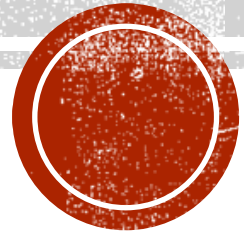
NOSQL UNIT-4

Dr. SELVA KUMAR S

B.M.S COLLEGE OF ENGINEERING



AGENDA

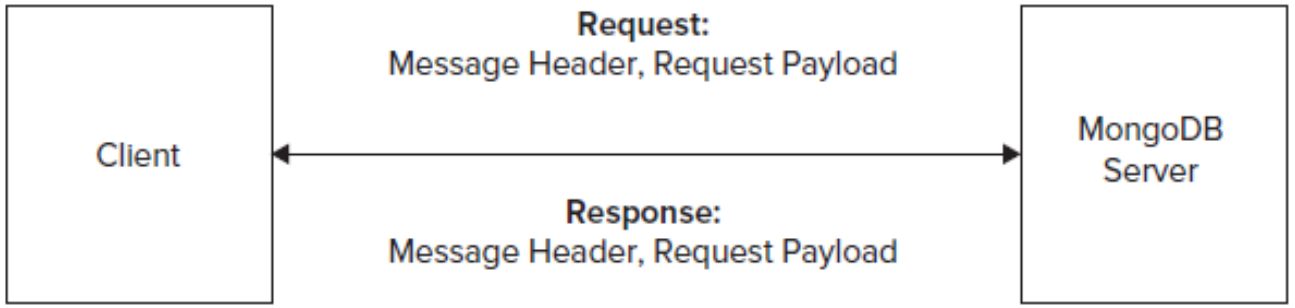
- Working with NOSQL:
  - Surveying Database Internals
- Using MySQL as NoSQL
- Web Frameworks and NoSQL
- Migrating from RDBMS to NoSQL

SURVEYING DATABASE INTERNALS

- MongoDB
- Membase
- Hypertable
- Apache Cassandra
- Berkley DB

MONGODB WIRE PROTOCOL

- Clients speak to a MongoDB server using a simple TCP/IP-based socket connection.
- The wire protocol used for the communication is a simple request-response-based socket protocol.
- The wire protocol headers and payload are BSON encoded.



MONGODB WIRE PROTOCOL OPERATIONS

- **OP_INSERT (code: 2002) —** Insert a document.
- **OP_UPDATE (code: 2001) —** Update a document.
- **OP_QUERY (code: 2004) —** Query a collection of documents.
- **OP_GET_MORE (code: 2005)** — Get more data from a query.
- **OP_REPLY (code: 1)** — Reply to a client request.
- **OP_KILL_CURSORS (code: 2007)** — Operation to close a cursor.
- **OP_DELETE (code: 2006)** — Delete a document.
- **OP_MSG (code: 1000)** — Generic message command.

HEADER FORMAT

- Every request and response message has a header. A standard message header has the following properties:
- **messageLength** — The length of the message in bytes.
- **requestID —** A unique message identifi er.
- **responseTo —** In the case of OP_QUERY and OP_GET_MORE the response from the database
- **opCode —** The operation code.

INSERTING A DOCUMENT

- When creating and inserting a new document, a client sends an OP_INSERT operation via a request that includes:
- **A message header —** A standard message header structure that includes messageLength**,** requestID**,** responseTo**,** and opCode**.**
- **An int32 value —** Zero (which is simply reserved for future use).
- **A cstring —** The fully qualifi ed collection name.
- **An array —** This array contains one or more documents that need to be inserted into a collection.

QUERYING A COLLECTION

- When querying for documents in a collection, a client sends an OP_QUERY operation via a request.
- **A message header —** A standard header with messageLength**,** requestID**,** responseTo**,** and opCode elements in it.
- **An int32 value —** Contains flags that represent query options.
- **A cstring** — Fully qualified collection name.
- **An int32 value** — Number of documents to skip.
- **Another int32 value —** Number of documents to return.
- **A query document in BSON format**
- **A document —** Representing the fi elds to return.

