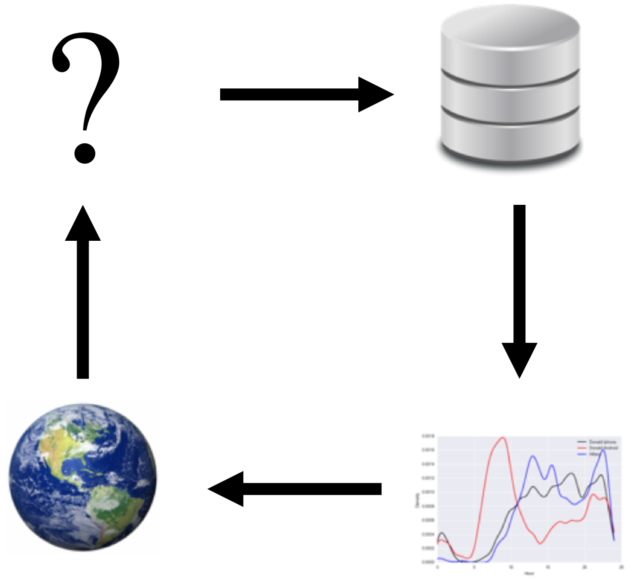
Lecture 11: Finish Web Technologies & Begin SQL Databases

Slides by:

Joseph E. Gonzalez

jegonzal@berkeley.edu



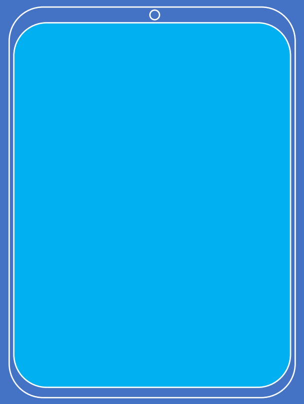
Last **Two** Lectures

- **Last Thursday:** String manipulation & Regular Expressions
  - guest lecture from the amazing Sam Lau
  - reviewed in section and in future labs & HWs
- **Last Tuesday:** HTTP, XML, and JSON
  - Pandas web tables support
  - Using the browser developer mode
  - JSON and basics of XML
  - Started HTTP request/response protocol and GET vs POST
  - Didn't finish REST and web-services …

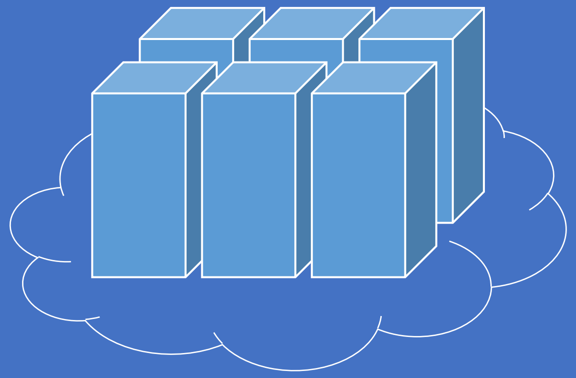
REST APIs

Example:

GET	/website/images	Get all images
POST	/website/images	Add an image
GET	/website/images/{id}	Get a an image
PUT	/website/images/{id}	Update an image
DELETE	/website/images/{id}	Delete an image



Client



Server

REST – Representational State Transfer

- A way of architecting widely accessible, efficient, and extensible web services (typically using HTTP)
- **Client-Server:** client and server are able to evolve independently
- **Stateless:** The server does not store any of the clients session state
- **Cacheable:** system should clearly define what functionality can be cached (e.g., GET vs POST requests)
- **Uniform Interface:** provide a consistent interface for getting and updating data in a system

Demo

TwitterAPI_REST_Example.ipynb

Scraping Ethics

- Don't violate terms of use for the service or data
- Scraping can cause result in degraded services for others
  - Many services are optimized for human user access patterns
  - Requests can be parallelized/distributed to saturate server
  - Each query may result in many database requests
- How to scrape ethically:
  - Used documented REST APIs – read terms of service
  - Examine at *robots.txt* (e.g., https://en.wikipedia.org/robots.txt)
  - Throttle request rates (sleep)
- Avoid getting Berkeley (or your organization) blocked from websites & services

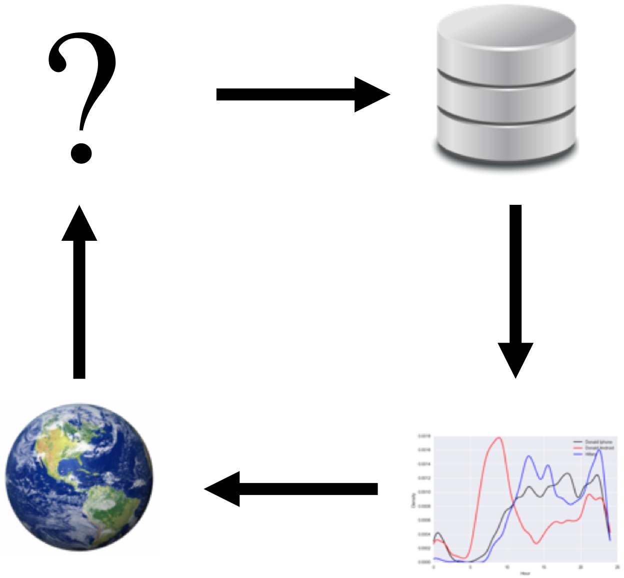
Databases and SQL
Part 1

Slides by:

Joseph E. Gonzalez & Joseph Hellerstein,

jegonzal@berkeley.edu

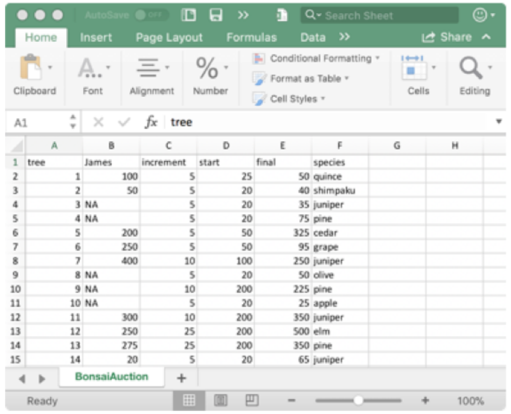
jhellerstein@berkeley.edu

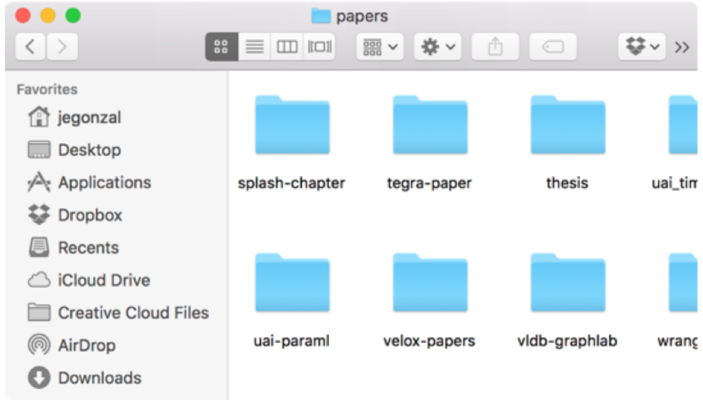


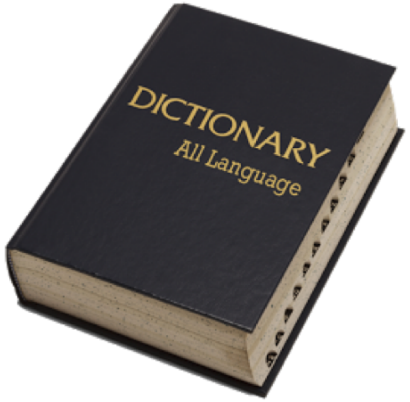
What is a database?

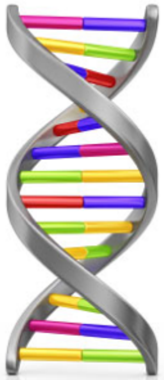
Defining Databases

- A **database** is an organized collection of data.









- A **database management systems (DBMS)** is a software system that **stores**, **manages**, and **facilitates access** to one or more databases.

Database Management Systems

- **Data storage**
  - Provide **reliable storage** to survive system crashes and disk failures
  - Special data-structures to **improve performance**
- **Data management**
  - Configure how data is **logically organized** and **who has access**
  - Ensure data **consistency properties** (e.g., positive bank account values)
- **Facilitate access**
  - Enable **efficient access** to the data
  - Supports user defined **computation** (queries) over data

Is **Pandas** a *Database Management System?*

- Data Storage?
  - Pandas doesn’t store data, this is managed by the filesystem
- Data Management?
  - Pandas does support changing the organization of data but doesn’t manage who can access the data
- Facilitate Access?
  - Pandas does support rich tools for computation over data

- Pandas is not generally considered a database management system but it often interacts with DBMSs

Why should I use a DBMS?

Why can't I just have my CSV files?

- DBMSs organize many related sources of information
- DBMSs enforce guarantees on the data
  - *Can be* used to **prevent data anomalies**
  - Ensure **safe concurrent operations** on data
- DBMSs can be **scalable**
  - Optimized to compute on data that **does not fit in memory**
  - **Parallel** computation and **optimized data structures**
- DBMSs **prevent data loss** from software/hardware **failures**

Widely Used DBMS Technologies

Common DBMS Systems

https://db-engines.com/en/ranking

Rank			DBMS	Database Model	Score		
Oct 2017	Sep 2017	Oct 2016			Oct 2017	Sep 2017	Oct 2016
1.	1.	1.	Oracle	Relational DBMS	1348.80	-10.29	-68.30
2.	2.	2.	MySQL	Relational DBMS	1298.83	-13.78	-63.82
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1210.32	-2.23	-3.86
4.	4.	↑ 5.	PostgreSQL	Relational DBMS	373.27	+0.91	+54.58
5.	5.	↓ 4.	MongoDB	Document store	329.40	-3.33	+10.60
6.	6.	6.	DB2	Relational DBMS	194.59	-3.75	+14.03
7.	7.	↑ 8.	Microsoft Access	Relational DBMS	129.45	+0.64	+4.78
8.	8.	↓ 7.	Cassandra	Wide column store	124.79	-1.41	-10.27
9.	9.	9.	Redis	Key-value store	122.05	+1.65	+12.51
10.	10.	↑ 11.	Elasticsearch	Search engine	120.23	+0.23	+21.12

Relational database management systems are widely used!

