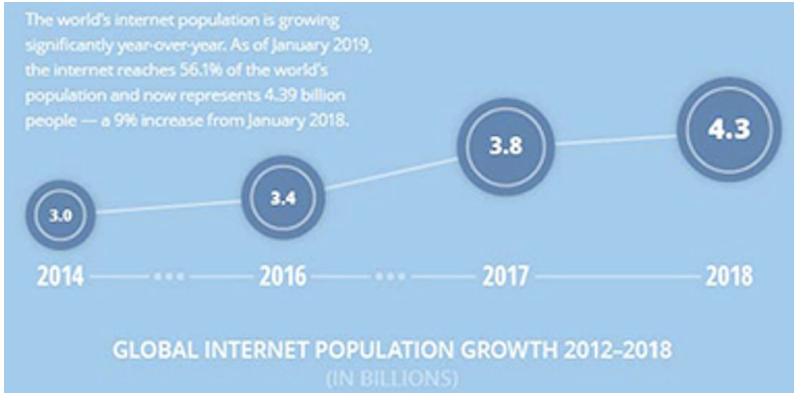
MONASH University

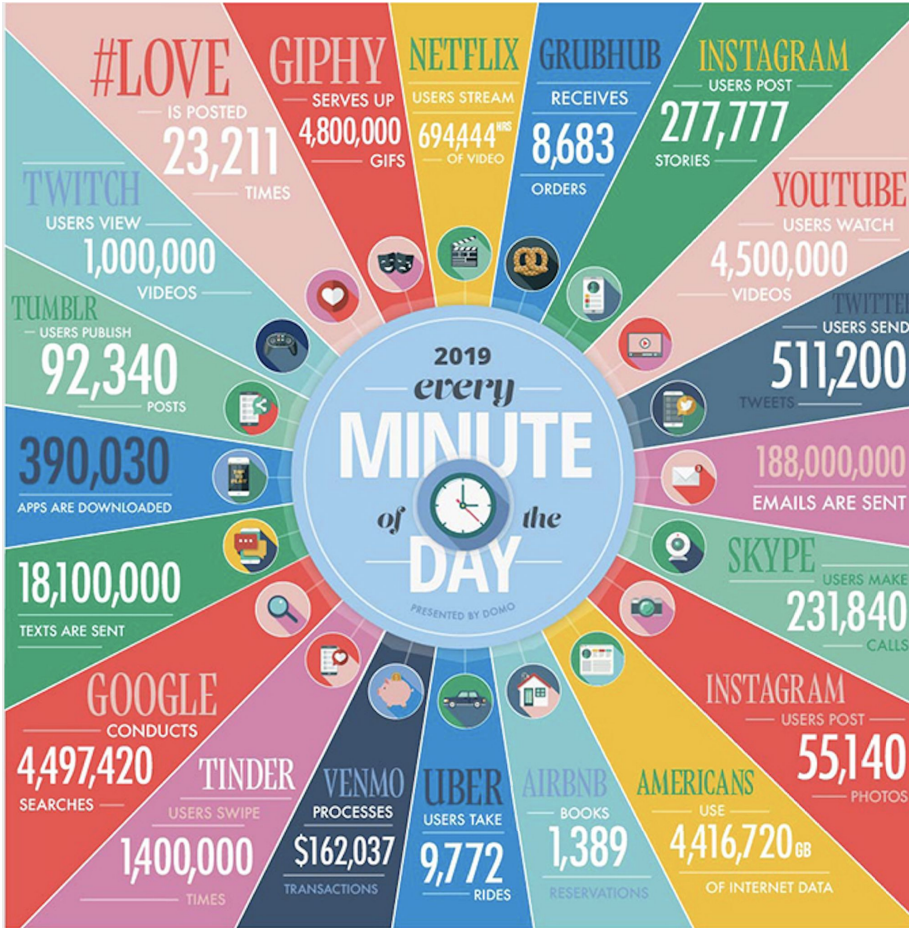
MONASH
INFORMATION
TECHNOLOGY

Non Relational Databases
Big Data

GROUP OF EIGHT AUSTRALIA

Data Growth

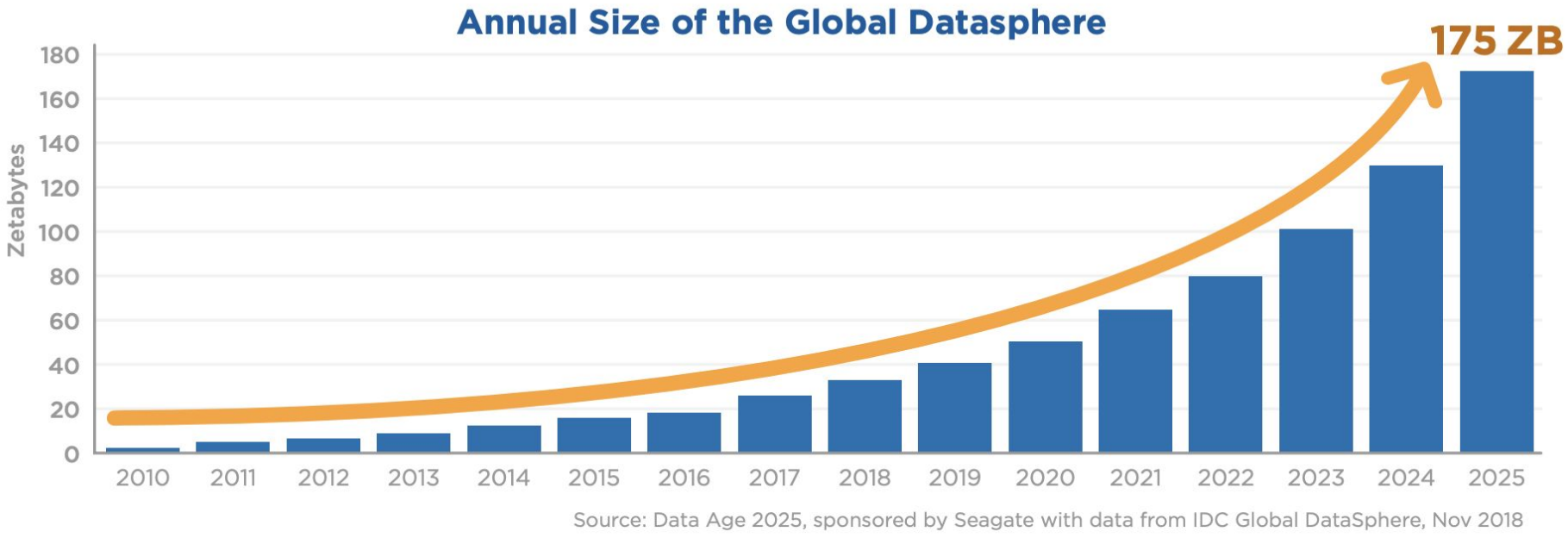




Source: https://www.domo.com/learn/data-never-sleeps-7#/

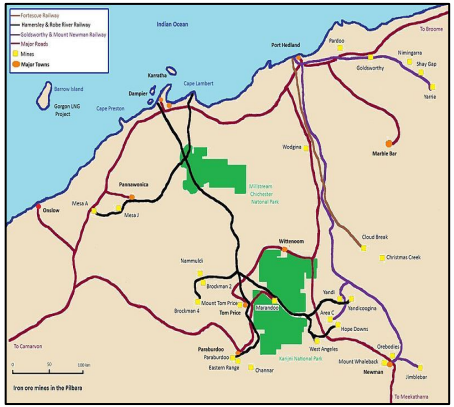
Data Growth

Figure 1 – Annual Size of the Global Datasphere



Source: https://www.seagate.com/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf

Railway In Mining



- Pilbara region, WA