RESPONSE MESSAGE

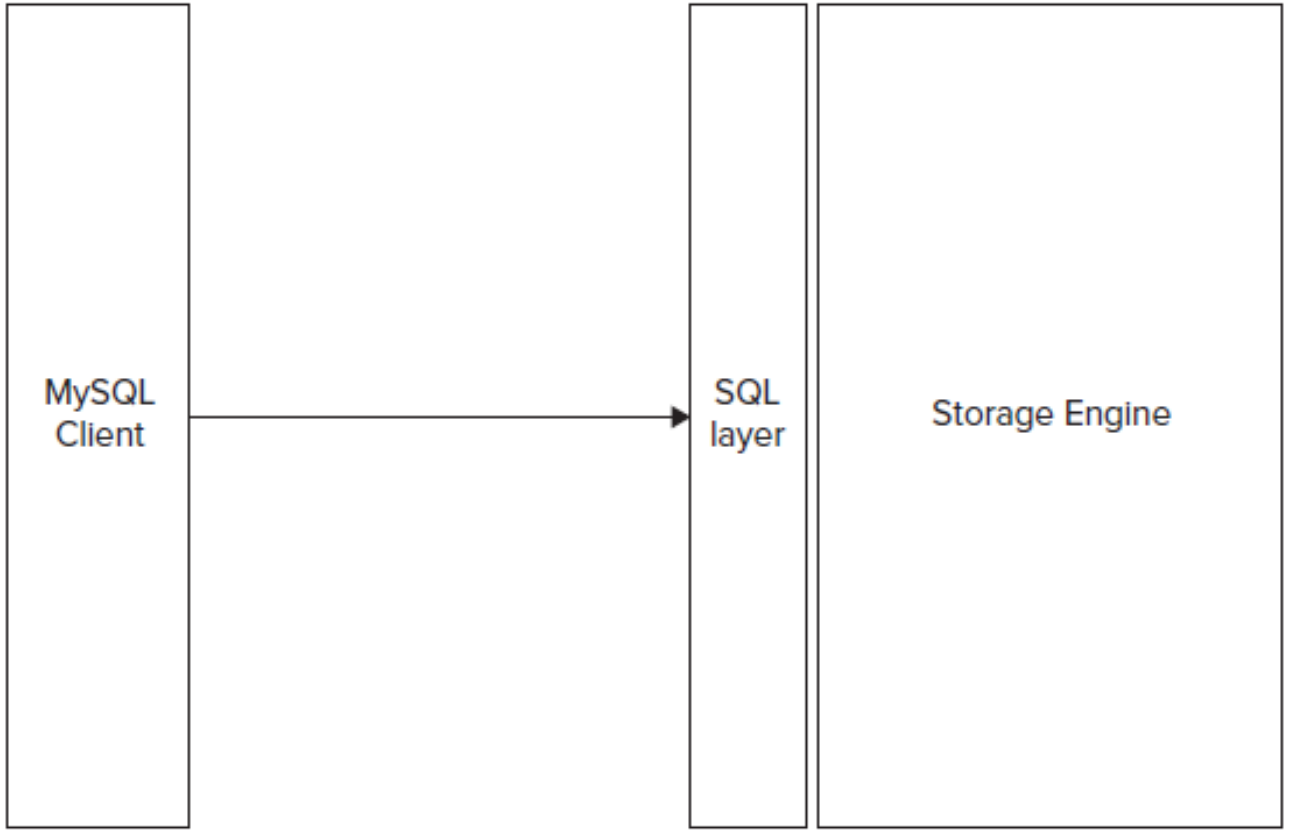
- An OP_REPLY message from the server includes:
- **A message header** — The message header in a client request and a server response is quite similar.
- **An int32 value —** Contains response flags that typically denote an error or exception situation.
- **An int64 value —** Contains the cursor id that allows a client to fetch more documents.
- **An int32 value —** Starting point in the cursor.
- **Another int32 value —** Number of documents returned.
- **An array —** Contains the documents returned in response to the query.

