UMBC

AN HONORS UNIVERSITY IN MARYLAND

CMSC 461, Database Management Systems
Fall 2014

MySQL Views & Comparing SQL to NoSQL

These slides are based on “Database System Concepts” book and slides, 6th edition, and the 2009/2012 CMSC 461 slides by Dr. Kalpakis

Jennifer Sleeman

http://www.csee.umbc.edu/~jsleem1/courses/461

Logistics

- Homework 1 is graded
- Phase 1 grading will begin this week
- Homework 2 is posted
- Phase 2 and data scripts will post soon

Lecture Outline

- Quick Introduction to Views
- NoSQL

Lecture Outline

- *Quick Introduction to Views*
- NoSQL

Views

- Relations stored in the database, logical model level
- May not be desirable for all users to see the entire logical model
- A 'view' of the relation with a subset of information may be more appropriate

Views: A Scenario

- Consider a person who needs to know an instructor's name and department.

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	68250.00
12121	Wu	Finance	94500.00
15151	Mozart	Music	42000.00
22222	Einstein	Physics	99750.00
32343	El Said	History	63000.00
33456	Gold	Physics	91350.00
45565	Katz	Comp. Sci.	78750.00
58583	Califieri	History	65100.00
76543	Singh	Finance	84000.00
76766	Crick	Biology	75600.00
83821	Brandt	Comp. Sci.	96600.00
98345	Kim	Elec. Eng.	84000.00

Views: A Scenario

- We don't necessarily want to share salary
- And possibly ID is not very useful

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	68250.00
12121	Wu	Finance	94500.00
15151	Mozart	Music	42000.00
22222	Einstein	Physics	99750.00
32343	El Said	History	63000.00
33456	Gold	Physics	91350.00
45565	Katz	Comp. Sci.	78750.00
58583	Califieri	History	65100.00
76543	Singh	Finance	84000.00
76766	Crick	Biology	75600.00
83821	Brandt	Comp. Sci.	96600.00
98345	Kim	Elec. Eng.	84000.00

Views: A Scenario

- Instead we may wish to provide this information only

name	dept_name
Srinivasan	Comp. Sci.
Wu	Finance
Mozart	Music
Einstein	Physics
El Said	History
Gold	Physics
Katz	Comp. Sci.
Califieri	History
Singh	Finance
Crick	Biology
Brandt	Comp. Sci.
Kim	Elec. Eng.

Views

- A view provides a mechanism to hide certain data from the view of certain users
- It also provides a way to create a personalized collection of relations
- Any relation that is not of the conceptual model but is made visible to a user as a "***virtual relation***" is called a ***view***.
- You can think of a view as a relation, select from it, join upon it, some views allow deletes, inserts and updates
- There is no data contained in the view, the view data is derived from other relations

View Definition

- A view is defined using the create view statement which has the form

  create view v as < query expression >

  where ***v*** is the view name and
  <query expression> is any legal SQL expression

- Once a view is defined, the view name can be used to refer to the virtual relation that the view generates
- View definition ***is not*** the same as creating a new relation by evaluating the query expression
  - a view definition results in a saved expression which is executed when the view is used

Examples of Views

- A view of instructors without their salary

```
create view faculty as
select ID, name, dept_name;
from instructor
```

- Find all instructors in the Biology department

```
select name
from faculty
where dept_name = 'Biology';
```

- Create a view of department salary totals

```
create view departments_total_salary(dept_name,
total_salary)
 as
select dept_name, sum (salary)
from instructor
group by dept_name;
```

Examples of Views

mysql> ***create view faculty as***
select name, dept_name from instructor;
Query OK, 0 rows affected (0.16 sec)

mysql> ***select * from faculty***;

```
+------------+------------+
| name       | dept_name  |
+------------+------------+
| Srinivasan | Comp. Sci. |
| Wu         | Finance    |
| Mozart     | Music      |
| Einstein   | Physics    |
| El Said    | History    |
| Gold       | Physics    |
| Katz       | Comp. Sci. |
| Califieri  | History    |
| Singh      | Finance    |
| Crick      | Biology    |
| Brandt     | Comp. Sci. |
| Kim        | Elec. Eng. |
+------------+------------+
```

12 rows in set (0.00 sec)

More Examples of Views

```
mysql> select * from faculty natural join course;
+------------+------------+----------+------------+---------+
| dept_name  | name       | course_id | title      | credits |
+------------+------------+----------+------------+---------+
| Comp. Sci. | Srinivasan | CS-190    | Game Design |       4 |
| Comp. Sci. | Srinivasan | CS-315    | Robotics    |       3 |
| Comp. Sci. | Katz       | CS-190    | Game Design |       4 |
| Comp. Sci. | Katz       | CS-315    | Robotics    |       3 |
| Biology    | Crick      | BIO-301   | Genetics    |       4 |
| Comp. Sci. | Brandt     | CS-190    | Game Design |       4 |
| Comp. Sci. | Brandt     | CS-315    | Robotics    |       3 |
+------------+------------+----------+------------+---------+
7 rows in set (0.01 sec)
```

Views Defined Using Other Views

- One view may be used in the expression defining another view
- A view relation v_1 is said to depend directly on a view relation v_2 if v_2 is used in the expression defining v_1
- A view relation v_1 is said to depend on view relation v_2 if either v_1 depends directly to v_2 or there is a path of dependencies from v_1 to v_2
- A view relation v is said to be recursive if it depends on itself

Views Defined Using Other Views

- ***create view*** physics_fall_2009 ***as***
- select course.course_id, sec_id, building, room_number
- from course, section
- where course.course_id = section.course_id
- and course.dept_name = ’Physics’
- and section.semester = ’Fall’
- and section.year = ’2009’;
-
- ***create view*** physics_fall_2009_watson as
- select course_id, room_number
- from physics_fall_2009
- where building= ’Watson’;
-