- Trains perform round trips from the mining site to the port
- Loaded minerals and ores

- Length: > 2KM
- Load: > 10 Ton/car
- Speed: 5-10 Km/hr

- Instrumented Ore Car (IOC)
- Expensive Sensors
- Trained Professionals to maintain the sensors

Challenges

(1)
Expensive sensors that require professionals to maintain

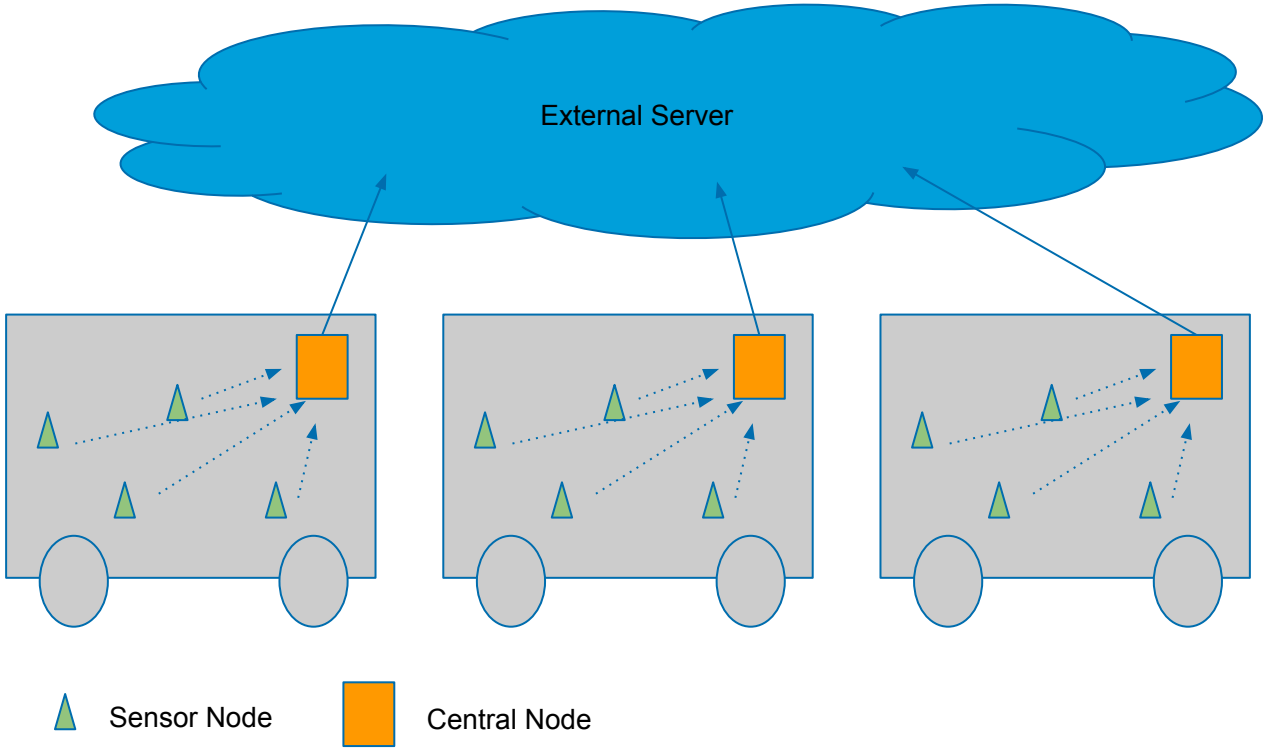
(2)
Large volume of data generated by the sensors

→ **Cheap, self-configured, massive array of sensors**

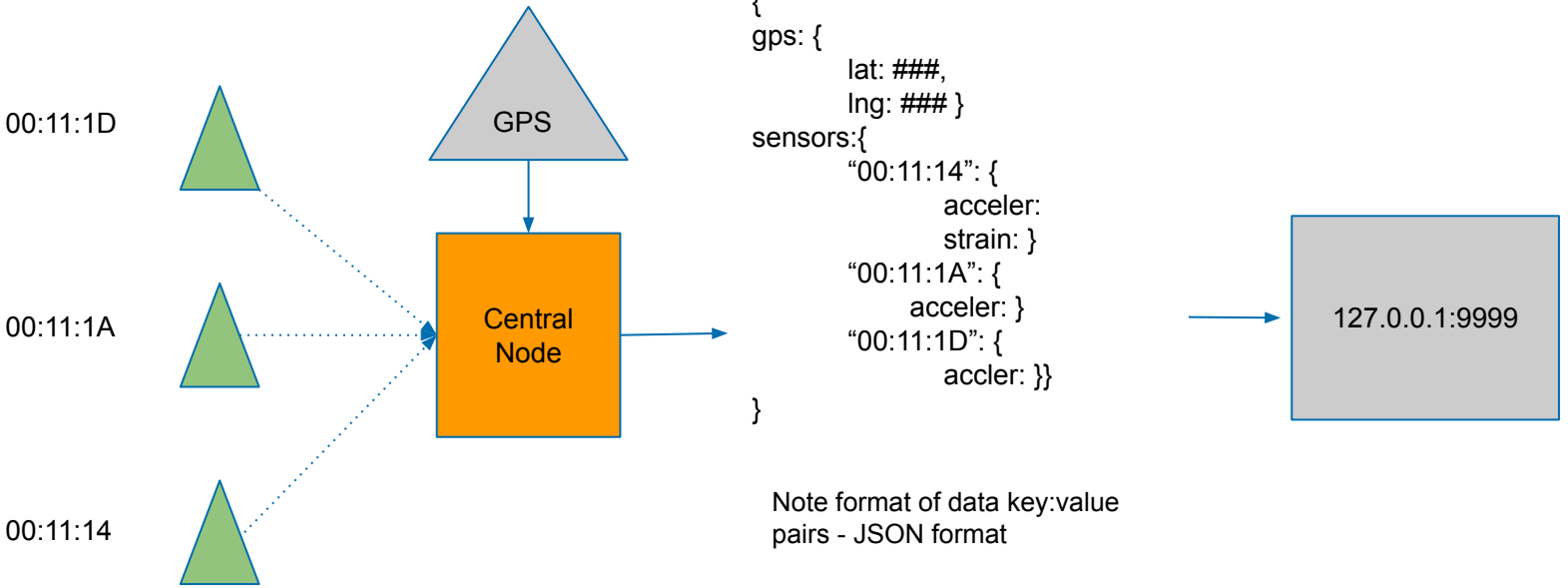
→ **Fast data processing and retrieval**