MONGODB DATABASE FILES

- MongoDB stores database and collection data in fi les that reside at a path specified by the –dbpath option to the mongod server program.
- The default value for dbpath is /data/db.
- Query for a collection storage properties
- db.movies.dataSize();
- db.movies.storageSize();
- db.movies.totalSize();
- db.movies.totalIndexSize();

```
> db.movies.validate();
{
"ns" : "mydb.movies",
"result" : "\nvalidate\n firstExtent:0:51a800 ns:mydb.movies\n
lastExtent:0:558b00 ns:mydb.movies\n # extents:4\n
datasize?:327280 nrecords?:3883 lastExtentSize:376832\n
padding:1\n first extent:\n loc:0:51a800 xnext:0:53bf00
xprev:null\n nsdiag:mydb.movies\n size:5888
firstRecord:0:51a8b0 lastRecord:0:51be90\n 3883 objects found,
nobj:3883\n 389408 bytes data w/headers\n 327280 bytes
data wout/headers\n deletedList: 1100000000001000000\n
deleted: n: 3 size: 110368\n nIndexes:2\n
mydb.movies.$_id_ keys:3883\n mydb.movies.$title_1 keys:3883\n",
"ok" : 1,
"valid" : true,
"lastExtentSize" : 376832
}
```

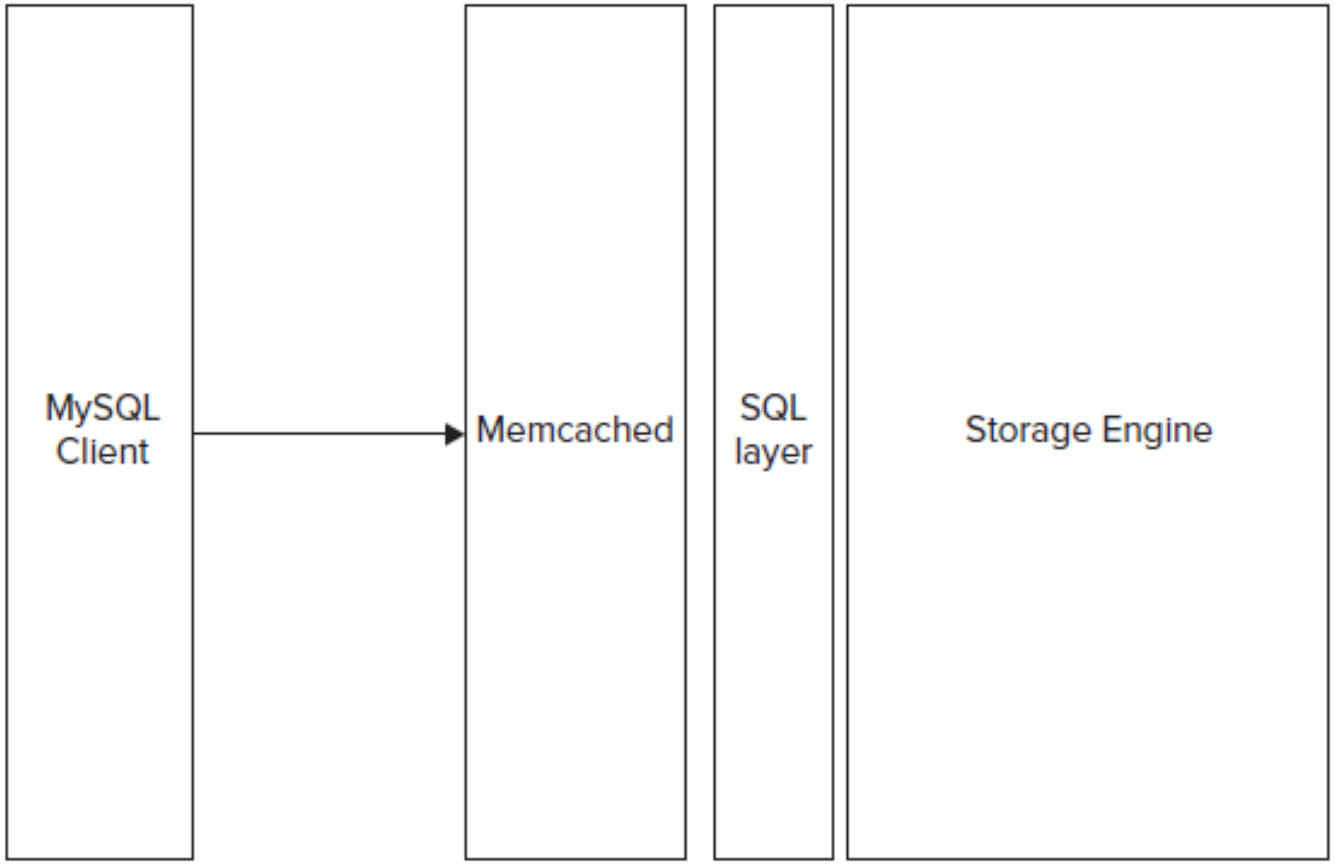
USING MYSQL AS A NOSQL SOLUTION



SQL OVERHEAD WITH CONCURRENCY

- Parsing SQL statements
- Opening tables
- Locking tables
- Making SQL execution plans
- Unlocking tables
- Closing tables
- Managing concurrent access using mutexes and threads

