



MONASH  
University

MONASH  
INFORMATION  
TECHNOLOGY

# FIT5137 – Advanced Database Technology

Week 7 – Designing Column-Oriented Databases

Semester 2, 2020

Developed by:

Dr. Agnes Haryanto

[Agnes.Haryanto@monash.edu](mailto:Agnes.Haryanto@monash.edu)

# Using FLUX

1. Visit <http://flux.qa/> on your internet enabled device
2. Log in using your Monash account (not required if you are already logged in to Monash)
3. Click on the “+” to join audience
4. Enter the Audience Code: **ARIW1B**
5. Select FIT5137 in the Active Presentation menu
6. Answer questions when they pop up

Alternatively, the quiz can be accessed through:

<https://flux.qa/ARIW1B>

The screenshot shows a web browser window titled "FLUX" with the URL <https://flux.qa/#/feeds/5d368...>. The page is titled "Lecture 0". Below the title are four icons: a document, a video camera, a microphone, and a clock. The main content area is titled "Current Polls". It displays a multiple-choice poll with the question "What did you have for dinner?". The options are listed in a vertical list:

- A Pasta
- B Rice
- C Pizza
- D Salad
- E Nothing

# Agenda

1. Primary & Compound Keys
2. Guidelines for Designing Tables
3. Guidelines for Indexing
4. Cassandra Data Model

# Primary & Compound Keys

# Create Table

- To create a table:
  - `CREATE TABLE user ( first_name text,  
last_name text,  
PRIMARY KEY (first_name)  
);`
- To get the description of the newly created table:
  - `DESCRIBE TABLE user;`

# Primary Key

- A primary key identifies the location and order of stored data. It is defined at the time of table creation using PRIMARY KEY.
- EXAMPLE:
  - CREATE TABLE user ( first\_name text PRIMARY KEY,  
                         last\_name text  
                       ) ;

# Compound Keys

- The primary key in Cassandra can be a compound key, which refers to a key consisting of multiple columns.
- When compound keys are specified, the first key is considered for partitioning of data and is called the *partition key*. The data in each partition is clustered and ordered by the remaining keys.
- EXAMPLE:
  - PRIMARY KEY ((key1), key2, key3, key4)
    - Here the data is partitioned by key1 and clustered by key2, key3, key4.
  - PRIMARY KEY ((key1, key2), key3, key4)
    - Here the data is partitioned by key1, key2 and clustered by key3, key4.

# Guidelines for Designing Tables & Indexing

# Column-Oriented Databases

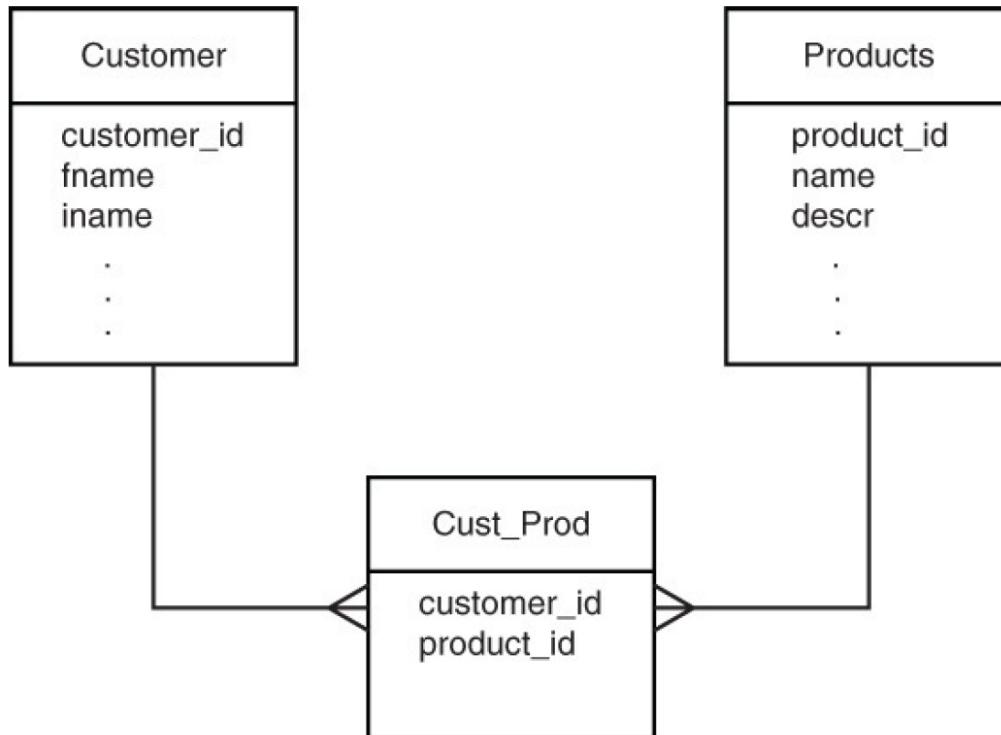
- Column-Oriented databases are implemented differently than relational databases.
- It is important to understand that:
  - Column family databases are implemented as sparse, multidimensional maps.
  - Columns can vary between rows.
  - Columns can be added dynamically.
  - Joins are not used; data is denormalized instead.
- These characteristics of column family databases will influence the database design guidelines.

# 1. Guidelines for Designing Tables

- Several guidelines when designing tables in Column-Oriented Databases:
  - Denormalize instead of join.
  - Make use of valueless columns.
  - Use both column names and column values to store data.
  - Model an entity with a single row.
  - Avoid hotspotting in row keys.
  - Keep an appropriate number of column value versions.
  - Avoid complex data structures in column values.
- It should be noted that some of these recommendations, such as using an appropriate number of column value versions, are not applicable to all column family database systems.

# Denormalize Instead of Join

- Column-oriented databases often need fewer tables than the relational databases.
  - It denormalize data to avoid the need for joins.
- Relational Database:
- Column-Oriented Database:



A diagram illustrating a column-oriented database structure. A Row Key points to two columns of data: Customer and Cust\_Prod. The Customer column contains rows for fname, lname, address, and a placeholder. The Cust\_Prod column contains rows for product\_id and product\_name.

Customer			
fname	lname	address	...
Mark	Jones	387 Main St.	

Cust_Prod		
38383	48282	59595
Dell Laptop	Apple iPhone	Galaxy Tab S



# Make Use of Valueless Columns

- Instead of having a column with a name like 'ProductPurchased1' with value 'PR\_B1839', the table simply stores the product ID as the column name.

ProductPurchased1
PR_B1839

VS

PR_B1839

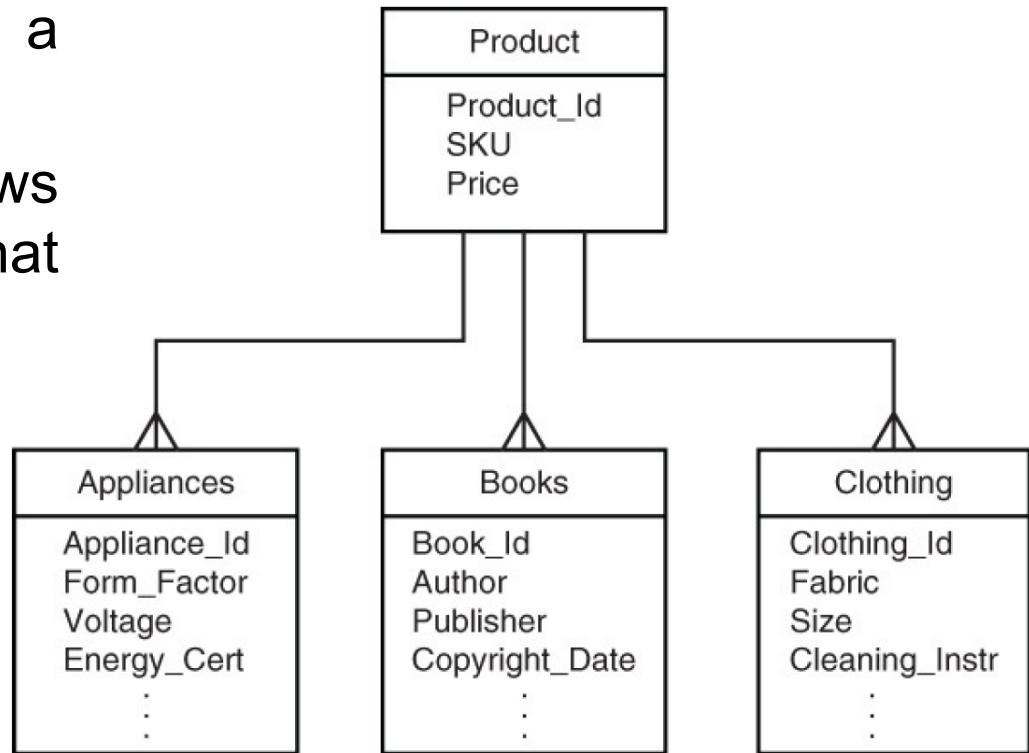
# Use Both Column Names and Column Values to Store Data

- Both the column name and the column value can store data.
- A variation on the use of valueless columns uses the column value for denormalization.
- The downside of denormalized data: Keeping a copy of the product name in the customer table will increase the amount of storage used.
- The benefit of denormalized data: The report of customers and the products they bought is produced by referencing only one table instead of two.
- In effect, we are trading the need for additional storage for improved read performance.

Cust_Prod		
Column Name	38383	48282
Column Value	Dell Laptop	Apple iPhone
Product ID	59595	Galaxy Tab S
Product Name		

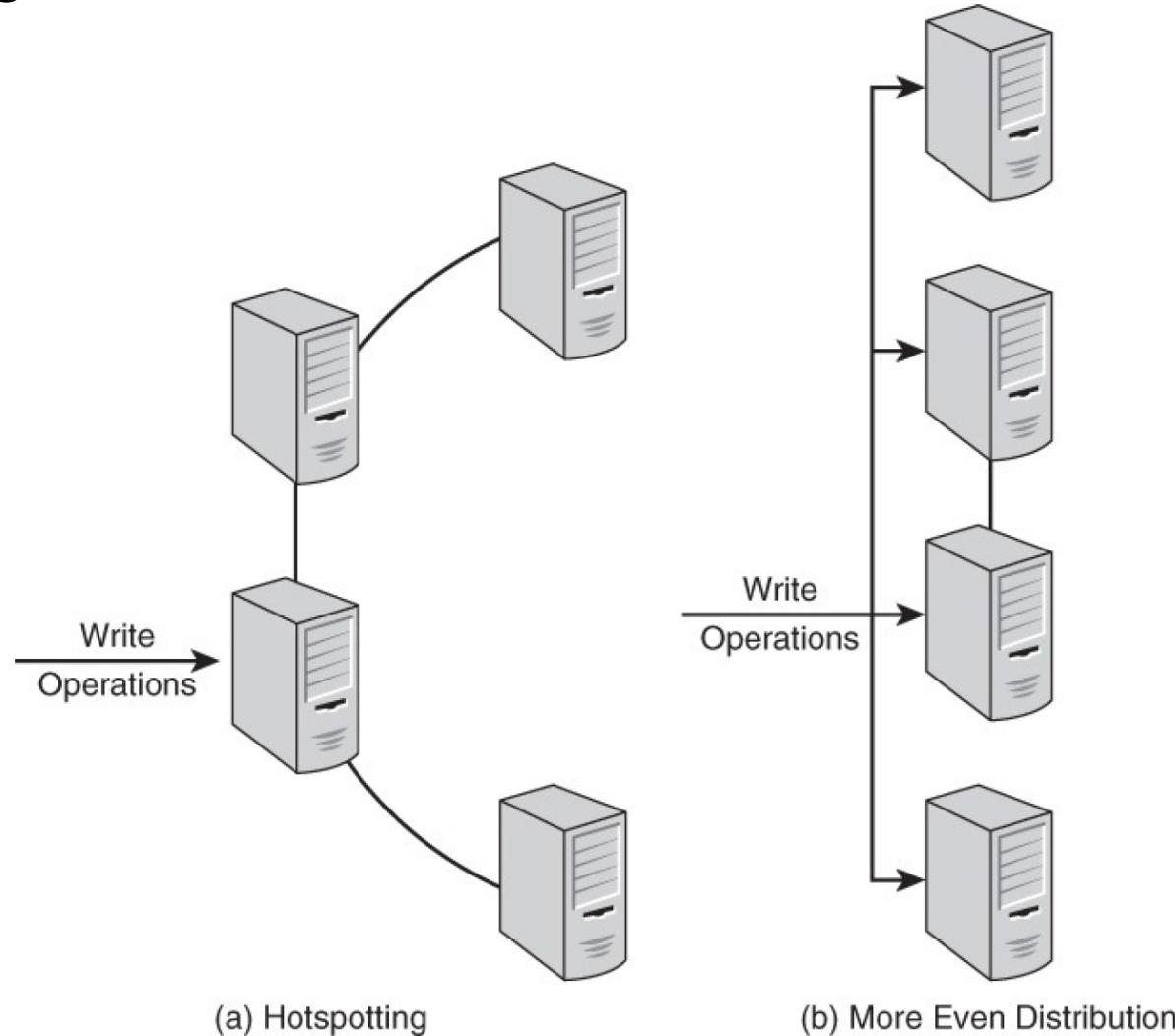
# Model an Entity with a Single Row

- A single entity, such as a particular customer or a specific product, should have all its attributes in a single row.
  - This can lead to cases in which some rows store more column values than others, but that is not uncommon in column family databases.
- Column family databases do not provide the same level of transaction control as relational databases. Typically, writes to a row are atomic. If you update several columns in a table, they will all be updated, or none of them will be.



# Avoid Hotspotting in Row Keys

- Distributed systems enable you to take advantage of large numbers of servers to solve problems. It is inefficient to direct an excessive amount of work at one or a few machines while others are underutilized.



# Keep an Appropriate Number of Column Value Versions

- HBase enables you to store multiple versions of a column value.
- Column values are timestamped so you can determine the latest and earliest values.
- Like other forms of version control, this feature is useful if you need to roll back changes you have made to column values.
- This is **not applicable** to Apache Cassandra.

Column Family		
Column Name <sub>1</sub>	Column Name <sub>2</sub>	Column Name <sub>3</sub>
value <sub>1a</sub> :timestamp <sub>1a</sub>	value <sub>2a</sub> :timestamp <sub>2a</sub>	value <sub>3a</sub> :timestamp <sub>3a</sub>
value <sub>1b</sub> :timestamp <sub>1b</sub>	value <sub>2b</sub> :timestamp <sub>2b</sub>	value <sub>3b</sub> :timestamp <sub>3b</sub>
value <sub>1c</sub> :timestamp <sub>1c</sub>	value <sub>2c</sub> :timestamp <sub>2c</sub>	value <sub>3c</sub> :timestamp <sub>3c</sub>

# Avoid Complex Data Structures in Column Values

- A JSON document about a customer, for example, might contain an embedded document storing address information, such as the following:

```
{  
    "customer_id":187693,  
    "name": "Kiera Brown",  
    "address" : {  
        "street" : "1232 Sandy Blvd.",  
        "city" : "Vancouver",  
        "state" : "Washington",  
        "zip" : "99121"  
    },  
    "first_order" : "01/15/2013",  
    "last_order" : " 06/27/2014"  
}
```

- This type of data structure may be stored in a column value, but it is not recommended unless there is a specific reason to maintain this structure.

# Avoid Complex Data Structures in Column Values

- Using separate columns for each attribute makes it easier to apply database features to the attributes. For example, creating separate columns for street, city, state, and zip means you can create secondary indexes on those values.
- Also, separating attributes into individual columns allows you to use different column families if needed. Both the ability to use secondary indexes and the option of separating columns according to how they are used can lead to improved database performance.

## 2. Guidelines for Indexing

- Indexes allow for rapid lookup of data in a table.
- It is helpful to distinguish two kinds of indexes:

- Primary Indexes

Primary indexes are indexes on the row keys of a table. They are automatically maintained by the column family database system.

- Secondary Indexes

Secondary indexes are indexes created on one or more column values. Either the database system or your application can create and manage secondary indexes. Not all column family databases provide automatically managed secondary indexes, but you can create and manage tables as secondary indexes in all column family database systems.

# Using Secondary Indexes Managed by the Column Family Database System

- If you need secondary indexes on column values and the column family database system provides automatically managed secondary indexes, then you should use them.
- The primary advantage of using automatically managed secondary indexes is they require less code to maintain than the alternative.
- There are times when you should not use automatically managed indexes. Avoid, or at least carefully test, the use of indexes in the following cases:
  - There is a small number of distinct values in a column.
  - There are many unique values in a column.
  - The column values are sparse.

# Using Secondary Indexes Managed by the Column Family Database System

- Columns with few distinct values are not good candidates for secondary indexes.
- When the number of distinct values in a column (known as the cardinality of the column) is small, indexes will not help performance much—it might even hurt.

Column Family				
Name	Address	Opt In?	City	.....
		Y		
		Y		
		N		
		Y		
		N		
		.		
		.		
		Y		
		N		

 Only Two Distinct Values

# Using Secondary Indexes Managed by the Column Family Database System

- Rows with too many distinct values are also not good candidates for indexes.
- Automatically managed indexes may not help much here because the index will have to maintain so much data it could take more time to search the index and retrieve the data than to scan the tables for the particular value.

Column Family				
Name	Address	City	State	Email
				ralken@gmail.com
				iman123@gmail.com
				dans37@yahoo.com
				marypdx@gmail.com
				gwashington@aol.com
				kcameron@future.com
				info@mybbiz.com
				.
				.
				.
				.

Many Distinct Values



# Using Secondary Indexes Managed by the Column Family Database System

- In cases where many of the rows do not use a column, a secondary index may not help.
  - Sparsely populated columns should not be indexed.

# When to Create and Manage Secondary Indexes Using Tables

**Customer**

<b>Row key</b>	<b>fname</b>	<b>Iname</b>	<b>street</b>	<b>city</b>	<b>state</b>
123	Jane	Smith	387 Main St	Boise	ID
287	Mark	Jones	192 Wellfleet Dr	Austin	TX
1987	Harsha	Badal	298 Commercial St	Provincetown	MA
2405	Senica	Washington	98 Morton Ave	Windsor	CT
3902	Marg	O'Malley	981 Circle Dr	Santa Fe	NM

**Product**

<b>Row key</b>	<b>name</b>	<b>descr</b>	<b>qty_avail</b>	<b>category</b>	
38383	Dell Latitude E6410	Laptop with ...	124	Computer	
48282	Apple iPhone	iPhone 6 with ...	345	Phone	
59595	Galaxy Tab S	Samsung tablet ...	743	Tablet	



# When to Create and Manage Secondary Indexes Using Tables

- To help generate reports that list all customers who bought a particular product.

**Cust\_by\_Prod**

Row key	123	287	1987	2405	3902
38383	Smith		Badal		
48282	Smith	Jones			O'Malley
59595				Washington	

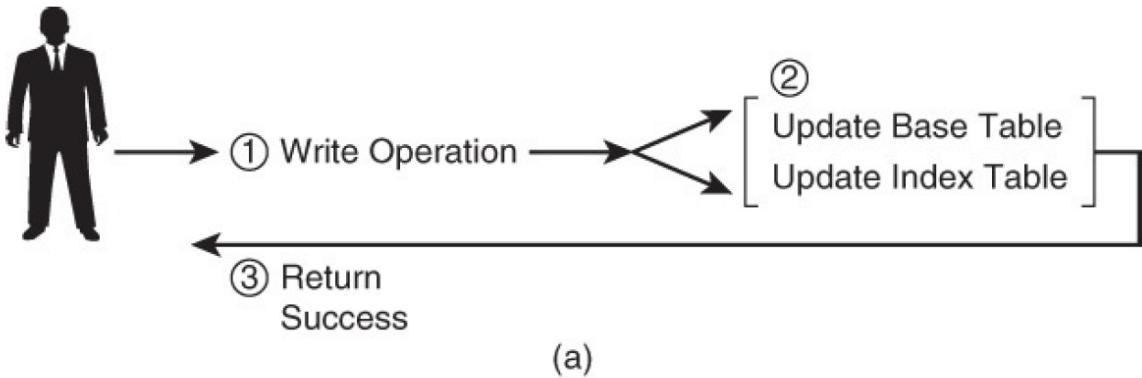
- They would also like a report on particular products and which customers bought them.

**Prod\_by\_Cust**

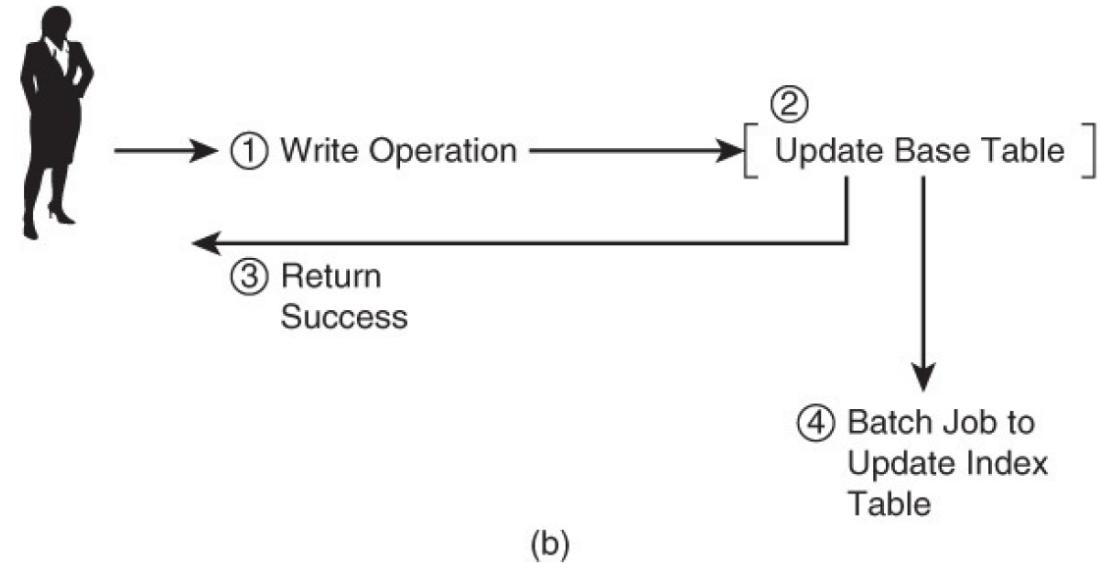
Row key	38383	48282	59595		
123	Dell Latitude E6410	Apple iPhone			
287		Apple iPhone			
1987	Dell Latitude E6410				
2405			Galaxy Tab S		
3902		Apple iPhone			

# When to Create and Manage Secondary Indexes Using Tables

- a) Updating an index table during write operations keeps data synchronized but increases the time needed to complete a write operation.



- b) Batch updates introduce periods of time when the data is not synchronized, but this may be acceptable in some case



# Cassandra Data Model

# Cassandra Data Model Rules

- Maximize the number of writes
  - Cassandra is optimized for high write performance.
  - Maximizing writes can be useful for read performance and data availability.
- Maximize Data Duplication
  - Data denormalization and data duplication are de-facto of Cassandra.
  - Disk space is generally the cheapest resource (compared to CPU, memory, disk IOPs, or network), and Cassandra is architected around that fact. In order to get the most efficient reads, you often need to duplicate data.
  - Data duplication provides instant data availability and no single point of failure.

# Data Modelling Goals

- Spread Data Evenly Around the Cluster
  - You want every node in the cluster to have roughly the same amount of data.
  - Rows are spread around the cluster based on a hash of the partition key, which is the first element of the PRIMARY KEY. So, the key to spreading data evenly is this: pick a good primary key.
- Minimize number of partitions read while querying data
  - Partitions are groups of rows that share the same partition key. When you issue a read query, you want to read rows from as few partitions as possible.
  - Need to choose a balanced number of partitions.

# Bad Primary Key

- Sample table MusicPlaylist:
  - Create table MusicPlaylist

```
(  
    songId int,  
    songName text,  
    year int,  
    singer text,  
    primary key(songId, songName)  
) ;
```

# Good Primary Key

- Sample table MusicPlaylist:
  - Create table MusicPlaylist

```
(songId int,  
songName text,  
year int,  
singer text,  
primary key((songId, year), songName)  
);
```
- Data will be clustered on the basis of SongName. In this table, each year, a new partition will be created. All the songs of the year will be on the same node.

# Handling One to One Relationship

- One to one relationship means two tables have one to one correspondence.
- Example:
  - Create table Student\_Course

```
(  
    Student rollno int primary key,  
    Student_name text,  
    Course_name text,  
) ;
```

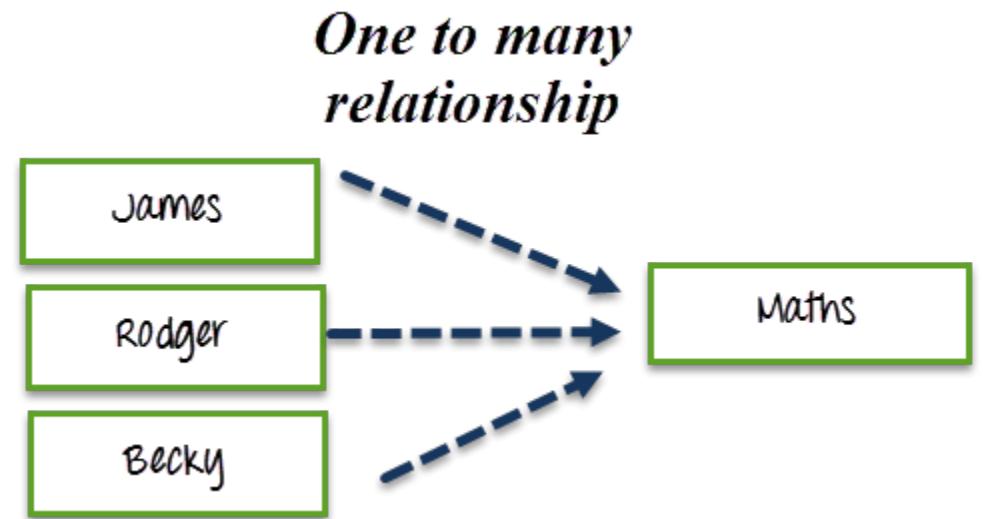
*One to one  
relationship*



# Handling One to Many Relationships

- One to many relationships means having one to many correspondence between two tables.
- Example:
  - Create table Student\_Course

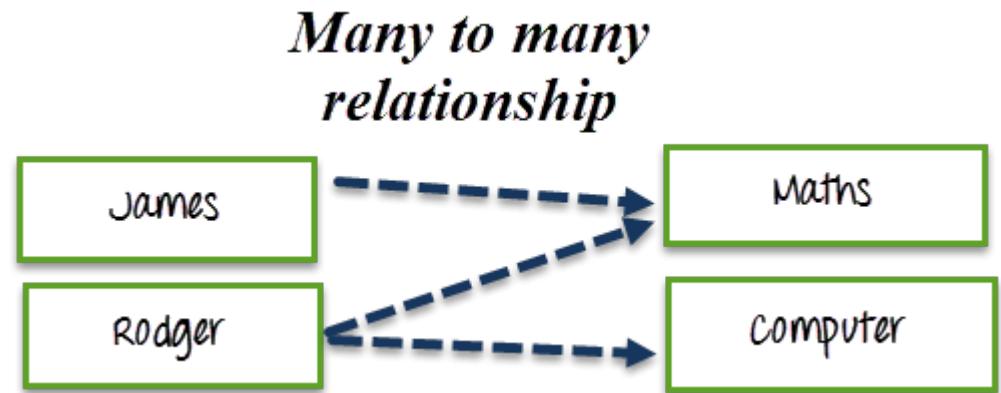
```
(  
    Student_rollno int,  
    Student_name text,  
    Course_name text,  
) ;
```
  - Select \* from Student\_Course where Course\_name='Course Name';



# Handling Many to Many Relationships

- Many to many relationships means having many to many correspondence between two tables.
- Example:

```
- Create table Student_Course  
(  
    Student_rollno int primary key,  
    Student_name text,  
    Course_name text,  
);  
- Create table Course_Student  
(  
    Course_name text primary key,  
    Student_name text,  
    student_rollno int  
);
```



# References

- [1] C. Carlos and M. Steven, Database systems: *Design, implementation, & management*. Cengage Learning, 12<sup>th</sup> ed., 2016.
- [2] C. Y. Kan, Cassandra Data Modeling and Analysis. Birmingham, UK: Packt Publishing, 1<sup>st</sup> ed., 2014.
- [3] D. Sullivan, NoSQL for Mere Mortals. Michigan, USA: Addison-Wesley Professional, 1<sup>st</sup> ed., 2015.
- [4] J. Carpenter and E. Hewitt, Cassandra: The Definitive Guide. Sebastopol, CA: O'Reilly Media, Inc., 2<sup>nd</sup> ed., 2016.
- [5] N. Neeraj, Mastering Apache Cassandra. Birmingham, UK: Packt Publishing, 1<sup>st</sup> ed., 2013.
- [6] R. Strickland, Cassandra 3.x High Availability. Birmingham, UK: Packt Publishing, 2<sup>nd</sup> ed., 2016.
- [7] S. Alapati, Expert Apache Cassandra Administration. Texas, USA: Apress, 1<sup>st</sup> ed., 2018.
- [8] Simplilearn, Apache Cassandra Advanced Architecture Tutorial. <https://www.simplilearn.com/cassandra-advanced-architecture-tutorial-video>.
- [9] Simplilearn, Apache Cassandra Architecture Tutorial. <https://www.simplilearn.com/cassandra-architecture-tutorial-video>.
- [10] W. Lemahieu, S. V. Broucke and B. Baesens, Principles of Database Management: *The Practical Guide to Storing, Managing and Analyzing Big and Small Data*. New York, NY, USA: Cambridge University Press, 1<sup>st</sup> ed., 2018.

## Tutorial Work 3

### Apache Cassandra

The main objectives of this tutorial are:

- To be able to do CRUD operations using Apache Cassandra.
- To be familiar with how Cassandra processes user queries.
- To be able to use secondary index.
- To be able to use collections in Apache Cassandra.
- To be able to use User-Defined Type (UDT).

**Important Reminder:**

- The unit consists of the Tutorial Work assessment (weighing 5%) conducted during week 4, 5, 7, 9, 10.
- **This tutorial work (weighing 1% out of 5%) is the first among the assessed tutorials works. It will be assessed during your tutorial time.**

**Marking Rubric:**

Mark	Description		
	Level of understanding	Preparation	Tutorial Task and Interview Questions
1	Exceptional	Student has prepared prior to the tutorial	Completed all of the tutorial tasks and has answered all interview questions without mistakes/errors
0.75	Competent	Student has prepared prior to the tutorial	either completed all of the tutorial tasks and has answered at least 50% of the interview questions without mistakes/errors  or: completed at least 50% of the tutorial tasks and has answered all interview questions without mistakes/errors
0.5	Moderate	Student has prepared prior to the tutorial	either: completed more than 50% of the tutorial tasks and answered less than 50% of the interview questions without mistakes/errors  or: completed less than 50% of the tutorial tasks and answered more than 50% of the interview questions without mistakes/errors  or: completed 50% of the tutorial tasks and answered 50% of the interview questions without mistakes/errors
0.25	Inadequate	Student has prepared prior to the tutorial	completed less than 50% of the tutorial tasks and answered less than 50% of the interview questions without mistakes/errors
0	No understanding	Student did not prepare prior to the tutorial	did not complete any of the tutorial tasks and did not answer any interview questions without mistakes/errors

## File Template for Answer

A file template (Week7\_CassandraExercise.cql) is provided for you on Moodle (under the Week 7 resources) to answer the tutorial work. You can use any word editor application (e.g. [Atom](#), [Notepad++](#)) to edit your answer in the provided file template.

## Connecting to Cassandra on Windows

1. Press the **Windows Key** → type **Run** in the search bar → **Enter**.
  2. Type **cmd** to open the **Command Prompt** → **Enter**.
  3. Go to **apache-cassandra-3.11.7\bin** and copy the directory.
  4. Type “**cd [directory]**”.
  5. Type **Cassandra** to start Cassandra server. The Command Prompt will load Cassandra. Wait until it stops loading.

```
Command Prompt
Microsoft Windows [Version 10.0.18362.295]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\agnes>Cassandra
Detected powershell execution permissions. Running with enhanced startup scripts.
*-----
*-----*
WARNING! Automatic page file configuration detected.
It is recommended that you disable swap when running Cassandra
for performance and stability reasons.

*-----*
*-----*
*-----*
*-----*

WARNING! Detected a power profile other than High Performance.
Performance of this node will suffer.
Modify conf\cassandra.env.ps1 to suppress this warning.

*-----*
*-----*

C:\Users\agnes>CompilerOracle: dontinline org/apache/cassandra/db/Columns$Serializer.deserializeLargeSubset (Lorg/apache/cassandra/io/util/DataInputPlus;Lorg/apache/cassandra/db/Columns;ILorg/apache/cassandra/db/Columns;
CompilerOracle: dontinline org/apache/cassandra/db/Columns$Serializer.serializeLargeSubset (Ljava/util/Collection;ILorg/apache/cassandra/db/Columns;ILorg/apache/cassandra/io/util/DataOutputPlus;)V
CompilerOracle: dontinline org/apache/cassandra/db/Columns$Serializer.serializeLargeSubsetSize (Ljava/util/Collection;ILorg/apache/cassandra/db/Columns;I)
```

6. Open another Command Prompt with the same Cassandra directory, then type **cqlsh**. The figure below indicates that you have successfully load Cassandra.

```
c:\ Command Prompt - cqlsh
Microsoft Windows [Version 10.0.18362.239]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\agnes>cqlsh

WARNING: console codepage must be set to cp65001 to support utf-8 encoding on Windows platforms.
If you experience encoding problems, change your console codepage with 'chcp 65001' before starting cqlsh.

Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.4 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
WARNING: pyreadline dependency missing. Install to enable tab completion.
cqlsh>
```

## Connecting to Cassandra on Mac

1. Start Cassandra.

```
$ brew services start cassandra
```

```
(base) Huashuns-MacBook-Pro:~ huashun$ brew services start cassandra
==> Successfully started `cassandra` (label: homebrew.mxcl.cassandra)
```

2. Go to /usr/local/Cellar/cassandra/3.11.7/bin.

```
$ cd /usr/local/Cellar/cassandra/3.11.7/bin
```

3. Initialize cassandra

```
$ cassandra
```

4. Open another terminal, then start CQL

```
$ cqlsh
```

```
-- 
(base) Huashuns-MacBook-Pro:~ huashun$ cqlsh
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.4 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
cqlsh>
```

## Cassandra

### Keyspace

- Create a new keyspace called `books_keyspace` in one cluster with a total of 3 nodes.
- Switch to the newly created keyspace.
- Alter the keyspace and update it to have a total node of 2 rather than 3.

### Create Operations

- Create a new column family called `books_by_author` that consists of the author name, book's publish year, book ID, book name, and rating. Make the author name, publish year, and book ID as the primary key. You should use UUID for the book ID.
- Insert the following data to the column family:

Author Name	Publish Year	Book Name	Rating
James Patterson	2008	Cross	3
Adam Liaw	2016	The Zen Kitchen	4.5
Rob Galea	2018	Breakthrough	4

Use UUID to create the book ID.

### Read Operations

- Find books that were written by James Patterson before 2009.
- Find books with rating of 4.5.  
Can you find any books? Write down your observation.

### Secondary Index

- Create a secondary index for rating.  
Can you find books with rating of 4.5 now?

### Timestamp

- Show the timestamp of book name.
- Show the timestamp of author's name.  
Write down your observation.
- Add a new row with the following details:  
Author's name : James Patterson  
Publish year : 2017

Book name	:	Manning
Rating	:	4.0

- d. Find all books written by James Patterson.

## Collections – Set

- Create a new column for emails that can store a set of email values.
- Add James Patterson’s email address: `james_patter@gmail.com`. Can you update all records of James Patterson with a single update command?
- If you cannot insert the email address from the previous question, insert the email address for each record of James Patterson.
- Add an email address for Adam Liaw: `adam_liaw@gmail.com`.
- Add another email address of Adam Liaw: `adamliauw@gmail.com`.
- Delete Adam Liaw’s email that we inserted first (`adam_liaw@gmail.com`).

## Collections – List

- Add a new column called series to store the list of other book titles of the series.
- Add “Along Came a Spider” as one of the series of James Patterson’s book titled Cross.
- Add another series of James Patterson’s Cross, which is “Jack and Jill” and “Cat and Mouse”.
- Delete “Jack and Jill” from the list.

## Collections – Map

- Add a new column called ISBN and store it as a Map collection:
- Add a new column to store this information for Adam Liaw’s The Zen Kitchen:  
ISBN-10: 0733634311  
ISBN-13: 978-0733634314
- Delete the ISBN-13 from Adam Liaw’s The Zen Kitchen.

## User-Defined Types (UDT)

- Create a UDT to store the product details. In this case, we want to store the number of pages, cover type (hardcover/softcover), book dimension length, width, height, and unit of measurement.

b. Add this detail to Adam Liaw's The Zen Kitchen:

Pages : 240  
Cover type: Hardcover  
Length : 20.8  
Width : 2.6  
Height : 25.6  
UOM : cm

**The End**



MONASH  
University

MONASH  
INFORMATION  
TECHNOLOGY

# FIT5137 – Advanced Database Technology

## Week 6 – Apache Cassandra Architecture & CQL

Semester 2, 2020

Developed by:

Dr. Agnes Haryanto

[Agnes.Haryanto@monash.edu](mailto:Agnes.Haryanto@monash.edu)

Huashun Li

[Huashun.Li@monash.edu](mailto:Huashun.Li@monash.edu)

# Using FLUX

1. Visit <http://flux.qa/> on your internet enabled device
2. Log in using your Monash account (not required if you are already logged in to Monash)
3. Click on the “+” to join audience
4. Enter the Audience Code: **XKV4LV**
5. Select FIT5137 in the Active Presentation menu
6. Answer questions when they pop up

Alternatively, the quiz can be accessed through:

<https://flux.qa/XKV4LV>

The screenshot shows a web browser window titled "FLUX" with the URL <https://flux.qa/#/feeds/5d368...>. The page is titled "Lecture 0". Below the title are four icons: a bar chart, a person, a gift, and a clock. The main content area is titled "Current Polls". It displays a multiple-choice poll with the question "What did you have for dinner?". The options are listed in a vertical list:

- A Pasta
- B Rice
- C Pizza
- D Salad
- E Nothing

# Agenda

1. Apache Cassandra Architecture (part 2)
2. CQL

# Apache Cassandra Architecture

# Apache Cassandra

- Cassandra was designed to address the following architecture requirements:
  - Highly fault tolerant with no single point of failure
    - If there are 100 nodes in a cluster and a node fails, the cluster should continue to operate.
  - Massive scalability
    - A cluster can hold hundreds or thousands of nodes.
    - It should be possible to add a new node to the cluster or even remove a node without stopping the cluster.

# Apache Cassandra

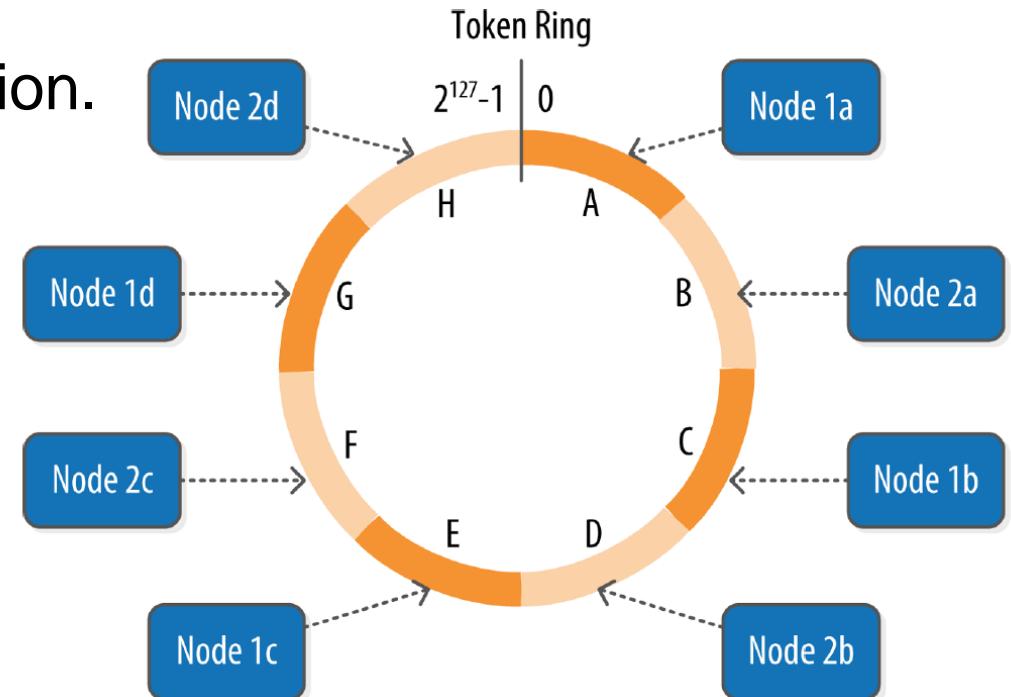
- Cassandra was designed to address the following architecture requirements:
  - Highly distributed
    - Both processing and data can be distributed.
    - Every node in the cluster is identical as there are no masters or slaves.
  - High performance of read and write data
    - Real-time processing of big data.

# Features of Apache Cassandra Architecture

- Cassandra has no master or slave nodes.
- It has a ring-type/peer-to-peer architecture.
- Data is automatically distributed across all the nodes.
- Data is replicated across the nodes for redundancy.
- Data is kept in memory and slowly written to the disk.
- Hash values of the keys are used to distribute the data among nodes in the cluster.
- Cassandra architecture supports multiple data centers.
- Data can be replicated across data centers.

# Rings and Tokens

- Cassandra represents the data managed by a cluster as a ring.
- Each node in a ring is described by a token.
- A token is a 64-bit ID used to identify each partition.
- Token range from  $-2^{63}$  to  $2^{63}-1$ .
- Data is assigned to nodes by using a hash function to calculate a token for the partition key.



# Effects of Apache Cassandra Architecture

- Cassandra architecture enables transparent distribution of data to nodes.
  - This means that you can determine the location of your data in the cluster based on the data.
- Any node can accept any request as there are no masters or slaves.
  - If a node has the data, it will return the data.
  - Else, it will send the request to the node that has the data.

# Effects of Apache Cassandra Architecture

- You can specify the number of replicas of the data to achieve the required level of redundancy.
  - For instance, if the data is very critical, you may want to specify a replication factor of 4 or 5.
  - If the data is not critical, you may specify just one or two.
- It also provides tunable consistency – the level of consistency can be specified as a trade-off with performance.
  - Transactions are always written to a commit-log on disk so that they are durable.