Needs expertise from Eng and IT

Network Structure



Central Node Process



Note format of data key:value pairs - JSON format

How Big is the Data?

Quantity	Data Returned
Timestamp	12-Jun-2015; 09:35:15
Geo-location	N35°43.57518,W078°49.78314
Direction	ToPort
Acceleration	0.285g
Pressure	65psi
Ambient temperature	73 degrees F
Surface temperature	78 degrees F
Humidity	35%

- 16 Sensors
- 200 Ore Cars
- 25 Records Per Second

16 * 200 * 25 = 80,000 records/sec

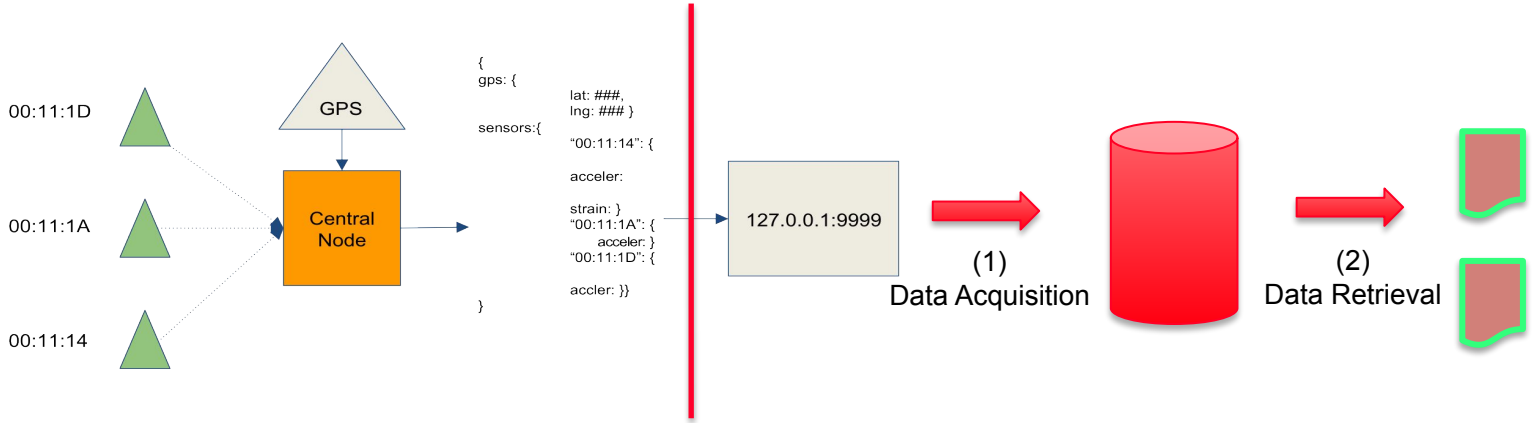
```
Welcome to Ubuntu 14.04.3 LTS (GNU/Linux 3.13.0-46-generic x86_64)

 * Documentation:  https://help.ubuntu.com/
ubuntu@master:~$ mongo
MongoDB shell version: 3.0.4
connecting to: test
2015-11-06T11:49:56.337+1100 I CONTROL  [initandlisten]
2015-11-06T11:49:56.337+1100 I CONTROL  [initandlisten] ** WARNING:
/sys/kernel/mm/transparent_hugepage/defrag is 'always'.
2015-11-06T11:49:56.337+1100 I CONTROL  [initandlisten] **        We suggest setting it to 'never'
2015-11-06T11:49:56.337+1100 I CONTROL  [initandlisten]
>  Use IRT
> db.sensordata.find().pretty()


{
                "_id" : ObjectId("5663ce2ce4b099b72ceca8c2"),
                 "gps": { "GPSLat" : -21.63893238,"GPSLon" : 116.70659242},
                "SomatTime" : 74711,
                "CarOrient" : 30.2,
                "EorL" : 1,
                "Direction" : "ToPort    ",
                "minSND" : 0,
                "iSegment" : 5876,
                "maxSND" : 0,
                "PipeA" : 0,
                "maxCFB" : 0,
                "minCFB" : 0,
                "Bounce" : 0,
                "minCFA" : 0,
                "maxCFA" : 0,
                "kmh" : 30.2,
                "PipeB" : 0,
                "Rock" : 0,
                "accR3" : 0,
                "accR4" : 0,
                "maxBounce" : 0,
                "LATACC" : 0
}


Type "it" for more
>
```

Big Data Processing



Two main problems:

(1) How to receive data … massive amount of data
(2) How to retrieve data … very fast

Big Data Characteristics

- Volume
  - The quantity of data to be stored
- Velocity
  - The speed at which data enters the system and must be processed
- Variety
  - Variations in the structure of the data to be stored

FIGURE 14.1 ORIGINAL VIEW OF BIG DATA

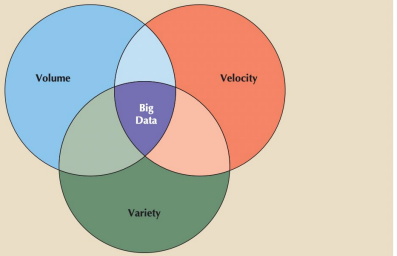
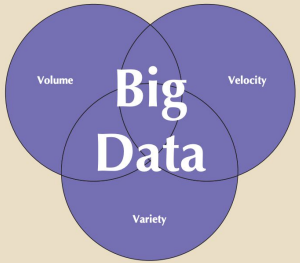


