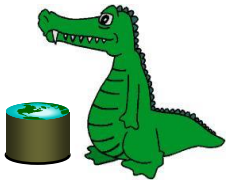
CAP4770/5771
Introduction to Data Science
Fall 2016

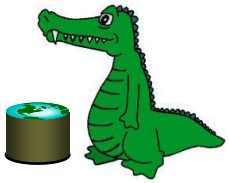
University of Florida, CISE Department
Prof. Daisy Zhe Wang

Based on notes from CS194 at UC Berkeley by Michael Franklin, John Canny, and Jeff Hammerbacher



Logistics

- Lab 2 Homework
  - Difficulty? Deadline?
- Lab 2 Quiz
  - Difficulty? Setup?
- Lab 3 (coming Monday)
- Midterm dates (Oct 10)



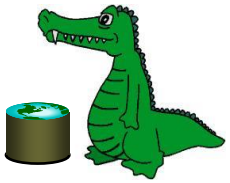
Review

- Exploratory Data Analysis
  - Sample Statistics
  - Distributions and CLT
  - Hypothesis Testing (Null/Alternative Hypothesis, Test Statistic, Sampling Dist…)
  - Visualizations for basic variable types
- More details on Hypothesis testing
  - Variance
  - Power
  - Z-score vs. P-value

Tabular Data Manipulation and Integration

Tabular Data Manipulation

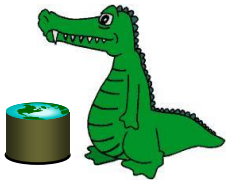
Data Integration



SQL - A language for Relational DBs*

- SQL = Structured Query Language
- Data Definition Language (DDL)
  - create, modify, delete relations
  - specify constraints
  - administer users, security, etc.
- Data Manipulation Language (DML)
  - Specify queries to find tuples that satisfy criteria
  - add, modify, remove tuples
- The DBMS is responsible for efficient evaluation.

* Developed at IBM by Donald D. Chamberlin and Raymond F. Boyce in the 1970s.
Used to be *SEQUEL* (*Structured English QUEry Language*)



Creating Relations in SQL

- Create the Students relation.
  - Note: the type (domain) of each field is specified, and enforced by the DBMS whenever tuples are added or modified.

```
CREATE TABLE Students
       (sid CHAR(20),
        name CHAR(20),
        login CHAR(10),
        age INTEGER,
        gpa FLOAT)
```

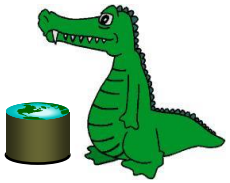


Table Creation (continued)

- Another example: the Enrolled table holds information about courses students take.

```
CREATE TABLE Enrolled
      (sid CHAR(20),
       cid CHAR(20),
       grade CHAR(2))
```

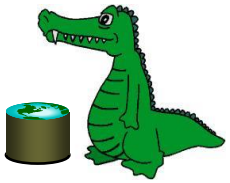
Adding and Deleting Tuples

- Can insert a single tuple using:

```
INSERT INTO Students (sid, name, login, age, gpa)
 VALUES ('53688', 'Smith', 'smith@ee', 18, 3.2)
```

- Can delete all tuples satisfying some condition (e.g., name = Smith):

```
DELETE
  FROM Students S
 WHERE S.name = 'Smith'
```



Queries in SQL

- Single-table queries are straightforward.
- To find all 18 year old students, we can write:

```
SELECT *
  FROM Students S
 WHERE S.age=18
```

- To find just names and logins, replace the first line:

```
SELECT S.name, S.login
```

Joins and Inference

- Chaining relations together is the basic inference method in relational DBs. It produces new relations (effectively new facts) from the data:

```
SELECT S.name, M.mortality
 FROM Students S, Mortality M
 WHERE S.Race=M.Race
```

S

Name	Race
Socrates	Man
Thor	God
Barney	Dinosaur
Blarney stone	Stone

M

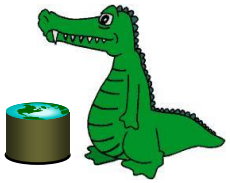
Race	Mortality
Man	Mortal
God	Immortal
Dinosaur	Mortal
Stone	Non-living

Joins and Inference

- Chaining relations together is the basic inference method in relational DBs. It produces new relations (effectively new facts) from the data:

```
SELECT S.name, M.mortality
 FROM Students S, Mortality M
 WHERE S.Race=M.Race
```

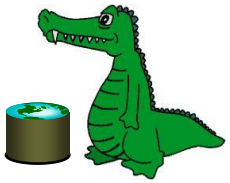
Name	Mortality
Socrates	Mortal
Thor	Immortal
Barney	Mortal
Blarney stone	Non-living



