MONASH University

MONASH INFORMATION TECHNOLOGY

Week 11 – Non Relational Databases
Big Data

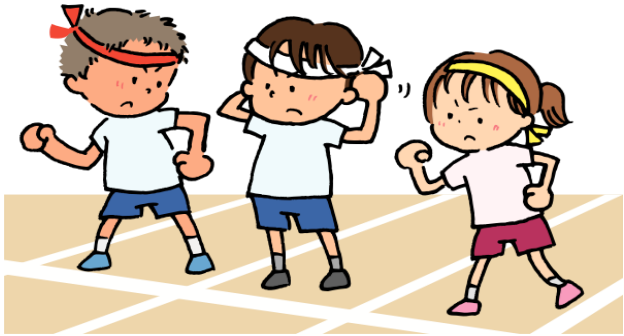
FIT3171 Databases
Semester 1 2022

Malaysia Campus

Preparation for the Forum - ready, set ……

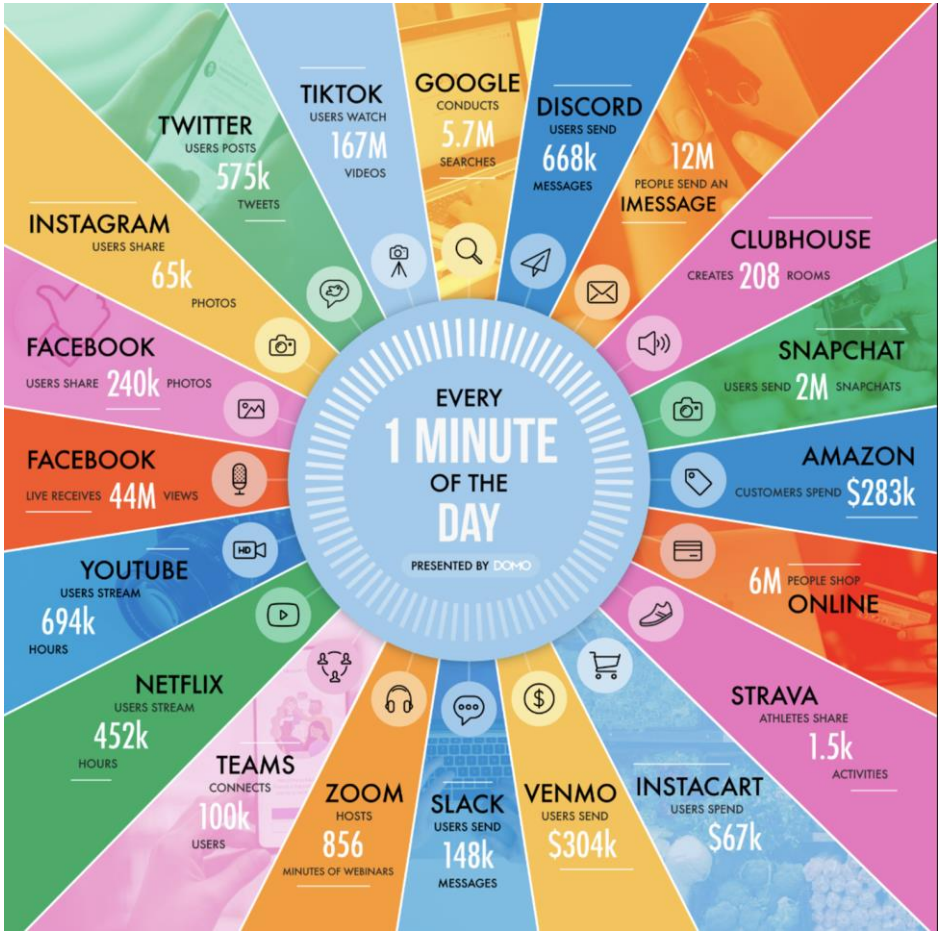
Please

- connect to Flux - flux.qa and be ready to answer questions
  - https://flux.qa/QBGYRS
- login to Oracle:
  - SQL Developer (MoVE or Local) OR
  - ORDS https://ora-fit.ocio.monash.edu:8441/ords/sql-developer



Data Growth 2021

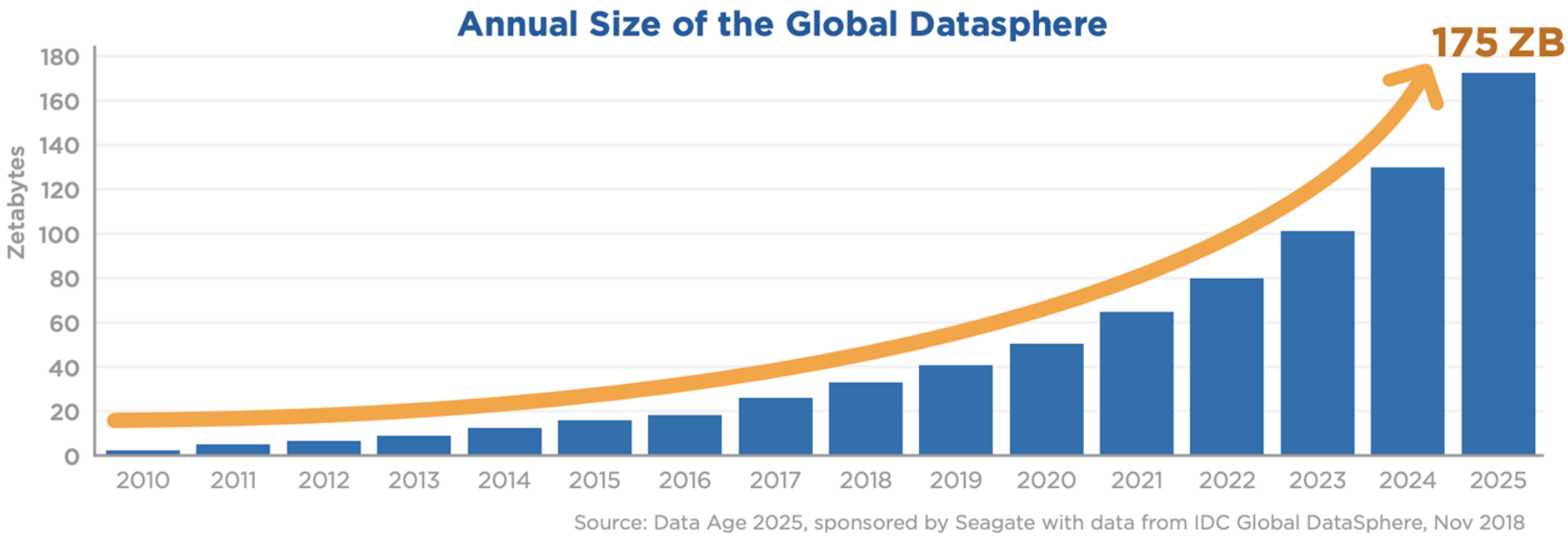
As of July 2021, the internet reaches 65% of the world's population and now represents 5.17 billion people—a 10% increase from January 2021. Of this total, 92.6 percent accessed the internet via mobile devices. According to Statista, the total amount of data consumed globally in 2021 was 79 zettabytes, an annual number projected to grow to over 180 zettabytes by 2025.