View Expansion

- A way to define the meaning of views defined in terms of other views
- Let view v_1 be defined by an expression e_1 that may itself contain uses of view relations
- View expansion of an expression repeats the following replacement step:
- ***repeat***
- Find any view relation v_i in e_1
- Replace the view v_i by the expression defining v_i
- ***until*** no more view relations are present in e_1
- As long as the view definitions are not recursive, this loop will terminate

View Expansion

- If we take the previously defined view and expand it

```
create view physics_fall_2009_watson as
(select course_id, room_number
from (select course.course_id, building, room_number
        from course, section
        where course.course_id = section.course_id
            and course.dept_name = 'Physics'
            and section.semester = 'Fall'
            and section.year = '2009')
where building= 'Watson';
```

Materialized Views

- Materializing a view: create a physical table containing all the tuples in the result of the query defining the view
- If relations used in the query are updated, the materialized view result becomes out of date
- Need to maintain the view, by updating the view whenever the underlying relations are updated.

Update of Views

- Can express updates, inserts and deletions using views
- Modifications through views can be problematic
  - Must be translated to the actual relations in the logical model

Update of Views

- If we define the following views:

create view *faculty* ***as***
select ID, name, dept_name;
from instructor

- then insert the following:

insert into faculty values ('30765', 'Green', 'Music');

- We must insert the tuple:

('30765', 'Green', 'Music', null)

- into the instructor relation since we need to provide a salary
- Or we have to reject the insert

Update of Views

- Another problem that occurs:

create view instructor_info as
select ID, name, building
from instructor, department
where instructor.dept_name= department.dept_name;

- Then we insert the following:

insert into instructor_info values ('69987', 'White', 'Taylor');

- Instructor:

```
+-----------+-------------+------+-----+
| Field     | Type        | Null | Key |
+-----------+-------------+------+-----+
| ID        | varchar(5)  | NO   | PRI |
| name      | varchar(20) | NO   |     |
| dept_name | varchar(20) | YES  | MUL |
| salary    | decimal(8,2)| YES  |     |
+-----------+-------------+------+-----+
```

Department:

```
+-----------+--------------+------+-----+
| Field     | Type         | Null | Key |
+-----------+--------------+------+-----+
| dept_name | varchar(20)  | NO   | PRI |
| building  | varchar(15)  | YES  |     |
| budget    | decimal(12,2)| YES  |     |
+-----------+--------------+------+-----+
```

Update of Views

- How do we know which department?
- If multiple departments in Taylor which to choose?
- What if no department related to building Taylor?
- Most SQL implementations allow updates only on simple views
- The from clause has only one database relation.
- The select clause contains only attribute names of the relation, and does not have any expressions, aggregates, or distinct specification.
- Any attribute not listed in the select clause can be set to null
- The query does not have a group by or having clause.

Errors from MySQL

```
mysql> select * from faculty;
+-----------+-----------+
| name      | dept_name |
+-----------+-----------+
| Srinivasan | Comp. Sci. |
| Wu         | Finance    |
| Mozart     | Music      |
| Einstein   | Physics    |
| El Said    | History    |
| Gold       | Physics    |
| Katz       | Comp. Sci. |
| Califieri  | History    |
| Singh      | Finance    |
| Crick      | Biology    |
| Brandt     | Comp. Sci. |
| Kim        | Elec. Eng. |
+-----------+-----------+
12 rows in set (0.00 sec)
```

create view faculty as
select name, dept_name from instructor

```
mysql>  insert into faculty values ('White', 'Math');
ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint fails
(`university`.`instructor`, CONSTRAINT `instructor_ibfk_1` FOREIGN KEY
(`dept_name`) REFERENCES `department` (`dept_name`) ON DELETE SET NULL)
```

Errors from MySQL

```
mysql> create view instructor_info as
    ->  select ID, name, building
    ->  from instructor, department
    ->  where instructor.dept_name=
department.dept_name;
Query OK, 0 rows affected (0.06 sec)

mysql> insert into instructor_info values ('69987', 'White',
'Taylor');ERROR 1394 (HY000): Can not insert into join
view 'university.instructor_info' without fields list
mysql> insert into instructor_info (ID,name,building) values
('69987', 'White', 'Taylor');
ERROR 1393 (HY000): Can not modify more than one
base table through a join view 'university.instructor_info'
```

Update Views

- Create view history_instructors as

```
select *
from instructor
where dept_name= ’History’;
```

- What happens if we insert (’25566’, ’Brown’, ’Biology’, 100000) into history_instructors?

Update Views

create view history_instructors as
*select **
from instructor where dept_name= 'History';

What happens if we insert ('25566', 'Brown', 'Biology', 100000) into history_instructors?

Instructor:

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	68250.00
12121	Wu	Finance	94500.00
15151	Mozart	Music	42000.00
22222	Einstein	Physics	99750.00
25566	***Brown***	***Biology***	***100000.00***
32343	El Said	History	63000.00
33456	Gold	Physics	91350.00
45565	Katz	Comp. Sci.	78750.00
58583	Califieri	History	65100.00
76543	Singh	Finance	84000.00
76766	Crick	Biology	75600.00
83821	Brandt	Comp. Sci.	96600.00
98345	Kim	Elec. Eng.	84000.00

mysql> select * from history_instructors;

ID	name	dept_name	salary
32343	El Said	History	63000.00
58583	Califieri	History	65100.00

Lecture Outline

- Quick Introduction to Views
- **NoSQL**

Why NoSQL?