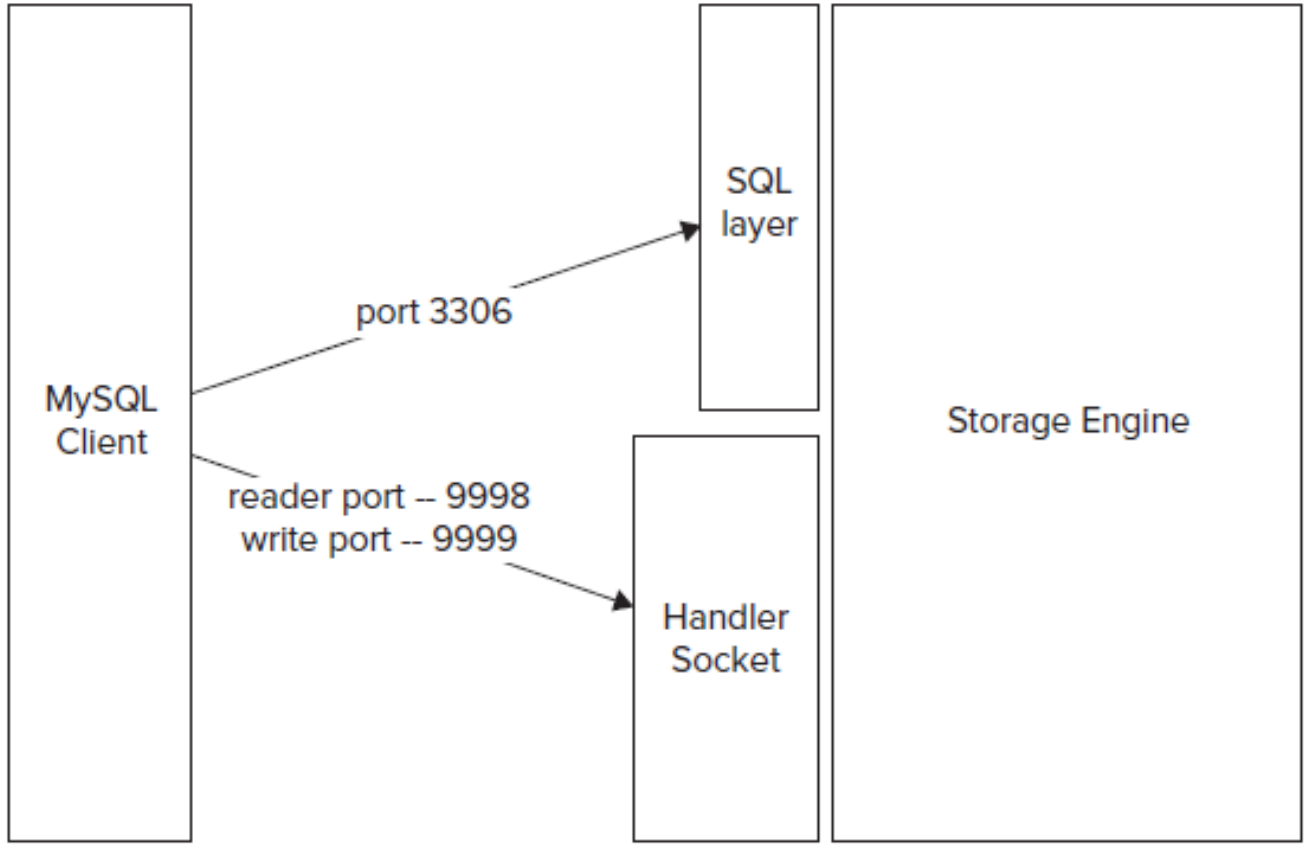
MEMCAHED IN MYSQL



DRAWBACK OF MEMCAHED IN MYSQL

- Data is in-memory in two places: the storage engine buffer and Memcached.
- Replication of data between the storage engine and Memcached can have inconsistent states of data.
- The data is fetched into Memcached via the SQL layer and so the SQL overhead is still present, even if it’s minimized.
- Memcached performance is superior only until all relevant data fits in memory. Disk I/O overheads can be high and can make the system slow.

HANDLER SOCKET PLUGIN



WEB FRAMEWORKS AND NOSQL

- Using Rails with NoSQL
- Using Django with NoSQL
- Using Spring Data

USING RAILS WITH NOSQL

- gem install rails
- rails new sample_app –skip-active-record
- gem install mongo_mapper
- require ‘rubygems’
- require ‘mongo’
- source ‘http://gemcutter.org’
- gem “rails”, “3.0.0”
- gem “mongo_mapper”

CONFIG/INITIALIZERS

```
MongoMapper.connection = Mongo::Connection.new('localhost', 27017)
MongoMapper.database = "#sample_app-#{Rails.env}"

if defined?(PhusionPassenger)
   PhusionPassenger.on_event(:starting_worker_process) do |forked|
     MongoMapper.connection.connect if forked
   end
end
```

SIMPLE MODEL

```
class UserData
    include MongoMapper::Document

    key :user_id, Integer
    key :user_name, String
end
```

CONTROLLER

```
class MyActionController < ApplicationController