Source: https://www.domo.com/learn/data-never-sleeps-9

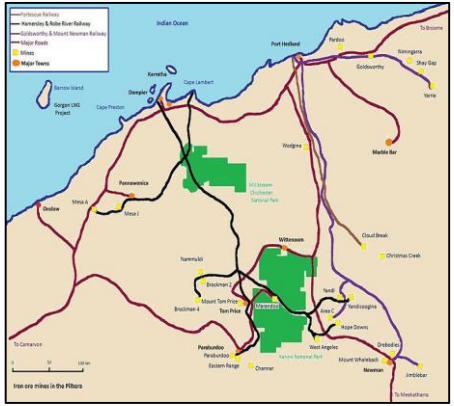
Data Growth

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

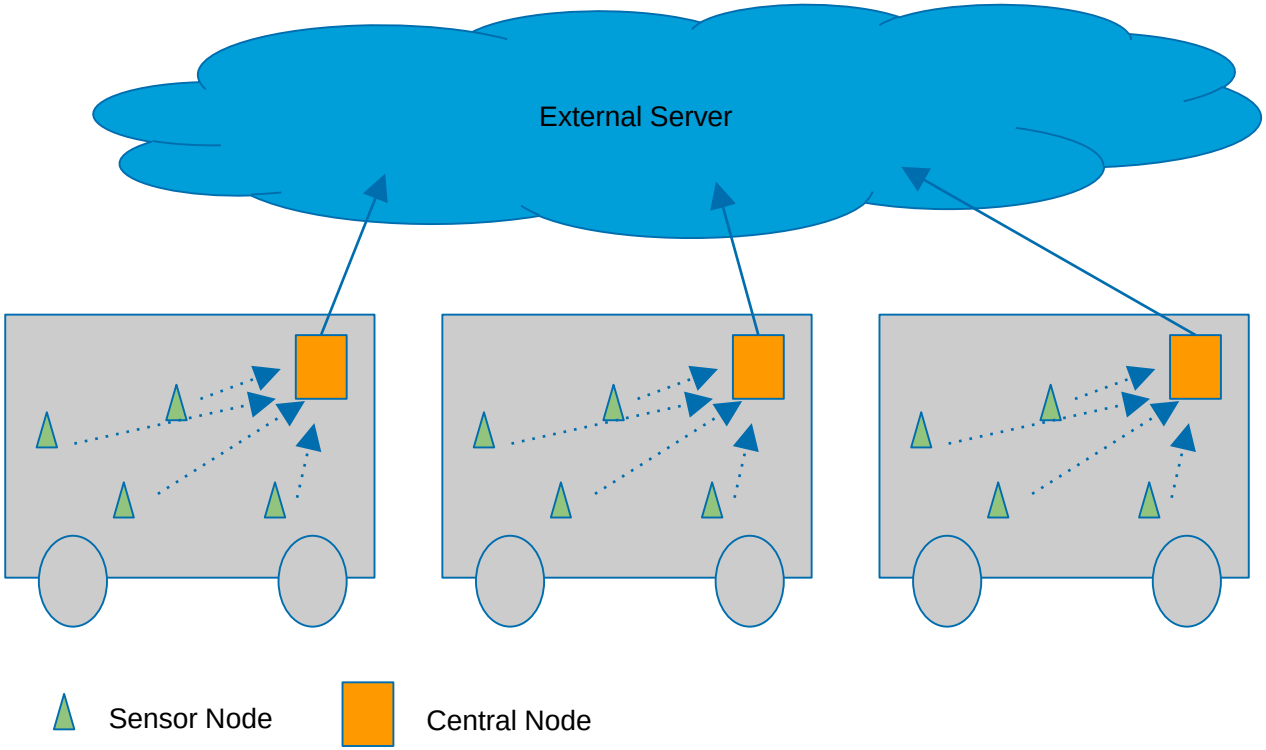


- Pilbara region, WA
- Trains perform round trips from the mining site to the port
- Loaded minerals and ores

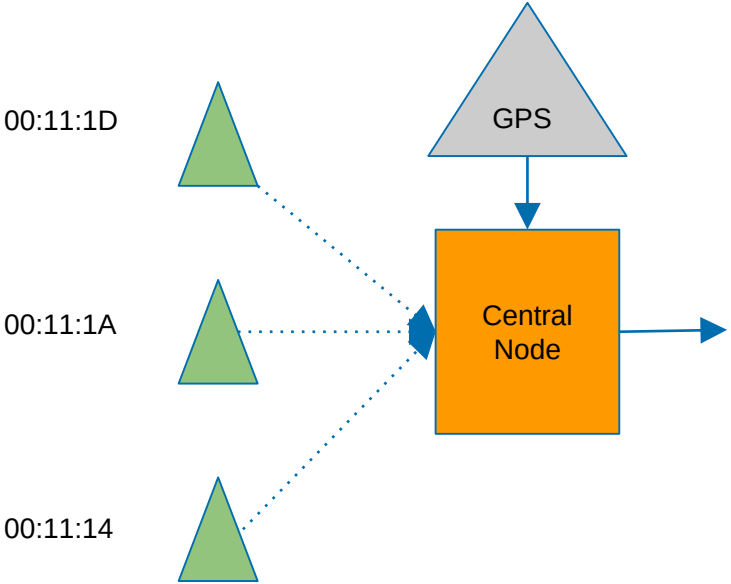
- Length: > 2KM
- Load: > 10 Ton/car
- Speed: 5-10 Km/hr

