

# From DBMS to BDMS



**SDSC** SAN DIEGO  
SUPERCOMPUTER CENTER

# After this lesson you will be able to:

- Explain at least 5 desirable characteristics of a Big Data Management System
- Explain the difference between ACID and BASE
- Describe what the CAP Theorem states
- List examples of BDMSs and describe some of their similarities and differences

# Desired Characteristics of BDMS

- **A flexible, semistructured data model**
  - “schema first” to “schema never”
- **Support for today’s common “Big Data data types”**
  - Textual, temporal, and spatial data values
- **A full query language**
  - Expectedly at least the power of SQL
- **An efficient parallel query runtime**

# Desired Characteristics of BDMS

- Wide range of query sizes
- Continuous data ingestion
  - Stream ingestion
- Scale gracefully to manage and query large volumes of data
  - Use large clusters
- Full data management capability
  - Ease of operational simplicity

# ACID and BASE

- ACID properties hard to maintain in a BDMS
- **BASE relaxes ACID**
  - BA: Basic Availability
  - S: Soft State
  - E: Eventual Consistency

# CAP Theorem

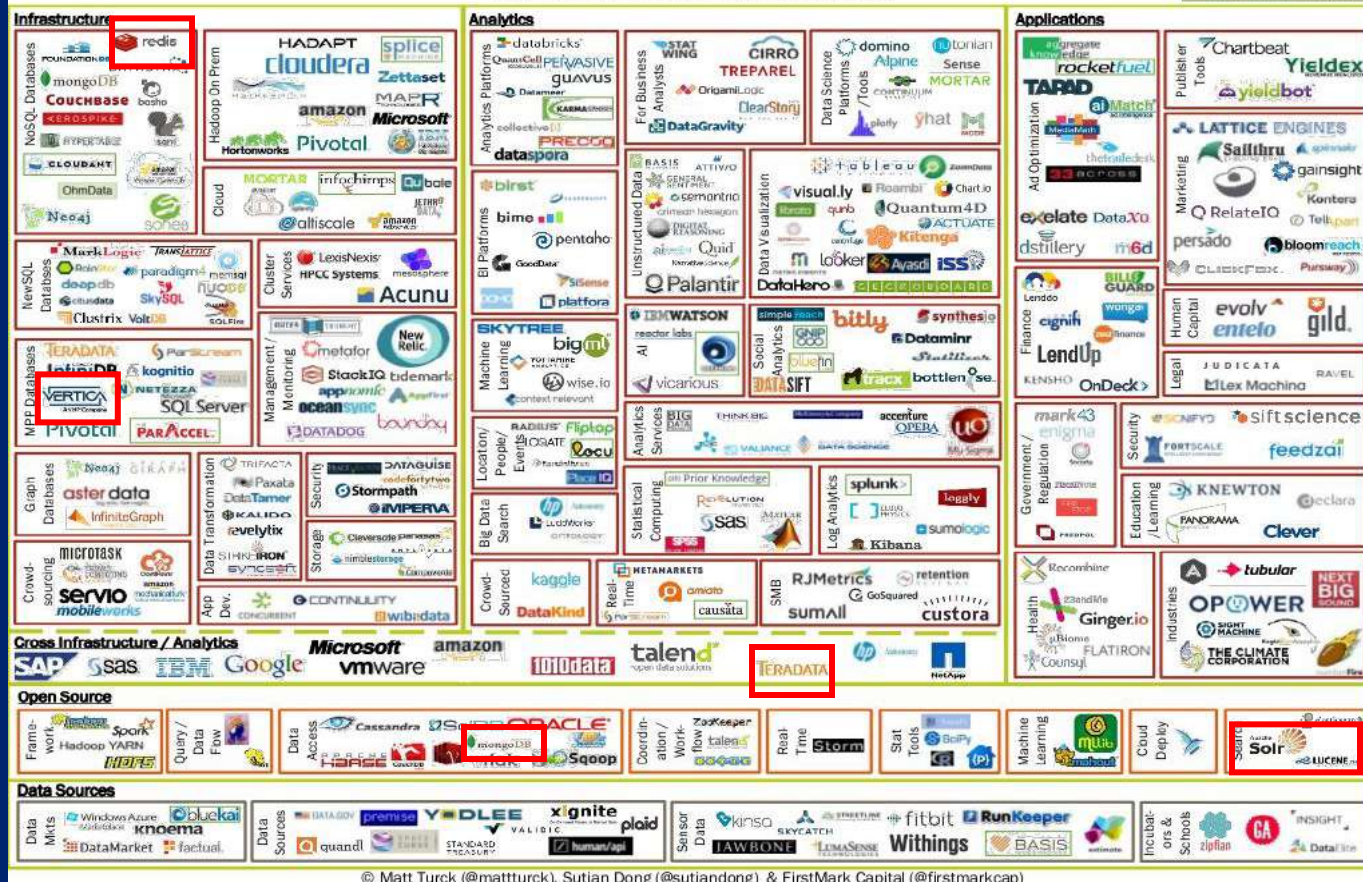
- A distributed computer system cannot simultaneously achieve...
  - Consistency
  - Availability
  - Partition Tolerance



