Data Storage

Marek J. Druzdzel

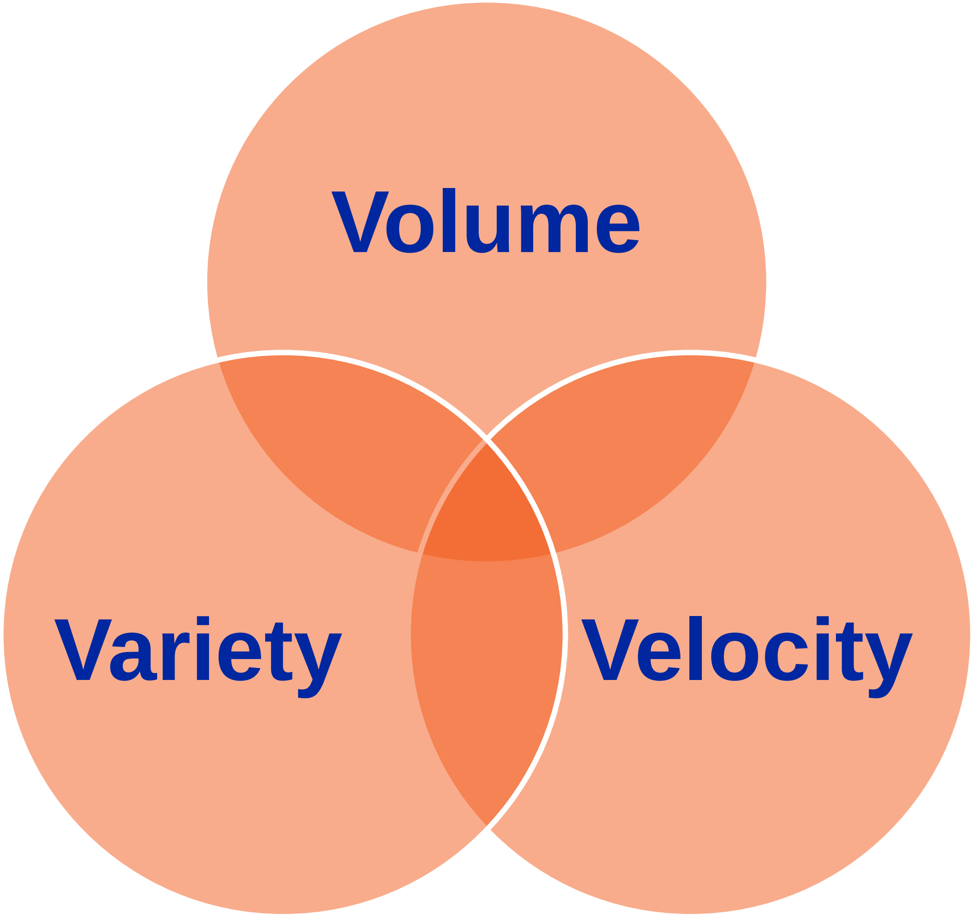
University of Pittsburgh
School of Information Sciences
and Intelligent Systems Program
marek@sis.pitt.edu
http://www.pitt.edu/~druzdzel

Outline

- Collecting data
- Relational Databases
- NoSQL Databases
    - Key-value databases
    - Document databases
    - Column-family stores
    - Graph databases
- Beyond NoSQL Databases

Collecting Data

Big Data's 3V



The quantity of data collected

- The New York Stock Exchange generates about one terabyte of new trade data per day.
- Facebook hosts approximately 10 billion photos, taking up one petabyte of storage.
- Ancestry.com, the genealogy site, stores around 2.5 petabytes (10^{15}) of data.
- The Internet Archive stores around 2 petabytes (10^{15}) of data, and is growing at a rate of 20 terabytes (10^{12}) per month.
- The Large Hadron Collider near Geneva, Switzerland, will produce about 15 petabytes (10^{15}) of data per year.

Reference: Tom White, Hadoop: The Definitive Guide, Third Edition, 2012

Storing Data: Relational Databases

Relational databases: Definition

- Relational database: A set of relations
- Relation: Made up of 2 parts:
  - Schema: specifies name of relation plus name and type of each column

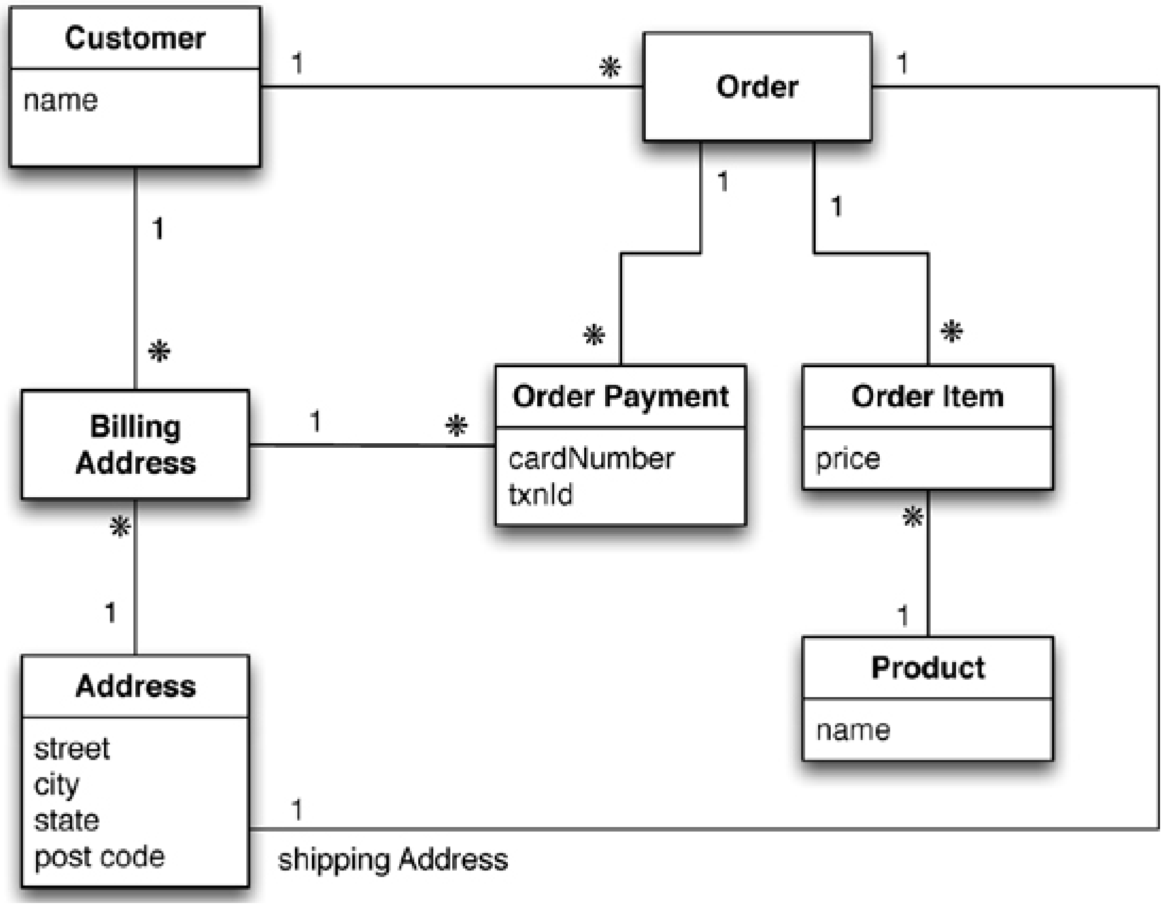
    Customer(id:int, name:string, gender:string, email:string)
  - Instance: a table, with rows and columns

id	name	gender	email
1	Marek Druzdzel	Male	marek@sis,pitt.edu
2	Marcin Kozniewski	Male	mak295@pitt.edu

  - Can think of a relation as a set of rows (tuples)

Reference for this section: NoSQL distilled: A brief guide to the emerging world of polyglotp ersistence / Pramod J Sadalage, Martin Fowler.