  def create_user

    @auser = UserData.create(
             {
               :user_id => 1,
               :user_name => "John Doe",
               :updated_at => Time.now
             })
    @auser.save()
  end
end
```

- get ‘my_action/create_user’

DJANGO WITH NOSQL

- Django is to the Python community what Rails is to Ruby developers.
- Django is a lightweight web framework that allows for rapid prototyping and fast development.

```
# models.py:

class MyModel(models.Model):
    name = models.CharField(max_length=64)
    last_modified = models.DateTimeField(auto_now=True)

def run_query(name, month):
    MyModel.objects.filter(name__iexact=name, last_modified__month=month)

# dbindexes.py:

from models import MyModel
from dbindexer.api import register_index
register_index(MyModel, {'name': 'iexact', 'last_modified': 'month'})
```

USING SPRING DATA

- Application.properties
- ```
  spring.data.mongodb.host=localhost
  spring.data.mongodb.port=27017
  spring.data.mongodb.database=springBootMongoDB
  ```
- Creating an entity class
- ```
  public class Book {

  @Id // making this variable as ID, will be auto-generated by MongoDB
  private String id;

  @NonNull
  private Integer bookId;
  @NonNull
  private String bookName;
  @NonNull
  private String bookAuthor;
  @NonNull
  private Double bookCost;
  }
  ```