# The Marketplace

## BIG DATA LANDSCAPE, VERSION 3.0

Exited: Acquisition or IPO



**pause**



# Redis – An Enhanced Key-Value Store

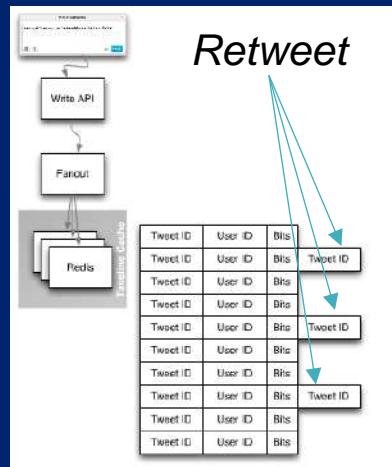
- In-memory data structure store
  - strings, hashes, lists, sets, sorted sets
- Look-up Problem
  - Case 1: (key:string, value:string)



- Keys may have internal structure and expiry
  - comment:1234:reply.to
  - Hierarchical keys: user.commercial, user.commercial.entertainment, user.commercial.entertainment.movie-industry

# Redis and Data Look-up

- **Case 2: (key:string, value: list)**
  - userID: [tweetID<sub>1</sub>, tweetID<sub>2</sub>, ...]
  - Ziplists compress lists
  - Twitter innovation: list of ziplists
    - <http://www.infoq.com/presentations/Real-Time-Delivery-Twitter>
    - <https://www.youtube.com/watch?v=rPgEKvWtozo>



# Redis and Data Look-up

- Case 3: (key:string, value: attribute-value pairs)
  - REDIS Hashes
    - std:101 name:"John Smith" dob:01-01-2000  
gender:M active:0 cgpa:2.9

# Redis and Scalability

- Partitioning and Replication

- Range partitioning

- Example: User record number 1-10000 goes to machine 1, 10001-20000 goes to machine 2, ...

- Hash partitioning

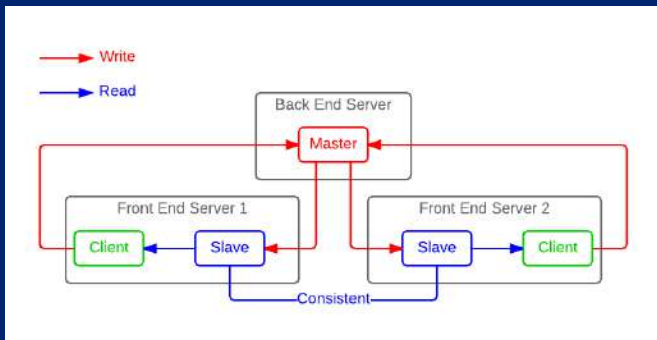
- Pick a key of a record, e.g., "abcde"
    - Using a hash function, turn it into a number, e.g., 152
    - $152 \bmod 10$  is 2, so the record goes to machine 2