Relational databases: Data Model



Data model oriented around a relational database (using UML notation)

Relational databases: Data Model

Customer

Id	Name
1	Martin

Orders

Id	CustomerId	ShippingAddressId
99	1	77

Product

Id	Name
27	NoSQL Distilled

BillingAddress

Id	CustomerId	AddressId
55	1	77

OrderItem

Id	OrderId	ProductId	Price
100	99	27	32.45

Address

Id	City
77	Chicago

OrderPayment

Id	OrderId	CardNumber	BillingAddressId	txnId
33	99	1000-1000	55	abelif879rft

Typical data using RDBMS data model

Relational databases: Integrity constraints (ICs)

- IC: condition that must be true for any instance of the database, e.g., domain constraints
  - ICs are specified when schema is defined
  - ICs are checked when relations are modified
- A legal instance of a relation is one that satisfies all specified ICs.
  - DBMS should not allow illegal instances

Relational databases: Primary key constraints

- A set of fields is a key for a relation if:
  1. No two distinct tuples can have same value in all key fields, and
  2. This is not true for any subset of the key
     - » If part 2 is false, then it is super key.
- K is a candidate key if K is minimal.
- Among all candidate keys we must select one that becomes the Primary Key

Relational databases: Foreign key constraints

- Foreign key: Set of fields in one relation that is used to refer to a tuple in another relation (must correspond to a primary key of the second relation)
- If all foreign key constraints are enforced, referential integrity is achieved

Customer	
Id	Name
1	Martin

Orders		
Id	CustomerId	ShippingAddressId
99	1	77

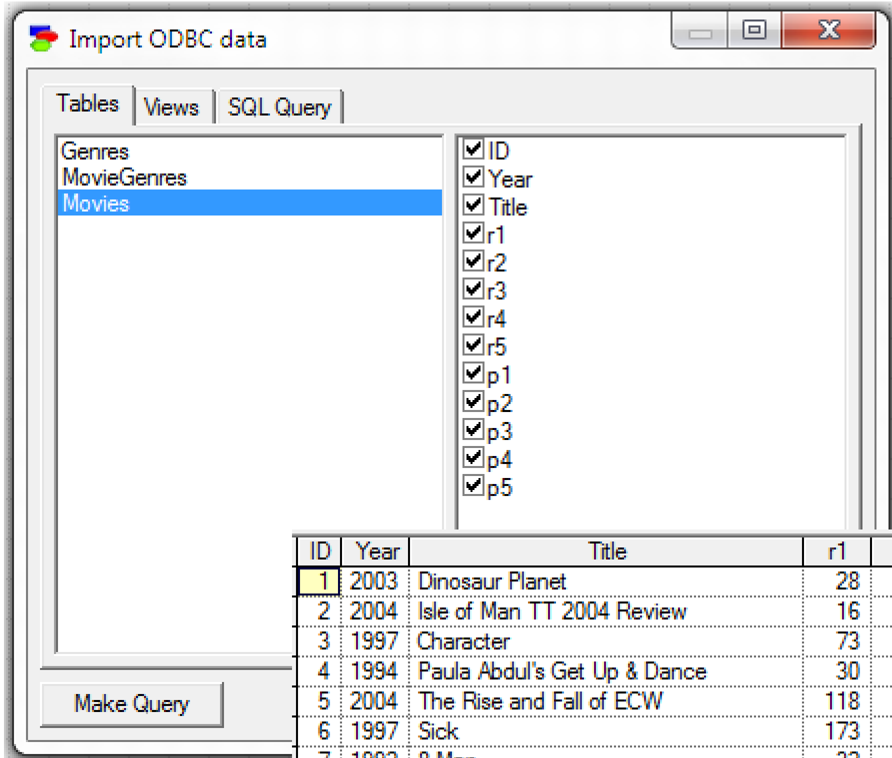
Relational databases: Normalization

- **1st Normal Form**: make tables flat (all elements atomic)
- **2nd Normal Form**: Every non-prime attribute depends on a candidate key or another non-prime attribute
- **3rd Normal Form**: some redundancy but dependency preserving
- **BCNF** (Boyce Codd Normal Form): no redundancy but not dependency preserving
- …
- Redundancy might lead to update anomalies
- Note: we are going to break normalization (denormalize) later

Relational databases: Query language

- A major strength of the relational model:
  - Supports simple, powerful querying of data (SQL)
    - » DDL (Data Description Language): create, drop, alter
    - » DML (Data Manipulation Language): insert, update, delete, select
    - » DCL (Data Control Language): grant, revoke
    - » TCL (Transaction Control Language): commit, rollback
- Queries can be written intuitively, and the DBMS is responsible for efficient evaluation
  - The key: precise semantics for relational queries
  - Allows the optimizer to extensively re-order operations, and still ensure that the answer does not change.

Relational databases: Example



Genres

ID	Name
12	Action
1	Adult
16	Adventure
10	Animation
22	Biography
6	Comedy
13	Crime
7	Documentary
2	Drama
11	Family
14	Fantasy
24	Film-Noir
25	Game-Show
8	History

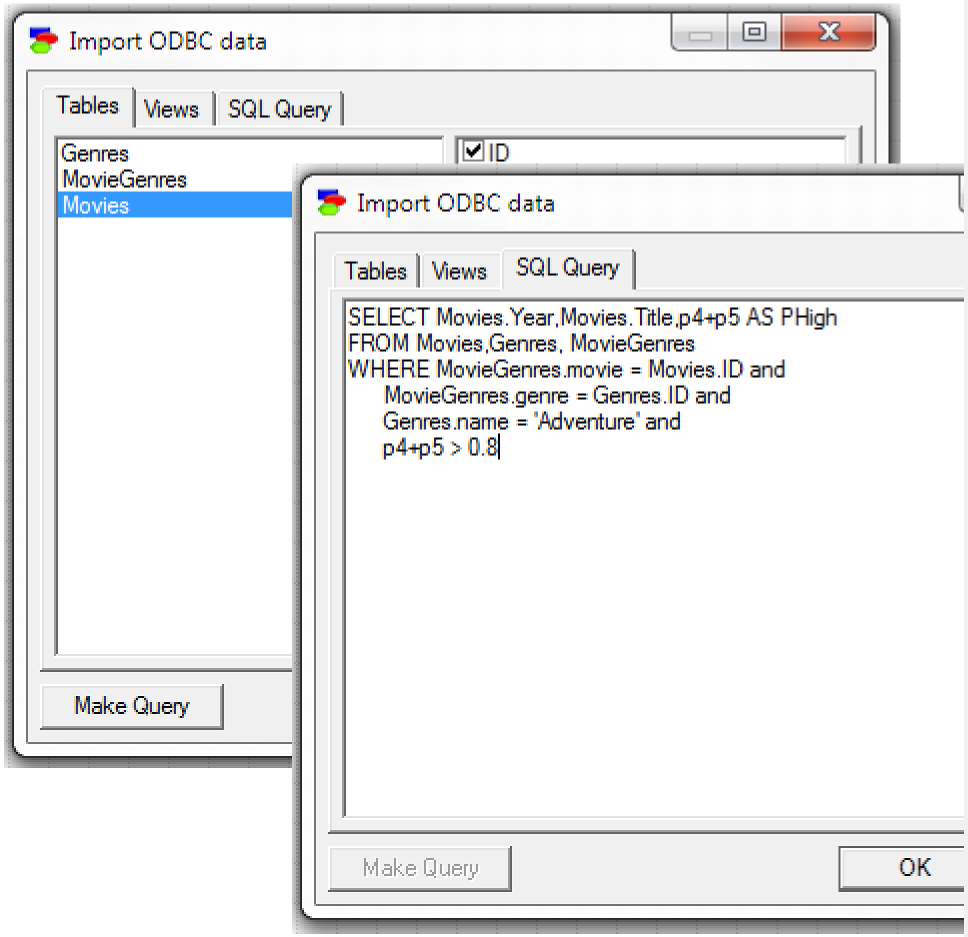
MovieGenres

movie	genre
13519	1
12056	1
14249	2
6745	3
6745	4
6745	5
12540	6
12540	4
15399	6
5183	7
15760	7
15760	3
8769	7
8769	8
8769	9
2945	6
2945	4
3921	10
3921	6
3921	11
13240	6
11625	6
10198	6
10198	11

Movies

ID	Year	Title	r1	r2	r3	r4	r5	p1	p2	p3	p4	p5
1	2003	Dinosaur Planet	28	31	136	207	145	0.05118	[illegible]	0.248629	0.378428	0.265082
2	2004	Isle of Man TT 2004 Review	16	11	35	42	41	0.1103	[illegible]	0.241379	0.289655	0.282759
3	1997	Character	73	153	554	875	357	0.03628	[illegible]	0.275348	0.434891	0.177435
4	1994	Paula Abdul's Get Up & Dance	30	34	40	19	19	0.2112	[illegible]	0.28169	0.133803	0.133803
5	2004	The Rise and Fall of ECW	118	51	140	327	504	0.1035	[illegible]	0.122807	0.286842	0.442105
6	1997	Sick	173	136	295	262	153	0.1697	[illegible]	0.2895	0.257115	0.150147
7	1992	8 Man	32	30	21	7	3	0.3440	[illegible]	0.225806	0.0752688	0.0322581
8	2004	What the #$*! Do We Know!?	2089	2505	3766	3587	2963	0.140107	0.168008	0.252582	0.240577	0.198726
9	1991	Class of Nuke 'Em High 2	21	25	28	11	10	0.221053	0.263158	0.294737	0.115789	0.105263
10	2001	Fighter	25	34	93	65	32	0.100402	0.136546	0.373494	0.261044	0.128514
11	1999	Full Frame: Documentary Shorts	20	38	75	46	19	0.10101	0.191919	0.378788	0.232323	0.0959596
12	1947	My Favorite Brunette	30	56	209	158	93	0.0549451	0.102564	0.382784	0.289377	0.17033
13	2003	Lord of the Rings: The Return of the Kin	0	1	9	35	80	0.0	0.008	0.072	0.28	0.64
14	1982	Nature: Antarctica	17	17	43	28	13	0.144068	0.144068	0.364407	0.237288	0.110169
15	1988	Neil Diamond: Greatest Hits Live	30	35	88	96	41	0.103448	0.12069	0.303448	0.331034	0.141379
16	1996	Screamers	158	511	1146	675	209	0.0585402	0.189329	0.424602	0.250093	0.0774361
17	2005	7 Seconds	669	1596	3013	1414	416	0.0941193	0.224536	0.423889	0.198931	0.0585256