Basic SQL Query

```
SELECT      [DISTINCT]  target-list
FROM        relation-list
WHERE       qualification
```

- *relation-list* : A list of relation names
  - possibly with a *range-variable* after each name
- *target-list* : A list of attributes of tables in *relation-list*
- *qualification* : Comparisons combined using AND, OR and NOT.
  - Comparisons are Attr *op* const or Attr1 *op* Attr2, where *op* is one of =≠<>≤≥
- *DISTINCT*: optional keyword indicating that the answer should not contain duplicates.
  - In SQL SELECT, the default is that duplicates are *not* eliminated! (Result is called a "multiset")



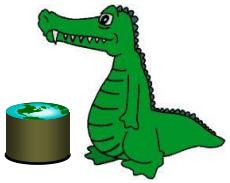
SQL Query Semantics

Semantics of an SQL query are defined in terms of the following conceptual evaluation strategy:

1. do FROM clause: compute *cross-product* of tables (e.g., Students and Enrolled).
2. do WHERE clause: Check conditions, discard tuples that fail. (i.e., "selection").
3. do SELECT clause: Delete unwanted fields. (i.e., "projection").
4. If DISTINCT specified, eliminate duplicate rows.

Probably the least efficient way to compute a query!

- An optimizer will find more efficient strategies to get the *same answer*.



SQL Inner Joins

```
SELECT S.name, E.classid
  FROM Students S (INNER) JOIN Enrolled E
  ON S.sid=E.sid
```

S

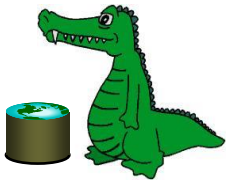
S.name	S.sid
Jones	11111
Smith	22222
Brown	33333

E

E.sid	E.classid
11111	History105
11111	DataScience194
22222	French150
44444	English10

S.name	E.classid
Jones	History105
Jones	DataScience194
Smith	French150

Note the previous version of this query (with no join keyword) is an “Implicit join”



SQL Inner Joins

```
SELECT S.name, E.classid
  FROM Students S (INNER) JOIN Enrolled E
  ON S.sid=E.sid
```

S

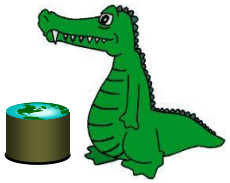
S.name	S.sid
Jones	11111
Smith	22222
Brown	33333

E

E.sid	E.classid
11111	History105
11111	DataScience194
22222	French150
44444	English10

S.name	E.classid
Jones	History105
Jones	DataScience194
Smith	French150

Unmatched keys



What kind of Join is this?

```
SELECT S.name, E.classid
  FROM Students S ?? Enrolled E
  ON S.sid=E.sid
```

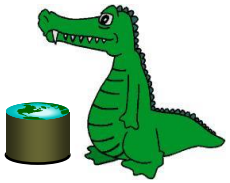
S

S.name	S.sid
Jones	11111
Smith	22222
Brown	33333

E

E.sid	E.classid
11111	History105
11111	DataScience194
22222	French150
44444	English10

S.name	E.classid
Jones	History105
Jones	DataScience194
Smith	French150
Brown	NULL



SQL Joins

```
SELECT S.name, E.classid
  FROM Students S LEFT OUTER JOIN Enrolled E
  ON S.sid=E.sid
```

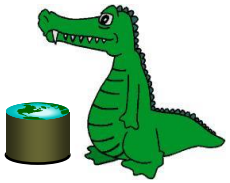
S

S.name	S.sid
Jones	11111
Smith	22222
Brown	33333

E

E.sid	E.classid
11111	History105
11111	DataScience194
22222	French150
44444	English10

S.name	E.classid
Jones	History105
Jones	DataScience194
Smith	French150
Brown	NULL



What kind of Join is this?

```
SELECT S.name, E.classid
  FROM Students S ?? Enrolled E
  ON S.sid=E.sid
```

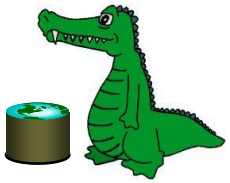
S

S.name	S.sid
Jones	11111
Smith	22222
Brown	33333

E

E.sid	E.classid
11111	History105
11111	DataScience194
22222	French150
44444	English10

S.name	E.classid
Jones	History105
Jones	DataScience194
Smith	French150
NULL	English10



SQL Joins

```
SELECT S.name, E.classid
  FROM Students S RIGHT OUTER JOIN Enrolled E
  ON S.sid=E.sid
```

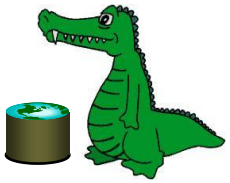
S

S.name	S.sid
Jones	11111
Smith	22222
Brown	33333

E

E.sid	E.classid
11111	History105
11111	DataScience194
22222	French150
44444	English10

