



NoSQL Databases

Data Models

Semi and Unstructured data

Types of NoSQL Databases

MongoDB

Unstructured data

- Big Data means various sources of information, such as:
 - Activity Data
 - Conversation Data
 - Photo and Video Image Data
 - Sensor Data
 - The Internet of Things Data
 - Satellite Data
 - Astronomical Data
- Data generated by such systems will not always be in a structured format, as seen in relational database
- In this lecture we will look at NoSQL databases, which are currently used to store Big Data

Data Models

- Data Models describe Data Characteristics
- For example, Oracle is based on the relational model and data is stored in a tabular format:

DEPT Table

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

Other structures include:

```
DEPT: {  
    deptno: number,  
    dname: string  
    loc: string  
}
```

For example:

```
DEPT: {  
    deptno: 10,  
    dname: 'ACCOUNTING',  
    loc: 'NEW YORK'  
}
```

Data Models – Structured data

- A table in a relational database is an example of structured data
- Example: DEPT and EMP

DEPT Table

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

All rows in both tables follow the same structure

All columns have the same data type

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7876	ADAMS	CLERK	7788	19-NOV-15	1100		20
7499	ALLEN	SALESMAN	7698	20-FEB-95	1600	300	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUN-89	2450		10
7902	FORD	ANALYST	7566	03-DEC-91	3000		20
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7566	JONES	MANAGER	7839	02-APR-91	2975		20
7839	KING	PRESIDENT		17-NOV-80	5000		10
7654	MARTIN	SALESMAN	7698	28-SEP-93	1250	1400	30
7934	MILLER	CLERK	7782	23-JAN-85	1300		10
7788	SCOTT	ANALYST	7566	16-OCT-15	3000		20
7369	SMITH	CLERK	7902	17-DEC-90	800		20
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7521	WARD	SALESMAN	7698	22-FEB-94	1250	500	30

Data Models – Semi-Structured data

- Not all data can conform to a structured model.
- What if the data for our DEPT table looked like this:

```
DEPT: {  
  deptno: 10,  
  dname: 'ACCOUNTING',  
  loc: 'NEW YORK',  
  auditor: 'KPMG'  
}
```

```
DEPT: {  
  deptno: 30,  
  dname: 'SALES',  
  loc: 'CHICAGO',  
  budget: 50,000  
}
```

Not all records
have the same
structure

```
DEPT: {  
  deptno: 20,  
  dname: 'RESEARCH',  
  loc: 'DALLAS',  
  prof: 'Prof Green',  
  status: 'Silver'  
}
```

```
DEPT: {  
  deptno: 40,  
  dname: 'OPERATIONS',  
  loc: 'BOSTON'  
}
```

There is some
structure, just not
constant

This data is called
semi-structured

Data Models – Unstructured data

- Some data may have no structure at all:

There is no apparent structure to this data.

Nor any clues to what the figures may mean.

Very hard to give meaning to this data!

1	0	1	3	4	5	6	7	8	10	11	12	13	14	15	16	16			
2	0	2	2	4	8	11	14	16	16	16									
3	0	4	13	14	14	15	17	19											
4	0	8	8	14	16	17	17	18	19	19									
5	0	4	5	5	5	7	7	14	14	15	17	17	19	19					
6	0	2	2	6	11	12	15												
7	0	3	4	7	10	13	13	15	16	16	17	18	19	19					
8	4	7																	
9	1	3	4	5	5	5	6	6	9	10	10	11	12	13	13	14	17	18	18
10	3	3	4	5	5	5	7	9	9	9	10	10	12	13	14	17	18		
11	3	3	3	8	9	11	11	12	15	15	17	18	19						
12	2	2	6	13	13	16													
13	7	9	11	16	17														
14	0	1	2	3	14	15	15	16	18	18	18								
15	2	2	4	7	8	9	11	11	13	13	15	16	16	18					
16	1	4	4	5	5	5	9	9	10	12	15	17	17	18	19				
17	0	1	3	4	4	6	7	9	10	14	14	16	17	17					
18	0	1	5	14															
19	3	8	10	16	17														
20	1	1	2	5	6	6	7	8	8	9	9	14	14	15	16	16	16	18	
21	2	5	8	10	19														

Data Models

- It helps to know what structure the data is in.
- Helps determine how the data could be stored or manipulated
- Data models also determine:
 - What operations can be carried out
 - E.g., numerical or data functions
 - What constraints can be added
 - E.g., Today's data minus the DOB must be greater than 18
- Structured data can be held in a relational database
- Need something different for semi-structured or unstructured data.
- This is where NoSQL database are becoming popular for handling semi-structured data.