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

Solution adopted: Network Structure



Central Node Process



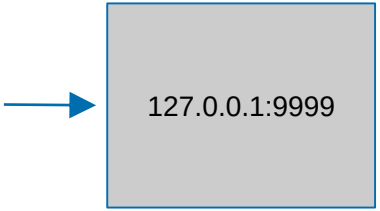

```
{
gps: {
                    lat: ###,
                    lng: ### }
sensors:{
                    “00:11:14”: {

                    acceler:

                    strain: }
                    “00:11:1A”: {
                          acceler: }
                    “00:11:1D”: {

                    accler: }}
}
```

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

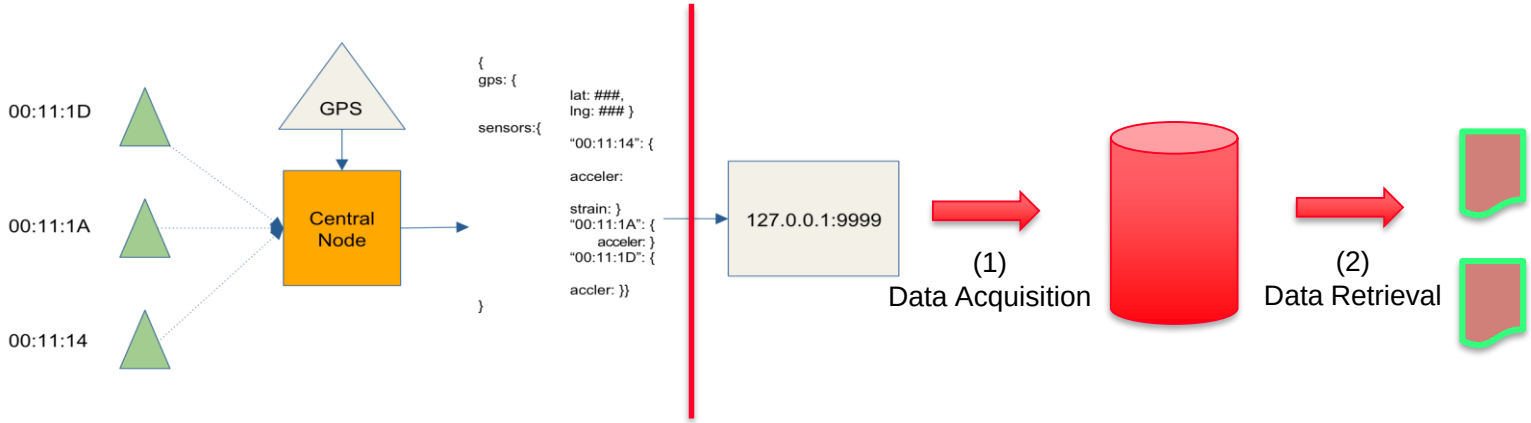
```
Welcome to Ubuntu 14.04.3 LTS (GNU/Linux 3.13.0-46-generic x86_64)

 * 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
 >
```

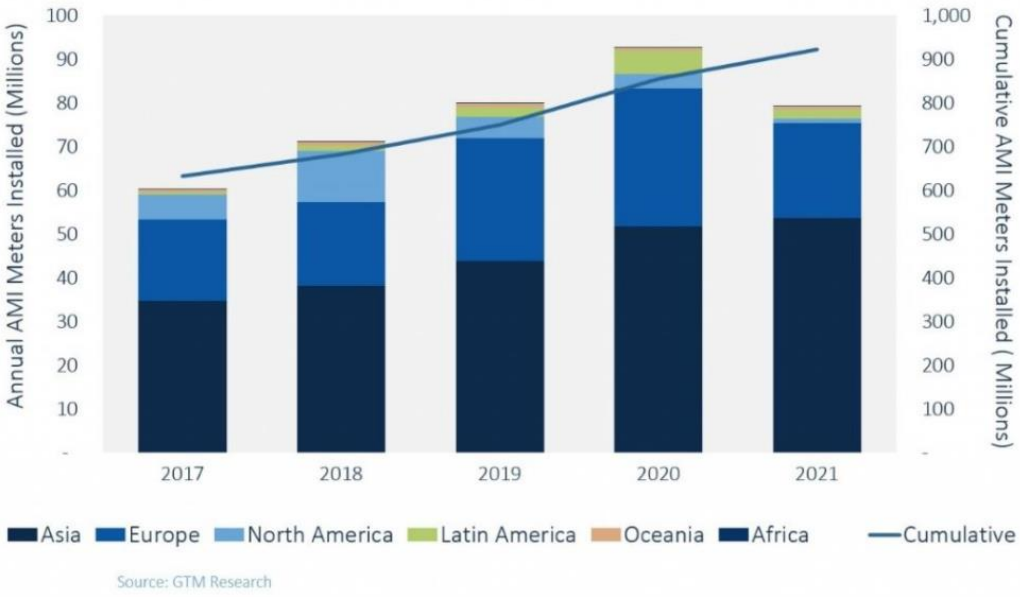
Big Data Processing



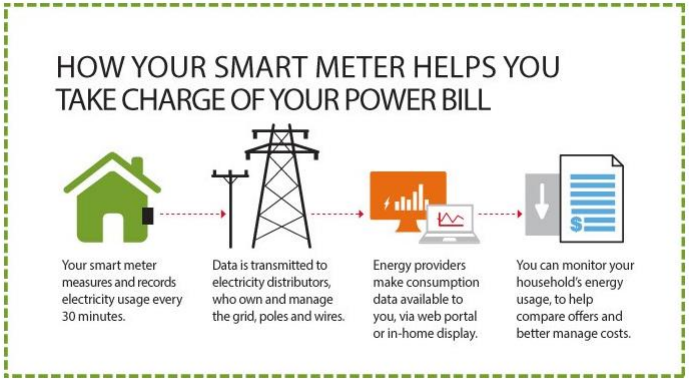
Two main problems:

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

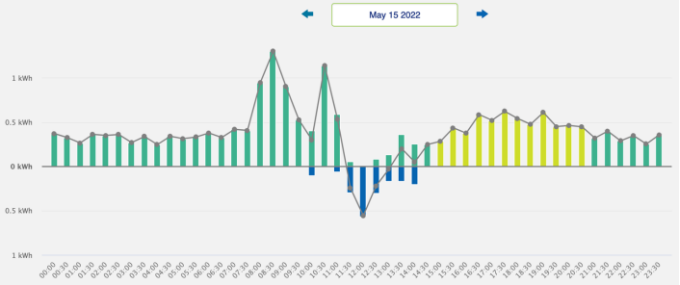
Advanced metering infrastructure (AMI) - Smart Meters



https://www.energynetworks.com.au/news/energy-insider/get-smart-when-will-australia-realise-the-benefits-of-smart-meters-and-iot/



https://www.victorianenergysaver.vic.gov.au/get-help-with-your-bills/smart-meters-and-how-they-work



Q1. Which of the following is NOT a characteristic of Big Data (multiple selections are possible):

A. Scaling Up
B. Volume
C. Veracity
D. Variety
E. Hadoop
F. Velocity
G. HDFS

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

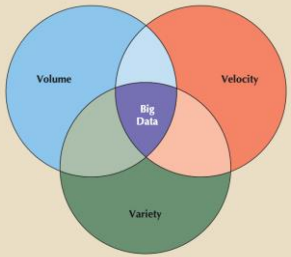
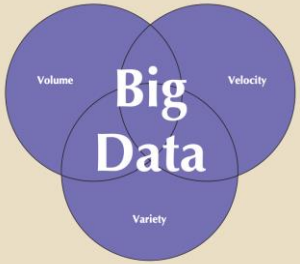


FIGURE 14.2 CURRENT VIEW OF BIG DATA



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 continued

- Big players, notably Google and Amazon chose Scale Out
  - 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: Dynamo
    - http://www.read.seas.harvard.edu/~kohler/class/cs239-w08/decandia07dynamo.pdf
    - Based on Dynamo: https://aws.amazon.com/dynamodb/

Scaling continued

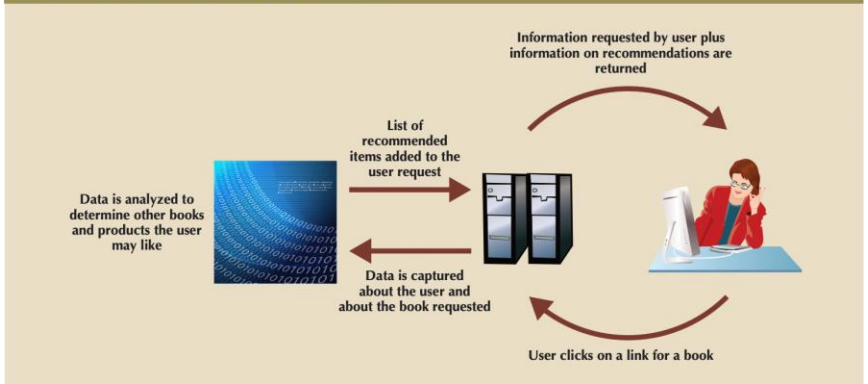
- Term "NoSQL" coined by John Oskarsson in 2009 after calling a ..."free meetup about “open source, distributed, non relational databases” or NOSQL for short"…
  - http://blog.oskarsson.nu/post/22996139456/nosql-meetup

- 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 ⇨ 1GB 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.

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
  - Distribution
    - Hadoop Distributed File System (HDFS): low-level distributed file processing system that can be used directly for data storage
  - Processing
    - MapReduce: programming model that supports processing large data sets

Q2. The four main categories of NoSQL databases are (select multiple answers):

A. Aggregate-aware
B. Key-Value
C. Document
D. JSON
E. Column-oriented
F. Graph

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)

FIGURE 14.7 KEY-VALUE DATABASE STORAGE

Bucket = Customer

Key	Value
10010	"LName Ramas FName Alfred Initial A Areacode 615 Phone 844-2573 Balance 0"
10011	"LName Dunne FName Leona Initial K Areacode 713 Phone 894-1238 Balance 0"
10014	"LName Orlando FName Myron Areacode 615 Phone 222-1672 Balance 0"