S.name	E.classid
Jones	History105
Jones	DataScience194
Smith	French150
NULL	English10



SQL Joins

```
SELECT S.name, E.classid
  FROM Students S ? JOIN Enrolled E
  ON S.sid=E.sid
```

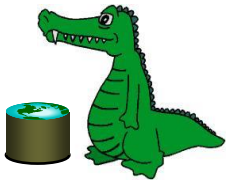
S

S.name	S.sid
Jones	11111
Smith	22222
Brown	33333

E

E.sid	E.classid
11111	History105
11111	DataScience194
22222	French150
44444	English10

S.name	E.classid
Jones	History105
Jones	DataScience194
Smith	French150
NULL	English10
Brown	NULL



SQL Joins

```
SELECT S.name, E.classid
  FROM Students S FULL OUTER JOIN Enrolled E
  ON S.sid=E.sid
```

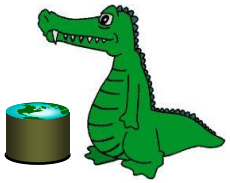
S

S.name	S.sid
Jones	11111
Smith	22222
Brown	33333

E

E.sid	E.classid
11111	History105
11111	DataScience194
22222	French150
44444	English10

S.name	E.classid
Jones	History105
Jones	DataScience194
Smith	French150
NULL	English10
Brown	NULL



What kind of Join is this?

```
SELECT S.name, E.classid
  FROM Students S ?? Enrolled E
  ON S.sid=E.sid
```

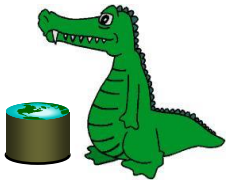
S

S.name	S.sid
Jones	11111
Smith	22222
Brown	33333

S.name	S.Sid
Jones	11111
Smith	22222

E

E.sid	E.classid
11111	History105
11111	DataScience194
22222	French150
44444	English10



SQL Joins

```
SELECT S.name, S.Sid
  FROM Students S LEFT SEMI JOIN Enrolled E
  ON S.sid=E.sid
```

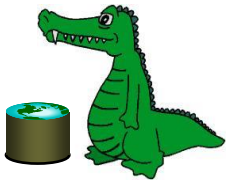
S

S.name	S.sid
Jones	11111
Smith	22222
Brown	33333

S.name	S.Sid
Jones	11111
Smith	22222

E

E.sid	E.classid
11111	History105
11111	DataScience194
22222	French150
44444	English10



What kind of Join is this?

```
SELECT *
  FROM Students S ?? Enrolled E
```

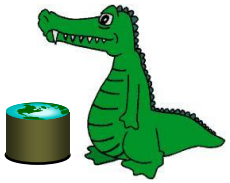
S

S.name	S.sid
Jones	11111
Smith	22222

E

E.sid	E.classid
11111	History105
11111	DataScience194
22222	French150

S.name	S.sid	E.sid	E.classid
Jones	11111	11111	History105
Jones	11111	11111	DataScience194
Jones	11111	22222	French150
Smith	22222	11111	History105
Smith	22222	11111	DataScience194
Smith	22222	22222	French150



SQL Joins

```
SELECT *
  FROM Students S CROSS JOIN Enrolled E
```

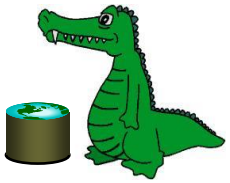
S

S.name	S.sid
Jones	11111
Smith	22222

E

E.sid	E.classid
11111	History105
11111	DataScience194
22222	French150

S.name	S.sid	E.sid	E.classid
Jones	11111	11111	History105
Jones	11111	11111	DataScience194
Jones	11111	22222	French150
Smith	22222	11111	History105
Smith	22222	11111	DataScience194
Smith	22222	22222	French150



What kind of Join is this?

```
SELECT *
  FROM Students S, Enrolled E
 WHERE S.sid <= E.sid
```

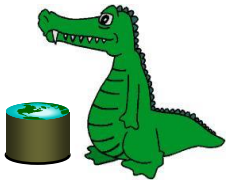
S

S.name	S.sid
Jones	11111
Smith	22222

E

E.sid	E.classid
11111	History105
11111	DataScience194
22222	French150

S.name	S.sid	E.sid	E.classid
Jones	11111	11111	History105
Jones	11111	11111	DataScience194
Jones	11111	22222	French150
Smith	22222	22222	French150



Theta Joins

```
SELECT *
  FROM Students S, Enrolled E
 WHERE S.sid <= E.sid
```