# Key structures

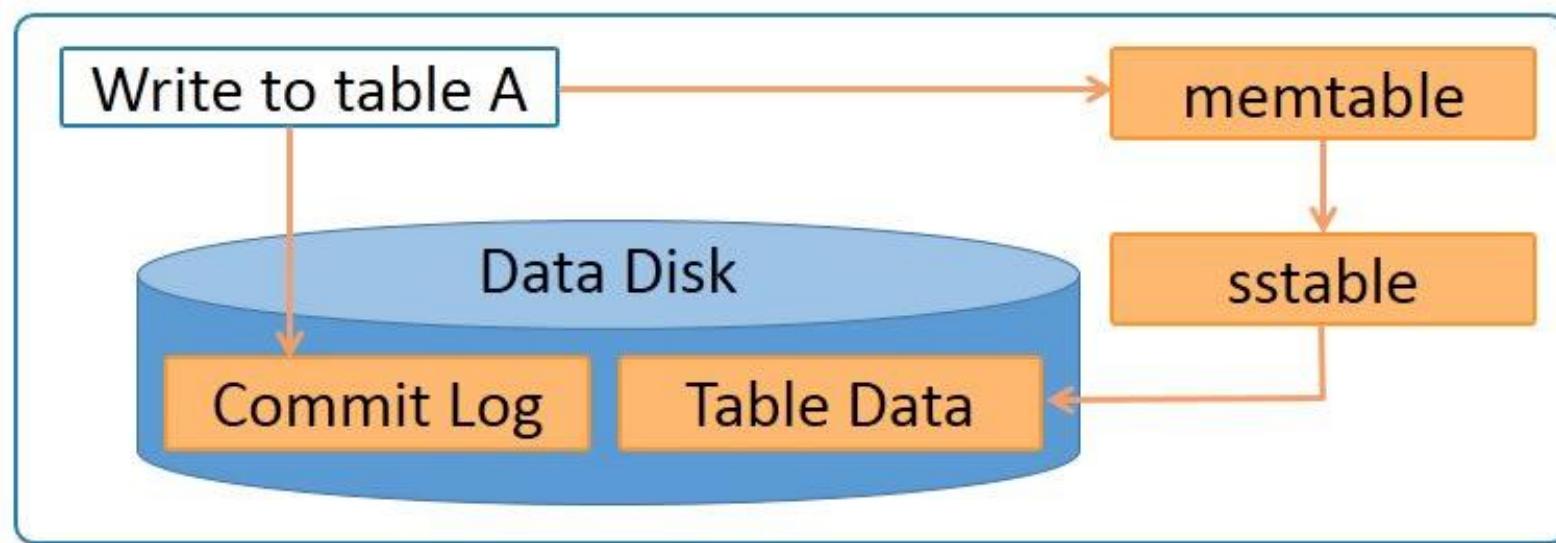
- Node: where you store your data.
- Data Center: a collection of related nodes.
- Cluster: contains one or more data centers.
- Commit log: all data is written first to the commit log for durability.
- Mem-table: a memory resident data structure.
- SSTable: a sorted string table (SSTable) is an immutable data file to which Cassandra writes memtables periodically.
- Bloom filter: the algorithms to test if an element is a member of a set.

# Cassandra Write Process

- Cassandra write steps:
  1. Data is written to a commitlog on disk.
  2. The data is sent to a responsible node based on the hash value.
  3. Nodes write data to an in-memory table called mem-table.
  4. From the mem-table, data is written to an SSTable in memory.
    - SSTable stands for Sorted String table. This has a consolidated data of all the updates to the table.
  5. From the SSTable, data is updated to the actual table.
  6. If the responsible node is down, data will be written to another node identified as tempnode. The tempnode will hold the data temporarily till the responsible node comes alive.

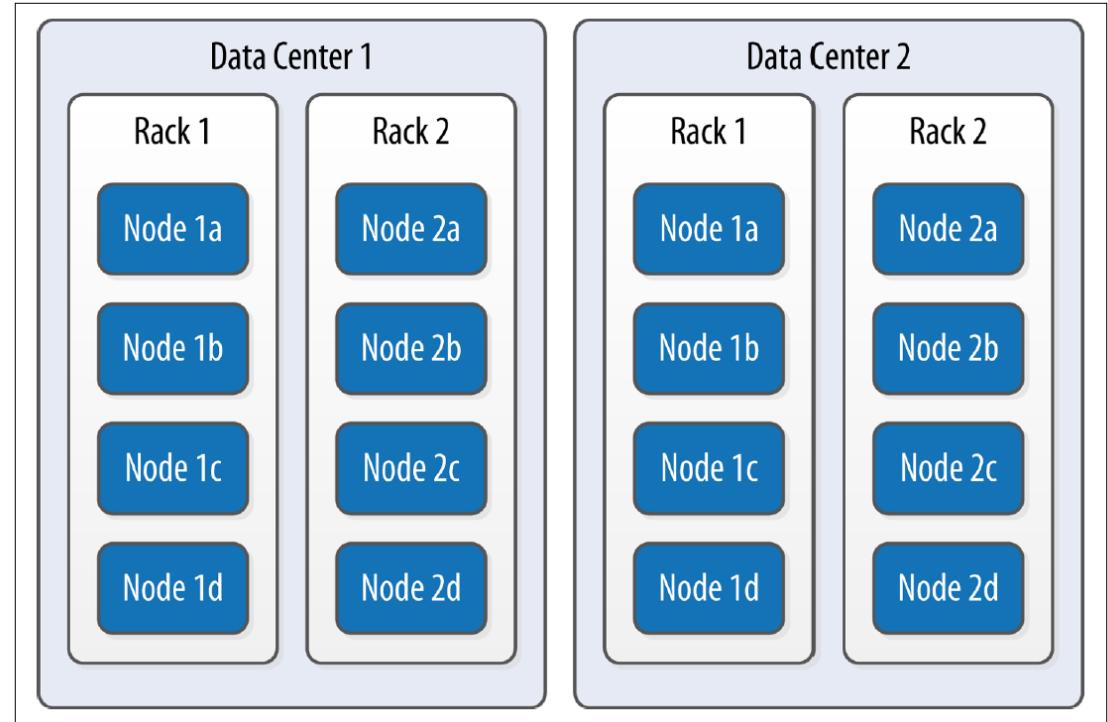
# Cassandra Write Process

- Data is written to a commitlog on disk for persistence.
- It is also written to an in-memory mem-table.
- Mem-table data is written to SSTable which is used to update the actual table.



# The Cassandra Architecture

- Data centers and racks
  - A rack is a logical set of nodes in close proximity to each other, perhaps on physical machines in a single rack of equipment
  - A data center is a logical set of racks, perhaps located in the same building and connected by reliable network



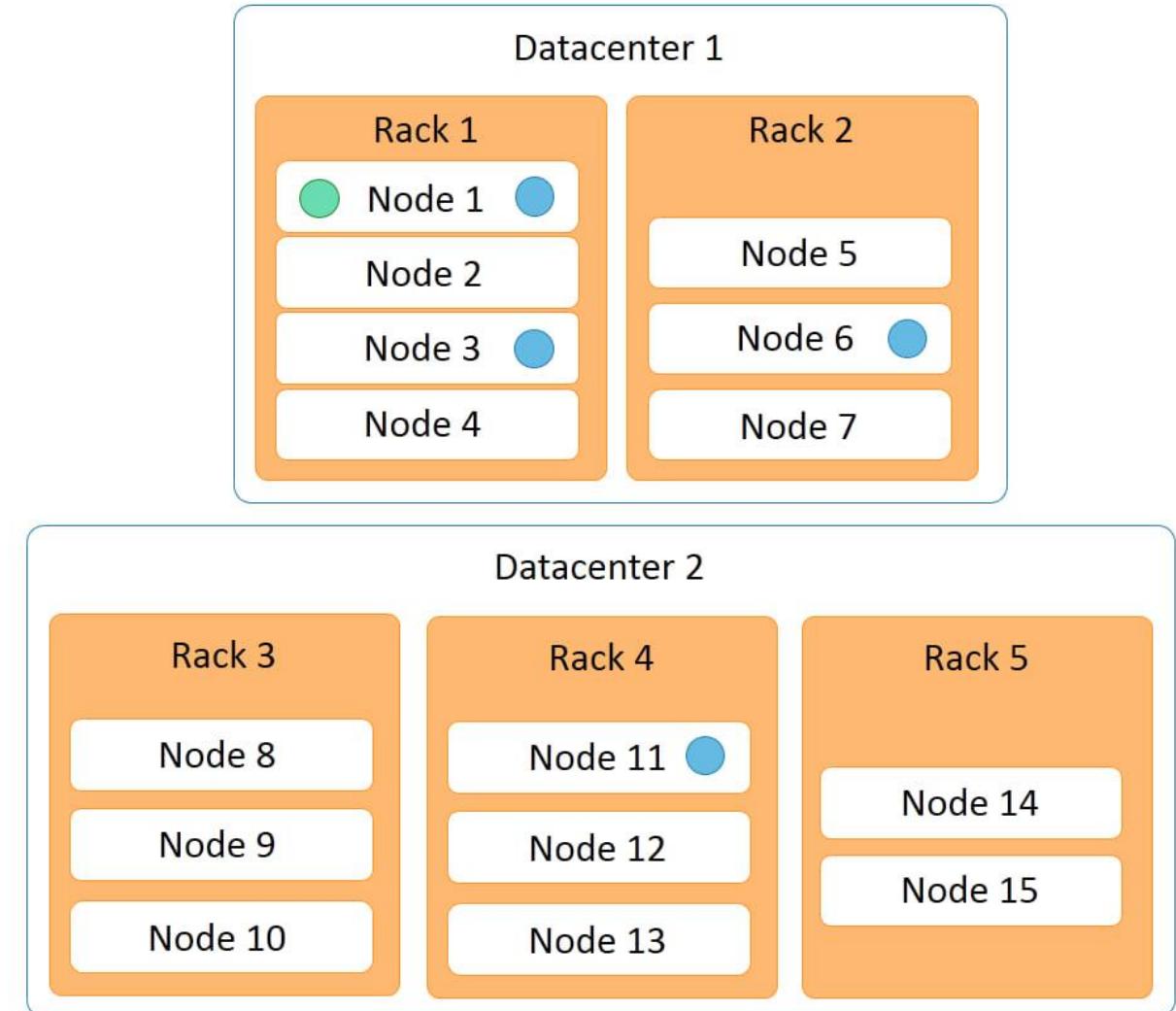
Topology of a sample cluster with data centers, racks, and nodes

# Cassandra Read Process

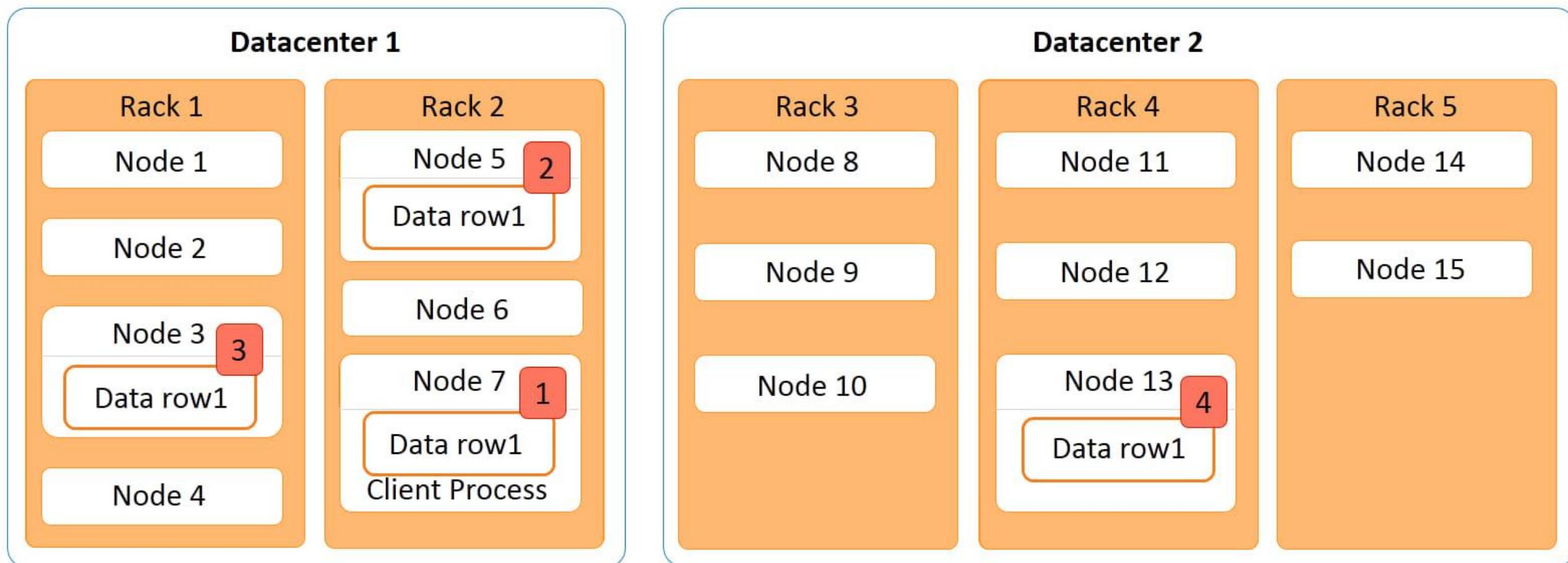
- The Cassandra read process ensures fast reads.
- If a node is down, data is read from the replica of the data. Priority for the replica is assigned on the basis of distance.
- Features of the Cassandra read process are:
  - Data on the same node is given first preference and is considered data local.
  - Data on the same rack is given second preference and is considered rack local.
  - Data on the same data center is given third preference and is considered data center local.
  - Data in a different data center is given the least preference.

# Cassandra Read Process

- This sample Cassandra cluster has two data centers:
  - Data center 1
    - Data center 1 has two racks.
    - Nodes 1 to 4 on rack 1, nodes 5 to 7 on rack 2.
  - Data center 2
    - Data center 2 has three racks.
    - Nodes 8 to 9 on rack 3, nodes 11 to 13 on rack 4, nodes 14 to 15 on rack 5.



# Cassandra Read Process – Example



# Data Partitions

- Cassandra performs transparent distribution of data by horizontally partitioning the data in the following manner:
  - A hash value is calculated based on the primary key of the data.
  - The hash value of the key is mapped to a node in the cluster
  - The first copy of the data is stored on that node.
  - The distribution is transparent as you can both calculate the hash value and determine where a particular row will be stored.

# Partitioners

- A partitioner determines how data is distributed across the nodes in the cluster.
- A partition key is used for data partitioning. Each row has a partition key that is used to identify the partition.
- Three types of partitioners are available in Cassandra:
  - **Murmur3Partitioner** – the default partitioner in Cassandra.
  - **RandomPartitioner** – similar to the Murmur3Partitioner except that it uses the MD5 (message-digest version 5) hash function to calculate the hash value.
  - **ByteOrderedPartitioner** – orders rows using partition key values.

# Replications

- Replication refers to the number of replicas that are maintained for each row.
- Replication provides redundancy of data for fault tolerance.
- A replication factor of  $N$  means that  $N$  copies of data are maintained in the system.
- Cassandra allows replication based on nodes, racks, and data centers.
- Replication across data centers guarantees data availability even when a data center is down.

# Replication strategies

- The replication strategy determines how multiple replicas of a data row are maintained.
- Replication is specified at the keyspace level, and different keyspaces can have different strategies.
- Cassandra replicates data across nodes in a manner transparent to the user.
- Replication factor is the number of nodes in your cluster that will receive copies of the same data.
- Cassandra provides two common replication strategies:
  - **SimpleStrategy**: places replicas at consecutive nodes around the ring.
  - **NetworkTopologyStrategy**: specify a different replication factor for each data center, allocates replicas to different racks in order to maximize availability.

# Snitches

- A snitch determines which data centers and racks nodes belong to.
- The job of a snitch is to determine relative host proximity for each node in a cluster.
- The snitch will figure out where nodes are in relation to other nodes.
- Two types of most popular snitches:
  - Simple Snitch - A simple snitch is used for single data centers with no racks.
  - Property File Snitch - A property file snitch is used for multiple data centers with multiple racks.
- Replication in Cassandra is based on the snitches.

# Failure Detection

- The effects of Node Failure:
  - Other nodes detect the node failure.
  - Request for data on that node is routed to other nodes that have the replica of that data.
  - Writes are handled by a temporary node until the node is restarted.
  - Any memtable or sstable data that is lost is recovered from commitlog.
  - A node can be permanently removed using the `nodetool` utility.

# Failure Detection

- The effects of Disk Failure:
  - The data on the disk becomes inaccessible.
  - Read of data from the node is not possible.
  - This issue will be treated as node failure for that portion of data.
  - Memtable and sstable will not be affected as they are in-memory tables.
  - Commitlog has replicas and they will be used for recovery.

# Failure Detection

- The effects of Rack Failure:
  - All the nodes on the rack become inaccessible.
  - Read of data from the rack nodes is not possible.
  - The reads will be routed to other replicas of the data.
  - This will be treated as if each node in the rack has failed.

# Failure Detection

- The effects of Data Center Failure:
  - All data in the data center will become inaccessible.
  - All reads have to be routed to other data centers.
  - The replica copies in other data centers will be used.
  - Though the system will be operational, clients may notice slowdown due to network latency. This is because multiple data centers are normally located at physically different locations and connected by a wide area network.

# Consistency levels

- Cassandra provides tunable consistency levels.
  - Can specify a consistency level on each read or write query.
    - For read queries, the consistency level specifies how many replica nodes must respond to a read request.
    - For write operations, the consistency level specifies how many replica nodes must respond for the write to be reported.
- 
- $R + W > N$  = strong consistency
  - R: read replica count W: write replica count N: replication factor

# Cassandra Query Language (CQL)

# CQL – Cassandra Query Language

- A simple way to manipulate the data you have stored.
- Syntax is similar with SQL.
- CQL has no concept of GROUP or JOIN, and very limited implementation of ORDER BY.

# Data Types – Native Types

Type	Constants supported	Description
text	string	UTF-8 encoded string
varchar	string	UTF-8 encoded string
int	integer	32-bit signed int
date	integer, string	A date with no corresponding time value
time	integer, string	A time with no corresponding date value
timestamp	integer, string	A timestamp with millisecond precision
uuid	uuid	A universally unique identifier with 128-bit number

More data types:

<http://cassandra.apache.org/doc/latest/cql/types.html>

# Create Keyspace

- CREATE KEYSPACE [IF NOT EXISTS] `keyspace_name`  
WITH REPLICATION = {`replication_map`}  
[AND DURABLE\_WRITES = true|false] ;
- Two options have to be specified:
  - ***replication***: This is a mandatory option that specifies the replica placement strategy and the number of replicas. The `replication_map` attribute allows you to specify the number of copies of the data in a datacenter. The replication option must contain the sub-option 'class', which specifies the replication strategy. A replication strategy tells Cassandra where it should store copies of the data in this keyspace across the cluster's datacenters and racks.
  - ***durable\_writes***: Tells Cassandra whether it ought to use the commit log for any updates in the current keyspace. The default value is true and it should be used in production databases.

# Create Keyspace

- CREATE KEYSPACE `ks_name`  
WITH replication = {'class': 'SimpleStrategy',  
'replication\_factor' : 3};
- CREATE KEYSPACE `ks_name`  
WITH replication = {'class': 'NetworkTopologyStrategy',  
'datacenter1' : 1, 'datacenter2' : 3}  
AND durable\_writes = false:

Name	Mandatory	Default	Description
replication	yes		Replication strategy and options for the keyspace
durable_writes	no	true	Whether to use the commit log for updates on this keyspace (disable)

# Create Keyspace

- SimpleStrategy
  - A simple strategy that defines a replication factor for data to be spread across the entire cluster.
  - `replication_factor` (mandatory argument) : the number of replicas to store per range.
- NetworkTopologyStrategy
  - A production ready replication strategy that allows to set the replication factor independently for each data-center.
  - `<datacenter>`: The number of replicas to store per range in the provided datacenter.
  - `replication_factor`: The number of replicas to use as a default per datacenter if not specifically provided. Note that this always defers to existing definitions or explicit datacenter settings. For example, to have three replicas per datacenter, supply this with a value of 3.

# Alter & Drop Keyspace

- Alter the existing keyspace:

- ```
ALTER KEYSPACE ks_name
    WITH replication = { 'class': 'SimpleStrategy',
        'replication_factor' : 4 };
```

- Drop a keyspace:

- ```
DROP KEYSPACE ks_name;
```

# Create Table

- To create a table:
  - `CREATE TABLE user ( first_name text,  
last_name text,  
PRIMARY KEY (first_name)  
);`
- To get the description of the newly created table:
  - `DESCRIBE TABLE user;`

# Write & Read

- To write a value (INSERT command):

- ```
INSERT INTO user (first_name, last_name )
VALUES ('Bill', 'Nguyen');
```

- To read (SELECT command):

- ```
SELECT * FROM user WHERE first_name='Bill';
```

- To count (SELECT COUNT command):

- ```
SELECT COUNT (*) FROM user;
```

# Update

- Update a column:

- ```
UPDATE user
    SET last_name = 'Smith'
  WHERE first_name = 'Bill';
```

# Alter Table

- Alter table to add a new column:
  - `ALTER TABLE user ADD title text;`
- To get the description of the altered table:
  - `DESCRIBE TABLE user;`

# Timestamps

- To view the timestamps that were generated for previous writes:
  - `SELECT first_name, last_name,  
writetime(last_name) FROM user;`
- Note: We are not allowed to ask for the timestamp on primary key columns.
- To specify a timestamp when performing writes:
  - `UPDATE user USING TIMESTAMP 1434373756626000  
SET last_name = 'Boateng' WHERE first_name = 'Mary' ;`
- Note: The timestamp must be later than the one from our SELECT command, or the UPDATE will be ignored.

# Time to live (TTL)

- The time to live (or TTL) is a value that Cassandra stores for each column value to indicate how long to keep the value.
- The TTL value defaults to null, meaning that data that is written will not expire.
- To show TTL:
  - ```
SELECT first_name, last_name, TTL(last_name)
  FROM user WHERE first_name = 'Mary';
```
- Adding TTL:
  - ```
UPDATE user USING TTL 3600 SET last_name =
  'McDonald' WHERE first_name = 'Mary' ;
```

# Collections

- Collection columns are used to represent a group of data in a single column.
- CQL supports 3 kind of collections:
  - Set – stores a collection of unordered elements
  - List – stores an ordered list of elements
  - Map – stores a collection of key/value pairs

# Collections – Set

- A set is an unordered group of values.
  - Returns query in alphabetical order.
  - E.g. { 'XYZ' , 'ABC' , 'PQR' }
- EXAMPLE:
  - ALTER TABLE user ADD emails **set<text>**;
  - UPDATE user SET emails = { 'mary@example.com' }  
WHERE first\_name = 'Mary';
  - UPDATE user SET emails = emails +  
{ 'mary.mcdonald.AZ@gmail.com' } WHERE first\_name = 'Mary';

# Collections – List

- A list is a sorted collection of non-unique values where elements are ordered by their position in the list.
  - E.g. [ '2011', '2012', '2013' ]
- EXAMPLE:
  - ALTER TABLE user ADD phone\_numbers **list<text>**;
  - UPDATE user SET phone\_numbers = [ '1-800-999-9999' ] WHERE first\_name = 'Mary';

# Collections – List

- A list is a sorted collection of non-unique values where elements are ordered by their position in the list
- EXAMPLE:
  - UPDATE user SET phone\_numbers = phone\_numbers + [ '480-111-1111' ] WHERE first\_name = 'Mary';
  - UPDATE user SET phone\_numbers[1] = '480-111-1111' WHERE first\_name = 'Mary';
  - UPDATE user SET phone\_numbers = phone\_numbers - [ '480-111-1111' ] WHERE first\_name = 'Mary';
  - DELETE phone\_numbers[0] from user WHERE first\_name = 'Mary';

# Collections – Map

- A map is a sorted set of key-value pairs, where keys are unique and the map is sorted by its keys.
  - Each element is internally stored as one Cassandra column
  - Each element can have an individual time-to-live
  - E.g. { 'key1': 'value1', 'key2': 'value2' }
- EXAMPLE:
  - ALTER TABLE user ADD login\_sessions **map<timeuuid, int>**;
  - UPDATE user SET login\_sessions = { now(): 13, now(): 18 } WHERE first\_name = 'Mary';

# Data Types – User-Defined Types (UDT)

- User-defined types (UDTs) can attach multiple data fields, each named and typed, to a single column.
- Cassandra allows nesting of collections, however we have to use `frozen<type>`.
- Freezing is a concept that the Cassandra community has introduced as a forward compatibility mechanism. For now, we can nest a collection within another collection by marking it as frozen.

# Data Types – User-Defined Types (UDT)

- EXAMPLE:

- CREATE TYPE address (
  - street text,
  - city text,
  - state text,
  - zip\_code int);
- DESCRIBE KEYSPACE my\_keyspace;
- ALTER TABLE user ADD addresses map<text, **frozen<address>**>;
- UPDATE user SET addresses = addresses + { 'home': { street: '7712 E. Broadway', city: 'Tucson', state: 'AZ', zip\_code: 85715} } WHERE first\_name = 'Mary';

# Secondary Indexes

- Cassandra does not allow you to query on column in a table that is not part of the primary key.
- Example:

```
cqlsh:my_keyspace> SELECT * FROM user WHERE last_name = 'Nguyen';
InvalidRequest: Error from server: code=2200 [Invalid query] message="Cannot execute this query as it might involve data
filtering and thus may have unpredictable performance. If you want to execute this query despite the performance unpredictability, use ALLOW FILTERING"
```

- To avoid the previous problem, we need to create a secondary index for the `last_name` column.
- A secondary index is an index on a column that is not part of the primary key.
- Solution:
  - `CREATE INDEX ON user ( last_name );`

# Secondary Indexes

- To create secondary index:

- CREATE INDEX ON user ( last\_name );

- To remove an index:

- DROP INDEX user\_last\_name\_idx;

# References

- [1] C. Carlos and M. Steven, Database systems: *Design, implementation, & management*. Cengage Learning, 12<sup>th</sup> ed., 2016.
- [2] C. Y. Kan, Cassandra Data Modeling and Analysis. Birmingham, UK: Packt Publishing, 1<sup>st</sup> ed., 2014.
- [3] D. Sullivan, NoSQL for Mere Mortals. Michigan, USA: Addison-Wesley Professional, 1<sup>st</sup> ed., 2015.
- [4] J. Carpenter and E. Hewitt, Cassandra: The Definitive Guide. Sebastopol, CA: O'Reilly Media, Inc., 2<sup>nd</sup> ed., 2016.
- [5] N. Neeraj, Mastering Apache Cassandra. Birmingham, UK: Packt Publishing, 1<sup>st</sup> ed., 2013.
- [6] R. Strickland, Cassandra 3.x High Availability. Birmingham, UK: Packt Publishing, 2<sup>nd</sup> ed., 2016.
- [7] S. Alapati, Expert Apache Cassandra Administration. Texas, USA: Apress, 1<sup>st</sup> ed., 2018.
- [8] Simplilearn, Apache Cassandra Advanced Architecture Tutorial. <https://www.simplilearn.com/cassandra-advanced-architecture-tutorial-video>.
- [9] Simplilearn, Apache Cassandra Architecture Tutorial. <https://www.simplilearn.com/cassandra-architecture-tutorial-video>.
- [10] W. Lemahieu, S. V. Broucke and B. Baesens, Principles of Database Management: *The Practical Guide to Storing, Managing and Analyzing Big and Small Data*. New York, NY, USA: Cambridge University Press, 1<sup>st</sup> ed., 2018.

# Tutorial 5

## Introduction to Apache Cassandra

# SOLUTIONS

The main objective of this tutorial is to get students to be more familiar with Apache Cassandra. Students are expected to be able to operate Apache Cassandra and performing simple create, read and delete operations.

# Connecting to Cassandra on Windows

1. Press the **Windows Key** → type **Run** in the search bar → **Enter**.
  2. Type **cmd** to open the **Command Prompt** → **Enter**.
  3. Go to **apache-cassandra-3.11.7\bin** and copy the directory.
  4. Type “**cd [directory]**”.
  5. Type **Cassandra** to start Cassandra server. The Command Prompt will load Cassandra. Wait until it stops loading.

```
Command Prompt
Microsoft Windows [Version 10.0.18362.295]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\agnes>Cassandra
Detected powershell execution permissions. Running with enhanced startup scripts.
*-----*
*-----*
WARNING! Automatic page file configuration detected.
It is recommended that you disable swap when running Cassandra
for performance and stability reasons.

*-----*
*-----*
*-----*
*-----*

WARNING! Detected a power profile other than High Performance.
Performance of this node will suffer.
Modify conf\cassandra.env.ps1 to suppress this warning.

*-----*
*-----*

C:\Users\agnes>CompilerOracle: dontinline org/apache/cassandra/db/Columns$Serializer.deserializeLargeSubset (Lorg/apache
/cassandra/io/util/DataInputPlus;Lorg/apache/cassandra/db/Columns;I)Lorg/apache/cassandra/db/Columns;
CompilerOracle: dontinline org/apache/cassandra/db/Columns$Serializer.serializeLargeSubset (Ljava/util/Collection;ILorg/
apache/cassandra/db/Columns;ILorg/apache/cassandra/io/util/DataOutputPlus;)V
CompilerOracle: dontinline org/apache/cassandra/db/Columns$Serializer.serializeLargeSubsetSize (Ljava/util/Collection;IL
org/apache/cassandra/db/Columns;I)
```

6. Open another Command Prompt with the same Cassandra directory, then type **cqlsh**. The figure below indicates that you have successfully load Cassandra.

```
c:\ Command Prompt - cqlsh
Microsoft Windows [Version 10.0.18362.239]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\agnes>cqlsh

WARNING: console codepage must be set to cp65001 to support utf-8 encoding on Windows platforms.
If you experience encoding problems, change your console codepage with 'chcp 65001' before starting cqlsh.

Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.4 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
WARNING: pyreadline dependency missing. Install to enable tab completion.
cqlsh>
```

## Connecting to Cassandra on Mac

1. Start Cassandra.

```
$ brew services start cassandra
```

```
(base) Huashuns-MacBook-Pro:~ huashun$ brew services start cassandra
==> Successfully started `cassandra` (label: homebrew.mxcl.cassandra)
```

2. Go to /usr/local/Cellar/cassandra/3.11.7/bin.

```
$ cd /usr/local/Cellar/cassandra/3.11.7/bin
```

3. Initialize cassandra

```
$ cassandra
```

4. Open another terminal, then start CQL

```
$ cqlsh
```

```
-- 
(base) Huashuns-MacBook-Pro:~ huashun$ cqlsh
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.4 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
cqlsh>
```

## Cassandra Basic

1. List all available keyspaces.

```
DESCRIBE KEYSPACES;
```

2. Create a new keyspace called `my_first_keyspace` with `SimpleStrategy` class and `replication_factor` of one.

```
CREATE KEYSPACE my_first_keyspace
WITH replication = {'class':'SimpleStrategy',
'replication_factor':1};
```

3. Check the newly created keyspace.

```
DESCRIBE KEYSPACE my_first_keyspace;
```

4. Switch to `my_first_keyspace`.

```
USE my_first_keyspace;
```

5. Create a new table called `student` with four columns that store student ID, first name, last name, and age. The type for student ID and age are int, while the type for first name and last name are text. The student ID should be set as the primary key for this table.

```
CREATE TABLE student ( student_id int,
                      first_name text,
                      last_name text,
                      age int,
                      PRIMARY KEY (student_id) );
```

6. Describe the newly created table.

```
DESCRIBE TABLE student;
```

7. Insert some values to the `student` table as follow.

<b>Student ID</b>	<b>First name</b>	<b>Last name</b>	<b>Age</b>
1001	John	Smith	25
1002	Mary	Citizen	33
1003	Wendy	Wheat	22

```
INSERT INTO student (student_id, first_name, last_name,
age)
```

```
VALUES (1001, 'John', 'Smith', 25);

INSERT INTO student (student_id, first_name, last_name,
age)
VALUES (1002, 'Mary', 'Citizen', 33);

INSERT INTO student (student_id, first_name, last_name,
age)
VALUES (1003, 'Wendy', 'Wheat', 22);
```

8. Using the SELECT command, read all the newly inserted data.

```
SELECT * FROM student;
```

9. Count how many rows have been inserted into the table.

```
SELECT COUNT (*) FROM student;
```

10. Find a student whose ID is 1002.

```
SELECT * FROM student WHERE student_id=1002;
```

11. Delete the last name of student number 1001.

```
DELETE last_name FROM student WHERE student_id=1001;
```

12. Delete the student record with student ID of 1003.

```
DELETE FROM student WHERE student_id=1003;
```

**The End**

# Tutorial 5

## Column-Oriented Databases

# SOLUTIONS

Discuss the following questions:

1. What is Column-Oriented DBMS?

Column-Oriented DBMS is a DBMS that stores data tables as sections of columns of data, rather than as rows of data as in most DBMS implementations.

2. What are the three components of data values in Column-Oriented Databases?

Data values are indexed by row identifier, column name, and time stamp.

3. Why are time stamps used in Column-Oriented Databases?

The time stamp orders versions of the column value and it allows applications to determine the latest version of a column value.

4. Identify one similarity between column-oriented databases and document databases.

Column-oriented and document databases support similar types of querying that allow users to select subsets of data available in a row.

Column-oriented databases, like document databases, do not require all columns in all rows. In both column-oriented and document databases, columns or fields can be added as needed by developers.

5. Identify one similarity between column-oriented databases and relational databases.

Both column-oriented databases and relational databases use unique identifiers for rows of data. These are known as row keys in column-oriented databases and as primary keys in relational databases. Both row keys and primary keys are indexed for rapid retrieval.

6. Describe the essential characteristics of a peer-to-peer architecture.

Peer-to-peer architectures have only one type of node. Any node can assume responsibility for any service or task that must be run in the cluster.

7. Why does Cassandra use a gossip protocol to exchange server status information?

An “all-servers-to-all-other-servers” protocol can quickly increase the volume of traffic on the network and the amount of time each server has to dedicate to communicating with other servers. The number of messages sent is a function of the number of servers in the cluster. If  $N$  is the number of servers, then  $N \times (N-1)$  is the number of messages needed to

update all servers with information about all other servers. Gossip protocols are more efficient because one server can update another server about itself as well as all the servers it knows about.

8. When would you use a column-oriented database instead of another type of NoSQL database?

Column-oriented databases are appropriate choices for large-scale database deployments that require high levels of write performance, a large number of servers, or multi-data center availability.

Column-oriented databases are also appropriate when a large number of servers are required to meet expected workloads.

9. How do columns in column-oriented databases differ from columns in relational databases?

Columns in column-oriented are dynamic. Columns in a relational database table are not as dynamic as in column-oriented databases. Adding a column in a relational database requires changing its schema definition. Adding a column in a column-oriented database just requires making a reference to it from a client application, for example, inserting a value to a column name.

10. When should columns be grouped together in a column family? When should they be in separate column families?

Columns that are frequently used together should be grouped in the same column family.

**The End**



MONASH  
University

MONASH  
INFORMATION  
TECHNOLOGY

# FIT5137 – Advanced Database Technology

## Week 5 – Introduction to Column-Oriented Database

Semester 2, 2020

Developed by:

Dr. Agnes Haryanto

[Agnes.Haryanto@monash.edu](mailto:Agnes.Haryanto@monash.edu)

Huashun Li

[Huashun.Li@monash.edu](mailto:Huashun.Li@monash.edu)

# Using FLUX

1. Visit <http://flux.qa/> on your internet enabled device
2. Log in using your Monash account (not required if you are already logged in to Monash)
3. Click on the “+” to join audience
4. Enter the Audience Code: **AL2WZE**
5. Select FIT5137 in the Active Presentation menu
6. Answer questions when they pop up

Alternatively, the quiz can be accessed through:

<https://flux.qa/AL2WZE>

The screenshot shows a web browser window titled "FLUX" with the URL <https://flux.qa/#/feeds/5d368...>. The page is titled "Lecture 0". Below the title are four icons: a bar chart, a person icon, a gift icon, and a clock icon. The main content area is titled "Current Polls". It displays a multiple-choice poll with the question "What did you have for dinner?". The options are listed in a vertical list:

- A Pasta
- B Rice
- C Pizza
- D Salad
- E Nothing

# Agenda

1. Introduction to Column-Oriented Database

2. Introduction to Cassandra

- Cassandra Architecture
- CQL Shell (CQLSH)

# Recall from week 1...

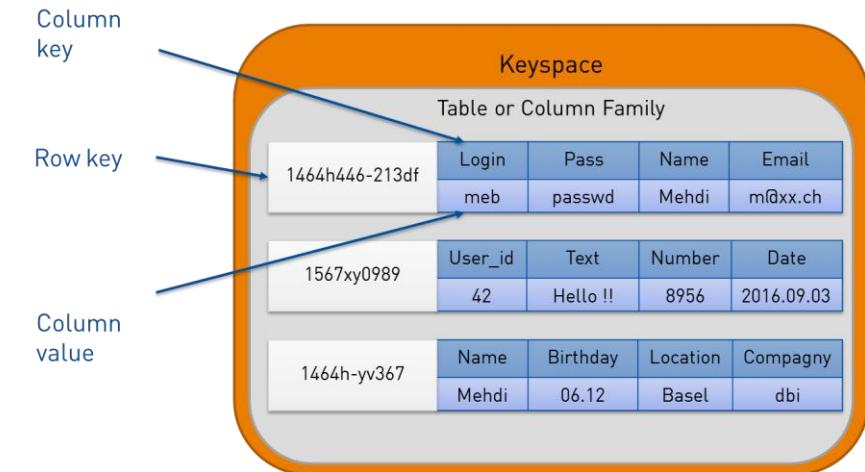
## NoSQL Databases

NoSQL Category	Example Databases
Key-value databases	Dynamo, Riak, Redis, Voldemort, ArangoDB, Aerospike, Apache Ignite
Document-oriented databases	MongoDB, CouchDB, OrientDB, RavenDB, ArangoDB, BaseX, Clusterpoint, Couchbase, Cosmos DB, RethinkDB, MarkLogic
Column-oriented databases	Cassandra, Hbase, Hypertable, Accumulo, Scylla
Graph databases	Neo4j, ArangoDB, GraphBase, AllegroGraph, InfiniteGraph, Apache Giraph, MarkLogic, OrientDB, Virtuoso

# Recall from week 1...

## Column-Oriented Database

- Originated from Google's BigTable product, HBase, and Cassandra.
- Also known as *column family database*.
- Organize data in **key-value pairs** with keys mapped to a set of columns in the value component.
  - **Column**: a key-value pair similar to a cell of data in the relational database
  - **Key**: name of the column
  - **Value**: data that is stored in the column
- A group of columns that are logically related are called a *super column*. All of the super columns are grouped together to create a *column family*. A column family is conceptually similar to a table in the relational model.
- Each row key in the column family can have different columns.



# Column-Oriented Databases

- A **column-oriented DBMS** is a DBMS that stores data tables as sections of columns of data, rather than as rows of data as in most DBMS implementations.
- Column with many null values, known as sparse data, can be dealt with more efficiently, without wasting storage capacity for the empty cells.

# Relational Databases

- Relational databases can be both row and column oriented.
- A row-based system is designed to efficiently return data for an entire row, which matches the common use case in which users wish to retrieve information about a particular object or entity.

<b>Id</b>	<b>Genre</b>	<b>Title</b>	<b>Price</b>	<b>Audiobook price</b>
1	Fantasy	My first book	20	30
2	Education	Beginners guide	10	null
3	Education	SQL strikes back	40	null
4	Fantasy	The rise of SQL	10	null

# Relational Databases

- Most database systems speed up searching operations by means of database indexes.

Id	Genre	Title	Price	Audiobook price
1	Fantasy	My first book	20	30
2	Education	Beginners guide	10	null
3	Education	SQL strikes back	40	null
4	Fantasy	The rise of SQL	10	null

Index based on price value:

Price value	Record identifiers
10	2,4
20	1
40	3

# Column-Oriented Databases

- In a column-oriented database, all values of a column are placed together on the disk, so the previous example table would be stored in this way:

Genre:	Fantasy: 1,4	Education: 2,3
Title:	My first book: 1	Beginners guide: 2
Price:	20: 1	10: 2,4
Audiobook price:	30: 1	40: 3

# Column-Oriented Databases

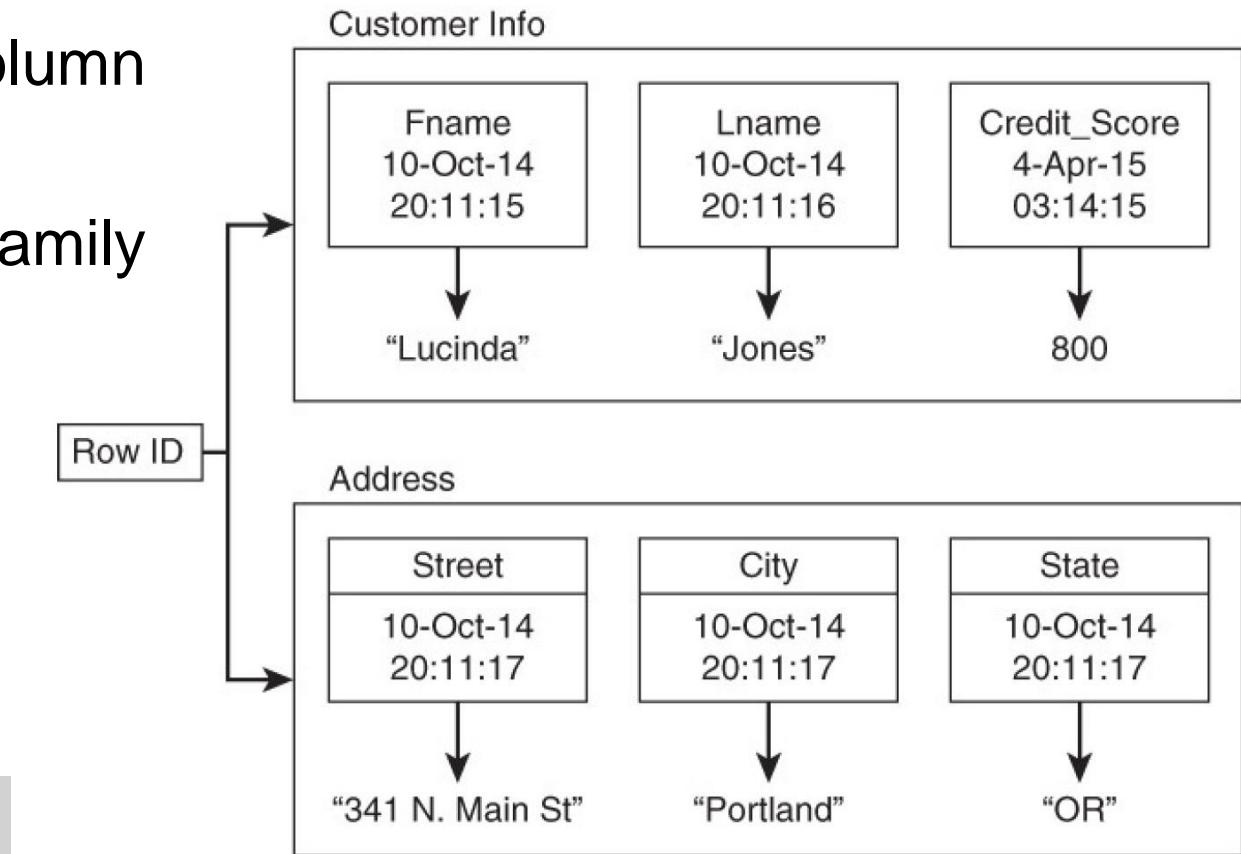
Advantages	Disadvantages
Speed up query processing and aggregation operations.	Retrieving all attributes pertaining to a single entity becomes less efficient
Efficient storage space consumption (great in dealing sparse data).	Join operations will be slowed down considerably (every column must be scanned in order to find values belonging to a certain foreign record identifier).

# In the Beginning, There Was Google BigTable

- The following are core features of Google BigTable:
  - Developers have dynamic control over columns.
  - Data values are indexed by row identifier, column name, and a time stamp.
  - Data modelers and developers have control over location of data.
  - Reads and writes of a row are atomic.
  - Rows are maintained in a sorted order.

