SQL

INTRODUCTION TO DATA SCIENCE

TIM KRASKA

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 (use Piazza if you still need a group)

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
- 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%).
- **List of potential data sources and some project ideas are online** http://cs.brown.edu/courses/csci1951-a/final_project.shtml
- **You are encouraged to find your own data and create your own project**
- Pre-proposal is not binding even for the mid-term report
- We will have a competition for the best Mid-term report/project

LAST LECTURE

- SQL basics:

  Select
  From
  Where
  Group By
  Having
  Order By

- Nested queries
- Set operations
- Null values

CLICKER: EXISTENTIAL QUANTIFICATION: EXISTS SUB-QUERIES

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

Professor

Person-Id	Name
1	Stan
2	Ugur
3	Eli

Lecture

Person-Id	Course-ID
1	CS127
1	CS227
3	CS155

What is the result of the query?

(a) {Stan, Eli}

(b) {Ugur}

(c) {Eli}

EXISTENTIAL QUANTIFICATION: EXISTS SUB-QUERIES

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

Professor

Person-Id	Name
1	Stan
2	Ugur
3	Eli

Lecture

Person-Id	Course-ID
1	CS127
1	CS227
3	CS155

EXISTENTIAL QUANTIFICATION: EXISTS SUB-QUERIES

```
select p.Name
from Professor p
where not exists ( select *
                   from Lecture l
                   where l.Person-ID = 1);
```

Professor

Person-Id	Name
1	Stan
2	Ugur
3	Eli

Lecture

Person-Id	Course-ID
1	CS127
1	CS227
3	CS155

EXISTENTIAL QUANTIFICATION: EXISTS SUB-QUERIES

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

Professor

Person-Id	Name
1	Stan
2	Ugur
3	Eli

Lecture

Person-Id	Course-ID
1	CS127
1	CS227
3	CS155

EXISTENTIAL QUANTIFICATION: EXISTS SUB-QUERIES

```
select p.Name
from Professor p
where not exists ( select *
                   from Lecture l
                   where l.Person-ID = 2);
```

Professor

Person-Id	Name
1	Stan
2	Ugur
3	Eli

Lecture

Person-Id	Course-ID
1	CS127
1	CS227
3	CS155

EXISTENTIAL QUANTIFICATION: EXISTS SUB-QUERIES

```
select p.Name
from Professor p
where not exists ( select *
                   from Lecture l
                   where l.Person-ID = 2);
```

Professor

Person-Id	Name
1	Stan
2	Ugur
3	Eli

Lecture

Person-Id	Course-ID
1	CS127
1	CS227
3	CS155

Result: {Ugur}

EXISTENTIAL QUANTIFICATION: EXISTS SUB-QUERIES

```
select p.Name
from Professor p
where not exists ( select *
                   from Lecture l
                   where l.Person-ID = 3);
```

Professor

Person-Id	Name
1	Stan
2	Ugur
3	Eli

Lecture

Person-Id	Course-ID
1	CS127
1	CS227
3	CS155

Result: {Ugur}

UNCORRELATED SUB-QUERY

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

Professor

Person-Id	Name
1	Stan
2	Ugur
3	Eli

Lecture

Person-Id	Course-ID
1	CS127
1	CS227
3	CS155

UNCORRELATED SUB-QUERY

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

Person-Id	Name
1	Stan
2	Ugur
3	Eli

Person-Id	Course-ID
1	CS127
1	CS227
3	CS155

UNCORRELATED SUB-QUERY

select Name

from Professor

where Person-ID **not in** (**select** Person-ID

from Lecture);

Person-Id	Name
1	Stan
2	Ugur
3	Eli

Person-Id	Course-ID
1	CS127
1	CS227
3	CS155

SYNTACTIC SUGAR

```
select *
from Student
where 1 <= Semester  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%´;

CLICKER

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)

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
⋮	⋮	⋮	⋮	⋮	⋮	⋮

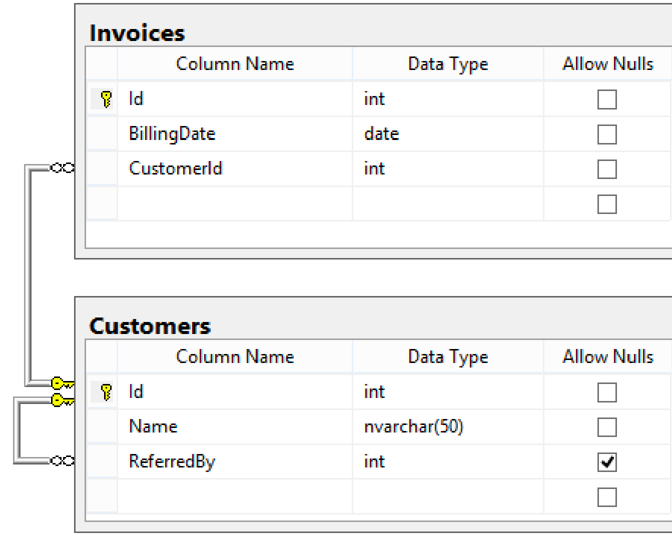
FULL OUTER JOINS

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	-	-	-	-	-

CLICKER

You have your first day as a Data Scientist at TheShop LLC. TheShop uses SQL server. A quick look using VisualStudio shows the simply database schema on the right. In order to analyze all the orders in Python, 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.



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 Right JOIN Customers c ON i.CustomerId = c.Id)
          Right 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"

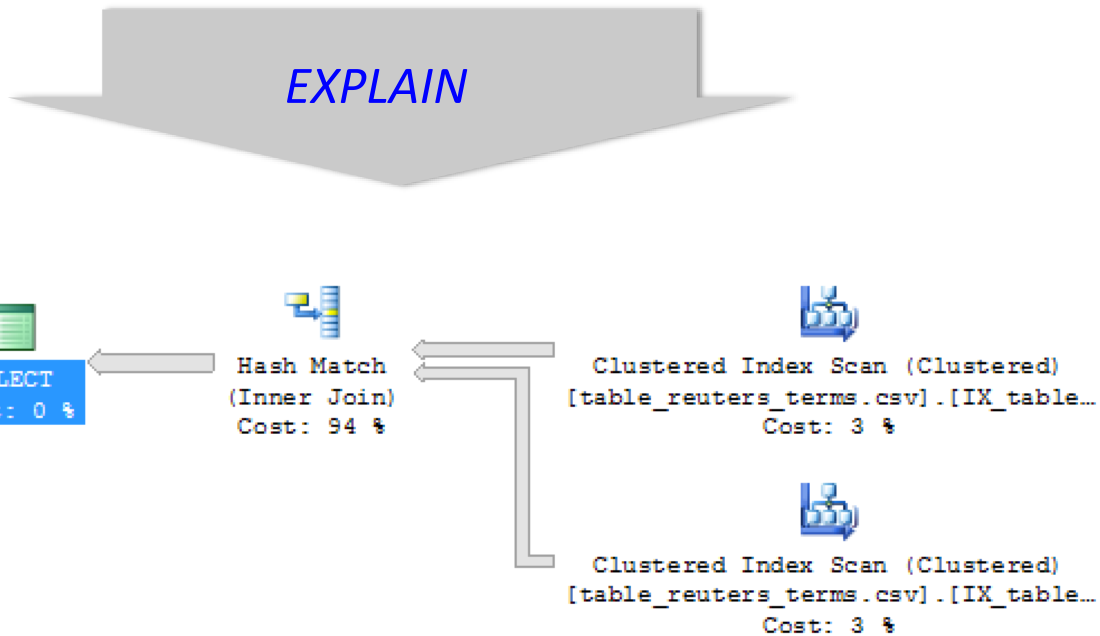
Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)