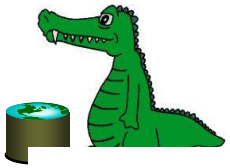
S

S.name	S.sid
Jones	11111
Smith	22222

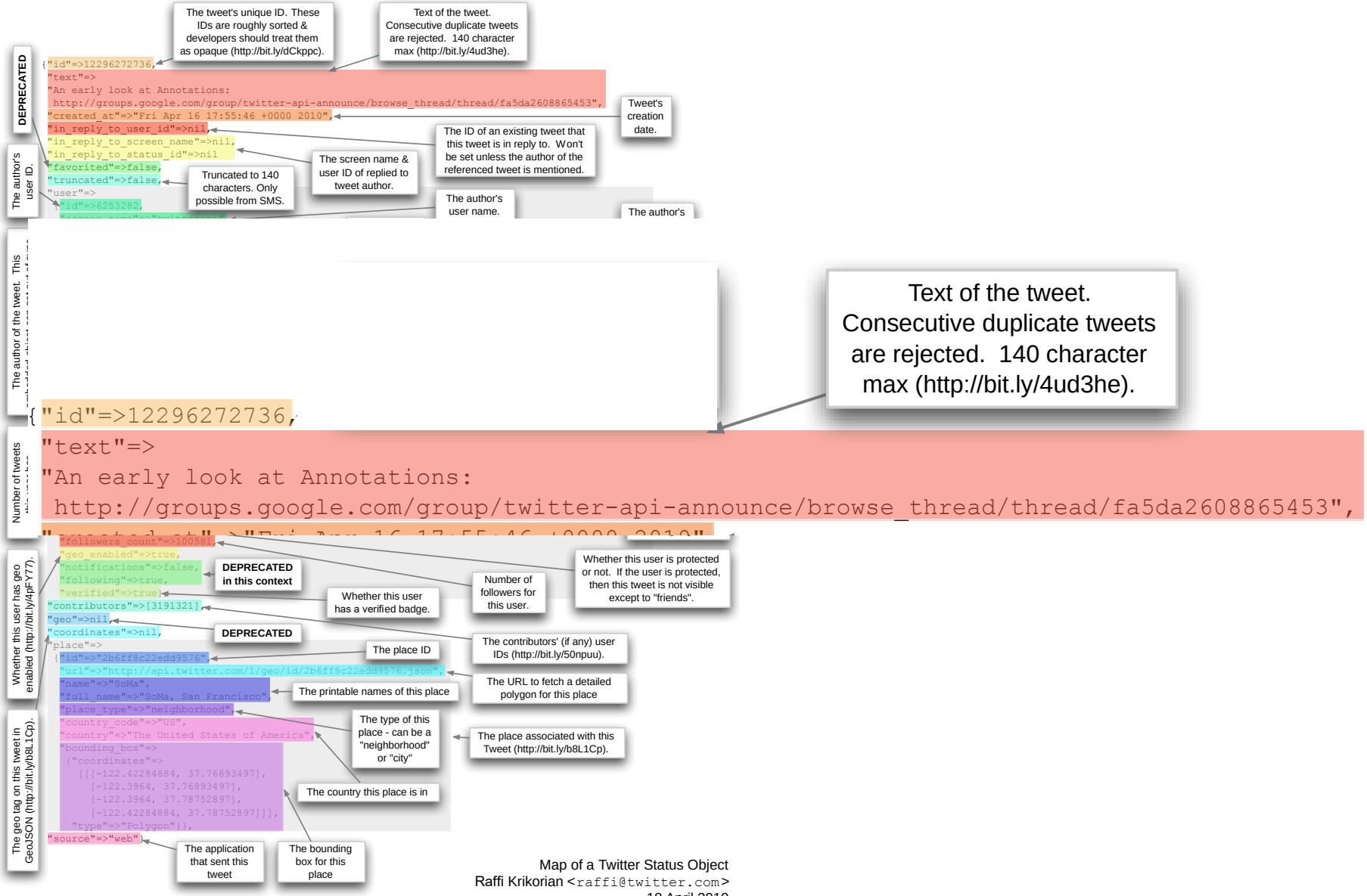
E

E.sid	E.classid
11111	History105
11111	DataScience194
22222	French150

S.name	S.sid	E.sid	E.classid
Jones	11111	11111	History105
Jones	11111	11111	DataScience194
Jones	11111	22222	French150
Smith	22222	22222	French150



Recall: Tweet JSON Format



Normalization

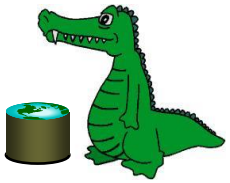
Raw twitter data storage is very inefficient because, e.g. user records are repeated with every tweet by that user.

Normalization is the process of minimizing data redundancy.