Relational databases: Query Example



Year	Title	PHigh
1959	North by Northwest	0.809471
1993	Star Trek: The Next Generation: Season 7	0.836429
1954	Seven Samurai	0.800669
2003	Read Or Die	0.827824
2001	Alias: Season 1	0.846131
2004	Samurai Champloo	0.8825
1991	Star Trek: The Next Generation: Season 5	0.834596
2004	Case Closed: Season 5	0.866141
2004	Teen Titans: Season 2	0.863637
2004	Lost: Season 1	0.941647
1992	Star Trek: The Next Generation: Season 6	0.846714
2000	Farscape: Season 2	0.81321
2005	Batman Begins	0.829358
2004	Farscape: The Peacekeeper Wars	0.831399
2002	Farscape: Season 4	0.840428
2002	Alias: Season 2	0.857559
1990	Star Trek: The Next Generation: Season 4	0.84949
2003	Alias: Season 3	0.849061
2002	Smallville: Season 2	0.80966
1987	The Princess Bride	0.832354
2004	Battlestar Galactica: Season 1	0.93818
1995	Ninja Scroll	0.804775
1981	Raiders of the Lost Ark	0.908969
2003	Smallville: Season 3	0.813642
2004	The Incredibles	0.842713
1963	The Great Escape	0.816221
2004	Harry Potter and the Prisoner of Azkaban	0.807584
2001	Farscape: Season 3	0.844224
1995	Toy Story	0.858613
2000	Gladiator	0.805826
1999	Toy Story 2	0.809113
1988	Star Trek: The Next Generation: Season 2	0.824505
2004	Smallville: Season 4	0.874619
1989	Indiana Jones and the Last Crusade	0.852522

Relational databases: ACID transactions

- Transaction is a unit of work performed within a database management system against a database
- Atomicity – all or nothing
- Consistency – bring database from one valid state to another
- Isolation - the intermediate state of a transaction is invisible to other transactions
- Durability - after a transaction successfully completes, changes to data persist and are not undone, even in the event of a system failure.

Storing Data: NoSQL Databases

NoSQL databases

NoSQL databases:

- Not using the relational model
- Schemaless
- Running well on clusters
- Tend to be open-source
- List of NoSQL databases (more than 150!): http://nosql-database.org/

No data model?

By Rich Tennant

Why are NoSQL databases interesting?

Two primary reasons:

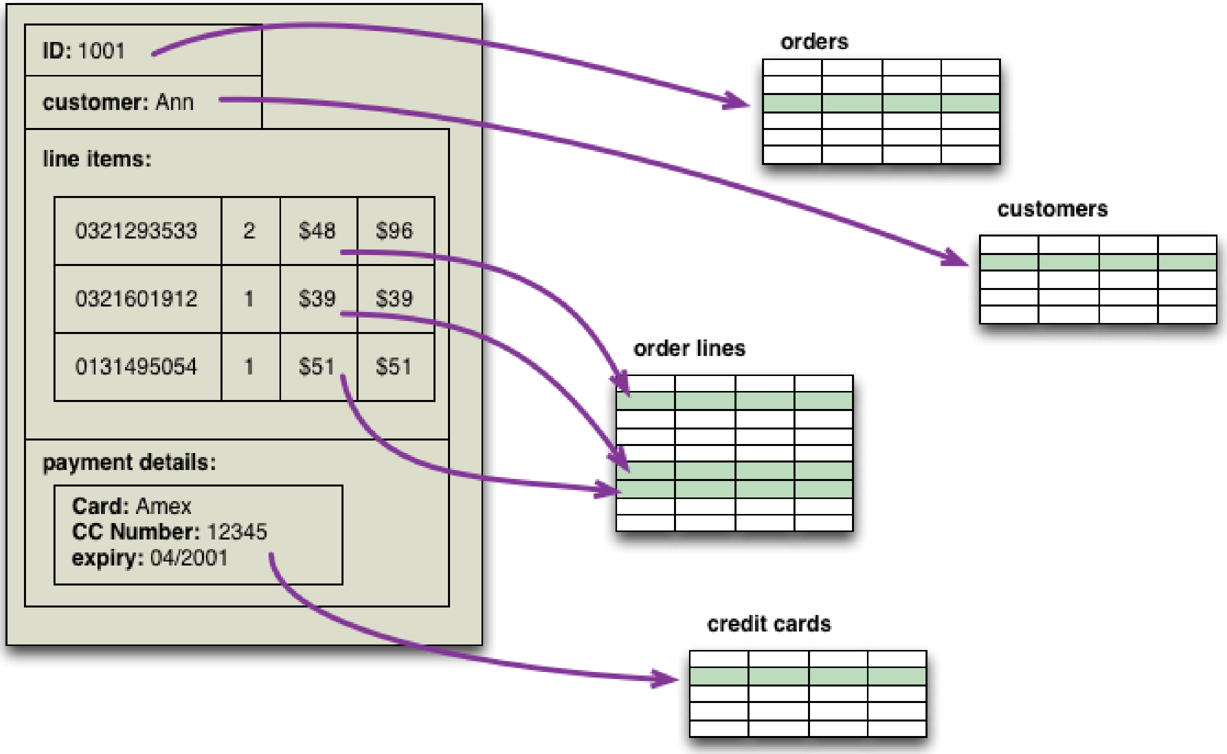
- Application development productivity
- Large-scale data

NoSQL databases: Data models

- Key-value databases:
  - BerkeleyDB, LevelDB, Memcached, Project Voldemort, Redis, *Riak*, …
- Document databases:
  - CouchDb, *MongoDB*, OrientDB, RavenDB, Terrastore, …
- Column-family stores:
  - Amazon SimpleDB, *Cassandra*, Hbase, Hypertable, …
- Graph databases:
  - FlockDB, HyperGraphDB, Infinite Graph, Neo4J, OrientDB, …
- Other

http://en.wikipedia.org/wiki/Data_warehouse

Aggregate data model: Example

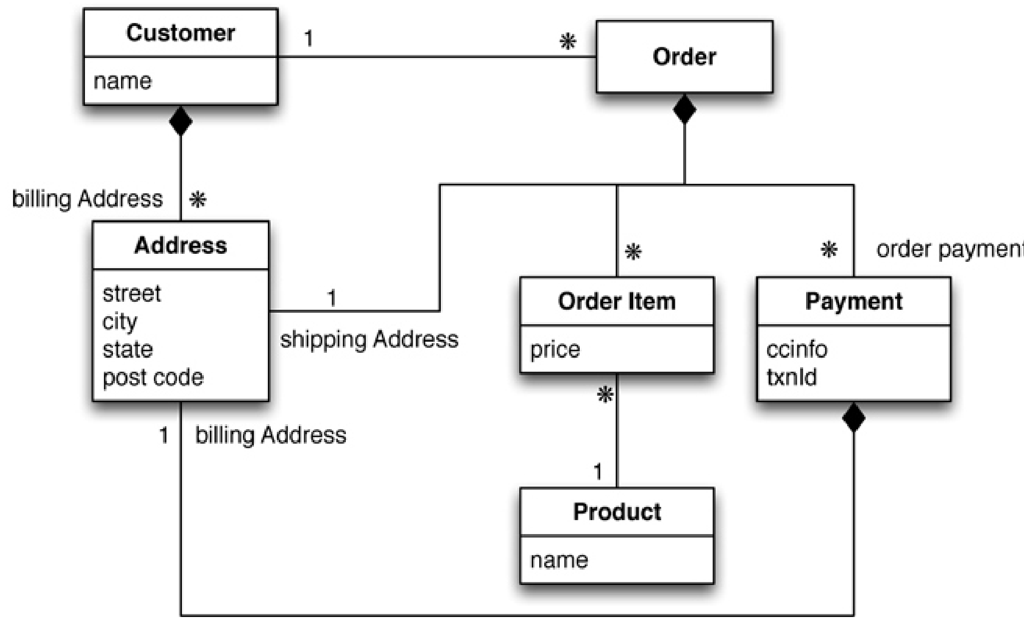


- DDD: Domain-Driven Design
- Apparently easier to program
- Definitely, makes it easier to store and process data on multiple computer clusters.