Relational Database Management Systems

- Relational databases are the traditional DBMS technology

ORACLE MySQL TERADATA Microsoft SQL Server PostgreSQL SQLite

- ***Logically*** organize data in **relations** (tables)

Sales relation:

Name	Prod	Price
Sue	iPod	$200.00
Joey	Bike	$333.99
Alice	Car	$999.00

Tuple (row)

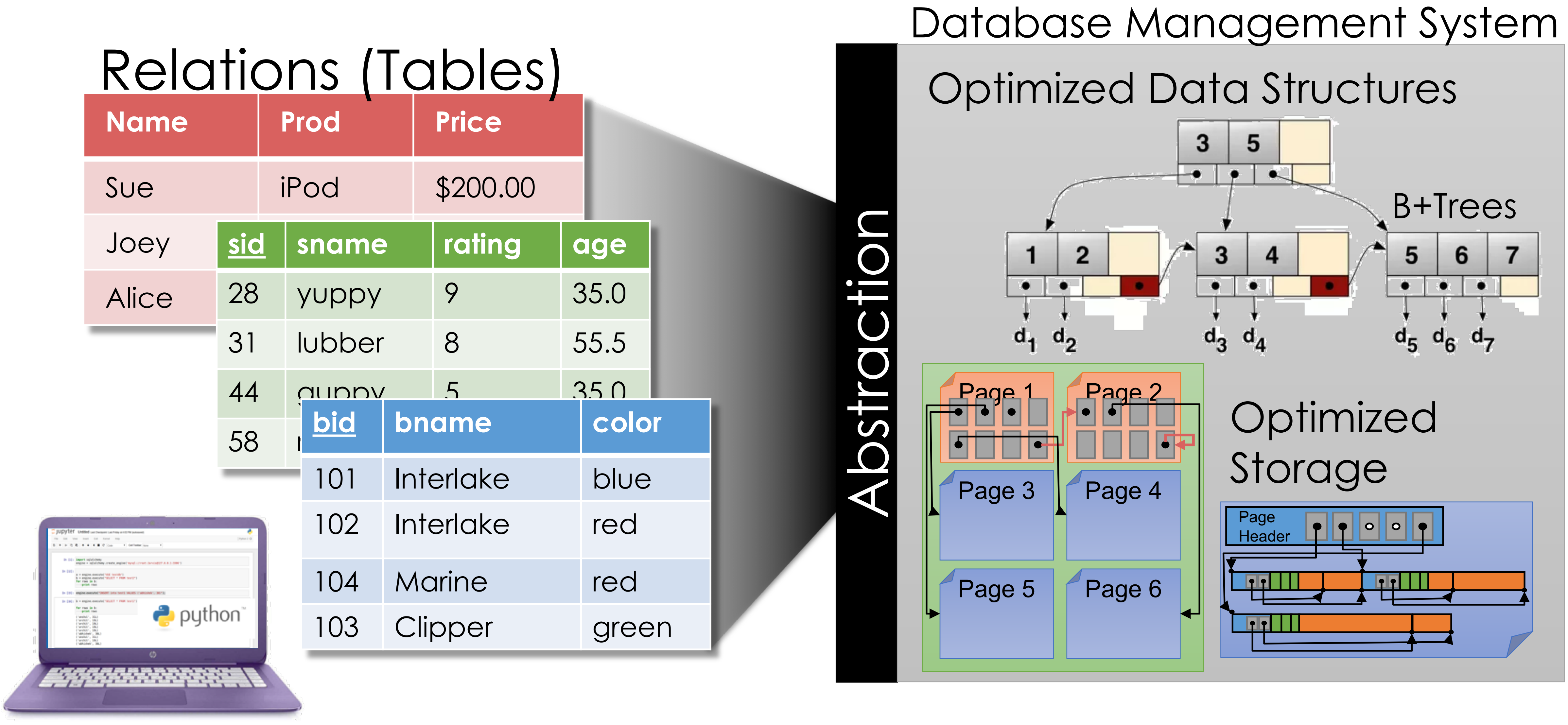
Attribute (column)

Describes *relationship:*

***Name** purchased*
***Prod** at **Price**.*

How is data **physically** stored?

Relational Data **Abstraction**



Physical Data Independence:

Database management systems **hide how data is stored** from end user applications

→ System can **optimize storage** and **computation** without changing applications.