Tweet id	User id	Location id	Body
11	111	1111	I need a Jamba juice
22	111	1111	Cal Soccer rules
33	111	2222	Why do we procrastinate?
44	222	3333	Close your eyes and push "go"

User.id	Name	Attribs…
111	Jones	
222	Smith	

Loc.id	Name	Attribs…
1111	Berkeley	
2222	Oakland	
3333	Hayward	



Normalization

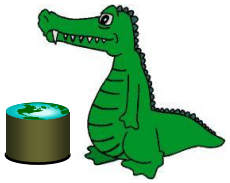
Normalized tables include only a foreign key to the information in another table for repeated data.

The original table is the result of inner joins between tables.

Tweet id	User id	Location id	Body
11	111	1111	I need a Jamba juice
22	111	1111	Cal Soccer rules
33	111	2222	Why do we procrastinate?
44	222	3333	Close your eyes and push "go"

User.id	Name	Attribs…
111	Jones	
222	Smith	

Loc.id	Name	Attribs…
1111	Berkeley	
2222	Oakland	
3333	Hayward	



Aggregate Queries

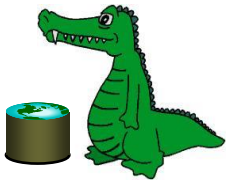
Including reference counts in the lookup tables allows you to perform aggregate queries on those tables alone:

Average age of users, most popular location,…

Tweet id	User id	Location id	Body
11	111	1111	I need a Jamba Juice
22	111	1111	Cal Soccer rules
33	111	2222	Why do we procrastinate?
44	222	3333	Close your eyes and push "go"

U.id	Name	Count	Attr..
111	Jones	3	
222	Smith	1	

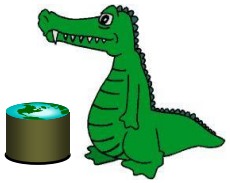
L.id	Name	Count	Attr…
1111	Berkeley	2	
2222	Oakland	1	
3333	Hayward	1	



Reductions and GroupBy

- One of the most common operations on Data Tables is aggregation or reduction (count, sum, average, min, max,…).
- They provide a means to see high-level patterns in the data, to make summaries of it etc.
- You need ways of specifying which columns are being aggregated over, which is the role of a GroupBy operator.

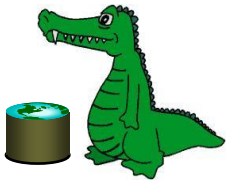
SID	Name	Course	Semester	Grade	GPA
111	Jones	Stat 134	F13	A	4.0
111	Jones	CS 162	F13	B-	2.7
222	Smith	EE 141	S14	B+	3.3
222	Smith	CS162	F14	C+	2.3
222	Smith	CS189	F14	A-	3.7



Reductions and GroupBy

SID	Name	Course	Semester	Grade	GPA
111	Jones	Stat 134	F13	A	4.0
111	Jones	CS 162	F13	B-	2.7
222	Smith	EE 141	S14	B+	3.3
222	Smith	CS162	F14	C+	2.3
222	Smith	CS189	F14	A-	3.7

```
SELECT SID, Name, AVG(GPA)
 FROM Students
 GROUP BY SID
```



Reductions and GroupBy

SID	Name	Course	Semester	Grade	GPA
111	Jones	Stat 134	F13	A	4.0
111	Jones	CS 162	F13	B-	2.7
222	Smith	EE 141	S14	B+	3.3
222	Smith	CS162	F14	C+	2.3
222	Smith	CS189	F14	A-	3.7

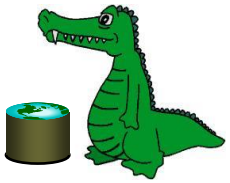
```
SELECT SID, Name, AVG(GPA)
 FROM Students
 GROUP BY SID
```

SID	Name	GPA
111	Jones	3.35
222	Smith	3.1

Pandas/Python

- **Series**: a named, ordered dictionary
  - The keys of the dictionary are the **indexes**
  - Built on NumPy's **ndarray**
  - Values can be any Numpy data type object

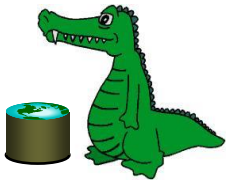
- **DataFrame**: a table with named columns
  - Represented as a Dict (col_name -> series)
  - Each Series object represents a column



Relational operations in Python

- Selection/Filtering (apply predicate to rows)
- Projection (select certain attributes)
- aggregate: sum, count, average, max, min
- sort/group by

- merge/concat:
  - union, intersection, difference, cartesian product (CROSS JOIN)
  - join: natural join (INNER JOIN), theta join, semi-join, etc.
- pivot/reshape



Pandas vs SQL

+ Pandas is lightweight and fast.

+ Full SQL expressiveness plus the expressiveness of Python, especially for function evaluation.