http://martinfowler.com/bliki/AggregateOrientedDatabase.html

Aggregate data model

```
// in customers
{
"id":1,
"name":"Martin",
"billingAddress":[{"city":"Chicago"}]
}
// in orders
{
"id":99,
"customerId":1,
"orderItems":[
   {
    "productId":27,
    "price": 32.45,
    "productName": "NoSQL Distilled"
   }
 ],
"shippingAddress":[{"city":"Chicago"}]
"orderPayment":[
  {
     "ccinfo":"1000-1000-1000-1000",
     "txnId":"abelif879rft",
     "billingAddress": {"city": "Chicago"}
 }],
}
```

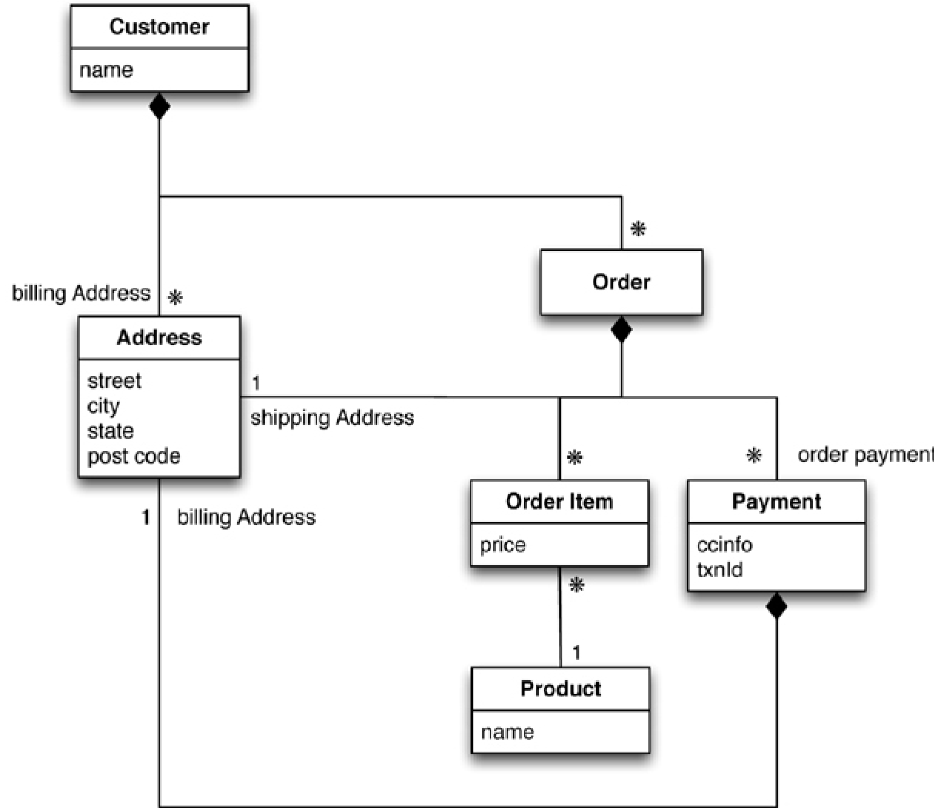


An aggregate data model

http://martinfowler.com/bliki/AggregateOrientedDatabase.html
NoSQL distilled : a brief guide to the emerging world of polyglot persistence / Pramod J Sadalage, Martin Fowler.

Aggregate data model

```
// in customers
{
"customer": {
"id": 1,
"name": "Martin",
"billingAddress": [{"city": "Chicago"}],
"orders": [
 {
   "id":99,
   "customerId":1,
   "orderItems":[
   {
   "productId":27,
   "price": 32.45,
   "productName": "NoSQL Distilled"
   }
  ],
  "shippingAddress":[{"city":"Chicago"}]
  "orderPayment":[
   {
   "ccinfo":"1000-1000-1000-1000",
   "txnId":"abelif879rft",
   "billingAddress": {"city": "Chicago"}
   }],
  }]}
}
```



An aggregate data model

http://martinfowler.com/bliki/AggregateOrientedDatabase.html

NoSQL distilled : a brief guide to the emerging world of polyglot persistence / Pramod J Sadalage, Martin Fowler.

Aggregate data model: Pros and cons

- Cons:
  - It is often difficult to draw aggregate boundaries well
  - Does not support ACID transactions (thus sacrifices consistency)
  - Some queries are easier (to the point of being practical) but others may be really hard (e.g., to get to product sales history, you'll have to dig into every aggregate in the database).
- Pros:
  - Helps greatly with running on a cluster: This is the main argument for NoSQL databases.

http://blog.dynatrace.com/2011/10/05/nosql-or-rdbms-are-we-asking-the-right-questions/

Aggregate data model: Key points

- **An aggregate is a collection of data that we interact with as a unit.**
- **Key-value, document, and column-family databases can all be seen as forms of aggregate-oriented database.**
- **Aggregates make it easier for the database to manage data storage over clusters.**

NoSQL distilled : a brief guide to the emerging world of polyglot persistence / Pramod J Sadalage, Martin Fowler.

Storing Data: NoSQL Database: Key-Value Databases

Key-value databases

- A key-value store is a simple hash table
  - Get the value for the key
  - Put a value for a key
  - Delete a key from the data store
- The value is a blob
- You can think of such databases as databases with only one table, which has two columns: ID and Value.

NoSQL distilled : a brief guide to the emerging world of polyglot persistence / Pramod J Sadalage, Martin Fowler.

Key-value databases: Riak

Terminology comparison:

RDBMS	Riak
Database	**Riak cluster**
Table	**Bucket**
Row	**Key-value**
RowID (at least in Oracle)	**key**

1,000s of startups, enterprises, and organizations have deployed Riak for their production systems.

References: http://wiki.basho.com/Riak.html
NoSQL distilled : a brief guide to the emerging world of polyglot persistence / Pramod J Sadalage, Martin Fowler.

Key-value databases: Riak

- Writing to the Riak bucket using the *store* API:

```
Bucket bucket = getBucket(bucketName);
IRiakObject riakObject = bucket.store(key, value).execute();
```

- Getting value for the key using *fetch* API:

```
Bucket bucket = getBucket(bucketName);
IRiakObject riakObject = bucket.fetch(key).execute();
byte[] bytes = riakObject.getValue();
String value = new String(bytes);
```

References: http://wiki.basho.com/Riak.html
NoSQL distilled : a brief guide to the emerging world of polyglot persistence / Pramod J Sadalage, Martin Fowler.

Key-value databases: Riak

Riak provides an HTTP-based interface (this allows all operations to be performed from a web browser or command line):

- curl -X PUT HTTP://127.0.0.1:8098/riak/images/1.jpg -H "Content-type: image/jpeg" --data-binary @images.jpg
- curl -i HTTP://127.0.0.1:8098/riak/images/1.jpg
- curl -v -X POST -d '

{ "lastVisit":1324669989288,

"user":{"customerId":“1",

"name":"buyer",

"countryCode":"US",

"tzOffset":0}

}' -H "Content-Type:application/json“ http://127.0.0.1:8098/riak/test/1

- curl -i HTTP://127.0.0.1:8098/riak/test/1

References: http://wiki.basho.com/Riak.html
NoSQL distilled : a brief guide to the emerging world of polyglot persistence / Pramod J Sadalage, Martin Fowler.

Key-value databases: Usage

- When to use:
  - Storing session information
  - User profiles, preferences
  - Shopping cart data
- When not to use
  - Relationships among data
  - Multi-operation transactions
  - Query by data
  - Operations by sets

References: http://wiki.basho.com/Riak.html
NoSQL distilled : a brief guide to the emerging world of polyglot persistence / Pramod J Sadalage, Martin Fowler.

Storing Data: NoSQL Database: Document Databases

Document databases

- Documents are the main concept here
  - DB stores and retrieves documents
- Documents stored are similar to each other but do not have to be exactly the same
  - Schemaless
- Documents are stored in the value part of the key-value store

NoSQL distilled : a brief guide to the emerging world of polyglot persistence / Pramod J Sadalage, Martin Fowler.

Document databases: What is document?

- One document:
  - { "firstname": "Martin",
    "likes": ["Biking", "Photography"],
    "lastcity": "Boston",
    "lastVisited":
    }

Schema can differ significantly among documents in the same database.