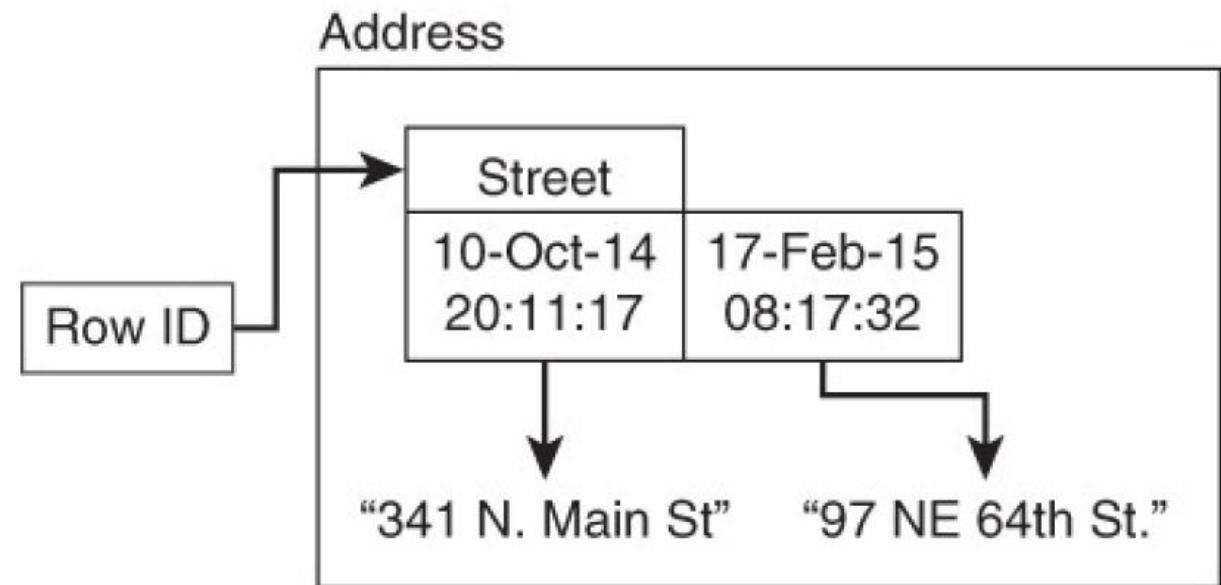
# Column-Oriented Databases

- Rows are composed of several column families. Each family consists of a set of related columns.
- A data value is indexed by a row, a column name, and a time stamp
- For example an address column family might contain:
  - Street address
  - City
  - State or province
  - Postal code
  - Country

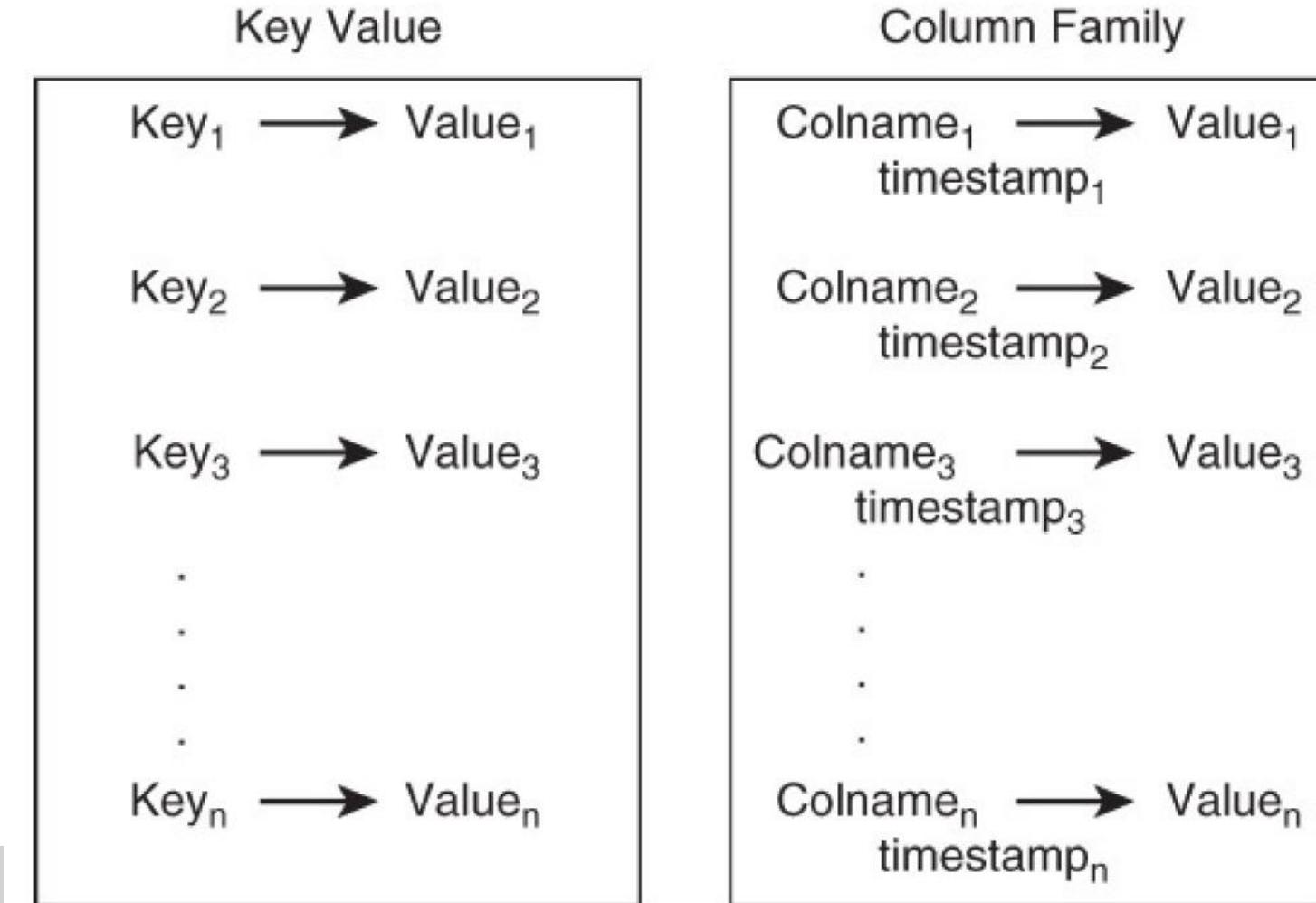


# Column-Oriented Databases

- Data values are indexed by row identifier, column name, and time stamp.
  - The row identifier is analogous to a primary key in a relational database.
- Multiple versions of a column value can exist. The latest version is returned by default when the column value is queried.



# Column-Oriented vs. Key-Value Databases



# Column-Oriented vs. Document-Oriented

## Document Database:

```
{fname: 'Lucinda',  
 lname: 'Jones',  
 Credit_Rating: 800,
```

### Address:

```
{Street: '341 N. Main St.',  
 City: 'Portland',  
 State: 'OR'  
 }
```

```
{fname: 'Frank',  
 lname: 'Antonio',  
 Credit_Rating: 768  
 }
```

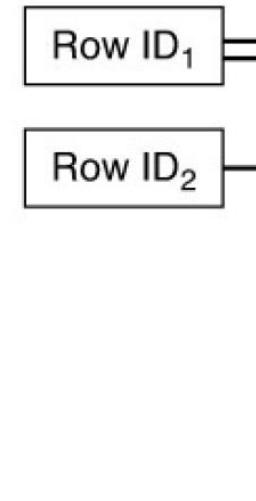
## Column Family Database:

### Customer\_Info

fname	Iname	Credit_Rating
10-Oct-14 20:11:15	10-Oct-14 20:11:16	04-Apr-15 03:14:15
“Lucinda”	“Jones”	800
fname	Iname	Credit_Rating
24-May-13 07:01:01	24-May-13 07:01:01	24-May-13 07:01:02
‘Frank’	‘Antonio’	768

### Address

Street	City	State
10-Oct-14 20:11:15	10-Oct-14 20:11:16	10-Oct-14 20:11:17
“341 N. Main St.”	“Portland”	‘OR’



# Apache Cassandra

# Apache Cassandra

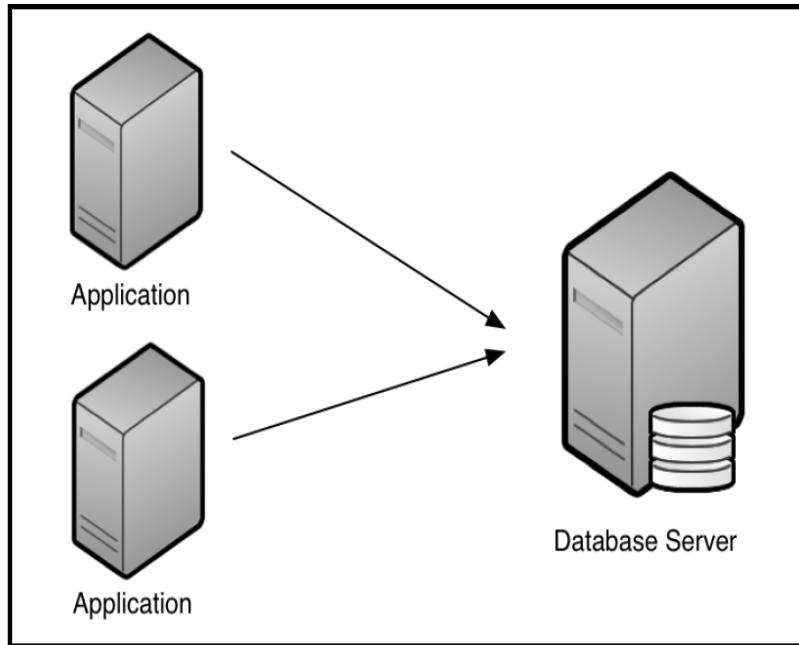
- A column-oriented database.
- A free, open source, distributed data storage system that differs sharply from relational database management systems (RDBMSs).
- Originated at Facebook in 2008 to solve its inbox search problem.
- A database designed for high availability, scalability, and consistency.



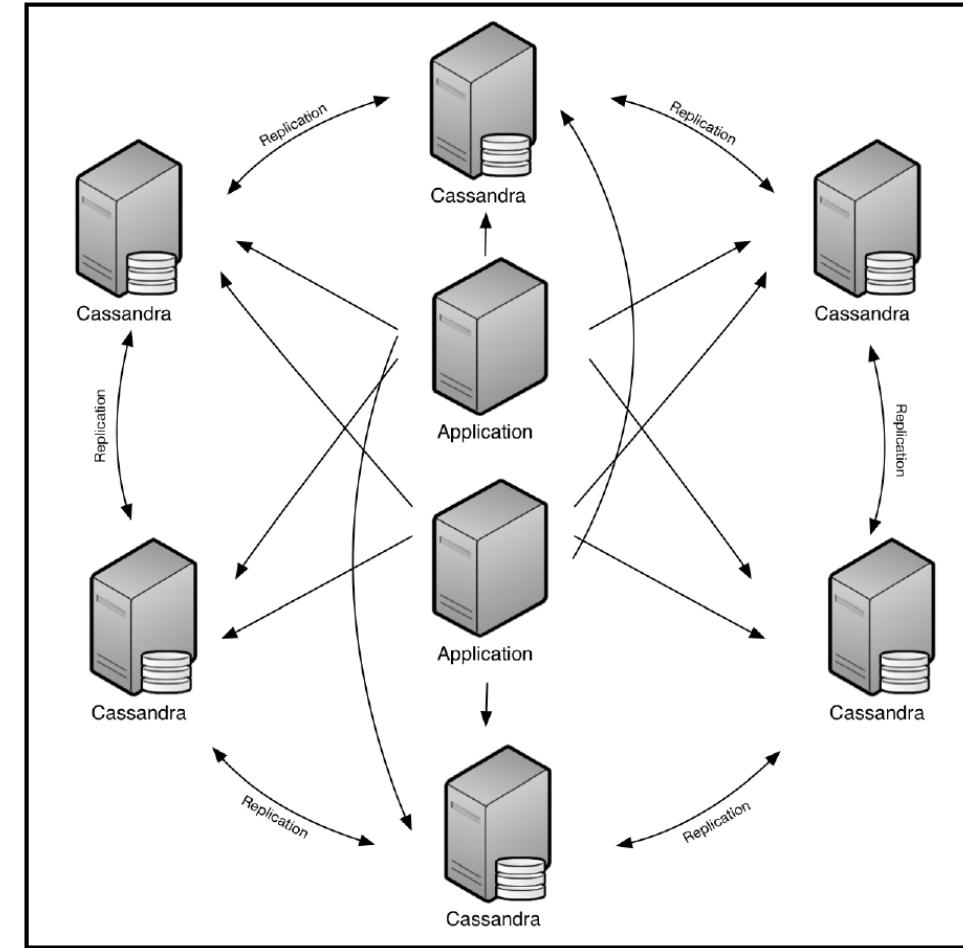
# Apache Cassandra Features

- Distributed
  - Databases can be spread across multiple servers.
- Decentralized
  - Every node in the cluster is identical.
- Elastically scalable
  - Read and write throughput both increase linearly as new machines are added, with no downtime or interruption to application.
- Highly available & Fault-tolerant
  - Data is automatically replicated to multiple nodes, across multiple data centers.

# Cassandra is a distributed database

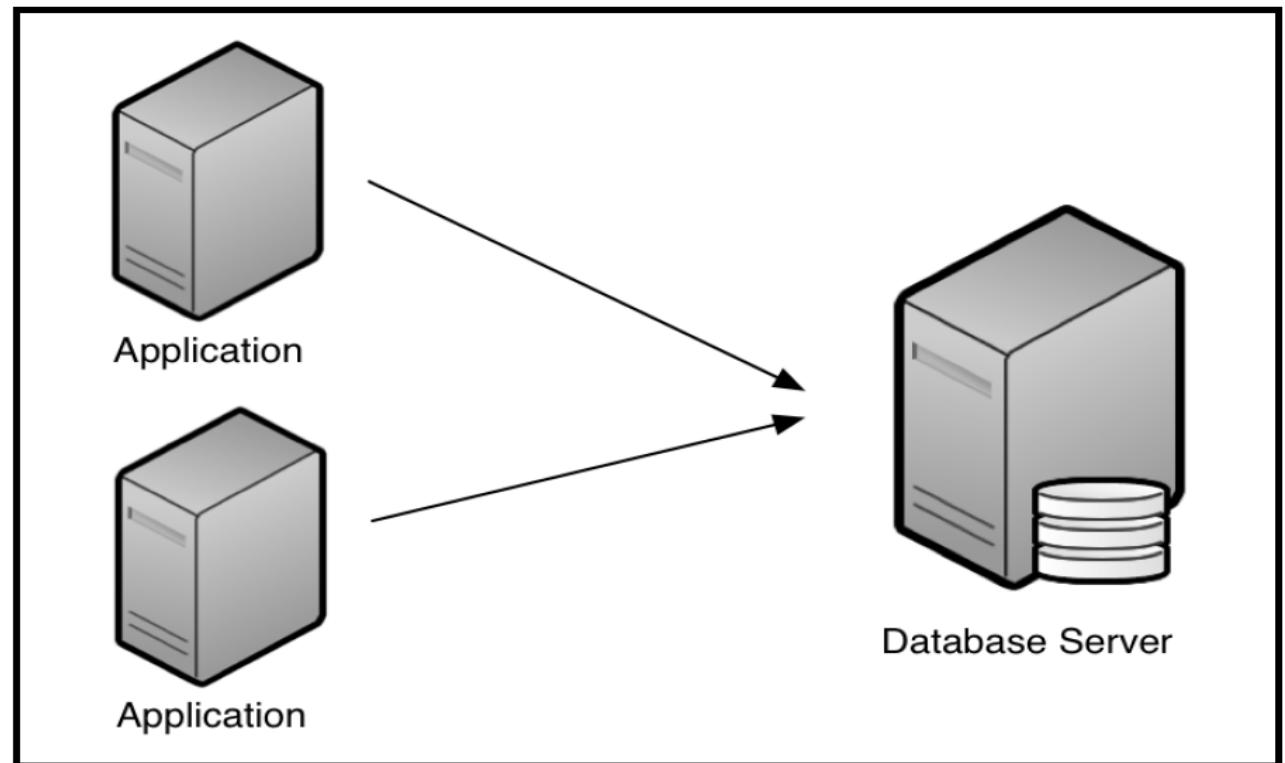


Monolithic Architecture

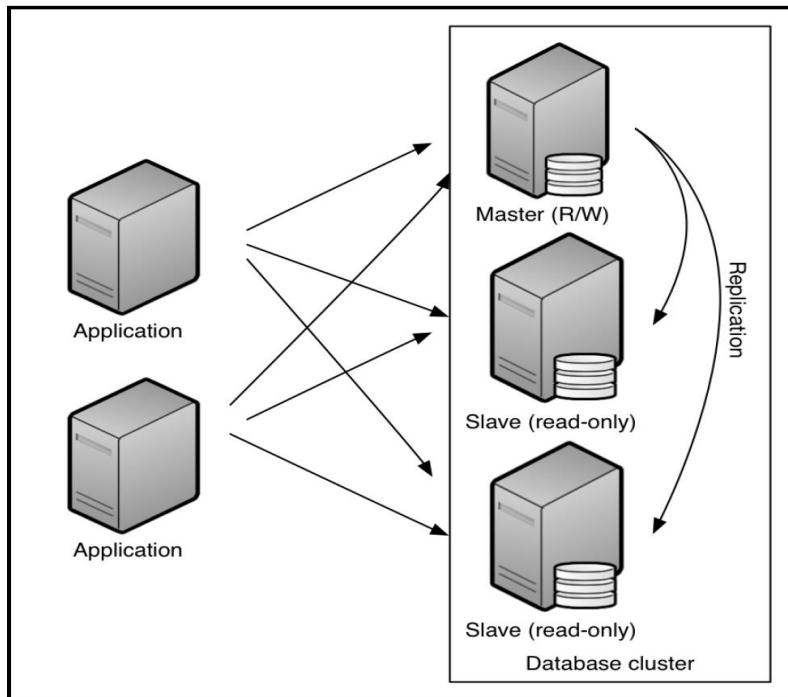


# Monolithic Architecture

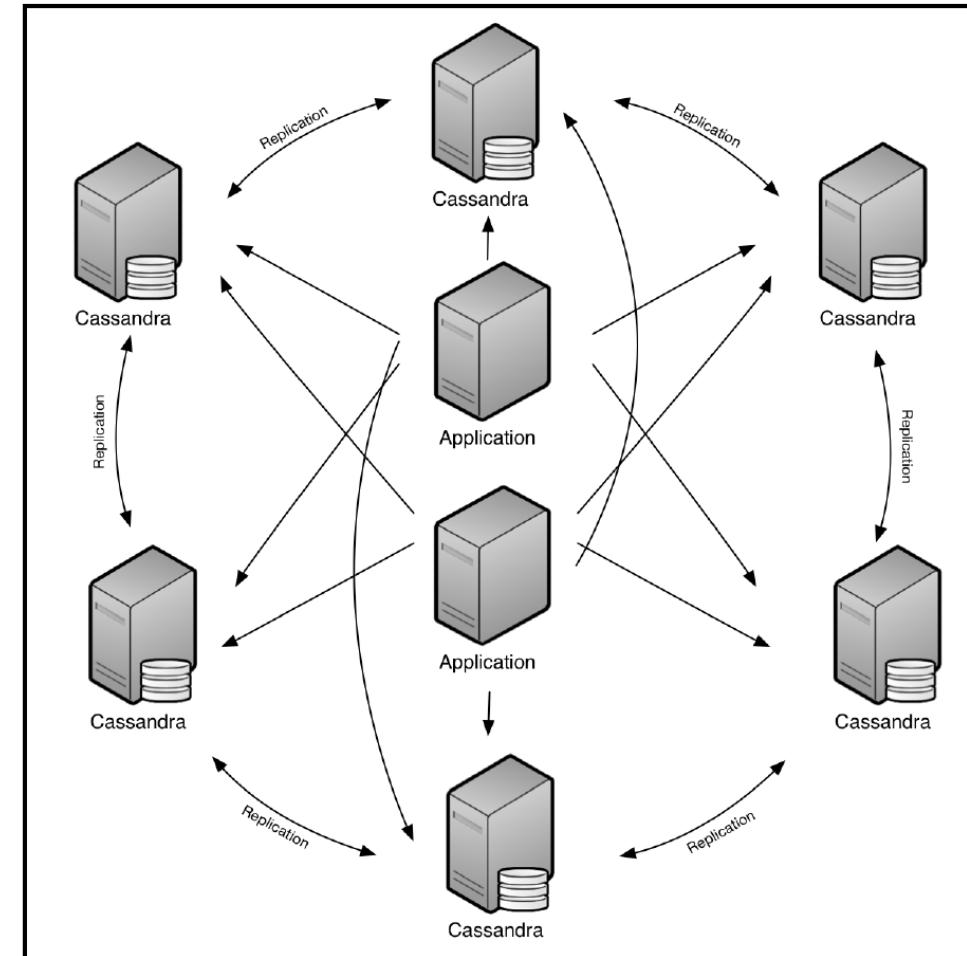
- All functions reside on a single machine
- Problems: single point of failure



# Cassandra is a decentralized database

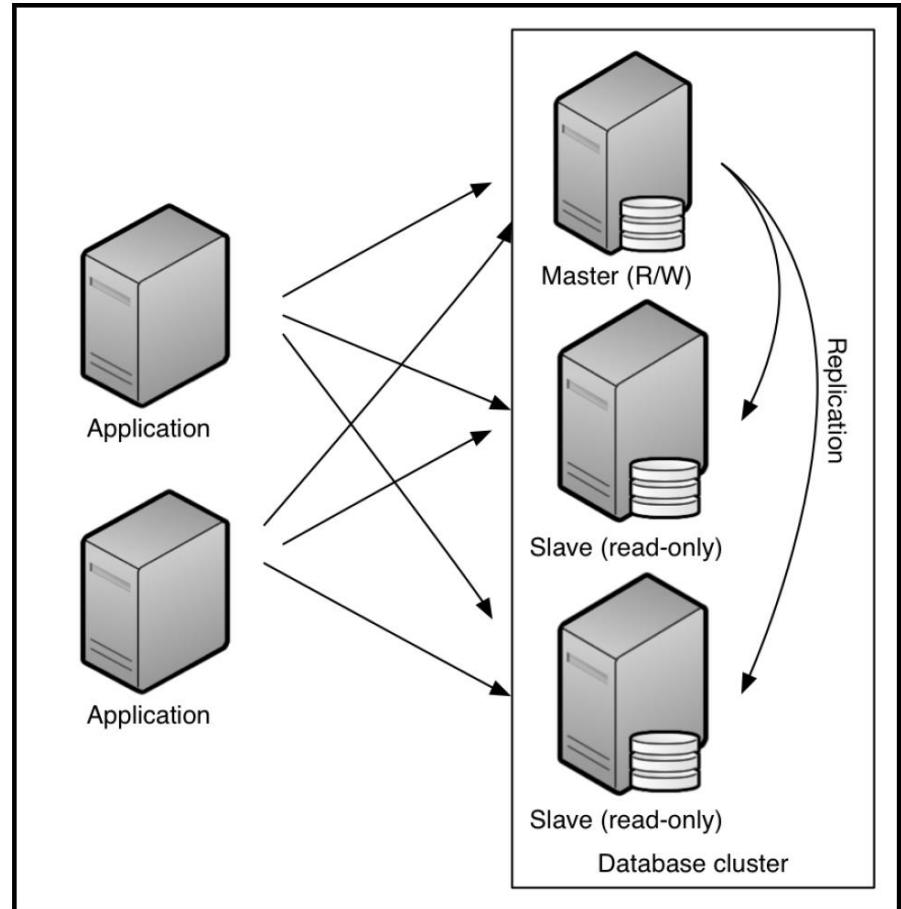


Master-Slave Model



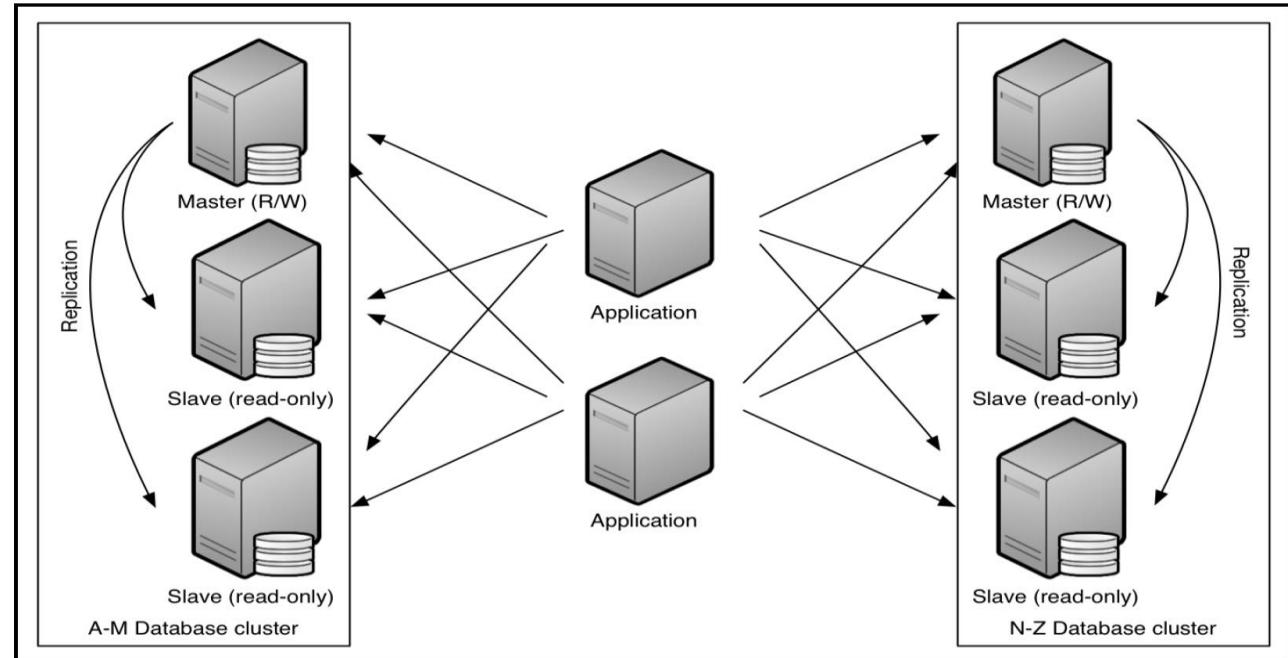
# The Master-slave model

- Many distributed databases still attempt to maintain ACID guarantees, leading to the master-slave model.
- For data consistency aspect:
  - Many servers handling requests, but only master server can perform writes
- Problem: a failure of the write master



# Using sharding to scale writes

- Sharding
  - an approach that enables higher write volumes
  - the data is partitioned into groups of keys
  - One or more masters can own a known set of keys
- Problems:
  - Requires manual shuffling of data
  - Failure points of the master nodes



# Handling the death of the leader

- Solution to increase availability in a master-slave system.
- Employ a master failover protocol.
- The protocol vary among implementations.
- Principle is that a new master is appointed when the previous one fails.

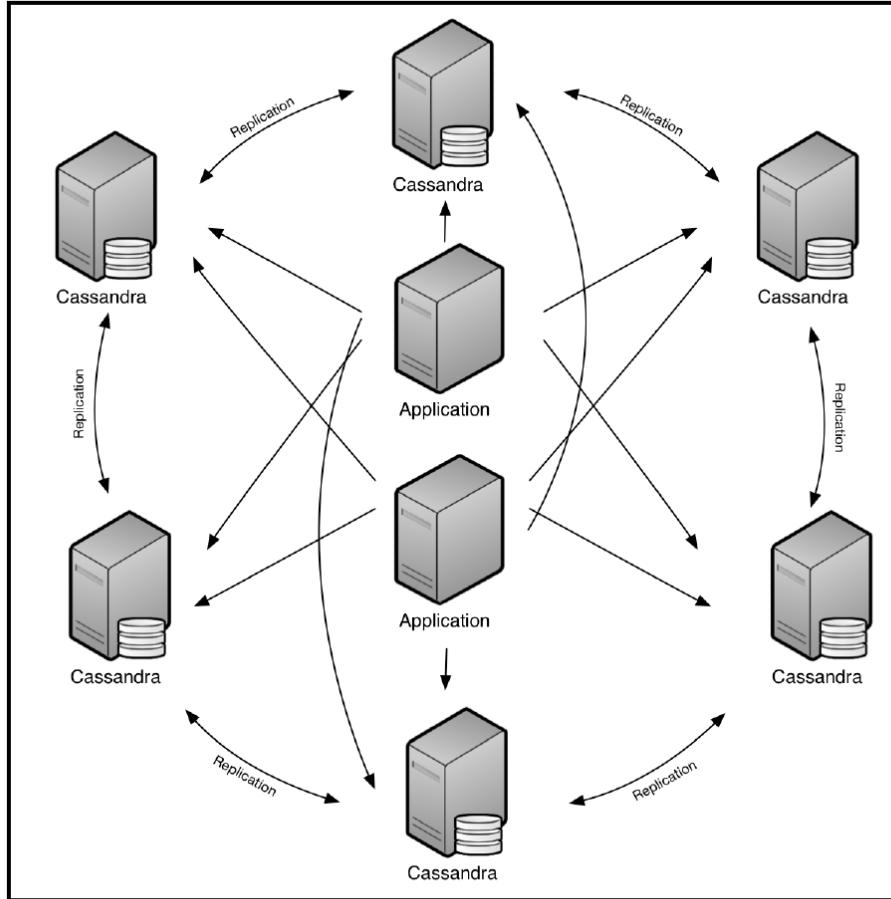
# Undesirable traits of leader election

- Applications must understand the database topology.
- Data partitions must be carefully planned.
- Writes are difficult to scale.
- A failover dramatically increases the complexity of the system in general, and especially so for multisite databases.
- Adding capacity requires reshuffling data with a potential for downtime.

# Cassandra is an elastically scalable database

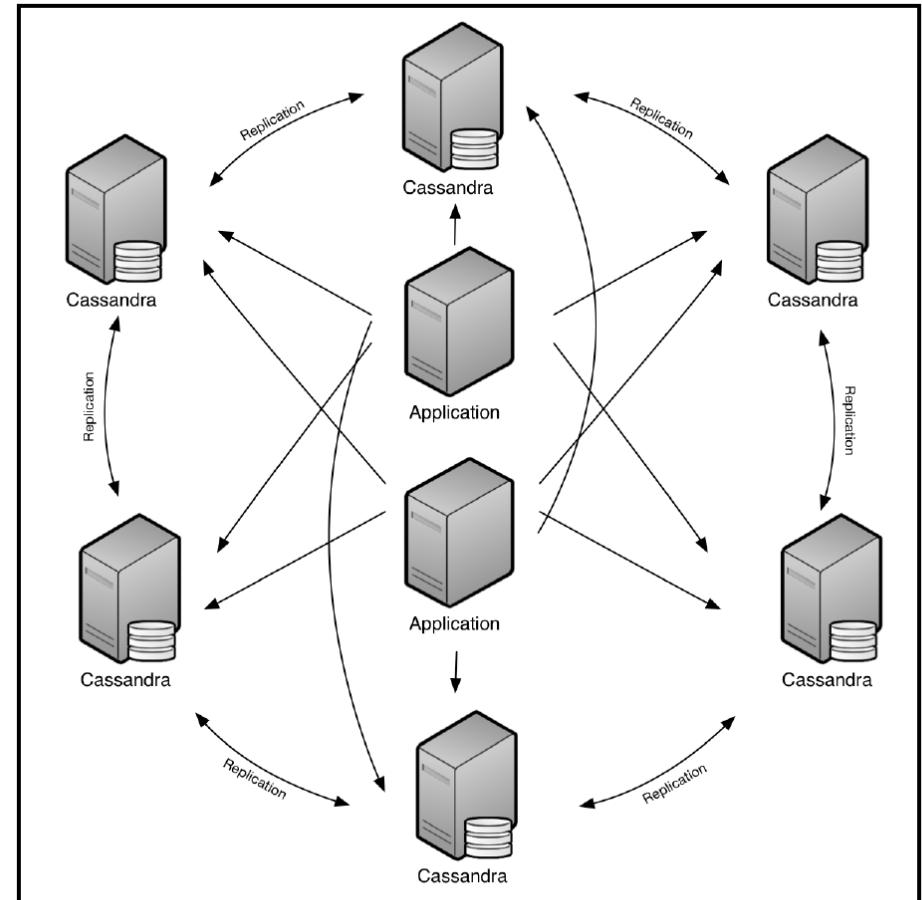
- Scalability – can continue serving a greater number of requests with little degradation in performance.
  - Vertical scaling: adding more hardware capacity and memory.
  - Horizontal scaling: adding more machines that have all or some of the data on them so that no one machine has to bear the entire burden of serving requests.
- Elastic scalability – refers to a special property of horizontal scalability, which means your cluster can seamlessly scale up and scale back down

# High availability and fault-tolerance



# Cassandra's peer-to-peer approach

- All nodes in a Cassandra cluster can accept reads and writes, no matter where the data being written or requested belongs in the cluster.
- Internode communication takes place by means of a gossip protocol, which allows all nodes to quickly receive updates without the need for a master coordinator



# Cassandra's peer-to-peer approach

- There are several advantages to the peer-to-peer approach:
  1. Simplicity.
  2. No node can be a single point of failure.
  3. Scaling up and down is fairly straightforward: Servers are added or removed from the cluster.
    - Servers in a peer-to-peer network communicate with each other and, eventually, new nodes are assigned a set of data to manage.
    - When a node is removed, servers hosting replicas of data from the removed node respond to read and write requests.

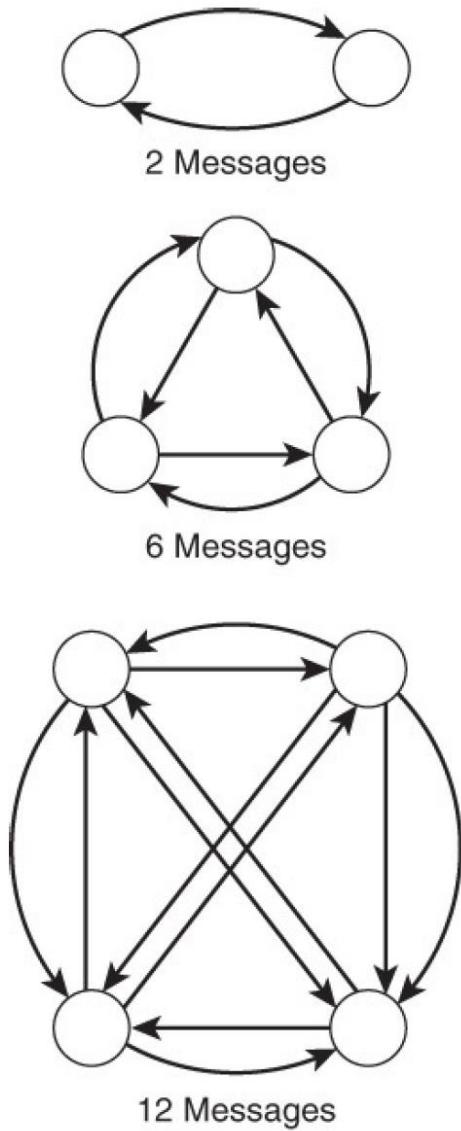
# Cassandra's peer-to-peer approach

- Because peer-to-peer networks do not have a single master coordinating server, the servers in the cluster are responsible for managing a number of operations that a master server would handle, including the following:
  - Sharing information about the state of servers in the cluster
  - Ensuring nodes have the latest version of data
  - Ensuring write data is stored when the server that should receive the write is unavailable
- Cassandra has protocols to implement all of these functions.

# Internode communications (Gossip)

- Gossip protocol:
  - Is a peer-to-peer communication protocol.
  - Periodically exchanges state information about themselves and other nodes in the cluster.
  - Runs every second on a timer and exchanges state message up to three other nodes.
  - Used as an automatic mechanism for replication in distributed databases
  - Older information is overwritten with the most current state for a particular node

# Internode communications (Gossip)



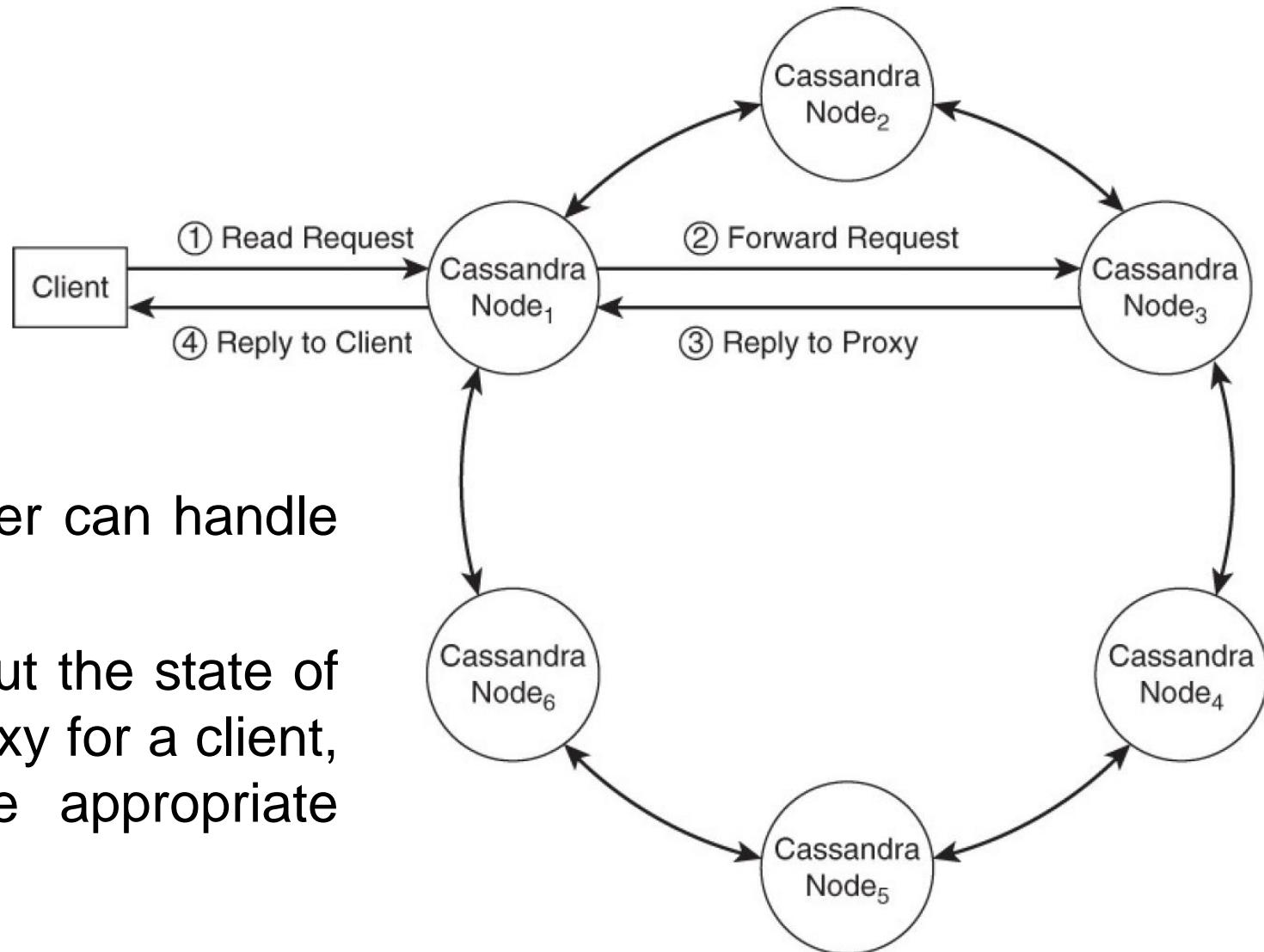
- The number of messages sent in a complete server-to-server communication protocol grows more rapidly each time a server is added to the cluster.

# How the Gossiper works

- Once per second, the gossiper will choose a random node in the cluster and initialize a gossip session with it.
  - Each round of gossip requires three messages.
- The gossip initiator sends its chosen friend a `GossipDigestSynMessage`.
- When the friend receives this message, it returns a `GossipDigestAckMessage`.
- When the initiator receives the ack message from the friend, it sends the friend a `GossipDigestAck2Message` to complete the round of gossip.

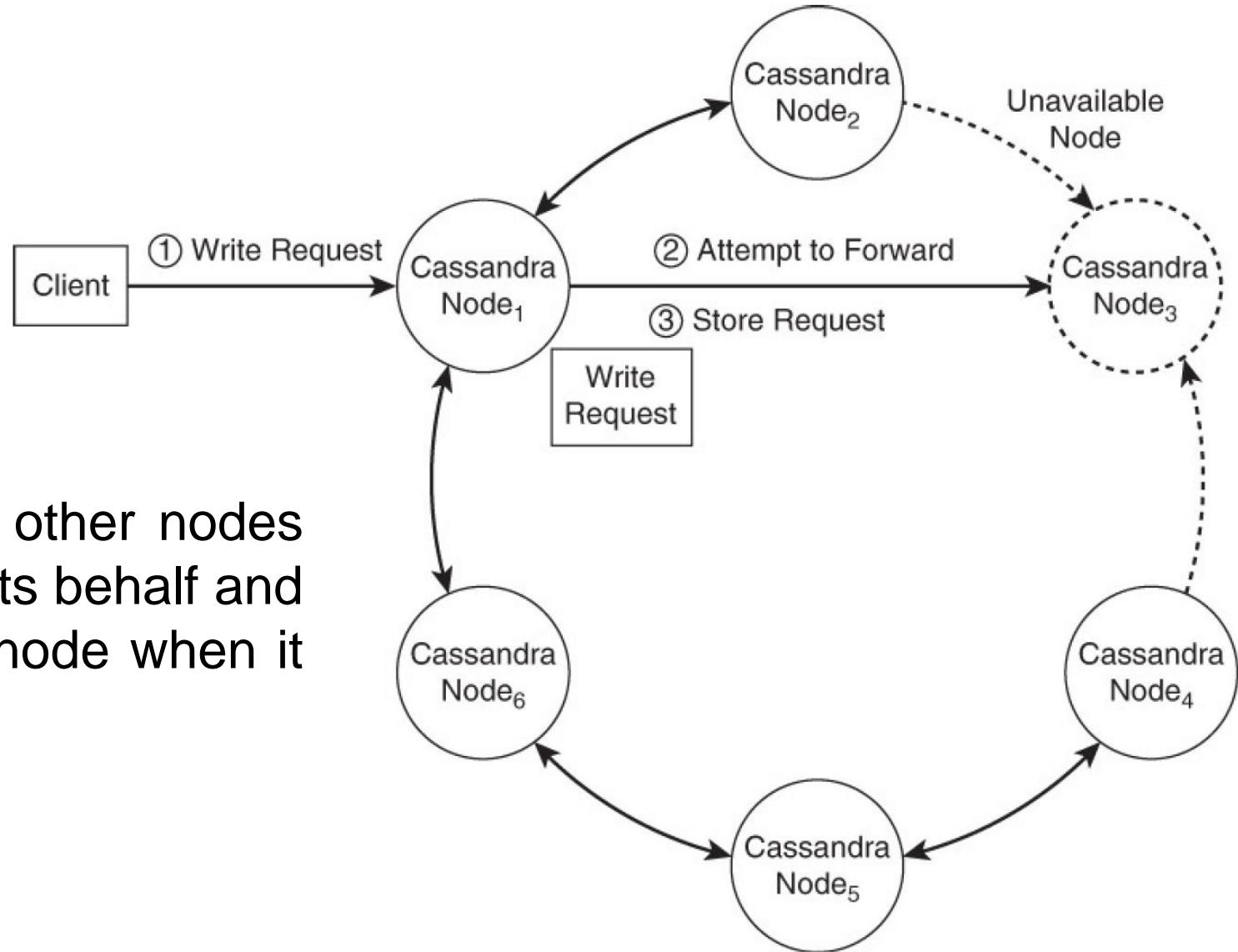
# Cassandra – Read

- Any node in a Cassandra cluster can handle a client request.
- All nodes have information about the state of the cluster and can act as a proxy for a client, forwarding the request to the appropriate node in the cluster.