- Scalability
  - Vertical
    - low performance
    - lots of work
    - expensive
  - Horizontal
    - Auto-sharding

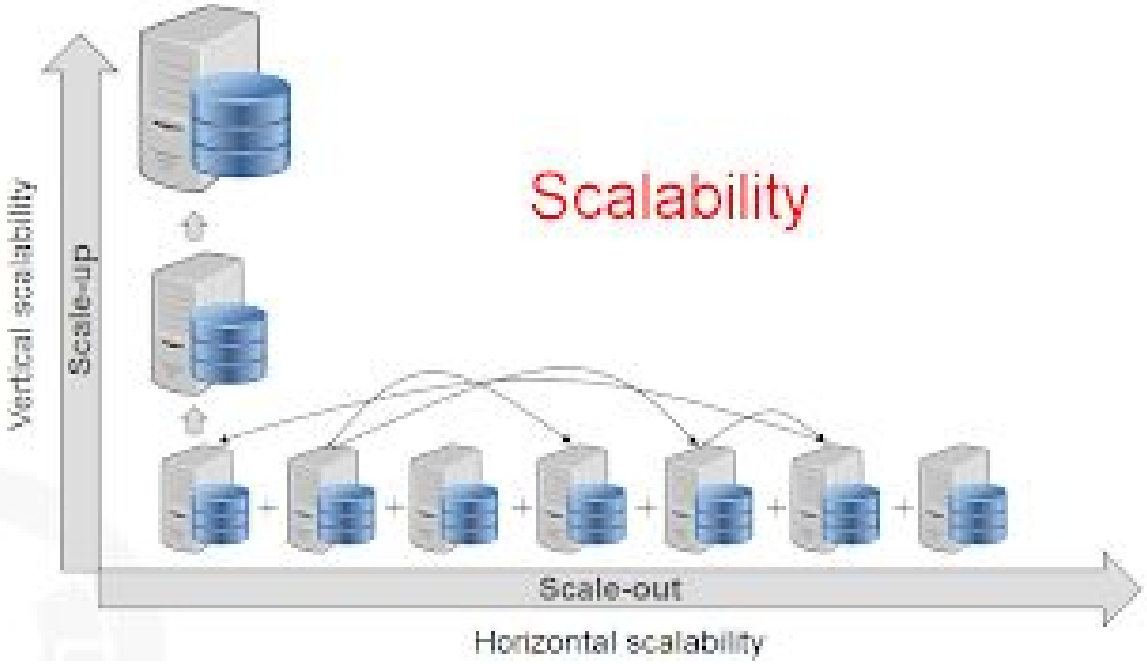


Image credit: http://www.differencebetween.info/difference-between-nosql-and-sql-database

Why NoSQL?

- Flexibility
  - System changes during developmental lifecycle
  - Difficult with relational model
  - Schema-free = rapid application development

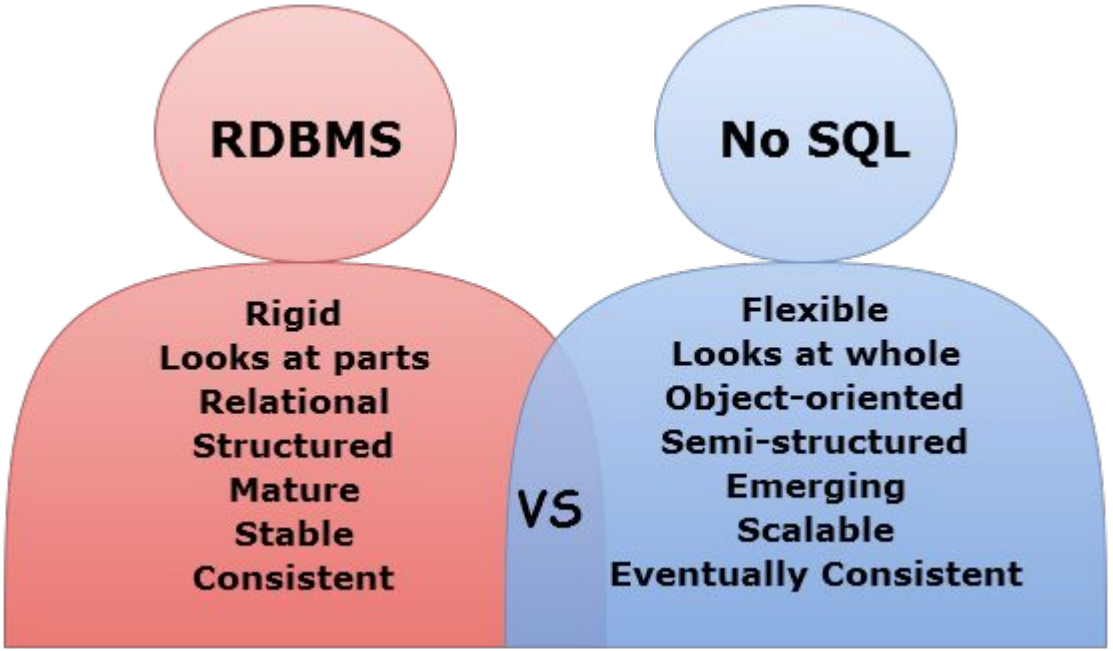
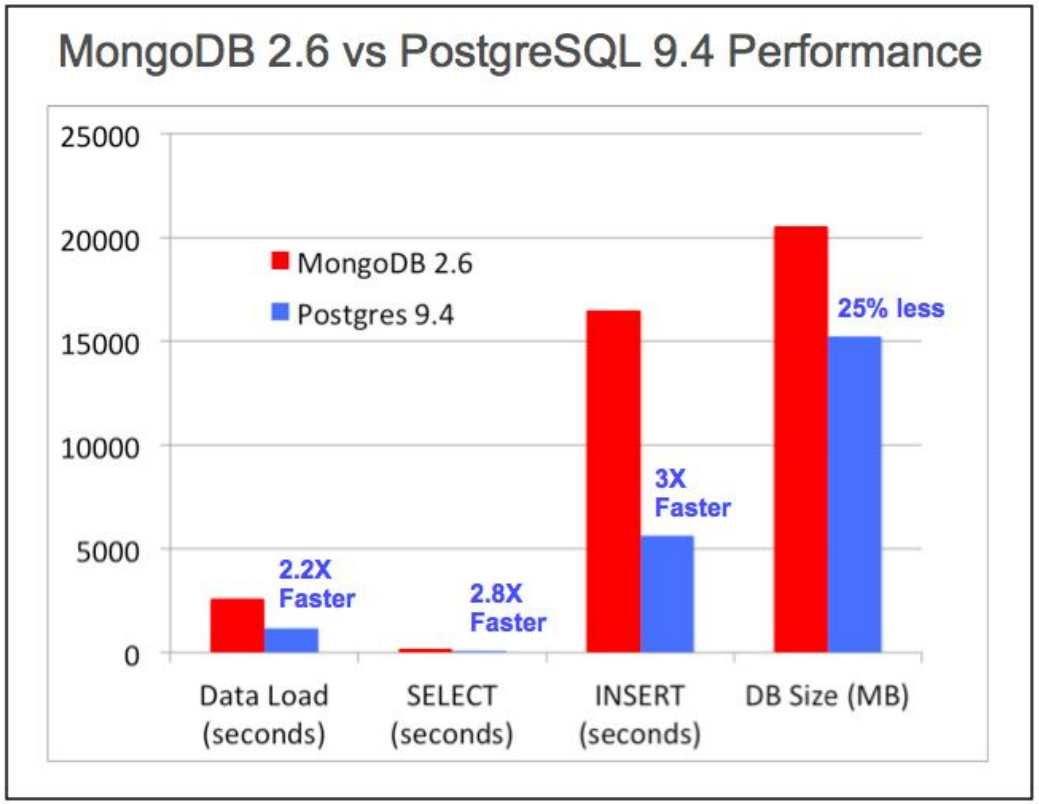