CREATING RUNNER CLASS

```
@Override
public void run(String... args) throws Exception {

// Removing old data if exists
bookRepo.deleteAll();

// Saving 4 books into DB
bookRepo.saveAll(Arrays.asList(
new Book(501, "Core Java", "Kathy Sierra", 1065.5),
new Book(502, "Spring in Action", "Craig Walls", 940.75),
new Book(503, "Hibernate in Action", "Gavin King", 889.25),
new Book(504, "Practical MongoDB", "Shakuntala Gupta", 785.0)
));
System.out.println("All Data saved into MongoDB");

// Updating ID(PK) manually (allowed) : It will create one new record
bookRepo.save(new Book("ISBN10:1484240251", 504,"Practical MongoDB", "Navin Sabharwal", 785.0));
// insert

// Printing all books
List<Book> bookList = bookRepo.findAll();
bookList.forEach(System.out::println);

}
```

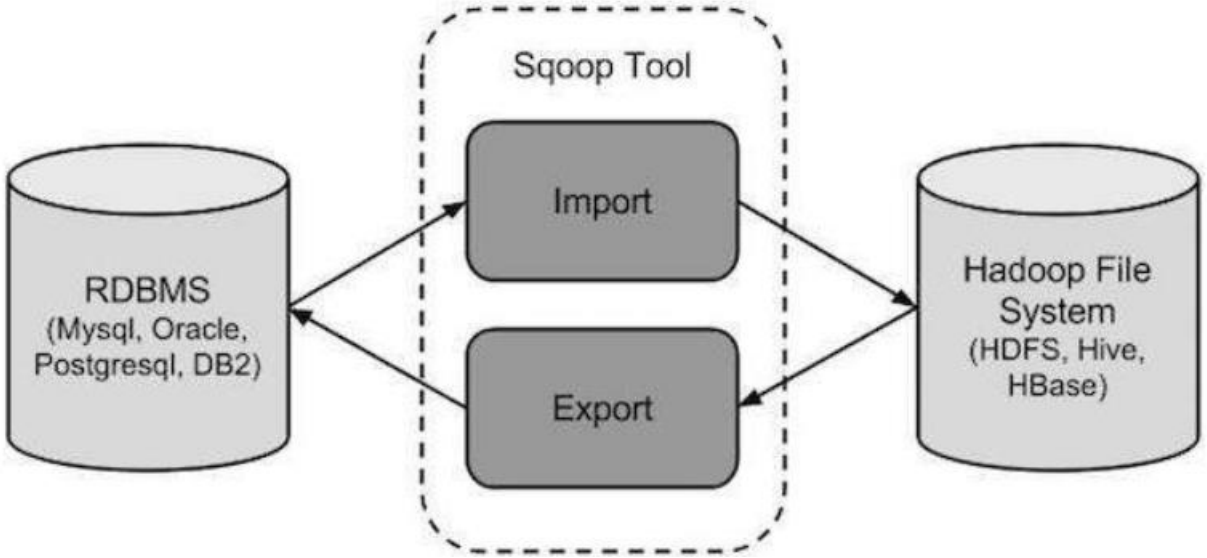
MIGRATING FROM RDBMS TO NOSQL

- Migrating from a structured schema to a schema-less form is not very hard.
- In many cases you could simply export the data from RDBMS tables and move them into NoSQL collections.
- However, things get complicated when the NoSQL database is a column-family, sorted ordered, or a key/value store.
- The greater impedance mismatch is around ad-hoc querying and secondary indexes, which are often difficult to support in a NoSQL environment.
- NoSQL looks at the data store from a query perspective and not from a generic storage viewpoint.

SQOOP

- To facilitate data importation from RDBMS to Hadoop for NoSQL-style manipulations.
- Sqoop is a command-line tool with the following capabilities:
  - Imports individual RDBMS tables or entire databases to fi les in HDFS
  - Generates Java classes to allow you to interact with your imported data
  - Provides the ability to import from SQL databases straight into your Hive data warehouse

HOW SQOOP WORKS?



THANK YOU