  - 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

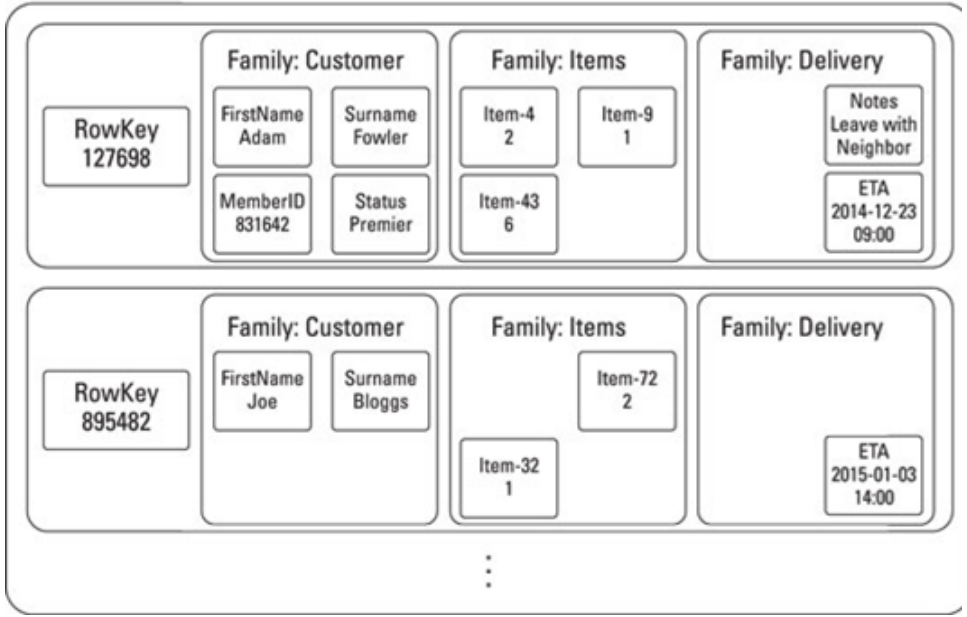
```
{  maker: "M.V. Agusta",
   type: "sportsbike",
   engine: {
      type: "internal combustion",
      cylinders: 4,
      displacement: 750
    },
   rake:7,
   trail:3.93
}

{   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

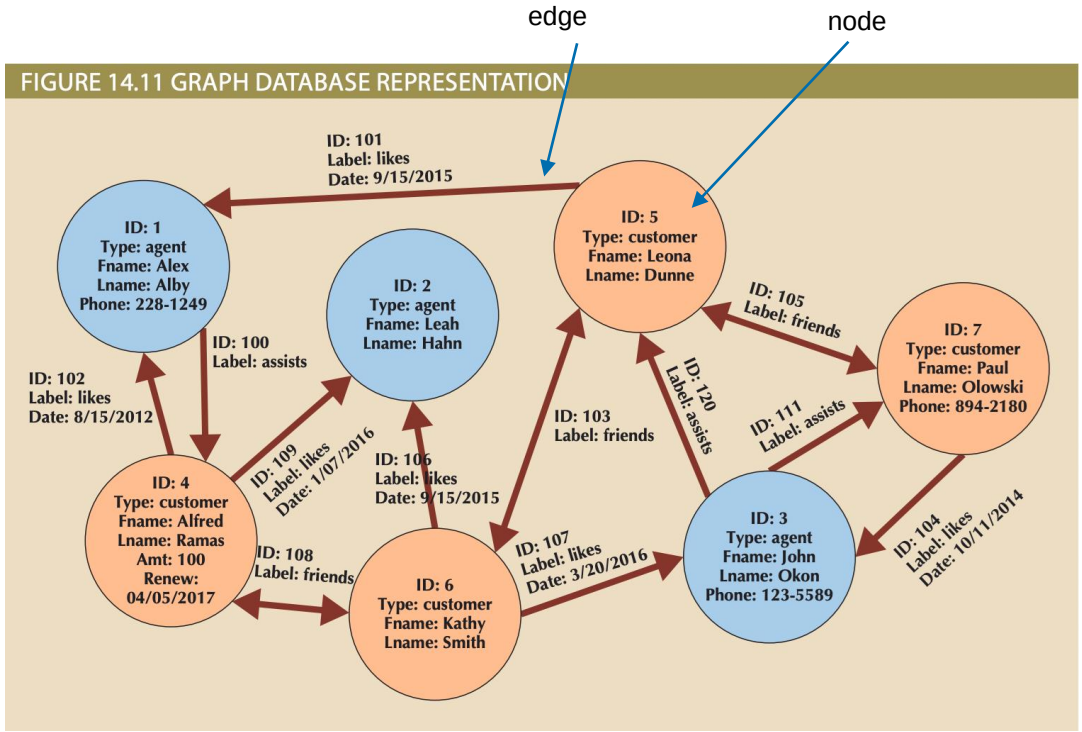


https://www.dummies.com/programming/big-data/columnar-data-in-nosql/

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



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

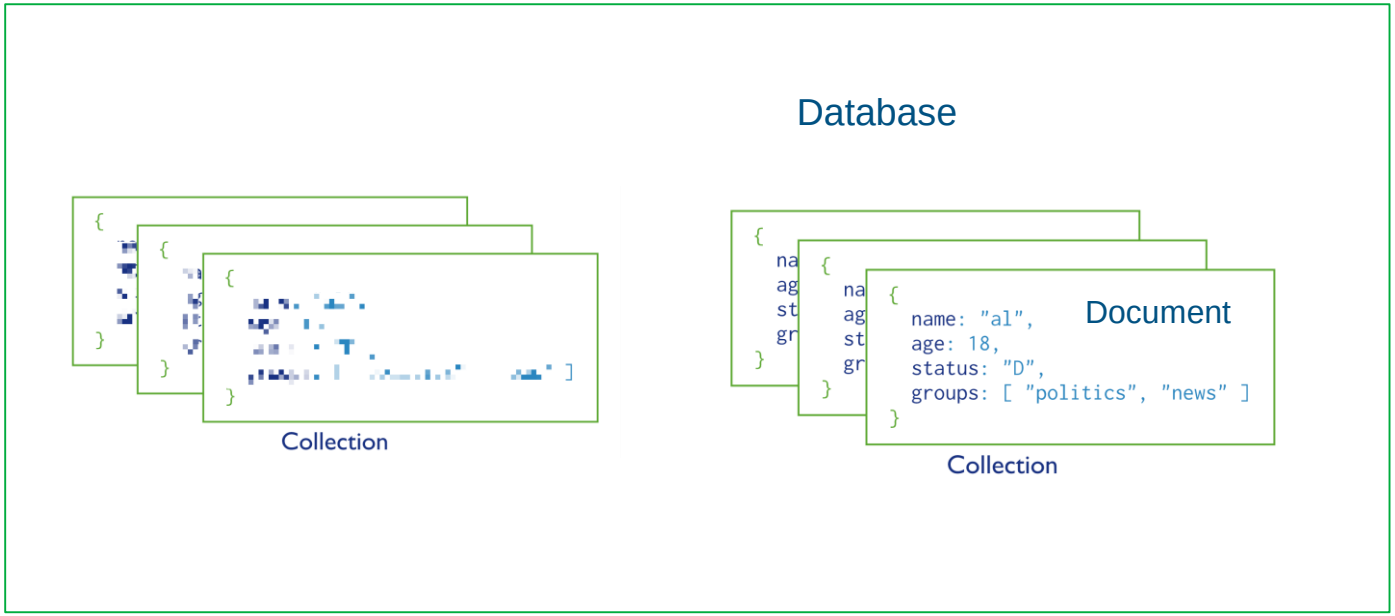
NoSQL Databases

- 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
  - **Mixtures of Sharding/Replication**
- Lots of interesting questions and research around consistency vs availability

MongoDB

- Document Database
  - Community edition available for download (not *required* for FIT3171 but may install - see Tutorial class)
    - 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)