Big Idea in Data Structures
Data Systems &
Computer Science

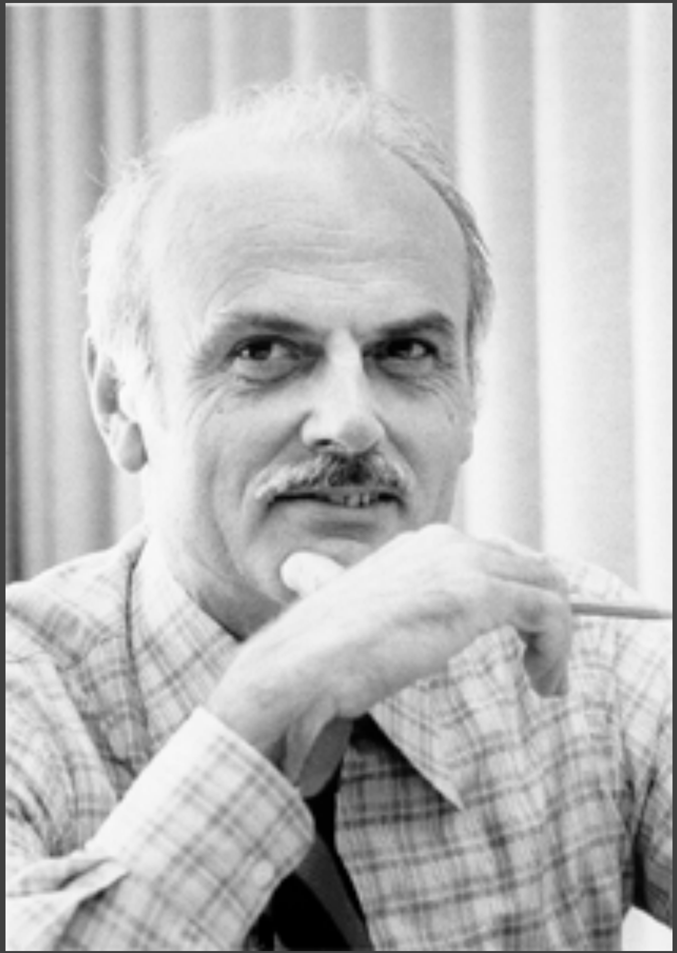
It wasn't always like this …

In a time long ago …

Before 1970's databases
were not routinely organized as tables.

Instead they exposed specialized
data structures designed for
specific applications.

Ted Codd and the ***Relational*** Model



Edgar F. "Ted" Codd (1923 - 2003)
Turing Award 1981

- **[1969] Relational model:** a mathematical abstraction of a database as sets
  - Independence of data from the physical properties of stage storage and representation
- **[1972] Relational Algebra & Calculus:** a collection of operations and a way defining logical outcomes for data transformations
  - **Algebra:** beginning of technologies like Pandas
  - **Calculus:** the foundation of modern SQL

Relational Database Management Systems

- Traditionally DBMS referred to relational databases
- *Logically* organize data in **relations** (tables)
- Structured Query Language **(SQL)** to define, manipulate and compute on data.
  - A common language spoken by many data systems
    - Some variations and deviations from the standard …
  - Describes logical organization of data as well as computation on data.

SQL

What not **How**

SQL is a **Declarative** Language

- **Declarative:** "Say ***what*** you want, not ***how*** to get it."
  - **Declarative Example:** *I want a table with columns "x" and "y" constructed from tables "A" and "B" where the values in "y" are greater than 100.00.*
  - **Imperative Example:** For each record in table "A" find the corresponding record in table "B" then drop the records where "y" is less than or equal to 100 then return the "x" and "y" values.
- **Advantages** of declarative programming
  - Enable the system to find the best way to achieve the result.
  - Often more compact and easier to learn for non-programmers
- **Challenges** of declarative programming
  - System performance depends heavily on automatic optimization
  - Limited language (not Turing complete)

Review of Relational Terminology

- *Database*: Set of Relations (i.e., one or more tables)
- *Relation* (*Table*):
  - *Schema*: description of columns, their types, and constraints
  - *Instance*: data satisfying the schema
- *Attribute* (*Column*)
- *Tuple* (*Record, Row*)
- *Schema of database* is set of schemas of its relations

Two sublanguages of SQL

- DDL – Data Definition Language
  - Define and modify schema
- DML – Data Manipulation Language
  - Queries can be written intuitively.

CAPITALIZATION IS **optional** BUT …
DATABASE PEOPLE PREFER TO YELL

Creating Tables & Populating Tables

CREATE TABLE ...

sid	sname	rating	age
1	Fred	7	22
2	Jim	2	39
3	Nancy	8	27

bid	bname	color
101	Nina	red
102	Pinta	blue
103	Santa Maria	red

sid	**bid**	**day**
1	102	9/12
2	102	9/13

```
CREATE TABLE Sailors (
   sid INTEGER,
   sname CHAR(20),
   rating INTEGER,
   age REAL,
   PRIMARY KEY (sid));
```

Columns have **names** and **types**

Specify **Primary Key** column(s)

```
CREATE TABLE Boats (
   bid INTEGER,
   bname CHAR (20),
   color CHAR(10),
   PRIMARY KEY (bid));
```

Specify **Foreign Key** relationships

```
 CREATE TABLE Reserves (
   sid INTEGER,
   bid INTEGER,
   day DATE,
  PRIMARY KEY (sid, bid, day),
  FOREIGN KEY (sid) REFERENCES Sailors,
  FOREIGN KEY (bid) REFERENCES Boats);
```

Semicolon at end of command

Common SQL Types (there are others...)

- **CHAR(size)**: Fixed number of characters
- **TEXT**: Arbitrary number of character strings
- **INTEGER & BIGINT**: Integers of various sizes
- **REAL & DOUBLE PRECISION**: Floating point numbers
- **DATE & DATETIME**: Date and Date+Time formats

See documentation for database system (e.g., Postgres)

More Creating Tables

```
CREATE TABLE students(
        name TEXT PRIMARY KEY,
        gpa REAL CHECK (gpa >= 0.0 and gpa <= 4.0),
        age INTEGER,
        dept TEXT,
        gender CHAR);
```

Imposing Integrity Constraints

Useful to ensure data quality…

	name	gpa	age	dept	gender
0	Sergey Brin	2.8	40	CS	M
1	Danah Boyd	3.9	35	CS	F
2	Bill Gates	1.0	60	CS	M
3	Hillary Mason	4.0	35	DATASCI	F
4	Mike Olson	3.7	50	CS	M
5	Mark Zuckerberg	4.0	30	CS	M
6	Sheryl Sandberg	4.0	47	BUSINESS	F
7	Susan Wojcicki	4.0	46	BUSINESS	F
8	Marissa Meyer	4.0	45	BUSINESS	F

