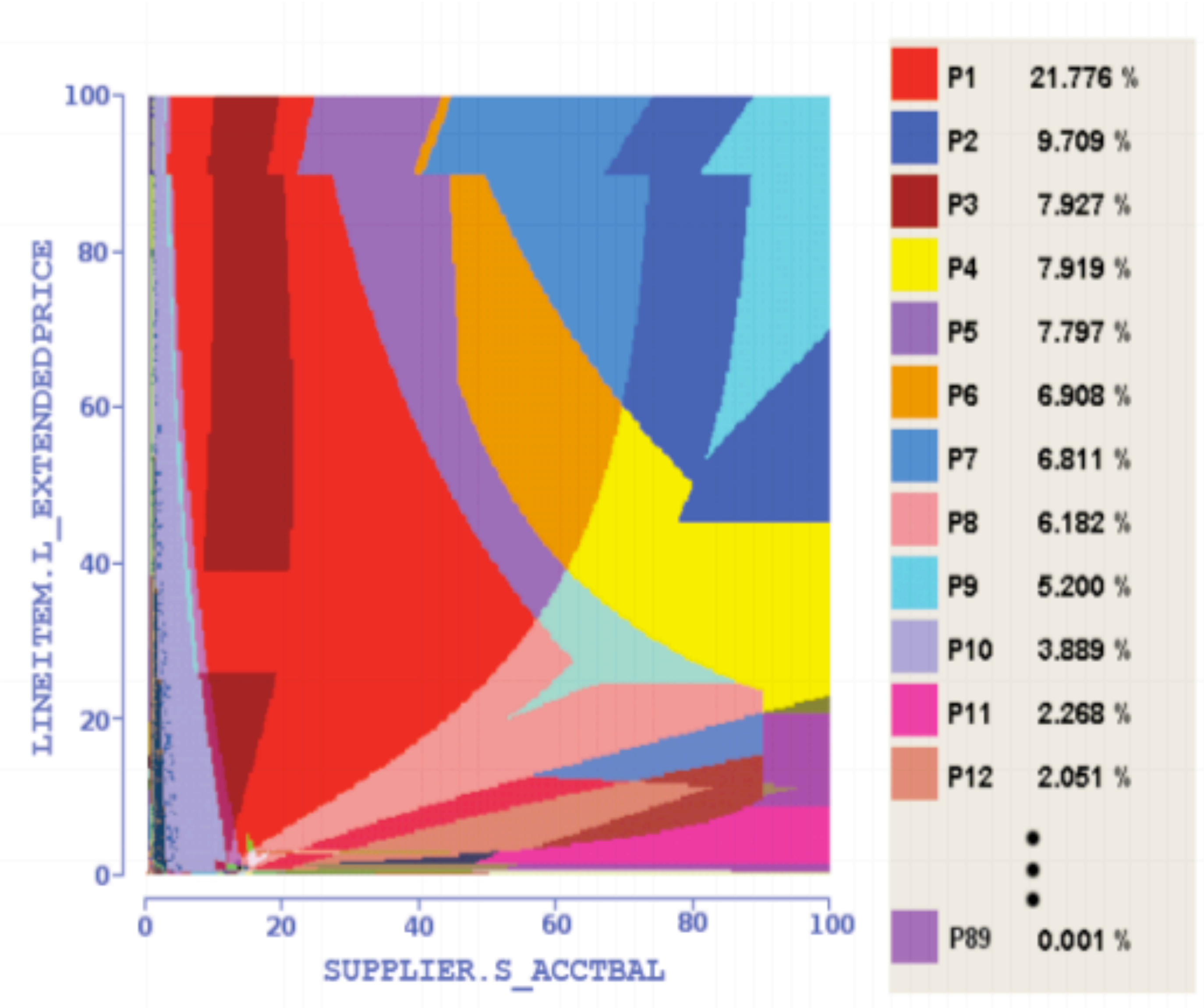
ALGEBRAIC OPTIMIZATION MATTERS



BTC 2010 Dataset

3B quads
623 GB processed

PICASSO QUERY PLAN DIAGRAMS

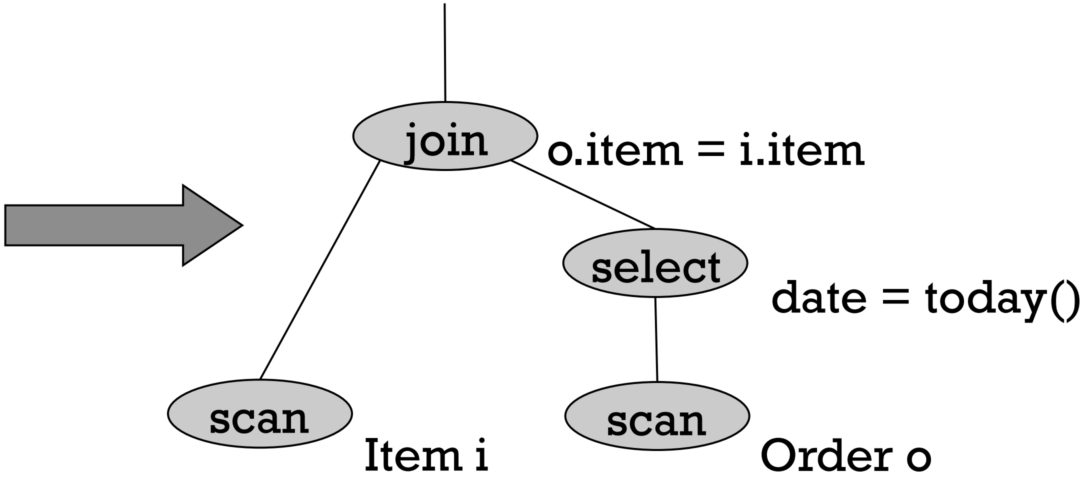


Haritsa, VLDB 2010

KEY IDEA: DECLARATIVE LANGUAGES

Find all orders from today, along with the items ordered

```
SELECT *
  FROM Order o, Item i
 WHERE o.item = i.item
   AND o.date = today()
```



SQL IS THE "WHAT" NOT THE "HOW"