```
{
  name: "sue",                    ← field: value
  age: 26,                        ← field: value
  status: "A",                    ← field: value
  groups: [ "news", "sports" ]    ← field: value
}
```

- 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, marked with [ ]

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 - Drone Rentals

```
{
    "drone_id": 111,
    "type": {
        "code": "PAPR",
        "model": "Parrot Pro",
        "manufacturer": "Parrot"
    },
    "carrying_capacity": 5,
    "pur_date": "2020-03-20",
    "pur_price": 4200,
    "total_flighttime": 100,
    "cost_per_hour": 45,
    "RentalInfo": [
        {
            "rent_no": 11,
            "bond": 150,
            "rent_out": "2020-04-26",
            "rent_in": "2020-04-28",
            "custtrain_id": 10
        }
    ]
}
```

- type - sub document
- RentalInfo - array of sub documents

Generate JSON object from Oracle Select - JSON functions

```
set pagesize 50 -- this sets the output page size, prevents the output heading appearing every 5 lines
SELECT
    JSON_OBJECT(
     'drone_id' VALUE drone_id,
     'type' VALUE JSON_OBJECT (
        'code' VALUE dt_code,
        'model' VALUE dt_model,
        'manufacturer' VALUE manuf_name
     ),
     'carrying_capacity' VALUE dt_carry_kg,
     'pur_date' VALUE to_char(drone_pur_date,'YYYY-MM-DD'),
     'pur_price' VALUE drone_pur_price,
     'total_flighttime' VALUE drone_flight_time,
     'cost_per_hour' VALUE drone_cost_hr,
     'RentalInfo' VALUE JSON_ARRAYAGG (
          JSON_OBJECT (
            'rent_no' VALUE rent_no,
            'bond' VALUE rent_bond,
            'rent_out' VALUE to_char(rent_out,'YYYY-MM-DD'),
            'rent_in' VALUE to_char(rent_in ,'YYYY-MM-DD'),
            'custtrain_id' VALUE ct_id
            )
            ORDER BY rent_no
          ) FORMAT JSON )
    || ','
FROM …
GROUP BY …
ORDER BY
    drone_id;
```

https://docs.oracle.com/en/database/oracle/oracle-database/12.2/adjsn/generation.html

Collect DRONE data from Oracle

SQL Worksheet History

0.036 seconds

Worksheet Query Builder

```
set pagesize 300

SELECT
    JSON_OBJECT(
     'drone_id' VALUE drone_id,
     'type' VALUE JSON_OBJECT (
        'code' VALUE dt_code,
        'model' VALUE dt_model,
        'manufacturer' VALUE manuf_name
     ),
     'carrying_capacity' VALUE dt_carry_kg,
     'pur_date' VALUE to_char(drone_pur_date,'YYYY-MM-DD'),
     'pur_price' VALUE drone_pur_price,
     'total_flighttime' VALUE drone_flight_time,
     'cost_per_hour' VALUE drone_cost_hr,
     'RentalInfo' VALUE JSON_ARRAYAGG (
          JSON_OBJECT (
            'rent_no' VALUE rent_no,
            'bond' VALUE rent_bond,
            'rent_out' VALUE to_char(rent_out,'YYYY-MM-DD'),
```

Script Output

Task completed in 0.036 seconds

```
JSON_OBJECT('DRONE_ID'VALUEDRONE_ID,'TYPE'VALUEJSON_OBJECT('CODE'VALUEDT_CODE,'MODEL'VALUEDT_M
--------------------------------------------------------------------------------
{"drone_id":100,"type":{"code":"DMA2","model":"DJI Mavic Air 2 Flymore Combo","manufacturer":"
{"drone_id":101,"type":{"code":"DMA2","model":"DJI Mavic Air 2 Flymore Combo","manufacturer":"
{"drone_id":102,"type":{"code":"DSPA","model":"DJI Spark","manufacturer":"DJI Da-Jiang Innovat
{"drone_id":103,"type":{"code":"DIN2","model":"DJI Inspire 2","manufacturer":"DJI Da-Jiang Inn
{"drone_id":111,"type":{"code":"PAPR","model":"Parrot Pro","manufacturer":"Parrot"},"carrying_
{"drone_id":112,"type":{"code":"PAPR","model":"Parrot Pro","manufacturer":"Parrot"},"carrying_
{"drone_id":113,"type":{"code":"PAPR","model":"Parrot Pro","manufacturer":"Parrot"},"carrying_
{"drone_id":117,"type":{"code":"PAPR","model":"Parrot Pro","manufacturer":"Parrot"},"carrying_
{"drone_id":118,"type":{"code":"SWPS","model":"SwellPro Spry","manufacturer":"SwellPro"},"carr
{"drone_id":119,"type":{"code":"DIN2","model":"DJI Inspire 2","manufacturer":"DJI Da-Jiang Inn
{"drone_id":120,"type":{"code":"DIN2","model":"DJI Inspire 2","manufacturer":"DJI Da-Jiang Inn

11 rows selected.
```

1. Create a text document dronedata.txt using the output via [Visual Studio Code for the Web]
2. Format as JSON and save as dronedata.json

mongoDB - CRUD: CREATE

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

```
drone> db.dronerent.insertOne({
...     "drone_id": 100,
...     "type": {
.....         "code": "DMA2",
.....         "model": "DJI Mavic Air 2 Flymore Combo",
.....         "manufacturer": "DJI Da-Jiang Innovations"
.....     },
...     "carrying_capacity": 0,
...     "pur_date": "2020-01-13",
...     "pur_price": 1494,
...     "total_flighttime": 100,
...     "cost_per_hour": 15,
...     "RentalInfo": [
...         {
.....             "rent_no": 1,
.....             "bond": 100,
.....             "rent_out": "2020-02-20",
.....             "rent_in": "2020-02-20",
.....             "custtrain_id": 1
.....         },
...         {
.....             "rent_no": 4,
.....             "bond": 100,
.....             "rent_out": "2020-02-22",
.....             "rent_in": "2020-02-25",
.....             "custtrain_id": 4
.....         }
...     ]
... });
{
  acknowledged: true,
  insertedId: ObjectId("6281ecefc6d5875c821f89cb")
}
```

Add the first document to MongoDB

https://www.mongodb.com/docs/v4.4/tutorial/getting-started/