Inserting Records into a Table

```
INSERT INTO students (name, gpa, age, dept, gender)   ← Optional
VALUES
        ('Sergey Brin', 2.8, 40, 'CS', 'M'),
        ('Danah Boyd', 3.9, 35, 'CS', 'F'),
        ('Bill Gates', 1.0, 60, 'CS', 'M'),
        ('Hillary Mason', 4.0, 35, 'DATASCI', 'F'),
        ('Mike Olson', 3.7, 50, 'CS', 'M'),
        ('Mark Zuckerberg', 4.0, 30, 'CS', 'M'),
        ('‘Sheryl Sandberg', 4.0, 47, 'BUSINESS', 'F'),
        ('Susan Wojcicki', 4.0, 46, 'BUSINESS', 'F'),
        ('Marissa Meyer', 4.0, 45, 'BUSINESS', 'F');

-- This is a comment.
-- Does the order matter? No
```

- Fields must be entered in order (record)
- Comma between records
- Must use the single quote (’) for strings.

Deleting and Modifying Records

- Records are deleted by specifying a condition:

```
DELETE FROM students
    WHERE LOWER(name) = 'sergey brin'
```

String Function

- Modifying records

```
UPDATE students
    SET gpa = 1.0 + gpa
    WHERE dept = ‘CS’;
```

- Notice that there is no way to modify records by location

Deleting and Modifying Records

- What is wrong with the following

```
UPDATE students
       SET gpa = 1.0 + gpa
       WHERE dept = 'CS';
```

```
CREATE TABLE students(
       name TEXT PRIMARY KEY,
       gpa FLOAT CHECK (gpa >= 0.0 and gpa <= 4.0),
       age INTEGER,
       dept TEXT,
       gender CHAR);
```

Update would violate
Integrity Constraints

	name	gpa	age	dept	gender
0	Sergey Brin	2.8	40	CS	M
1	Danah Boyd	3.9	35	CS	F
2	Bill Gates	1.0	60	CS	M
3	Hillary Mason	4.0	35	DATASCI	F
4	Mike Olson	3.7	50	CS	M
5	Mark Zuckerberg	4.0	30	CS	M
6	Sheryl Sandberg	4.0	47	BUSINESS	F
7	Susan Wojcicki	4.0	46	BUSINESS	F
8	Marissa Meyer	4.0	45	BUSINESS	F

Querying Tables

SQL DML:
Basic Single-Table Queries

```
SELECT [DISTINCT] <column expression list>
  FROM <single table>
[WHERE <predicate>]
[GROUP BY <column list>
 [HAVING <predicate>] ]
[ORDER BY <column list>];
```

- Elements of the basic select statement
- [Square brackets] are optional expressions.

Basic Single-Table Queries

```
SELECT [DISTINCT] <column expression list>
  FROM <single table>
[WHERE <predicate>]
[GROUP BY <column list>
 [HAVING <predicate>] ]
[ORDER BY <column list>] ;
```

- Simplest version is straightforward
  - Produce all tuples in the table that **satisfy the predicate**
  - Output the expressions in the **SELECT list**
  - **Expression** can be a **column reference**, or an **arithmetic expression over column refs**

Find the name and GPA for all CS Students

```
SELECT
  FROM
 WHERE
```

name	gpa
Danah Boyd	3.9
Mike Olson	3.7
Mark Zuckerberg	4.0
Bill Gates	2.0

	name	gpa	age	dept	gender
0	Sergey Brin	2.8	40	CS	M
1	Danah Boyd	3.9	35	CS	F
2	Bill Gates	1.0	60	CS	M
3	Hillary Mason	4.0	35	DATASCI	F
4	Mike Olson	3.7	50	CS	M
5	Mark Zuckerberg	4.0	30	CS	M
6	Sheryl Sandberg	4.0	47	BUSINESS	F
7	Susan Wojcicki	4.0	46	BUSINESS	F
8	Marissa Meyer	4.0	45	BUSINESS	F

SELECT DISTINCT

```
SELECT DISTINCT dept
  FROM students
 [WHERE <predicate>]
 [GROUP BY <column list>
 [HAVING <predicate>] ]
[ORDER BY <column list>] ;
```

dept
CS
BUSINESS
DATASCI

	name	gpa	age	dept	gender
0	Sergey Brin	2.8	40	CS	M
1	Danah Boyd	3.9	35	CS	F
2	Bill Gates	1.0	60	CS	M
3	Hillary Mason	4.0	35	DATASCI	F
4	Mike Olson	3.7	50	CS	M
5	Mark Zuckerberg	4.0	30	CS	M
6	Sheryl Sandberg	4.0	47	BUSINESS	F
7	Susan Wojcicki	4.0	46	BUSINESS	F
8	Marissa Meyer	4.0	45	BUSINESS	F

- DISTINCT flag specifies removal of duplicates before output

ORDER BY

```
SELECT name, gpa, age
  FROM students
 WHERE dept = 'CS'
[GROUP BY <column list>
 [HAVING <predicate>] ]
 ORDER BY gpa, name;
```

name	gpa	age
Bill Gates	2.0	60
Mike Olson	3.7	50
Danah Boyd	3.9	35
Mark Zuckerberg	4.0	30

- ORDER BY clause specifies output to be sorted
  - ***Lexicographic*** ordering

ORDER BY

```
SELECT name, gpa, age
  FROM students
 WHERE dept = 'CS'
[GROUP BY <column list>
 [HAVING <predicate>] ]
 ORDER BY gpa DESC, name ASC;
```

name	gpa	age
Mark Zuckerberg	4.0	30
Danah Boyd	3.9	35
Mike Olson	3.7	50
Bill Gates	2.0	60