# Redis and Scalability

- Partitioning and Replication

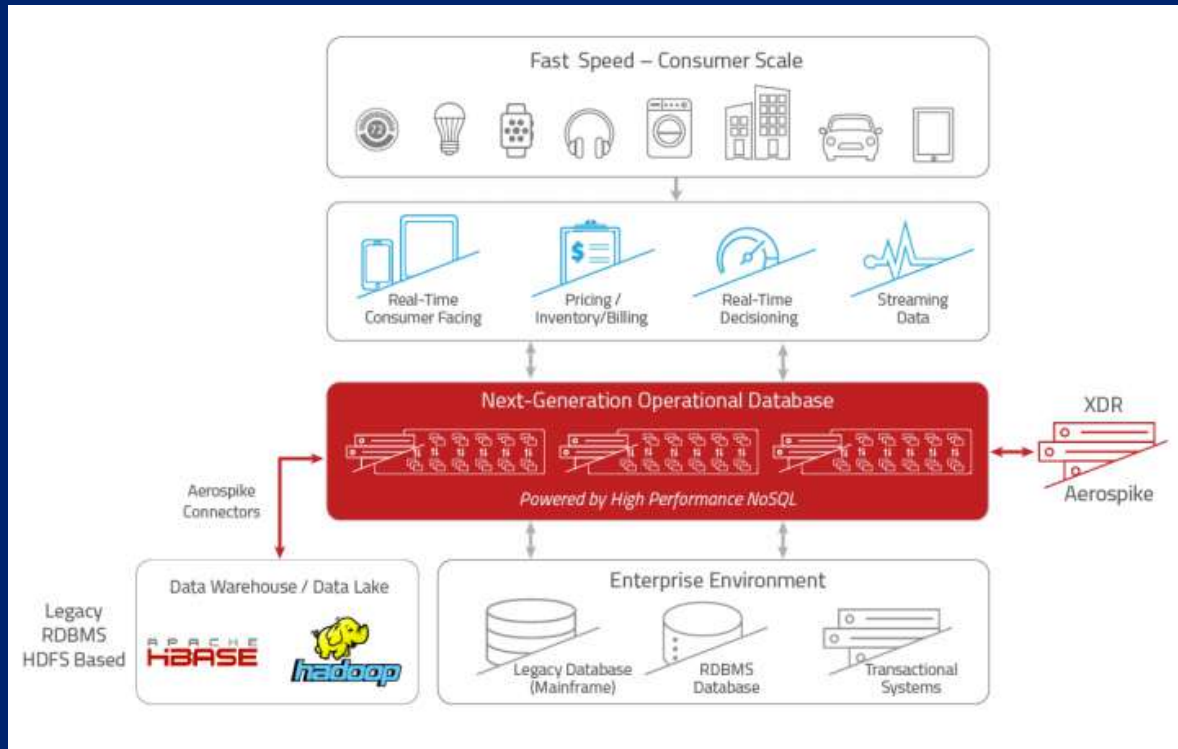
- Master-Slave mode replication

- Clients write to master, master replicates to slaves
    - Clients read from slaves to scale up read performance
    - Slaves are mostly consistent