```
MongoDB Web Shell

Click to connect
Connecting...
MongoDB shell version v4.4.6
connecting to: mongodb://127.0.0.1:27017/?
authSource=admin&compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("38477088-a500-473d-b3e3-68937a69af18") }
MongoDB server version: 4.4.6
type "help" for help
>>>
```

Full

Reset

Clear

Which version of MongoDB is running: **`db.version()`**
Which databases do you have access to: **`show dbs`**
Create/use the drone database: **`use drone`**
Which database am I in: **`db`**
What collections do I have in this database: **`show collections`**

Add the first document to MongoDB continued

```
>>> use drone
switched to db drone
>>> db.dronerent.insertOne({
...     "drone_id": 100,
...     "type": {
...         "code": "DMA2",
...         "model": "DJI Mavic Air 2 Flymore Combo",
...         "manufacturer": "DJI Da-Jiang Innovations"
...     },
...     "carrying_capacity": 0,
...     "pur_date": "2020-01-13",
...     "pur_price": 1494,
...     "total_flighttime": 100,
...     "cost_per_hour": 15,
...     "RentalInfo": [
...         {
...             "rent_no": 1,
```

Full

Reset

Clear

Success:

```
{
    "acknowledged" : true,
    "insertedId" : ObjectId("6281b9d67c28b58e72528d61")
}
```

Now insert the remainder in one insertMany (note the use of an array [ ] to contain the set of documents)

mongoDB - CRUD: RETRIEVE

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

```
drone> show collections
dronerent
drone> db.dronerent.find()
[
  {
    _id: ObjectId("6281dee6c6d5875c821f89b6"),
    drone_id: 100,
    type: {
      code: 'DMA2',
      model: 'DJI Mavic Air 2 Flymore Combo',
      manufacturer: 'DJI Da-Jiang Innovations'
    },
    carrying_capacity: 0,
    pur_date: '2020-01-13',
    pur_price: 1494,
    total_flighttime: 100,
    cost_per_hour: 15,
    RentalInfo: [
      {
        rent_no: 1,
        bond: 100,
        rent_out: '2020-02-20',
        rent_in: '2020-02-20',
        custtrain_id: 1
      },
      {
        rent_no: 4,
        bond: 100,
        rent_out: '2020-02-22',
        rent_in: '2020-02-25',
        custtrain_id: 4
```

mongoDB - CRUD: RETRIEVE continued

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

```
drone> db.dronerent.find({},{_id: 0})
[
  {
    drone_id: 100,
    type: {
      code: 'DMA2',
      model: 'DJI Mavic Air 2 Flymore Combo',
      manufacturer: 'DJI Da-Jiang Innovations'
    },
    carrying_capacity: 0,
    pur_date: '2020-01-13',
    pur_price: 1494,
    total_flighttime: 100,
    cost_per_hour: 15,
    RentalInfo: [
      {
        rent_no: 1,
        bond: 100,
        rent_out: '2020-02-20',
        rent_in: '2020-02-20',
        custtrain_id: 1
      },
      {
        rent_no: 4,
        bond: 100,
        rent_out: '2020-02-22',
        rent_in: '2020-02-25',
        custtrain_id: 4
      }
    ]
  },
  {
    drone_id: 101,
    type: {
      code: 'DMA2',
      model: 'DJI Mavic Air 2 Flymore Combo',
```

```
drone> db.dronerent.find({},{_id: 0, "drone_id":1, "type.model":1})
[
  { drone_id: 100, type: { model: 'DJI Mavic Air 2 Flymore Combo' } },
  { drone_id: 101, type: { model: 'DJI Mavic Air 2 Flymore Combo' } },
  { drone_id: 102, type: { model: 'DJI Spark' } },
  { drone_id: 103, type: { model: 'DJI Inspire 2' } },
  { drone_id: 111, type: { model: 'Parrot Pro' } },
  { drone_id: 112, type: { model: 'Parrot Pro' } },
  { drone_id: 113, type: { model: 'Parrot Pro' } },
  { drone_id: 117, type: { model: 'Parrot Pro' } },
  { drone_id: 118, type: { model: 'SwellPro Spry' } },
  { drone_id: 119, type: { model: 'DJI Inspire 2' } },
  { drone_id: 120, type: { model: 'DJI Inspire 2' } }
]
```

- count documents returned

```
drone> db.dronerent.find({}).count()
11
drone>
```

mongoDB - CRUD: RETRIEVE continued

- Find some documents
  - Predicate Operators: https://docs.mongodb.com/manual/reference/operator/query/
    - Example: `{ <field>: { $eq: <value> } }  => db.inventory.find( { qty: { $eq: 20 } } )`

Comparison

For comparison of different BSON type values, see the specified BSON comparison order.

Name	Description
`$eq`	Matches values that are equal to a specified value.
`$gt`	Matches values that are greater than a specified value.
`$gte`	Matches values that are greater than or equal to a specified value.
`$in`	Matches any of the values specified in an array.
`$lt`	Matches values that are less than a specified value.
`$lte`	Matches values that are less than or equal to a specified value.
`$ne`	Matches all values that are not equal to a specified value.
`$nin`	Matches none of the values specified in an array.

Logical

Name	Description
`$and`	Joins query clauses with a logical `AND` returns all documents that match the conditions of both clauses.
`$not`	Inverts the effect of a query expression and returns documents that do *not* match the query expression.
`$nor`	Joins query clauses with a logical `NOR` returns all documents that fail to match both clauses.
`$or`	Joins query clauses with a logical `OR` returns all documents that match the conditions of either clause.

mongoDB - CRUD: RETRIEVE continued

