dm22s1

Topic 03 : Data Handling

Part 01 : Data-Storage

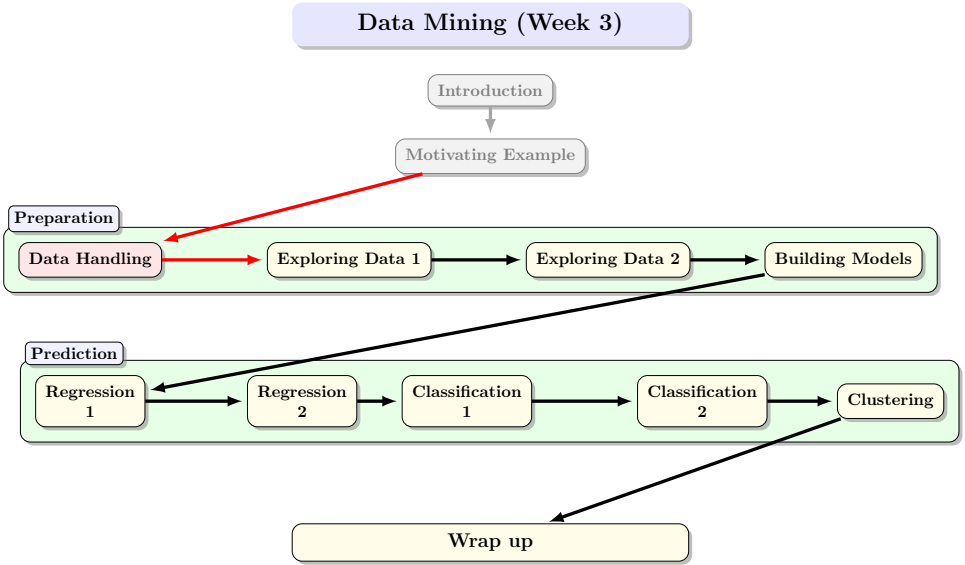
Dr Bernard Butler
Department of Computing and Mathematics, WIT.
(bernard.butler@setu.ie)

Autumn Semester, 2022

Outline

- Types of storage model
- Storage operations

Where we are in the module



Review of Weeks 1 and 2

- The module introduction provided an overview of the module as a whole, outlining the history of data mining, process models, the machine learning topics we cover and how the module is delivered and assessed
- The first lab was about
  - installing the necessary software on your laptop
  - using your existing knowledge and skills to investigate the marks dataset
- The pandas overview features very helpful advice which will prove invaluable to you in programming assessments
- We classified the iris data using a simple algorithm, showing that machine learning does not need to be complicated
- The second lab was an opportunity to exercise your new pandas skills on the marks dataset
- To date, we have considered small datasets, which are OK for teaching, but "real world"' data is larger and more messy
- Today we continue this introduction as we cover foundational aspects, this time regarding *Data Engineering*: how data is stored and processed at scale
- In later weeks, our attention switches to the *machine learning* pipeline, which forms the bulk of this module

Overview of today's lecture

Data Infrastructures

- Data infrastructures form the foundation of data mining
- Data does not exist in a vacuum
  - it is stored somewhere, if only temporarily
  - it moves somewhere else (for a variety of reasons)
- Data can be represented in many formats:
  - basic, with separate metadata: CSV, TSV, COBOL-based, ...
  - complex, with metadata included: XML, JSON, YAML, Avro, ...
- Data can be "text" or binary-valued
- Data operations can be represented as CRUD or HTTP verbs

Why is data stored?

There are two basic use cases for data storage

Persistence

Within an application, we often need to save data temporarily when handing it over to a separate entity for further processing. Often, the data is saved from/loaded into memory to make recovery easier. Generally, data persisted like this has little visibility or use for data mining purposes.

Archive

Between applications, data can be saved to more permanent storage as a way of keeping a record of key results, e.g., bank transactions, for audit, BI or similar purposes. Generally, data stored in this way *is useful* for data mining purposes.

Note that data mining workflows are themselves applications! However, they use persistence sparingly (because disk operations are slow) and add value to archived data.

Databases and Database Management Systems: A review

Definition 1

A database is a collection of data that is organised to facilitate data management operations, especially *Reading*, *Writing*, *Updating* and *Deleting*.

Definition 2

A DBMS is *software* that manages a database, and enables users to apply data management operations, to control access to the data and to configure how it is stored.

DBMS have a long and rich history, with many interesting aspects that are out of scope here. For data mining purposes, the main interest is in the data model, format and access method.

The Data Model

Definition 3

A data model is an abstract model that organizes elements of data and standardizes how they relate to one another and to properties of the real world entities they represent.