Image credit: https://www.tatvasoft.com/blog/why-nosql-is-required-when-rdbms-is-proven/

Why NoSQL?

- Performance
  - Cross table queries, joining
  - Doesn’t map into software objects well
  - No cross queries/data implemented through objects

Why NoSQL? Performance?



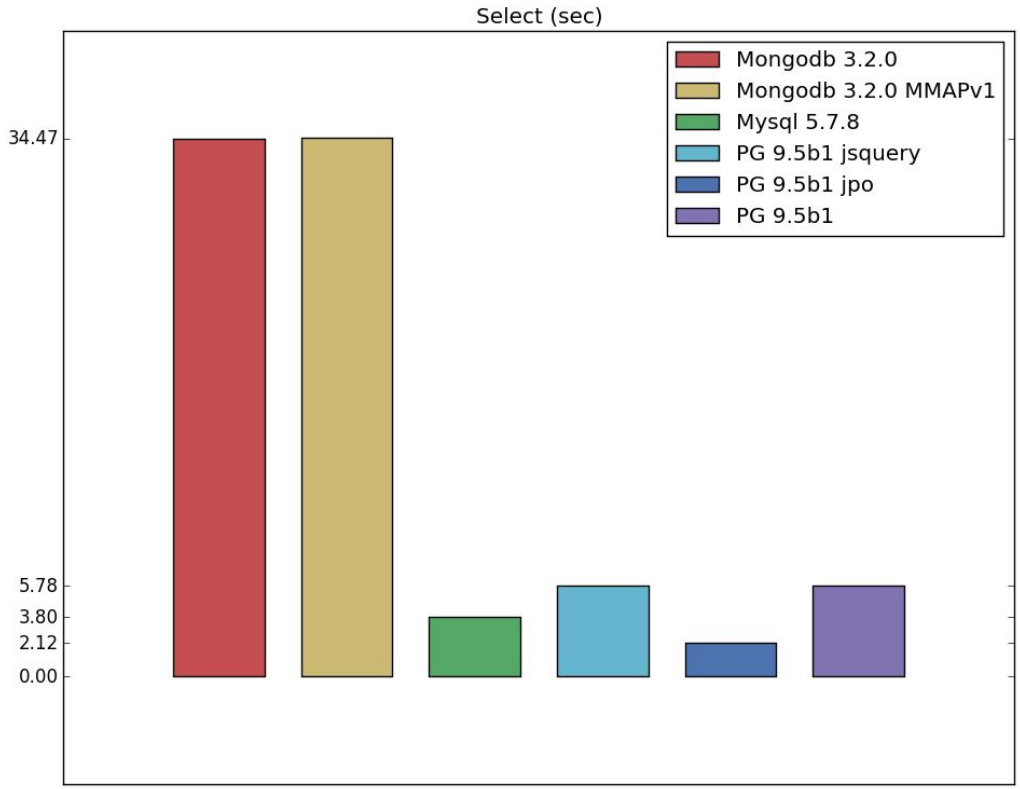


Image credit: http://maurizioturatti.com/blog/2015/01/06/using-nosql-wrong-reason/
Image credit: http://erthalion.info/2015/12/29/json-benchmarks/


BUT YOU SAID IT
WOULD GET BETTER!