```
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 = 'Providence'
```

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

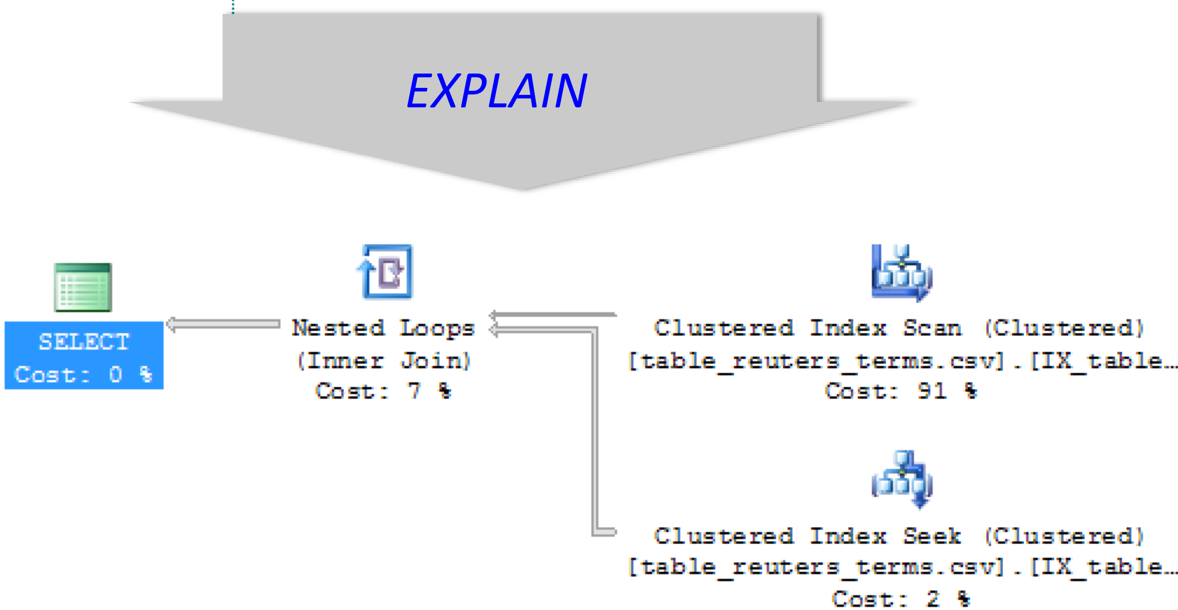
Exposing the Algebra: Microsoft SQL Server

```
select a.term_id, b.term_id
from [billhowe].[reuters] a, [billhowe].[reuters] b
where a.doc_id = b.doc_id
  and a.term_id != b.term_id
```



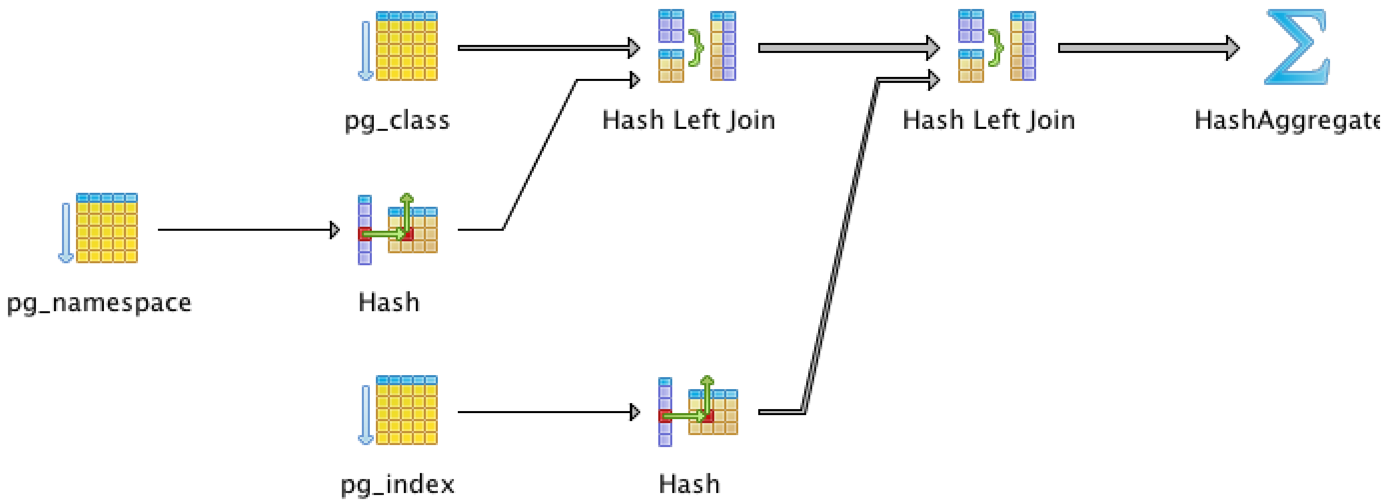
Exposing the Algebra: Microsoft SQL Server