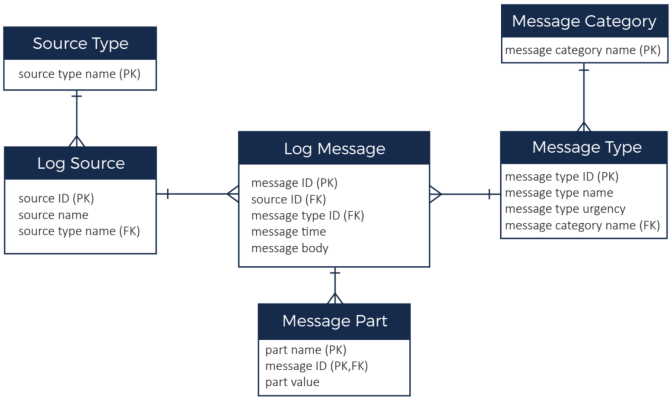
Data models come in a multitude of forms, but a common classification is into *relational* and *non-relational*.
They represent a trade-off between competing objectives, namely the model's alignment with how the data is

- structured for data input (Reads)
- structured for data output (Writes and/or Updates)
- viewed by its owners/users (stakeholders have their own perspectives)
- to be placed in the physical storage infrastructure, which is often distributed/replicated for scalability and/or resilience

The Relational Model

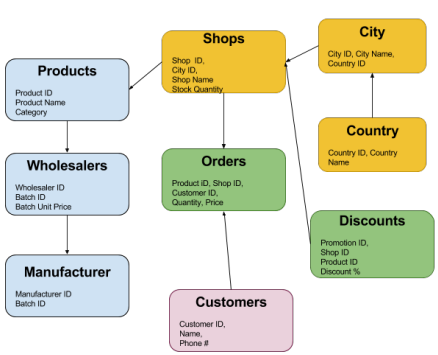
The diagram says that entities like `Message Type` and `Log Message` are linked by a Primary Key-to-Foreign Key relationship in the sense that each `Log Message` has a single `Message Type`, but the same `Message Type` can be assigned to many `Log Message`s.

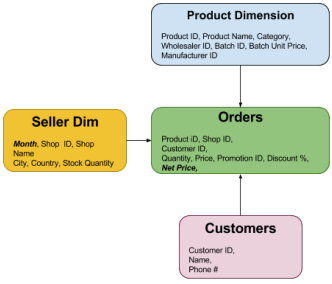
The relational model is well-established, commonly used and (in normalised form) is a good fit to many OLTP workloads. For OLAP and/or mainly Read operations, a normalised model is more convenient.



Source: https://www.instaclustr.com/cassandra-nosql-data-model-design-2/

Data normalisation



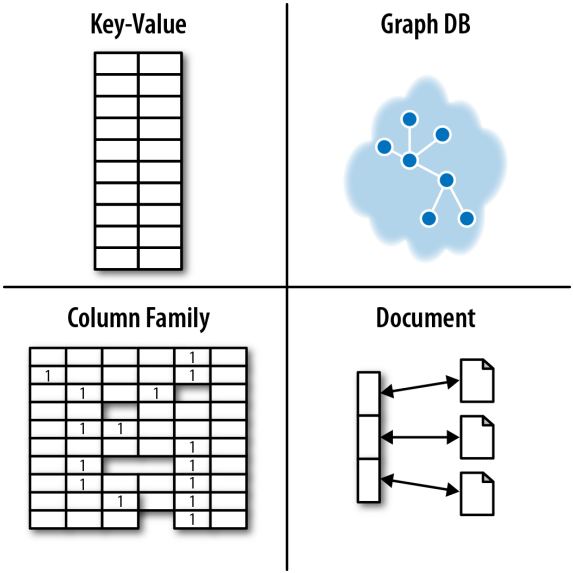


Source: https://www.techinasia.com/4-ways-fix-slow-dashboards

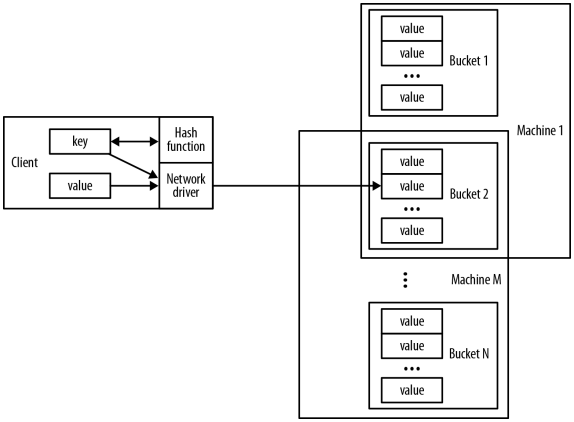
Normalised data is broken into smaller pieces, so Create, Update and Delete operations tend to be more specific (smaller) and the data storage requirement is generally smaller too. So (non-concurrent) Write performance can be good, but concurrent writes are slowed by the need for *locks*. By contrast, denormalised data often has redundancy but that can be good (better support for error-checking) and users of the data often prefer to get the data in one piece, rather than having to join it together (more effort and a *leaky abstraction*) at Query (Read) time.