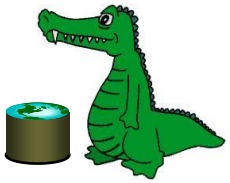
+ Integration with plotting functions like Matplotlib.

- Tables must fit into memory.

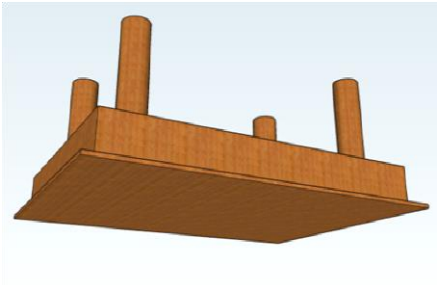
- No post-load indexing functionality: indices are built when a table is created.

- No transactions, journaling, etc.

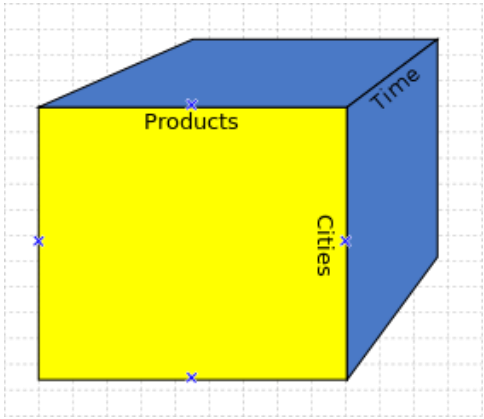
- Large, complex joins probably slower.

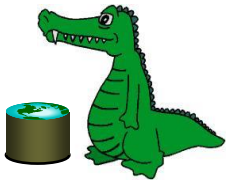


The other view of tables: OLAP



- OnLine Analytical Processing
- Conceptually like an n-dimensional spreadsheet (Cube)
- Columns become dimensions (discrete)
- The goal: live interaction with numerical data for business intelligence

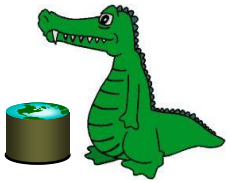




The other view of tables: OLAP

From a table to a cube:

name	classid	Semester	Grade	Units
Jones	History105	F13	3.3	4.0
Jones	DataScience194	S12	4.0	3.0
Jones	French150	F14	3.7	4.0
Smith	History105	S15	2.3	3.0
Smith	DataScience194	F14	2.7	3.0
Smith	French150	F13	3.0	4.0



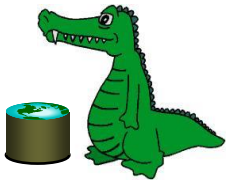
From tables to OLAP cubes

From a table to a cube: (attributes types)

name	classid	Semester	Grade	Units
Jones	History105	F13	3.3	4.0
Jones	DataScience194	S12	4.0	3.0
Jones	French150	F14	3.7	4.0
Smith	History105	S15	2.3	3.0
Smith	DataScience194	F14	2.7	3.0
Smith	French150	F13	3.0	4.0

Variables used as qualifiers
(In where, GroupBy clauses)
Normally discrete

Variables we want to measure
Normally numeric

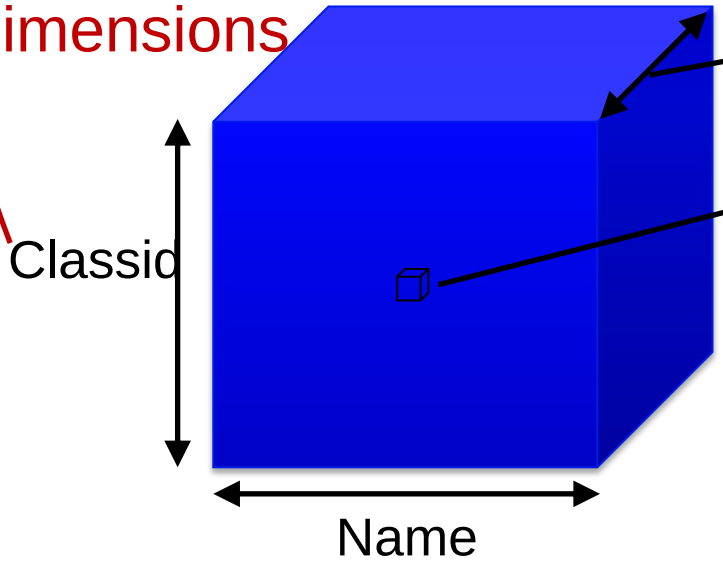


Constructing OLAP cubes

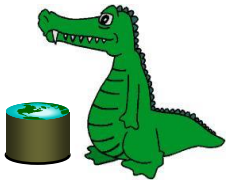
name	classid	Semester	Grade	Units
Jones	History105	F13	3.3	4.0
Jones	DataScience194	S12	4.0	3.0
…	…	…	…	…