FIGURE 14.2 CURRENT VIEW OF BIG DATA



Big Data Characteristics: Volume

TABLE 14.1

STORAGE CAPACITY UNITS

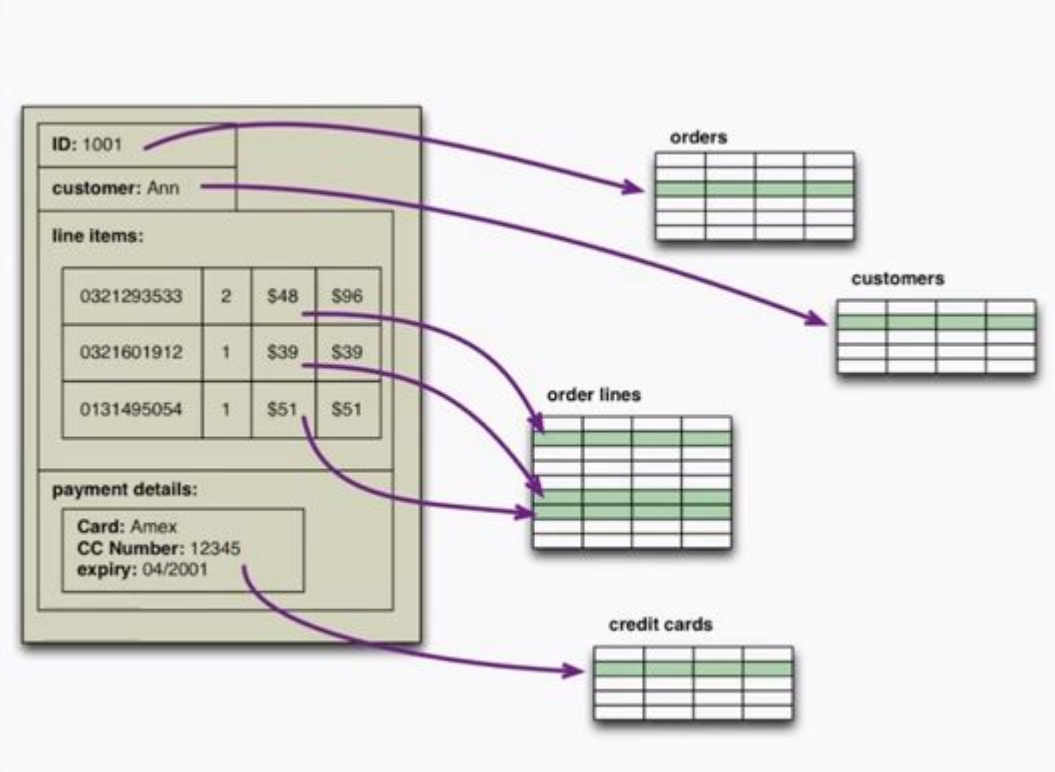
TERM	CAPACITY	ABBREVIATION
Bit	0 or 1 value	b
Byte	8 bits	B
Kilobyte	1024* bytes	KB
Megabyte	1024 KB	MB
Gigabyte	1024 MB	GB
Terabyte	1024 GB	TB
Petabyte	1024 TB	PB
Exabyte	1024 PB	EB
Zettabyte	1024 EB	ZB
Yottabyte	1024 ZB	YB

* Note that because bits are binary in nature and are the basis on which all other storage values are based, all values for data storage units are defined in terms of powers of 2. For example, the prefix *kilo* typically means 1000; however, in data storage, a kilobyte = 2^{10} = 1024 bytes.

- Scaling up: keeping the same number of systems but migrating each one to a larger system
- Scaling out: when the workload exceeds server capacity, it is spread out across a number of servers

Scaling

- How do we scale current relational systems? SQL designed for database as a single physical entity
  - Purchase bigger "boxes": costly and has real limits
  - Increase the number of processors, yielding parallel computation/database with complex issues to handle
  - Distribute database – challenges to maintain ACID transaction principles and issues of availability/consistency
- The rise of OO programming in the 80's also highlighted a problem known as the "Impedance Mismatch"
  - The program treats items as objects, but they need to be mapped to relational tables ("de aggregating" the object)
  - Also issues about "private" vs "public" (relational about need, OO absolute characteristic of data)



https://www.youtube.com/watch?v=qI_g07C_Q5I

Scaling continued

- Big players, notably Google and Amazon chose a different path
  - Lots and lots of smaller boxes ("commodity" servers)
  - Non relational structure
  - Google: Bigtable
    - https://research.google/pubs/pub27898/
    - https://cloud.google.com/bigtable/docs/overview
    - Used for wide range of apps Gmail, Google Earth, YouTube
  - Amazon: Dyanmo
    - http://www.read.seas.harvard.edu/~kohler/class/cs239-w08/decandia07dynamo.pdf
    - Based on Dynamo: https://aws.amazon.com/dynamodb/

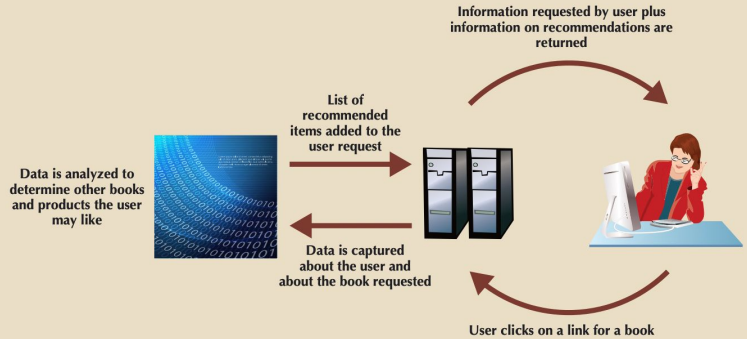
Scaling continued

- Term "NoSQL" coined by John Oskarsson in 2009 after calling a ..."free meetup about “open source, distributed, non relational databases” or NOSQL for short"…
  - http://blog.oskarsson.nu/post/22996139456/nosql-meetup
- Characteristics
  - Non relational,
  - mostly open source,
  - distributed (cluster friendly),
  - schema-less (no fixed storage schema)

Big Data Characteristics: Velocity

- Stream processing: focuses on input processing and requires analysis of data stream as it enters the system
  - CERN Large Hadron Collider 600TB per second ⇨ 1 GB per second
- Feedback loop processing: analysis of data to produce actionable results

FIGURE 14.3 FEEDBACK LOOP PROCESSING



Big Data Characteristics: Variety

- Structured data: fits into a predefined data model
  - Relational databases
  - Incoming data decomposed under normalisation rules to fit the data model
- Unstructured data: does not fit into a predefined model
  - Big Data requires that the data is captured in its natural format as generated without imposing a data model on it
- Semi structured data: combines elements of both

TABLE 14.2

ADDITIONAL Vs OF BIG DATA

CHARACTERISTIC	DESCRIPTION
Variability	Data meaning changes based on context.
Veracity	Data is correct.
Value (Viability)	Data can provide meaningful information.
Visualization	Data can be presented in such a way as to make it understandable.