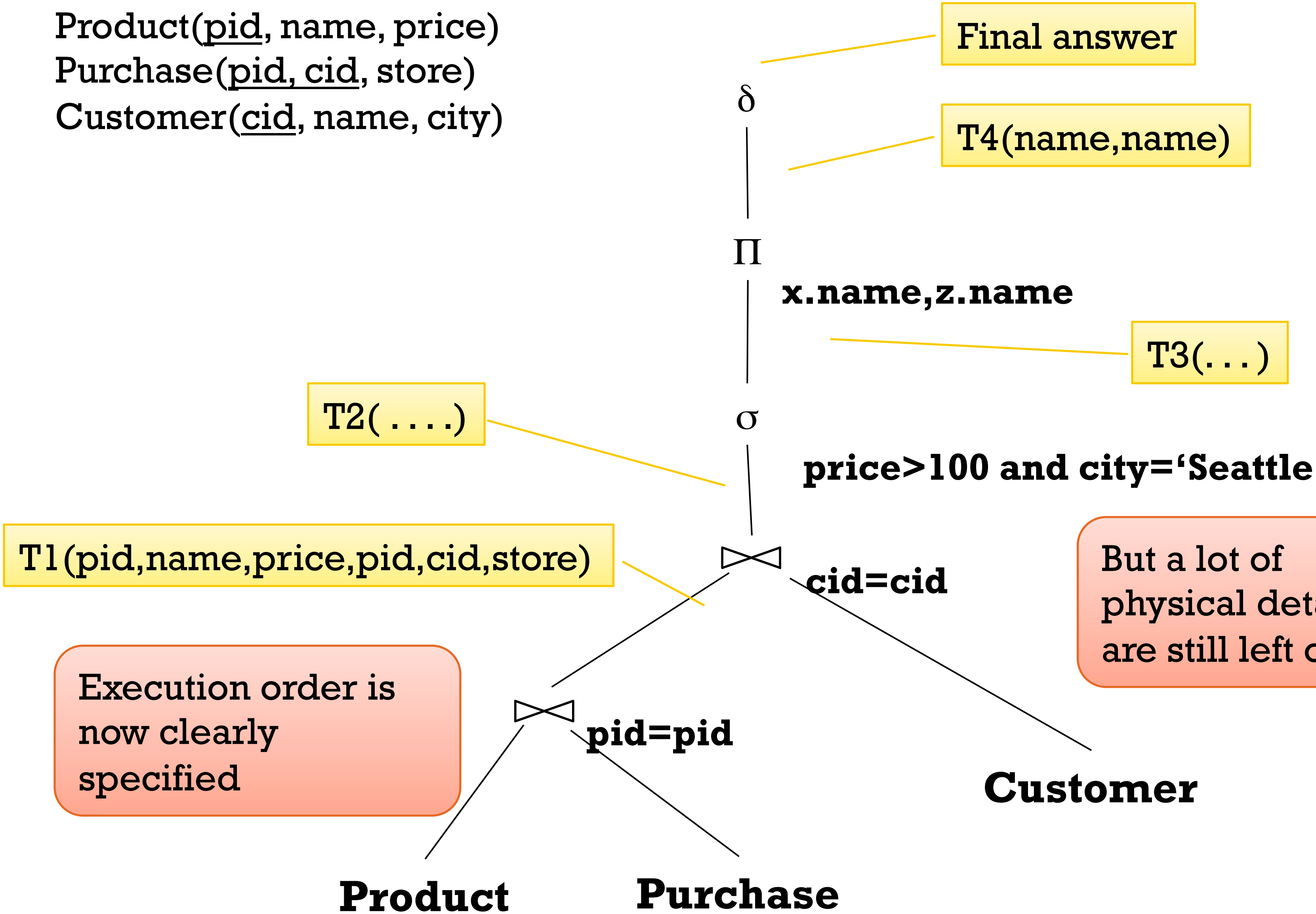
Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)

```
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

RELATIONAL ALGEBRA

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



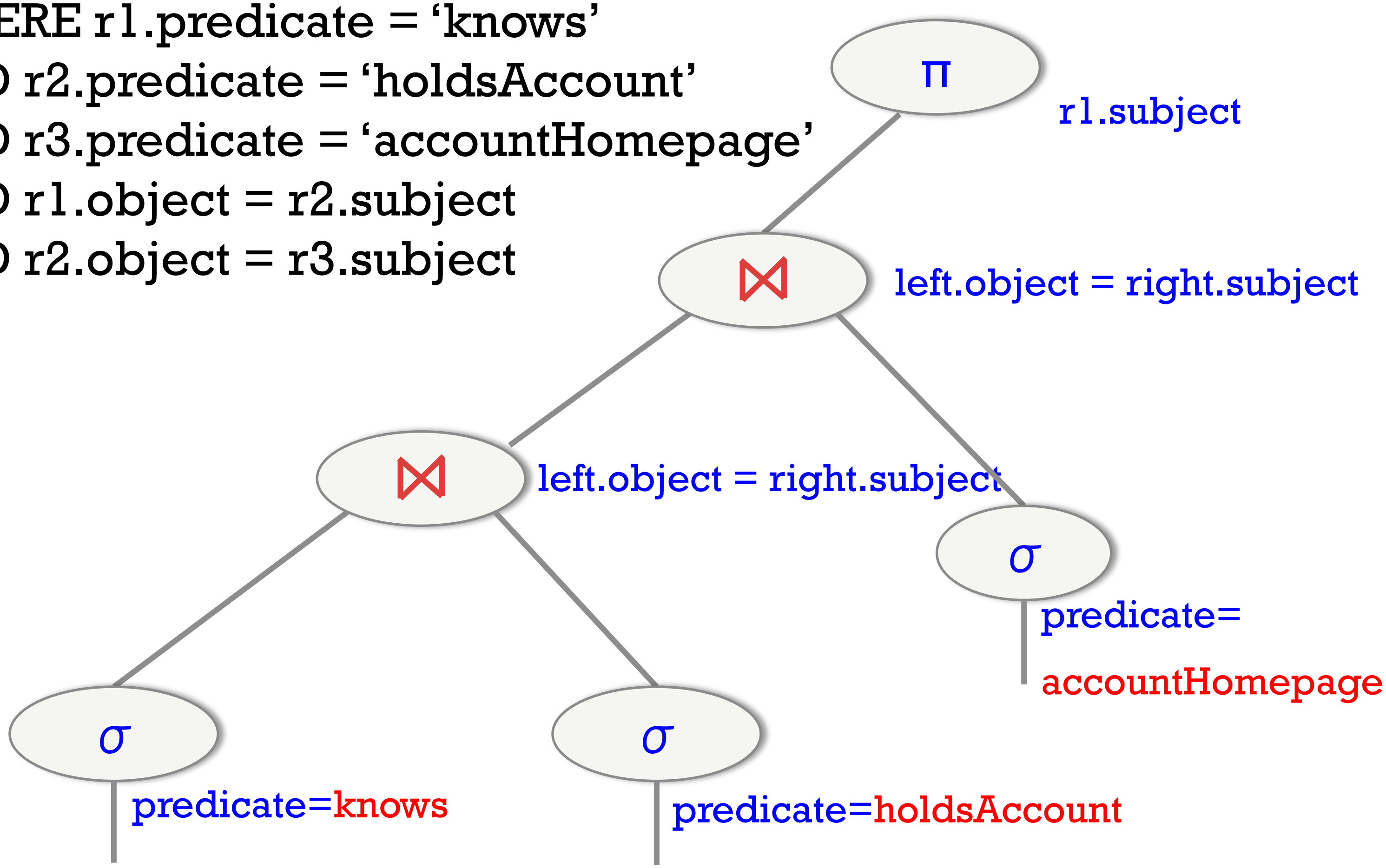
Execution order is now clearly specified

But a lot of physical details are still left open!

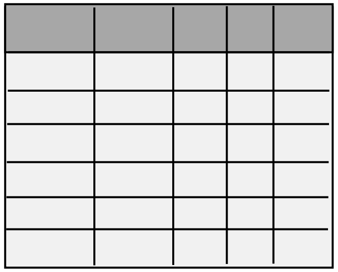
ANOTHER EXAMPLE

R(subject, predicate, object)

SELECT r1.subject
FROM R r1, R r2, R r3
WHERE r1.predicate = ‘knows’
AND r2.predicate = ‘holdsAccount’
AND r3.predicate = ‘accountHomepage’
AND r1.object = r2.subject
AND r2.object = r3.subject



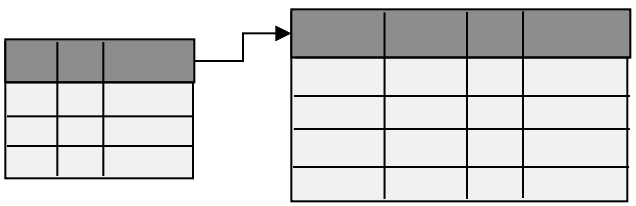
KEY IDEA: "LOGICAL DATA INDEPENDENCE"



views