Cube dimensions

Cube values

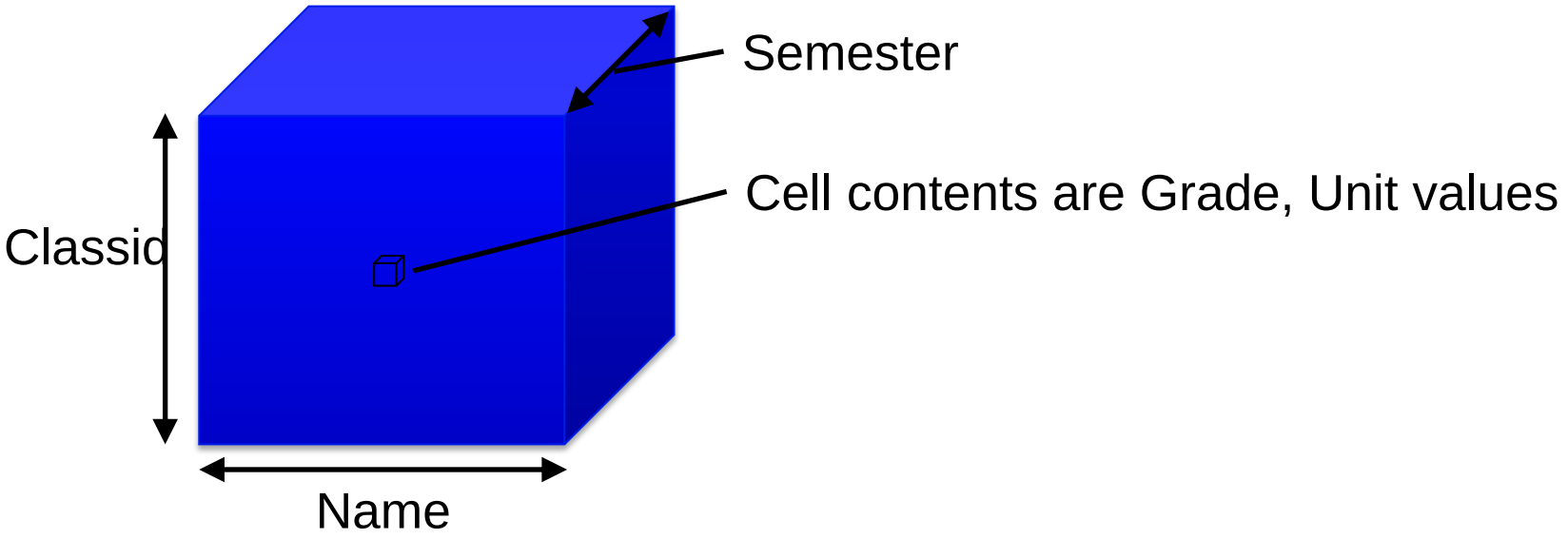


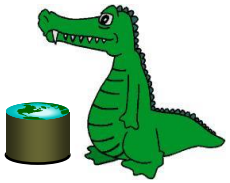
Cell contents are Grade, Unit values



Queries on OLAP cubes

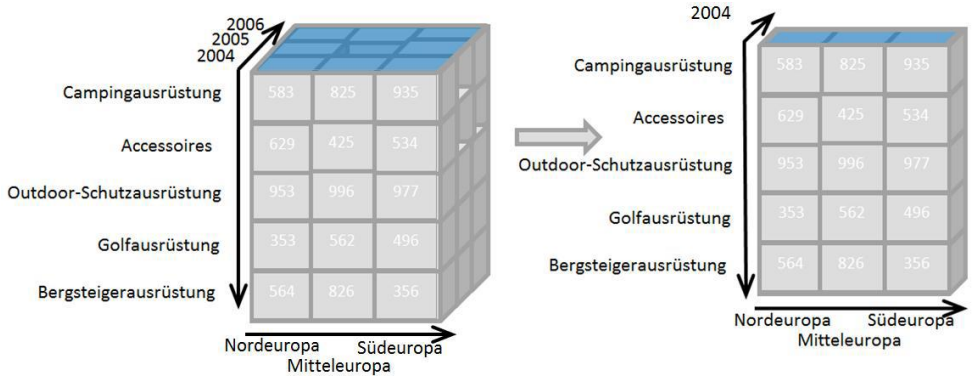
- Once the cube is defined, its easy to do aggregate queries by projecting along one or more axes.
- E.g. to get student GPAs, we project the Grade field onto the student (Name) axis.
- In fact, such aggregates are precomputed and maintained automatically in an OLAP cube, so queries are instantaneous.