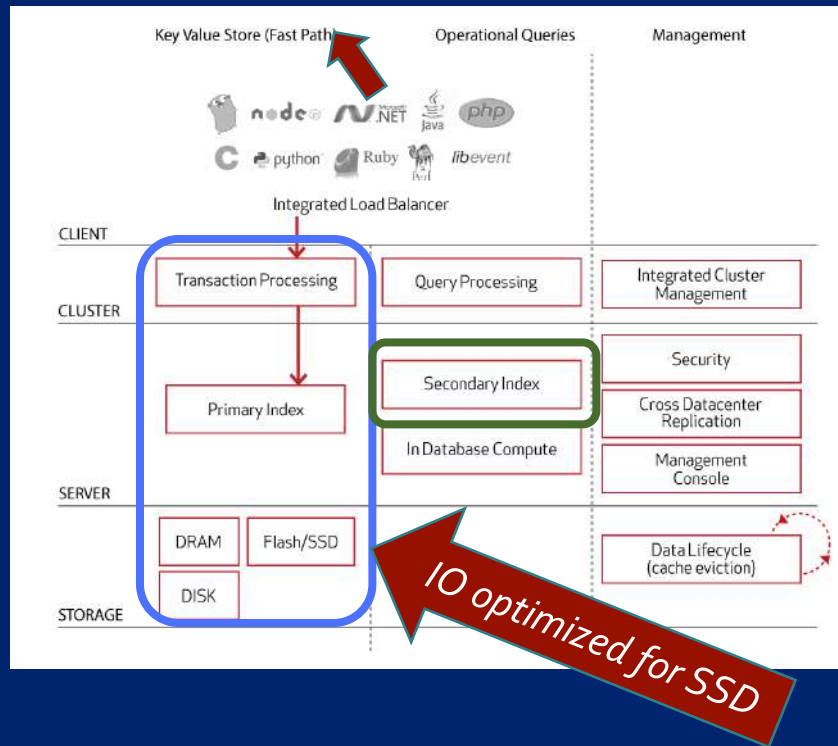
**pause**

# Aerospike – a New Generation KV Store





# Aerospike Architecture



# Querying Aerospike

- **Data types**

- Standard scalar, lists, maps, geospatial, large objects

- **KV store operations**

- Geospatial queries like point-in-polygon

- **AQL: an SQL-like language**

- `SELECT name, age FROM users.profiles`

# Transactions in Aerospike

- **Aerospike ensures ACID**
  - Consistency – all copies of a data item are in sync
    - Uses synchronous write to replicas
    - Mechanisms to relax immediate consistency
- **Durability**
  - Flash storage
  - Replication management
- **Network partitioning reduced**
  - Tighter cluster control

**pause**

# AsterixDB – a DBMS for Semistructured Data

# AsterixDB – a DBMS for Semistructured Data

*An abbreviated Tweet*

```
{
  "created_at": "Thu Oct 21 16:02:46 +0000 2010",
  "entities": {
    "user_mentions": [
      {
        "name": "Gnip, Inc.",
        "screen_name": "gnip"
      }
    ]
  },
  "text": "what we've been up to at @gnip -- delivering data to happy customers http://gnip.com/success\_stories",
  "id": 28039652140,
  "geo": null,
  "retweet_count": null,
  "in_reply_to_user_id": null,
  "user": {
    "name": "Gnip, Inc.",
    "lang": "en",
    "followers_count": 260,
    "friends_count": 71,
    "statuses_count": 302,
    "screen_name": "gnip"
  },
}
```

# Semistructured Schema

```
create dataverse LittleTwitterDemo;
```

```
create type TwitterUserType as open {
```

```
  screen-name: string,  
  lang: string,  
  friends_count: int32,  
  statuses_count: int32,  
  id: int32,  
  followers_count: int32
```

```
}
```

```
create type TweetMessageType as closed {
```

```
  tweetid: string,  
  user: TwitterUserType,  
  geo: point?,  
  created_at: datetime,  
  referred-topics: {{ string }},  
  text: string
```

```
}
```

```
create dataset TweetMessages(TweetMessageType)  
primary key tweetid;
```



# Options for Querying in AsterixDB

- AQL is a natively-supported query language

```
for $user in dataset TwitterUsers order by  
$user.followers_count desc, $user.lang asc return $user
```

- Hive queries

```
SELECT a.val, b.val FROM a LEFT OUTER JOIN b ON  
(a.key=b.key)
```

- Xquery
- Hadoop MR jobs
- SQL++ (coming up)