Q1. Match names

- Volume
  - The quantity of data to be stored
- Velocity
  - The speed at which data enters the system and must be processed
- Variety
  - Variations in the structure of the data to be stored
- Variability
  - Data meaning changes depending on context
- Veracity
  - Correctness of the data
- Value
  - Data can provide meaningful information
- Visualisation
  - Data can be presented in a way which can be easily understood

Hadoop

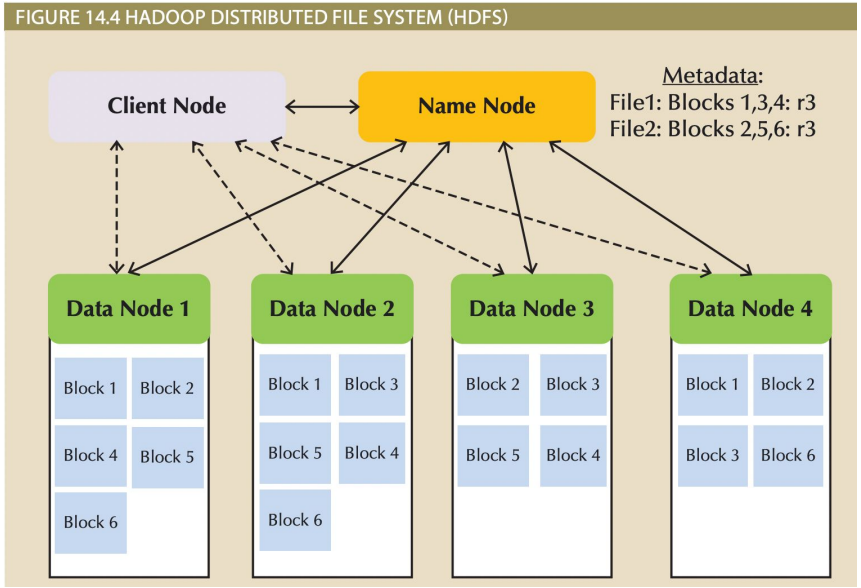
- Hadoop is not a database
  - De facto standard for most Big Data storage and processing
  - Java-based framework for distributing and processing very large data sets across clusters of computers
  - https://www.geeksforgeeks.org/hadoop-ecosystem/
- Important components
  - Hadoop Distributed File System (HDFS): low-level distributed file processing system that can be used directly for data storage
  - MapReduce: programming model that supports processing large data sets

Hadoop

- Hadoop Distributed File System (HDFS)
  - Based on several key assumptions
    - High volume: default block sizes is 64 MB (cf Oracle RDBMS 4K or 8K) and can be configured to even larger values
    - Write-once, read-many: model simplifies concurrency issues and improves data throughput
    - Streaming access: optimized for batch processing of entire files as a continuous stream of data
    - Fault tolerance: designed to replicate data across many different devices so that when one fails, data is still available from another device

Hadoop

- Hadoop uses several types of nodes; computers that perform one or more types of tasks within the system
  - Data node store the actual file data
  - Name node contains file system metadata
  - Client node makes requests to the file system as needed to support user applications
  - Data node communicates with name node and send back block reports and heartbeats

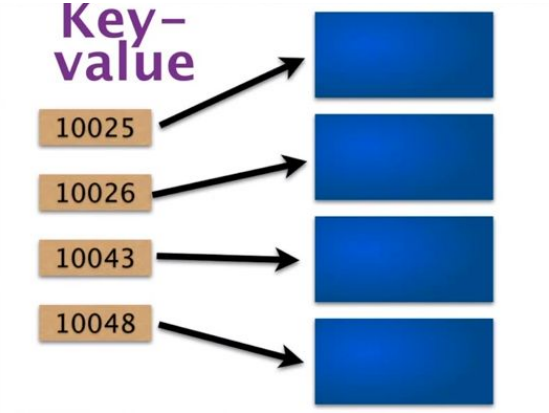


Hadoop

- MapReduce
  - Framework used to process large data sets across clusters
    - Breaks down complex tasks into smaller subtasks, performing the subtasks, and producing a final result
    - Map function takes a collection of data and sorts and filters it into a set of key-value pairs
      - Mapper program performs the map function
    - Reduce summaries results of map function produce a single result
      - Reducer program performs the reduce function

NoSQL Data Models

- Key-value store
  - Each item stored consists of a key and value pair (the value may be a numeric, a document, an image etc)



  - Oracle NoSQL database (community edition available)
    - https://www.oracle.com/au/database/technologies/related/nosql.html

NoSQL Data Models continued

- Document
  - Each item is stored as a document (normally BSON or JSON document, but could be XML)
  - Note the variable structure and embedded documents

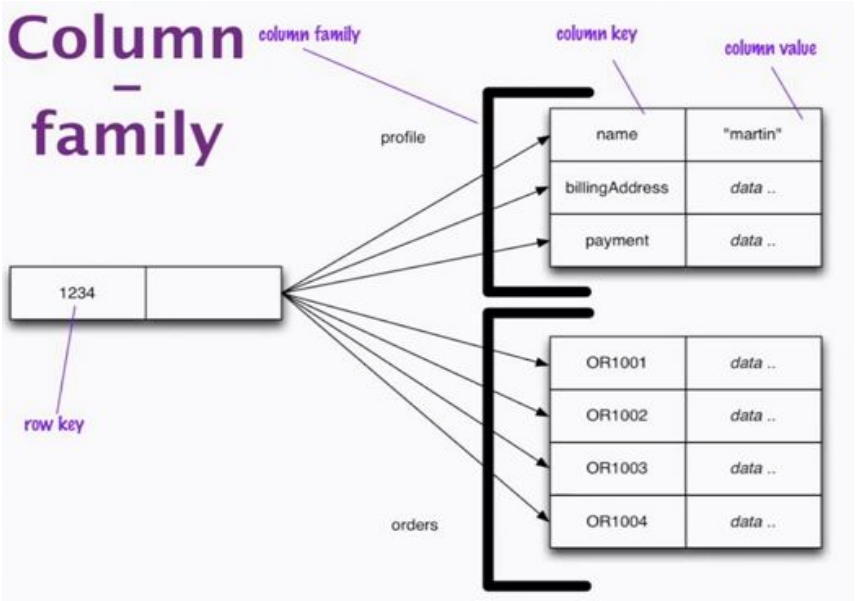
```
{  maker: "M.V. Agusta",
   type: "sportsbike",
   engine: {
      type: "internal combustion",
      cylinders: 4,
      displacement: 750
    },
   rake:7,
   trail:3.93
}
```

```
{   maker : "M.V. Agusta",
    type : "Helicopter",
    engine : {
        type : "turboshaft",
        layout : "axial",
        massflow : 1318
    },
    Blades : 4,
    undercarriage : "fixed"
}
```