```sql
select a.term_id, b.term_id
from [billhowe].[reuters] a, [billhowe].[reuters] b
where a.doc_id = b.doc_id
  and a.term_id != b.term_id
  and a.term_id = 'parliament'
```



Exposing the Algebra: PostgreSQL

EXPLAIN SELECT ….



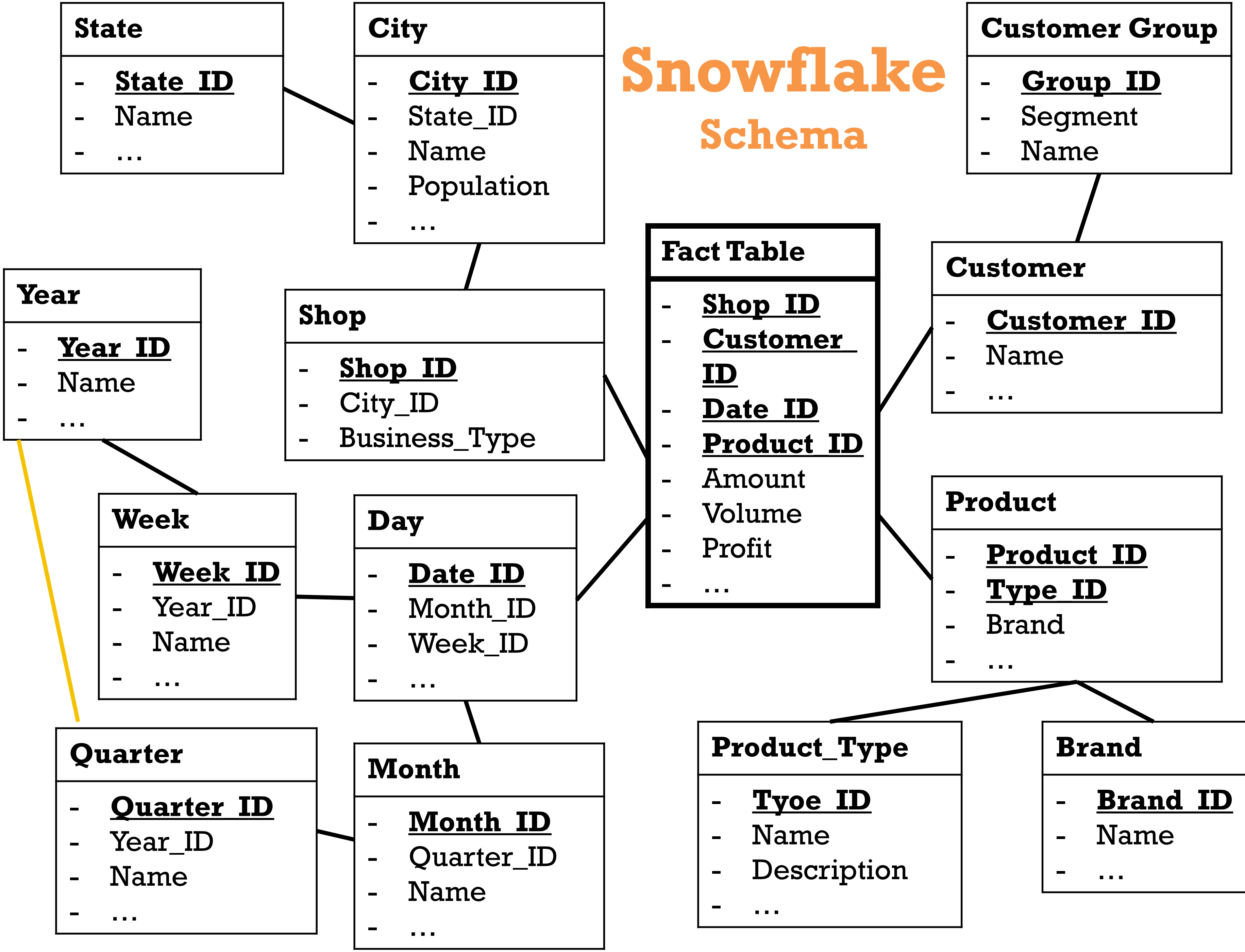
screenshot from pgAdmin3

STAR VS SNOWFLAKE SCHEMA

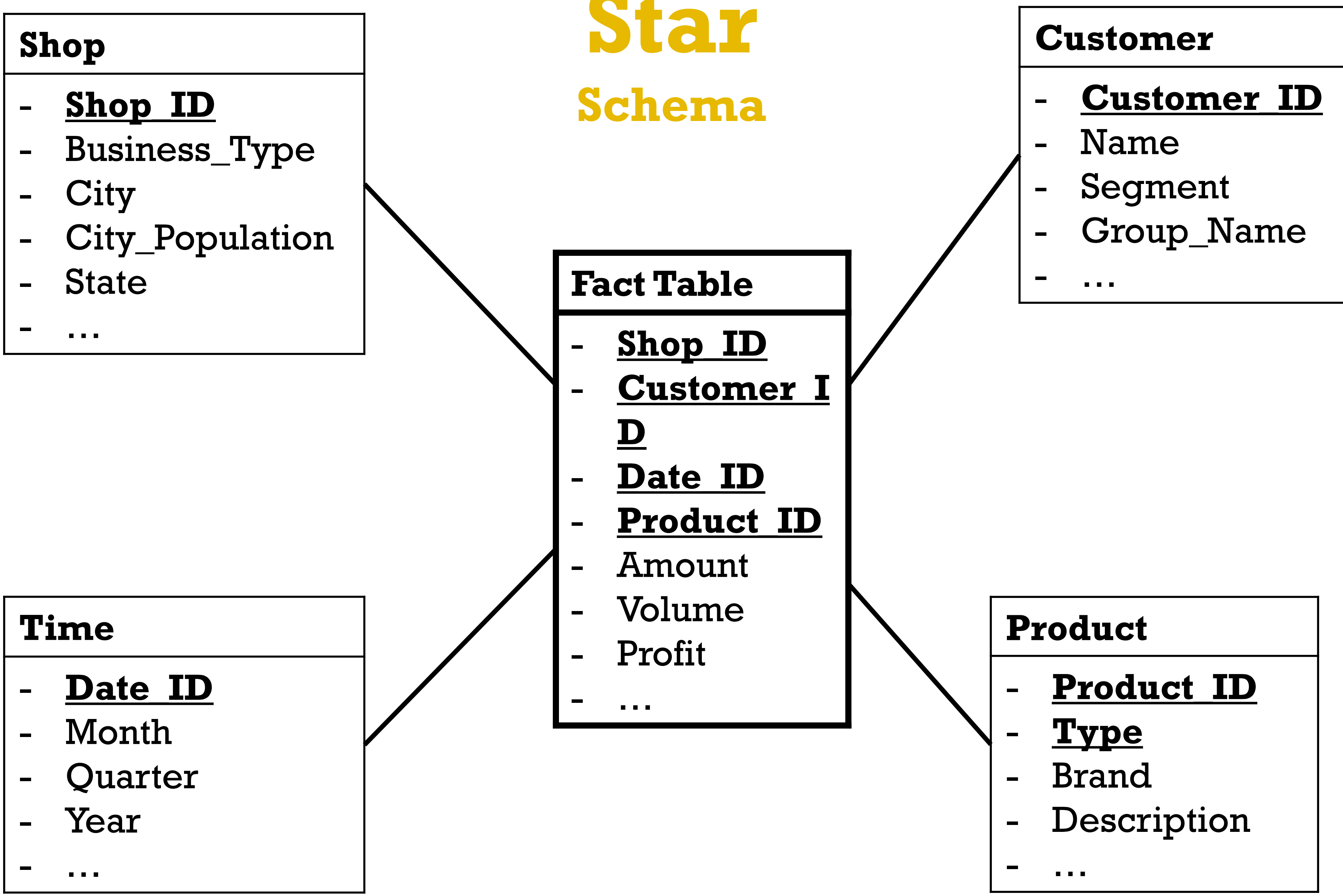
DATA WAREHOUSES

teaching datascience .org

Snowflake Schema



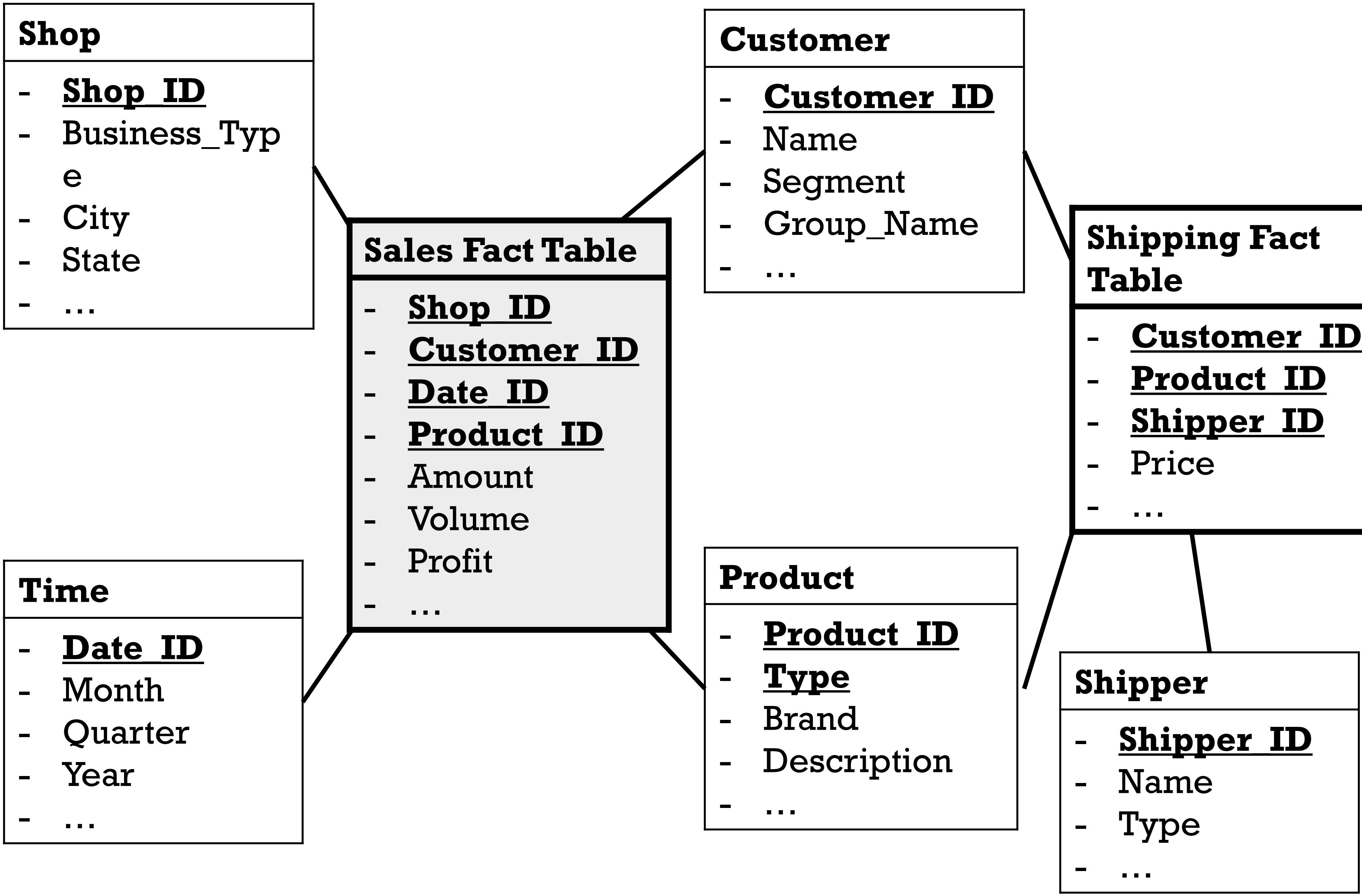
Star
Schema


Shop
- Shop_ID
- Business_Type
- City
- City_Population
- State
- …
Customer
- Customer_ID
- Name
- Segment
- Group_Name
- …
Fact Table
- Shop_ID
- Customer_ID
- Date_ID
- Product_ID
- Amount
- Volume
- Profit
- …
Time
- Date_ID
- Month
- Quarter
- Year
- …
Product
- Product_ID
- Type
- Brand
- Description
- …

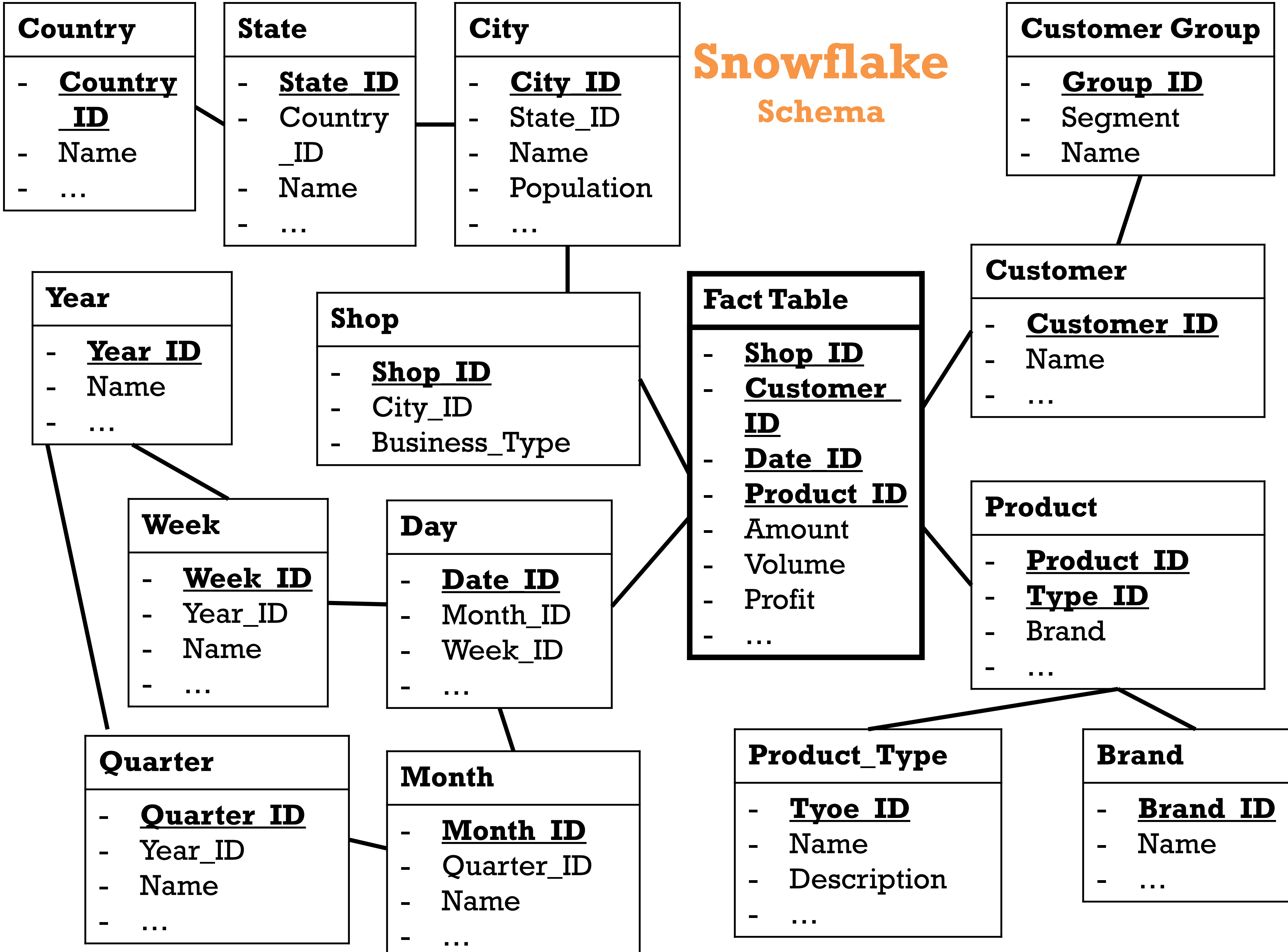
STAR VS. SNOWFLAKE SCHEMA

	Snowflake	Star
Normalization/ De-Normalization	Dimension Tables are in Normalized form but Fact Table is still in De-Normalized form	Both Dimension and Fact Tables are in De-Normalized form
Space	Smaller	Bigger (**Redundancy**)
Query Performance	More Joins → slower	Fewer Joins → faster
Ease of Use	Complex Queries	Pretty Simply Queries
When to use	When dimension table is relatively big in size, snowflaking is better as it reduces space.	When dimension table contains less number of rows, we can go for Star schema.

Galaxy / Fact Constellation
Schema