NoSQL

- Increasingly used to manage a variety of data types in big data.
- These data stores are databases that do not represent data in a table format with columns and rows as with conventional relational databases.
- Examples of these data stores include Cassandra, MongoDB and HBASE.
- NoSQL data stores provide APIs to allow users to access data.
- These APIs can be used directly or in an application that needs to access the data.
- Additionally, most NoSQL systems provide data access via a web service interface, such a REST

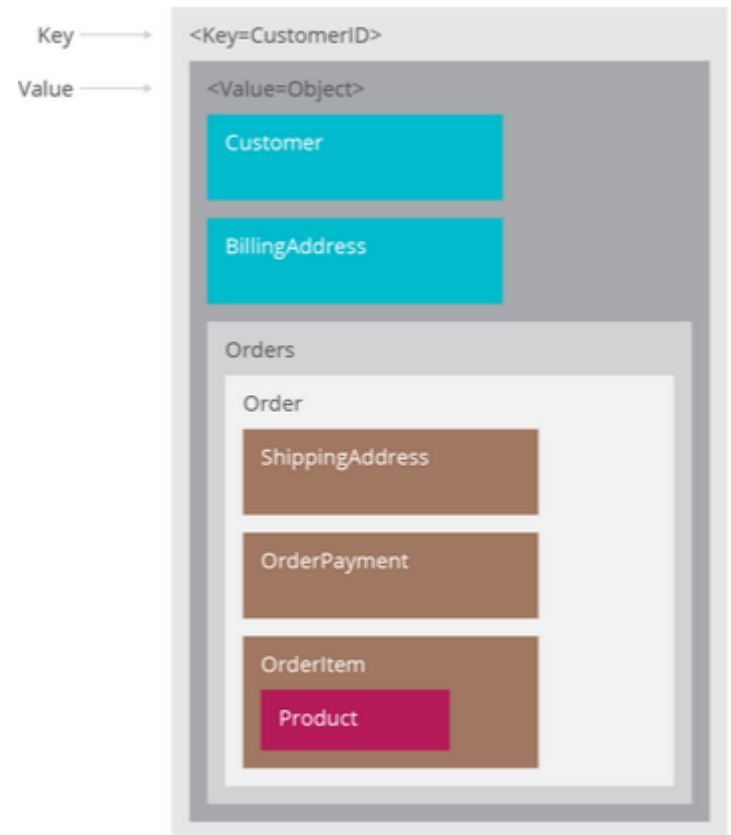


Types of NoSQL Databases

- There are four main types of NoSQL databases:
 - Key-Value Databases
 - Document Databases
 - Column family stores
 - Graph Databases

Key Value Databases

- Key-value stores are the simplest NoSQL data stores to use from an API perspective.
- The client can either get the value for the key, put a value for a key, or delete a key from the data store.
- The value is a blob that the data store just stores, without caring or knowing what's inside. It is the responsibility of the application to understand what was stored.
- Key-value stores always use primary-key access, so they generally have great performance and can be easily scaled.



Key Value Databases

■ Examples include:

- Riak (<http://basho.com/riak/>)
- Redis (<http://redis.io/>) often referred to as Data Structure server
- Memcached (<http://memcached.org/>)
- Berkeley DB now Oracle Berkeley DB 12c
- HamsterDB (<http://hamsterdb.com/>) especially suited for embedded use
- Amazon DynamoDB (<http://aws.amazon.com/documentation/dynamodb/>) (not open-source)
- Project Voldemort (<http://www.project-voldemort.com/voldemort/>)
- Couchbase (<http://www.couchbase.com/>)

Key Value Databases

- Note, all the previous products are not the same and have major differences.
 - For example: Memcached data is not persistent whereas Riak is.
 - This means if using:
 - Memcached and the node goes down all the data is lost and needs to be refreshed from the original source data;
 - Riak you do not need to worry about losing data, but we need to consider how to update data