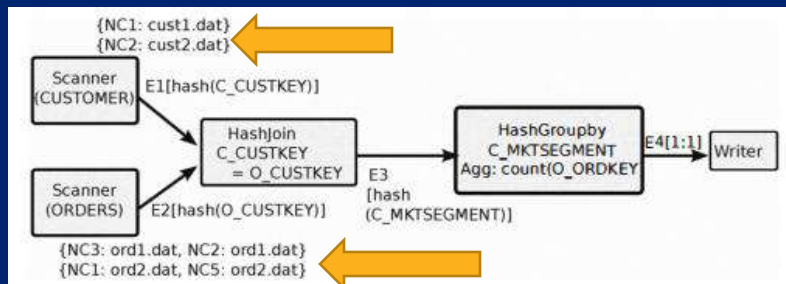
# Operating Over a Cluster

- **Hyracks**

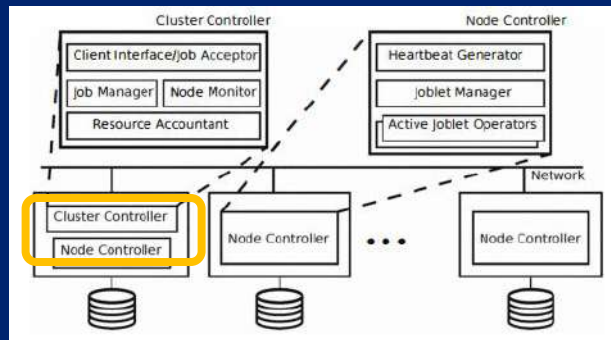
- Query execution engine for partitioned parallel execution of queries

- **Example**

- CUSTOMER (C\_CUSTKEY, C\_MKTSEGMENT, ... )
- ORDERS (O\_ORDERKEY, O\_CUSTKEY, ... )



## *Hyracks Job Management*



# Accessing External Data

- Real-time data *from files in a directory path*

```
create dataset Tweets (Tweet)  
  primary key id;
```

```
create feed TestFileFeed using localfs  
  ("path"="127.0.01:///Users/adc/text/"),  
  ("format"="adm"), ("type-name"="Tweet"),  
  ("expression"=".*\\.adm");
```

```
connect feed TestFileFeed to dataset Tweets;
```

# Accessing External Data

- Real-time data *from an external API*

```
use dataverse feeds;  
create dataset Tweets (Tweet)  
  primary key id;  
create feed TwitterFeed if not exists using "push_twitter"  
  (("type-name"="Tweet"),  
   ("consumer.key"="some-key"),  
   ("consumer.secret"="some-secret"),  
   ("access.token"="some-token"),  
   ("access.token.secret"="some-token-secret"));  
connect feed TwitterFeed to dataset Tweets;
```

**pause**

# Solr – Managing Text

- Basic challenges with text

- Defining a match

- Analyze  $\approx$  Analyse  $\approx$  ANALYZE? Lexical difference, capitalization
    - abc:def-230-39  $\approx$  abcdef23039? Structural punctuations
    - “Barak Hussein Obama”  $\approx$  “Barak Obama”  $\approx$  “Barak H. Obama”  $\approx$  “B. H. Obama”? Nominal Variations
    - Mom  $\approx$  mother? Synonyms
    - Dr.  $\approx$  Doctor Abbreviation
    - USA  $\approx$  “United States of America” Initialism
    - “The tradition is completely American. Students should ...”
      - Should this match the query “American students”?
    - “Mrs. Clinton also said ...”
      - Should this match the query “mrs Clinton”?

# Inverted Index

- **Vocabulary**

- All terms in a collection of documents
  - Multi-word terms, synonym sets

- **Occurrence**

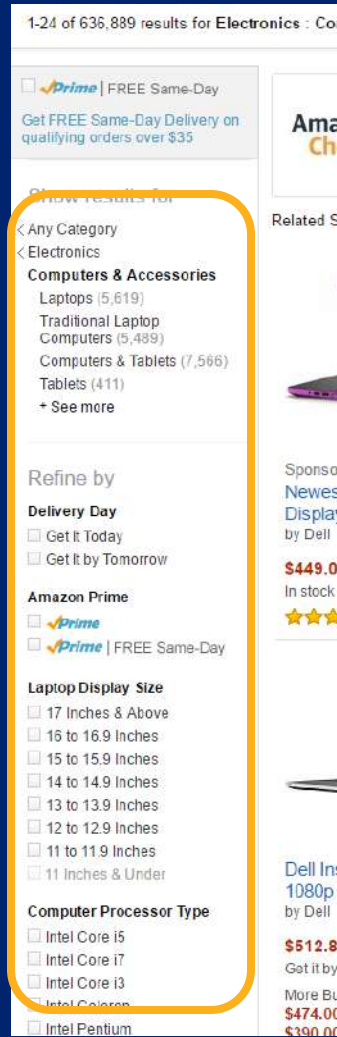
- For each term in the collection
  - List of doc ID
  - List of doc ID, [position of occurrence]
  - Other statistics like tf, idf, ...