Snowflake Schema



2 DIMENSIONAL CASE

Dimensions

Date:month
June→Augu
2006

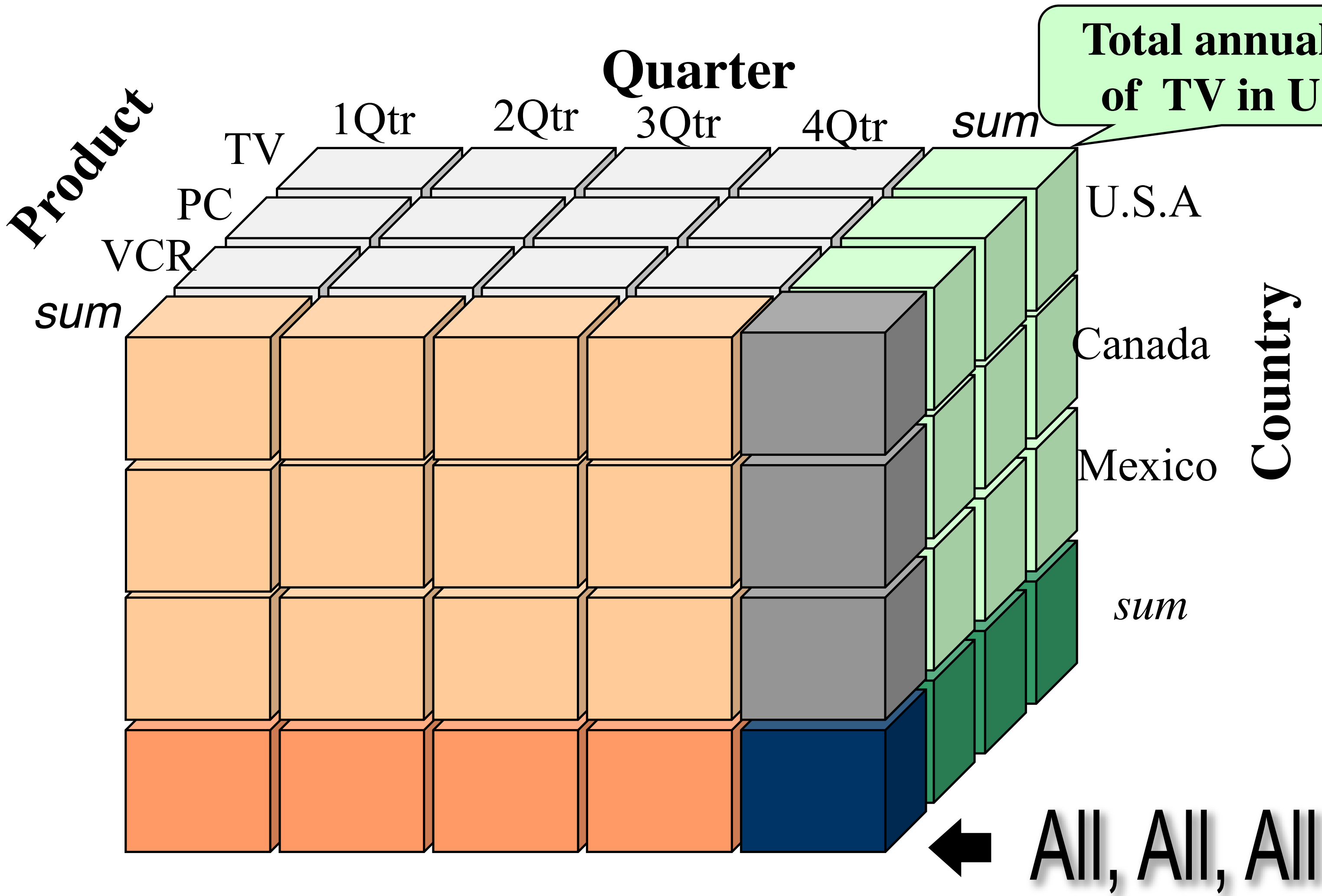
	Blue	Red	Orange	Total
June	51	25	158	234
July	58	20	120	198
August	65	22	51	138
Total	174	67	329	570

Data Points/
1st level of aggregation

Aggregated
w.r.t. X-dim

Aggregated
w.r.t. Y-dim

Aggregated
w.r.t. X and Y



Total annual sales of TV in U.S.A.
Quarter
Product
1Qtr
2Qtr
3Qtr
4Qtr
sum
TV
PC
VCR
sum
U.S.A
Canada
Mexico
sum
Country
All, All, All

TYPICAL OLAP OPERATIONS

What does OLAP stand for?
What OLTP?

Roll up (drill-up): summarize data

by climbing up hierarchy or by dimension reduction

Drill down (roll down): reverse of roll-up

from higher level summary to lower level summary or detailed data, or introducing new dimensions

Slice and dice: *project and select*

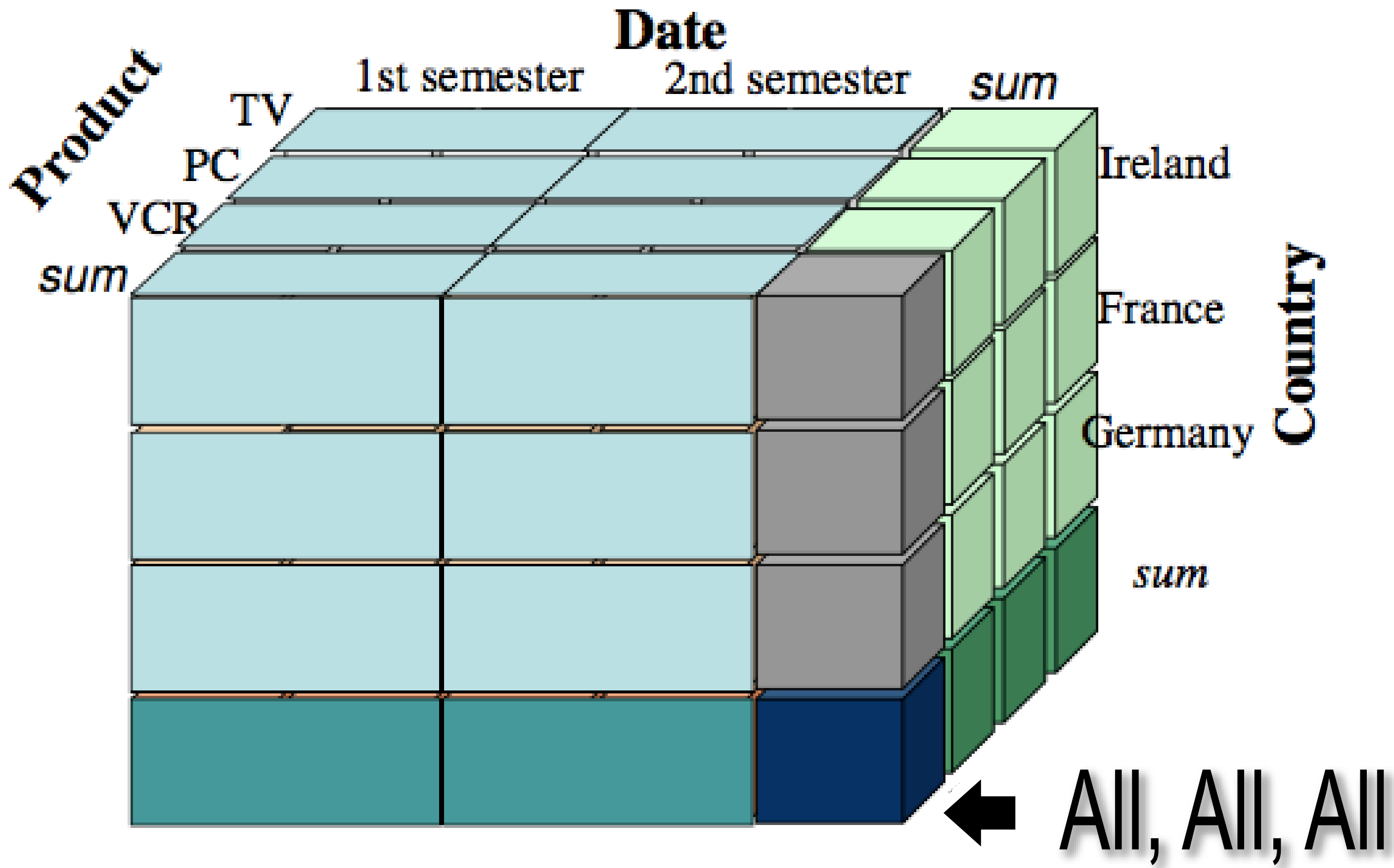
Pivot (rotate): *reorient the cube, visualization, 3D to series of 2D planes.*

Other operations

drill across: involving (across) more than one fact table

drill through: through the bottom level of the cube to its back-end relational tables (using SQL)

ROLLUP



Date
1st semester
2nd semester
sum
Product
TV
PC
VCR
sum
Ireland
France
Germany
sum
Country
All, All, All

REPRESENTING A CUBE

select *semester as date*, country, sum(sales)
from *sales*
group by cube(*semester,country*)

Date	Country	Sales
1st semester	Ireland	20
1st semester	France	126
1st semester	Germany	56
1st semester	**null**	202