Document Databases

- Documents are the main concept in document databases.
- The database stores and retrieves documents, which can be XML, JSON, BSON, etc.
- These documents are self-describing, hierarchical tree data structures which can consist of maps, collections, and scalar values.
- The documents stored are similar to each other but do not have to be exactly the same.
- Document databases store documents in the value part of the key-value store
 - think about document databases as key-value stores where the value is examinable.
- Document databases such as MongoDB provide a rich query language and constructs such as database, indexes etc. allowing for easier transition from relational databases.

```
<Key=CustomerID>
{
  "customerid": "fc986e48ca6" ← Key
  "customer":
  {
    "firstname": "Pramod",
    "lastname": "Sadhalage",
    "company": "ThoughtWorks",
    "likes": [ "Biking", "Photography" ]
  }
  "billingaddress":
  { "state": "AK",
    "city": "DILLINGHAM",
    "type": "R"
  }
}
```

Examples include:
[MongoDB](#), [CouchDB](#),
[Terrastore](#), [OrientDB](#),
[RavenDB](#) and Lotus
Notes

Column Family stores



- Column-family databases store data in column families as rows that have many columns associated with a row key.
- Column families are groups of related data that is often accessed together.
- For a Customer, we would often access their Profile information at the same time, but not their Orders.
- Each column family can be compared to a container of rows in an RDBMS table where the key identifies the row and the row consists of multiple columns.
- The difference is that various rows do not have to have the same columns, and columns can be added to any row at any time without having to add it to other rows

Column Family stores

- Examples include:
 - Apache Cassandra
 - HBase
 - Hypertable
 - Amazon DynamoDB
- Apache Cassandra was a top level Apache project born at Facebook and built on Amazon's Dynamo and Google's BigTable
- Cassandra aims to run on top of an infrastructure of hundreds of nodes
 - Possibly spread across different data centres in multiple geographic areas
 - Users include Apple, Comcast, eBay and Netflix

Column Family stores

Each record is divided into Column Families

Each row has a Key

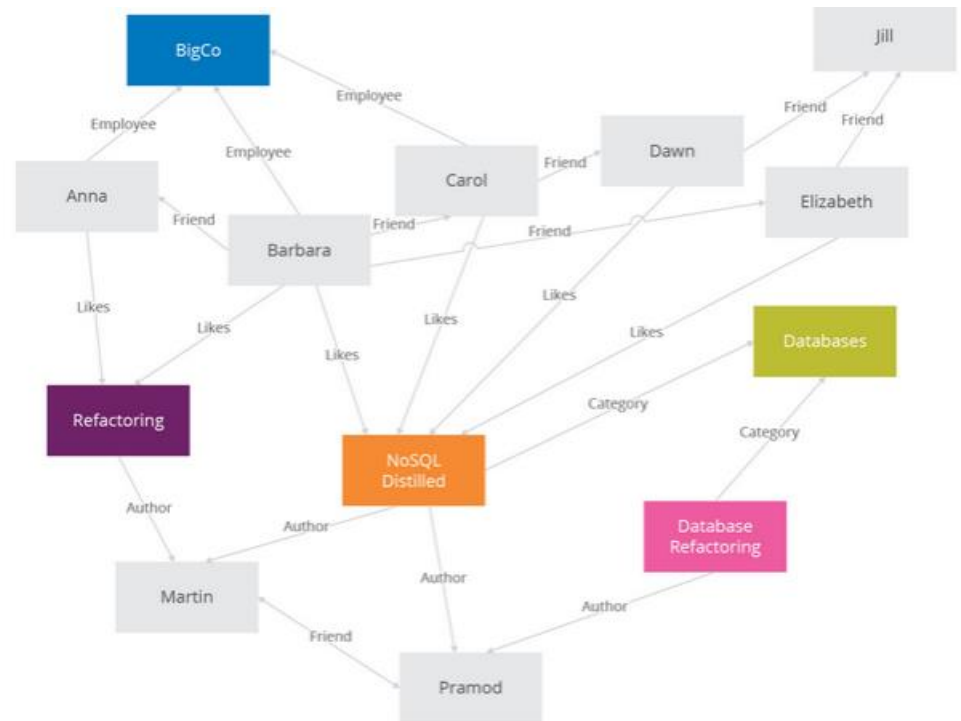
PERSON TABLE					
row key	personal_data		demographic		...
PersonID	Name	Address	BirthDate	Gender	...
1	H. Houdini	Budapest, Hungary	1926-10-31	M	
2	D. Copper	New Jersey, USA	1956-09-16	M	
3	Merlin	Stonehenge, England	1136-12-03	F	
...	
500,000,000	F. Cadillac	Nevada, USA	1964-01-07	M	