# Cassandra – Write

- If a node is unavailable, then other nodes can receive write requests on its behalf and forward them to the intended node when it becomes available.



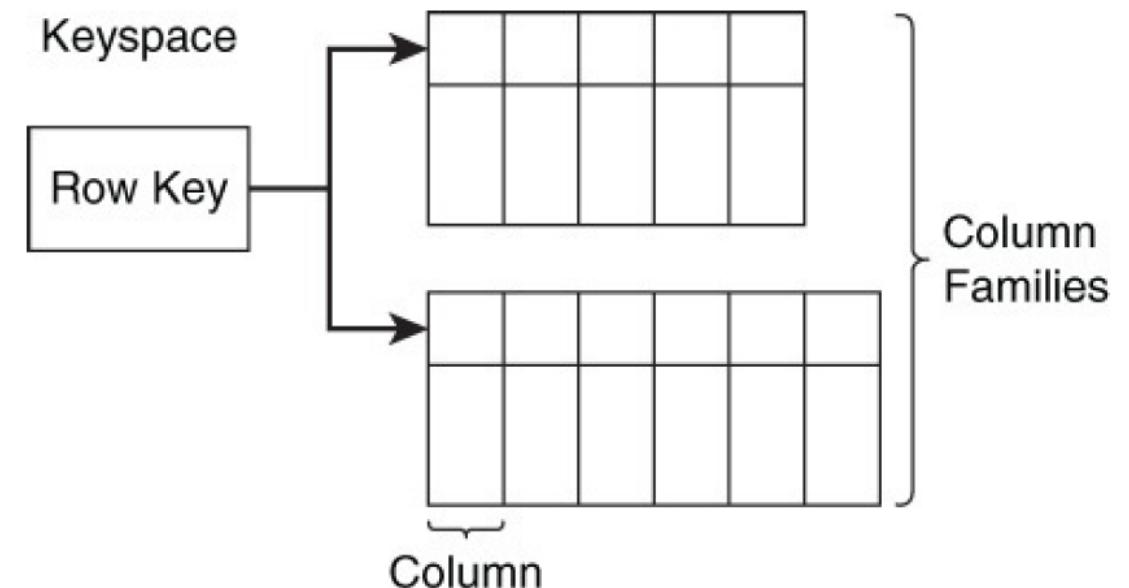
# When to Use Column-Oriented Databases?

- Column family databases are appropriate choices for large-scale database deployments that require high levels of write performance, a large number of servers or multi-data center availability.
  - Cassandra supports multi-data centre deployment, including multi-data centre replication.
- Write-intensive operations, such as those found in social networking applications, are good candidates for using column family databases.
  - Cassandra's peer-to-peer architecture with support for hinted handoff means the database will always be able to accept write operations as long as at least one node is functioning and reachable.
- Column family databases are also appropriate when a large number of servers are required to meet expected workloads.

# Terminology

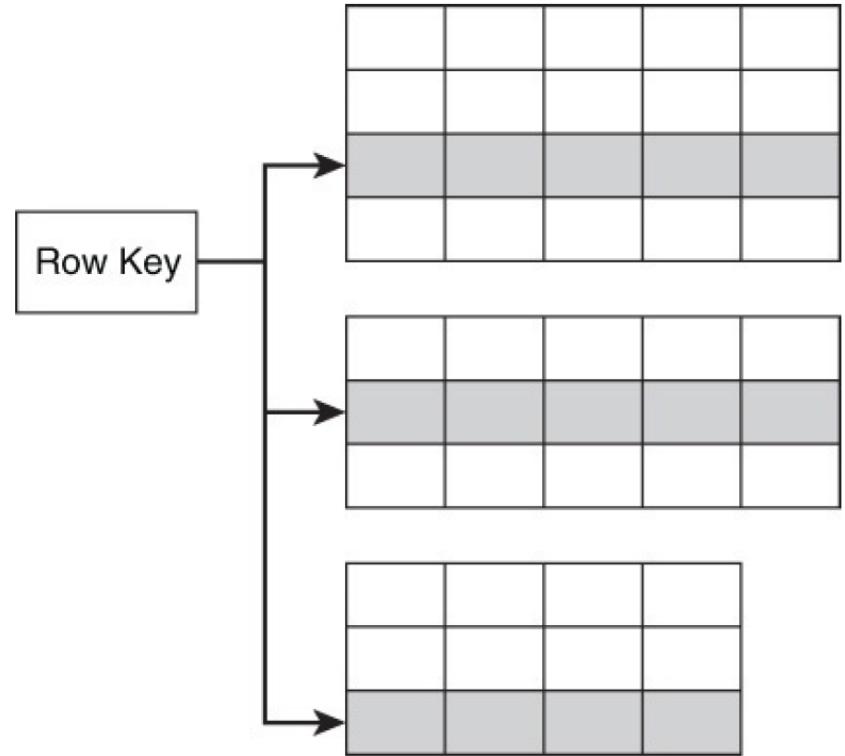
## ■ Keyspace

- A keyspace is the top-level data structure that logically holds column families, row keys, and related data structures. Typically, there is one keyspace per application.
- A keyspace is analogous to a schema in a relational database.



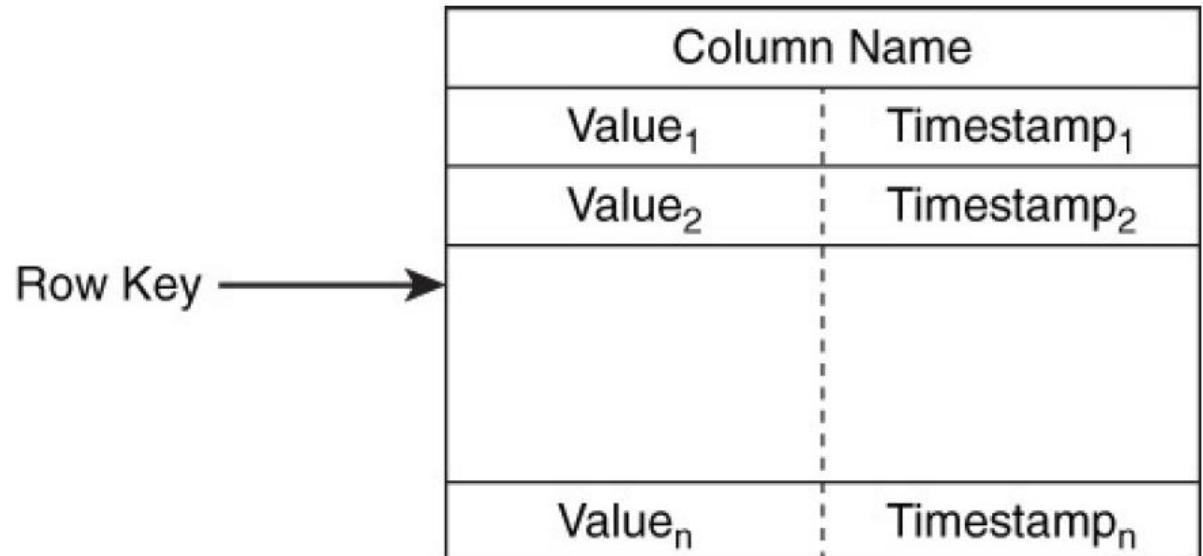
# Terminology

- Row Key
  - A row key uniquely identifies a row and has a role in determining the order in which data is stored.
  - It serves some of the same purposes as a primary key in a relational database.
- In Cassandra, rows are stored in an order determined by an object known as a partitioner.
- Cassandra uses a random partitioner by default.
  - As the name implies, the partitioner randomly distributes rows across nodes.
  - Cassandra also provides an order-preserving partitioner, which can provide lexicographic ordering.



# Terminology

- Column
  - A column, along with a row key and version stamp, uniquely identifies values.
- Columns have three parts:
  - A column name
  - A time stamp or other version stamp
  - A value



A diagram illustrating a column-oriented database structure. On the left, a horizontal arrow labeled "Row Key" points to the leftmost column of a table. The table has four columns: "Column Name" (header), "Value<sub>1</sub>", "Timestamp<sub>1</sub>", and "Value<sub>n</sub>". A vertical dashed line separates the "Value" and "Timestamp" columns. The "Column Name" header applies to all columns. The "Value" and "Timestamp" columns are repeated for each row, with the "n" indicating multiple rows.

Column Name			
	Value <sub>1</sub>	Timestamp <sub>1</sub>	
	Value <sub>2</sub>	Timestamp <sub>2</sub>	
	Value <sub>n</sub>	Timestamp <sub>n</sub>	

# Terminology

## ■ Column Families

- Column families are collections of related columns. Columns that are frequently used together should be grouped into the same column family.
- Column families are stored in a keyspace.
- Each row in a column family is uniquely identified by a row key.
- Column families are analogous to relational database tables: They store multiple rows and multiple columns.

Street	City	State	Province	Zip	Postal Code	Country
178 Main St.	Boise	ID		83701		U.S.
89 Woodridge	Baltimore	MD		21218		U.S.
293 Archer St.	Ottawa		ON		K1A 2C5	Canada
8713 Alberta DR	Vancouver		BC		VSK 0AI	Canada

# CQLSH

# CQL- Cassandra Query Language

- A simple way to manipulate the data you have stored.
- Syntax is similar with SQL.
- CQL has no concept of GROUP or JOIN, and very limited implementation of ORDER BY.

# Using Cassandra

- Starting the server
  - Cassandra

```
ca Command Prompt
Microsoft Windows [Version 10.0.18362.295]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\agnes>Cassandra
Detected powershell execution permissions. Running with enhanced startup scripts.
*-----*
*-----*
*-----*
*-----*

WARNING! Automatic page file configuration detected.
It is recommended that you disable swap when running Cassandra
for performance and stability reasons.

*-----*
*-----*
*-----*
*-----*

WARNING! Detected a power profile other than High Performance.
Performance of this node will suffer.
Modify conf\cassandra-env.ps1 to suppress this warning.

*-----*
*-----*

C:\Users\agnes>CompilerOracle: dontinline org/apache/cassandra/db/Columns$Serializer.deserializeLargeSubset (Lorg/apache/
/cassandra/io/util/DataInputPlus;Lorg/apache/cassandra/db/Columns;I)Lorg/apache/cassandra/db/Columns;
CompilerOracle: dontinline org/apache/cassandra/db/Columns$Serializer.serializeLargeSubset (Ljava/util/Collection;ILorg/
apache/cassandra/db/Columns;ILorg/apache/cassandra/io/util/DataOutputPlus;)V
CompilerOracle: dontinline org/apache/cassandra/db/Columns$Serializer.serializeLargeSubsetSize (Ljava/util/Collection;IL
org/apache/cassandra/db/Columns;I)I
```

```
ca Command Prompt
608, 6752326174351411741, 6812203637877176746, 6822529115892277513, 6832955196583004842, 700262294451494778, 7006777706
105829064, 706284290281036485, 716481444730208141, 7169289300961876625, 7170355781388566671, 723907086089302642, 725096
2133106543773, 7252157898389116036, 729123997251670932, 7302197399080998724, 7307244025170999720, 7308622642427957092,
733484207322931589, 7401969648597908406, 7540602173729915359, 7559891780623580174, 761547639469683645, 76931986922583490
25, 7844629827816745155, 791225993380583144, 7930122995524232025, 7975445965904083785, 8300444126466300992, 833315573315
127802, 8552043845335699757, 8613296785285389915, 8628546053674472951, 86391957299869461, 8801945547326807134, 8809850
909177235390, 8900966662138010949, 9080986765846146674, 913844876659463123, 9193766908328460797, 9201824182081692134, 9
51251827600193902, 957972844823219835]
INFO [main] 2019-07-28 13:25:02,932 StorageService.java:1483 - JOINING: Finish joining ring
INFO [main] 2019-07-28 13:25:03,178 StorageService.java:2327 - Node localhost/127.0.0.1 state jump to NORMAL
```

- Running the CQL Shell
  - cqlsh

```
ca Command Prompt - cqlsh
Microsoft Windows [Version 10.0.18362.239]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\agnes>cqlsh

WARNING: console codepage must be set to cp65001 to support utf-8 encoding on Windows platforms.
If you experience encoding problems, change your console codepage with 'chcp 65001' before starting cqlsh.

Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.4 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
WARNING: pyreadline dependency missing. Install to enable tab completion.
cqlsh>
```

# Using Cassandra

- To learn about the current cluster you're working in:
  - DESCRIBE CLUSTER;
- To list all available keyspaces in the cluster:
  - DESCRIBE KEYSPACES;
- Change keyspaces:
  - USE <keyspace\_name>
- To show all tables:
  - DESCRIBE TABLES;

# Using Cassandra

- To create a keyspace:
  - CREATE KEYSPACE my\_keyspace WITH replication = { 'class': 'SimpleStrategy', 'replication\_factor': 1 };
- To check the newly created keyspace:
  - DESCRIBE KEYSPACE my\_keyspace

# Using Cassandra

- To create a table:
  - `CREATE TABLE user ( first_name text,  
last_name text,  
PRIMARY KEY (first_name)  
);`
- To get the description of the newly created table:
  - `DESCRIBE TABLE user;`

# Using Cassandra

- To write a value (INSERT command):

- ```
INSERT INTO user (first_name, last_name )
VALUES ('Bill', 'Nguyen');
```

- To read (SELECT command):

- ```
SELECT * FROM user WHERE first_name='Bill';
```

- To count (SELECT COUNT command):

- ```
SELECT COUNT (*) FROM user;
```

# Using Cassandra

- To delete a column (DELETE command):
  - `DELETE last_name FROM user WHERE first_name='Bill';`
- To delete the entire row (DELETE command):
  - `DELETE FROM user WHERE first_name='Bill';`

# Using Cassandra

- To remove all data from a table (TRUNCATE command):
  - TRUNCATE user;
- To delete table schema (DROP TABLE command):
  - DROP TABLE user;

# References

- [1] C. Carlos and M. Steven, Database systems: *Design, implementation, & management*. Cengage Learning, 12<sup>th</sup> ed., 2016.
- [2] C. Y. Kan, Cassandra Data Modeling and Analysis. Birmingham, UK: Packt Publishing, 1<sup>st</sup> ed., 2014.
- [3] D. Sullivan, NoSQL for Mere Mortals. Michigan, USA: Addison-Wesley Professional, 1<sup>st</sup> ed., 2015.
- [4] J. Carpenter and E. Hewitt, Cassandra: The Definitive Guide. Sebastopol, CA: O'Reilly Media, Inc., 2<sup>nd</sup> ed., 2016.
- [5] N. Neeraj, Mastering Apache Cassandra. Birmingham, UK: Packt Publishing, 1<sup>st</sup> ed., 2013.
- [6] R. Strickland, Cassandra 3.x High Availability. Birmingham, UK: Packt Publishing, 2<sup>nd</sup> ed., 2016.
- [7] S. Alapati, Expert Apache Cassandra Administration. Texas, USA: Apress, 1<sup>st</sup> ed., 2018.
- [8] W. Lemahieu, S. V. Broucke and B. Baesens, Principles of Database Management: *The Practical Guide to Storing, Managing and Analyzing Big and Small Data*. New York, NY, USA: Cambridge University Press, 1<sup>st</sup> ed., 2018.

# Tutorial Work 2

## MongoDB Aggregation

# SOLUTIONS

Tutorial objectives:

- To be able to perform aggregation in MongoDB.
- To understand how aggregation pipeline works in MongoDB.
- To be able to solve complex queries in document databases.

**Important Reminder:**

- The unit consists of the Tutorial Work assessment (weighing 5%) conducted during week 4, 5, 7, 9, 10.
- **This tutorial work (weighing 1% out of 5%) is the first among the assessed tutorials works. It will be assessed during your tutorial time.**

**Marking Rubric:**

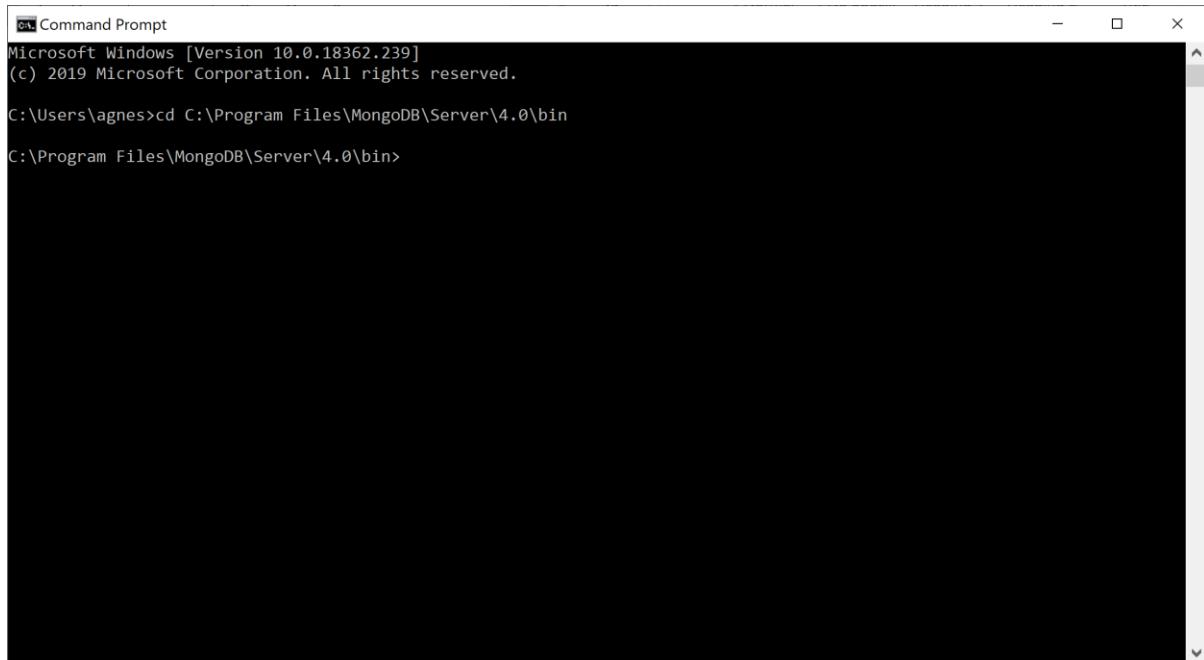
| Mark | Description            |                                               |                                                                                                                                                                                                                                                                                                                                                                                                        |
|------|------------------------|-----------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|      | Level of understanding | Preparation                                   | Tutorial Task and Interview Questions                                                                                                                                                                                                                                                                                                                                                                  |
| 1    | Exceptional            | Student has prepared prior to the tutorial    | Completed all of the tutorial tasks and has answered all interview questions without mistakes/errors                                                                                                                                                                                                                                                                                                   |
| 0.75 | Competent              | Student has prepared prior to the tutorial    | either<br>completed all of the tutorial tasks and has answered at least 50% of the interview questions without mistakes/errors<br><br>or:<br>completed at least 50% of the tutorial tasks and has answered all interview questions without mistakes/errors                                                                                                                                             |
| 0.5  | Moderate               | Student has prepared prior to the tutorial    | either:<br>completed more than 50% of the tutorial tasks and answered less than 50% of the interview questions without mistakes/errors<br><br>or:<br>completed less than 50% of the tutorial tasks and answered more than 50% of the interview questions without mistakes/errors<br><br>or:<br>completed 50% of the tutorial tasks and answered 50% of the interview questions without mistakes/errors |
| 0.25 | Inadequate             | Student has prepared prior to the tutorial    | completed less than 50% of the tutorial tasks and answered less than 50% of the interview questions without mistakes/errors                                                                                                                                                                                                                                                                            |
| 0    | No understanding       | Student did not prepare prior to the tutorial | did not complete any of the tutorial tasks and did not answer any interview questions without mistakes/errors                                                                                                                                                                                                                                                                                          |

## File Template for Answer

A file template (`Week5_MongoDBexercise.js`) is provided for you on Moodle (under the Week 5 resources) to answer the tutorial work. You can use any word editor application (e.g. [Atom](#), [Notepad++](#)) to edit your answer in the provided file template.

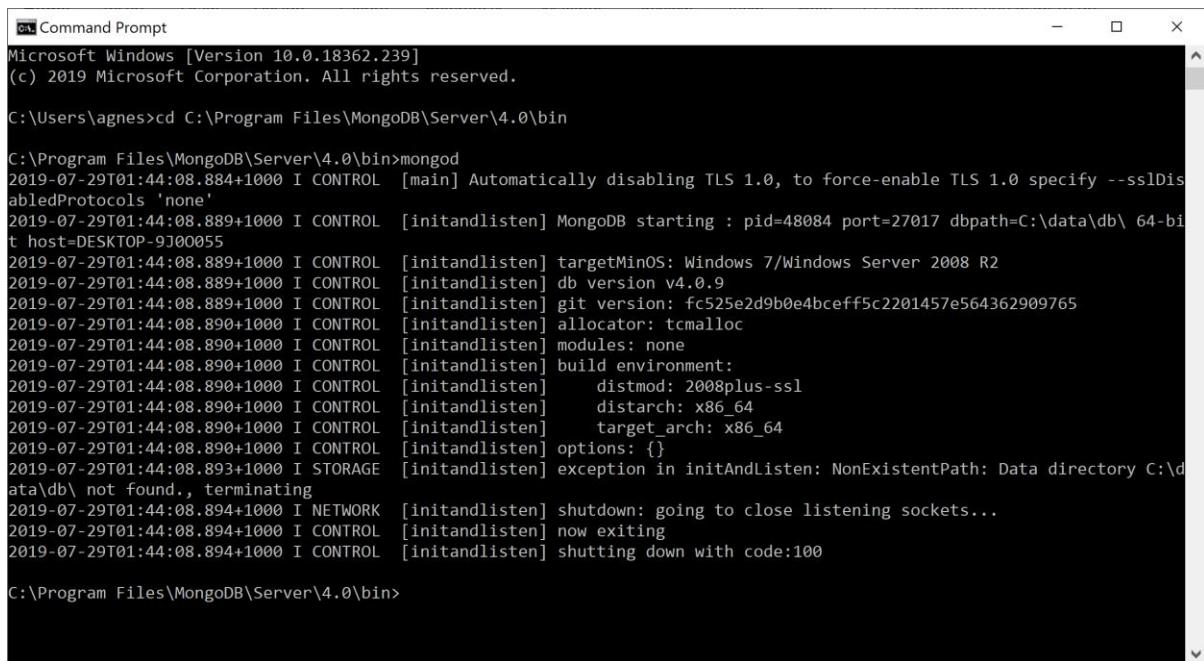
## Connecting to Mongo Shell on Windows

1. Press the **Windows Key** → type **Run** in the search bar → **Enter**.
2. Type **cmd** to open the **Command Prompt** → **Enter**.
3. Type “**cd C:\Program Files\MongoDB\Server\4.4\bin**” (the directory path might be different if you did not install MongoDB in C: drive).



The screenshot shows a Windows Command Prompt window titled "Command Prompt". The window title bar says "Command Prompt". The window content area shows the following text:  
Microsoft Windows [Version 10.0.18362.239]  
(c) 2019 Microsoft Corporation. All rights reserved.  
C:\Users\agnes>cd C:\Program Files\MongoDB\Server\4.0\bin  
C:\Program Files\MongoDB\Server\4.0\bin>

4. Type **mongod** to start MongoDB server.



```

C:\ Command Prompt
Microsoft Windows [Version 10.0.18362.239]
(c) 2019 Microsoft Corporation. All rights reserved.

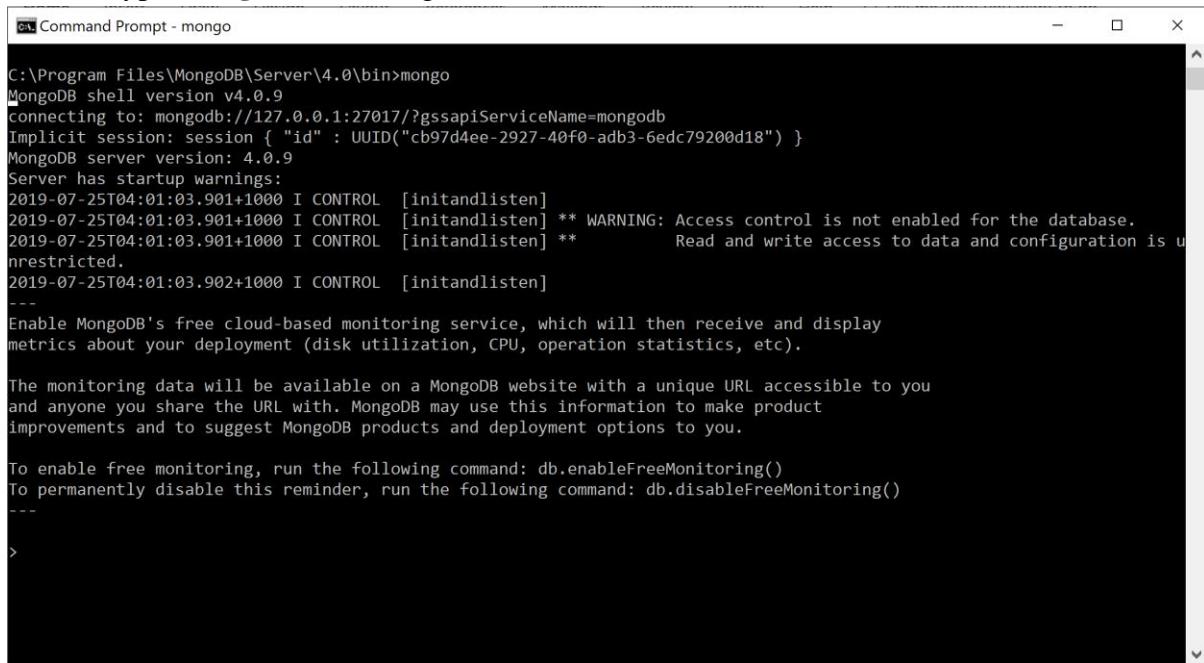
C:\Users\agnes>cd C:\Program Files\MongoDB\Server\4.0\bin

C:\Program Files\MongoDB\Server\4.0\bin>mongod
2019-07-29T01:44:08.884+1000 I CONTROL  [main] Automatically disabling TLS 1.0, to force-enable TLS 1.0 specify --sslDisabledProtocols 'none'
2019-07-29T01:44:08.889+1000 I CONTROL  [initandlisten] MongoDB starting : pid=48084 port=27017 dbpath=C:\data\db\ 64-bit host=DESKTOP-9J00055
2019-07-29T01:44:08.889+1000 I CONTROL  [initandlisten] targetMinOS: Windows 7/Windows Server 2008 R2
2019-07-29T01:44:08.889+1000 I CONTROL  [initandlisten] db version v4.0.9
2019-07-29T01:44:08.889+1000 I CONTROL  [initandlisten] git version: fc525e2d9b0e4bceff5c2201457e564362909765
2019-07-29T01:44:08.890+1000 I CONTROL  [initandlisten] allocator: tcmalloc
2019-07-29T01:44:08.890+1000 I CONTROL  [initandlisten] modules: none
2019-07-29T01:44:08.890+1000 I CONTROL  [initandlisten] build environment:
2019-07-29T01:44:08.890+1000 I CONTROL  [initandlisten]   distmod: 2008plus-ssl
2019-07-29T01:44:08.890+1000 I CONTROL  [initandlisten]   distarch: x86_64
2019-07-29T01:44:08.890+1000 I CONTROL  [initandlisten]   target_arch: x86_64
2019-07-29T01:44:08.893+1000 I CONTROL  [initandlisten] options: {}
2019-07-29T01:44:08.893+1000 I STORAGE  [initandlisten] exception in initAndListen: NonExistentPath: Data directory C:\data\db\ not found., terminating
2019-07-29T01:44:08.894+1000 I NETWORK  [initandlisten] shutdown: going to close listening sockets...
2019-07-29T01:44:08.894+1000 I CONTROL  [initandlisten] now exiting
2019-07-29T01:44:08.894+1000 I CONTROL  [initandlisten] shutting down with code:100

C:\Program Files\MongoDB\Server\4.0\bin>

```

## 5. Type mongo to run Mongo Shell.



```

C:\ Command Prompt - mongo
C:\Program Files\MongoDB\Server\4.0\bin>mongo
MongoDB shell version v4.0.9
connecting to: mongod://127.0.0.1:27017/?gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("cb97d4ee-2927-40f0-adb3-6edc79200d18") }
MongoDB server version: 4.0.9
Server has startup warnings:
2019-07-25T04:01:03.901+1000 I CONTROL  [initandlisten]
2019-07-25T04:01:03.901+1000 I CONTROL  [initandlisten] ** WARNING: Access control is not enabled for the database.
2019-07-25T04:01:03.901+1000 I CONTROL  [initandlisten] **             Read and write access to data and configuration is unrestricted.
2019-07-25T04:01:03.902+1000 I CONTROL  [initandlisten]
---
Enable MongoDB's free cloud-based monitoring service, which will then receive and display metrics about your deployment (disk utilization, CPU, operation statistics, etc).

The monitoring data will be available on a MongoDB website with a unique URL accessible to you and anyone you share the URL with. MongoDB may use this information to make product improvements and to suggest MongoDB products and deployment options to you.

To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---

>

```

## Connecting to Mongo Shell on Mac

1. Run the Mongo daemon, in one terminal window  
 > ~/mongodb/bin/mongod  
 This will start the Mongo Server.
2. Open another terminal, run the mongo shell  
 > ~/mongodb/bin/mongo

## MongoDB Aggregation Pipeline

### Part 1

Using the persons.json data that you imported in Week 4 Indexing Tutorial, answer these queries:

1. Display all names (both first name and last name) in ascending order.

```
db.contacts.aggregate([
  { $project : { _id : 0, "name.first" : 1, "name.last" : 1 }
}, {
  { $sort : {"name.first" : 1, "name.last" : 1} }
])
```

2. Show all people whose age are over 18 years old but less than 60 years old.

```
db.contacts.aggregate([
  { $match: {"dob.age" : {$gt: 18}, "dob.age" : {$lt: 60} }
})
])
```

3. Calculate the number of people based on the time zone. You can use the time zone description to group people together.

```
db.contacts.aggregate([
  { $group : { _id : "$location.timezone.description",
totalPeople : { $sum : 1 } } }
]).pretty()
```

4. Count how many people live in Melbourne who are over 50 years old.

```
db.contacts.aggregate([
  { $match : { "location.city": "melbourne", "dob.age" : {$gt: 50} } },
  { $count : "number_of_people" }
]).pretty()
```

5. Find the minimum and maximum registered age.

```
db.contacts.aggregate([
  { $group : { _id : null, min_age : { $min : "$registered.age" },
max_age : { $max: "$registered.age" } } }
]).pretty()
```

6. Count how many people live in Melbourne but whose nationality is **not** AU.

```
db.contacts.aggregate([
  { $match : { "location.city": "melbourne", nat: { $ne: "AU" } }
}, {
  { $count : "number_of_people" }
})
```

```
])
```

7. Show all people who live in the time zone “+10:00” but they are not located in Tasmania, Victoria, or New South Wales.

```
db.contacts.aggregate([
  { $match : { "location.timezone.offset" : "+10:00",
    "location.state": { $nin: ["tasmania", "victoria", "new south
    wales"] } } }
]).pretty()
```

8. Find the total number of people for each gender in each nationality and sort the result based on the nationality in ascending order.

```
db.contacts.aggregate([
  { $group : { _id: { gender : "$gender", nationality : "$nat" },
    totalPersons: { $sum: 1 } } },
  { $sort : {"_id.nationality" : 1} }
])
```

9. Show the top 5 nationalities.

```
db.contacts.aggregate([
  { $group : { _id : "$nat", totalPeople : { $sum : 1 } } },
  { $sort : { totalPeople : -1 } },
  { $limit : 5 }
]).pretty()
```

10. Using \$concat, display all person names in this format: “mr carl jacobs”.

```
db.contacts.aggregate([
  { $project: { personName: { $concat: [ "$name.title", " ",
    "$name.first", " ", "$name.last" ] } } },
  { $project: { _id:0 } }
])
```

11. Show the name, email, and registration date of all people who registered after 2014.

```
db.contacts.aggregate([
  { $project: { _id: 0, name: 1, email: 1, registered_date: { $convert: { input: '$registered.date', to: 'date' } } },
    { $match : { registered_date : { $gte: new ISODate("2015-01-
01T00:00:00Z") } } }
])
```

12. Display all **unique** cities and sort it in ascending order. Make sure that there are no duplicates in your output.

```
db.contacts.aggregate([
  { $group : { _id : "$location.city", cities : { $push :
    "$location.city" } } },
  { $sort : { cities : 1 } },
  { $project: { cities: 0 } }
```

])

## **Part 2**

Create a new database called universityDB and insert the documents below into a new collection called universities.

```
{
  country : "Australia",
  state : "VIC",
  name : "Monash University",
  initial: "MonU",
  location : {
    type : "Point",
    coordinates : [ -37.907803, 145.133957 ]
  },
  students : [
    { year : 2015, number : 70071 },
    { year : 2016, number : 72334 },
    { year : 2017, number : 78257 },
    { year : 2018, number : 80141 },
    { year : 2019, number : 84538 }
  ]
}

{
  country : "Australia",
  state : "NSW",
  name : "Sydney University",
  initial: "SydU",
  location : {
    type : "Point",
    coordinates : [ -33.865143, 151.209900 ]
  },
  students : [
    { year : 2015, number : 53166 },
    { year : 2016, number : 51913 },
    { year : 2017, number : 59129 },
    { year : 2018, number : 61715 },
    { year : 2019, number : 64713 }
  ]
}

{
  country : "Australia",
  state : "VIC",
  name : "Melbourne University",
  initial: "MelU",
}
```

```

    location : {
      type : "Point",
      coordinates : [ -37.809811, 144.965195 ]
    },
    students : [
      { year : 2015, number : 45993 },
      { year : 2016, number : 48088 },
      { year : 2017, number : 51282 },
      { year : 2018, number : 54989 },
      { year : 2019, number : 58023 }
    ]
  }
}

```

Create another collection called `courses` and insert these documents:

```

{
  university : "MonU",
  name : "Computer Science",
  level : "Advanced"
}

{
  university : "MonU",
  name : "Engineering",
  level : "Intermediate"
}

{
  university : "MonU",
  name : "Communication",
  level : "Beginner"
}

```

Answer these queries:

1. Find all universities in VIC.

```
db.universities.aggregate([
  { $match : { state : "VIC" } }
]) .pretty()
```

2. Show the university name, state, and country for all universities.

```
db.universities.aggregate([
  { $project : { _id : 0, name : 1, state : 1, country : 1 }
}]
) .pretty()
```

3. Count how many universities in each state.

```
db.universities.aggregate([
  { $group : { _id:"$state", "totalUni": { $sum:1} } }
```

```
])
```

4. Using \$out, carry the results of your aggregation in the previous question into a new collection called aggResults.

```
db.universities.aggregate([
  { $group : { _id:"$state", "totalUni": { $sum:1} } },
  { $out : 'aggResults' }
])
```

Show all collections in this database. Write down your observation.

5. Run the following:

```
db.universities.aggregate([
  { $match : { initial : "MonU" } },
  { $unwind : "$students" }
]).pretty()
```

What does \$unwind do? Write down your observation.

6. Calculate the total number of students for each university.

```
db.universities.aggregate([
  { $unwind : "$students" },
  { $group : { _id : "$name", number_of_students : { $sum : "$students.number" } } }
])
```

7. Calculate the total students in VIC in 2018.

```
db.universities.aggregate([
  { $unwind : "$students" },
  { $match : { state : "VIC", "students.year" : 2018 } },
  { $group : { _id : "$state", number_of_students : { $sum : "$students.number" } } }
])
```

8. Using \$lookup, do left outer join between universities and courses.

```
db.universities.aggregate([
  { $lookup : {
      from : "courses",
      localField : "initial",
      foreignField : "university",
      as : "courses"
    }
  }
]).pretty()
```

9. Show the top 2 highest number of students, together with the corresponding year.

```
db.universities.aggregate([
```

```
{ $match : { initial : "MonU" } },
{ $unwind : '$students' },
{ $project : { _id : 0, 'students.year' : 1,
'students.number' : 1 } },
{ $sort : { 'students.number' : -1 } },
{ $limit : 2 }
]).pretty()
```

10. Using `$addFields`, add the foundation year of Monash University, which was on 1958, to your output.

```
db.universities.aggregate([
{ $match : { initial : "MonU" } },
{ $addFields : { foundation_year : 1958 } }
]).pretty()
```

**The End**



**MONASH**  
University

MONASH  
INFORMATION  
TECHNOLOGY

# **FIT5137 – Advanced Database Technology**

Week 4 – Data Modelling with Document-Oriented Database

Semester 2, 2020

Developed by:

Dr. Agnes Haryanto

[Agnes.Haryanto@monash.edu](mailto:Agnes.Haryanto@monash.edu)

# Using FLUX

1. Visit <http://flux.qa/> on your internet enabled device
2. Log in using your Monash account (not required if you are already logged in to Monash)
3. Click on the “+” to join audience
4. Enter the Audience Code: **R8IFH4**
5. Select FIT5137 in the Active Presentation menu
6. Answer questions when they pop up

Alternatively, the quiz can be accessed through:

<https://flux.qa/R8IFH4>

The screenshot shows a web browser window titled "FLUX" with the URL <https://flux.qa/#/feeds/5d368...>. The page is titled "Lecture 0". Below the title are four icons: a bar chart, a video camera, a trophy, and a clock. The main content area is titled "Current Polls". It displays a multiple-choice poll with the question "What did you have for dinner?". The options are listed in a vertical list:

- A Pasta
- B Rice
- C Pizza
- D Salad
- E Nothing

# Agenda

1. Understanding document schemas
2. Modelling relations

# Document Database

- Documents in document databases combine structure and content.
- Documents in document databases are not constrained to a predefined set of tags for specifying structure.
- Instead, developers are free to choose the terms they need to structure their content just as data modelers choose table and column names for relational databases.

```
{  
    "_id" : 1005,  
    "firstName" : "Wendy",  
    "lastName" : "Wheat",  
    "address" : {  
        "streetaddress" : "6 Algorithm Street",  
        "suburb" : "Malvern",  
        "state" : "VIC",  
        "postcode" : 3144  
    },  
    "gender" : "female",  
    "course" : "GDS",  
    "year" : 2019,  
    "offcampus" : true,  
    "emailaddress" : "wwheat@yahoo.com",  
    "birthday" : ISODate("1990-10-30T00:00:00Z")  
}
```

# Document Database

- One of the advantages of document databases is that they provide flexibility when it comes to the structure of documents.
- If only 10% of your documents need to record certain information, why should you have to clutter the other 90% with unused fields? You do not have to when using document databases.

| ID | First Name | Middle Name | Last Name | Age | Interest          |
|----|------------|-------------|-----------|-----|-------------------|
| 1  | John       | Anthony     | Smith     |     | Music             |
| 2  | Nick       |             | Nice      |     | Sports,<br>Movies |
| 3  | Wendy      | Wilma       | Wheat     | 30  |                   |

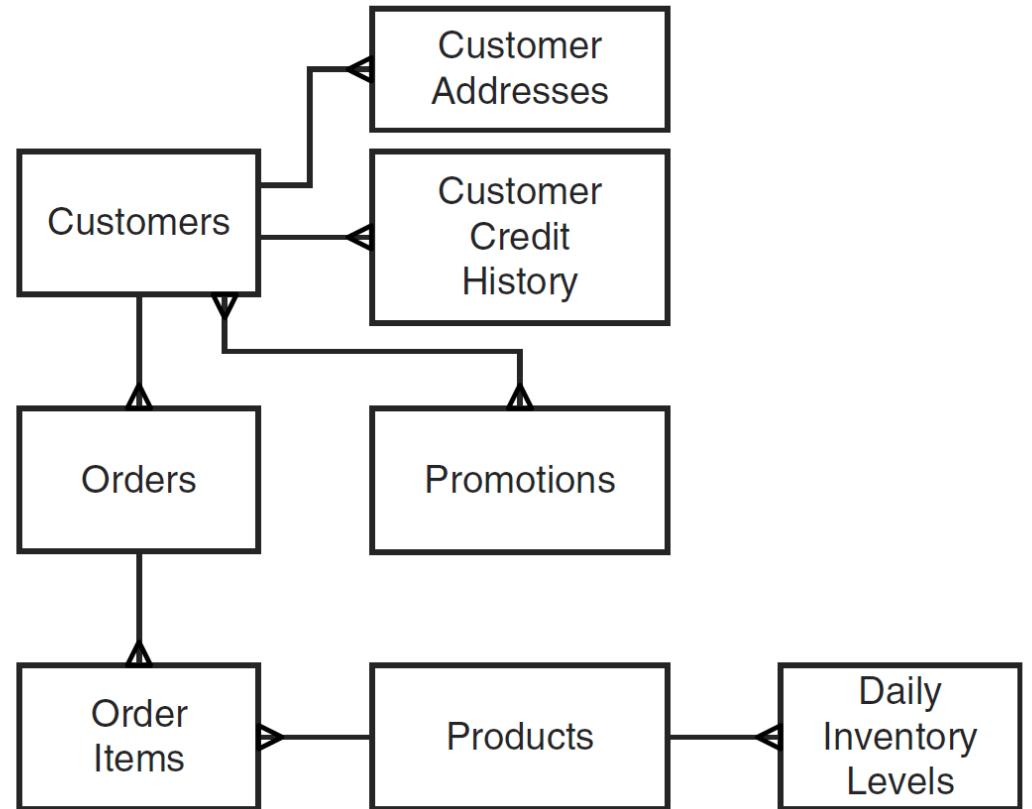
```
{id: 1, firstName: "John", middleName: "Anthony", lastName: "Smith", interest: "Music" }  
{id: 2, firstName: "Nick", lastName: "Nice", interest: ["Sports", "Movies"] }  
{id: 3, firstName: "Wendy", middleName: "Wilma", lastName: "Wheat", age: 30 }
```

# MongoDB – Schema-less

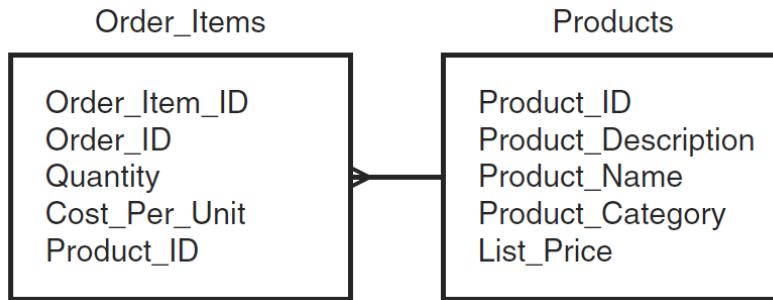
- MongoDB enforces no schemas
  - Documents do not require to use the same schema inside one collection.
  - Can mix different schemas into one collection.
- However, being schema-less does not mean that we cannot use some kind of schema for our data.
  - We need to think of an efficient way for our application later.

# Relational Database

- Relational Database utilizes the rules of normalization.
- A typical relational data model is designed to avoid data anomalies when inserts, updates, or deletes are performed.
- In theory, a data modeler will want to eliminate redundancy to minimize the chance of introducing anomalies.