MongoDB - https://www.mongodb.com/

NoSQL Data Models continued

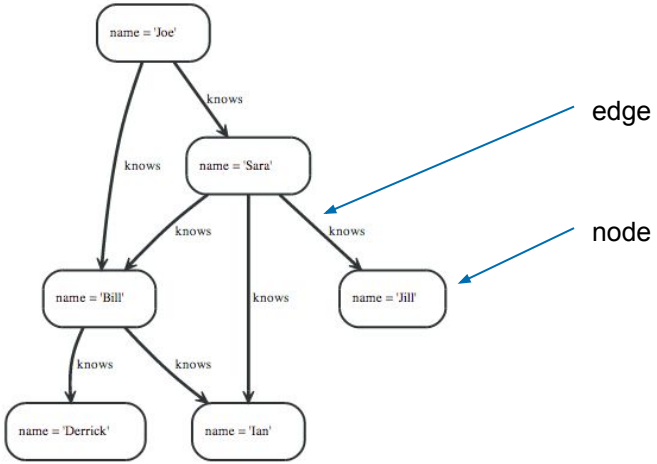
- Column Family (also called Wide Column Store)
  - Key points to a set of multiple column values containing related data arranged by column family



Cassandra (used on eBay): https://cassandra.apache.org/

NoSQL Data Models continued

- Graph - based on a graph structure
  - Unlike the previous three which are aggregation oriented, the graph model views data at a highly non aggregated level
  - Based on graph theory
  - Navigate via relationships (edges) between nodes
  - Examples
    - Neo4j
    - HyperGraphDB



To find out the friends of Joe's friends that are not already his friends, the query looks like this:

Query.

```
START joe=node:node_auto_index(name = "Joe")
MATCH joe-[:knows*2..2]-friend_of_friend
WHERE not(joe-[:knows]-friend_of_friend)
RETURN friend_of_friend.name, COUNT(*)
ORDER BY COUNT(*) DESC, friend_of_friend.name
```

https://neo4j.com/docs/stable/cypher-cookbook-friend-finding.html

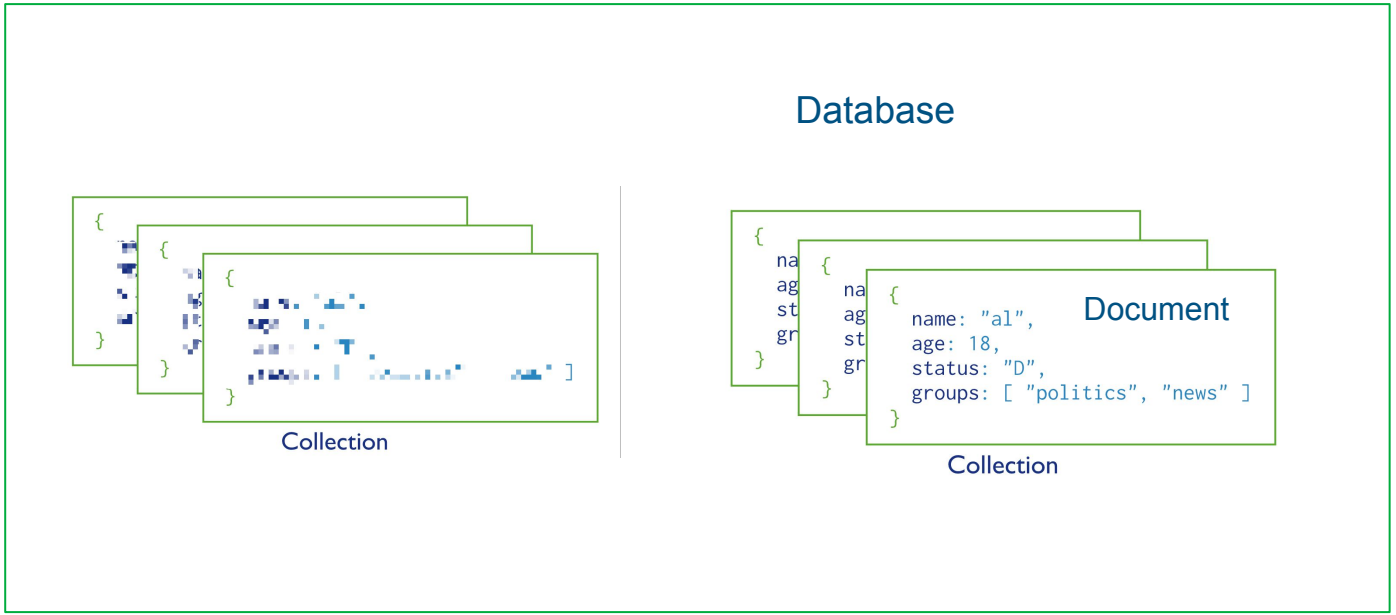
NoSQL Databases

- Comparison of NoSQL databases
  - https://hostingdata.co.uk/nosql-database/ currently lists 200+ NoSQL databases, including some outside these four models.
- Data is distributed on multiple machines via:
  - Sharding
    - One copy of the data spread across multiple machines, or
  - Replication
    - Same data is spread across multiple machines, increased availability and resilience
- Lots of interesting questions and research around consistency vs availability

MongoDB

- Document Database
  - Community edition available for download (not required for FIT9132)
    - https://www.mongodb.com/download-center/community
  - MongoDB Shell
    - https://docs.mongodb.com/manual/tutorial/getting-started/
  - Database
    - show dbs, use *dbname,* db.dropDatabase(), db (show current db)
    - Contains collections
      - show collections
        - » collection contains documents

MongoDB - Database



Documents -> Collections -> Database

Document structure

- MongoDB stores data records as BSON documents (binary JSON documents)

- Document composed of field-values pairs
  - Field names may be enclosed in quotes (allows spaces in name)
  - "groups" field above holds an array of strings