This was not possible in RDBMS databases.

- Another document:
  - {
    "firstname": "Pramod",
    "citiesvisited": ["Chicago", "London", "Pune", "Bangalore"],
    "addresses": [
    { "state": "AK",
    "city": "DILLINGHAM",
    "type": "R"
    },
    { "state": "MH",
    "city": "PUNE",
    "type": "R" }
    ],
    "lastcity": "Chicago"
    }

NoSQL distilled : a brief guide to the emerging world of polyglot persistence / Pramod J Sadalage, Martin Fowler.

Document databases: MongoDB

Terminology comparison:

mongoDB
{name: "mongo", type:"DB"}

RDBMS	MongoDB
Database	**MongoDB**
Table	**Collection**
Row	**Document**
Rowid (at least Oracle)	**_id**
Join	**DBRef**

MTV MUSIC TELEVISION craigslist Disney Forbes foursquare

References: http://www.mongodb.org/
NoSQL distilled : a brief guide to the emerging world of polyglot persistence / Pramod J Sadalage, Martin Fowler.

Document databases: MongoDB

- [DEMO] - http://www.mongodb.org/#
- Save document to MongoDB:
  - db.docName.save({firstname: 'Martin', likes: ['Biking', 'Photography'], lastcity: 'Boston', lastVisited:''});
- Save another document to MongoDB:
  - db.docName.save({firstname: 'Pramod', citiesvisited: ['Chicago', 'London', 'Pune', 'Bangalore'], addresses: [{ state: 'AK', city: 'DILLINGHAM', type: 'R'}, { state: 'MH', city: 'PUNE', type: 'R' }], lastcity: 'Chicago'});

References: http://www.mongodb.org/
NoSQL distilled : a brief guide to the emerging world of polyglot persistence / Pramod J Sadalage, Martin Fowler.

Document databases: MongoDB

mongoDB
{name: "mongo", type:"DB"}

[DEMO] - http://www.mongodb.org/#

Querying:

- **All document in docName colleciton:**
  db.docName.find();
  SQL: select * from docName
- **Documents which satisfy a condition:**
  db.docName.find({firstname:"Martin"});
  Equivalent to SQL query:
  select * from docName where firstname = "Martin"

References: http://www.mongodb.org/
NoSQL distilled : a brief guide to the emerging world of polyglot persistence / Pramod J Sadalage, Martin Fowler.

Document databases: MongoDB

mongoDB
{name: "mongo", type:"DB"}

- **SQL:**
  - SELECT * FROM customerOrder, orderItem, product
    WHERE
    customerOrder.orderId = orderItem.customerOrderId
    AND orderItem.productId = product.productId
    AND product.name LIKE '%Big Data%'
- **MongoDB:**
  - db.docName.find({"orders.productName":'Big Data'});

References: http://www.mongodb.org/
NoSQL distilled : a brief guide to the emerging world of polyglot persistence / Pramod J Sadalage, Martin Fowler.

Document databases: Usage

- When to Use:
  - Event Logging
  - Content Management Systems, Blogging Platforms
  - Web Analytics or Real-Time Analytics
  - E-Commerce Applications
- When Not to Use
  - Complex Transactions Spanning Different Operations
  - Queries against Varying Aggregate Structure

NoSQL distilled : a brief guide to the emerging world of polyglot persistence / Pramod J Sadalage, Martin Fowler.

Key-value vs. document data models

	Key-Value Model	Document Model
Aggregate	**Opaque**	**Transparent**
Access	**Only lookup based on key (return whole aggregation)**	**Queries based on fields in the aggregate (can retrieve parts)**
Index	**-**	**Can create indexes based on the contents of the aggregate**

http://blog.dynatrace.com/2011/10/05/nosql-or-rdbms-are-we-asking-the-right-questions/

Storing Data: NoSQL Database: Column-Family Stores

Column-oriented data model

Id	First Name	Last Name	Salary
1	Joe	Smith	40000
2	Mary	Jones	50000
3	Mike	Johnson	45000

1, Joe, Smith, 40000
2, Mary, Jones, 50000
3, Mike, Johnson, 45000

1, 2, 3
Joe, Mary, Mike
40000,50000,45000

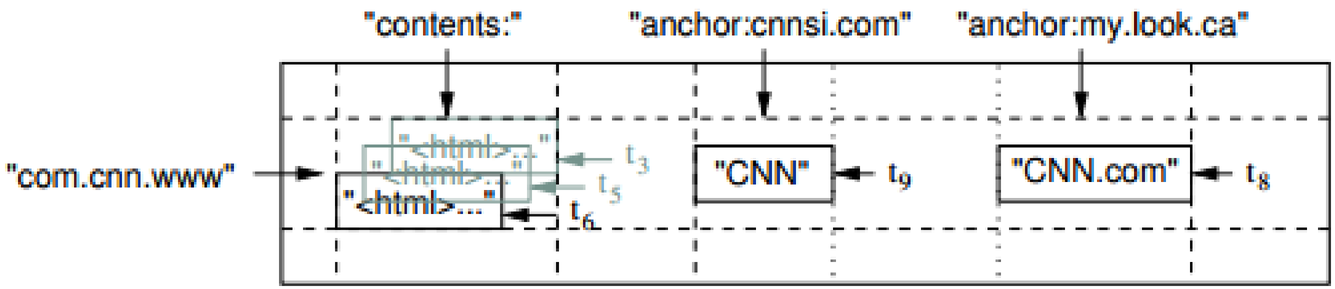
http://en.wikipedia.org/wiki/Column-oriented_DBMS

Column-oriented data model

- Column-oriented organizations are more efficient when an aggregate needs to be computed over many rows but only for a notably smaller subset of all columns of data, because reading that smaller subset of data can be faster than reading all data.
- Column-oriented organizations are more efficient when new values of a column are supplied for all rows at once, because that column data can be written efficiently and replace old column data without touching any other columns for the rows.
- Row-oriented organizations are more efficient when many columns of a single row are required at the same time, and when row-size is relatively small, as the entire row can be retrieved with a single disk seek.
- Row-oriented organizations are more efficient when writing a new row if all of the column data are supplied at the same time, as the entire row can be written with a single disk seek.

http://en.wikipedia.org/wiki/Column-oriented_DBMS

Column-oriented data model



- **Data indexed by:**
  - **(row:string, column:string, time:int64) → string**
- **#of distinct column families - small (in hundreds)**
- **unbounded number of columns**
- **Data processing is pushed to the application**

NoSQL distilled : a brief guide to the emerging world of polyglot persistence / Pramod J Sadalage, Martin Fowler.

Column-family databases

Conceptual View:

Row Key	Time Stamp	Column Family *contents*	Column Family *anchor*
“com.cnn.www”	**t9**		**anchor:cnnsi.com=“CNN”**
“com.cnn.www”	**t8**		**anchor:my.look.ca=“CNN.com”**
“com.cnn.www”	**t6**	**contents:html=“<html>…”**	
“com.cnn.www”	**t5**	**contents:html=“<html>…”**	
“com.cnn.www”	**t3**	**contents:html=“<html>…”**	

Physical View

Column Family *anchor*

Row Key	Time Stamp	Column Family *anchor*
“com.cnn.www”	**t9**	**anchor:cnnsi.com=“CNN”**
“com.cnn.www”	**t8**	**anchor:my.look.ca=“CNN.com”**

Column Family *contents*

Row Key	Time Stamp	Column Family *contents*
“com.cnn.www”	**t6**	**contents:html=“<html>…”**
“com.cnn.www”	**t5**	**contents:html=“<html>…”**
“com.cnn.www”	**t3**	**contents:html=“<html>…”**

http://hbase.apache.org/book.html#datamodel

Column-family databases: HBase

- HBase is the Hadoop database
  - Distributed, scalable, big data store
- Use HBase when you need random, real-time read/write access to your Big Data
- Goal is to host very large tables:
  - billions of rows X millions of columns
- HBase is an open-source, distributed, versioned, column-oriented store modeled after Google's Bigtable

http://hbase.apache.org/

Column-family databases: HBase

- No SQL-like query language
  - Java API
  - HBase Shell
    - » create ‘test’, ‘cf’
    - » put ‘test’, ‘row1’, ‘cf:a’, ‘value1’
    - » put ‘test’, ‘row2’, cf:b’, ‘value2’
    - » get ‘test’, ‘row1’
      - COLUMN CELL
        cf:a timestamp=1288380727188, value=value1
- HBql – separate project to simplify the usage of Hbase (hbql.com)