# Relational Database – The Need for Joins



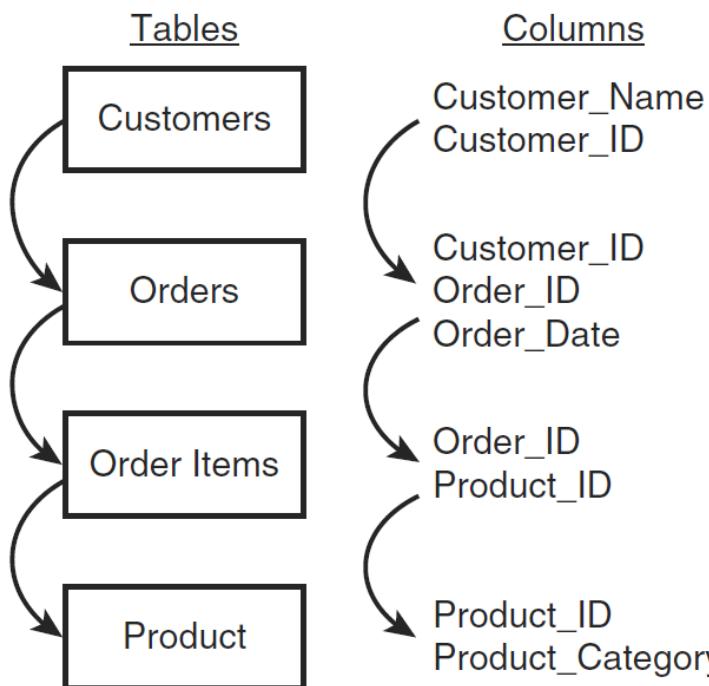
- Normalized models such as this minimize redundant data and avoid the potential for data anomalies.

| Order Items   |          |          |               |            |
|---------------|----------|----------|---------------|------------|
| Order_Item_ID | Order_ID | Quantity | Cost_Per_Unit | Product_ID |
| 1298          | 789      | 1        | \$25.99       | 345        |
| 1299          | 789      | 2        | \$20.00       | 372        |
| 1300          | 790      | 1        | \$12.50       | 591        |
| 1301          | 790      | 1        | \$20.00       | 372        |
| 1302          | 790      | 3        | \$12.99       | 413        |

| Products   |                                                             |                    |                        |            |
|------------|-------------------------------------------------------------|--------------------|------------------------|------------|
| Product_ID | Product_Description                                         | Product_Name       | Product_Category       | List_Price |
| 345        | Easy clean tablet cover that fits most 10" Android tablets. | Easy Clean Cover   | Electronic Accessories | 25.99      |
| 372        | Lightweight blue ear buds with comfort fit.                 | Acme Ear Buds      | Electronic Accessories | 20         |
| 413        | Set of 10 dry erase markers.                                | 10-Pack Markers    | Office Supplies        | 15         |
| 420        | 60"x48" whiteboard with marker and eraser holder.           | Large Whiteboard   | Office Supplies        | 56.99      |
| 591        | Pack of 100 individually wrapped screen wipes.              | Screen Clean Wipes | Office Supplies        | 12.99      |

# Executing Joins: The Heavy Lifting of Relational Databases

- Analyzing customers who bought a particular type of product requires three joins between four tables.



- Sample pseudocode for printing the name of customers who have purchased electronic accessories in the last 12 months:

```
for cust in get_customers():
    for order in get_customer_orders(cust.customer_id):
        if today() - 365 <= order.order_date:
            for order_item in get_order_items
                (order.order_id):
                    if 'electronic accessories' =
                        get_product_category(order_item.product_id):
                        customer_set = add_item
                            (customer_set,cust.name);

for customer_name in customer_set:
    print customer_name;
```

# Data Schemas & Data Modelling

Which Data does my App need or generate?

- Defines the field you will need and how they relate.
- *Example:* User information, product information, orders information, etc.

Where do I need my Data?

- Defines your required collections & field grouping.
- *Example:* Welcome page, product list page, orders page, etc.

Which kind of Data or Information do I want to display?

- Defines which queries you will need.
- *Example:* Welcome page, product names, product page, etc.

How often do I want to fetch my Data?

- Defines whether you should optimize for easy fetching.
- *Example:* For every page reload.

How often do I write or change my Data?

- Defines whether you should optimize for easy writing.
- *Example:* Orders are often changed/written while product data are rarely changed.

# Document Structure

- **Nested/Embedded Documents**

- Embedded documents capture relationships between data by storing related data in a single document structure.
- MongoDB documents make it possible to embed document structures in a field or array within a document. These denormalized data models allow applications to retrieve and manipulate related data in a single database operation.

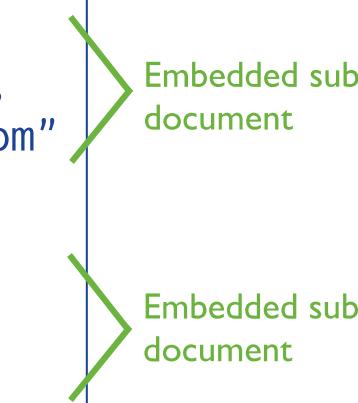
- **References**

- References store the relationships between data by including links or references from one document to another.
- Applications can resolve these references to access the related data. Broadly, these are normalized data models.

# Embedded Data Model

- In general, use embedded data models when:
  - There is “contains” relationships between entities (One-to-One Relationships).
  - There is One-to-Many relationships between entities. In these relationships the “many” or child documents always appear with or are viewed in the context of the “one” or parent 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

# Embedded Data Model



Embedded data models allow applications to store related pieces of information in the same database record.

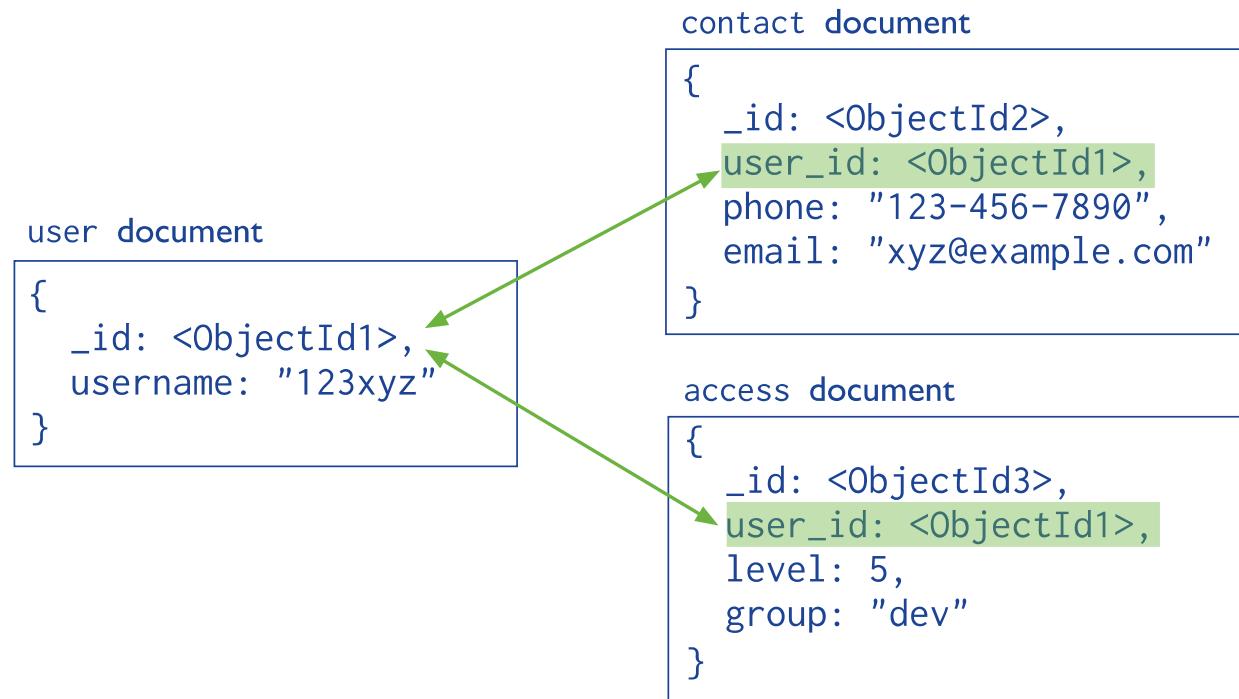


Applications may need to issue fewer queries and updates to complete common operations.



# References Data Model

- In general, use normalized data models:
  - When embedding would result in duplication of data but would not provide sufficient read performance advantages to outweigh the implications of the duplication.
  - To represent more complex Many-to-Many relationships.
  - To model large hierarchical data sets.



# One-to-One Relationship

- Normalized example:

- Patron document:

```
{  
  _id: "joe",  
  name: "Joe Bookreader"  
}
```

- Address document:

```
{  
  patron_id: "joe",  
  street: "123 Fake Street",  
  city: "Faketon",  
  state: "MA",  
  zip: "12345"  
}
```

- Embedded example:

```
{  
  _id: "joe",  
  name: "Joe Bookreader",  
  address: {  
    street: "123 Fake Street",  
    city: "Faketon",  
    state: "MA",  
    zip: "12345"  
  }  
}
```

# One-to-Many Relationship – Embedded

- Normalized example:

```
{  
  "_id": "joe",  
  "name": "Joe Bookreader"  
}  
  
{  
  "patron_id": "joe",  
  "street": "123 Fake Street",  
  "city": "Faketon",  
  "state": "MA",  
  "zip": "12345"  
}  
  
{  
  "patron_id": "joe",  
  "street": "1 Some Other Street",  
  "city": "Boston",  
  "state": "MA",  
  "zip": "12345"  
}
```

- Embedded example:

```
{  
  "_id": "joe",  
  "name": "Joe Bookreader",  
  "addresses": [  
    {  
      "street": "123 Fake Street",  
      "city": "Faketon",  
      "state": "MA",  
      "zip": "12345"  
    },  
    {  
      "street": "1 Some Other Street",  
      "city": "Boston",  
      "state": "MA",  
      "zip": "12345"  
    }  
  ]  
}
```



# One-to-Many Relationship – References

Embedding example:

```
{  
    title: "MongoDB: The Definitive Guide",  
    author: [ "Kristina Chodorow", "Mike Dirolf" ],  
    published_date: ISODate("2010-09-24"),  
    pages: 216,  
    language: "English",  
    publisher: {  
        name: "O'Reilly Media",  
        founded: 1980,  
        location: "CA"  
    }  
  
{  
    title: "50 Tips and Tricks for MongoDB Developer",  
    author: "Kristina Chodorow",  
    published_date: ISODate("2011-05-06"),  
    pages: 68,  
    language: "English",  
    publisher: {  
        name: "O'Reilly Media",  
        founded: 1980,  
        location: "CA"  
    }  
}
```

Reference example:

```
{  
    name: "O'Reilly Media",  
    founded: 1980,  
    location: "CA",  
    books: [123456789, 234567890, ...]  
}  
  
{  
    _id: 123456789,  
    title: "MongoDB: The Definitive Guide",  
    author: [ "Kristina Chodorow", "Mike Dirolf" ],  
    published_date: ISODate("2010-09-24"),  
    pages: 216,  
    language: "English"  
}  
  
{  
    _id: 234567890,  
    title: "50 Tips and Tricks for MongoDB Developer",  
    author: "Kristina Chodorow",  
    published_date: ISODate("2011-05-06"),  
    pages: 68,  
    language: "English"  
}
```

Reference example:

```
{  
    _id: "oreilly",  
    name: "O'Reilly Media",  
    founded: 1980,  
    location: "CA"  
}  
  
{  
    _id: 123456789,  
    title: "MongoDB: The Definitive Guide",  
    author: [ "Kristina Chodorow", "Mike Dirolf" ],  
    published_date: ISODate("2010-09-24"),  
    pages: 216,  
    language: "English",  
    publisher_id: "oreilly"  
}  
  
{  
    _id: 234567890,  
    title: "50 Tips and Tricks for MongoDB Developer",  
    author: "Kristina Chodorow",  
    published_date: ISODate("2011-05-06"),  
    pages: 68,  
    language: "English",  
    publisher_id: "oreilly"  
}
```

# Relations

| Nested/Embedded Documents                                                                                               | References                                                                                       |
|-------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------|
| Group data together logically.                                                                                          | Split data across collections.                                                                   |
| Great for data that belongs together and is not really overlapping with other data.                                     | Great for related but shared data as well as for data which is used in relations and standalone. |
| Avoid super-deep nesting (100+ levels) or extremely long arrays (since MongoDB has 16MB size limit for every document). | Allows you to overcome nesting and size limits (by creating new documents).                      |

# One-to-Many Relationship

- Cases:
  - One order can have many order items.
  - One apartment building can have many apartments.
  - One organization can have many departments.
  - One product can have many parts.

# Many-to-Many Relationship

- Cases:
  - Doctors can have many patients and patients can have many doctors.
  - Operating system user groups can have many users and users can be in many operating system user groups.
  - Students can be enrolled in many courses and courses can have many students enrolled.
  - People can join many clubs and clubs can have many members.

# Many-to-Many Relationship

## Courses:

```
{  
  { courseID: 'C1667',  
    title: 'Introduction to Anthropology',  
    instructor: 'Dr. Margret Austin',  
    credits: 3,  
    enrolledStudents: ['S1837', 'S3737', 'S9825' ...  
      'S1847'] },  
  { courseID: 'C2873',  
    title: 'Algorithms and Data Structures',  
    instructor: 'Dr. Susan Johnson',  
    credits: 3,  
    enrolledStudents: ['S1837', 'S3737', 'S4321', 'S9825'  
      ... 'S1847'] },  
  { courseID: C3876,  
    title: 'Macroeconomics',  
    instructor: 'Dr. James Schulen',  
    credits: 3,  
    enrolledStudents: ['S1837', 'S4321', 'S1470', 'S9825'  
      ... 'S1847'] },  
  ...  
}
```

## Students:

```
{  
  { studentID: 'S1837',  
    name: 'Brian Nelson',  
    gradYear: 2018,  
    courses: ['C1667', C2873, 'C3876'] },  
  { studentID: 'S3737',  
    name: 'Yolanda Deltor',  
    gradYear: 2017,  
    courses: [ 'C1667', 'C2873' ] },  
  ...  
}
```

# MongoDB – \$lookup

- \$lookup performs a left outer join to an unsharded collection in the same database to filter in documents from the “joined” collection for processing.
- Syntax:

```
{  
  $lookup:  
    {  
      from: <collection to join>,  
      localField: <field from the input documents>,  
      foreignField: <field from the documents of the "from" collection>,  
      as: <output array field>  
    }  
}
```

# MongoDB – \$lookup

- Example:

```
db.patron.aggregate([  
  {  
    $lookup:  
      {  
        from: "address",  
        localField: "_id",  
        foreignField: "patron_id",  
        as: "addresses"  
      }  
  }  
)
```

# References

- [1] Academind, “MongoDB – The Complete Developer’s Guide”  
<https://www.academind.com/learn/mongodb/>.
- [2] D. Sullivan, NoSQL for Mere Mortals. Michigan, USA: Addison-Wesley Professional, 1<sup>st</sup> ed., 2015.
- [3] K. Chodorow, MongoDB: The Definitive Guide. Sebastopol, CA, USA: O’Reilly Media, Inc., 2<sup>nd</sup> ed., 2013.
- [4] MongoDB, “Data Models” <https://docs.mongodb.com/manual/data-modeling/>.

## Tutorial 3

# MongoDB Text Search **SOLUTIONS**

This tutorial requires you to use the `students` collection from Tutorial Work 1.

| <b>ID</b> | <b>First name</b> | <b>Last name</b> | <b>Address</b>        |                |              |                 | <b>Gender</b> | <b>Course</b> | <b>Year</b> | <b>Off-Campus</b> | <b>Email Address</b>                  |
|-----------|-------------------|------------------|-----------------------|----------------|--------------|-----------------|---------------|---------------|-------------|-------------------|---------------------------------------|
|           |                   |                  | <b>Street Address</b> | <b>Suburb</b>  | <b>State</b> | <b>Postcode</b> |               |               |             |                   |                                       |
| 1001      | John              | Smith            | 123 Monash Drive      | Clayton        | VIC          | 3168            | Male          | MIT           | 2019        | false             | jsmith@gmail.com,<br>jsmith@yahoo.com |
| 1002      | Mary              | Citizen          | 900 Dandenong Road    | Caulfield East | VIC          | 3145            | Female        | MDS           | 2018        | true              | mcitizen@gmail.com                    |
| 1003      | Fred              | Bloggs           | 90 Wellington Road    | Clayton        | VIC          | 3168            | Male          | MIT,<br>MBIS  | 2017        | false             | fredb@gmail.com                       |
| 1004      | Nick              | Nice             | 3 Robinson Avenue     | Kew            | VIC          | 3080            | Male          | MCS           | 2018        | false             | nicknice@gmail.com                    |
| 1005      | Wendy             | Wheat            | 6 Algorithm Street    | Malvern        | VIC          | 3144            | Female        | GDS           | 2019        | true              | wwheat@yahoo.com                      |

*Table 1 – students collection*

### **Text Search**

- Find the students who are using Gmail.  
Can you find it without knowing the full email address?
- Type the following code:

```
db.students.createIndex({emailAddress:"text"})
```

What does the code above do?

- Now type in the following code:

```
db.students.find({$text:{$search:"gmail"}})
```

Can you find the students who are using Gmail now?

- Show students who are off-campus students and using Gmail. Sort the result by the email address in ascending order.

```
db.students.find(
    {$text: {$search:"gmail"}, "offCampus": true},
    {"firstName": 1, "lastName": 1}
).sort({"emailAddress": 1})
```

- Open MongoDB Compass, then connect to localhost. In the list of databases, you will have the `studentEnrolment`. Select the `studentEnrolment` database.

| Database Name    | Storage Size | Collections | Indexes |  |
|------------------|--------------|-------------|---------|--|
| admin            | 16.0KB       | 0           | 1       |  |
| config           | 24.0KB       | 0           | 2       |  |
| local            | 36.0KB       | 1           | 1       |  |
| studentEnrolment | 16.0KB       | 1           | 1       |  |

6. Currently, you only have student collection in this database. Select the students collection and observe the documents inside this collection, including their data type.
7. Go to the Indexes tab. How many indexes are there? Write down your observation.

studentEnrolment.students

| Documents                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | Aggregations     | Explain Plan   | Indexes                  |                   |                  |                       |      |      |       |            |      |      |         |         |                          |        |  |                   |      |         |                         |          |  |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|----------------|--------------------------|-------------------|------------------|-----------------------|------|------|-------|------------|------|------|---------|---------|--------------------------|--------|--|-------------------|------|---------|-------------------------|----------|--|
| DOCUMENTS 4                                                                                                                                                                                                                                                                                                                                                                                                                                                                    | TOTAL SIZE 1.3KB | AVG. SIZE 323B | INDEXES 2                | TOTAL SIZE 56.0KB | AVG. SIZE 28.0KB |                       |      |      |       |            |      |      |         |         |                          |        |  |                   |      |         |                         |          |  |
| <a href="#">CREATE INDEX</a> <table border="1"> <thead> <tr> <th>Name and Definition ^</th> <th>Type</th> <th>Size</th> <th>Usage</th> <th>Properties</th> <th>Drop</th> </tr> </thead> <tbody> <tr> <td>_id_</td> <td>REGULAR </td> <td>36.9 KB</td> <td>12 since Sat Aug 22 2020</td> <td>UNIQUE </td> <td></td> </tr> <tr> <td>emailAddress_text</td> <td>TEXT </td> <td>20.5 KB</td> <td>1 since Sun Aug 23 2020</td> <td>COMPOUND </td> <td></td> </tr> </tbody> </table> |                  |                |                          |                   |                  | Name and Definition ^ | Type | Size | Usage | Properties | Drop | _id_ | REGULAR | 36.9 KB | 12 since Sat Aug 22 2020 | UNIQUE |  | emailAddress_text | TEXT | 20.5 KB | 1 since Sun Aug 23 2020 | COMPOUND |  |
| Name and Definition ^                                                                                                                                                                                                                                                                                                                                                                                                                                                          | Type             | Size           | Usage                    | Properties        | Drop             |                       |      |      |       |            |      |      |         |         |                          |        |  |                   |      |         |                         |          |  |
| _id_                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | REGULAR          | 36.9 KB        | 12 since Sat Aug 22 2020 | UNIQUE            |                  |                       |      |      |       |            |      |      |         |         |                          |        |  |                   |      |         |                         |          |  |
| emailAddress_text                                                                                                                                                                                                                                                                                                                                                                                                                                                              | TEXT             | 20.5 KB        | 1 since Sun Aug 23 2020  | COMPOUND          |                  |                       |      |      |       |            |      |      |         |         |                          |        |  |                   |      |         |                         |          |  |

8. Go to the Explain Plan tab and type in `{$text:$search:"gmail"}` in the Filter field. Have a look at the Query Performance Summary.

| Documents                                                                                                                                                                                                                                                                                                                          | Aggregations | Explain Plan | Indexes |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------|--------------|---------|
| <span> FILTER <code>{\$text:\$search:"gmail"}</code></span> <span> OPTIONS</span> <span> EXPLAIN</span> <span> RESET</span> <span>...</span>                                                                                                                                                                                       |              |              |         |
| <span>VIEW DETAILS AS</span> <span>VISUAL TREE</span> <span>RAW JSON</span>                                                                                                                                                                                                                                                        |              |              |         |
| <b>Query Performance Summary</b> <ul style="list-style-type: none"> <li> Documents Returned: 3</li> <li> Index Keys Examined: 3</li> <li> Documents Examined: 3</li> <li> Actual Query Execution Time (ms): 0</li> <li> Sorted in Memory: no</li> <li> Query used the following index:<br/><code>_fts text _ftsx</code></li> </ul> |              |              |         |

Write down your observation.

**The End**

# Tutorial 3

## MongoDB Basic

# SOLUTIONS

**Tutorial Objective:**

1. to become familiar with the MongoDB environment, in both
  - a. Mongo Shell and
  - b. MongoDB Compass,
2. to be able to do basic MongoDB CRUD Operations,
3. to assess student's knowledge on the basic MongoDB Operations.

**Important Reminder:**

- The unit consists of the Tutorial Work assessment (weighing 5%) conducted during week 4, 5, 7, 9, 10.
- **This tutorial work (weighing 1% out of 5%) is the first among the assessed tutorials works. It will be assessed during your tutorial time.**

**Marking Rubric:**

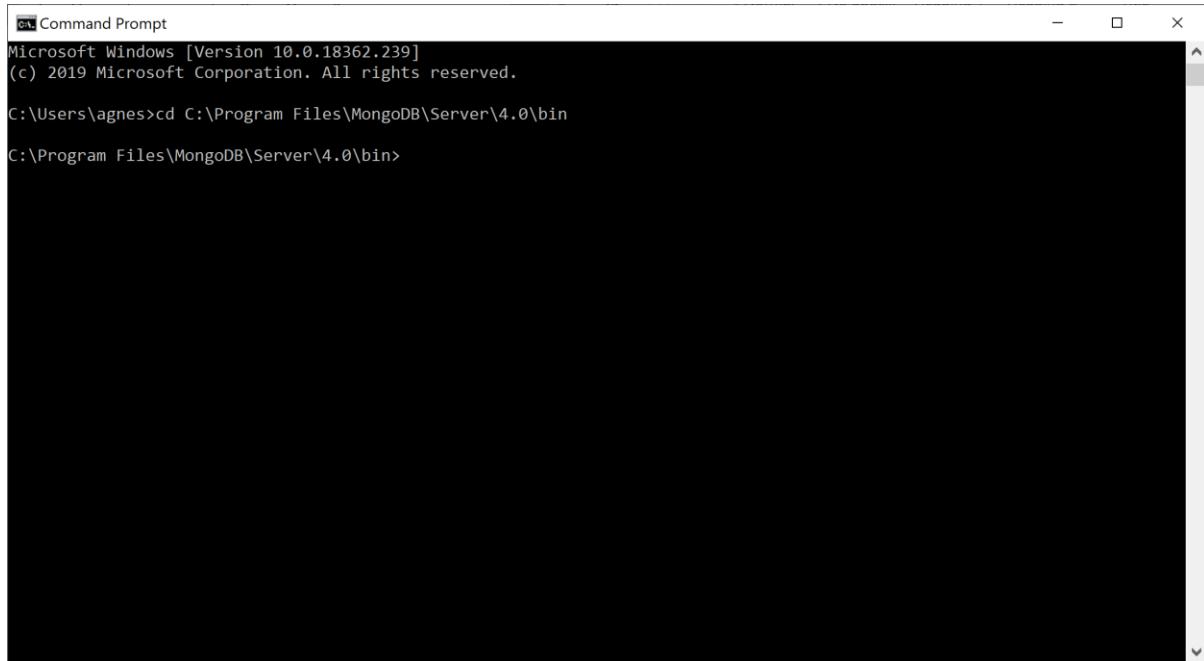
| Mark | Description            |                                               |                                                                                                                                                                                                                                                                                                                                                                                                        |
|------|------------------------|-----------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|      | Level of understanding | Preparation                                   | Tutorial Task and Interview Questions                                                                                                                                                                                                                                                                                                                                                                  |
| 1    | Exceptional            | Student has prepared prior to the tutorial    | Completed all of the tutorial tasks and has answered all interview questions without mistakes/errors                                                                                                                                                                                                                                                                                                   |
| 0.75 | Competent              | Student has prepared prior to the tutorial    | either<br>completed all of the tutorial tasks and has answered at least 50% of the interview questions without mistakes/errors<br><br>or:<br>completed at least 50% of the tutorial tasks and has answered all interview questions without mistakes/errors                                                                                                                                             |
| 0.5  | Moderate               | Student has prepared prior to the tutorial    | either:<br>completed more than 50% of the tutorial tasks and answered less than 50% of the interview questions without mistakes/errors<br><br>or:<br>completed less than 50% of the tutorial tasks and answered more than 50% of the interview questions without mistakes/errors<br><br>or:<br>completed 50% of the tutorial tasks and answered 50% of the interview questions without mistakes/errors |
| 0.25 | Inadequate             | Student has prepared prior to the tutorial    | completed less than 50% of the tutorial tasks and answered less than 50% of the interview questions without mistakes/errors                                                                                                                                                                                                                                                                            |
| 0    | No understanding       | Student did not prepare prior to the tutorial | did not complete any of the tutorial tasks and did not answer any interview questions without mistakes/errors                                                                                                                                                                                                                                                                                          |

## File Template for Answer

A file template (`Week4_MongoDBexercise.js`) is provided for you [on Moodle \(under the Week 4 resources\)](#) to answer the tutorial work. You can use any word editor application (e.g. [Atom](#), [Notepad++](#)) to edit your answer in the provided file template.

## Connecting to Mongo Shell on Windows

1. Press the **Windows Key** → type **Run** in the search bar → **Enter**.
2. Type **cmd** to open the **Command Prompt** → **Enter**.
3. Type “**cd C:\Program Files\MongoDB\Server\4.4\bin**” (the directory path might be different if you did not install MongoDB in C: drive).



The screenshot shows a Windows Command Prompt window titled "Command Prompt". The window title bar says "Command Prompt". The window content area shows the following text:  
Microsoft Windows [Version 10.0.18362.239]  
(C) 2019 Microsoft Corporation. All rights reserved.  
C:\Users\agnes>cd C:\Program Files\MongoDB\Server\4.0\bin  
C:\Program Files\MongoDB\Server\4.0\bin>

4. Type **mongod** to start MongoDB server.

```

C:\Command Prompt
Microsoft Windows [Version 10.0.18362.239]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\agnes>cd C:\Program Files\MongoDB\Server\4.0\bin

C:\Program Files\MongoDB\Server\4.0\bin>mongod
2019-07-29T01:44:08.884+1000 I CONTROL  [main] Automatically disabling TLS 1.0, to force-enable TLS 1.0 specify --sslDisabledProtocols 'none'
2019-07-29T01:44:08.889+1000 I CONTROL  [initandlisten] MongoDB starting : pid=48084 port=27017 dbpath=C:\data\db\ 64-bit host=DESKTOP-9J00055
2019-07-29T01:44:08.889+1000 I CONTROL  [initandlisten] targetMinOS: Windows 7/Windows Server 2008 R2
2019-07-29T01:44:08.889+1000 I CONTROL  [initandlisten] db version v4.0.9
2019-07-29T01:44:08.889+1000 I CONTROL  [initandlisten] git version: fc525e2d9b0e4bceff5c2201457e564362909765
2019-07-29T01:44:08.890+1000 I CONTROL  [initandlisten] allocator: tcmalloc
2019-07-29T01:44:08.890+1000 I CONTROL  [initandlisten] modules: none
2019-07-29T01:44:08.890+1000 I CONTROL  [initandlisten] build environment:
2019-07-29T01:44:08.890+1000 I CONTROL  [initandlisten]   distmod: 2008plus-ssl
2019-07-29T01:44:08.890+1000 I CONTROL  [initandlisten]   distarch: x86_64
2019-07-29T01:44:08.890+1000 I CONTROL  [initandlisten]   target_arch: x86_64
2019-07-29T01:44:08.893+1000 I CONTROL  [initandlisten] options: {}
2019-07-29T01:44:08.893+1000 I STORAGE  [initandlisten] exception in initAndListen: NonExistentPath: Data directory C:\data\db\ not found., terminating
2019-07-29T01:44:08.894+1000 I NETWORK  [initandlisten] shutdown: going to close listening sockets...
2019-07-29T01:44:08.894+1000 I CONTROL  [initandlisten] now exiting
2019-07-29T01:44:08.894+1000 I CONTROL  [initandlisten] shutting down with code:100

C:\Program Files\MongoDB\Server\4.0\bin>

```

## 5. Type mongo to run Mongo Shell.

```

C:\Command Prompt - mongo
C:\Program Files\MongoDB\Server\4.0\bin>mongo
MongoDB shell version v4.0.9
connecting to: mongod://127.0.0.1:27017/?gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("cb97d4ee-2927-40f0-adb3-6edc79200d18") }
MongoDB server version: 4.0.9
Server has startup warnings:
2019-07-25T04:01:03.901+1000 I CONTROL  [initandlisten]
2019-07-25T04:01:03.901+1000 I CONTROL  [initandlisten] ** WARNING: Access control is not enabled for the database.
2019-07-25T04:01:03.901+1000 I CONTROL  [initandlisten] **             Read and write access to data and configuration is unrestricted.
2019-07-25T04:01:03.902+1000 I CONTROL  [initandlisten]
---
Enable MongoDB's free cloud-based monitoring service, which will then receive and display metrics about your deployment (disk utilization, CPU, operation statistics, etc).

The monitoring data will be available on a MongoDB website with a unique URL accessible to you and anyone you share the URL with. MongoDB may use this information to make product improvements and to suggest MongoDB products and deployment options to you.

To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---

>

```

## Connecting to Mongo Shell on Mac

1. Run the Mongo daemon, in one terminal window  
> ~/mongodb/bin/mongod  
This will start the Mongo Server.
2. Open another terminal, run the mongo shell  
> ~/mongodb/bin/mongo

## MongoDB Basic – CRUD Exercise

### CREATE

1. Create a database called studentEnrolment.

```
use studentEnrolment
```

Create a collection called students.

```
db.createCollection("students")
```

2. Insert the following data into students collection:

| ID   | First name | Last name | Address            |                |       |          | Gender | Course    | Year | Off-Campus | Email Address                         |
|------|------------|-----------|--------------------|----------------|-------|----------|--------|-----------|------|------------|---------------------------------------|
|      |            |           | Street Address     | Suburb         | State | Postcode |        |           |      |            |                                       |
| 1001 | John       | Smith     | 123 Monash Drive   | Clayton        | VIC   | 3168     | Male   | MIT       | 2019 | false      | jsmith@gmail.com,<br>jsmith@yahoo.com |
| 1002 | Mary       | Citizen   | 900 Dandenong Road | Caulfield East | VIC   | 3145     | Female | MDS       | 2018 | true       | mcitizen@gmail.com                    |
| 1003 | Fred       | Bloggs    | 90 Wellington Road | Clayton        | VIC   | 3168     | Male   | MIT, MBIS | 2017 | false      | fredb@gmail.com                       |
| 1004 | Nick       | Nice      | 3 Robinson Avenue  | Kew            | VIC   | 3080     | Male   | MCS       | 2018 | false      | nicknice@gmail.com                    |
| 1005 | Wendy      | Wheat     | 6 Algorithm Street | Malvern        | VIC   | 3144     | Female | GDS       | 2019 | true       | wwheat@yahoo.com                      |

```
db.students.insert({
  "_id": 1001,
  "firstName": "John",
  "lastName": "Smith",
  "address": {
    "streetaddress": "123 Monash Drive",
    "suburb": "Clayton",
    "state": "VIC",
    "postcode": 3168
  },
  "gender": "male",
  "course": "MIT",
  "year": 2019,
  "offcampus": false,
  "emailaddress": [
    "jsmith@gmail.com",
    "jsmith@yahoo.com"
  ]
})
```

```
db.students.insert({
  "_id": 1002,
  "firstName": "Mary",
  "lastName": "Citizen",
  "address": {
    "streetaddress": "900 Dandenong Road",
    "suburb": "Caulfield East",
    "state": "VIC",
```

```

        "postcode": 3145
    },
    "gender": "female",
    "course": "MDS",
    "year": 2018,
    "offcampus": true,
    "emailaddress": "mcitizen@gmail.com"
})

db.students.insert({
    "_id": 1003,
    "firstName": "Fred",
    "lastName": "Bloggs",
    "address": {
        "streetaddress": "90 Wellington Road",
        "suburb": "Clayton",
        "state": "VIC",
        "postcode": 3168
    },
    "gender": "male",
    "course": [
        "MIT",
        "MBIS"
    ],
    "year": 2017,
    "offcampus": false,
    "emailaddress": "fredb@gmail.com"
})

db.students.insert({
    "_id": 1004,
    "firstName": "Nick",
    "lastName": "Nice",
    "address": {
        "streetaddress": "3 Robinson Avenue",
        "suburb": "Kew",
        "state": "VIC",
        "postcode": 3080
    },
    "gender": "male",
    "course": "MCS",
    "year": 2018,
    "offcampus": false,
    "emailaddress": "nicknice@gmail.com"
})

db.students.insert({
    "_id": 1005,
    "firstName": "Wendy",
    "lastName": "Wheat",
    "address": {
        "streetaddress": "6 Algorithm Street",
        "suburb": "Malvern",
        "state": "VIC",
        "postcode": 3144
    },
    "gender": "female",
    "course": "GDS",
    "year": 2019,
    "offcampus": true,
    "emailaddress": "wwheat@yahoo.com"
})

```

## READ

1. List all students in the `students` collection.

```
db.students.find()
db.students.find().pretty()
```

2. Find student who are doing MIT course.

```
db.students.find({"course": "MIT"})
db.students.find({"course": "MIT"}).pretty()
```

3. List all students who live in Clayton.

```
db.students.find({"address.suburb": "Clayton"})
db.students.find({"address.suburb": "Clayton"}).pretty()
```

4. List all male students who are enrolled after 2017.

```
db.students.find(
    {"gender": "male", "year": {$gt:2017}}
).pretty()
```

5. Show all students who live in Malvern or Caulfield.

```
db.students.find(
    {$or: [
        { "address.suburb": "Malvern" },
        { "address.suburb": "Caulfield" }
    ]
}
```

6. Display the name of all students and their course.

```
db.students.find(
    { },
    {"firstName":1, "lastName":1, "course":1}
)
```

7. Count how many students who are doing MBIS. (Hint: `count()`)

```
db.students.find({"course": "MBIS"}).count()
```

8. Count the number of students who are enrolled in between 2018 to 2019.

```
db.students.find(
    {"year": {$gte:2018, $lte:2019} }
).count()
```

9. Find 3 students who are enrolled before 2019 and they are doing either MIT or MDS.

```
db.students.find(
    {"year": {$lt: 2019}, "course": {$in: ["MIT", "MDS"]}})
```

```
).limit(3)
```

10. Show students who are not doing MIT. Display only the names of these students and sort their names in ascending order.

```
db.students.find(
  {"course": {$ne: "MIT"}},
  {"firstName": 1, "lastName": 1, "_id": 0}
).sort({"firstName": 1})
```

11. Find students who are doing double degree in both MIT and MBIS.

```
db.students.find({"course": ["MIT", "MBIS"]})
```

## UPDATE

1. Add Wendy Wheat's birthday, which is on 30<sup>th</sup> October 1990.

```
db.students.updateOne(
  { "_id": 1005 },
  {
    $set: { "birthday": new Date("1990-10-30") }
  }
)
```

2. If you stored email address for Nick Nice in a string, change it to an Array and add a new email address for Nick Nice: nnice@gmail.com.

```
db.students.updateOne(
  { "_id": 1004 },
  {
    $set: {
      "emailaddress": ["nicknice@gmail.com", "nnice@gmail.com"]
    },
    $currentDate: { lastModified: true }
  }
)
```

3. Add another email address for Nick Nice: nnice@monash.edu. (Hint: \$push)

```
db.students.update(
  { "_id": 1004 },
  { $push: { "emailaddress": "nnice@monash.edu" } }
)
```

4. Find students who are doing MIT course and add these units:

- Unit Code: FIT5137, Unit Name: Advanced Database Technology
- Unit Code: FIT9132, Unit Name: Introduction to Databases

You have to ensure that each unit is stored as an object.

```
db.students.updateMany(
  { "course": "MIT" },
  {
    $set: {
      "units": [
        {
          "unitCode": "FIT5137",
          "unitName": "Advanced Database Technology"
        },
        {
          "unitCode": "FIT9132",
          "unitName": "Introduction to Databases"
        }
      ],
      $currentDate: { lastModified: true }
    }
  )
)
```

5. Remove the birthday field that we entered for Wendy Wheat. (Hint: \$unset)

```
db.students.updateOne(
  { "_id": 1004 },
  {
    $unset: { "birthday": "" }
  }
)
```

6. Replace the details of Fredd Bloggs to the following:

Student ID: 1003

First name: Fred

Last name: Bloggs

Gender: Male

Course:

1. Course name: MIT, Year: 2017
2. Course name: MBIS, Year: 2018

Off-campus: false

Email: fredb@gmail.com

```
db.students.replaceOne(
  { "_id": 1003 },
  {
    "_id": 1003,
    "firstName": "Fred",
    "lastName": "Bloggs",
    "gender": "male",
    "course": [
      {
        "courseName": "MIT",
        "year": 2017
      },
      {
        "courseName": "MBIS",
        "year": 2018
      }
    ]
  }
)
```

```
        "courseName": "MBIS",
        "year": 2018
    }
],
"offcampus": false,
"emailaddress": "fredb@gmail.com"
}
)
```

## DELETE

1. Delete students who are enrolled before 2019 and are off-campus students.

```
db.students.deleteMany(
  {"year": {$lt:2019}, "offcampus": true}
)
```

## The End



MONASH  
University

MONASH  
INFORMATION  
TECHNOLOGY

# FIT5137 – Advanced Database Technology

Week 3 – MongoDB

Semester 2, 2020

Developed by:

Dr. Agnes Haryanto

[Agnes.Haryanto@monash.edu](mailto:Agnes.Haryanto@monash.edu)

# Agenda

## 1. Mongo Shell

- CRUD Operations – revisit

## 2. Indexing

## 3. Aggregation

# Using FLUX

1. Visit <http://flux.qa/> on your internet enabled device
2. Log in using your Monash account (not required if you are already logged in to Monash)
3. Click on the “+” to join audience
4. Enter the Audience Code: **F2LU2L**
5. Select FIT5137 in the Active Presentation menu
6. Answer questions when they pop up

Alternatively, the quiz can be accessed through:

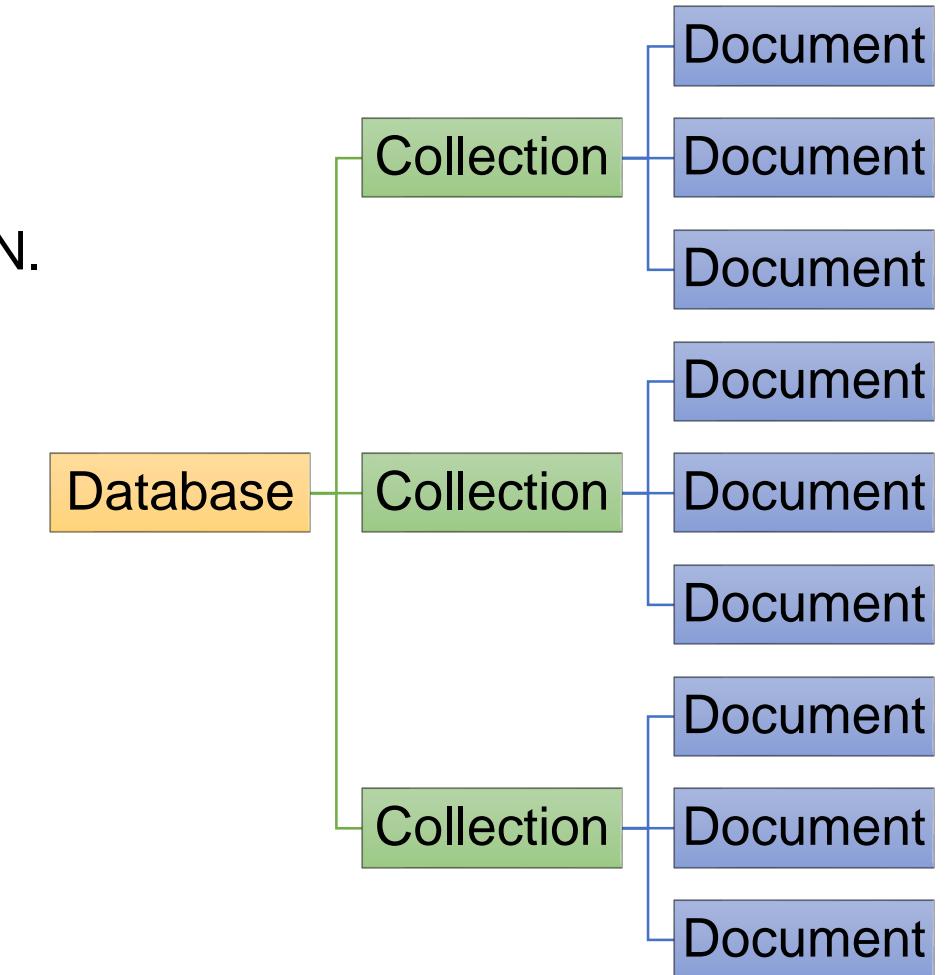
<https://flux.qa/F2LU2L>

The screenshot shows a web browser window titled "FLUX" with the URL <https://flux.qa/#/feeds/5d368...>. The page is titled "Lecture 0". Below the title are four icons: a bar chart, a video camera, a trophy, and a clock. The main content area is titled "Current Polls". It displays a multiple-choice poll with the question "What did you have for dinner?". The options are listed in a vertical list:

- A Pasta
- B Rice
- C Pizza
- D Salad
- E Nothing

# MongoDB

- **Document:** the basic unit of data in MongoDB.
  - Similar to a row in relational database.
  - Documents are structured in BSON – similar to JSON.
    - A data structure made up of **field** and **value** pairs.
- **Collection:** a grouping of MongoDB documents.
  - Similar to a table in relational database.
- **Database:** a physical container for collections.



# Mongo Shell

# Explore Databases and Collections

- To show active database
  - db
- To list all databases
  - show dbs
- Change database
  - use <database\_name>
- To show all collections in database
  - show collections

# Collections

- Create a new collection
  - `db.createCollection("<collection name>")`
- List all collections
  - `show collections`
- Delete a collection
  - `db.getCollection("<collection name>").drop()`
  - `db.<collection name>.drop()`
- Delete database
  - `db.dropDatabase()`

# MongoDB CRUD Operations

- **Create** – add new documents to a collection.
- **Read** – retrieve documents from a collection.
- **Update** – modify existing documents in a collection.
- **Delete** – remove documents from a collection.