Relationships – structuring documents

```
{
  _id: <ObjectId1>,
  username: "123xyz",
  contact: {
             phone: "123-456-7890",
             email: "xyz@example.com"
           },
  access: {
            level: 5,
            group: "dev"
          }
}
```

Embedded sub-document

Embedded sub-document

user document

```
{
  _id: <ObjectId1>,
  username: "123xyz"
}
```

contact document

```
{
  _id: <ObjectId2>,
  user_id: <ObjectId1>,
  phone: "123-456-7890",
  email: "xyz@example.com"
}
```

access document

```
{
  _id: <ObjectId3>,
  user_id: <ObjectId1>,
  level: 5,
  group: "dev"
}
```

Denormalised – Embedded Documents

Normalised – References

Document sample - Famous Database Personalities

```
{
  "_id": 1,
  "name" : { "first" : "Edward", "last" : "Codd" },
  "contribs" : [ "Database Relational Model", "Database Technology", "OLAP"  ],
  "awards" : [
    {
      "award" : "Turing Award",
      "year" : 1981,
      "by" : "Association for Computing Machinery"
    }
  ]
}
```

- name - sub document
- contribs - array of string values
- awards - array of sub documents

Q2. We wish to create a JSON document to record member login attempts to our membership server.

For each member who attempts to login we wish to record the members login username and their first and last names

For each login attempt we wish to record the date/time of the attempt eg. 1st June 2020 8PM, the IP from which the attempt was made (IPv4 is sufficient for this task eg 132.12.34.123) and the success of the login (yes or no). The date should be expressed in the date format: YYYY-mm-ddTHH:MM:ss eg. 2020-06-01T20:00:00 for the date/time above. Assign each member a unique id starting at 1

Create a JSON document for this scenario containing 1 members attempts to login (you do not need to be concerned about the accuracy of the data, simply invent the values). The user should have 2 login attempts

To validate your JSON test your document at: https://codebeautify.org/jsonvalidator

```
{
    "_id": 1,
    "username": "ghopper",
    "name": {
        "firstname": "Grace",
        "lastname": "Hopper"
    },
    "loginattempts": [
        {
            "attemptat": "2020-06-01T20:00:00",
            "attemptip": "132.12.34.123",
            "attemptyorn": "N"
        },
        {
            "attemptat": "2020-06-01T20:05:00",
            "attemptip": "132.12.34.123",
            "attemptyorn": "Y"
        }
    ]
}
```

mongoDB - CRUD: CREATE

- create collection by inserting documents
  - db.collection.insertOne (….. JSON …..);
  - db.collection.insertMany - insert an array of JSON documents
    - insertMany ([JSON1, JSON2, …]);

```
> db.famous.insertOne ({
...    "_id": 1,
...    "name" : { "first" : "Edward", "last" : "Codd" },
...    "contribs" : [ "Database Relational Model", "Database Technology", "OLAP"
 ],
...    "awards" : [
...      {
...        "award" : "Turing Award",
...        "year" : 1981,
...        "by" : "Association for Computing Machinery"
...      }
...    ]
... });
{ "acknowledged" : true, "insertedId" : 1 }
>
```

mongoDB - CRUD: RETRIEVE

- Documents retrieved by find method on collection
  - db.famous.find ({}); or db.famous.find ({}).pretty()
    - find all

```
> show collections
famous
> db.famous.find ({});
{ "_id" : 1, "name" : { "first" : "Edward", "last" : "Codd" }, "contribs" : [ "D
atabase Relational Model", "Database Technology", "OLAP" ], "awards" : [ { "awar
d" : "Turing Award", "year" : 1981, "by" : "Association for Computing Machinery"
 } ] }
{ "_id" : 2, "name" : { "first" : "Chris", "last" : "Date" }, "contribs" : [ "Da
tabase Relational Model" ], "awards" : [ ] }
{ "_id" : 3, "name" : { "first" : "Peter", "last" : "Chen" }, "contribs" : [ "En
tity Relationship Modelling", "CASE", "IoT" ], "awards" : [ { "award" : "AAAI Al
len Newell Award", "year" : 1981, "by" : "Association for Computing Machinery" }
, { "award" : "Harry H. Goode Memorial Award", "year" : 2003, "by" : "IEEE Compu
ter Society" } ] }
{ "_id" : 4, "name" : { "first" : "Michael", "last" : "Stonebraker" }, "contribs
" : [ "Database Technology", "OLAP", "Ingres", "postgres" ], "awards" : [ { "awa
rd" : "Edgar F. Codd Innovations Award", "year" : 1992, "by" : "Association for
Computing Machinery" }, { "award" : "John von Neumann Medal", "year" : 2005, "by
" : "IEEE Computer Society" }, { "award" : "Turing Award", "year" : 2014, "by" :
 "Association for Computing Machinery" } ] }
```

mongoDB - CRUD: RETRIEVE continued

- Limit output to specified field (project fields) 1 display 0 suppress

```
> db.famous.find({},{_id: 0})
{ "name" : { "first" : "Edward", "last" : "Codd" }, "contribs" : [ "Database Relational Model", "Database Technology", "
OLAP" ], "awards" : [ { "award" : "Turing Award", "year" : 1981, "by" : "Association for Computing Machinery" } ] }
{ "name" : { "first" : "Chris", "last" : "Date" }, "contribs" : [ "Database Relational Model" ], "awards" : [ ] }
{ "name" : { "first" : "Peter", "last" : "Chen" }, "contribs" : [ "Entity Relationship Modelling", "CASE", "IoT" ], "awa
rds" : [ { "award" : "AAAI Allen Newell Award", "year" : 1981, "by" : "Association for Computing Machinery" }, { "award"
 : "Harry H. Goode Memorial Award", "year" : 2003, "by" : "IEEE Computer Society" } ] }
{ "name" : { "first" : "Michael", "last" : "Stonebraker" }, "contribs" : [ "Database Technology", "OLAP", "Ingres", "pos
tgres" ], "awards" : [ { "award" : "Edgar F. Codd Innovations Award", "year" : 1992, "by" : "Association for Computing M
achinery" }, { "award" : "John von Neumann Medal", "year" : 2005, "by" : "IEEE Computer Society" }, { "award" : "Turing
Award", "year" : 2014, "by" : "Association for Computing Machinery" } ] }
> db.famous.find({},{_id: 1, "name.first": 1, "name.last": 1})
{ "_id" : 1, "name" : { "first" : "Edward", "last" : "Codd" } }
{ "_id" : 2, "name" : { "first" : "Chris", "last" : "Date" } }
{ "_id" : 3, "name" : { "first" : "Peter", "last" : "Chen" } }
{ "_id" : 4, "name" : { "first" : "Michael", "last" : "Stonebraker" } }
```