***Inverted index***



# Solr Functionality

- Enterprise Search Platform
- Inverted index
  - For every field in a structured text document
    - indexes text, numbers, geographic information, ...
- Faceted Search
- Term highlighting



# Solr Functionality

- Index-time Analyzers

- Tokenizers
- filters

```
<fieldType name="text" class="solr.TextField">
  <analyzer type="index">
    <tokenizer class="solr.StandardTokenizerFactory"/>
    <filter class="solr.StandardFilterFactory"/>
    <filter class="solr.LowerCaseFilterFactory"/>
    <filter class="solr.SynonymFilterFactory"
synonyms="synonyms.txt" ignoreCase="true" expand="true" />
    <filter class="solr.EnglishPorterFilterFactory"/>
  </analyzer>
</fieldType>
```

# Solr Functionality

- Query-time Analyzers

```
<analyzer type="query">
  <tokenizer class="solr.PatternTokenizerFactory"
pattern="[\\s,\\.;]+"/>
  <filter class="solr.StandardFilterFactory"/>
  <filter class="solr.LowerCaseFilterFactory"/>
  <filter class="solr.CommonGramsFilterFactory"
words="stopwords.txt" ignoreCase="true"/>
  <filter class="solr.StopFilterFactory"
ignoreCase="true" words="stopwords.txt"/>
  <charFilter
class="solr.HTMLStripCharFilterFactory"/>
  <filter class="solr.PorterStemFilterFactory"/>
</analyzer>
```

# Solr Queries

## Consider the CSV file

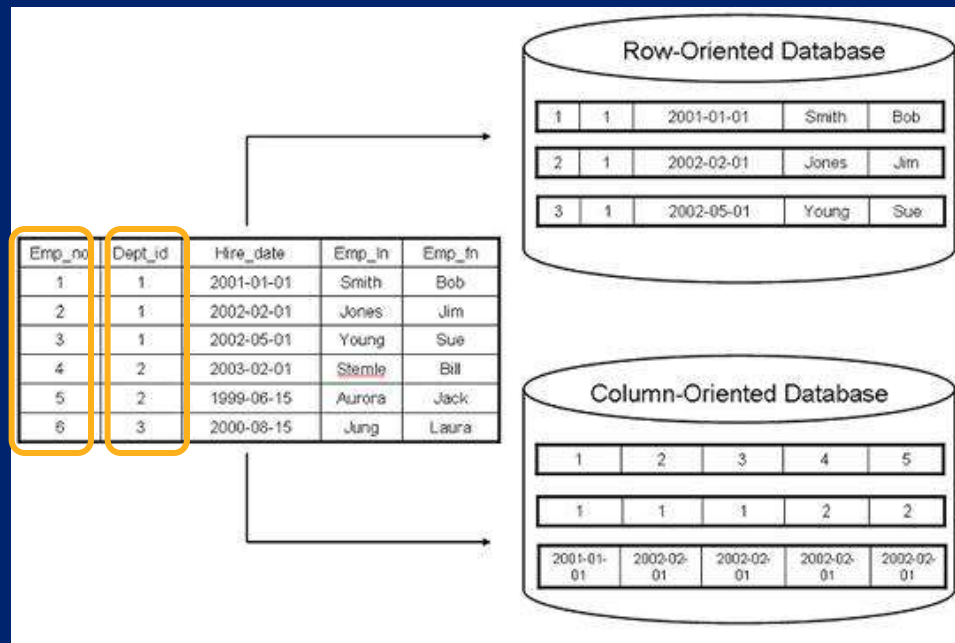
```
id,cat,pubyear_i,title,author,series_s,sequence_i  
book1,fantasy,2001,American Gods,Neil Gaiman,American Gods,  
book2,fantasy,1996,A Game of Thrones,George R.R. Martin,A Song of Ice and Fire,1  
book3,fantasy,1999,A Clash of Kings,George R.R. Martin,A Song of Ice and Fire,2  
book4,sci-fi,1951,Foundation,Isaac Asimov,Foundation Series,1  
book5,sci-fi,1952,Foundation and Empire,Isaac Asimov,Foundation Series,2  
book6,sci-fi,1992,Snow Crash,Neal Stephenson,Snow Crash,  
book7,sci-fi,1984,Neuromancer,William Gibson,Sprawl trilogy,1  
book8,fantasy,1985,The Black Company,Glen Cook,The Black Company,1  
book9,fantasy,1965,The Black Cauldron,Lloyd Alexander,The Chronicles of Prydain,2
```