# MongoDB CRUD

## CREATE

# CRUD – Insert

- **Insert One or Many Documents**

- `db.<collection name>.insert (<Object> or <Array of Objects>)`

- **Insert One Document**

- `db.<collection name>.insertOne (<Object>)`

- **Insert Many Documents**

- `db.<collection name>.insertMany (<Array of Objects>)`

# CRUD – Insert Behavior

- Collection Creation
  - When inserting a document into a collection that does not currently exist, the insert operations will also create the collection.
- `_id` Field
  - Each document stored in a collection requires a unique `_id` field, which is similar to a primary key.
  - When inserting a document without the `_id` field, MongoDB will automatically assign an `ObjectID` for the `_id` field.
- Atomicity
  - All write operations in MongoDB are atomic on the level of a single document.

# CRUD – Insert One Document

- To insert a single document:

- db.<collection name>.insertOne(<Object>)

- For example:

- db.inventory.insertOne({  
    item: "canvas",  
    qty: 100,  
    tags: ["cotton"],  
    size: { h: 28, w: 35.5, uom: "cm" }  
})

# CRUD – Insert One Document

- To insert a single document:

- db.<collection name>.insertOne(<Object>)

- For example – with ID:

- db.inventory.insertOne({  
    \_id:1,  
    item: "mat",  
    qty: 50,  
    size: { h: 20, w: 23.5, uom: "cm" }  
})

# CRUD – Insert One Document

- For example:

```
➤ db.inventory.insertOne ({  
    item: "pen",  
    qty: 100,  
    tags: ["blue", "black", "red"],  
    size: { h: 14, w: 2.5, uom: "cm" },  
    purchaseDate: new Date("2019-08-11")  
})
```

To insert **Date**  
data type

# CRUD – Insert Many Documents

- To insert multiple documents:

- db.<collection name>.insertMany(<Array of Objects>)

- For example:

- db.inventory.insertMany([  
    { item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status: "A" },  
    { item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" }, status: "A" },  
    { item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" }, status: "D" },  
    { item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" }, status: "D" },  
    { item: "postcard", qty: 45, size: { h: 10, w: 15.25, uom: "cm" }, status: "A" }  
]);

# MongoDB CRUD READ

# **CRUD – Read**

- **Find Documents**

- `db.<collection name>.find(<query>, <fields>)`

- **Find One Document**

- `db.<collection name>.findOne(<query>, <fields>)`

# CRUD – Find All Documents

- Select all documents in a collection:
  - db.inventory.find( )
  - db.inventory.find( {} )
  - db.inventory.find( {} ).pretty()
- SQL:
  - SELECT \* FROM inventory

# CRUD – Query Documents

- Specify equality condition – by ID:

➤ db.inventory.find( { \_id: 1 } )

- SQL:

➤ SELECT \* FROM inventory WHERE \_id = 1

# CRUD – Query Documents

- Specify equality condition:

- db.inventory.find( { status: "D" } )

- SQL:

- SELECT \* FROM inventory WHERE status = "D"

# CRUD – Query Documents

- Specify conditions using query operators:

➤ db.inventory.find( { status: { \$in: [ "A", "D" ] } } )

- SQL:

➤ SELECT \* FROM inventory WHERE status in ("A", "D")

# CRUD – Query Documents

- Specify **AND** conditions:

- db.inventory.find( { status: "A", qty: { \$lt: 30 } } )

- Specify **AND** conditions:

- db.inventory.find( { \$and: [ {status: "A"}, {qty: { \$lt: 30 }} ] } )

- SQL:

- SELECT \* FROM inventory WHERE status = "A" AND qty < 30

# CRUD – Query Documents

- Specify OR conditions:

- db.inventory.find(  
    { \$or: [ { status: "A" }, { qty: { \$lt: 30 } } ] }

- SQL:

- SELECT \* FROM inventory WHERE status = "A" OR qty < 30

# CRUD – Query Documents

- Specify **AND** as well as **OR** conditions:

```
➤ db.inventory.find( {  
    status: "A",  
    $or: [ { qty: { $lt: 30 } }, { item: /^p/ } ]  
} )
```

- SQL:

```
➤ SELECT * FROM inventory WHERE status = "A" AND ( qty <  
30 OR item LIKE "p%")
```

# CRUD – Query Documents

- Match an embedded/nested document:

➤ `db.inventory.find( { size: { h: 14, w: 21, uom: "cm" } } )`

- Equality matches on embedded document require an exact match of the specified <value> document, including the field order.

The example below is not the same as the example above:

➤ `db.inventory.find( { size: { w: 21, h: 14, uom: "cm" } } )`

# CRUD – Query Documents

- Specify equality match on a nested field:

- db.inventory.find( { "size.uom": "in" } )

- Specify match using query operator on a nested field:

- db.inventory.find( { "size.h": { \$lt: 15 } } )

- Specify AND condition on a nested field:

- db.inventory.find( { "size.h": { \$lt: 15 }, "size.uom": "in", status: "D" } )

# CRUD – Query Documents

- Match an array:

- db.inventory.find( { tags: ["red", "blank"] } )

- Query an array for an element:

- db.inventory.find( { tags: "red" } )

- Query for an element by the array index position:

- db.inventory.find( { "dim\_cm.1": { \$gt: 25 } } )

- Query an array by array length:

- db.inventory.find( { "tags": { \$size: 3 } } )

# CRUD – Query Documents (Projection)

- Project all:
  - `db.<collection name>.find( { }, { } )`
- Project certain field:
  - `db.<collection name>.find( { }, { KEY:1 } )`
- Note: the `_id` is always included in the projection by default.

# CRUD – Query Documents (Projection)

- For example:
  - db.inventory.find( { }, {item:1, qty:1} )
- SQL:
  - SELECT \_id, item, qty FROM inventory
- For example (to omit the \_id):
  - db.inventory.find( { }, {item:1, qty:1, \_id:0} )
- SQL:
  - SELECT item, qty FROM inventory

# CRUD – Query Documents (Limit & Skip)

- Limit the records to be displayed:

- db.<collection name>.find( ).limit( <number> )

- Skip records:

- db.<collection name>.find( ).skip( <number> )

# CRUD – Query Documents (Sort)

- Sort the documents to be displayed:
  - `db.<collection name>.find( ).sort( {KEY:1} )`
  - `db.<collection name>.find( ).sort( {KEY:-1} )`
- Note: 1 is for ascending order, -1 is for descending order

# MongoDB CRUD UPDATE

# **CRUD – Update**

- **Update One Document**
  - `db.<collection name>.updateOne(<filter>, <update>, <options>)`
- **Update Many Documents**
  - `db.<collection name>.updateMany(<filter>, <update>, <options>)`
- **Replace a single document within the collection based on filter**
  - `db.<collection name>.replaceOne(<filter>, <update>, <options>)`

# CRUD – Update

- Update One Document

- db.<collection name>.updateOne(<filter>, <update>, <options>)

- For example:

- db.inventory.updateOne(  
    { item: "paper" } ,  
    {  
        \$set: { "size.uom": "cm", status: "P" } ,  
        \$currentDate: { lastModified: true }  
    }  
)

# CRUD – Update

- Update Many Documents

- db.<collection name>.updateMany(<filter>, <update>, <options>)

- For example:

- db.inventory.updateMany(  
    { "qty": { \$lt: 50 } },  
    {  
        \$set: { "size.uom": "in", status: "P" },  
        \$currentDate: { lastModified: true }  
    }  
)

# CRUD – Update

- Replace a single document within the collection based on filter
  - db.<collection name>.replaceOne(<filter>, <update>, <options>)
- For example:
  - db.inventory.replaceOne(

```
{ item: "paper" },  
{ item: "paper", instock: [ { warehouse: "A", qty: 60 }, {  
warehouse: "B", qty: 40 } ] }
```

)

# MongoDB CRUD

## DELETE

# CRUD – Delete

- Delete One Document
  - db.<collection name>.deleteOne()
- Delete Many Documents
  - db.<collection name>.deleteMany()
- Delete All Documents
  - db.<collection name>.deleteMany( {} )

# CRUD – Delete

- Delete One Document

- db.<collection name>.deleteOne(<filter>)

- For example:

- db.inventory.deleteOne( { status: "D" } )

# CRUD – Delete

- Delete Many Documents

- db.<collection name>.deleteMany(<filter>)

- For example:

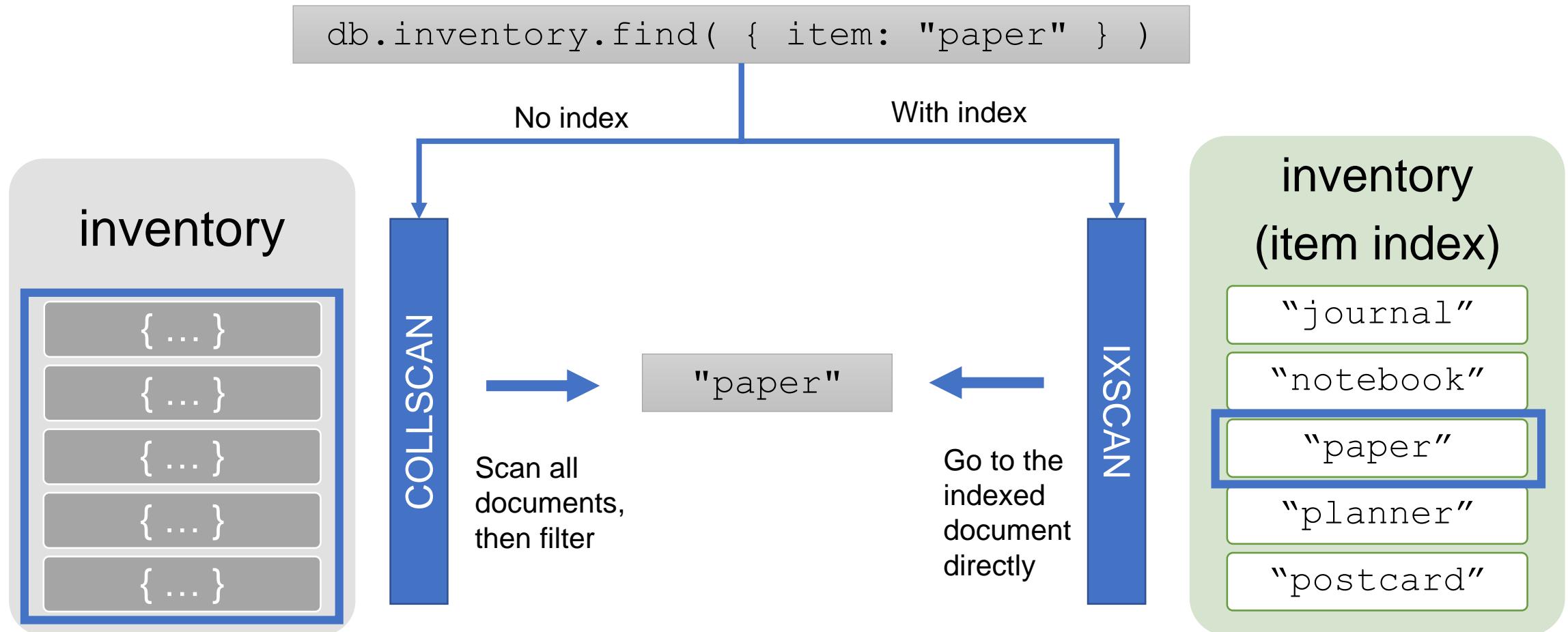
- db.inventory.deleteMany({ status : "A" })

# MongoDB Indexing

# Indexing

- MongoDB have to do *collection scan* (to scan every document in a collection) to retrieve documents that match user's query statement.
- Through indexing, query processing can be accelerated.
- Indexes support efficient data retrieval as MongoDB can use the index to limit the number of documents it must inspect.

# Indexing



# Indexing

- To create an index:
  - `db.<collection name>.createIndex( { KEY:1 } )`
- For example:
  - `db.contacts.createIndex( { "dob.age":1 } )`
- Note: to index in ascending order, use 1. To index in descending order, use -1.

# Indexing

- To remove an index:

- `db.<collection name>.dropIndex( { KEY:1 } )`

- For example:

- `db.contacts.dropIndex( { "dob.age":1 } )`

- To get all existing indexes:

- `db.<collection name>.getIndexes( )`

- For example:

- `db.contacts.getIndexes( )`

# Indexing

- To create a Compound Index:
  - `db.<collection name>.createIndex( { KEY1:1, KEY2:1 } )`
- For example:
  - `db.contacts.createIndex( { "dob.age":1, gender:1 } )`
- Note: this will create a single index (not two indices) where every value is connected based on the specified keys.

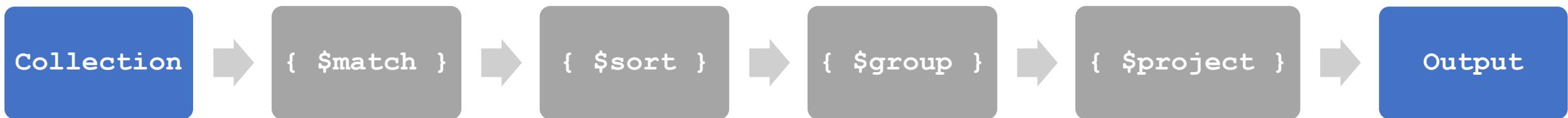
# MongoDB Aggregation

# Aggregation

- Aggregation operations process data records and return computed results.
- Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result.
- MongoDB provides three ways of aggregation:
  - Aggregation pipeline
  - Map-reduce function
  - Single purpose aggregation methods

# Aggregation Pipeline

- MongoDB's aggregation framework is modeled on the concept of data processing pipelines.
- Documents enter a multi-stage pipeline that transforms the documents into an aggregated result.



# Aggregation Pipeline

- Characteristics:
  - Every stage receives the output of the previous stage.
  - Should be in an array as there are certain steps to be followed.
- Syntax:
  - `db.<collection>.aggregate( [ { <stage> } , ... ] )`

# Aggregation Pipeline Stages

- \$match – Filters the documents to pass only the documents that match the specified condition(s) to the next pipeline stage
- \$match:
  - { \$match: { <query> } }
- For example:
  - db.persons.aggregate([  
    { \$match: {gender: "female"} }  
]) .pretty();

# Aggregation Pipeline Stages

- \$group – groups documents by some specified expression and outputs to the next stage a document for each distinct grouping.
- \$group :
  - { \$group: { \_id: <expression>, <field1>: { <accumulator1> : <expression1> }, ... } }
- Accumulator for \$group aggregation:
  - \$sum, \$avg, \$max, \$min, \$first, \$last, etc. ...
- For example:
  - db.persons.aggregate([  
    { \$match: { gender: "female" } },  
    { \$group: { \_id: { state: "\$location.state" }, totalPersons: {  
        \$sum: 1 } } }  
]) .pretty();

# Aggregation Pipeline Stages

- \$sort – sorts all input documents and returns them to the pipeline in sorted order.
- \$sort:
  - { \$sort: { <field1>: <sort order>, <field2>: <sort order> ... } }
- For example:
  - db.persons.aggregate([

```
{ $match: { gender: "female" } },
{ $group: { _id: { state: "$location.state" } ,
  totalPersons: { $sum: 1 } } } ,
{ $sort: { totalPersons: -1 } }
]) .pretty();
```

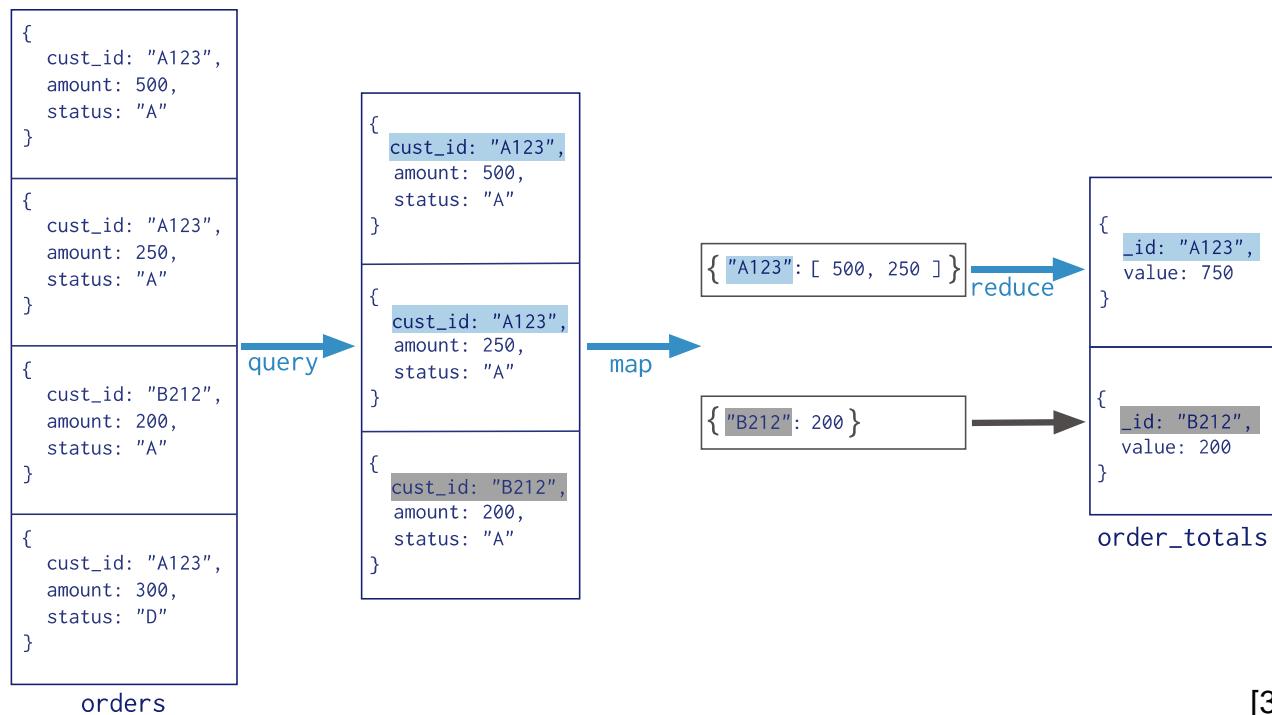
# Aggregation Pipeline Stages

- \$project – Passes along the documents with the requested fields to the next stage in the pipeline.
- \$project:
  - { \$project: { <specification(s)> } }
- For example:
  - db.persons.aggregate([  
    { \$project: {  
        \_id: 0, name: 1, email: 1, birthdate: { \$convert: {  
            input: '\$dob.date', to: 'date' } }, age: "\$dob.age" } }  
  ]).pretty();

# Map-Reduce

- Map-reduce uses custom JavaScript functions to perform the map and reduce operations
- Map-reduce operations have two phases:
  - A **map** stage that processes each document and *emits* one or more objects for each input document.
  - **Reduce** phase that combines the output of the map operation.
  - Optionally, map-reduce can have a *finalize* stage to make final modifications to the result.

```
Collection  
db.orders.mapReduce(  
  map → function() { emit( this.cust_id, this.amount ); },  
  reduce → function(key, values) { return Array.sum( values ) },  
  query → { query: { status: "A" },  
  output → out: "order_totals"  
 }  
)
```



# Single Purpose Aggregation Operations

- MongoDB also provides:
  - `db.<collection>.estimatedDocumentCount()`
  - `db.<collection>.count()`
  - `db.<collection>.distinct()`
- All of the above operations aggregate documents from a single collection.

# References

- [1] Academind, “MongoDB – The Complete Developer’s Guide”  
<https://www.academind.com/learn/mongodb/>.
- [2] K. Chodorow, MongoDB: The Definitive Guide. Sebastopol, CA, USA: O’Reilly Media, Inc., 2<sup>nd</sup> ed., 2013.
- [3] MongoDB, “Aggregation” <https://docs.mongodb.com/manual/aggregation/>.
- [4] MongoDB, “Aggregation Pipeline Stages”  
<https://docs.mongodb.com/manual/reference/operator/aggregation-pipeline/>.
- [5] MongoDB, “Indexes” <https://docs.mongodb.com/manual/indexes/>.
- [6] MongoDB, “MongoDB CRUD Operations”  
<https://docs.mongodb.com/manual/crud/>.

## Tutorial 2

### MongoDB Compass

# SOLUTIONS

The objective of this tutorial is to understand how MongoDB works through the usage of MongoDB Compass. You are also expected to be able to do MongoDB CRUD Operations on MongoDB Compass.

### CRUD Exercise

1. Open MongoDB Compass, then Connect to localhost.

| Database Name | Storage Size | Collections | Indexes |                                                                                     |
|---------------|--------------|-------------|---------|-------------------------------------------------------------------------------------|
| admin         | 16.0KB       | 0           | 1       |  |
| config        | 24.0KB       | 0           | 2       |  |
| local         | 36.0KB       | 1           | 1       |  |

2. Add new database called `restaurantdb` and a collection called `restaurant`.

3. Insert the following data into `restaurant` collection:

- i. Name : Theio Theo  
Address : 5/7, Waverley Road, Malvern East VIC 3145, Melbourne  
Phone number : 03 8657 8585  
Cuisines : Greek, Healthy Food  
Average cost (\$) : 100  
Opening Hours : 5pm-10pm  
BYO : true
- ii. Name : Portofino Pizza  
Address : 884 Glen Huntly Road, Caulfield VIC 3145, Melbourne  
Phone number : 03 9528 6693  
Cuisines : Pizza, Italian  
Average cost (\$) : 60  
Opening Hours : 5pm-10pm  
BYO : true
- iii. Name : The Cheesecake Shop  
Address : 882 Glen Huntly Road, Caulfield VIC 3145, Melbourne  
Phone number : 03 9528 5203  
Cuisines : Desserts  
Average cost (\$) : 20  
Opening Hours : 9am-7.30pm  
BYO : false
- iv. Name : Cool Rays  
Address : 71 Terry Street, Healesville VIC 3777  
Phone number : 03 9876 1234  
Cuisines : Desserts  
Average cost (\$) : 20  
Opening Hours : 9am-7.30pm

- BYO : false
- v. Name : Humble Rays  
 Address : 71 Bouverie Street, Carlton VIC 3053, Melbourne  
 Phone number : 03 8657 8526  
 Cuisines : Coffee and Tea, Cafe Food, Asian Fusion  
 Average cost (\$) : 50  
 Opening Hours : 8am-4pm  
 BYO : false
- vi. Name : The Crux & Co  
 Address : 35 Albert Road, South Melbourne VIC 3205, Melbourne  
 Phone number : 03 9820 1081  
 Cuisines : Coffee and Tea, Cafe Food  
 Average cost (\$) : 60  
 Opening Hours : 8am-4pm  
 BYO : false
- vii. Name : Advieh  
 Address : 71B Gamon Street, Seddon VIC 3011, Melbourne  
 Phone number : 03 8592 7384  
 Cuisines : Middle Eastern, Coffee and Tea, Cafe Food  
 Average cost (\$) : 50  
 Opening Hours : 8am-3:30pm, 6pm-10pm  
 BYO : false
- viii. Name : Suda  
 Address : Shop C2, 550 Lonsdale Street, CBD, VIC 3000  
 Phone number : 03 9942 6422  
 Cuisines : Korean, Asian  
 Average cost (\$) : 60  
 Opening Hours : 12noon – 2:30pm, 5pm – 10pm  
 BYO : false
- ix. Name : Devon  
 Address : 76 Devonshire Street, Surry Hills NSW 2010, Sydney  
 Phone number : 02 9211 8777  
 Cuisines : Coffee and Tea, Cafe Food  
 Average cost (\$) : 60  
 Opening Hours : 8am-3.30pm  
 BYO : true
- x. Name : The Grounds of Alexandria Cafe  
 Address : Shop 7A, 2 Huntley Street, Alexandria NSW 2015, Sydney  
 Phone number : 02 9699 2225  
 Cuisines : Coffee and Tea, Salad, Poké, Cafe Food  
 Average cost (\$) : 80  
 Opening Hours : 7.30am-4pm  
 BYO : false
- xi. Name : Chefs Gallery  
 Address : Shop 12, Regent Place Arcade, 501 George Street, CBD NSW 2000, Sydney  
 Phone number : 02 9267 8877  
 Cuisines : Chinese, Asian  
 Average cost (\$) : 70  
 Opening Hours : 11.30am-10pm  
 BYO : true  
 Known for : Unique Chinese dining experience with handmade noodles, dim sum and chapas (Chinese style tapas)
- xii. Name : Ciao Italia  
 Address : 273 Mill Point Road, South Perth, Perth, WA 6151  
 Phone number : 08 9368 5500

```
Cuisines      : Italian, Seafood
Average cost ($) : 65
Opening Hours   : 5pm-10pm
BYO           : true
```

Note: some data can be stored as an object or array. For instance, the address can be stored as an object as shown below.

```
_id: ObjectId("5d451b986b3d34f35a085609")
name: "Humble Rays"
✓ address: Object
  street: "71 Bouverie Street"
  suburb: "Carlton"
  state: "VIC"
  postcode: 3053
  phone: "03 8354 8459"
✓ cuisines: Array
  0: "Coffee and Tea"
  1: "Cafe Food"
  2: "Asian Fusion"
averagecost: 50
opening_hours: "8am-4pm"
byo: false
```

The following shows another possibility of how you can store the data (note the openingHours).

```
_id: ObjectId("5f3e25eb27a68f4dc421b58b")
name: "Suda"
✓ address: Object
  street: "Shop C2, 550 Lonsdale Street"
  suburb: "CBD"
  state: "VIC"
  postcode: 3000
  phone: "03 9942 6422"
✓ cuisines: Array
  0: "Korean"
  1: "Asian"
  averageCost: 60
✓ openingHours: Array
  ✓ 0: Object
    openingTime: "12noon"
    closingTime: "2:30pm"
  ✓ 1: Object
    openingTime: "5pm"
    closingTime: "10pm"
byo: false
```

#### 4. Change the view to TABLE.

|    | <code>_id</code>                      | <code>name</code>                | <code>address</code> | <code>phone</code> | <code>cuisines</code> |
|----|---------------------------------------|----------------------------------|----------------------|--------------------|-----------------------|
| 1  | <code>5d451b986b3d34f35a085605</code> | "Theio Theo"                     | { } 4 fields         | "03 8657 8585"     | [ ] 2 elements        |
| 2  | <code>5d451b986b3d34f35a085606</code> | "Portofino Pizza"                | { } 4 fields         | "03 9528 6693"     | [ ] 2 elements        |
| 3  | <code>5d451b986b3d34f35a085607</code> | "The Cheesecake Shop"            | { } 4 fields         | "03 9528 5203"     | [ ] 1 elements        |
| 4  | <code>5d451b986b3d34f35a085608</code> | "Cool Rays"                      | { } 4 fields         | "03 9876 1234"     | [ ] 1 elements        |
| 5  | <code>5d451b986b3d34f35a085609</code> | "Humble Rays"                    | { } 4 fields         | "03 8354 8459"     | [ ] 3 elements        |
| 6  | <code>5d451b986b3d34f35a08560a</code> | "The Crux & Co"                  | { } 4 fields         | "03 9820 1081"     | [ ] 2 elements        |
| 7  | <code>5d451b986b3d34f35a08560b</code> | "Advieh"                         | { } 4 fields         | "03 8592 7384"     | [ ] 3 elements        |
| 8  | <code>5d451b986b3d34f35a08560c</code> | "Suda"                           | { } 4 fields         | "03 9942 6422"     | [ ] 2 elements        |
| 9  | <code>5d451b986b3d34f35a08560d</code> | "Devon"                          | { } 4 fields         | "02 9211 8777"     | [ ] 2 elements        |
| 10 | <code>5d451b986b3d34f35a08560e</code> | "The Grounds of Alexandria Cafe" | { } 4 fields         | "02 9699 2225"     | [ ] 4 elements        |
| 11 | <code>5d451b986b3d34f35a08560f</code> | "Chefs Gallery"                  | { } 4 fields         | "02 9267 8877"     | [ ] 2 elements        |
| 12 | <code>5d451b986b3d34f35a085610</code> | "Ciao Italia"                    | { } 4 fields         | "08 9368 5500"     | [ ] 2 elements        |

Write down your observation(s).

#### 5. Read Operation.

##### 5.1. Show all restaurants.

```
{ }
```

##### 5.2. Show restaurants that do not provide BYO.

```
{ "byo" : false }
```

##### 5.3. List all restaurants that are in VIC.

```
{ "address.state" : "VIC" }
```

##### 5.4. Find restaurants that serve Italian cuisine.

```
{ "cuisines" : "Italian" }
```

##### 5.5. Display the restaurant's name and cuisines for all restaurants.

FILTER : { }

PROJECT : { name:1, cuisines:1 }

- 5.6. Display the restaurant's name, address and cuisines, but exclude the ID, for all restaurants.

FILTER : { }

PROJECT : { name:1, address:1, cuisines:1, \_id:0 }

- 5.7. Display the restaurant's name, suburb, postcode and phone number, but exclude the ID, for all restaurants.

FILTER : { }

PROJECT : { name:1, "address.suburb":1,  
"address.postcode":1, phone:1, \_id:0 }

- 5.8. Show 5 restaurants with the restaurant's name, average cost, cuisines and phone number, but exclude the ID.

FILTER : { }

PROJECT : { name:1, averageCost:1, cuisines:1, phone:1,  
\_id:0 }

LIMIT : 5

- 5.9. Display the restaurant's name, suburb, average cost, cuisines and phone number, but exclude the ID, for all restaurants in VIC.

FILTER : { "address.state" : "VIC" }

PROJECT : { name:1, "address.suburb":1, averageCost:1,  
cuisines:1, phone:1, \_id:0 }

- 5.10. Display the restaurant's name, suburb, average cost, cuisines and phone number, but exclude the ID, for all restaurants in VIC.

FILTER : { "address.state" : "VIC" }

PROJECT : { name:1, "address.suburb":1, averageCost:1,  
cuisines:1, phone:1, \_id:0 }

- 5.11. Show 5 restaurants in VIC with the restaurant's name, suburb, average cost, cuisines and phone number, but exclude the ID.

FILTER : { "address.state" : "VIC" }

PROJECT : { name:1, "address.suburb":1, averageCost:1,  
cuisines:1, phone:1, \_id:0 }

LIMIT : 5

- 5.12. Display the next 5 restaurants in VIC after skipping first 5 (from the previous question).

FILTER : { "address.state" : "VIC" }

PROJECT : { name:1, "address.suburb":1, averageCost:1, cuisines:1, phone:1, \_id:0 }

SKIP : 5

LIMIT : 5

- 5.13. Find restaurants with the average cost less than \$50.

{ averageCost: { \$lt: 50 } }

- 5.14. Find restaurants with the average cost more than \$50 but less than equal to \$100.

{ averageCost: { \$gt: 50, \$lte: 100 } }

- 5.15. Find the restaurants that serve “Coffee and Tea” and their average cost is less than \$60.

{ cuisines: "Coffee and Tea", averageCost: { \$lt: 60 } }

- 5.16. Find the restaurants that do not serve any cuisine of “Café Food” and their average cost is over \$20.

{ cuisines: { \$ne : "Cafe Food"}, averageCost: { \$gt: 20 } }

- 5.17. Find the restaurants that do not serve any cuisine of “Café Food” and their average cost is over \$20 and they are not located in VIC.

{ cuisines: { \$ne: "Cafe Food"}, averageCost: { \$gt: 20 }, "address.state": { \$ne: "VIC" } }

- 5.18. Sort all restaurants according to the restaurant’s name in ascending order.

FILTER : { }

SORT : { name : 1 }

- 5.19. Display the restaurant’s name, postcode, opening hours, and BYO. Sort the name in descending order.

FILTER : { }

PROJECT : { name:1, "address.postcode":1, openingHours:1, byo:1 }

SORT : { name : -1 }

- 5.20. Display the restaurant's name, postcode, opening hours, and BYO. Sort the name and postcode in ascending order.

FILTER : { }

PROJECT : { name:1, "address.postcode":1, openingHours:1, byo:1 }

SORT : { name : 1, "address.postcode" : 1 }

- 5.21. Find the restaurants that do not serve any cuisine of “Café Food” and their average cost is over \$20 and they are not located in VIC. Sort the results by the average cost.

FILTER : { cuisines: { \$ne: "Cafe Food"}, averageCost: { \$gt: 20 }, "address.state": { \$ne: "VIC" } }

SORT : { averageCost : 1 }

- 5.22. Find the restaurants that serve Seafood or Salad. Sort the results by the average cost in descending order.

FILTER : { \$or: [ { cuisines : "Salad" }, { cuisines : "Seafood" } ] }

SORT : { averageCost : -1 }

- 5.23. Find restaurants that are in Caulfield or Malvern East or Carlton that have the average cost less than \$60.

{ averageCost : {\$lt: 60}, \$or: [ { "address.suburb" : "Caulfield" }, { "address.suburb" : "Malvern East" }, { "address.suburb" : "Carlton" } ] }

Another solution:

{ averageCost : {\$lt: 60}, "address.suburb" : { \$in: [ "Caulfield", "Malvern East", "Carlton" ] } }

- 5.24. Find restaurants that are NOT in Caulfield or Malvern East or Carlton that have the average cost less than \$60.

{ averageCost : {\$lt: 60}, "address.suburb" : { \$nin: [ "Caulfield", "Malvern East", "Carlton" ] } }

- 5.25. Show restaurant that has the “known for” information.

{ known\_for : { \$exists : true } }

## 6. Update Operation.

6.1. Change the average cost of Portofino Pizza to \$40.

6.2. Add a field called `lastModified` in the document that you recently updated.  
Change the data type to Date and insert the value with today's date.

```

1  _id:ObjectId("5d451b986b3d34f35a085606")
2  name :"Portofino Pizza "
3  <address :Object
4    street :"884 Glen Huntly Road "
5    suburb :"Caulfield "
6    state :"VIC "
7    postcode :3145
8    phone :"03 9528 6693 "
9  <cuisines :Array
10   0 :"Pizza "
11   1 :"Italian "
12  averagecost :40
13  opening_hours :"5pm-10pm "
14  byo :true
15  lastModified :2019-08-03T13:00:00.000+00:00

```

ObjectId  
String  
Object  
String  
String  
String  
Double  
String  
Array  
String  
String  
Double  
String  
Boolean  
Date

CANCEL UPDATE

6.3. Add Italian cuisine to Cool Rays.

6.4. Add a field called `lastModified` in Cool Rays. Change the data type to Date and insert the value with today's date.

## 7. Delete Operation.

7.1. Find restaurant named Cool Rays. Delete this restaurant.

7.2. Can we undo our delete action in MongoDB?

In relational databases, “rollback” feature is available where we can return the database to its previous state. This is due to the reason that it must maintain the data integrity. However, this feature is not available in MongoDB as it does not comply to the ACID principles.

**The End**



MONASH  
University

MONASH  
INFORMATION  
TECHNOLOGY

# FIT5137 – Advanced Database Technology

## Week 2 – Introduction to MongoDB

Semester 2, 2020

Developed by:

Dr. Agnes Haryanto

[Agnes.Haryanto@monash.edu](mailto:Agnes.Haryanto@monash.edu)

# Agenda

1. Introduction to MongoDB
2. MongoDB Compass
3. CRUD Operations on MongoDB Compass
4. Introduction to Mongo Shell

# Using FLUX

1. Visit <http://flux.qa/> on your internet enabled device
2. Log in using your Monash account (not required if you are already logged in to Monash)
3. Click on the “+” to join audience
4. Enter the Audience Code: **DELYJJ**
5. Select FIT5137 in the Active Presentation menu
6. Answer questions when they pop up

The screenshot shows a browser window titled "FLUX" with the URL <https://flux.qa/#/feeds/5d368...>. The page is titled "Lecture 0". Below the title are four icons: a bar chart, a person icon, a gift icon, and a clock icon. The main content area is titled "Current Polls". It displays a poll with the question "What did you have for dinner?". The poll has five options, each in a separate box:

- A Pasta
- B Rice
- C Pizza
- D Salad
- E Nothing

# Recall from week 1...

## NoSQL Databases

| NoSQL Category              | Example Databases                                                                                                      |
|-----------------------------|------------------------------------------------------------------------------------------------------------------------|
| Key-value databases         | Dynamo, Riak, Redis, Voldemort, ArangoDB, Aerospike, Apache Ignite                                                     |
| Document-oriented databases | <b>MongoDB</b> , CouchDB, OrientDB, RavenDB, ArangoDB, BaseX, Clusterpoint, Couchbase, Cosmos DB, RethinkDB, MarkLogic |
| Column-oriented databases   | Cassandra, Hbase, Hypertable, Accumulo, Scylla                                                                         |
| Graph databases             | Neo4j, ArangoDB, GraphBase, AllegroGraph, InfiniteGraph, Apache Giraph, MarkLogic, OrientDB, Virtuoso                  |

# Recall from week 1...

## Document-Oriented Database

- Document-oriented databases are inherently a subclass of the key-value database.
- The document can be in any encoded format, such as **XML**, **JSON** (JavaScript Object Notation), or **BSON** (Binary JSON).
- In Key-Value databases, the databases do not attempt to understand the content of the value component. However, the document databases do understand the meaning of the value component.
- Document-oriented databases are schema-less.
  - No predefined structure on the stored data.
  - Each document can have its own structure.
- Document-oriented databases group documents into logical groups called **collections**.

| Collection = Customer |                                                                                                   |
|-----------------------|---------------------------------------------------------------------------------------------------|
| Key                   | Document                                                                                          |
| 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"}              |

# What is MongoDB?

- Document-oriented database.
- A powerful, flexible, and scalable general-purpose database.
- It combines the ability to **scale out** with **various features** like secondary indexes, range queries, sorting, aggregations, and geospatial indexes.

# MongoDB – Ease of Use

- Replaces the concept of **row** in a relational database with a more flexible model, the **document**.
- Complex hierarchical relationships is possible to be represented with a single record.
- No predefined schemas.
  - The document's keys and values are not of fixed types or sizes.

# MongoDB – Easy Scaling

- MongoDB is designed to scale out.
- The document-oriented data model makes it easier to split up data across multiple servers.
- MongoDB automatically takes care of balancing data and load across a cluster, redistributing documents automatically and routing user requests to the correct machines.
  - Allows developers to focus on programming the application, not scaling it.

# MongoDB – Tons of features ...

- MongoDB is intended to be a general-purpose database.
- Main Features:
  - **Create, Read, Update, Delete (CRUD) Operations**
  - **Indexing**
    - MongoDB supports generic secondary indexes, allowing a variety of fast queries, and provides unique, compound, geospatial, and full-text indexing capabilities as well.
  - **Aggregation**
    - MongoDB supports an “aggregation pipeline” that allows to build complex aggregations from simple pieces and allow the database to optimize it.

# MongoDB – Tons of features ...

- Main Features:
  - Special collection types
    - MongoDB supports time-to-live collections for data that should expire at a certain time, such as sessions. It also supports fixed-size collections, which are useful for holding recent data, such as logs.
  - File storage
    - MongoDB supports an easy-to-use protocol for storing large files and file metadata.
- Different from relational databases, MongoDB does not support joins and complex multirow transactions since both of these features are difficult to provide efficiently in a distributed system.

# MongoDB – ... without sacrificing speed

- Almost every aspect of MongoDB was designed to maintain high performance.
  - MongoDB adds dynamic padding to documents and pre-allocates data files to trade extra space usage for consistent performance.
  - It uses as much of RAM as it can as its cache and attempts to automatically choose the correct indexes for queries.

# MongoDB

- **Document:** the basic unit of data in MongoDB.
  - Similar to a row in relational database (but much more expressive).
  - Documents are structured in BSON – similar to JSON.
    - A data structure made up of **field** and **value** pairs.
- **Collection:** a grouping of MongoDB documents.
  - Similar to a table in relational database.
  - Typically, all documents in a collection have a similar or related purpose.
- **Database:** a physical container for collections.

# Documents

- Example:

```
{  
  "name": "Wilma",           ← field: value  
  "age": 20,                 ← field: value  
  "course": "BITS",          ← field: value  
  "units": ["FIT1001", "FIT1002"] ← field: value  
}
```

- The values of fields may include other documents, arrays, and arrays of documents.

# Documents

- Example:

```
{  
  "name": "Wilma",  
  "age": 20,  
  "course": "BITS",  
  "units": ["FIT1001", "FIT1002"]  
}
```

| <b>name</b> | <b>age</b> | <b>course</b> | <b>units</b>        |
|-------------|------------|---------------|---------------------|
| Wilma       | 25         | BITS          | FIT1001,<br>FIT1002 |

# Documents

- Example:

```
{  
    "name": "Wilma",  
    "age": 20,  
    "course": "BITS",  
    "units": [  
        { "code": "FIT1001", "semester": 1 },  
        { "code": "FIT1002", "semester": 2 }  
    ]  
}
```

| <b>name</b> | <b>age</b> | <b>course</b> | <b>units</b>              |
|-------------|------------|---------------|---------------------------|
| Wilma       | 20         | BITS          | FIT1001 S1,<br>FIT1002 S2 |

# JSON

- JSON syntax is derived from JavaScript object notation syntax:
  - Data is in name/value pairs
  - Data is separated by commas
  - Curly braces hold objects
  - Square brackets hold arrays

# JSON

- In JSON, values must be one of the following data types:

- String
- Number
- Object (JSON object)
- Array
- Boolean
- Null

The diagram shows a JSON object represented by curly braces {}, containing several properties and their values. Annotations with arrows point from labels to specific parts of the JSON code:

- An arrow points from the word "String" to the value "Wilma" in the "name" property.
- An arrow points from the word "Number" to the value 20 in the "age" property.
- An arrow points from the word "Array" to the value ["BITS", "BSE"] in the "course" property, which is enclosed in a black rectangular box.
- An arrow points from the word "Object" to the value [ { "code": "FIT1001", "semester": 1 }, { "code": "FIT1002", "semester": 2 } ] in the "units" property, which is also enclosed in a black rectangular box.
- An arrow points from the word "Boolean" to the value true in the "onCampus" property.

```
{  
  "name": "Wilma",  
  "age": 20,  
  "course": [ "BITS", "BSE" ],  
  "units": [  
    { "code": "FIT1001", "semester": 1 },  
    { "code": "FIT1002", "semester": 2 }  
  ],  
  "onCampus": true  
}
```

# MongoDB Data Type

- String
- Integer
- Boolean
- Double
- Min/ Max keys
- Arrays
- Timestamp
- Object
- Null
- Symbol
- Date
- Object ID
- Binary data
- Code
- Regular expression

# MongoDB Compass

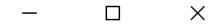
# MongoDB Compass



- GUI for MongoDB.
- Good for someone new to MongoDB.
  - Easy to use and navigate.
  - Visual data exploration.
  - The content of the data can be analyze and understood even with little knowledge of MongoDB query syntax.

# Databases Tab

MongoDB Compass Community - localhost:27017



Connect View Help

**Local**

3 DBS 1 COLLECTIONS C

☆ FAVORITE

HOST  
localhost:27017

CLUSTER  
Standalone

EDITION  
MongoDB 4.4.0 Community

Filter your data

> admin

> config

> local

+

Databases

CREATE DATABASE

| Database Name | Storage Size | Collections | Indexes |
|---------------|--------------|-------------|---------|
| admin         | 20.0KB       | 0           | 1       |
| config        | 12.0KB       | 0           | 2       |
| local         | 36.0KB       | 1           | 1       |

# MongoDB CRUD Operations

- **Create** – add new documents to a collection.
- **Read** – retrieve documents from a collection.
- **Update** – modify existing documents in a collection.
- **Delete** – remove documents from a collection.