```
db.dronerent.find ({})
db.dronerent.find ({}).count()
db.dronerent.find ({},{_id: 0, "drone_id": 1, "type.model": 1})
```

Find some documents

a. find the details of drone id 102
b. find the details of all drones of type DIN2
c. find the details of all drones which have a carrying capacity > 4
    - ➢ display drone id, model and cost per hour
d. find the details of all drones which have a carrying capacity <= 5 and a cost per hour of < 50
    - ➢ display drone id, carrying capacity and cost per hour
e. how many drones are still on loan?
f. which drones are still out on loan
    - ➢ display drone id, when the drone went out, check your answer by doing a count first

mongoDB - CRUD: RETRIEVE continued

Find some documents

a. find the details of drone id 102

```
db.dronerent.find ({"drone_id": {$eq: 102}})
```

a. find the details of all drones of type DIN2

```
db.dronerent.find ({"type.code": {$eq: "DIN2"}})
```

a. find the details of all drones which have a carrying capacity > 4
   - ➢ display drone id, model and cost per hour

```
db.dronerent.find (
        {"carrying_capacity": {$gt: 4}},
        {"drone_id": 1, "type.model": 1,
"cost_per_hour": 1, "_id": 0}
)
```

mongoDB - CRUD: RETRIEVE continued

d. find the details of all drones which have a carrying capacity <= 5 and a cost per hour of < 50

- ➢ display drone id, carrying capacity and cost per hour

```
db.dronerent.find (
    { $and: [{"carrying_capacity": {$lte: 5}},{"cost_per_hour": {$lt:
50}}]},
    {"drone_id": 1, "carrying_capacity": 1, cost_per_hour: 1, "_id": 0}
)
```

e. how many drones are still on loan?

```
db.dronerent.find ({"RentalInfo.rent_in": {$eq: null}}).count()
```

f. which drones are still out on loan

- ➢ display drone id, when the drone went out, check your answer by doing a count first

```
db.dronerent.find (
{"RentalInfo.rent_in": {$eq: null}},
{"drone_id":1, "RentalInfo.rent_out":1, "_id":0 }
)
```

mongoDB - CRUD: UPDATE

- Update documents via update or updateOne
  - uses $set to assign value
  - updateOne ({*query condition*},{*update to carry out*}

```
drone> db.dronerent.find (
...     {"drone_id": {$eq: 103}},
...     {"type.model": 1, "pur_date": 1, "total_flighttime" :1, "_id": 0 }
... )
[
  {
    type: { model: 'DJI Inspire 2' },
    pur_date: '2020-01-13',
    total_flighttime: 200
  }
]
drone> db.dronerent.updateOne (
...     { "drone_id": {$eq: 103}}, {$set: {"total_flighttime": 230} }
... )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
drone> db.dronerent.find( { "drone_id": { $eq: 103 } }, { "type.model": 1, "pur_date": 1,
 "total_flighttime": 1, "_id": 0 })
[
  {
    type: { model: 'DJI Inspire 2' },
    pur_date: '2020-01-13',
    total_flighttime: 230
  }
]
drone>
```

mongoDB - CRUD: UPDATE

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

```
drone> db.dronerent.find( { $and: [{ "drone_id": { $eq: 118 } }, { "RentalInfo.rent_out": "2021-04-
13" }] }, { "RentalInfo": { $slice: -1 }, "_id": 0 })
[
  {
    drone_id: 118,
    type: { code: 'SWPS', model: 'SwellPro Spry', manufacturer: 'SwellPro' },
    carrying_capacity: 0,
    pur_date: '2020-04-01',
    pur_price: 1599,
    total_flighttime: 56.3,
    cost_per_hour: 16,
    RentalInfo: [
      {
        rent_no: 25,
        bond: 180,
        rent_out: '2021-04-13',
        rent_in: null,
        custtrain_id: 18
      }
    ]
  }
]
drone> db.dronerent.updateOne (
...     { $and: [ {"drone_id": {$eq: 118}}, {"RentalInfo.rent_out": "2021-04-13"}]},
...     {$set: {"RentalInfo.$.rent_in": "2021-04-18"} }
... )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

mongoDB - CRUD: UPDATE Result

- Update within an array

BEFORE

```
drone> db.dronerent.find( { $and: [{ "drone_id": { $eq: 118 } }, { "RentalInf
o.rent_out": "2021-04-13" }] }, { "RentalInfo": { $slice: -1 }, "_id": 0 })
[
  {
    drone_id: 118,
    type: { code: 'SWPS', model: 'SwellPro Spry', manufacturer: 'SwellPro' },
    carrying_capacity: 0,
    pur_date: '2020-04-01',
    pur_price: 1599,
    total_flighttime: 56.3,
    cost_per_hour: 16,
    RentalInfo: [
      {
        rent_no: 25,
        bond: 180,
        rent_out: '2021-04-13',
        rent_in: null,
        custtrain_id: 18
      }
    ]
  }
]
```

AFTER

```
drone> db.dronerent.find( { $and: [{ "drone_id": { $eq: 118 } }, { "RentalInf
o.rent_out": "2021-04-13" }] }, { "RentalInfo": { $slice: -1 }, "_id": 0 })
[
  {
    drone_id: 118,
    type: { code: 'SWPS', model: 'SwellPro Spry', manufacturer: 'SwellPro' },
    carrying_capacity: 0,
    pur_date: '2020-04-01',
    pur_price: 1599,
    total_flighttime: 56.3,
    cost_per_hour: 16,
    RentalInfo: [
      {
        rent_no: 25,
        bond: 180,
        rent_out: '2021-04-13',
        rent_in: '2021-04-18',
        custtrain_id: 18
      }
    ]
  }
]
drone>
```

mongoDB - CRUD: DELETE

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

```
drone> db.dronerent.find({}).count()
11
drone> db.dronerent.deleteOne({"drone_id": {$eq: 103}})
{ acknowledged: true, deletedCount: 1 }
drone> db.dronerent.find({}).count()
10
drone> db.dronerent.deleteMany({"carrying_capacity": {$gt: 4}})
{ acknowledged: true, deletedCount: 6 }
drone> db.dronerent.find({}).count()
4
drone>
```

- Remove current database (local client only)
  - db.dropDatabase()