http://hbase.apache.org/book/quickstart.html#shell_exercises

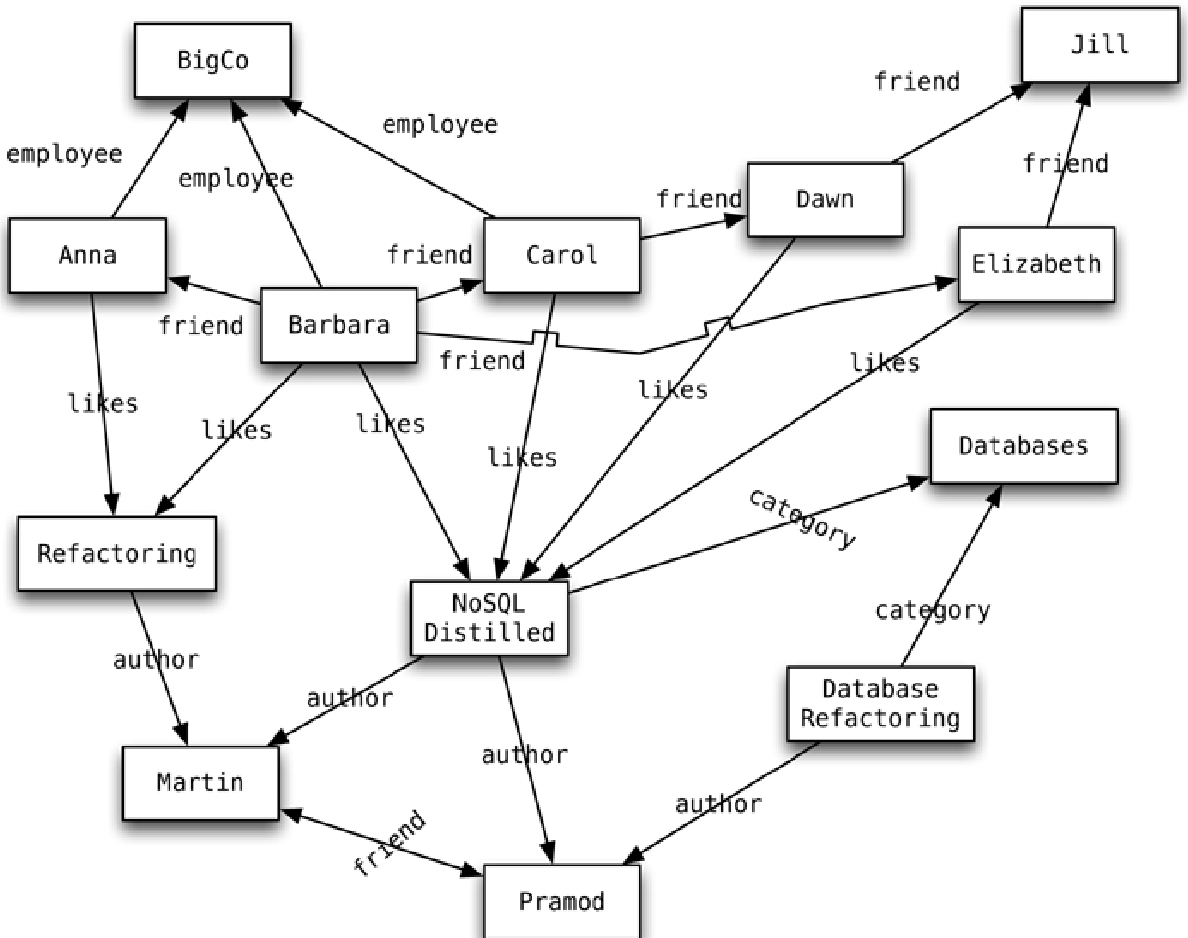
Column-family databases: Usage

- When to use:
  - Event logging
  - Content management systems, blogging platforms
  - Expiring usage
  - Need aggregate using, e.g., SUM or AVG
- When not to use
  - Frequent changes to the database (inserts and deletes maybe expensive)

References: http://wiki.basho.com/Riak.html; NoSQL distilled : a brief guide to the emerging world of polyglot persistence / Pramod J Sadalage, Martin Fowler.

Storing Data: NoSQL Database:
Graph Databases

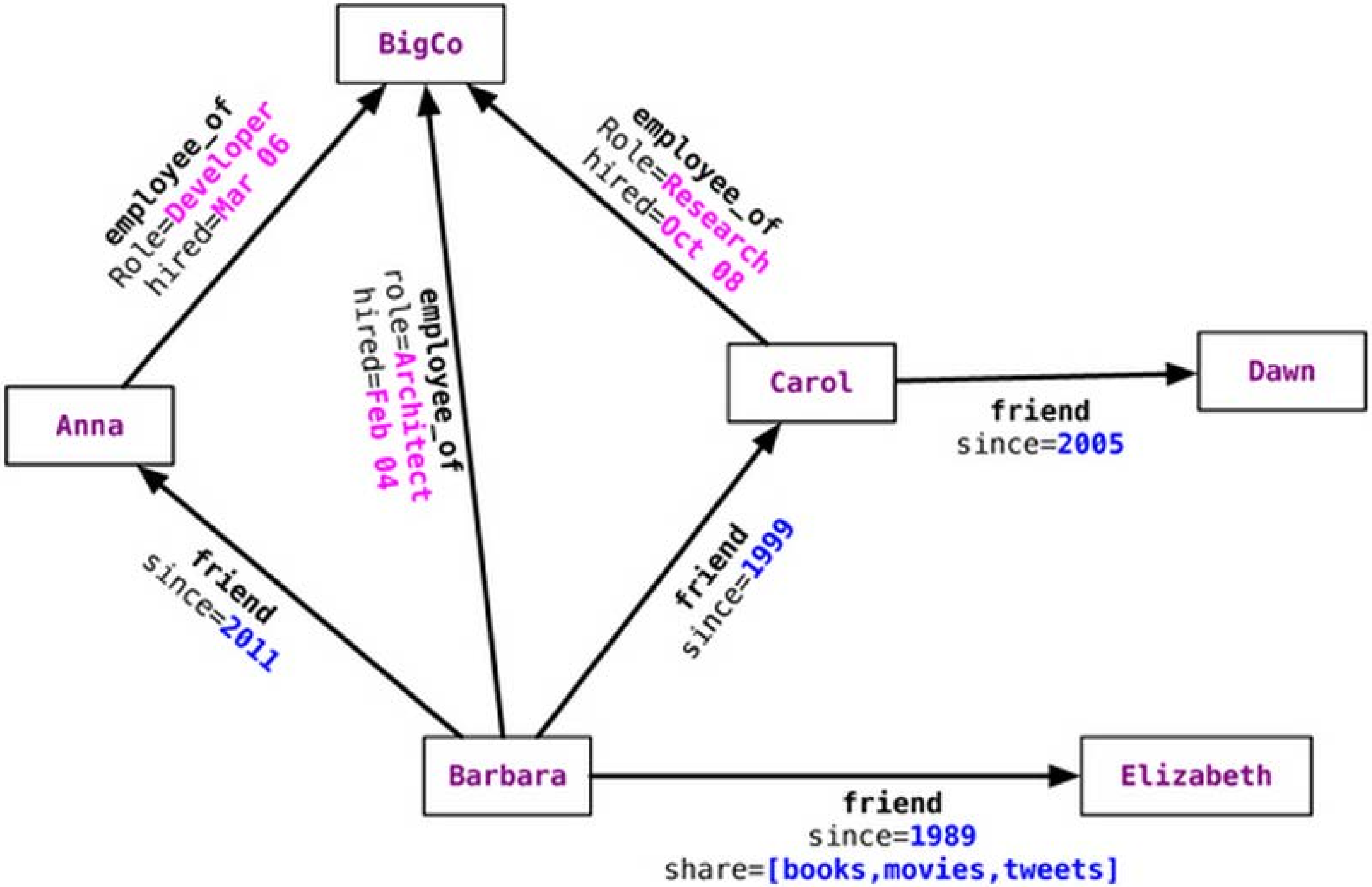
Graph databases



An example graph structure

NoSQL distilled : a brief guide to the emerging world of polyglot persistence / Pramod J Sadalage, Martin Fowler.

Graph databases



Relationships with properties

NoSQL distilled : a brief guide to the emerging world of polyglot persistence / Pramod J Sadalage, Martin Fowler.

Graph Databases: Neo4j

- **Node creation:**
  - `Node martin = graphDb.createNode();`
  - `martin.setProperty("name", "Martin");`
  - `Node pramod = graphDb.createNode();`
  - `pramod.setProperty("name", "Pramod");`
- **Relationship creation:**
  - `martin.createRelationshipTo(pramod, FRIEND);`
  - `pramod.createRelationshipTo(martin, FRIEND);`

NoSQL distilled : a brief guide to the emerging world of polyglot persistence / Pramod J Sadalage, Martin Fowler.