- Ascending order by default
  - DESC flag for descending, ASC for ascending
  - Can mix and match, lexicographically

Aggregates

```
SELECT AVG(gpa)
  FROM students
 WHERE dept = 'CS'
[GROUP BY <column list>
 [HAVING <predicate>] ]
[ORDER BY <column list>] ;
```

avg
3.4

- Before producing output, compute a summary statistic
  - Aggregates include: SUM, COUNT, MAX, MIN, …
- Produces 1 row of output → **Still a table**
- Note: can use DISTINCT *inside* the agg function
  - SELECT COUNT(DISTINCT name) …

GROUP BY

```
SELECT dept, AVG(gpa)
  FROM students
[WHERE <predicate>]
 GROUP BY dept
 [HAVING <predicate>]
[ORDER BY <column list>] ;
```

dept	avg
CS	3.4
BUSINESS	4.0
DATASCI	4.0

- Partition table into groups with same GROUP BY column values
  - Group By takes a list of columns
- Produce an aggregate result per group

What does the following Produce?

```
SELECT name, AVG(gpa)
  FROM students
[WHERE <predicate>]
 GROUP BY dept
 [HAVING <predicate>]
[ORDER BY <column list>] ;
```

	name	gpa	age	dept	gender
0	Sergey Brin	2.8	40	CS	M
1	Danah Boyd	3.9	35	CS	F
2	Bill Gates	1.0	60	CS	M
3	Hillary Mason	4.0	35	DATASCI	F
4	Mike Olson	3.7	50	CS	M
5	Mark Zuckerberg	4.0	30	CS	M
6	Sheryl Sandberg	4.0	47	BUSINESS	F
7	Susan Wojcicki	4.0	46	BUSINESS	F
8	Marissa Meyer	4.0	45	BUSINESS	F

- An error! (why?)
  - What name should be used for each group?

What if we wanted to only consider departments that have greater than two students?

```
SELECT dept, AVG(gpa)
  FROM students
[WHERE <predicate>]
 GROUP BY dept
 [HAVING <predicate>]
[ORDER BY <column list>] ;
```

What if we wanted to only consider departments that have greater than two students?

```
 SELECT dept, AVG(gpa)
   FROM students
¿ WHERE COUNT(*) > 2
  GROUP BY dept
  [HAVING <predicate>]
 [ORDER BY <column list>] ;
```

- Doesn’t work …
- WHERE clause is applied before GROUP BY
  - You cannot have aggregation functions in the where clause

HAVING

```
SELECT dept, AVG(gpa)
  FROM students
[WHERE <predicate>]
 GROUP BY dept
  HAVING COUNT(*) > 2
[ORDER BY <column list>] ;
```

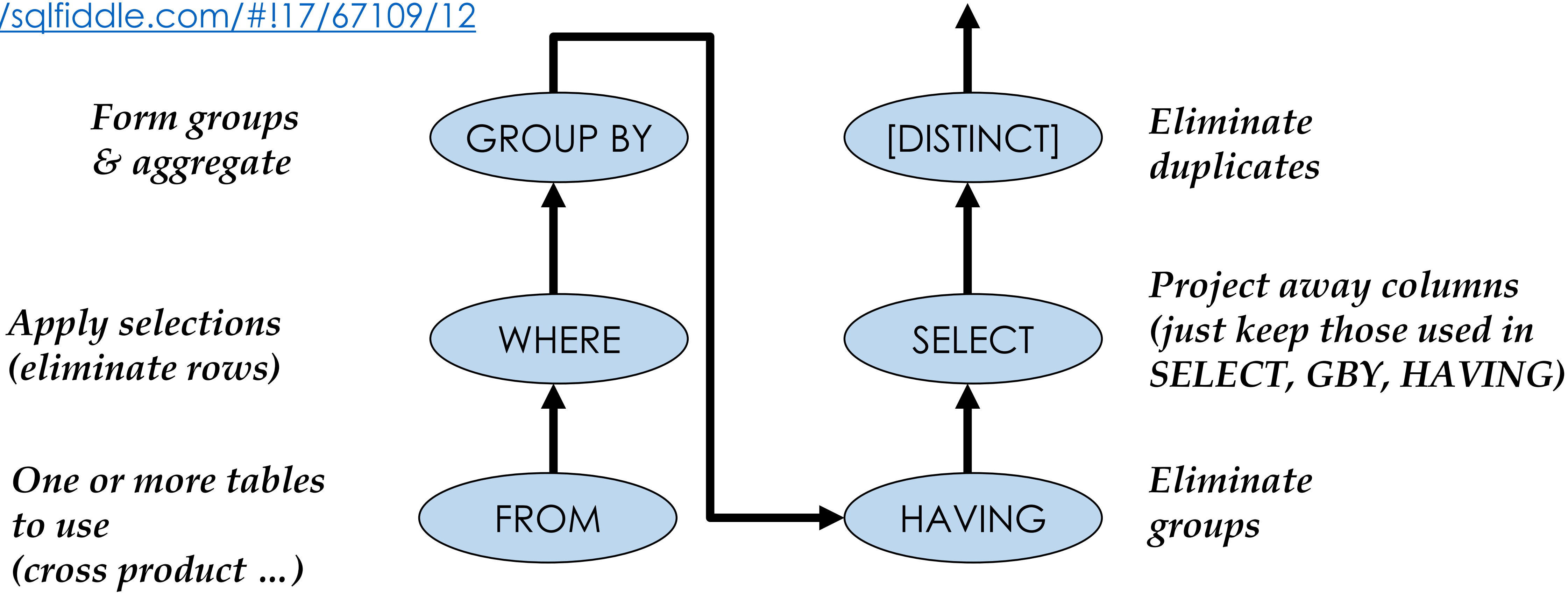
avg	dept
3.4	CS
4.0	BUSINESS

- The HAVING predicate is applied ***after*** *grouping* and *aggregation*
  - Hence can contain anything that could go in the SELECT list
- HAVING can only be used in aggregate queries