Figure 2: Census Data in Column Families

Each column family consists of one or more Columns

Graph Databases

- Graph databases allow you to store entities and relationships between these entities.
- Entities are also known as nodes, which have properties.
- Think of a node as an instance of an object in the application.
- Relations are known as edges that can have properties. Edges have directional significance;
- Nodes are organized by relationships which allow you to find interesting patterns between the nodes.
- The organization of the graph lets the data to be stored once and then interpreted in different ways based on relationships.



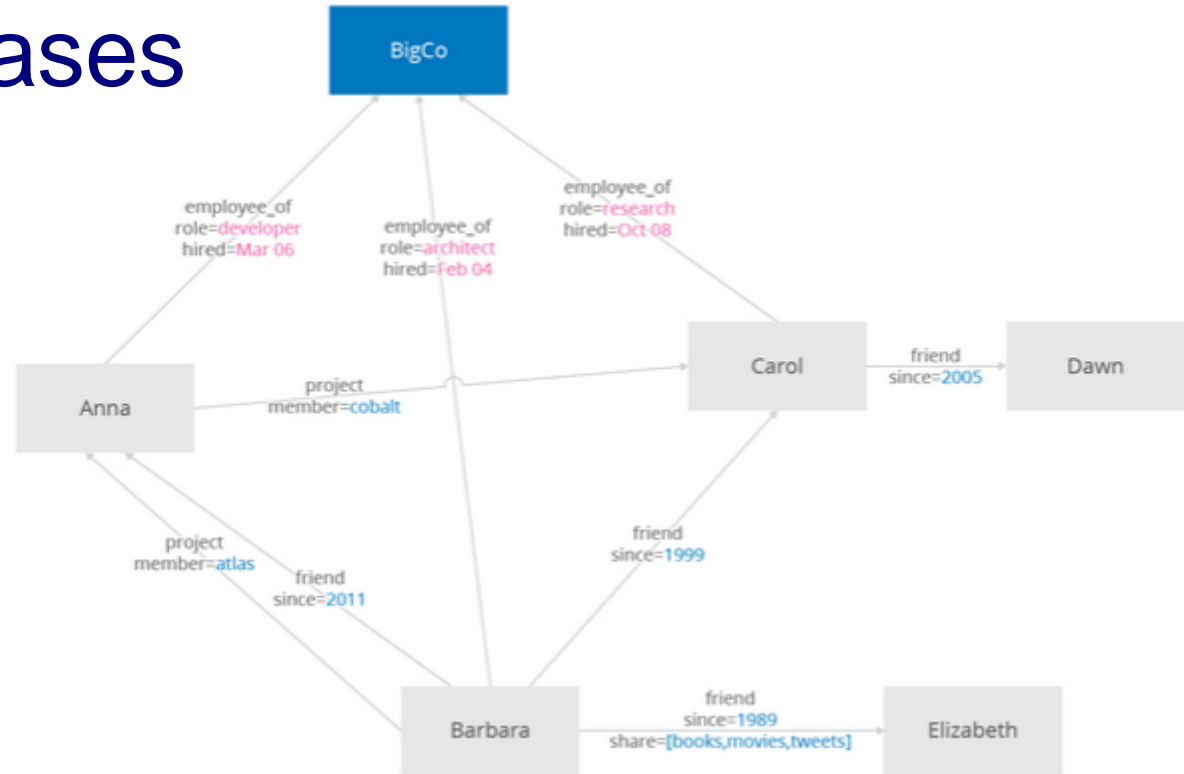
Graph Databases

- A relationship in a RDBMS is graph-like structure for a single type of relationship, for example, "who is my manager".
- In relational databases we model the graph beforehand based on the Traversal we want; if the Traversal changes, the data will have to change.
- Adding another relationship to the mix usually means schema changes.
- This is not the case when we are using graph databases.
- In graph databases, traversing the joins or relationships is very fast.
- The relationship between nodes is not calculated at query time, but is persisted as a relationship.
- Traversing persisted relationships is faster than calculating them for every query.

Graph Databases

Nodes can have different types of relationships between them, allowing you to both represent relationships between the domain entities and to have secondary relationships.