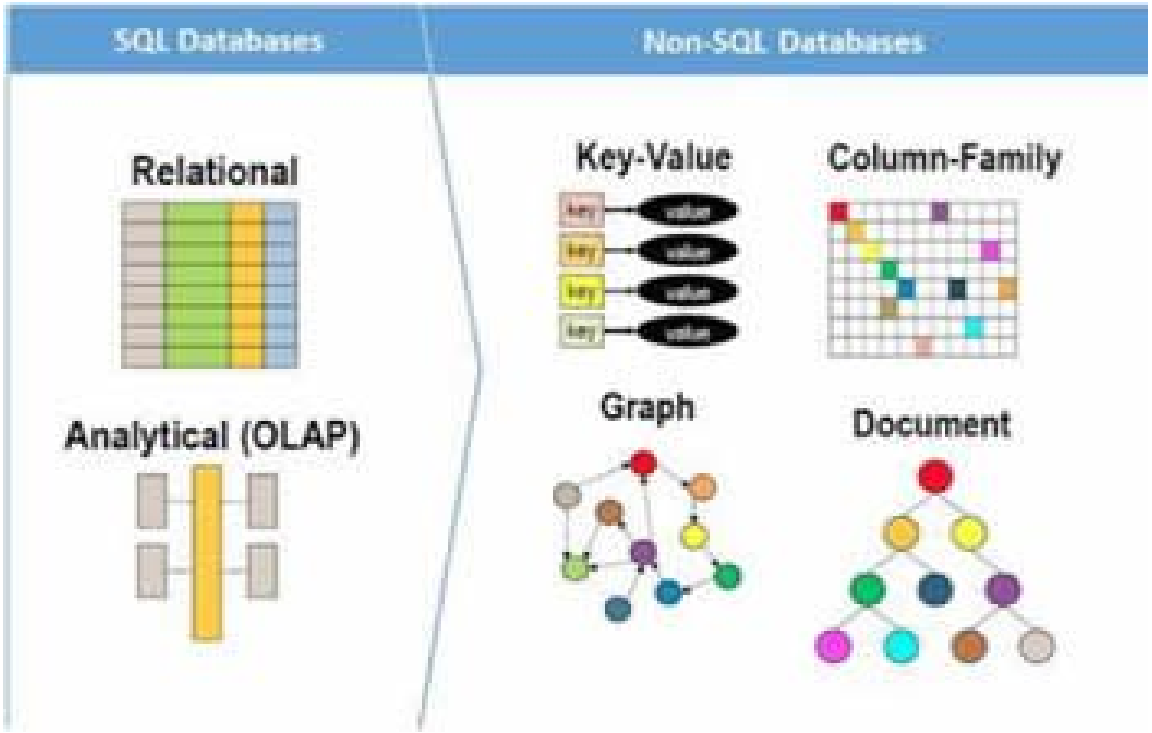
Comparing SQL and NoSQL

	SQL	*NoSQL*
Model Type	Relational	Non -Relational
Data	Small – Medium data sets	Large data sets
Schema	Static (Schema based)	Dynamic(Schema less)
Scalability	Vertical	Horizontal
Language	SQL to query data	NoSQL – JSON to query
Joins	Used for complex queries	No joins
Support	Great Support	Community Support
Flexibility	Rigid schema	Flexible
Auto Elasticity	Requires downtime	Automatic, No outage
Transaction	ACID	CAP Theorem
Current State	Mature	Emerging
Data structure	Structured e.g. Tables	Semi-structured –JSON
Examples	Oracle, Microsoft SQL Server,MySQL	MongoDB Cassandra,HBase,CouchDB