Conceptual SQL Evaluation

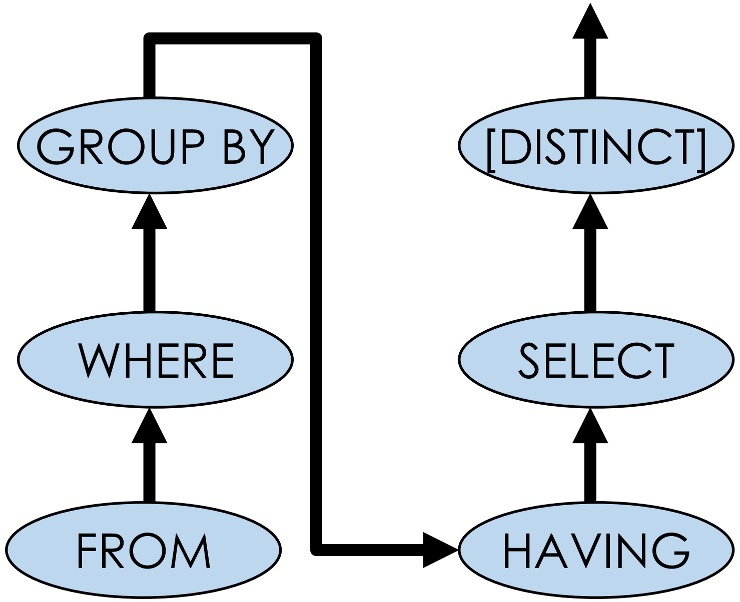
```
SELECT      [DISTINCT]  target-list
FROM        relation-list
WHERE       qualification
GROUP BY    grouping-list
HAVING      group-qualification
```

Try Queries Here
http://sqlfiddle.com/#!17/67109/12



Putting it all together

```
SELECT dept, AVG(gpa) AS avg_gpa, COUNT(*) AS size
  FROM students
 WHERE gender = 'F'
 GROUP BY dept
HAVING COUNT(*) > 2
 ORDER BY avg_gpa DESC
```

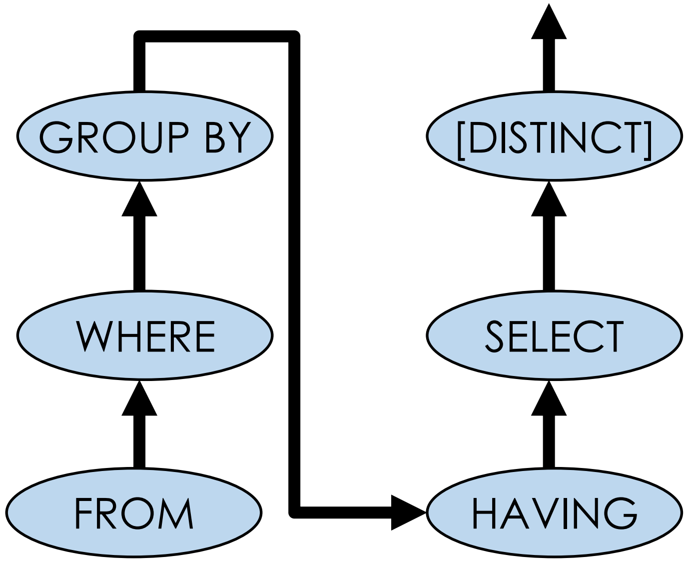


What does this compute?

http://bit.ly/ds100-sp18-sql

Putting it all together

```
SELECT dept, AVG(gpa) AS avg_gpa, COUNT(*) AS size
  FROM students
 WHERE gender = 'F'
 GROUP BY dept
HAVING COUNT(*) > 2
 ORDER BY avg_gpa DESC
```



What does this compute?

- ➢ The average GPA of female students and number of female students in each department where there are at least 3 female students in that department. The results are ordered by the average GPA.

How do you interact with a database?

What is the DBMS?

- Server
- Software
- A library

Answer: It can be all of these.