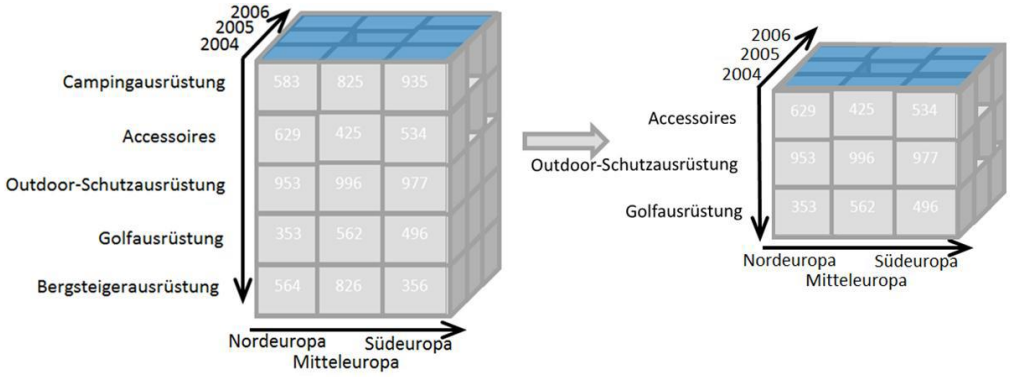


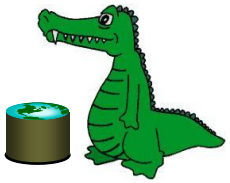
OLAP

- Slicing:
  fixing one or
  more variables



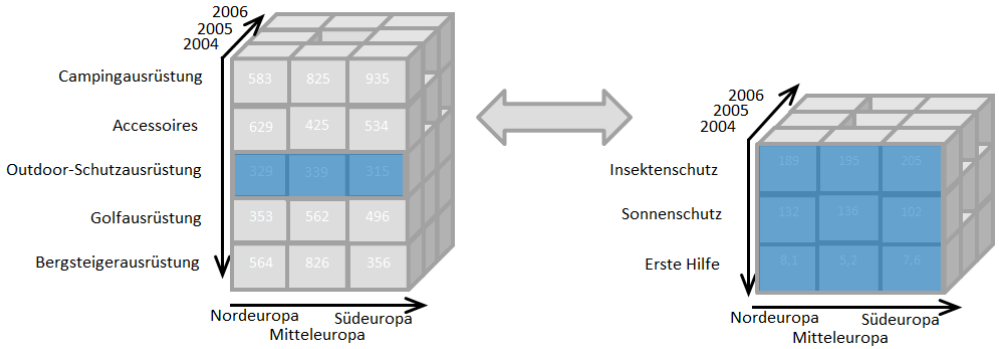
- Dicing:
  selecting a range
  values for one or
  more variables



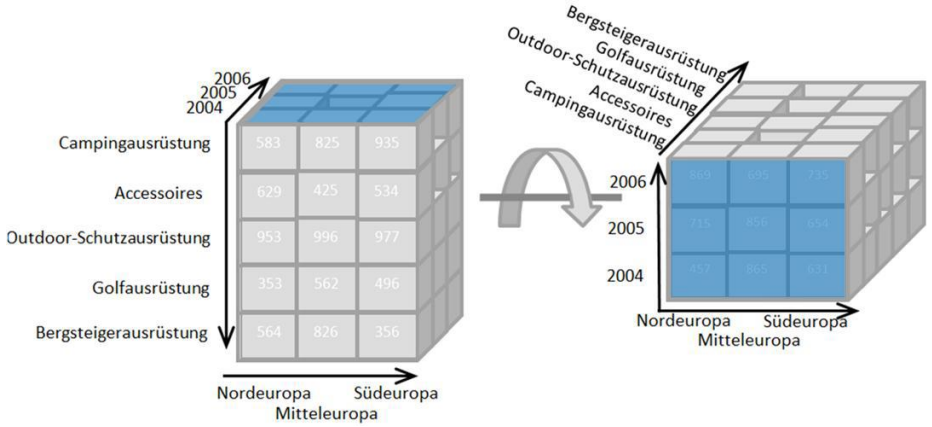


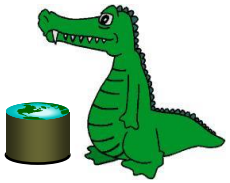
OLAP

- Drilling Up/Down (change levels of a hierarchically-indexed variable)



- Pivoting: produce a two-axis view for viewing as a spreadsheet.





Hierarchical-indexed variables

- To support real-time querying, OLAP DBs store *aggregates* of data values along many dimensions.
- This works best if axes can be tree-structured. E.g time can be expressed as a hierarchy

hour ← day ← week ← month ← year