2nd semester	Ireland	23
2nd semester	France	138
2nd semester	Germany	48
2nd semester	**null**	209
null	Ireland	43
null	France	264
null	Germany	104
null	**null**	411

REPRESENTING A CUBE

select *semester as date*, country, sum(sales)
from *sales*
group by roll-up(*semester,country*)

Date	Country	Sales
1st semester	Ireland	20
1st semester	France	126
1st semester	Germany	56
2nd semester	Ireland	23
2nd semester	France	138
2nd semester	Germany	48
1st semester	**null**	202
2nd semester	**null**	209
null	**null**	411

MDX

Multidimensional Expressions (MDX) is a query language for cubes

- Supported by many data warehouses
- Input and output are cubes

SELECT { [Measures].[Store Sales] } ON COLUMNS, { [Date].[2002], [Date].[2003] }

ON ROWS FROM Sales

WHERE ([Store].[USA].[CA])

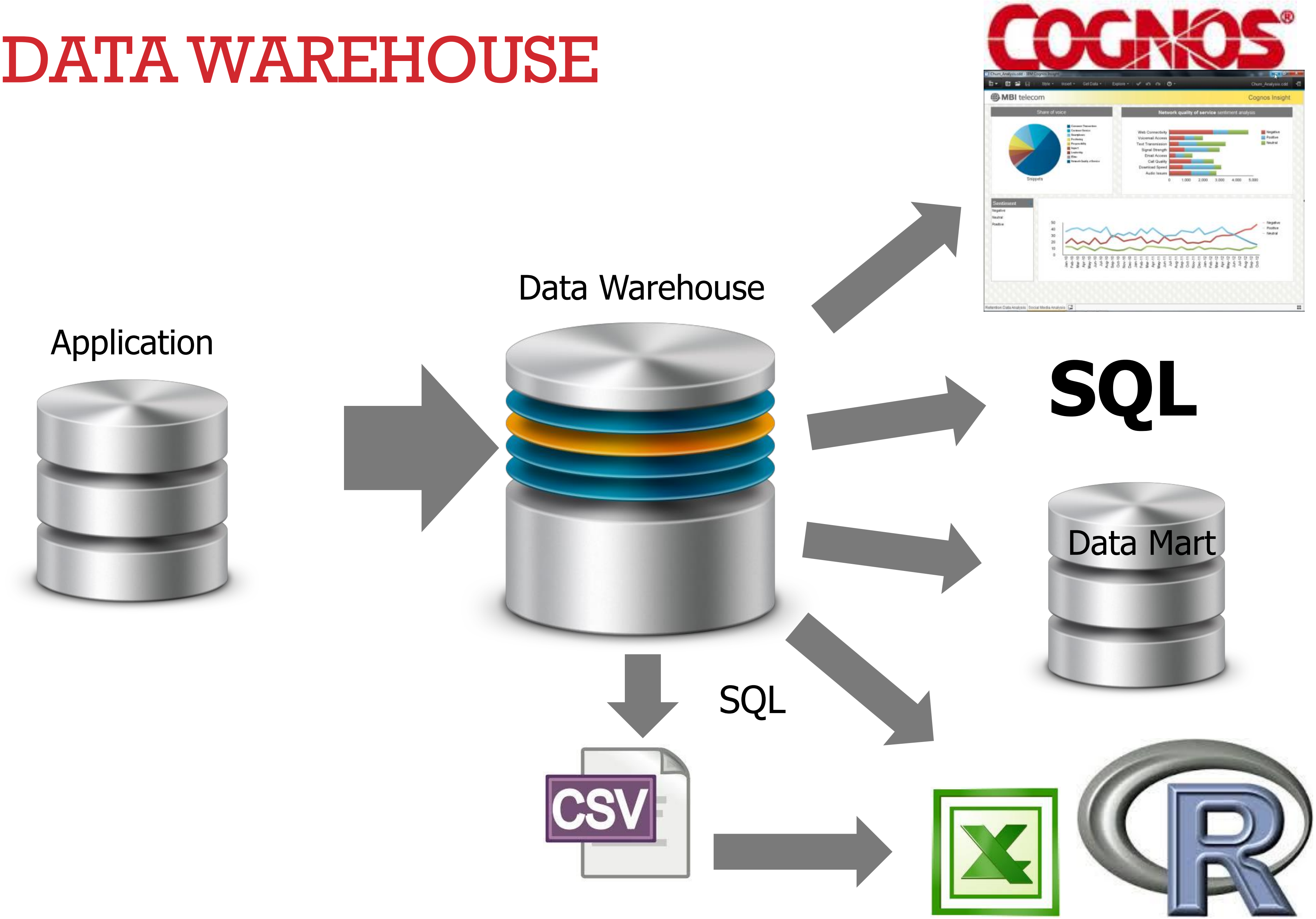
ENTITY RESOLUTION

CS1951A INTRO TO DATA SCIENCE

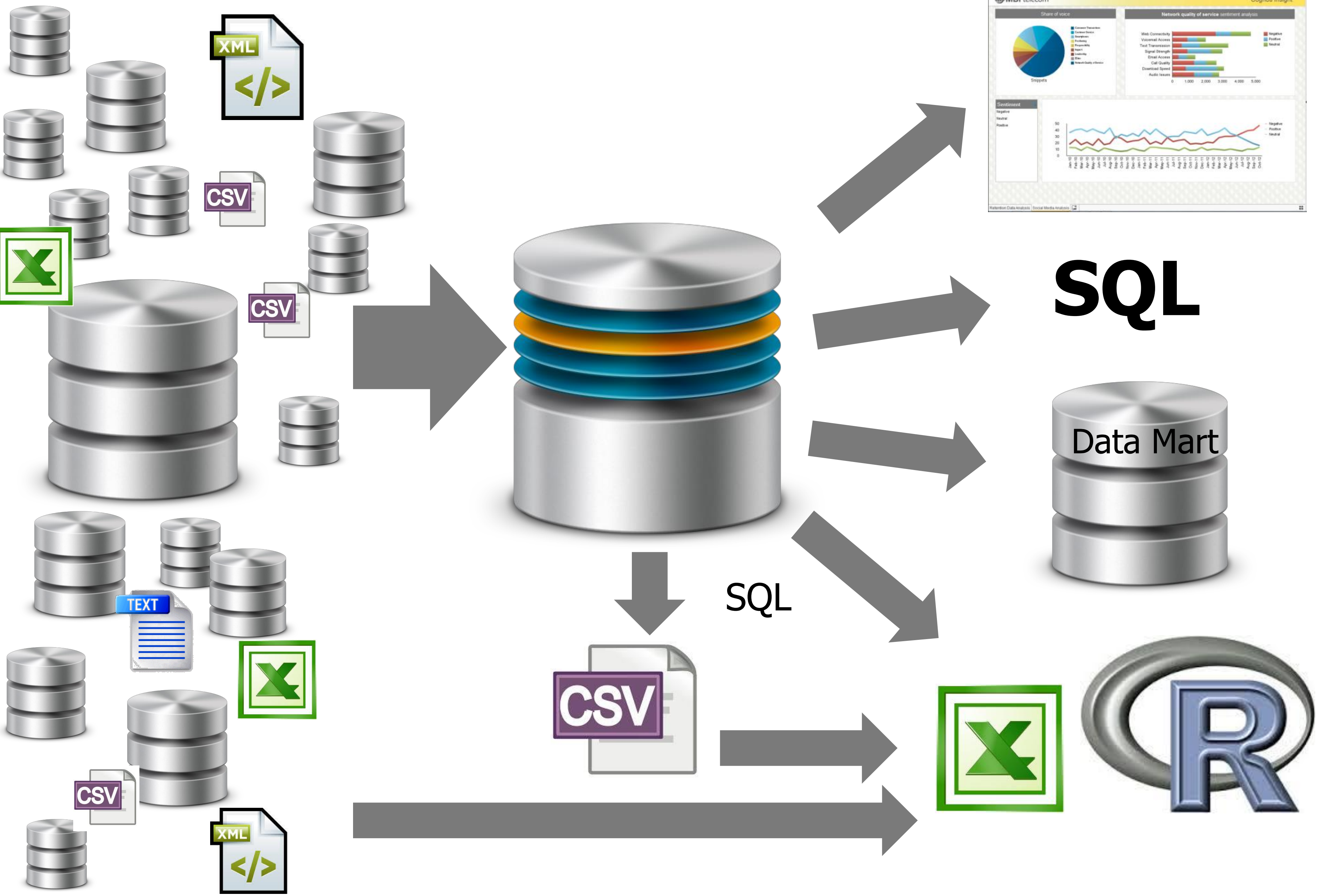
teaching datascience .org

Tim Kraska <tim_kraska@brown.edu>

DATA WAREHOUSE



DATA WAREHOUSE



DATA INTEGRATION

- **E**xtract-**T**ransform-**L**oad
  - “Old” term
  - Schema-centric
  - Batch-oriented
- Data Wrangling
  - Hipster term
  - Less structured
  - Ad-hoc

DATA INTEGRATION

SCHEMA MATCHING

ID	Name	Address	Zip	State	City	Phone	E-Mail
1	Tim Kraska	135 Watermann St,	02906	Providence	RI	+1 234234 234	Tim_kraska@brown.edu
…	…	…				..	..

ID	Name	Address	Phone	E-Mail