```
SELECT *
  FROM my_sequences
```

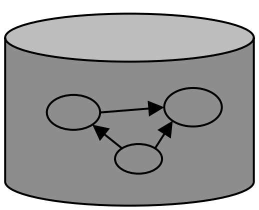
logical data independence



relations

```
SELECT seq
  FROM ncbi_sequences
 WHERE seq =
 'GATTACGATATTA' ;
```

physical data independence



files and
pointers

```
f = fopen( 'table_file' );
fseek(10030440);
while (True) {
  fread(&buf, 1, 8192, f);
  if (buf == GATTACGATATTA) {
     . . .
```

WHAT ARE VIEWS?

A view is just a query with a name

We can use the view just like a real table

Why can we do this?

Because we know that every query returns a relation: We say that the language is "algebraically closed"

VIEW EXAMPLE

A view is a relation defined by a query

Purchase(customer, product, store)
Product(<u>pname</u>, price)

StorePrice(store, price)

```
CREATE VIEW  StorePrice AS
SELECT x.store, y.price
FROM  Purchase x, Product y
WHERE x.pid = y.pid
```

This is like a new table
StorePrice(store,price)

StorePrice(store, price)

HOW TO USE A VIEW?

Customer(cid, name, city)
Purchase(customer, product, store)
Product(pname, price)

A "high end" store is a store that sold some product over 1000. For each customer, find all the high end stores that they visit. Return a set of (customer-name, high-end-store) pairs.

```
SELECT DISTINCT z.name, u.store
FROM Customer z, Purchase u, StorePrice v
WHERE z.cid = u.customer
AND u.store = v.store
AND v.price > 1000
```

KEY IDEA: INDEXES

Databases are especially, but not exclusively, effective at "Needle in Haystack" problems:

- **Extracting small results from big datasets**
- **Transparently provide "old style" scalability**
- **Your query will always* finish, regardless of dataset size.**

- **Indexes are easily built and automatically used when appropriate**

```
CREATE INDEX seq_idx ON sequence(seq);