Image Credit: https://www.udemy.com/nosql-databases-for-beginners/

NoSQL - The Landscape

- Document DBs
- Key-Value
- Graph
- Big Table/Tabular
- Object



http://www.differencebetween.info/difference-between-nosql-and-sql-database

NoSQL - The Landscape

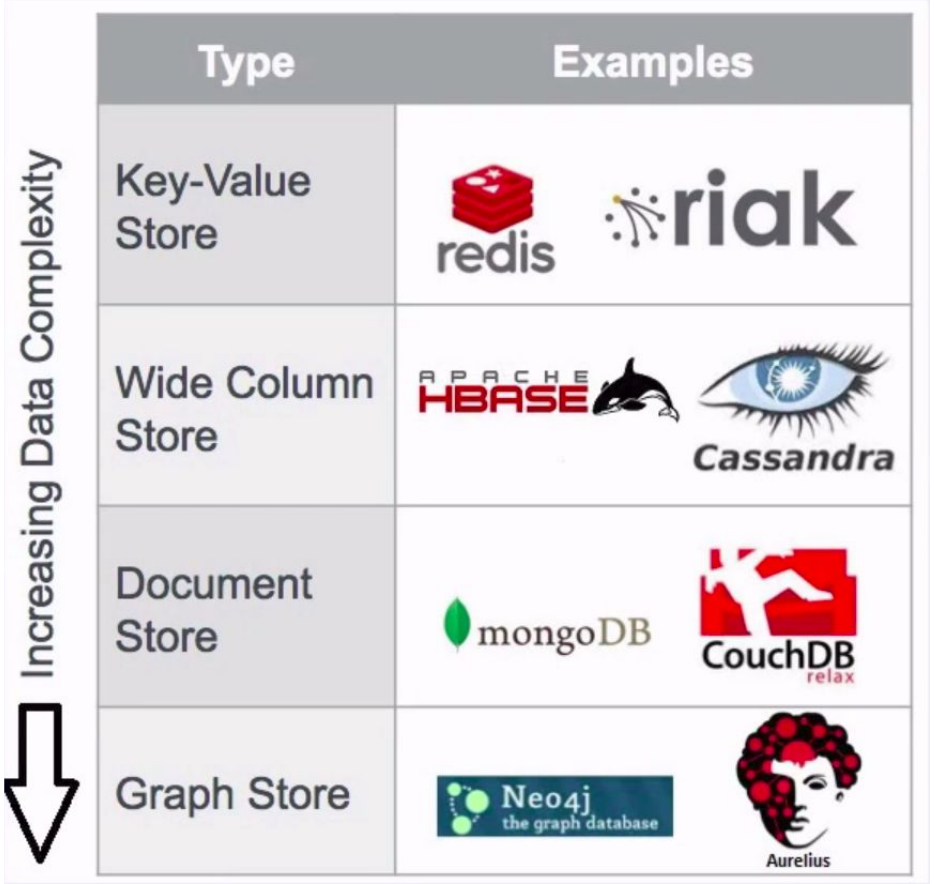


Image credit: https://www.udemy.com/nosql-databases-for-beginners/

NoSQL - MongoDB

- Document DBs
- MongoDB
  - high performance
  - easily scalable

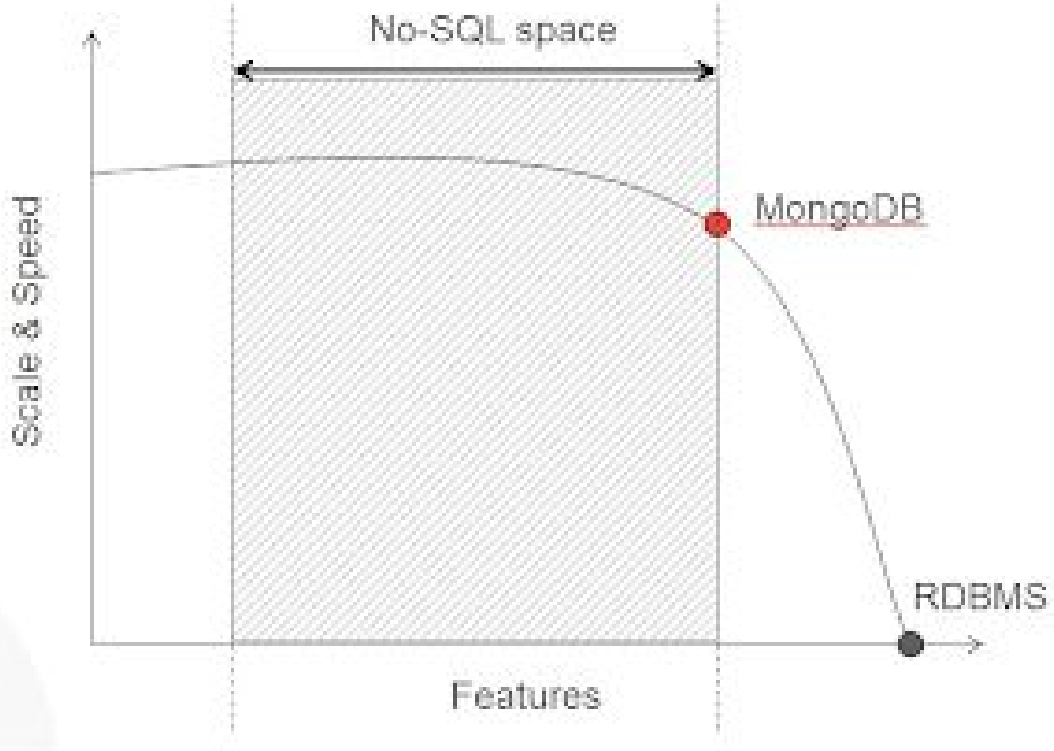


Image credit: http://sql-vs-nosql.blogspot.com/2013/10/the-base-difference-between-sql-and.html

MongoDB - the basics

- Documents stored as documents (JSON-like)
  - BSON (Binary representation) of JSON
- Follow similar structures as in programming languages

Example JSON Document

XML

```
<Node>
  <id>10002</id>
  <Name>john</Name>
</Node>
<Node>
  <id>10003</id>
  <Name>Scott</Name>
</Node>
<Node>
  <id>10004</id>
  <Name>Mohan</Name>
</Node>
<Node>
  <id>10001</id>
  <Name>Deepak </Name>
</Node>
```

JSON

```
[
    {
        "id":10002,
        "name":"john"
    },
    {
        "id":10003,
        "name":"Scott"
    },
    {
        "id":10004,
        "name":"Mohan"
    },
    {
        "id":10001,
        "name":"Deepak"
    }
]
```

Image credit: http://sqllearnergroups.blogspot.com/2014/03/how-to-get-json-format-through-sql.html

MongoDB - A Collection

- A collection is a group of MongoDB documents
- Similar to a table in MySQL grouping of MongoDB documents.

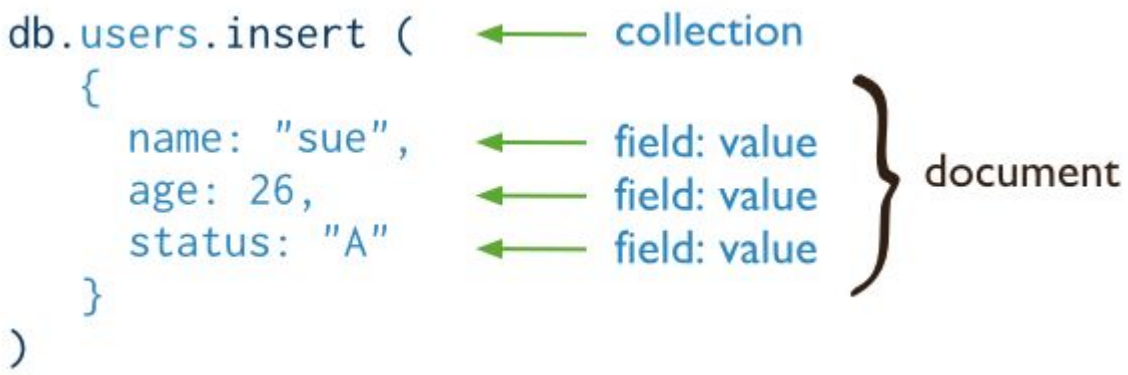


Image credit: https://docs.mongodb.com

MongoDB - A Document

- A document is a ‘record’ in a MongoDB collection
- There can be multiple documents in a collection
- And each document can contain different fields