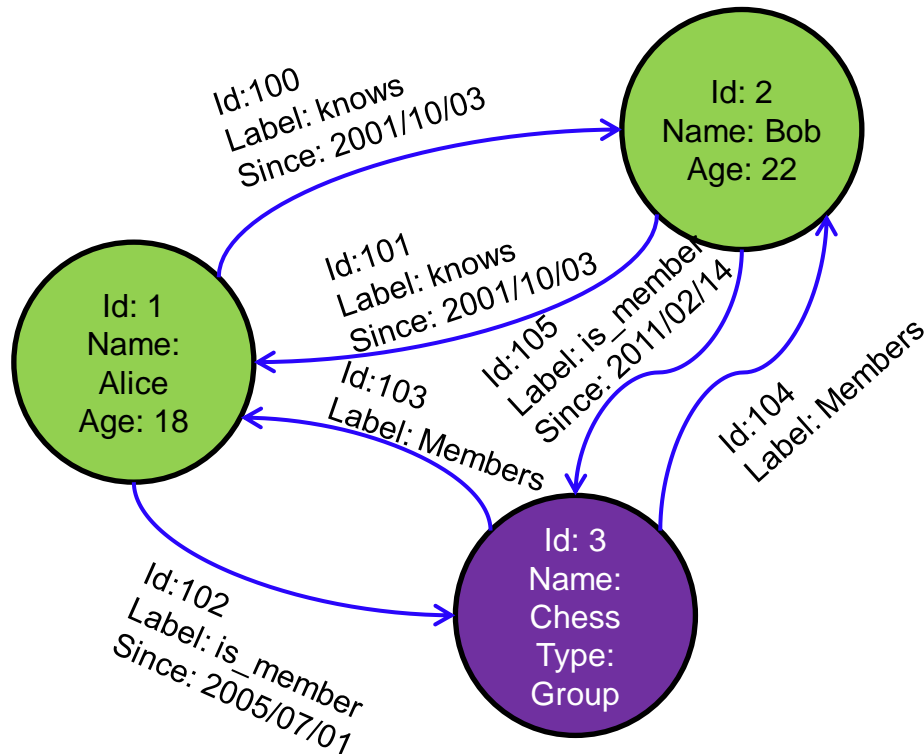
For example, for things like category, path, time-trees, quad-trees for spatial indexing, or linked lists for sorted access.



Since there is no limit to the number and kind of relationships a node can have, they all can be represented in the same graph database.

Graph Databases

- Data are represented as vertices and edges



- Graph databases are powerful for graph-like queries (e.g., find the shortest path between two elements)
- E.g., Neo4j and VertexDB

Graph Databases

- Examples include:

- Neo4J

- Infinite Graph

- OrientDB

- FlockDB

- This is a special case: it only supports single-depth relationships or adjacency lists, where you cannot traverse more than one level deep for relationships.

Choosing NoSQL database

- Some general guidelines (Sadalage 2014):
 - Key-value databases
 - generally useful for storing session information, user profiles, preferences, shopping cart data.
 - avoid using Key-value databases when there is a need to query by data, have relationships between the data being stored or we need to operate on multiple keys at the same time.
 - Document databases
 - generally useful for content management systems, blogging platforms, web analytics, real-time analytics, ecommerce-applications.
 - avoid using document databases for systems that need complex transactions spanning multiple operations or queries against varying aggregate structures.
 - Column family databases
 - generally useful for content management systems, blogging platforms, maintaining counters, expiring usage, heavy write volume such as log aggregation.
 - avoid using column family databases for systems that are in early development, changing query patterns.
 - Graph databases
 - well suited to problem spaces where we have connected data, such as social networks, spatial data, routing information for goods and money, recommendation engines