Graph Databases: Neo4j, Querying

Create index:

```
Transaction transaction = graphDb.beginTx();
    try {
        Index<Node> nodeIndex = graphDb.index().forNodes("nodes");
        nodeIndex.add(martin, "name", martin.getProperty("name"));
        nodeIndex.add(pramod, "name", pramod.getProperty("name"));
        transaction.success();
    } finally {
        transaction.finish();
    }
```

Relationship creation:

```
Node martin = nodeIndex.get("name", "Martin").getSingle();
    allRelationships = martin.getRelationships();
incomingRelations = martin.getRelationships(Direction.INCOMING);
```

NoSQL distilled : a brief guide to the emerging world of polyglot persistence / Pramod J Sadalage, Martin Fowler.

Graph Databases: Neo4j, Querying

Neo4j
the graph database

- Traverse the graphs at any depth:
  - Node barbara = nodeIndex.get("name", "Barbara").getSingle();
  - Traverser friendsTraverser = barbara.traverse(Order.BREADTH_FIRST,
    StopEvaluator.END_OF_GRAPH,
    ReturnableEvaluator.ALL_BUT_START_NODE,
    EdgeType.FRIEND,
    Direction.OUTGOING);
- Finding paths between two nodes:
  - Node barbara = nodeIndex.get("name", "Barbara").getSingle();
    Node jill = nodeIndex.get("name", "Jill").getSingle();
    PathFinder<Path> finder = GraphAlgoFactory.allPaths(
    Traversal.expanderForTypes(FRIEND,Direction.OUTGOING)
    ,MAX_DEPTH);
    Iterable<Path> paths = finder.findAllPaths(barbara, jill);

NoSQL distilled : a brief guide to the emerging world of polyglot persistence / Pramod J Sadalage, Martin Fowler.

Graph Databases: Neo4j, Cypher QL

Cypher query language:

- **START beginingNode = (beginning node specification)**
  - **MATCH (relationship, pattern matches)**
  - **WHERE (filtering condition: on data in nodes and relationships)**
  - **RETURN (What to return: nodes, relationships, properties)**
  - **ORDER BY (properties to order by)**
  - **SKIP (nodes to skip from top)**
  - **LIMIT (limit results)**

NoSQL distilled : a brief guide to the emerging world of polyglot persistence / Pramod J Sadalage, Martin Fowler.

Graph Databases: Neo4j, Cypher QL

- **Find all nodes connected to Barbara, either incoming or outgoing:**
  - **START barbara = node:nodeIndex(name ="Barbara")**

    MATCH (barbara)--(connected_node)

    RETURN connected_node
- **When we are interested in directional significance:**
  - **For incoming relationship**

    MATCH (barbara) <-- (connected_node)
  - **For outgoing relationship**

    MATCH (barbara) --> (connected_node)

NoSQL distilled : a brief guide to the emerging world of polyglot persistence / Pramod J Sadalage, Martin Fowler.

Graph Databases: Neo4j, Cypher QL

- Match can also be done on specific relationships using the :RELATIONSHIP_TYPE convention and returning the required fields or nodes:
  - START barbara = node:nodeIndex(name = "Barbara")

    MATCH (barbara)-[:FRIEND]->(friend_node)

    RETURN friend_node.name,friend_node.location
- Query for relationships where a particular relationship property exists:
  - START barbara = node:nodeIndex(name = "Barbara")

    MATCH (barbara)-[relation]->(related_node)

    WHERE type(relation) = 'FRIEND'

    RETURN related_node.name, relation.since

NoSQL distilled : a brief guide to the emerging world of polyglot persistence / Pramod J Sadalage, Martin Fowler.

Graph databases: Usage

- When to Use:
  - Connected Data
  - Routing, Dispatch, and Location-Based Services
  - Recommendation Engines
- When Not to Use
  - When you want to update all or a subset of entities
  - Not “graph” data model

NoSQL distilled : a brief guide to the emerging world of polyglot persistence / Pramod J Sadalage, Martin Fowler.

NoSQL databases: Summary

- Key-value databases
- Document databases
- Column-family stores
- Graph databases

NoSQL Databases: Goals

- Not using the relational model
- Schemaless
- Running well on clusters
- Aggregates – nested data stored together

NoSQL: Schemaless

- Relational DB:
  - You first have to define a schema for your database
- NoSQL:
  - Storing data is more casual:
    - » Key-value store – allow any data under a key
    - » Document DB – no restrictions on the structure of the document
    - » Column-family DB – allow rows have different columns, any data under any column
    - » Graph DB – allow freely adding new edges and properties to nodes and edges

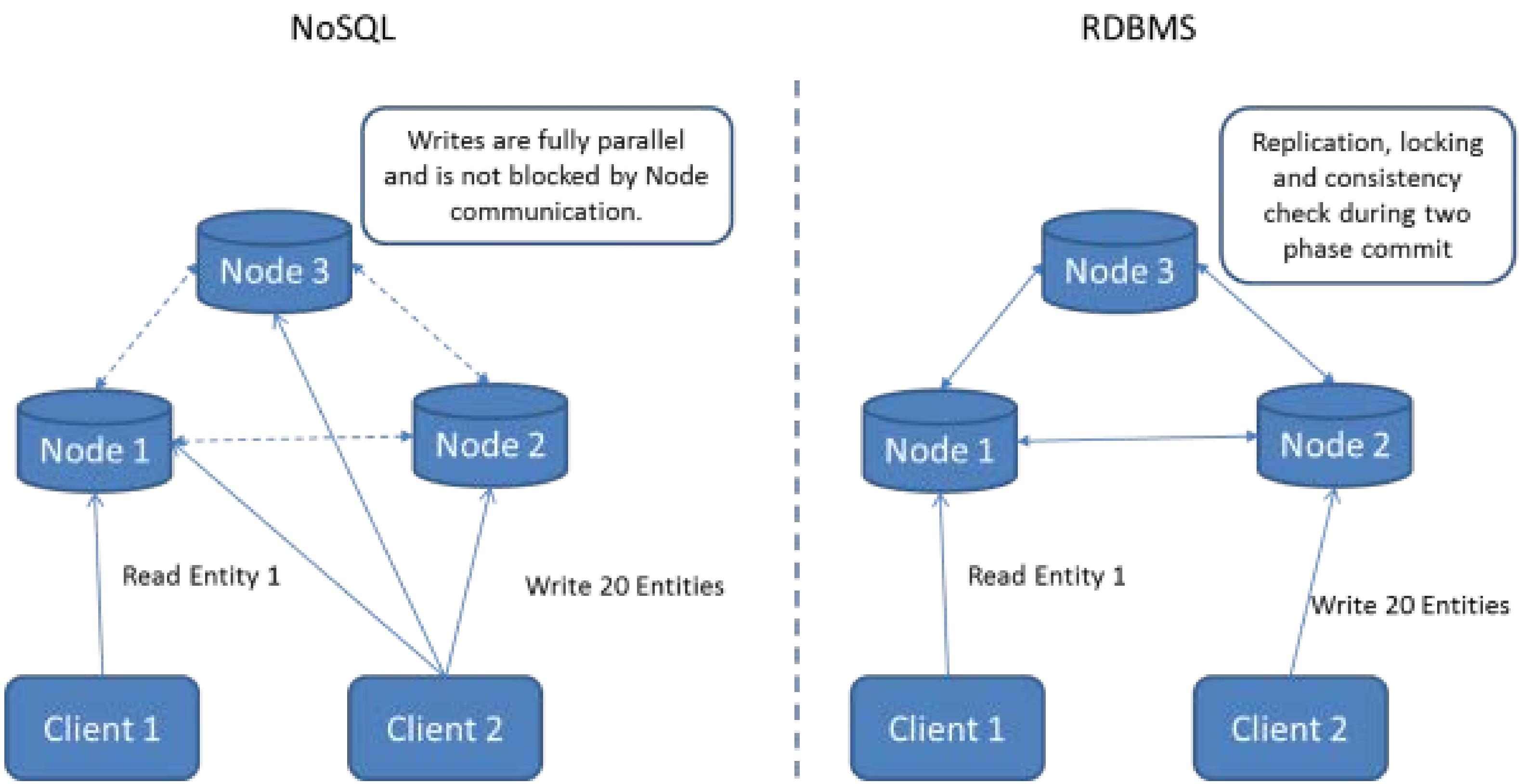
NoSQL: Schemaless

- Advantages:
  - Easy to handle changes
  - Easy to deal with non-uniform data
    - » where each record has different set of fields
- Disadvantages:
  - Database remains ignorant of the schema
    - » Can’t validate data types
  - Implicit schema in the application code
    - » Bad and/or complicated code
    - » Multiple application access the same database

NoSQL Databases – BASE properties

- **Basically available: Use replication to reduce the likelihood of data unavailability and use "sharding" (partitioning the data among many different storage servers) to make any remaining failures partial. The result is a system that is always available, even if subsets of the data become unavailable for short periods of time.**
- **Soft state: While ACID systems assume that data consistency is a hard requirement, NoSQL systems allow data to be inconsistent and relegate designing around such inconsistencies to application developers.**
- **Eventually consistent: Although applications must deal with instantaneous consistency, NoSQL systems ensure that at some future point in time the data assumes a consistent state. In contrast to ACID systems that enforce consistency at transaction commit, NoSQL guarantees consistency only at some undefined future time.**

Why an RDBMS does not scale and many NoSQL solutions do?