mongoDB - CRUD: RETRIEVE continued

- Find some documents

  - db.famous.find ({ *predicate* });
    - find according to predicate, count() to count instances
      - `db.famous.find ({_id: 2});`
      - `db.famous.find ({contribs: "Database Technology"})`
      - `db.famous.find ({contribs: /.*Database.*/})`
      - `db.famous.find ({ "name.first": "Peter"})`
      - `db.famous.find ({"awards.by": "IEEE Computer Society" })`
      - `db.famous.find ({"awards.by": "IEEE Computer Society" }).count()`
    - Operators: https://docs.mongodb.com/manual/reference/operator/query/
      - `db.famous.find ({ $and: [ {contribs: "Database Technology"}, {contribs: "OLAP"}]})`
      - `db.famous.find ({$or: [{"awards.year": 2003},{"awards.year": 2005}]})`

mongoDB - CRUD: UPDATE

- Update documents via update or updateOne
  - uses $set to assign value
  - updateOne ({*query condition*},{*update to carry out*}
    - `db.famous.updateOne ( {"_id": 2},{ $set: {"name.first": "Christopher"}})`

```
> db.famous.findOne({_id:2},{"name.first": 1,"name.last": 1})
{ "_id" : 2, "name" : { "first" : "Chris", "last" : "Date" } }
> db.famous.updateOne ( {"_id": 2},{ $set: {"name.first": "Christopher"}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
> db.famous.findOne({_id:2},{"name.first": 1,"name.last": 1})
{ "_id" : 2, "name" : { "first" : "Christopher", "last" : "Date" } }
```

- Update within an array
  - $ placeholder to update the first element that matches the query condition

```
> db.famous.insertOne(
... {
...   "_id": 10,
...   "name" : { "first" : "Dummy", "last" : "One" },
...   "contribs" : [  ],
...   "awards" : [
...     {
...       "award" : "Award 1",
...       "year" : 2001,
...       "by" : "Made Up"
...     },
...     {
...       "award" : "Award 2",
...       "year" : 2005,
...       "by" : "Made Up"
...     }
...   ]
... });
{ "acknowledged" : true, "insertedId" : 10 }
> db.famous.updateOne ( { $and: [{_id: 10},{"awards.award": "Award 1"},{"awards.year": 2001}]}, { $set: {"awards.$.year": 2020}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
> db.famous.find({_id: 10}).pretty()
{
        "_id" : 10,
        "name" : {
                "first" : "Dummy",
                "last" : "One"
        },
        "contribs" : [ ],
        "awards" : [
                {
                        "award" : "Award 1",
                        "year" : 2020,
                        "by" : "Made Up"
                },
                {
                        "award" : "Award 2",
                        "year" : 2005,
                        "by" : "Made Up"
                }
        ]
}
>
```

mongoDB - CRUD: DELETE

- Delete a document
  - via db.famous.deleteOne or db.famous.deleteMany

```
> db.famous.find({}).count()
5
> db.famous.deleteOne({_id: 10})
{ "acknowledged" : true, "deletedCount" : 1 }
> db.famous.find({}).count()
4
> db.famous.find({})
{ "_id" : 1, "name" : { "first" : "Edward", "last" : "Codd" }, "contribs" : [ "Database Relational Model", "Database
Technology", "OLAP" ], "awards" : [ { "award" : "Turing Award", "year" : 1981, "by" : "Association for Computing Mach
inery" } ] }
{ "_id" : 2, "name" : { "first" : "Chris", "last" : "Date" }, "contribs" : [ "Database Relational Model" ], "awards"
: [ ] }
{ "_id" : 3, "name" : { "first" : "Peter", "last" : "Chen" }, "contribs" : [ "Entity Relationship Modelling", "CASE",
 "IoT" ], "awards" : [ { "award" : "AAAI Allen Newell Award", "year" : 1981, "by" : "Association for Computing Machin
ery" }, { "award" : "Harry H. Goode Memorial Award", "year" : 2003, "by" : "IEEE Computer Society" } ] }
{ "_id" : 4, "name" : { "first" : "Michael", "last" : "Stonebraker" }, "contribs" : [ "Database Technology", "OLAP",
"Ingres", "postgres" ], "awards" : [ { "award" : "Edgar F. Codd Innovations Award", "year" : 1992, "by" : "Associatio
n for Computing Machinery" }, { "award" : "John von Neumann Medal", "year" : 2005, "by" : "IEEE Computer Society" },
{ "award" : "Turing Award", "year" : 2014, "by" : "Association for Computing Machinery" } ] }
> db.famous.deleteMany({"awards.year": 1981})
{ "acknowledged" : true, "deletedCount" : 2 }
> db.famous.find({}).count()
2
> db.famous.find({})
{ "_id" : 2, "name" : { "first" : "Chris", "last" : "Date" }, "contribs" : [ "Database Relational Model" ], "awards"
: [ ] }
{ "_id" : 4, "name" : { "first" : "Michael", "last" : "Stonebraker" }, "contribs" : [ "Database Technology", "OLAP",
"Ingres", "postgres" ], "awards" : [ { "award" : "Edgar F. Codd Innovations Award", "year" : 1992, "by" : "Associatio
n for Computing Machinery" }, { "award" : "John von Neumann Medal", "year" : 2005, "by" : "IEEE Computer Society" },
{ "award" : "Turing Award", "year" : 2014, "by" : "Association for Computing Machinery" } ] }
>
```

Generate JSON object from Oracle Select

```
set pagesize 50 -- this sets the output page size, prevents
                -- the output heading appearing every 5 lines

SELECT
    JSON_OBJECT (
        '_id' VALUE studid,
        'name' VALUE studfname
                     || ' '
                     || studlname,
        'contactInfo' VALUE JSON_OBJECT (
                                 'address' VALUE studaddress,
                                 'phone' VALUE rtrim(studphone),
                                 'email' VALUE studemail
                             )
    FORMAT JSON )
    || ','
FROM
    uni.student;
```

Q3. Run the query on the previous slide in SQL and paste the output to Moodle

```
set pagesize 50
SELECT
    JSON_OBJECT (
        '_id' VALUE studid,
        'name' VALUE studfname
                     || ' '
                     || studlname,
        'contactInfo' VALUE JSON_OBJECT (
                                'address' VALUE studaddress,
                                'phone' VALUE rtrim(studphone),
                                'email' VALUE studemail
                            )
    FORMAT JSON )
    || ','
FROM
    uni.student;
```