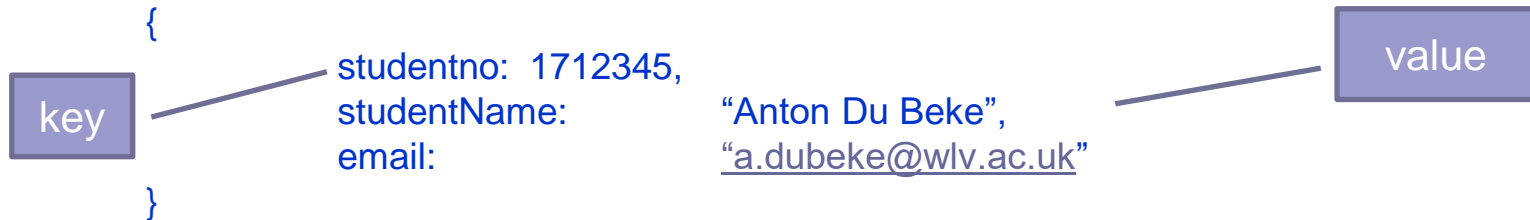
MongoDB

- A MongoDB database works on the concept of collections and documents:
- Database
 - Physical container for the collections.
- Collection
 - Collection is a group of MongoDB documents.
 - It is the equivalent of an RDBMS table, however, they do not enforce a schema, so documents within a collection can have different fields.
 - However, all documents in a collection should have a similar purpose.
 - For example, in e:Vision there could be a collection for student details and module information

MongoDB

■ Document

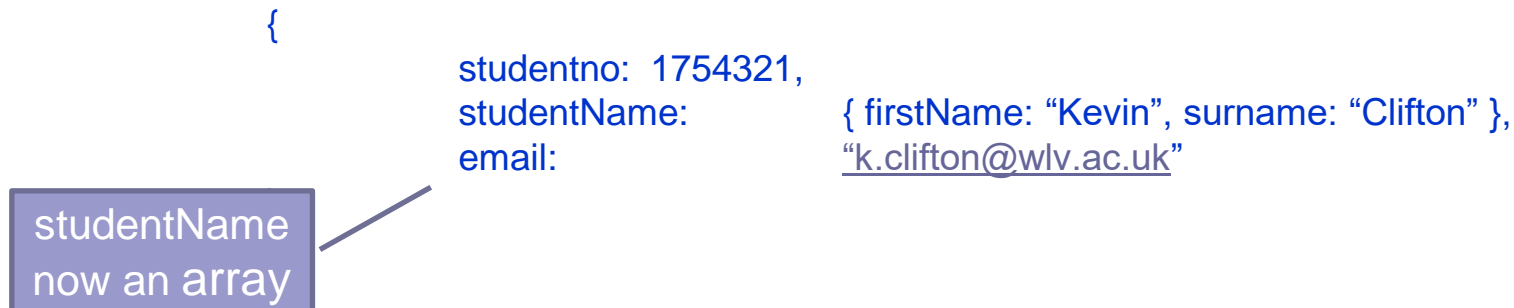
- A document is a set of key-value pairs:



- Documents have a dynamic schema, which means that documents in the same collection do not need to have the same set of columns or structure.



- Common fields in a collection's documents may hold different types of data too.



Document v's Relational Databases

Comparison of terminology in MongoDB (Document database) compared to Oracle (relational database):

Oracle	MongoDB
Table	Collection
Record	Document
Column	Key
Field	Value



DEPT Table

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

Table

Column

Record
(row)

Field (Data item)

Collection

Document

```
csl-student.wlv.ac.uk - PuTTY
> db.dept.find().pretty()
{
  "_id" : ObjectId("5a044a4080c6005cf8dec105"),
  "DEPTNO" : 10,
  "DNAME" : "ACCOUNTING",
  "LOC" : "NEW YORK"
}
{
  "_id" : ObjectId("5a044a4080c6005cf8dec106"),
  "DEPTNO" : 20,
  "DNAME" : "RESEARCH",
  "LOC" : "DALLAS"
}
{
  "_id" : ObjectId("5a044a4080c6005cf8dec107"),
  "DEPTNO" : 30,
  "DNAME" : "SALES",
  "LOC" : "CHICAGO"
}
{
  "_id" : ObjectId("5a044a4180c6005cf8dec108"),
  "DEPTNO" : 40,
  "DNAME" : "OPERATIONS",
  "LOC" : "BOSTON"
}
>
```

Key

Value

Using MongoDB – Collections

- If a collection does not exist, MongoDB will automatically create the collection when you first store data for that collection.
- You can create one in advance, if you want to set any parameters, such as the maximum size.
- To create a new collection:
`db.createCollection('depts', {max: 20})`
 - This creates a collection called depts that can hold a maximum of 20 documents.
- To show what collections you have:
`show collections`