NoSQL differs to RDBMS in the way entities get distributed and that no consistency is enforced across those entities

http://blog.dynatrace.com/2011/10/05/nosql-or-rdbms-are-we-asking-the-right-questions/

NoSQL databases

NoSQL databases:

- Not using the relational model ✓
- Schemaless ✓
- Running well on clusters ✓

Storing Data: Beyond NoSQL

Beyond NoSQL

- File systems
- XML Databases
- Object Databases
- Others …

File systems

- Simple and widely implemented
  - Most of the devices have one or another file system
- More like key-value stores with hierarchic key
  - No support for queries
- Little control over concurrency
  - Simple locking
- Cope with very large entities
  - Video, audio
- Very good for sequence access
- Works best for relatively small number of large files that can be processed in big chunks

XML databases

- Document-like databases
  - Documents are compatible with XML and various XML technologies are used to manipulate the document
- Allow to define schema
  - DTD, XML Schema
- Allow to perform transformation
  - XSLT
- Allow to query documents:
  - XPath, XQuery
  - SQL/XML

Object databases

- Mapping from in-memory data structures to relational tables
- Close integration with the application

Storage: Summary
When and why you (should) choose an RDBMS?

- Table based
- Relations between distinct Table Entities and Rows
- Referential Integrity
- ACID Transactions
- Arbitrary Queries and Joins

http://blog.dynatrace.com/2011/10/05/nosql-or-rdbms-are-we-asking-the-right-questions/

Storage: Summary
Why an RDBMS might not be right for you?

- If you just want to store your application entities in a persistent and consistent way
- If you have hierarchical application objects and need some query capability into them
- If you ever tried to store large trees or networks you will know that an RDBMS is not the best solution here
- If you are running in the Cloud and need to run a distributed database for durability and availability.
- You might already use a data warehouse for your analytics. If your data grows to large to be processed on a single machine, you might look into hadoop or any other solution that supports distributed Map/Reduce.

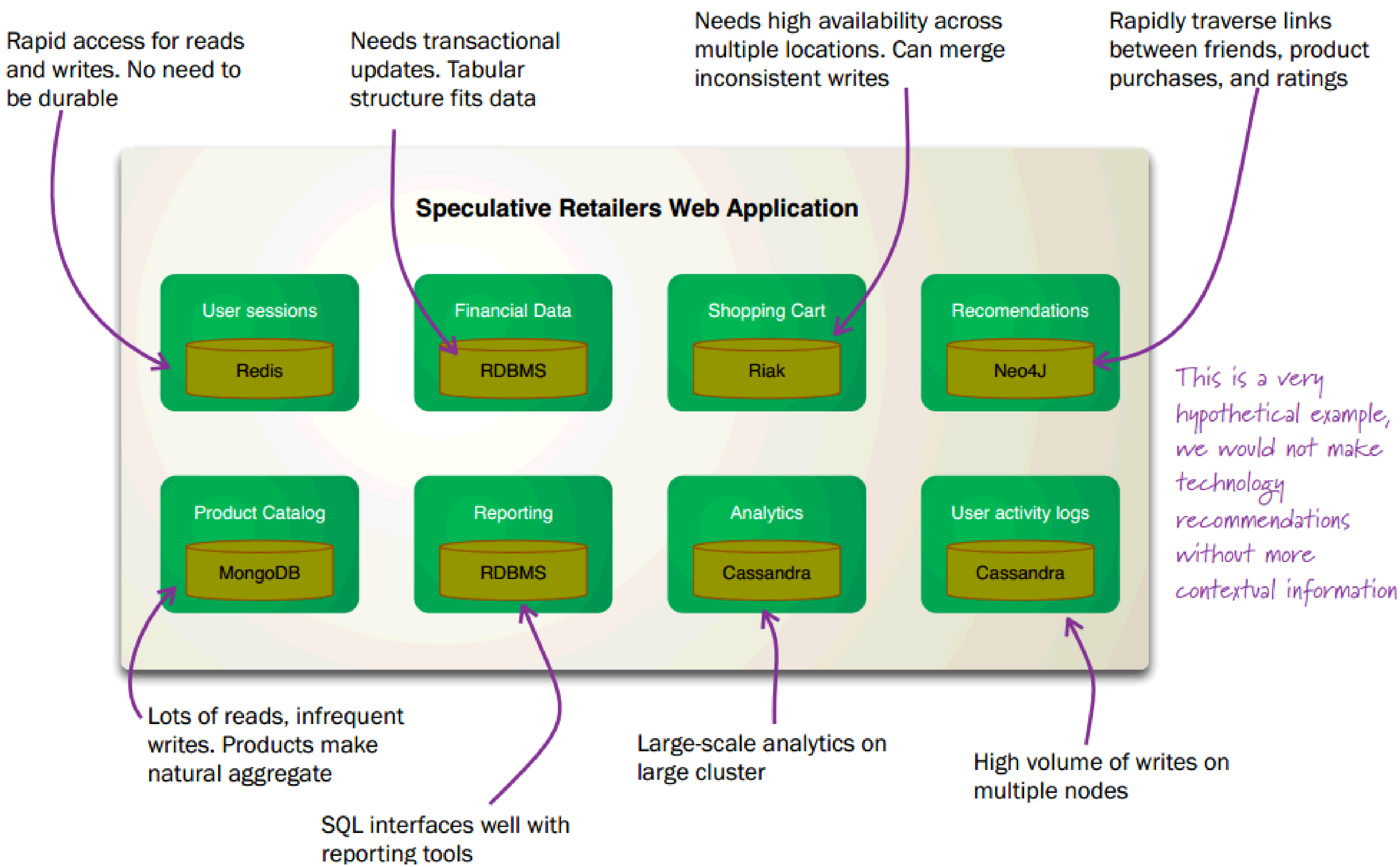
http://blog.dynatrace.com/2011/10/05/nosql-or-rdbms-are-we-asking-the-right-questions/

Storage: Summary
Hybrid Systems? (examples)

- F1 - The Fault-Tolerant Distributed RDBMS Supporting Google's Ad Business
  - Jeff Shute, Mircea Oancea, Stephan Ellner, Ben Handy, Eric Rollins, Bart Samwel, Radek Vingralek, Chad Whipkey, Xin Chen, Beat Jegerlehner, Kyle Littlefield, and Phoenix Tong. 2012. F1: the fault-tolerant distributed RDBMS supporting google's ad business. In Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data (SIGMOD '12). ACM, New York, NY, USA, 777-778. DOI=10.1145/2213836.2213954 http://doi.acm.org/10.1145/2213836.2213954
- Oracle In-Database Hadoop: When MapReduce Meets RDBMS
  - Xueyuan Su and Garret Swart. 2012. Oracle in-database hadoop: when mapreduce meets RDBMS. In Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data (SIGMOD '12). ACM, New York, NY, USA, 779-790. DOI=10.1145/2213836.2213955 http://doi.acm.org/10.1145/2213836.2213955

http://blog.dynatrace.com/2011/10/05/nosql-or-rdbms-are-we-asking-the-right-questions/

Polyglot Persistence



http://martinfowler.com/articles/nosql-intro.pdf

Suggested readings

Raghu Ramakrishnan, Johannes Gehrke. Database Management Systems. 3d Edition. WCB/McGraw-Hill 2003

Michael Stonebraker and Joseph M. Hellerstein , What Goes Around Comes Around (http://mitpress.mit.edu/books/chapters/0262693143chapm1.pdf)

NoSQL distilled: a brief guide to the emerging world of polyglotp ersistence / Pramod J Sadalage, Martin Fowler.

Tom White, Hadoop: The Definitive Guide, 3nd Edition, 2012

Fay Chang et. al, Bigtable: A Distributed Storage System for Structured Data

Jeffrey Dean and Sanjay Ghemaw. MapReduce: simplified data processing on large clusters. Commun. ACM 51, 1 (January 2008)

http://hadoop.apache.org/

http://hadoop.apache.org/docs/r0.20.2/hdfs_design.html

http://hive.apache.org/docs/r0.9.0/

http://hbase.apache.org/

http://pig.apache.org/docs/r0.8.1/

http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/en/us/pubs/archive/38125.pdf