1	Tim Kraska	135 Watermann St, 02906 Providence, RI	+1 234234 234	Tim_kraska@brown.edu
…	…	…	..	..

ID	Name
1	Tim Kraska
…	…

AddressID	Person-ID	Address	Phone-Nb	E-Mail
1	1	135 Watermann St, 02906 Providence	+1 234234 234	Tim_kraska@brown.edu
1	1	222 Hope St, 02906 Providence	980 – 0803284	tim_kraska@brown.edu

CLICKER:
HOW MANY TABLES HAS A (TYPICAL) SAP'S ERP INSTALLATION?

(a) 100 - 1.000

(b) 1.000 - 10.000

(c) 10.000 - 100.000

(d) > 100.000

SCHEMA MATCHING: IDEAS?

ID	Name	Address	Zip	State	City	Phone	E-Mail
1	Tim Kraska	135 Watermann St,	02906	Providence	RI	+1 234234 234	Tim_kraska@brown.edu
…	…	…				..	..

ID	Name	Addr	Mobile	E-Mail
1	Tim Kraska	135 Watermann St, 02906 Providence, RI	+1 234234 234	Tim_kraska@brown.edu
…	…	…	..	..

ID	Name
1	Tim Kraska
…	…

Address ID	Person-ID	Address	Phone-Nb	E-Mail
1	1	135 Watermann St, 02906 Providence	+1 234234 234	Tim_kraska@brown.edu
1	1	222 Hope St, 02906 Providence	980 – 0803284	tim_kraska@brown.edu

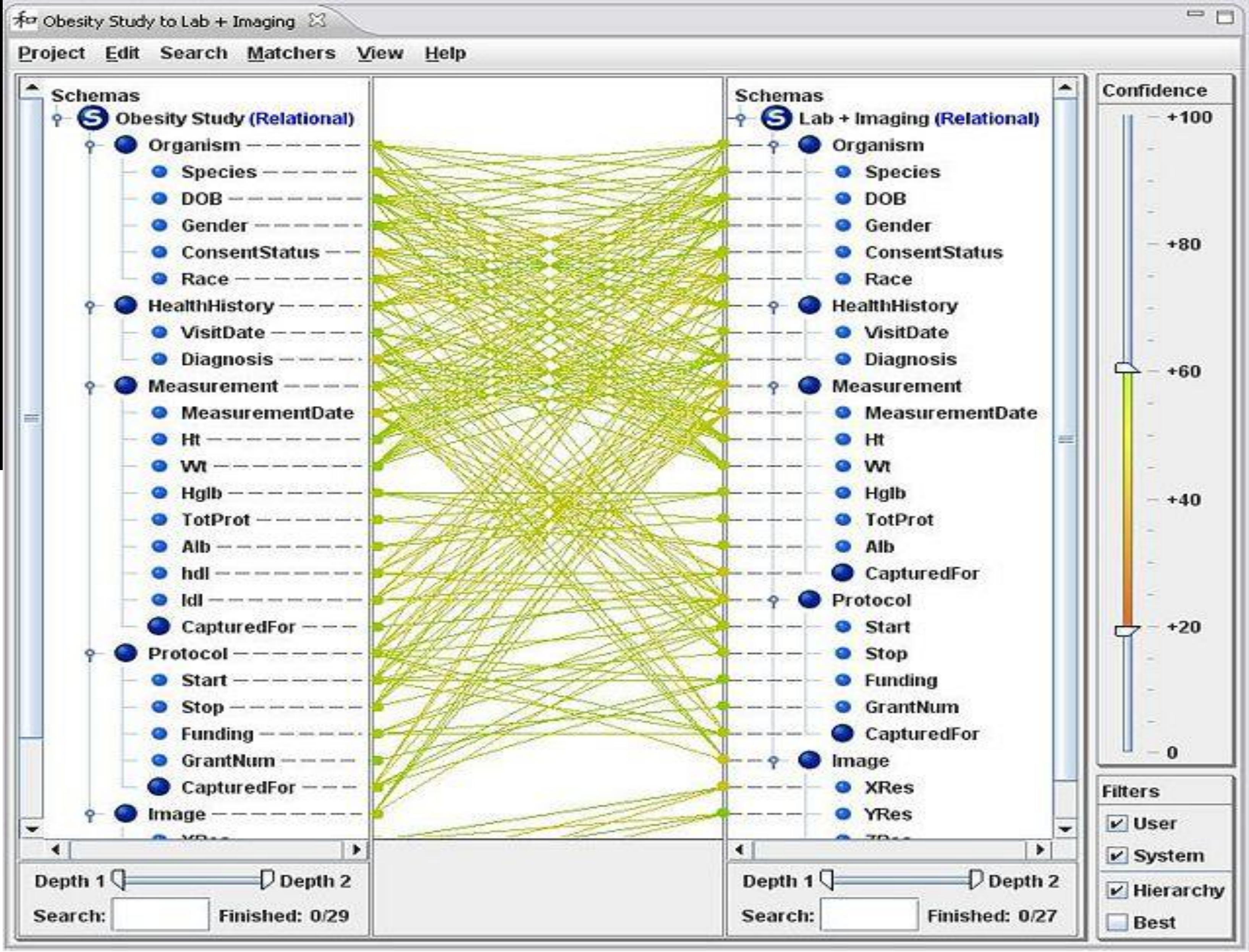
SCHEMA MATCHING - TECHNIQUES

- **Instance vs Schema**: consider instance data or schema information.
- **Element vs Structure**: matching performed for individual schema element (attribute), or for combinations of elements (structure).
- **Language vs Constraint**: use linguistic information (names and textual description) or constraint information (key, relationship)
- **Matching Cardinality**: the overall match result may relate one or more elements of one schema to one or more elements of the other (1:1, 1:n, m:n).