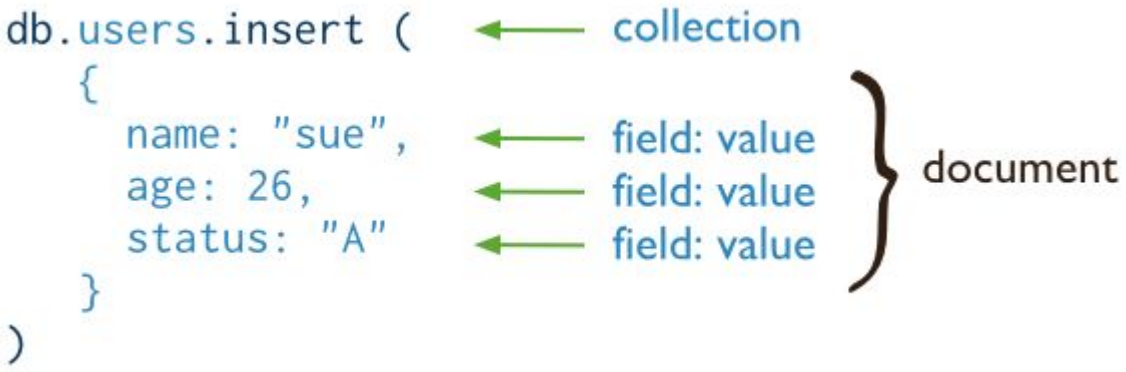


Image credit: https://docs.mongodb.com

MongoDB - A Field

- A field is a name-value pair in a document
- Fields are similar to MySQL columns

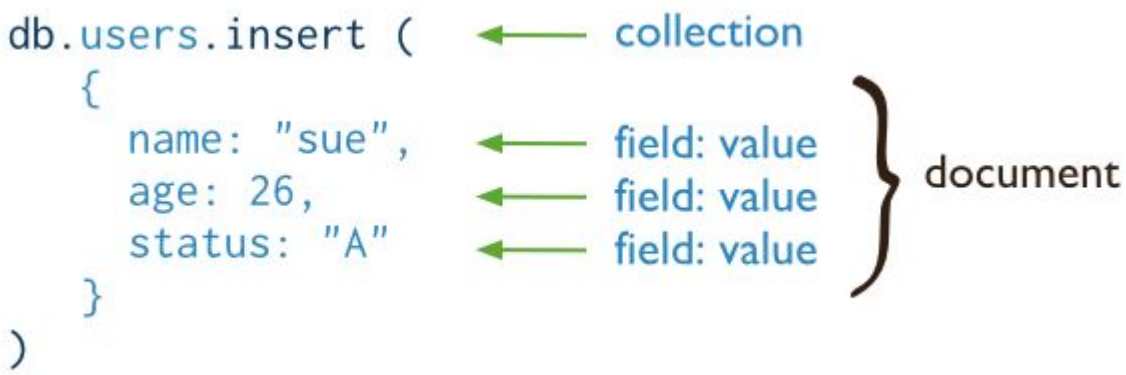


Image credit: https://docs.mongodb.com

MongoDB - Types

- Many of the types are similar to MySQL
- However, there is support for more advanced types (i.e. Javascript)
- Every type has a number that can be referenced:

```
{ field: { $type: <BSON type> } }
```

Type	Number	Notes
Double	1	
String	2	
Object	3	
Array	4	
Binary data	5	
Undefined	6	Deprecated.
Object id	7	
Boolean	8	
Date	9	
Null	10	
Regular Expression	11	
JavaScript	13	
Symbol	14	Deprecated.
JavaScript (with scope)	15	
32-bit integer	16	
Timestamp	17	
64-bit integer	18	
Min key	255	Query with -1.
Max key	127	

https://docs.mongodb.com

MongoDB - A Document

- And each document can contain different fields
- Including embedded sub-documents

Image credit: https://docs.mongodb.com

MongoDB - the basics

- Selecting a database to use:

```
use helloMongoDB
```

- Inserting a document into the database:

```
db.helloMyCollection.insert( { name: “Jenn” } )
```

- Inserting multiple documents into the database:

```
db.helloMyCollection.insert( [{ name: “Emmie” }
,{name:  “Alex”} ])
```

MongoDB - the basics

- Removing a document from the collection:

```
db.helloMyCollection.remove(name:  “Jenn”)
```

- Remove all documents from collection:

```
db.helloMyCollection.remove({})
```

- Drop collection:

```
db.helloMyCollection.drop()
```

https://docs.mongodb.com

MongoDB - the basics

- Querying:

```
db.helloMyCollection.find()
```

- Querying with criteria:

```
db.helloMyCollection.find(“name”:”jenn”)
```

MongoDB - the basics

- Querying with ‘where’ clauses

Operation	Syntax	RDBMS Equivalent
Equality	{<key>:<value>}	where field = value
Less Than	{<key>:{$lt:<value>}}	where field < value
Less Than Equals	{<key>:{$lte:<value>}}	where field <= value
Greater Than	{<key>:{$gt:<value>}}	where field > value
Greater Than Equals	{<key>:{$gte:<value>}}	where field >= value
Not Equals	{<key>:{$ne:<value>}}	where field != value

Table credit: https://www.tutorialspoint.com/mongodb/mongodb_query_document.htm

Comparing MySQL and MongoDB

RDBMS	MongoDB
Database	Database
Table	Collection
Tuple/Row	Document
column	Field
Table Join	Embedded Documents
Primary Key	Primary Key (Default key _id provided by mongo db itself)