# CRUD – Create a Database

MongoDB Compass Community - localhost:27017

Connect View Help

Local

3 DBS 1 COLLECTIONS C

☆ FAVORITE

HOST  
localhost:27017

CLUSTER  
Standalone

EDITION  
MongoDB 4.4.0 Community

Filter your data

> admin

> config

> local

+

Databases

CREATE DATABASE

New database can be created with a single click

| Database Name | Storage Size | Collections | Indexes |
|---------------|--------------|-------------|---------|
| admin         | 20.0KB       | 0           | 1       |
| config        | 12.0KB       | 0           | 2       |
| local         | 36.0KB       | 1           | 1       |

# CRUD – Create a Database

Create Database

**Database Name**

**Collection Name**

Capped Collection i

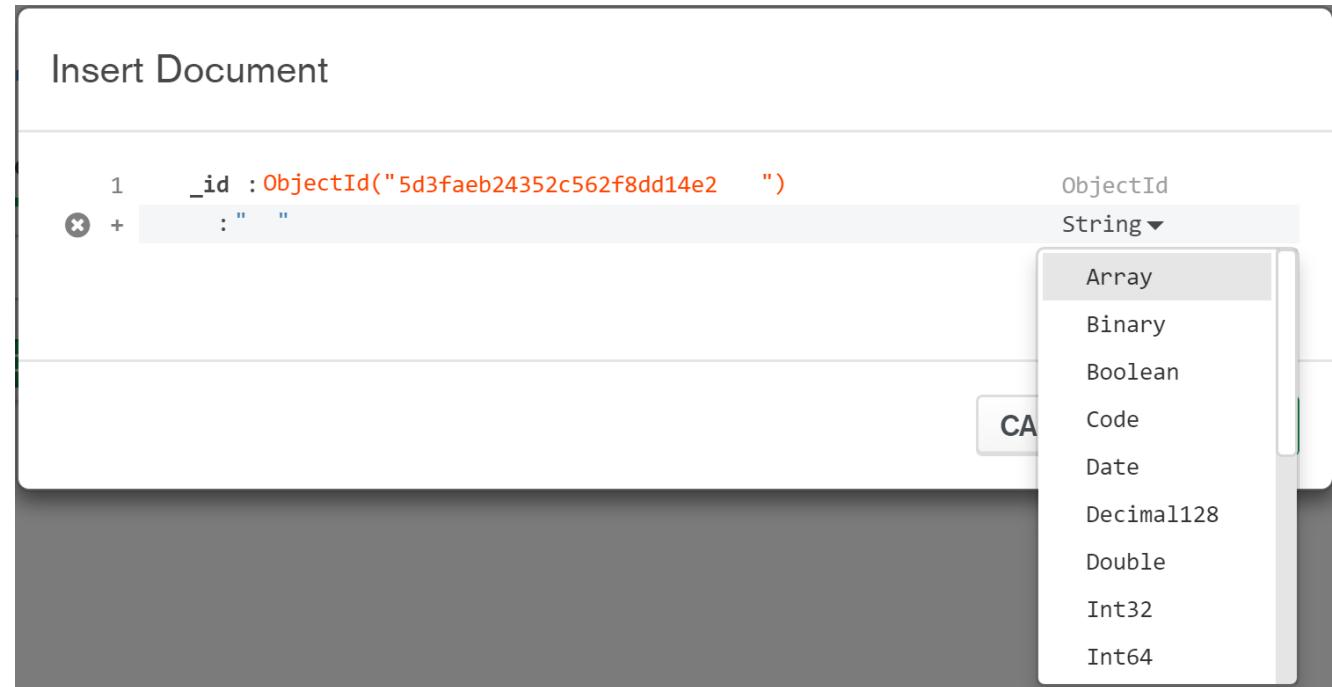
Use Custom Collation i

Before MongoDB can save your new database, a collection name must also be specified at the time of creation. [More Information](#)

**CANCEL** **CREATE DATABASE**

# CRUD – Create a Document

- Insert a Single Document
  - The data type has to be specified

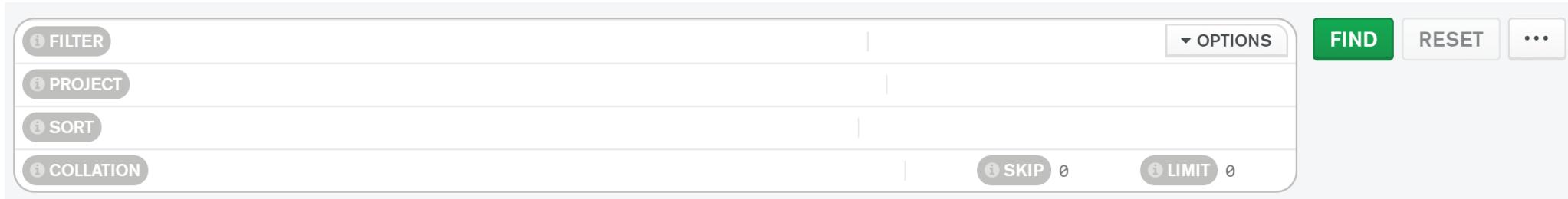


- MongoDB Compass does not support multiple documents insertion.

# CRUD – Create a Document

| Item     | Status | Size                         | In-stock                                       |
|----------|--------|------------------------------|------------------------------------------------|
| Journal  | A      | h: 14<br>w: 21<br>uom: cm    | warehouse: A<br>qty: 5                         |
| Notebook | A      | h: 8.5<br>w: 11<br>uom: in   | warehouse: C<br>qty: 5                         |
| Paper    | D      | h: 8.5<br>w: 11<br>uom: in   | warehouse: A<br>qty: 60                        |
| Planner  | D      | h: 22.85<br>w: 30<br>uom: cm | warehouse: A<br>qty: 40                        |
| Postcard | A      | h: 10<br>w: 15.25<br>uom: cm | warehouse: B, qty: 15<br>warehouse: C, qty: 35 |

# CRUD – Query Documents



- **FILTER** – to filter records; similar to WHERE clause in SQL SELECT statement.
- **PROJECT** – to specify which fields to return in the resulting data.
- **SORT** – to specify the sort order of the returned documents.
- **COLLATION** – to specify language-specific rules for string comparison.
- **SKIP** – to specify how many documents to skip before returning the result set.
- **LIMIT** – to specify the maximum number of documents to return.

# CRUD – Query Documents

- Select all documents in a collection
  - { }
- SQL:
  - SELECT \* FROM <table>

# CRUD – Query Documents

- To find documents based on a certain filter:

- { <field1>: <value1>, ... }

- For example:

- { status: "D" }

- SQL:

- SELECT \* FROM inventory WHERE status = "D"

# CRUD – Query Documents

- To find documents based on a certain filter:

- { <field1>: { <operator1>: <value1> } , ... }

- For example:

- { status: { \$in: [ "A", "D" ] } }

- SQL:

- SELECT \* FROM inventory WHERE status in ("A", "D")

# CRUD – MongoDB Query Operators

| 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.                   |
| \$exists | Matches documents that have the specified field.                    |

# CRUD – Query Documents

- Specify AND conditions:

- { status: "A", qty: { \$lt: 30 } }

- SQL:

- SELECT \* FROM inventory WHERE status = "A" AND qty < 30

# CRUD – Query Documents

- Specify OR conditions:

- { \$or: [ { status: "A" }, { qty: { \$lt: 30 } } ] }

- SQL:

- SELECT \* FROM inventory WHERE status = "A" OR qty < 30

# CRUD – Query Documents

- Specify **AND** as well as **OR** conditions:

➤ { status: "A", \$or: [ { qty: { \$lt: 30 } }, { item: /<sup>^</sup>p/ } ] }

- SQL:

➤ SELECT \* FROM inventory WHERE status = "A" AND (qty < 30 OR item LIKE "p%")

- MongoDB also supports regular expressions **\$regex** queries to perform string pattern matches.

# CRUD – Query Documents

- PROJECT – display all:

➤ { }

```
_id: ObjectId("5f3054d70e76e8146f92fc78")
unitCode: "FIT1001"
unitname: "Computer Systems"
offerring: "S2"
year: 2020
> tutors: Array
campus: "Clayton"
```

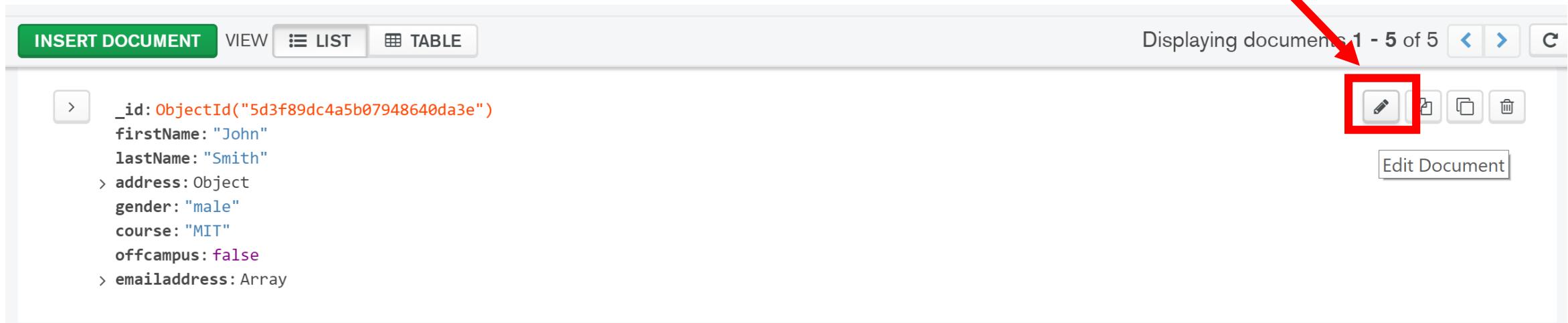
- PROJECT – display unit code and unit name while hiding the ID:

➤ { unitCode:1, unitname:1, \_id:0 }

```
unitCode: "FIT1001"
unitname: "Computer Systems"
```

# CRUD – Update

Click on this icon to update



DISPLAYING DOCUMENTS 1 - 5 OF 5

INSERT DOCUMENT VIEW LIST TABLE

\_id: ObjectId("5d3f89dc4a5b07948640da3e")  
firstName: "John"  
lastName: "Smith"  
> address: Object  
gender: "male"  
course: "MIT"  
offcampus: false  
> emailaddress: Array

Edit Document

# CRUD – Update

```
1  _id: ObjectId("5d3f89dc4a5b07948640da3e")          ObjectId
2  firstName : "John"                                  String
3  lastName : "Smith"                                 String
4  > address : Object                                Object
5  gender : "male"                                   String
6  course : "MIT"                                    String
7  offcampus : false                                 Boolean
8  ▼ emailaddress : Array                           Array
9    0 : "jsmith@gmail.com"                         String
10   1 : "jsmith@yahoo.com"                         String
```

CANCEL UPDATE

# CRUD – Delete a Database

MongoDB Compass Community - localhost:27017

Connect View Collection Help

Local

3 DBS 1 COLLECTIONS C

CREATE DATABASE

HOST  
localhost:27017

CLUSTER  
Standalone

EDITION  
MongoDB 4.4.0 Community

Filter your data

> admin

> config

> lecture2

> local

Databases

| Database Name | Storage Size | Collections | Indexes |
|---------------|--------------|-------------|---------|
| admin         | 20.0KB       | 0           | 1       |
| config        | 24.0KB       | 0           | 2       |
| lecture2      | 4.0KB        | 1           | 1       |
| local         | 36.0KB       | 1           | 1       |

Simply click and confirm

The screenshot shows the MongoDB Compass interface with the 'Databases' tab selected. On the left, there's a sidebar with connection details: HOST localhost:27017, CLUSTER Standalone, and EDITION MongoDB 4.4.0 Community. Below that is a 'Filter your data' section and a list of databases: admin, config, lecture2, and local. The main area displays a table of databases with columns: Database Name, Storage Size, Collections, and Indexes. The 'config' database is highlighted with a red box and has a red arrow pointing to its delete icon. The delete icons for all databases are shown as small trash cans.

# CRUD – Delete a Database

MongoDB Compass Community - localhost:27017

Connect View Collection Help

Local

3 DBS 1 COLLECTIONS C

☆ FAVORITE

HOST  
localhost:27017

CLUSTER  
Standalone

EDITION  
MongoDB 4.4.0 Community

Filter your data

> admin

> config

> lecture2

> local

Databases

CREATE DATABASE

Database Name

admin

config

lecture2

local

36.0KB

Indexes

1

2

1

1

Drop Database

⚠ To drop **lecture2** type the database name **lecture2**.

lecture2

CANCEL

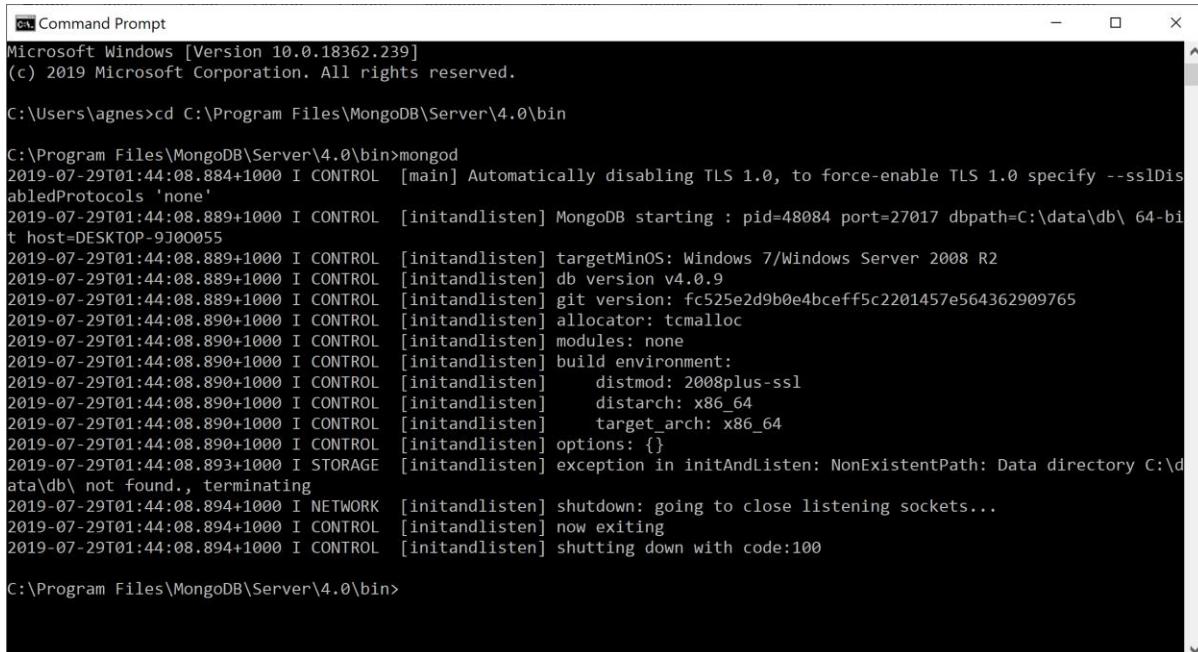
DROP DATABASE

The screenshot shows the MongoDB Compass interface with a modal dialog titled "Drop Database". The modal contains a warning message: "⚠ To drop **lecture2** type the database name **lecture2**." Below the message is an input field containing the text "lecture2". At the bottom of the modal are two buttons: "CANCEL" and a red "DROP DATABASE" button. The background of the interface shows a list of databases: "admin", "config", "lecture2" (which is highlighted), and "local". The "lecture2" database has 36.0KB of data and 1 index.

# Mongo Shell

# Using MongoDB Shell

- Run MongoDB server
  - mongod



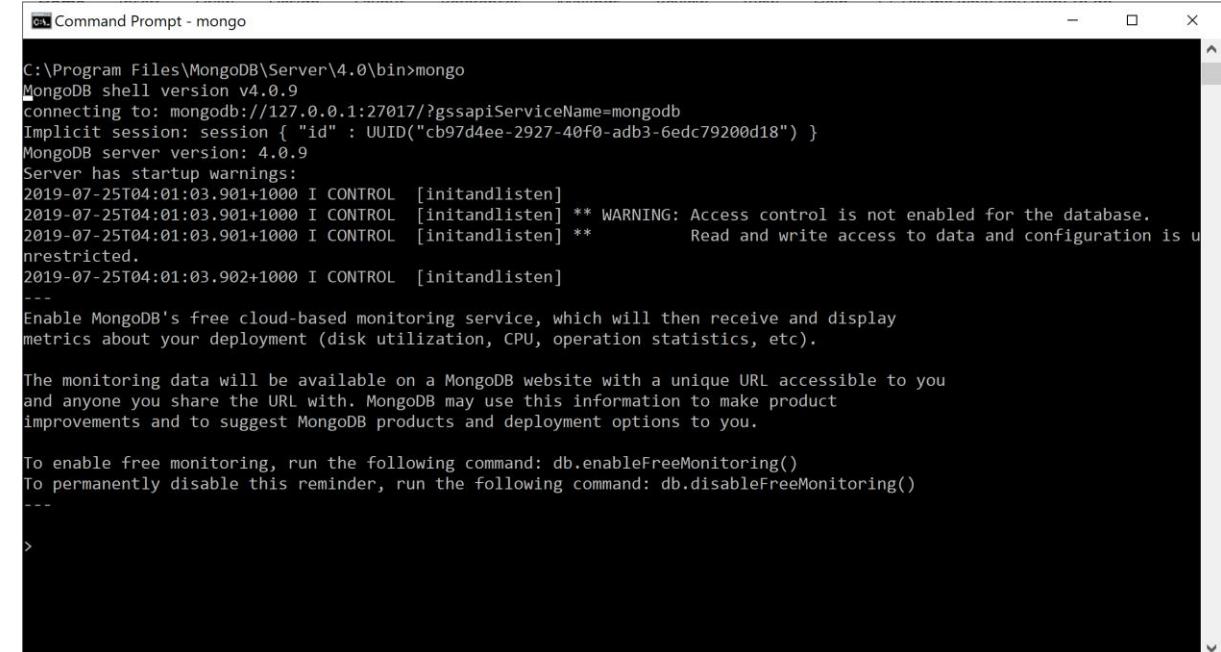
```
Microsoft Windows [Version 10.0.18362.239]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\agnes>cd C:\Program Files\MongoDB\Server\4.0\bin

C:\Program Files\MongoDB\Server\4.0\bin>mongod
2019-07-29T01:44:08.884+1000 I CONTROL  [main] Automatically disabling TLS 1.0, to force-enable TLS 1.0 specify --sslDisabledProtocols 'none'
2019-07-29T01:44:08.889+1000 I CONTROL  [initandlisten] MongoDB starting : pid=48084 port=27017 dbpath=C:\data\db\ 64-bit host=DESKTOP-9J00055
2019-07-29T01:44:08.889+1000 I CONTROL  [initandlisten] targetMinOS: Windows 7/Windows Server 2008 R2
2019-07-29T01:44:08.889+1000 I CONTROL  [initandlisten] db version v4.0.9
2019-07-29T01:44:08.889+1000 I CONTROL  [initandlisten] git version: fc525e2d9b0e4bceff5c2201457e564362909765
2019-07-29T01:44:08.890+1000 I CONTROL  [initandlisten] allocator: tcmalloc
2019-07-29T01:44:08.890+1000 I CONTROL  [initandlisten] modules: none
2019-07-29T01:44:08.890+1000 I CONTROL  [initandlisten] build environment:
2019-07-29T01:44:08.890+1000 I CONTROL  [initandlisten]   distmod: 2008plus-ssl
2019-07-29T01:44:08.890+1000 I CONTROL  [initandlisten]   distarch: x86_64
2019-07-29T01:44:08.890+1000 I CONTROL  [initandlisten]   target_arch: x86_64
2019-07-29T01:44:08.890+1000 I CONTROL  [initandlisten] options: {}
2019-07-29T01:44:08.893+1000 I STORAGE [initandlisten] exception in initAndListen: NonExistentPath: Data directory C:\data\db\ not found., terminating
2019-07-29T01:44:08.894+1000 I NETWORK  [initandlisten] shutdown: going to close listening sockets...
2019-07-29T01:44:08.894+1000 I CONTROL  [initandlisten] now exiting
2019-07-29T01:44:08.894+1000 I CONTROL  [initandlisten] shutting down with code:100

C:\Program Files\MongoDB\Server\4.0\bin>
```

- Run Mongo Shell
  - mongo



```
C:\Program Files\MongoDB\Server\4.0\bin>mongo
MongoDB shell version v4.0.9
connecting to: mongodb://127.0.0.1:27017/?gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("cb97d4ee-2927-40f0-adb3-6edc79200d18") }
MongoDB server version: 4.0.9
Server has startup warnings:
2019-07-25T04:01:03.901+1000 I CONTROL  [initandlisten]
2019-07-25T04:01:03.901+1000 I CONTROL  [initandlisten] ** WARNING: Access control is not enabled for the database.
2019-07-25T04:01:03.901+1000 I CONTROL  [initandlisten] ** Read and write access to data and configuration is unrestricted.
2019-07-25T04:01:03.902+1000 I CONTROL  [initandlisten]
---
Enable MongoDB's free cloud-based monitoring service, which will then receive and display metrics about your deployment (disk utilization, CPU, operation statistics, etc).

The monitoring data will be available on a MongoDB website with a unique URL accessible to you and anyone you share the URL with. MongoDB may use this information to make product improvements and to suggest MongoDB products and deployment options to you.

To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---

>
```

# Explore Databases and Collections

- To show active database
  - db
- To list all databases
  - show dbs
- Change database
  - use <database\_name>
- To show all collections in database
  - show collections

# Collections

- Create a new collection
  - `db.createCollection("<collection name>")`
- List all collections
  - `show collections`
- Delete a collection
  - `db.getCollection("<collection name>").drop()`
  - `db.<collection name>.drop()`
- Delete database
  - `db.dropDatabase()`

# CRUD – Insert

- **Insert One or Many Documents**

- `db.<collection name>.insert (<Object> or <Array of Objects>)`

- **Insert One Document**

- `db.<collection name>.insertOne (<Object>)`

- **Insert Many Documents**

- `db.<collection name>.insertMany (<Array of Objects>)`

# CRUD – Insert One Document

- To insert a single document:

- db.<collection name>.insertOne(<Object>)

- For example:

- db.car.insertOne({  
    name: "honda",  
    make: "accord",  
    year: 2010  
})

# CRUD – Insert Many Documents

- To insert multiple documents:

- db.<collection name>.insertMany(<Array of Objects>)

- For example:

- db.car.insertMany( [  
    { name: "Honda", make: "accord", year: 2012 },  
    { name: "Mazda", make: "mazda2", year: 2015 },  
    { name: "Toyota", make: "camry", year: 2019 }  
] );

# **CRUD – Read**

- **Find Documents**

- `db.<collection name>.find(<query>, <fields>)`

- **Find One Document**

- `db.<collection name>.findOne(<query>, <fields>)`

# CRUD – Find All Documents

- Select all documents in a collection:

- db.car.find()
- db.car.find().pretty()

- SQL:

- SELECT \* FROM car

# CRUD – Find One Document

- Select one document in a collection

- db.<collection name>.findOne(<query>, <fields>)

- For example:

- db.car.findOne({ name: "honda" })

- SQL:

- SELECT \* FROM car WHERE name = "honda"

# CRUD – Find One Document

- Select one document in a collection

- db.<collection name>.findOne(<query>, <fields>)

- For example:

- db.car.findOne({ name: "honda" }, { name: 1, make: 1 })

- SQL:

- SELECT name, make FROM car WHERE name = "honda"

# **CRUD – Update**

- **Update One Document**
  - `db.<collection name>.updateOne(<filter>, <update>, <options>)`
- **Update Many Documents**
  - `db.<collection name>.updateMany(<filter>, <update>, <options>)`
- **Replace a single document within the collection based on filter**
  - `db.<collection name>.replaceOne(<filter>, <update>, <options>)`

# CRUD – Update

- Update One Document

- db.<collection name>.updateOne(<filter>, <update>, <options>)

- For example:

- db.car.updateOne(  
    { "name" : "honda" } ,  
    {  
        \$set: { "name" : "ford" } ,  
        \$currentDate: { lastModified: true }  
    }  
)

# CRUD – Update

- Update Many Documents

- db.<collection name>.updateMany(<filter>, <update>, <options>)

- For example:

- db.car.updateMany(  
  { "year": { \$gt: 2015 } },  
  {  
    \$set: { "name" : "holden" },  
    \$currentDate: { lastModified: true }  
  }  
)

# CRUD – Update

- Replace a single document within the collection based on filter

➤ db.<collection name>.replaceOne(<filter>, <update>, <options>)

- For example:

➤ db.car.replaceOne(  
  { "name" : "honda" },  
  { "name" : "Honda", "make" : "Jazz" }  
)

# CRUD – Delete

- Delete One Document
  - db.<collection name>.deleteOne()
- Delete Many Documents
  - db.<collection name>.deleteMany()
- Delete All Documents
  - db.<collection name>.deleteMany( {} )

# References

- [1] K. Chodorow, MongoDB: The Definitive Guide. Sebastopol, CA, USA: O'Reilly Media, Inc., 2<sup>nd</sup> ed., 2013.
- [2] MongoDB, “BSON Types”  
<https://docs.mongodb.com/manual/reference/bson-types/>.
- [3] MongoDB, “MongoDB Compass”  
<https://docs.mongodb.com/compass/current/>.
- [4] MongoDB, “MongoDB CRUD Operations”  
<https://docs.mongodb.com/manual/crud/>.
- [5] w3schools.com, “JSON Syntax”  
[https://www.w3schools.com/js/js\\_json\\_syntax.asp](https://www.w3schools.com/js/js_json_syntax.asp).

# Tutorial 1

## Introduction to Non-Relational Databases

# SOLUTIONS

Discuss the following questions:

1. What is the difference between Relational Database and Non-Relational Database?

Relational Database follows relational data model and uses Structured Query Language (SQL), in which the operations are based on relational algebra. Relational Database also complies the ACID (Atomicity, Consistency, Isolation, and Durability) transaction properties.

Non-Relational Database on the other hand does not follow relational model since it is usually schema-free. It does not comply the ACID properties, but it follows BASE (Basically Available, Soft State, Eventual consistency) principles.

2. What is Big Data?

Big Data commonly denotes to a set of data that represents the characteristics of volume, velocity, and variety in which this set of data is unsuitable to be managed by a relational database management system.

3. What are the traditional 3 Vs of Big Data? Briefly, define each.

Volume, Velocity, and Variety are the traditional 3 Vs of Big Data. Volume refers to the quantity of data to be stored. Velocity refers to the speed at which data is entering the system. Variety refers to the variations in the structure of the data to be stored.

4. Explain why companies like Google and Amazon were among the first to address the Big Data problem.

The use of Internet increased significantly back in the 1990s and a lot of commercial websites started to gain popularity to millions of new users to online transactions. Subsequently after the dot-com bubble burst in 1990s, a lot of start-up web-based companies failed to survive. But for the companies that still survived, they faced a significant growth of users and data within a very short time. Google and Amazon were among those companies in which they were the first to feel the pressure of handling Big Data.

5. Explain the difference between *scaling up* and *scaling out*.

Scaling up involves migrating and improving system to a larger system while maintaining the same number of systems.

Scaling out involves spreading out workloads across several servers when the workloads surpass the capacity of the current server.

6. How is stream processing different from feedback loop processing?

Stream processing focuses on input processing and it requires analysis of the data stream as it enters the system.

Feedback loop processing focuses on output and it refers to the analysis of the data to produce actionable results.

## 7. What is polyglot persistence?

Polyglot persistence is the concept of using multiple variety of data storage and management technologies within an organization's infrastructure.

## 8. What is NoSQL?

NoSQL stands for Not-only SQL. It is a new generation of database that address the issues in Big Data. It is non-relational, distributed, open-source, and horizontal scalable. Characteristically, NoSQL is:

- Schema-less; not based on the relational model.
- Support distributed database architectures.
- Provide high scalability, high availability, and fault tolerance.
- Support very large amounts of sparse data.
- Designed mainly towards performance rather than transaction consistency.

## 9. What are the four basic categories of NoSQL databases? List some examples of database technologies for each category.

- **Key-value databases**

Example DBs: Dynamo, Riak, Redis, Voldemort, ArangoDB, Aerospike, Apache Ignite

- **Document databases**

Example DBs: MongoDB, Apache CouchDB, Couchbase, Cosmos DB, OrientDB, Qizx, RethinkDB

- **Column-oriented databases**

Example DBs: Cassandra, Hbase, Hypertable, Accumulo, Scylla

- **Graph databases**

Example DBs: Neo4j, ArangoDB, GraphBase, AllegroGraph, InfiniteGraph, Apache Giraph, MarkLogic, OrientDB, Virtuoso

## 10. How are the value components of a key-value database and a document database different?

In the Key-Value databases, the databases do not attempt to understand the content of the value component. It is an unintelligent way of storing data as the DBMS does not attempt to understand the meaning of the data at all, so we must rely on the applications that use the data to understand the meaning of the data in the value component.

However, the document databases do understand the meaning of the value component.

11. What is the difference between a column and a super column in a column family database?

Column is a key-value pair, like a cell of data in the relational database. All columns in a column-oriented database are relatively independent of each other. Meanwhile, a super column is a group of columns that are logically related.

12. Explain why graph databases tend to struggle with scaling out.

Graph databases are designed to store data with complex relationships. The data in a graph database are tightly integrated to one another. Scaling out involves moving data across several servers, which means that the data needs to be relatively independent in order to be spread across different servers. Because of the highly related data nature of graph databases, scaling out will cause communication overhead. Therefore, graph databases tend to perform best in centralized or lightly clustered environments, instead of a scaled-out environment.



**MONASH**  
University

MONASH  
INFORMATION  
TECHNOLOGY

# **FIT5137 – Advanced Database Technology**

Week 1b – Introduction to Non-Relational Database

Semester 2, 2020

Developed by:

Dr. Agnes Haryanto

[Agnes.Haryanto@monash.edu](mailto:Agnes.Haryanto@monash.edu)

# Agenda

1. Relational Database vs Non-Relational Database
2. Issues in Big Data era
3. NoSQL Databases
  - NoSQL Categories

# Database

“A **database** is a shared, integrated computer structure that stores a collection of the following:

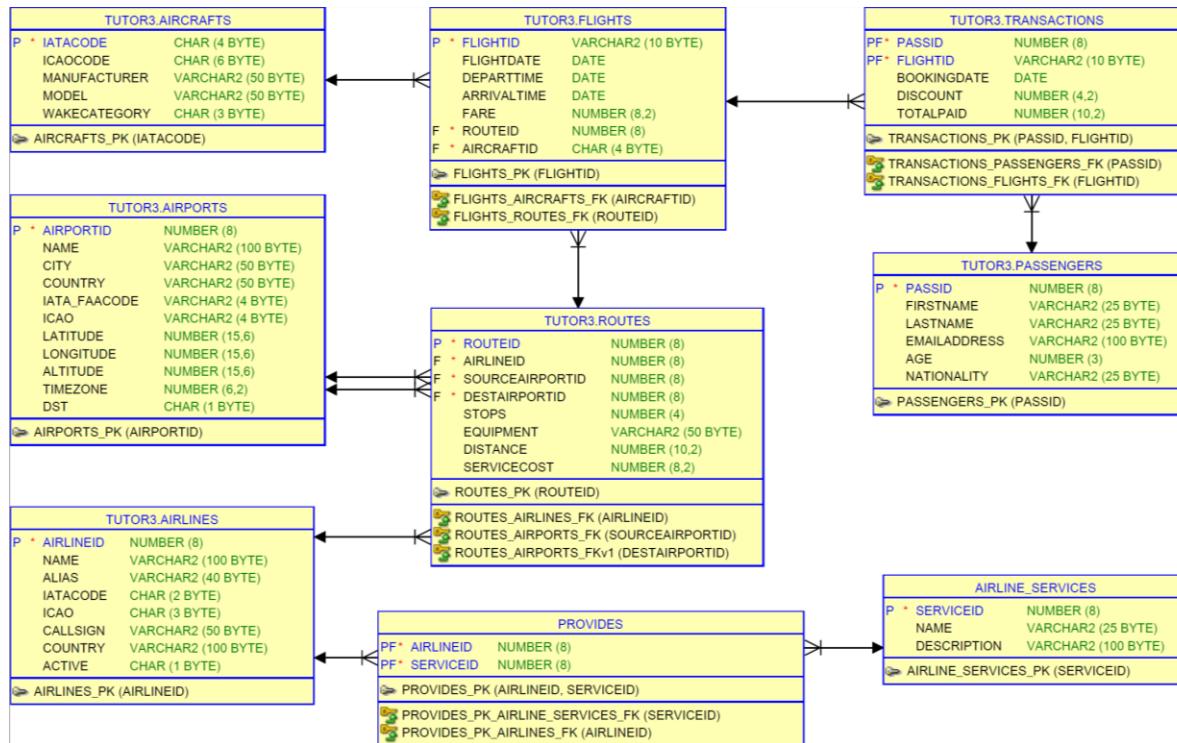
- End-user data—that is, raw facts of interest to the end user
- Metadata, or data about data, through which the end-user data is integrated and managed” [1]

“A **database** is a collection of data, typically describing the activities of one or more related organizations” [2]

“A **database management system (DBMS)** is a collection of programs that manages the database structure and controls access to the data stored in the database.” [1]

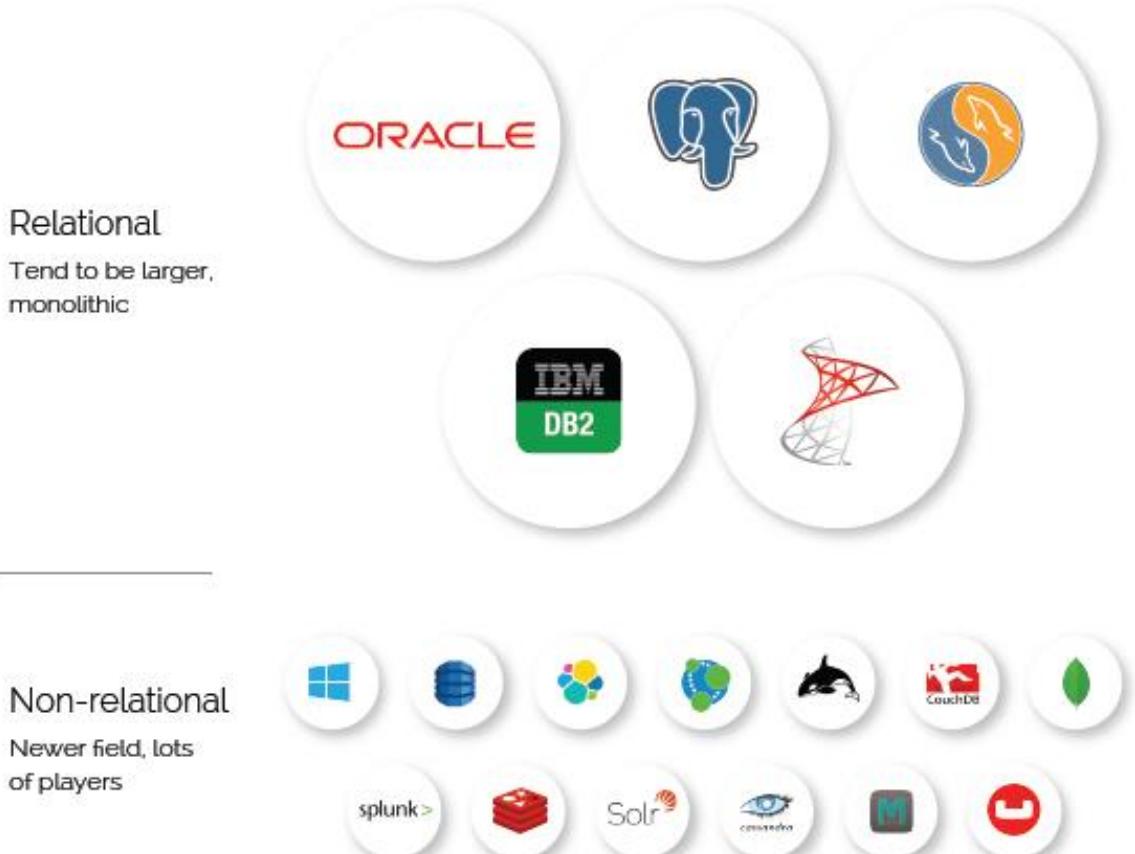
# Relational Database

- Follow relational data model
  - Based on relational algebra
- Use Structured Query Language (SQL)
  - Oracle SQL Database, MySQL, MS Access, SQL Server
- ACID (Atomicity, Consistency, Isolation, Durability) compliant



# Non-Relational Database

- Does not follow relational model
- Schema-less data model
- Known as **NoSQL** databases
- **BASE (Basically Available, Soft state, Eventual consistency)** principles



# History

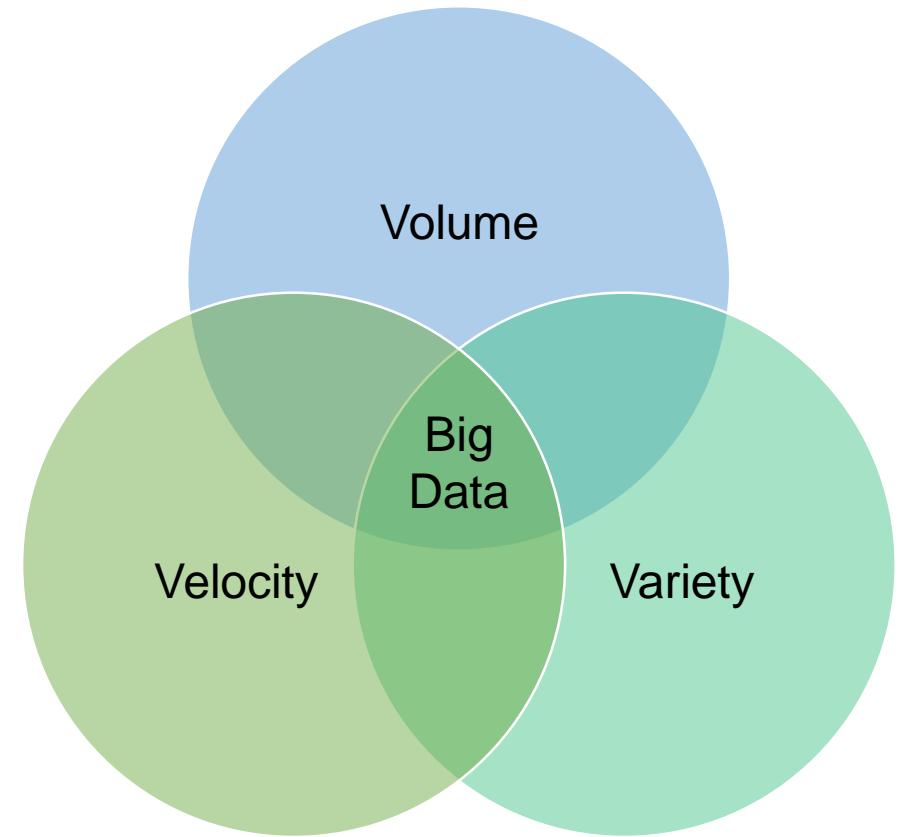
- 1990s
  - Dot-com bubble burst – growth of web commerce companies
  - Companies like Google and Amazon faced the pressure in managing a rapid growth in data
- 2000s – social media era
  - Twitter – over 500 million tweets posted everyday
    - Twitter's MySQL database repeatedly overloaded
  - Facebook – over 500 TB of data produces everyday

# Big Data

- Massive volume of data is generated everyday and it is getting more complex.
- It is difficult to manage the data using traditional relational database management system.
- Big Data – commonly refers to a set of data that exhibits the characteristics of *volume*, *velocity*, and *variety* (the “3 Vs”).

# Characteristics of Big Data

- Volume — the amount of data to be stored
- Velocity — the speed at which data is entering the system
- Variety — the variations in the structure of the data to be stored



# Volume – Scaling up

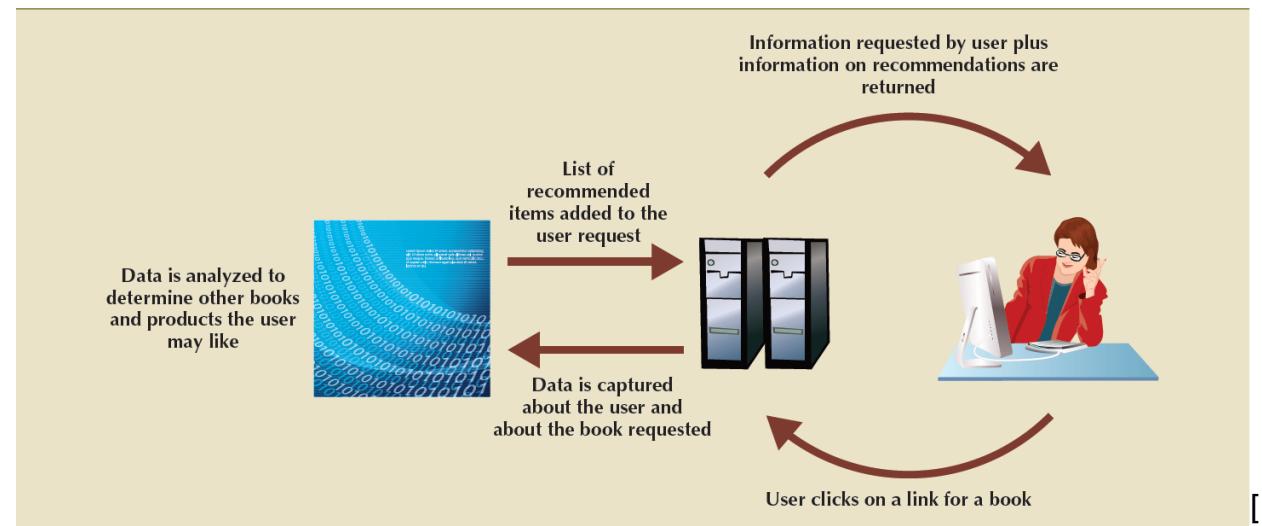
- Scaling up is maintaining the same amount of systems, but migrating each system to a larger one
  - For example, changing from a server with 16 CPU cores and a 1 TB storage system to a server with 64 CPU cores and a 100 TB storage system.
- Scaling up involves moving to larger and faster systems.
- However, there are limits to how large and fast a single system can be.
- The costs of these high-powered systems increase at a dramatic rate.

# Volume – Scaling out

- Scaling out means that when the workload surpasses the server's capacity, the workload is shared across a number of servers.
- This is also referred to as *clustering*
  - Forming a cluster of low-cost servers to share a workload.
- This can help to reduce the overall cost of the computing resources.
  - For example, it is cheaper to buy ten 100 TB storage systems than it is to buy a single 1 PB storage system.

# Velocity

- The velocity in Big Data processing can be broken into two categories:
  - Stream processing**  
Focuses on input processing and it requires analysis of the data stream as it enters the system.
  - Feedback loop processing**  
Refers to the analysis of the data to produce actionable results.



# Variety

- Data can be considered to be *structured*, *unstructured*, or *semi-structured*.
  - **Structured data** – data that has been organized to fit a predefined data model.
  - **Unstructured data** – data that is not organized to fit into a predefined data model.
  - **Semi-structured data** – data that combines elements of both structured and unstructured data.

# Big Data – Issues

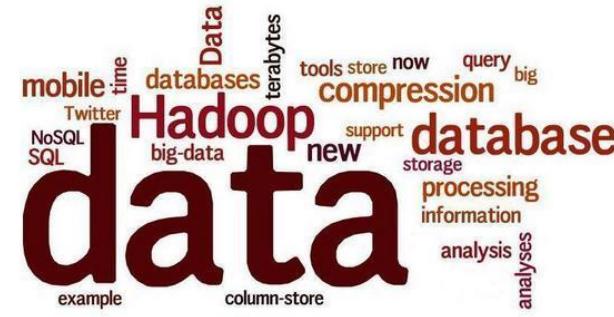
- The need of moving to a larger storage device increases when the amount of data to be stored increases.
  - Problem with *scaling up* and *scaling out*.
- Finding a balance between massive data management and cost.
- The need of rapid data processing.
- No “one-size-fits-all” cure to data management needs.
  - **Polyglot persistence** – using multiple data storage and management technologies within an organization’s infrastructure.