- **Auxiliary Information**: the use of auxiliary information (dictionaries, pervious matching results, user input,..)
- Semantic difference are especially problematic (e.g., Price in Euro vs Dollar, Fahrenheit vs Celsius, etc.)

OpenII

Open Source Tools for Faster Data Integration

OpenII



Obesity Study to Lab + Imaging
Project Edit Search Matchers View Help
Schemas
Obesity Study (Relational)
Organism
Species
DOB
Gender
ConsentStatus
Race
HealthHistory
VisitDate
Diagnosis
Measurement
MeasurementDate
Ht
Wt
Hglb
TotProt
Alb
hdl
ldl
CapturedFor
Protocol
Start
Stop
Funding
GrantNum
CapturedFor
Image
Depth 1
Depth 2
Search:
Finished: 0/29
Schemas
Lab + Imaging (Relational)
Organism
Species
DOB
Gender
ConsentStatus
Race
HealthHistory
VisitDate
Diagnosis
Measurement
MeasurementDate
Ht
Wt
Hglb
TotProt
Alb
CapturedFor
Protocol
Start
Stop
Funding
GrantNum
CapturedFor
Image
XRes
YRes
Depth 1
Depth 2
Search:
Finished: 0/27
Confidence
+100
+80
+60
+40
+20
0
Filters
User
System
Hierarchy
Best

DATA INTEGRATION

Schema Alignment

deduplication, entity clustering, merge/purge, fuzzy match, record linkage, approximate match...

EXAMPLE

ID	Product Name	Price
r1	iPad Two 16GB WiFi White	$490
r2	iPad 2nd generation 16GB WiFi White	$469
r3	iPhone 4th generation White 16GB	$545
r4	Apple iPhone 4 16GB White	$520
r5	Apple iPhone 3rd generation Black 16GB	$375