Image Credit: https://www.slideshare.net/EnochJoshua1/mongodb-for-beginners

Comparing MySQL and MongoDB

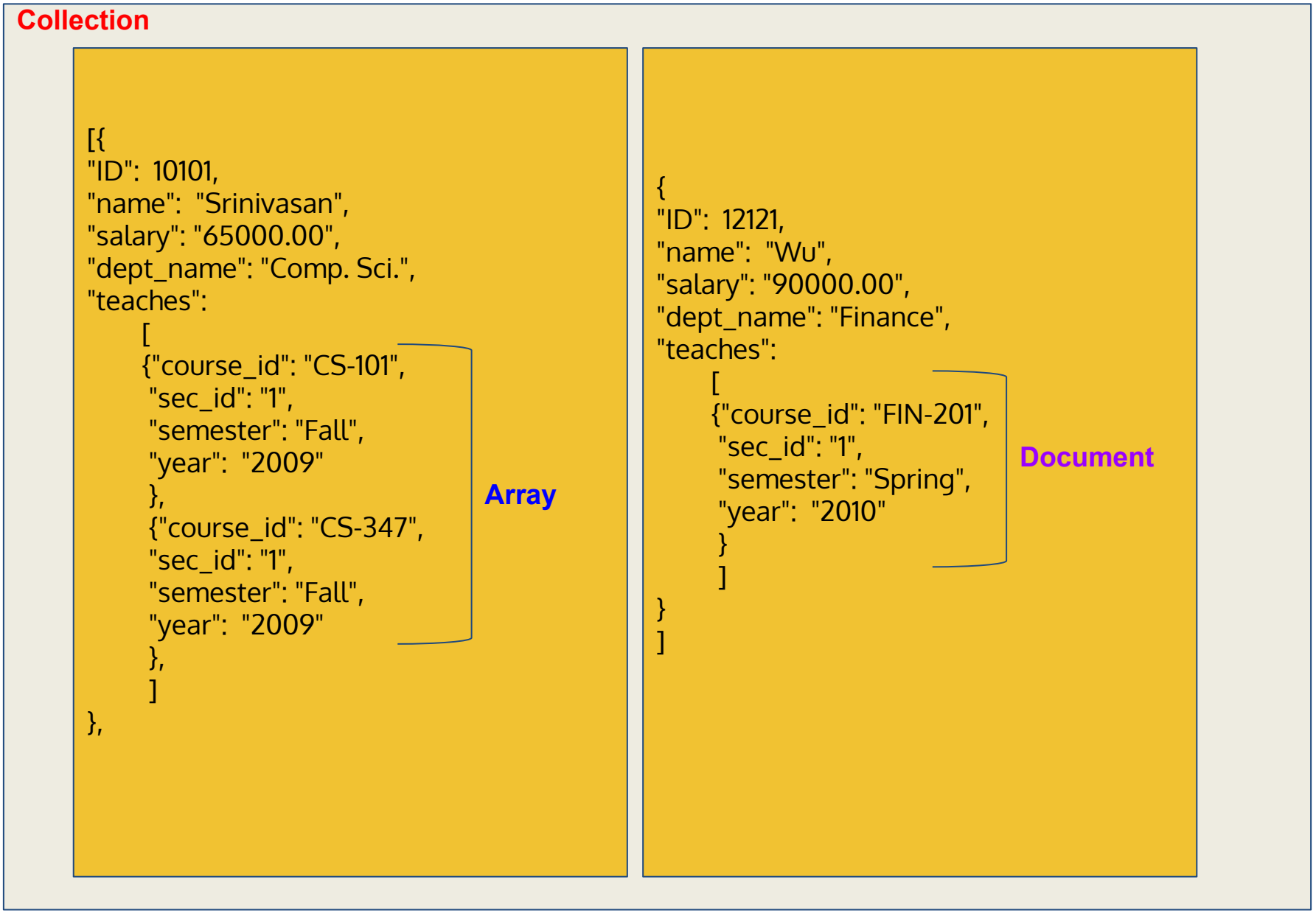
```
+-------+------------+------------+----------+
| ID    | name       | dept_name  | salary   |
+-------+------------+------------+----------+
| 10101 | Srinivasan | Comp. Sci. | 65000.00 |
| 12121 | Wu         | Finance    | 90000.00 |
| 15151 | Mozart     | Music      | 40000.00 |
| 32343 | El Said    | History    | 60000.00 |
| 45565 | Katz       | Comp. Sci. | 75000.00 |
| 58583 | Califieri  | History    | 62000.00 |
| 76543 | Singh      | Finance    | 80000.00 |
| 83821 | Brandt     | Comp. Sci. | 92000.00 |
| 98345 | Kim        | Elec. Eng. | 80000.00 |
+-------+------------+------------+----------+
```

```
+-------+-----------+--------+----------+------+
| ID    | course_id | sec_id | semester | year |
+-------+-----------+--------+----------+------+
| 10101 | CS-101    | 1      | Fall     | 2009 |
| 45565 | CS-101    | 1      | Spring   | 2010 |
| 83821 | CS-190    | 1      | Spring   | 2009 |
| 83821 | CS-190    | 2      | Spring   | 2009 |
| 10101 | CS-315    | 1      | Spring   | 2010 |
| 45565 | CS-319    | 1      | Spring   | 2010 |
| 83821 | CS-319    | 2      | Spring   | 2010 |
| 10101 | CS-347    | 1      | Fall     | 2009 |
| 98345 | EE-181    | 1      | Spring   | 2009 |
| 12121 | FIN-201   | 1      | Spring   | 2010 |
| 32343 | HIS-351   | 1      | Spring   | 2010 |
| 15151 | MU-199    | 1      | Spring   | 2010 |
+-------+-----------+--------+----------+------+
```

```
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| ID        | varchar(5)  | NO   | PRI | NULL    |       |
| name      | varchar(20) | NO   |     | NULL    |       |
| dept_name | varchar(20) | YES  | MUL | NULL    |       |
| salary    | decimal(8,2)| YES  |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
```

```
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| ID        | varchar(5)  | NO   | PRI | NULL    |       |
| course_id | varchar(8)  | NO   | PRI | NULL    |       |
| sec_id    | varchar(8)  | NO   | PRI | NULL    |       |
| semester  | varchar(6)  | NO   | PRI | NULL    |       |
| year      | decimal(4,0)| NO   | PRI | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
```

Comparing MySQL and MongoDB



Lecture Outline

- Quick Introduction to Views
- NoSQL