Using MongoDB – Documents

- Data is stored as **BSON** documents in a collection.
 - **BSON** is a binary representation of **JSON** documents.
- **JSON** is built on two structures:
 - A collection of name/value pairs
 - An ordered list of values, such as an array
 - See <http://www.json.org/> or <http://bsonspec.org/> for further details
- To add new data, involves the use of the INSERT command:

```
db.collectionName.insert( {  
    key1: 'value1',  
    keyN: valueN } )
```
- This is similar to an INSERT statement in SQL, where the collectionName would be the table and the key is the column and the value is the data to be stored.
- For example (SQL):

```
INSERT into collectionName (key1, keyN)  
VALUES ('value1', 'valueN');
```

MongoDB – Documents

- Unlike the relational model, you do not need to define a schema in advance for our documents, with the equivalent of a CREATE TABLE statement.
- In fact the structure of the document can change from one document to another.
- This is why it is referred to as being schema-less.
- In reality, most collections would contain documents of a similar type, e.g., a collection of student information.

Object Ids

- MongoDB creates a unique object id for objects in the database.

- For example:

```
db.dept.find().pretty()
```

- Will show an object id for each department document, along the lines of:

```
"_id" : ObjectId("58245944d6473dc2e8d23a26")
```

- This is a system generated, but you can define your own object ids too (like a primary key)

- For example:

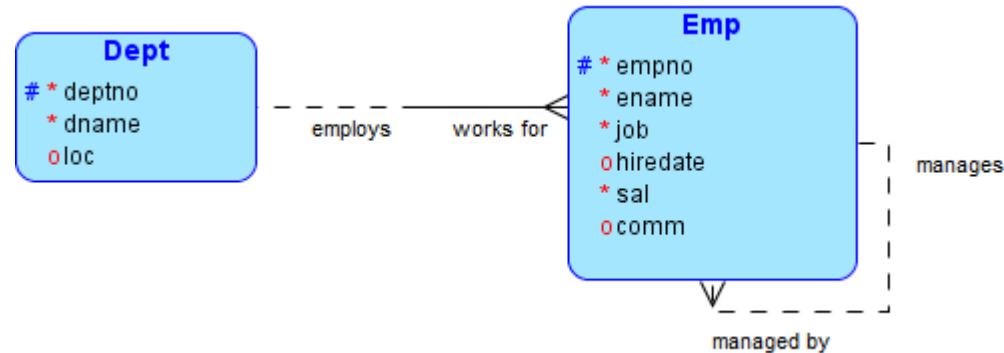
```
db.projCollection.insert( {  
  _id: 10,  
  projno: 110,  
  proj_name: 'Oracle Project',  
  budget: 10000 } )
```

```
db.projCollection.insert( {  
  projno: 140,  
  proj_name: 'ObjectId Project',  
  budget: 25000 } )
```

Or omitting an _id
generates a system id

DEPT/EMP schema

■ Taking our DEPT/EMP scenario:



and data:

EMP Table

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7876	ADAMS	CLERK	7788	19-NOV-15	1100		20
7499	ALLEN	SALESMAN	7698	20-FEB-95	1600	300	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUN-89	2450		10
7902	FORD	ANALYST	7566	03-DEC-91	3000		20
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7566	JONES	MANAGER	7839	02-APR-91	2975		20
7839	KING	PRESIDENT		17-NOV-80	5000		10
7654	MARTIN	SALESMAN	7698	28-SEP-93	1250	1400	30
7934	MILLER	CLERK	7782	23-JAN-85	1300		10
7788	SCOTT	ANALYST	7566	16-OCT-15	3000		20
7369	SMITH	CLERK	7902	17-DEC-90	800		20
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7521	WARD	SALESMAN	7698	22-FEB-94	1250	500	30

DEPT Table

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

Collection Example: Dept

- To create the equivalent of our Dept table:

```
db.dept.insert( [
```

```
{
```

```
  _id: 10,
```

```
  deptno: 10,
```

```
  dname: "ACCOUNTING",
```

```
  loc: "NEW YORK"
```

```
},
```

```
{
```

```
  _id: 20,
```

```
  deptno: 20,
```

```
  dname: "RESEARCH",
```

```
  loc: "DALLAS"
```

```
}
```

```
])
```

Use square brackets to add more than one record

Object identifier

DEPT Table

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

Character data must be in double quotes

Collection Example: Emp

Add some employees:

```
db.emp.insert( [  
  {  
    empno: 7876,  
    ename: "ADAMS",  
    job: "CLERK",  
    mgr: 7788,  
    hiredate: "19-NOV-2015",  
    sal: 1100,  
    deptno: 20  
  },  
  {  
    empno: 7499,  
    ename: "ALLEN",  
    job: "SALESMAN",  
    mgr: 7698,  
    hiredate: "20-FEB-1995",  
    sal: 1600,  
    comm: 300,  
    deptno: 30  
  }  
])
```

Note we have not defined any object IDs this time.