r6	iPhone 4 32GB White	$599
r7	Apple iPad2 16GB WiFi White	$499
r8	Apple iPod shuffle 2GB Blue	$49
r9	Apple iPod shuffle USB Cable	$19

REAL WORLD DATA

What is wrong here?

Id	Name	Street	City	State	P-Code	Age
1	J Smith	123 University Ave	Seattle	Washington	98106	42
2	Mary Jones	245 3rd St	Redmond	WA	98052-1234	30
3	Bob Wilson	345 Broadway	Seattle	Washington	98101	19
4	M Jones	245 Third Street	Redmond	NULL	98052	299
5	Robert Wilson	345 Broadway St	Seattle	WA	98101	19
6	James Smith	123 Univ Ave	Seatle	WA	NULL	41
7	J Widom	123 University Ave	Palo Alto	CA	94305	NULL
…	…	…	…	…	…	…

REAL WORLD DATA

Duplicate Records

Inconsistent representation

Customer

Id	Name	Street	City	State	P-Code	Age
1	J Smith	123 University Ave	Seattle	Washington	98106	42
2	Mary Jones	245 3rd St	Redmond	WA	98052-1234	30
3	Bob Wilson	345 Broadway	Seattle	Washington	98101	19
4	M Jones	245 Third Street	Redmond	NULL	98052	299
5	Robert Wilson	345 Broadway St	Seattle	WA	98101	19
6	James Smith	123 Univ Ave	Seatle	WA	NULL	41
7	J Widom	123 University Ave	Palo Alto	CA	94305	NULL
…	…	…	…	…	…	…

Typos

Missing Information

REAL WORLD DATA

- How many customers do I have?

```
select count(*)
from customer
```

Wrong answer because of duplicate records!

- How many customers by state?

```
select count(*)
from customer
group by state
```

State	Count
AL	60
…	…
…	…
WA	1200
Washington	50
Wasington	2

What about if you give this data to a ML algorithm?