SELECT seq
  FROM sequence
 WHERE seq = 'GATTACGATATTA';
```

***almost**

IN-DATABASE ANALYTICS

BILL HOWE, PHD
DIRECTOR OF RESEARCH, SCALABLE DATA ANALYTICS
UNIVERSITY OF WASHINGTON ESCIENCE INSTITUTE

teaching datascience .org

"There is no point in bringing data … into the data warehouse environment without integrating it. If the data arrives at the data warehouse in an unintegrated state, it cannot be used to support a corporate view of data. And a corporate view of data is one of the essences of the architected environment."

- Inmon, 2005, "Building the Data Warehouse"

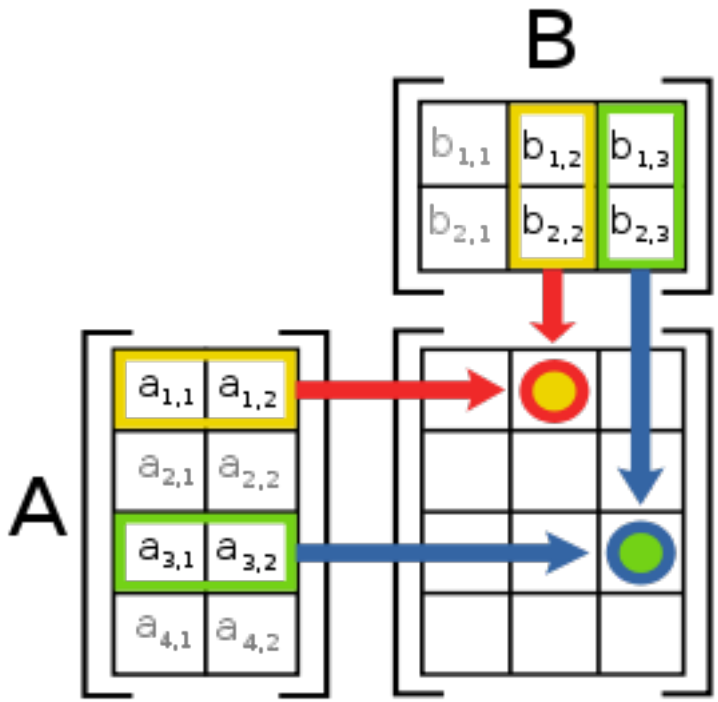
Not a good fit for analytics!

MATRIX ADDITION, VECTOR REPRESENTATION

A(row_number, row float[])
B(row_number, row float[])

```
SELECT A.row_number, A.vector + B.vector
FROM A, B
WHERE A.row_number = B.row_number;
```

MATRIX MULTIPLICATION



MATRIX MULTIPLICATION, VECTOR REPRESENTATION

A(row_number, row float[])

SELECT 1, array_accum(row_number, vector*v) FROM A;

$$\vec{x} \cdot \vec{y} = \Sigma_i x_i y_i$$

MATRIX TRANSPOSE, VECTOR REPRESENTATION

A(row_number, row float[])

```
SELECT S.col_number, array_accum(A.row_number, A.vector[S.col_number])
FROM A, generate_series(1,3) AS S(col_number)
GROUP BY S.col_number;
```

generate_series is a *table function*

MATRIX MULTIPLICATION, SPARSE REPRESENTATION

A(row_number, column_number, value)
B(row_number, column_number, value)

```
SELECT A.row_number, B.column_number, SUM(A.value * B.value)
FROM A, B
WHERE A.column_number = B.row_number
GROUP BY A.row_number, B.column_number
```

ASIDE: USER-DEFINED FUNCTIONS AND TYPES

Scalar functions

```
SELECT myfunc(r.a, r.b)…
```

Aggregate functions

```
SELECT concat(r.s) …
```

Table functions

```
SELECT … FROM tablefunc(a,b)
```

EXPERIMENT DESIGN

```
CREATE VIEW design AS
SELECT a.trial_id,
floor (100 * random()) AS row_id
FROM generate_series(1,10000) AS a (trial_id),
generate_series(1,3) AS b (subsample_id)
```

```
CREATE VIEW trials AS
SELECT d.trial_id, AVG(a.values) AS avg_value
FROM design d, T
WHERE d.row_id = T.row_id
GROUP BY d.trial_id
```

src: Cohen et al., VLDB 2009

Prior to the implementation of this functionality within the DBMS, one Greenplum customer was accustomed to calculating the OLS by exporting data and importing the data into R for calculation, a process that took several hours to complete. They reported signicant performance improvement when they moved to running the regression within the DBMS. Most of the benet derived from running the analysis in parallel close to the data with minimal data movement.

src: Cohen et al., VLDB 2009

SLIDES CAN BE FOUND AT:

TEACHINGDATASCIENCE.ORG

teaching
datascience
.org