Queries

- Collections can be queried using the find() method.

- The basic syntax is:

`db.collectionName.find()`

- This will return the data in an unstructured way.

- To improve this, use the pretty() method:

`db.collectionName.find().pretty()`

- For example:

`db.dept.find().pretty()`

find()

- `find()` can also contain parameters to restrict what data is returned

- Similar to the `WHERE` part of a SQL statement

- For example, to find the Sales department details:

```
db.dept.find({dname:"SALES"})
```

- Produces:

```
{ "_id" : 30, "deptno" : 30, "dname" : "SALES", "loc" : "CHICAGO" }
```

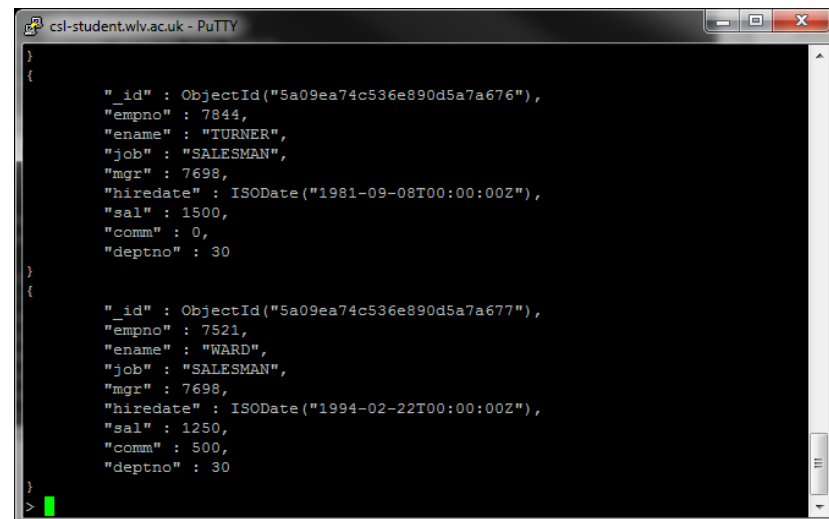
- Find the employees in department 30:

```
db.emp.find({deptno:30})
```

Can use `pretty()` afterwards:

```
db.emp.find({deptno:30}).pretty()
```

Note the difference in `_ids` values



```
csl-student.wlv.ac.uk - PuTTY
{
  "_id" : ObjectId("5a09ea74c536e890d5a7a676"),
  "empno" : 7844,
  "ename" : "TURNER",
  "job" : "SALESMAN",
  "mgr" : 7698,
  "hiredate" : ISODate("1981-09-08T00:00:00Z"),
  "sal" : 1500,
  "comm" : 0,
  "deptno" : 30
}
{
  "_id" : ObjectId("5a09ea74c536e890d5a7a677"),
  "empno" : 7521,
  "ename" : "WARD",
  "job" : "SALESMAN",
  "mgr" : 7698,
  "hiredate" : ISODate("1994-02-22T00:00:00Z"),
  "sal" : 1250,
  "comm" : 500,
  "deptno" : 30
}
>
```

find()

- Note, everything is case sensitive!
 - The following will return nothing:
 - `db.dept.find({DNAME:"SALES"})`
 - `db.DEPT.find({dname:"SALES"})`
 - `db.dept.find({dname:"sales"})`
 - And these will return an error message:
 - `DB.dept.find({dname:"SALES"})`
 - `db.dept.FIND({dname:"SALES"})`
- column names can be in double quotes too:
 - `db.dept.find({"dname":"SALES"})`

Conclusion

- Big data is a big research area currently
 - The data generated will not be going away anytime soon, so need an effectively way of handling it
- NoSQL
 - Lots of different types of projects.
 - Many of the examples listed are open-source.
 - The fact there is no well-defined definition could lead to its downfall.
 - RDBMS could evolve to provide better support for non-structured data.