# Big Data – Issues

- What happens if:
  - Blackberry had immediately acted upon the emerging Apple smartphone technology.
  - MySpace had also reacted to the big data challenge that Facebook encountered.
  - Blockbuster (video rental chain) had responded to the Netflix business model in time.
  - Barnes & Noble had developed a feasible Internet strategy before Amazon.

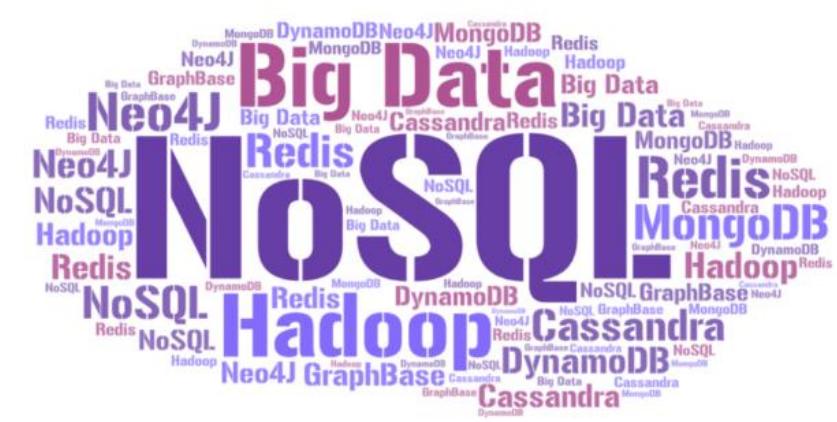
# NoSQL

- NoSQL stands for **Not Only SQL**.
- A new generation of database to address issues in Big Data.
  - Non-relational databases approach to data management.
- NoSQL is designed to efficiently handle:
  - extraordinary volume of data,
  - variety of data types and structures,
  - velocity of data operations.
- NoSQL feature: shared-nothing horizontal scaling – replicating and partitioning data over many servers.
- Not ACID compliant, but follow BASE principle.
  - ACID properties are compromised in order to obtain higher performance and scalability.



# NoSQL's Characteristics

- Not based on the relational model and SQL (schema-free).
  - Support distributed database architectures.
  - Provide *high scalability*, *high availability*, and *fault tolerance*.
  - Able to support very large amounts of sparse data.
  - Designed mainly towards performance rather than transaction consistency.



# NoSQL Categories

- Key-value databases
- Document-oriented databases
- Column-oriented databases
- Graph databases

Because of the broad variability of NoSQL software, the categories above are not necessarily all-encompassing. Some software can also be classified into multiple categories.

# NoSQL Categories

| NoSQL Category              | Example Databases                                                                                              |
|-----------------------------|----------------------------------------------------------------------------------------------------------------|
| Key-value databases         | Dynamo, Riak, Redis, Voldemort, ArangoDB, Aerospike, Apache Ignite                                             |
| Document-oriented databases | MongoDB, CouchDB, OrientDB, RavenDB, ArangoDB, BaseX, Clusterpoint, Couchbase, Cosmos DB, RethinkDB, MarkLogic |
| Column-oriented databases   | Cassandra, Hbase, Hypertable, Accumulo, Scylla                                                                 |
| Graph databases             | Neo4j, ArangoDB, GraphBase, AllegroGraph, InfiniteGraph, Apache Giraph, MarkLogic, OrientDB, Virtuoso          |

# Key-value databases

- Data are stored as a collection of **key-value pairs**.
  - **Key**: an identifier for the value
  - **Value**: can be anything like text, an XML document, a numeric, a document or an image.
- Unintelligent way of storing data as the DBMS does not attempt to understand the meaning of the data.
  - Have to rely on the applications that use the data to understand the meaning of the data in the value component.
- No foreign keys – relationships cannot be tracked.
- Extremely fast and scalable for basic processing.

| Key | Value            |
|-----|------------------|
| K1  | AAA,BBB,CCC      |
| K2  | AAA,BBB          |
| K3  | AAA,DDD          |
| K4  | AAA,2,01/01/2015 |
| K5  | 3,ZZZ,5623       |

# Key-value databases

- The key-value structure can be used as a general data modeling technique when attributes are numerous but actual data values are scarce.
- Key-value pairs are normally managed into **buckets** (a logical grouping of keys).
- Key values must be unique within a bucket but can be duplicated across buckets.
- Operations of key-value databases are only *get*, *store*, and *delete*.
- All data operations based on the bucket plus the key. It is not possible to query the data based on anything in the value component of the key-value pair.

| 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"          |



# Key-value databases

- Twitter uses Redis to deliver your Twitter timeline.



# Key-value databases

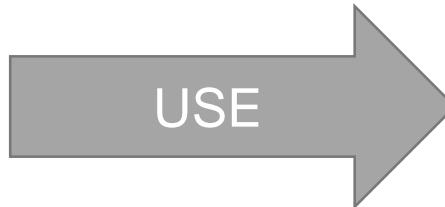
- Pinterest uses Redis to store lists of users, followers, unfollowers, boards, and more.



# Key-value databases

- Coinbase, a digital currency exchange, uses Redis to enforce rate limits and guarantee correctness of Bitcoin transactions.

coinbase



USE



# Key-value databases

- Quora uses Memcached to cache results from slower, persistent databases.



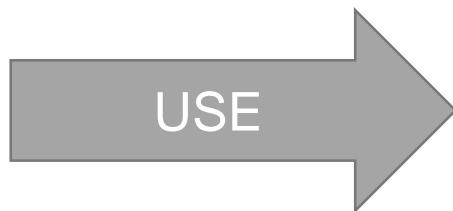
# Document-oriented database

- Document-oriented databases are inherently a subclass of the key-value database.
- The document can be in any encoded format, such as **XML**, **JSON** (JavaScript Object Notation), or **BSON** (Binary JSON).
- In Key-Value databases, the databases do not attempt to understand the content of the value component. However, the document databases do understand the meaning of the value component.
- Document-oriented databases are schema-less.
  - No predefined structure on the stored data.
  - Each document can have its own structure.
- Document-oriented databases group documents into logical groups called **collections**.

| Collection = Customer |                                                                                                   |
|-----------------------|---------------------------------------------------------------------------------------------------|
| Key                   | Document                                                                                          |
| 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"}              |

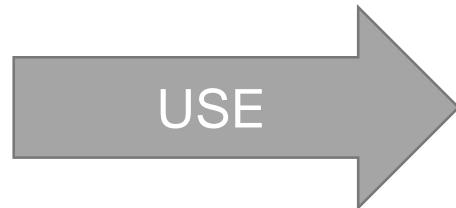
# Document-oriented database

- SEGA uses MongoDB for handling 11 million in-game accounts.



# Document-oriented database

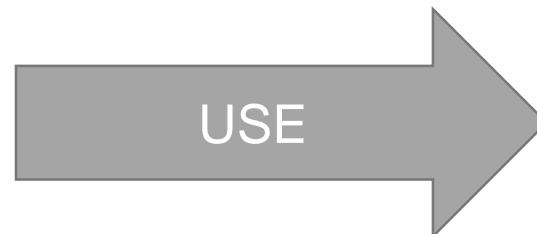
- Cisco moved its VSRM (video session and research manager) platform to Couchbase to achieve greater scalability.



Couchbase

# Document-oriented database

- Aer Lingus uses MongoDB with Studio 3T to handle ticketing and internal apps.



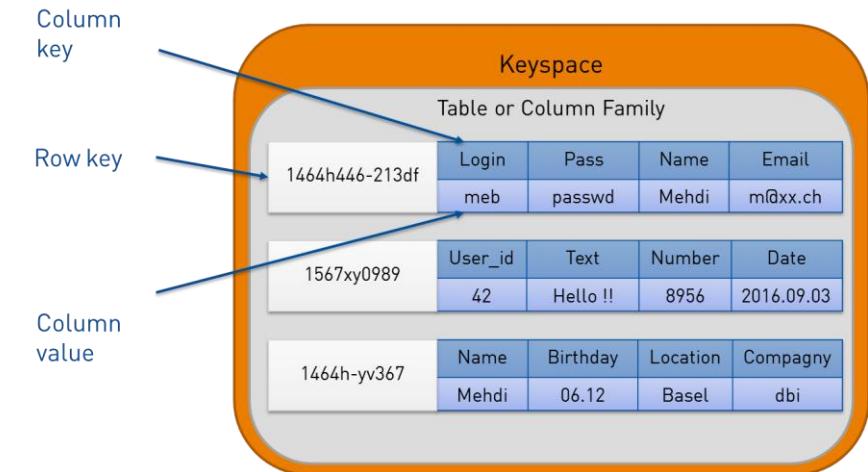
# Document-oriented database

- Built on MongoDB, The Weather Channel's iOS and Android apps deliver weather alerts to 40 million users in real-time.



# Column-oriented database

- Originated from Google's BigTable product, HBase, and Cassandra.
- Also known as *column family database*.
- Organize data in **key-value pairs** with keys mapped to a set of columns in the value component.
  - **Column**: a key-value pair similar to a cell of data in the relational database
  - **Key**: name of the column
  - **Value**: data that is stored in the column
- A group of columns that are logically related are called a *super column*. All of the super columns are grouped together to create a *column family*. A column family is conceptually similar to a table in the relational model.
- Each row key in the column family can have different columns.

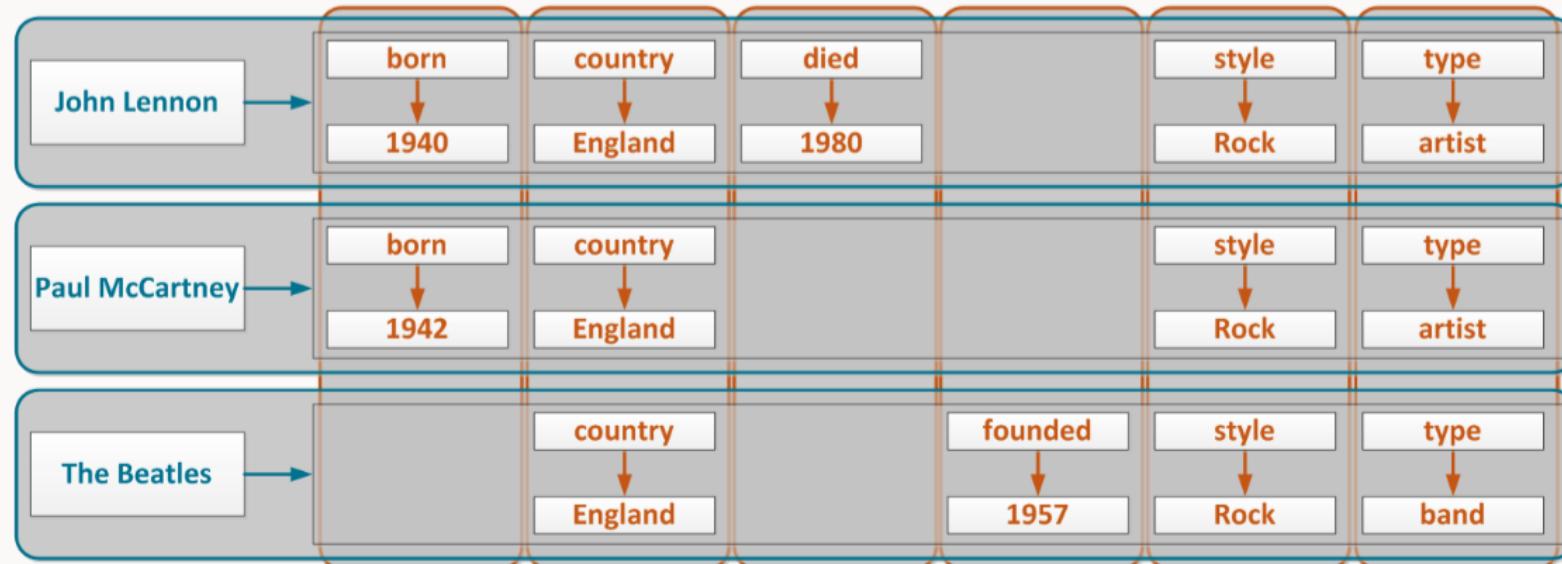


# Column-oriented database

- Table with single-row partitions

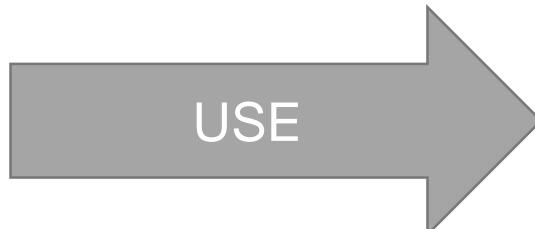


- Column family view



# Column-oriented database

- Spotify uses Cassandra to store user profile attributes and metadata about artists, songs, etc. for better personalization.



# Column-oriented database

- Facebook originally uses Cassandra to manage their Inbox Search.
- They initially built its revamped Messenger on top of HBase, but is now also used for other Facebook services like the Nearby Friends feature and search indexing.



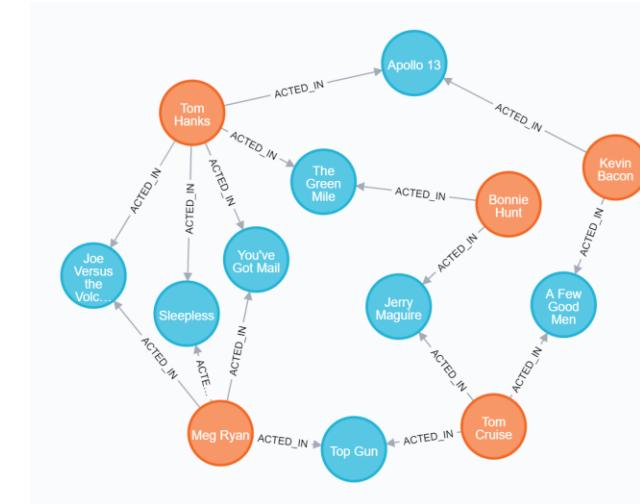
# Column-oriented database

- Outbrain, a web advertising platform, uses Cassandra to serve over 190 billion personalized content recommendations each month.



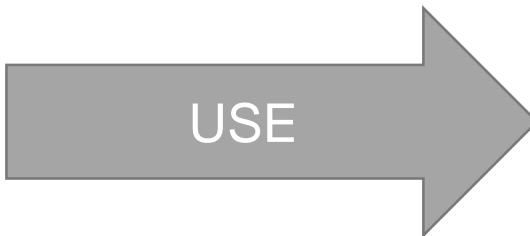
# Graph database

- Based on graph theory.
  - Components of graph databases are nodes, edges, and properties
- Store data about relationship-rich environment.
- Social networks such as Facebook, Twitter and Instagram are the drivers for developing graph databases.
  - Tracking relationships between objects especially in social networks becomes as important as the data itself.
- Characteristics: do not force data to fit predefined structures, do not support SQL, and are optimized to provide velocity of processing,
- Graph databases do not scale very well to clusters as they specialize in highly related data, not independent pieces of data.
- Perform best in centralized or lightly clustered environments similar to relational databases.



# Graph database

- Walmart uses Neo4j to provide customers personalized, relevant product recommendations and promotions.



# Graph database

- Medium, an online publishing platform, uses Neo4j to build their social graph to enhance content personalization.



**Medium**



# Graph database

- Cisco uses Neo4j to mine customer support cases to anticipate bugs.



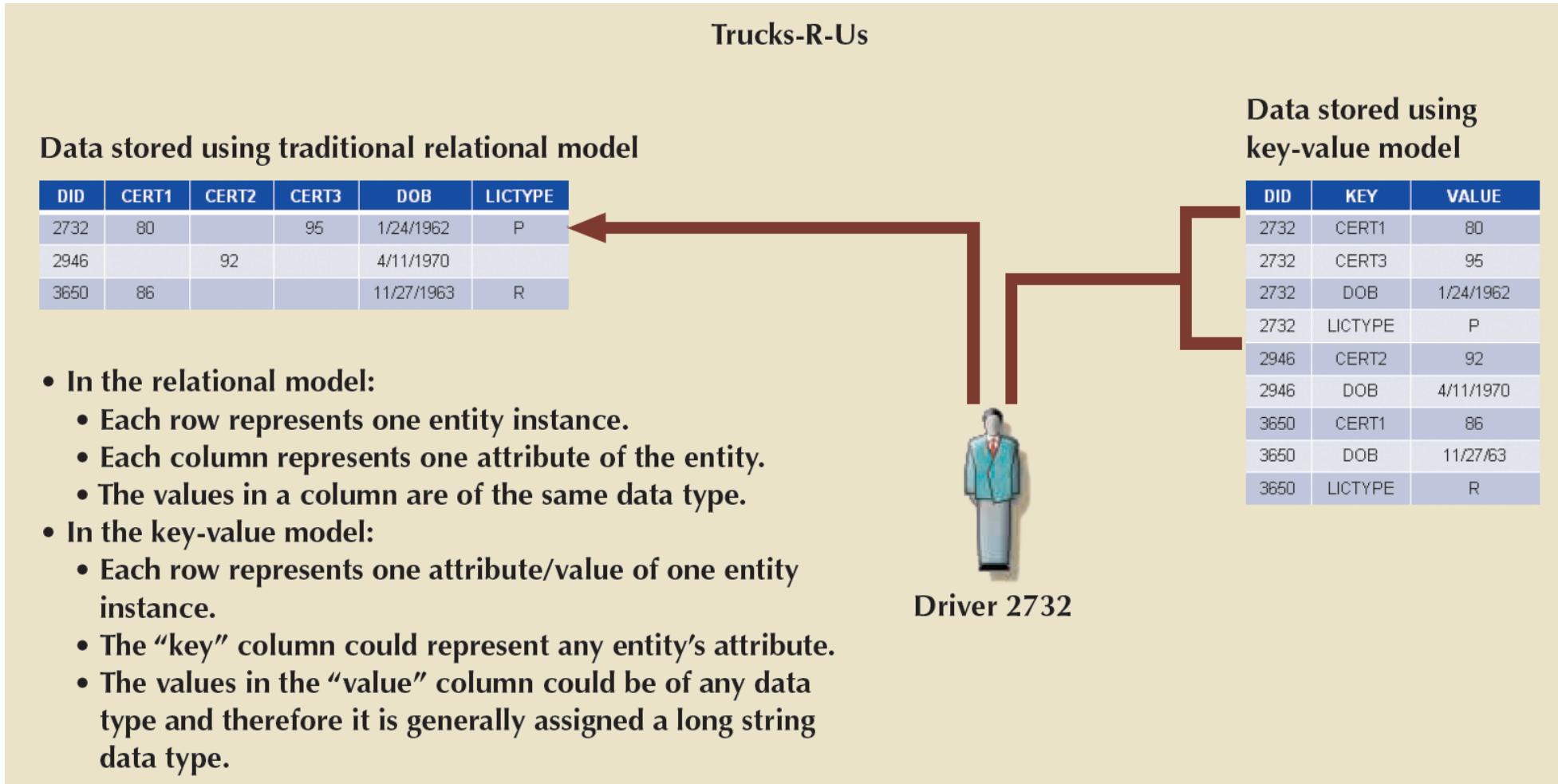
# NoSQL

| Advantages                                                                                                                                                                                                                                                    | Disadvantages                                                                                                                                                                                                                                                                                        |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ol style="list-style-type: none"><li>1. High scalability, availability, and fault tolerance are provided.</li><li>2. It uses low-cost commodity hardware.</li><li>3. It supports Big Data.</li><li>4. Key-value model improves storage efficiency.</li></ol> | <ol style="list-style-type: none"><li>1. Complex programming is required.</li><li>2. There is no relationship support—only by application code.</li><li>3. There is no transaction integrity support.</li><li>4. In terms of data consistency, it provides an eventually consistent model.</li></ol> |

# Relational Model

| Advantages                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | Disadvantage                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ol style="list-style-type: none"><li>1. Structural independence is promoted by the use of independent tables. Changes in a table's structure do not affect data access or application programs.</li><li>2. Tabular view substantially improves conceptual simplicity, thereby promoting easier database design, implementation, management, and use.</li><li>3. Ad hoc query capability is based on SQL.</li><li>4. Powerful RDBMS isolates the end user from physical-level details and improves implementation and management simplicity.</li></ol> | <ol style="list-style-type: none"><li>1. The RDBMS requires substantial hardware and system software overhead.</li><li>2. Conceptual simplicity gives relatively untrained people the tools to use a good system poorly, and if unchecked, it may produce the same data anomalies found in file systems.</li><li>3. It may promote islands of information problems as individuals and departments can easily develop their own applications.</li></ol> |

# NoSQL database vs RDBMS



# In this unit...

1. Document-oriented database:
  - W2, W3, W4



2. Column-oriented database:
  - W5, W6, W7



3. Graph database:
  - W8, W9, W10, W11



# References

- [1] C. Coronel and S. Morris, Database systems: *Design, implementation, & management*. Cengage Learning, 12<sup>th</sup> ed., 2016.
- [2] R. Ramakrishnan and J. Gehrke, Database Management Systems. New York, NY, USA: McGraw-Hill, Inc., 2<sup>nd</sup> ed., 2000.
- [3] MongoDB, “NoSQL Databases Explained” <https://www.mongodb.com/nosql-explained>.
- [4] Oracle, “What Is Big Data?” <https://www.oracle.com/au/big-data/guide/what-is-big-data.html>.
- [5] Studio 3T, “The Main NoSQL Database Types” <https://studio3t.com/whats-new/nosql-database-types/>.



**MONASH**  
University

MONASH  
INFORMATION  
TECHNOLOGY

# **FIT5137 – Advanced Database Technology**

## Week 1a – Overview

Semester 2, 2020

Developed by:

Dr. Agnes Haryanto

[Agnes.Haryanto@monash.edu](mailto:Agnes.Haryanto@monash.edu)

# Teaching Team

- Lecturer



**Agnes Haryanto**  
Agnes.Haryanto@monash.edu

- Tutors



**Farah Kabir (Head Tutor)**  
Farah.Kabir@monash.edu



**Arif Hidayat**  
Arif.Hidayat@monash.edu



**Alina Bhari**  
Alina.Bhari@monash.edu



**Chaluka Salgado**  
Chaluka.Salgado@monash.edu

- Chief Examiner



**Assoc. Prof. David Taniar**  
David.Taniar@monash.edu



**David Cheng**  
David.Chengzarate@monash.edu



**Ningran Li**  
Icley.Li@monash.edu



**Shuyi Sun**  
Shuyi.Sun@monash.edu



**Lily Xiaojiao Du**  
Xiaojiao.Du@monash.edu

# Overview

1. Non-relational database systems (NoSQL focused)
2. Main requirement:
  - Knowledge of relational database
  - Programming skill

# Learning Outcomes

On successful completion of this unit, students should be able to:

1. describe various types of non-relational database systems, including NoSQL;
2. design and model document-store and wide column-store databases;
3. compare and contrast between relational and non-relational database modelling;
4. explain the concepts of transactions in non-relational systems;
5. implement document-store and wide column-store systems;
6. construct applications using a graph database system;
7. demonstrate graph query processing.

# Why take this unit?

- Big Data era
- A lot of job opportunities but not enough experts
- Understand how big companies like Facebook, Google, and ebay manage their data

# Why take this unit?

LinkedIn Jobs ▾ NoSQL Australia Join now Sign in

Most relevant Any Time Company Location Job Type Experience Level

Turn on job alerts Off

308 Results for "NoSQL in Australia" (12 new)

**Solutions Architect - Startup**  
Amazon Web Services (AWS)  
Melbourne, Victoria, Australia  
The ideal candidate will possess customer facing skills that will allow them to present Amazon products and services well and speak with ...  
4 days ago

**Specialist Manager - Analytics & AI**  
Deloitte Australia  
Melbourne, Victoria, Australia  
Database and programming languages experience and data manipulation and integration skills using a range of cloud-based SQL and NoSQL ...  
6 days ago

**Engineer**  
ANZ  
Melbourne, Victoria, Australia  
We are looking for a skilled UI Engineer to join our team. To ensure success as a UI Engineer, you should have in-depth knowledge of ...  
2 weeks ago

**DynamoDB/NoSQL Solutions Architect**  
Amazon.com  
Sydney, New South Wales, Australia

**aws**

**Solutions Architect - Startup**  
Amazon Web Services (AWS) · Melbourne, Victoria, Australia  
4 days ago · 63 applicants  
 See who Amazon Web Services (AWS) has hired for this role  
[Apply on company website](#)

**Description**

Are you passionate about helping Startup Companies use cloud computing to develop and deploy large scale applications? Do you have a unique

Show more ▾

Seniority level Employment type  
**Not Applicable** Full-time

Job function Industries  
**Consulting, Information Technology, Engineering** Computer Software, Information Technology and Services, Internet

 Referrals increase your chances of interviewing at Amazon Web Services (AWS) by 2x

# Technologies you are going to learn

1. [MongoDB](#)



2. [Apache Cassandra](#)



3. [Neo4j](#)



# Job Opportunities



Jobs ▾

MongoDB

Australia



Join now

Sign in

Most relevant ▾

Any Time ▾

Company ▾

Location ▾

Job Type ▾

Experience Level ▾

Turn on job alerts

Off

171 Results for "MongoDB in Australia" (5 new)



**Engineer**

ANZ

Melbourne, Victoria, Australia

We are looking for a skilled UI Engineer to join our team. To ensure success as a UI Engineer, you should have in-depth knowledge of ...

2 weeks ago



**SQL Database Engineer**

Nuix

Sydney, New South Wales, Australia

Nuix is seeking highly motivated Microsoft SQL Database Administrators who are passionate about technology. Necessary skills include ...

4 weeks ago



**Python Developer**

FluroSat

Sydney, New South Wales, Australia

Simply something "huge" and "sexy" that they've "been looking for!". With experience with relational databases – either MySQL, ...

7 days ago



**Data Engineer with Python and Spark**

Parktrail Associates



**Engineer**

ANZ · Melbourne, Victoria, Australia

2 weeks ago · 52 applicants



See who ANZ has hired for this role

[Apply on company website](#)

We are looking for a skilled UI Engineer to join our team. In this role, you will be responsible for developing and implementing user interface components using React.js concepts and workflows such as Redux, Flux, and Webpack. You will also be responsible for profiling and improving front-end performance

Show more ▾

Seniority level

Not Applicable

Employment type

Full-time

Job function

Engineering, Information Technology

Industries

Accounting, Banking, Financial Services



MONASH  
University

# Job Opportunities

Find Jobs   Company Reviews   Find Salaries   Find Resumes   Employers / Post Job

**indeed**

What: Job title, keywords, or company  
Where:     Advanced Job Search

mongodb jobs

My recent searches: neo4j, Graph Database, Nosql, » clear searches

Sort by: relevance - date

Salary Estimate: \$90,000+ (216), \$110,000+ (119), \$130,000+ (42), \$150,000+ (21), \$170,000+ (12)

Job Type: Full-time (129), Permanent (67), Contract (41), Part-time (8), Temporary (6) [+ more »](#)

**Upload your resume - Let employers find you**

Page 1 of 253 jobs

**Database Administrator - MongoDB**  
Ignite, Melbourne VIC, **\$600 - \$900 a day**  
The successful candidate will have core knowledge and experience supporting MongoDB and have exposure in emerging cloud technologies....  
[Quick apply](#)

Sponsored [save job](#)

**Full Stack Developer**  
News Corp Australia ★★★★☆ 82 reviews, Sydney Central Business District NSW  
We are currently looking for an experienced Full Stack Developer to support revenue-generating, customer-facing web applications for our fast-paced global data...  
Sponsored [save job](#)

**Big Data Lead/ Architect**  
Cognizant Technology Solutions ★★★★☆ 11,524 reviews, Melbourne VIC, **\$100,000 - \$180,000 a year**  
Should have experience of using working on NoSQL data like HBase, Cassandra, and MongoDB etc. CTSH is a provider of information technology...

Get new jobs for this search by email  
My email:   
[Activate](#)

By creating a job alert or receiving recommended jobs, you agree to our [Terms](#). You can change your consent settings at any time by unsubscribing or as detailed in our terms.

Company with mongodb jobs: Ignite

Ignite: Ignite delivers specialist recruitment, on demand and outsourced people services in key specializations across Australia and China  
[Test Specialist](#), [ICT Testers](#), [Technical Writer](#)

[https://au.indeed.com/advanced\\_search?q=mongodb](https://au.indeed.com/advanced_search?q=mongodb)

# Job Opportunities



136 Results for "Cassandra in Australia" (5 new)



## Back End Senior Software Engineer

Agoda

Perth, AU

What we are looking for. Agoda Platform team is looking for developers to work on mission critical systems that deals with the designing ...

4 weeks ago



## Back End Senior Software Engineer

Agoda

Melbourne, AU

What we are looking for. Agoda Platform team is looking for developers to work on mission critical systems that deals with the designing ...

4 weeks ago



## Postgres DBA / Support Engineer

en world group

Sydney, New South Wales, Australia

2+ years of work experience as a PostgreSQL DBA, providing 24/7 support to mission-critical applications and databases. Our client, a ...

2 days ago · [Easy Apply](#)

Technical Lead



## Back End Senior Software Engineer

Agoda · Perth, AU

4 weeks ago · [Be among the first 25 applicants](#)

[Apply on company website](#)

***This is a full time position working onsite in our office in Bangkok. Full relocation and Visa sponsorship provided.***

### Overview

Agoda is a Booking Holdings (BKNG) company, the world's leading provider of brands that help people book great experiences through technology. And as a Booking Holdings company, we are part of the largest online travel company in the world. Technology is not just what we do – it's at the heart of who we are. We have the dynamism and short chain of command of a startup and the capital to make things happen. We love innovation and putting new technologies to work to extend our lead on the competition.



MONASH  
University

# Job Opportunities

[Find Jobs](#)[Company Reviews](#)[Find Salaries](#)[Find Resumes](#)[Employers / Post Job](#)**What**

Job title, keywords, or company

cassandra

**Where****Find jobs**[Advanced Job Search](#)

cassandra jobs

My recent searches

mongodb

neo4j

Graph Database

Nosql

[» clear searches](#)

Sort by:

relevance - date

Salary Estimate

\$80,000 + (106)

\$100,100 + (70)

\$120,000 + (37)

\$140,000 + (19)

\$162,200 + (10)

Job Type

Full-time (49)

Permanent (33)

Contract (18)

Temporary (7)

Internship (2)

[Upload your resume - Let employers find you](#)

Page 1 of 122 jobs

## ETL Solution Modeller

Cognizant Technology Solutions ★★★★★ 11,524 reviews

Sydney NSW

**\$80,000 - \$150,000 a year**

Experience with NoSQL databases, such as HBase, Cassandra, MongoDB or similar. CTSH) is a leading provider of information technology, consulting, and business...

[Quick apply](#)Sponsored [save job](#)

## Microservice Framework Developer-Lead

Softvision ★★★★★ 15 reviews

Rhodes NSW

**\$90,000 - \$110,000 a year**

Hands on experience in No sql and time-series databases ( Mongo/ Cassandra/ InfluxDb/ Graphite). Softvision creates impactful end-to-end digital products and...

[Quick apply](#)Sponsored [save job](#)

## Bigdata Developer

Cognizant Technology Solutions ★★★★★ 11,524 reviews

Sydney NSW

**\$100,000 - \$140,000 a year**

Get new jobs for this search by email

My email:

**Activate**

By creating a job alert or receiving recommended jobs, you agree to our [Terms](#). You can change your consent settings at any time by unsubscribing or as detailed in our terms.

# Job Opportunities

[Find Jobs](#)[Company Reviews](#)[Find Salaries](#)[Find Resumes](#)[Employers / Post Job](#)

**What**  
Job title, keywords, or company

**Where****Find jobs**[Advanced Job Search](#)[Graph Database jobs](#)[My recent searches](#)[Nosql](#)[» clear searches](#)[Sort by:](#)[relevance - date](#)[Salary Estimate](#)[\\$81,800 + \(26\)](#)[\\$103,700 + \(15\)](#)[\\$125,700 + \(9\)](#)[\\$144,900 + \(6\)](#)[Job Type](#)[Full-time \(16\)](#)[Permanent \(10\)](#)[Contract \(4\)](#)[Location](#)[Sydney NSW \(14\)](#)**Upload your resume - Let employers find you**

Page 1 of 31 jobs

## BigData Analyst

**Cognizant Technology Solutions** ★★★★★ 11,524 reviews

Perth WA

**\$70,000 - \$100,000 a year**

Create data definitions for new database file/table development and/or changes to existing ones as needed for analysis....

**Quick apply**Sponsored [save job](#)

## Senior Consultant - Big Data

**Deloitte** ★★★★★ 8,213 reviews

Melbourne VIC

RDBMS, Big Data, columnar, NoSQL or graph databases); This could be a data mapping, a database, a migration/ingestion prototype or a solution roadmap ;...

Sponsored by **FlexCareers** [save job](#)

## Senior Consultant - Data Engineering

**Deloitte** ★★★★★ 8,213 reviews

Sydney NSW

RDBMS, Big Data, columnar, NoSQL or graph databases); This could be a data mapping, a database, a migration/ingestion prototype or a solution roadmap ;...

Get new jobs for this search by email

My email:

**Activate**

By creating a job alert or receiving recommended jobs, you agree to our [Terms](#). You can change your consent settings at any time by unsubscribing or as detailed in our terms.

# Job Opportunities

[Find Jobs](#)[Company Reviews](#)[Find Salaries](#)[Find Resumes](#)[Employers / Post Job](#)

What

Job title, keywords, or company

neo4j

Where

[Find jobs](#)[Advanced Job Search](#)

neo4j jobs

My recent searches

[Graph Database](#)[Nosql](#)[» clear searches](#)

Sort by:

[relevance - date](#)

Salary Estimate

\$100,600 + (7)

\$131,800 + (4)

Job Type

Full-time (5)

Permanent (3)

[Upload your resume - Let employers find you](#)

Page 1 of 9 jobs

## Data Warehouse Engineer

Latitude Financial Services ★★★★☆ 13 reviews

Docklands VIC

Solid experience in multiple database technologies such as Distributed Processing (Spark, EMR), traditional RDBMS (MS SQL Server, Oracle, MySQL, PostgreSQL),...

9 days ago [save job](#) more...

## Backend Software Engineer

Megaport ★★★★☆ 8 reviews

Brisbane QLD

We need a senior back end developer familiar with the Spring/Java stack, and interested in applying technologies such as Kotlin, Neo4J, Machine Learning etc....

8 months ago [save job](#) more...

## Full-Stack Data Scientist

Opus Recruitment Solutions ★★★★☆ 7 reviews

Sydney NSW

**\$120,000 - \$150,000 a year**

ANALYTICS | CUSTOMER INSIGHTS | SYDNEY | PYTHON |  
RAPID GROWTH | SaaS | GLOBAL SCALABILITY AND  
PRESENCE | MACHINE LEARNING | BEHAVIOURAL  
INSIGHTS | MARKETING...

[https://au.indeed.com/advanced\\_search?q=neo4j](https://au.indeed.com/advanced_search?q=neo4j) days ago [save job](#) more...

Get new jobs for this search by email

My email:

[Activate](#)

By creating a job alert or receiving recommended jobs, you agree to our [Terms](#). You can change your consent settings at any time by unsubscribing or as detailed in our terms.

# Companies using MongoDB

 mongoDB. Products Solutions Customers Resources

Learn What is MongoDB? Contact Search Sign In 

Try Free



Personalization based on real-time  
data is the key success factor for e-  
commerce sites."

— Peter Wolter, OTTO

 Tweet



The best thing from the get-go was  
MongoDB's engineers and  
developer-first approach."

— Tony Tam, Reverb Technologies





# Companies using Cassandra



## Who Uses Cassandra?



GitHub



CISCO  
CISCO



GAMEFLY



comcast



MONASH  
University

# Companies using Cassandra

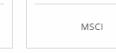


## Who else Uses Cassandra?



MONASH  
University

# Companies using Neo4j

|                                                                                                                          |                                                                                                             |                                                                                                                                          |                                                                                                                                 |                                                                                                      |                                                                                                                          |                                                                                                                          |                                                                                                                                    |                                                                                                                          |                                                                                                                                                  |                                                                                                                           |                                                                                                                         |                                                                                                     |
|--------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------|
|  <b>Walmart</b>                          |  <b>comcast</b>            |  <b>Adobe</b>                                           |  <b>eBay</b>                                   |  <b>Mesosphere</b>  |  <b>COMMIT</b>                          |  <b>KCOM Group</b>                    |  <b>gousto</b>                                  |  <b>Pitney Bowes</b>                  |  <b>Europeana</b>                                             |  <b>k12</b>                            |  <b>Glowbl</b>                       |                                                                                                     |
|  <b>ICIJ</b><br>The ICIJ - Panama Papers |  <b>digitate</b>           |  <b>LOCKHEED MARTIN</b>                                 |  <b>U.S. Army</b>                              |  <b>onefinestay</b> |  <b>maaii<br/>Let's connect</b>         |  <b>ic entropy</b>                    |  <b>ProIntegra</b>                              |  <b>Fortune 200<br/>Hospitality</b>   |  <b>e-Spirit</b>                                              |  <b>TOMTOM</b>                         |  <b>Medium</b>                       |                                                                                                     |
|  <b>dun&amp;bradstreet</b>               |  <b>Global 50<br/>Bank</b> |  <b>DZD<br/>Deutsches Zentrum für Diabetesforschung</b> |  <b>PERFORM<br/>SPORTS CONTENT &amp; MEDIA</b> |  <b>Sodifrance</b>  |  <b>zenoss</b>                          |  <b>doximity</b>                      |  <b>metic.fr</b>                                |  <b>Softwire</b>                      |  <b>megree</b>                                                |  <b>FACEIT<br/>CHALLENGE YOUR GAME</b> |  <b>LINKURIOUS</b>                   |                                                                                                     |
|  <b>airbnb</b>                           |  <b>UBS</b>                |  <b>MONSANTO</b>                                        |  <b>Microsoft</b>                              |  <b>LARUS</b>       |  <b>NII<br/>国家情報学研究所</b>                |  <b>dop</b>                           |  <b>VIRTUAL<br/>INSTRUMENTS</b>                 |  <b>chillisoft</b>                    |  <b>seedcloud<br/>bringing great technology ideas to life</b> |  <b>innoQ</b>                          |  <b>Stitched</b>                     |                                                                                                     |
|  <b>IBM</b>                              |  <b>novo nordisk</b>       |  <b>ICIJ</b><br>The ICIJ - Swiss Leaks Story            |  <b>CATERPILLAR</b>                            |  <b>workjam</b>     |  <b>Pondera Solutions</b>               |  <b>GRAPHILEON</b>                    |  <b>ACUO</b>                                    |  <b>nulli<br/>Identity Management</b> |  <b>LendingClub</b>                                           |  <b>InfoAdvisors</b>                   |  <b>WORLD<br/>ECONOMIC<br/>FORUM</b> |                                                                                                     |
|  <b>Previa</b>                          |  <b>VOLVO</b>             |  <b>AstraZeneca</b>                                    |  <b>Fortune 50<br/>Retailer</b>               |  <b>audiocodes</b> |  <b>CAMBRIDGE<br/>UNIVERSITY PRESS</b> |  <b>holaa</b>                        |  <b>MARLINK<br/>Connect smarter. Anywhere.</b> |  <b>FINO</b>                         |  <b>kantwert</b>                                             |  <b>GLIDEWELL<br/>LABORATORIES</b>    |  <b>MIROCULUS</b>                   |                                                                                                     |
|  <b>KERBEROS</b>                       |  <b>die Bayerische</b>   |  <b>telenor</b>                                       |  <b>NASA</b>                                 |  <b>nettbus</b>   |  <b>ROPOSO</b>                        |  <b>SG<br/>SCIENTIFIC GAMES</b>     |  <b>scribestar</b>                            |  <b>Novartis</b>                    |  <b>DB</b>                                                  |  <b>VINELAB</b>                      |  <b>TECH MARIONETTE</b>            |                                                                                                     |
|  <b>WDS</b>                           |  <b>orange</b>           |  <b>Telia</b>                                         |  <b>StarHub</b>                              |  <b>Cerved</b>    |  <b>atpco</b>                         |  <b>Swiss International Air Lines</b> |  <b>FT<br/>FINANCIAL<br/>TIMES</b>              |  <b>LinkedIn China</b>                |  <b>Cisco</b>                                                 |  <b>MSCI</b>                           |  <b>CenturyLink</b>                  |  <b>migRaven</b> |

# Topics Covered

- NoSQL databases
  - 1. Document-oriented database
  - 2. Column-oriented database
  - 3. Graph database
- Explore both the practical and theoretical sides of NoSQL technologies

# Software

1. MongoDB – Shell: version 4, Compass: 1.21.2

<https://www.mongodb.com/try/download/shell>

<https://www.mongodb.com/try/download/compass>

2. Apache Cassandra 3.11

<http://cassandra.apache.org/download/>

3. Neo4j 1.3.3

<https://neo4j.com/download/>

# Lectures

- Lecture recordings – available under the Weekly resources on Moodle
- Lecture Q&A Session (on Zoom)
  - Clayton Campus: Thursday 18:00

Check Class Streaming page on Moodle for Zoom link

# Laboratories

- Labs and times:
  - Check Allocate+ for your schedule
  - Check Class Streaming page on Moodle for Zoom link
  - Any problem with the time allocation? Contact the **Timetabling Team**
- Labs:
  - Labs start from **Week 2**.
  - Please note that after we return to campus, only the Labs will be offered in a semi-face to face format with alternate weeks being on campus. More information will be given when we get to that point.

# Assessments

| Assessment task                      | Value | Due date                                      | Notes                                                                       |
|--------------------------------------|-------|-----------------------------------------------|-----------------------------------------------------------------------------|
| Assignment 1 - MongoDB and Cassandra | 20%   | Week 8 - Wednesday, 7 October 2020, 11:55pm   | Group of 2 students                                                         |
| Assignment 2 - Neo4j                 | 10%   | Week 12 - Wednesday, 4 November 2020, 11:55pm | Individual assessment                                                       |
| FLUX Participation                   | 5%    | Quiz opens from Monday to Wednesday each week | 80% participation is required                                               |
| Tutorial Work                        | 5%    | During tutorial Week 4, 5, 7, 9, 10           | Combination of Coding exercises and other activities, Individual assessment |
| Examination                          | 60%   | To be advised                                 | 2 hours 10 minutes e-Exam                                                   |

# Assessments

- Unit's assessments consist of non-exam components (40%) and exam (60%)
- To pass the unit you must attempt/achieve:
  - non-exam components, and the examination
  - > 45% of the possible marks in the non-exam assessments
  - > 45% of the possible marks in the exam
  - > 50% of possible marks

# Using FLUX

1. Visit <http://flux.qa/> on your internet enabled device
2. Log in using your Monash account (not required if you are already logged in to Monash)
3. Click on the “+” to join audience
4. Enter the Audience Code: **B79W3E**
5. Select FIT5137 in the Active Presentation menu
6. Answer questions when they pop up

The screenshot shows a web browser window titled "FLUX" with the URL <https://flux.qa/#/feeds/5d368...>. The page is titled "Lecture 0". Below the title are four icons: a bar chart, a person icon, a gift icon, and a clock icon. The main content area is titled "Current Polls". It displays a multiple-choice poll with the question "What did you have for dinner?". The options are listed as follows:

- A Pasta
- B Rice
- C Pizza
- D Salad
- E Nothing



# FLUX question

- My previous database study was:
  - A. FIT9132
  - B. FIT2094
  - C. FIT3171
  - D. I received credit based on other database studies
  - E. I did not do any database study