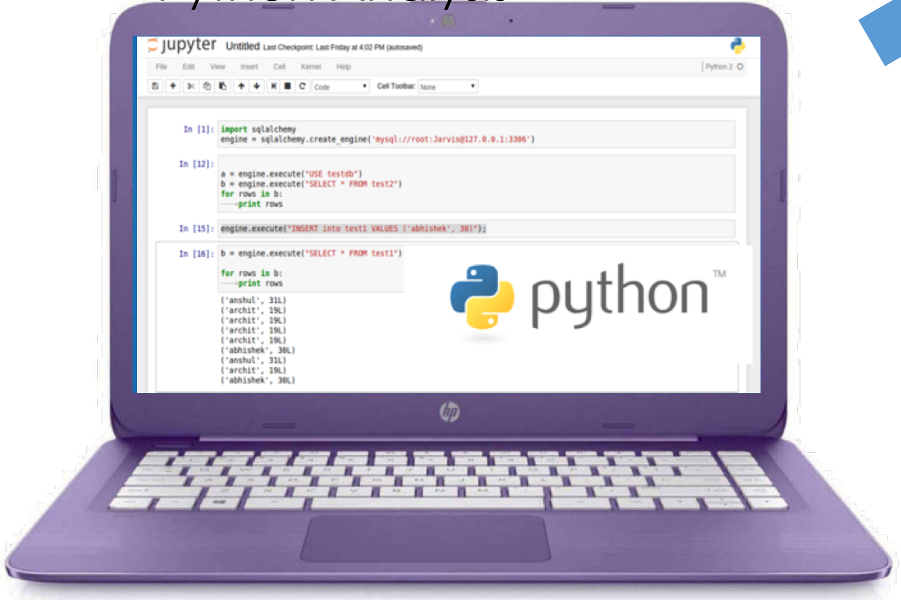
Interacting with a DBMS

DBMS Server

Query

```
SELECT * FROM sales
WHERE price > 100.0
```

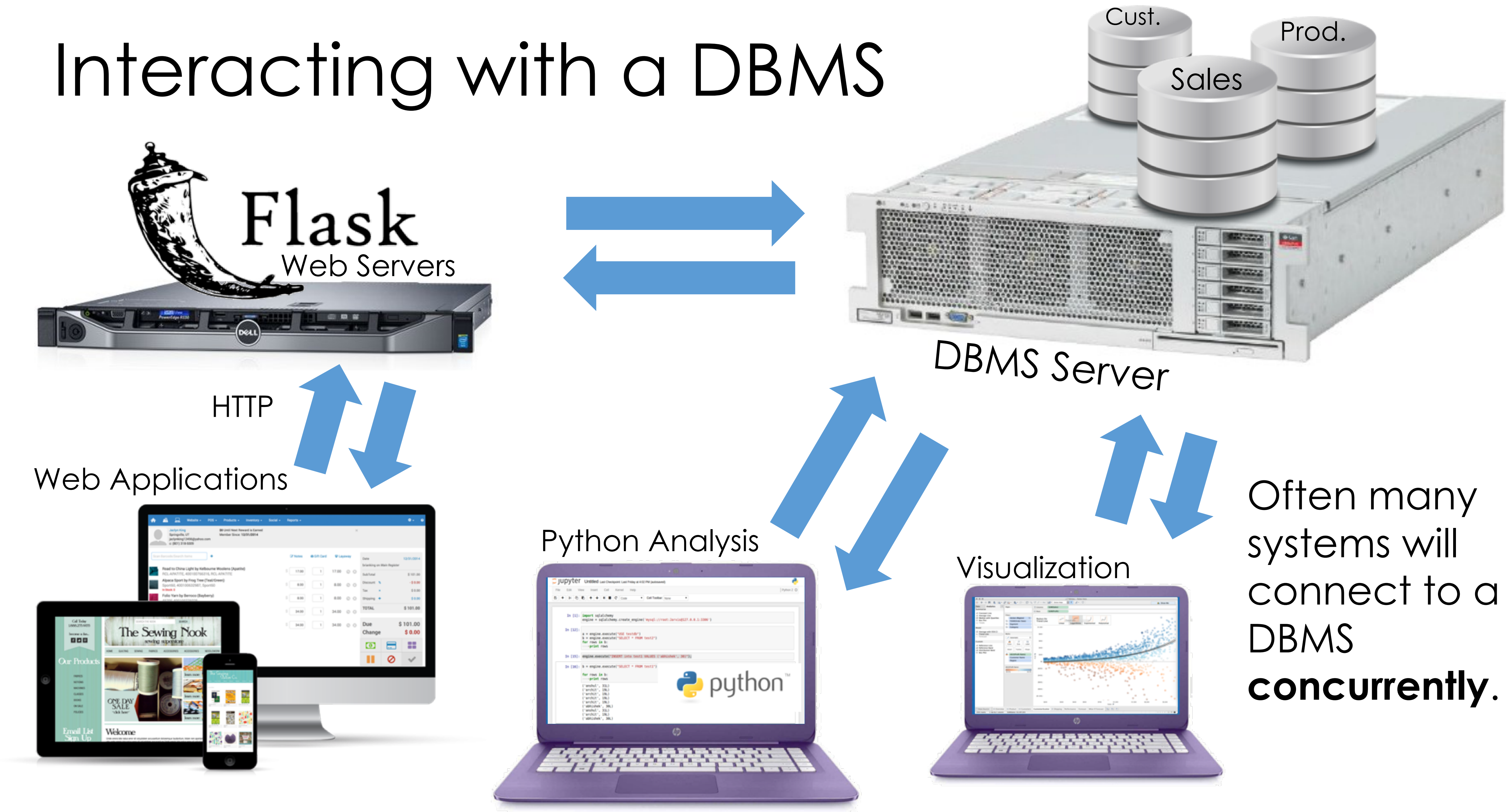
Python Analysis



Response

Date	Purchase ID	Name	Price
9/20/2012	1234	Sue	$200.00
8/21/2012	3453	Joe	$333.99

Interacting with a DBMS



Often many systems will connect to a DBMS **concurrently**.

Break

Why are databases drawn as “cans”

Platters on a Disk Drive

Looks Like?

Platters on a Disk Drive

Looks Like?

1956: IBM MODEL 350 RAMAC
First Commercial Disk Drive
5MB @ 1 ton

Simple Single Table Query Demo