# Solr Queries

- All books
  - `http://localhost:8983/solr/query? q=*.*`
- All books with “black” in the title field, return author, title
  - `http://localhost:8983/solr/query? q=title:black &fl=author,title`
- All books sort by `pubyear_i` in descending order
  - `http://localhost:8983/solr/query? q=*.*)&sort=pubyear_i desc`
- Above query but facet by all values in the `cat` field
  - `http://localhost:8983/solr/query? q=*.*)&sort=pubyear_i desc &facet=true&facet.field=cat`

**pause**

# Vertica – a Columnar DBMS



## • Column Store

- Store data column-wise
- A query only uses the columns needed
- Usually much faster for queries even for large data



# Space Efficiency

- Column stores keep columns in sorted order
- Values in columns can be compressed
  - Run-length encoding
    - 1/1/2007 – 16 records
  - Frame-of-reference encoding
    - Fix a number and only record the difference
- Compression saves storage space

Txn Date	CustomerID	Trade
1/17/2007, 16	0000001	520.88 76.00 88.00 160.00 22.00 22.00 22.00 22.00 22.00 22.00 22.00 22.00 22.00 22.00 22.00 22.00
1/17/2007	0	
1/17/2007	2	
1/17/2007	2	
1/17/2007	4	
1/17/2007	10	
1/17/2007	10	23.21
1/17/2007	19	344.44
1/17/2007	25	21.30
1/17/2007	49	23.92
1/17/2007	50	50.22
1/17/2007	51	38.22
1/17/2007	52	21.92
1/17/2007	67	74.26
1/17/2007	68	152.49
1/17/2007	70	89.23

# Working with Vertica

- **Column-Groups**

- Frequently co-accessed columns behave as mini row-stores within the column store

- **Update performance slower**

- Internal conversion from row-representation to column-representation

- **Enhanced suite of analytical operations**

- Window statistics
  - Ticks (ts TIMESTAMP, Stock varchar(10), Bid float

# Vertica and Analytic Functions

```
'2011-07-12 10:23:54', 'abc', 10.12
'2011-07-12 10:23:58', 'abc', 10.34
'2011-07-12 10:23:59', 'abc', 10.75
'2011-07-12 10:25:15', 'abc', 11.98
'2011-07-12 10:25:16', 'abc'
'2011-07-12 10:25:22', 'xyz', 45.16
'2011-07-12 10:25:27', 'xyz', 49.33
'2011-07-12 10:31:12', 'xyz', 65.25
'2011-07-12 10:31:15', 'xyz'
```

```
SELECT ts, bid, AVG(bid)
  OVER(ORDER BY ts
        RANGE BETWEEN INTERVAL '40
seconds' PRECEDING AND CURRENT ROW)
FROM ticks
WHERE stock = 'abc'
GROUP BY bid, ts
ORDER BY ts;
```

ts	bid	?column?
-----+-----+-----		
2011-07-12 10:23:54	10.12	10.12
2011-07-12 10:23:58	10.34	10.23
2011-07-12 10:23:59	10.75	10.40333333333333
2011-07-12 10:25:15	11.98	11.98
2011-07-12 10:25:16		11.98

(5 rows)

# Vertica and Distributed R

- **Distributed R**

- High-performance statistical analysis
- Master node: schedules tasks and sends code
- Worker nodes: maintain data partitions and compute
- Uses a data structure called dArray or distributed array