NoSQL database options

SQL is the most common language for accessing relational databases. Non-relational DBMS often use (proprietary) languages based on SQL. NoSQL databases are often designed with high availability and scalability as critical requirements. NoSQL databases are indicated for massive, append-only use cases such as system or user-content interaction logs.



Key-Value databases



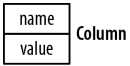
- Key-value data model is very simple.
- Logically it is a distributed hashmap of *aggregate*s (single values or data structures comprising multiple values).
- The type of each aggregate does not need to be the same
- The keys are hashed and mapped onto *buckets* that are distributed across *machines*.
- Examples include Dynamo, Redis, MemcacheDB, etc.

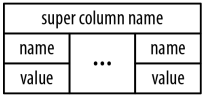
Document databases

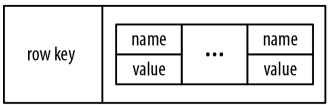
Relational	Document-oriented
database	database
table	collection
row	document
table join	embedded/linked documents
rows have same structure	row structure can vary

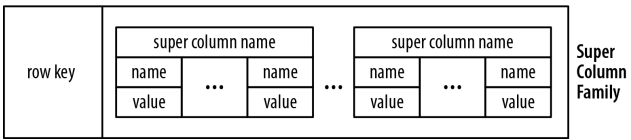
Document-oriented database is a subclass of key-value stores. The value is no longer *opaque*, but it has structure (often a JSON fragment). Often explicit *sharding* is used to distribute documents. Examples include MongoDB, CouchDB.

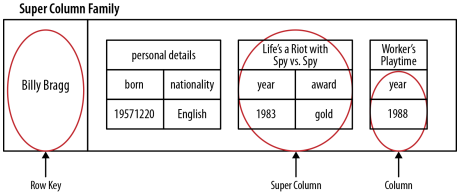
Column Family databases





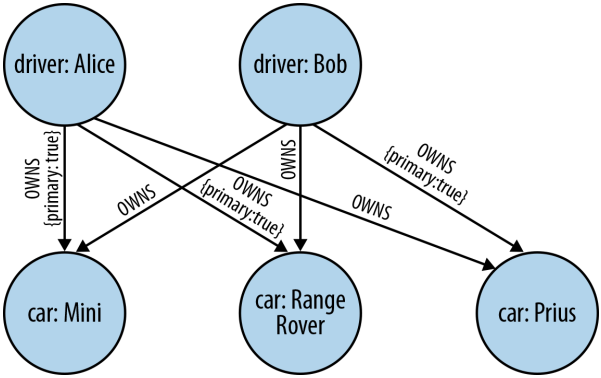






Logically it is a hashmap of hashmaps (nested key-value pairs, grouped). The inner hashmaps are like rows/documents. There is lots of flexibility, e.g., they handle sparsity well. The model was introduced by Google BigTable, and can be found in Hadoop HBase (realised as partitions containing buckets) and (in modified form) in Cassandra.

Graph databases



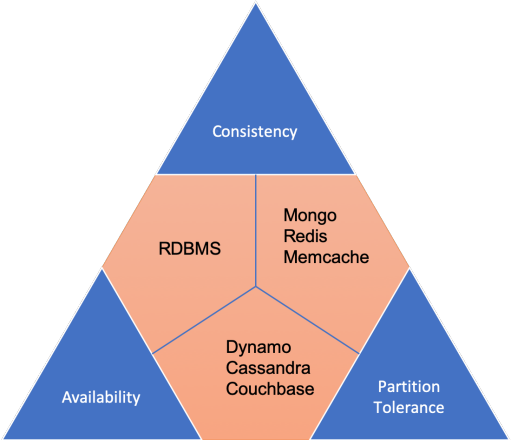
Labeled property graph model shown, e.g., Neo4j, can be implemented with index-free adjacency. Alternative graph model is RDF triple store, for semantic web linked data. Triple stores (e.g., Jena) use extra nodes instead of properties. Languages include cypher, gremlin and SPARQL. Excellent for (social) network applications.

NewSQL databases

Feature	OldSQL	NoSQL	NewSQL
Relational	✔	✘	✔
SQL	✔	✘	✔
ACID transactions	✔	✘	✔
Horizontal scalability (deployment)	✘	✔	✔
Scalability (performance)	✘	✔	✔
Schema-less	✘	✔	✘

NewSQL databases share many of the advantages of “OldSQL” and NoSQL. Three categories exist: optimised storage engines, automatic sharding, and new architectures where distributed computing is central, rather than an add-on. Examples include Google Spanner, MS Azure Cosmos DB, VoltDB, NuoDB.

CAP Theorem



Source: https://www.linkedin.com/pulse/cap-theorem-architectural-tool-vinay-avasthi/

- Conjecture (Brewer, 2000): databases can offer no more than 2 of C,A,P at a time when a failure occurs.
- Thus databases tend to pick 2 of the 3 to guarantee.
- When *partitions* occur, a relational DBMS cannot guarantee C and A simultaneously - it has to address one first.
- By contrast, A CP DBMS like Mongo would have difficulty with Availability when a failure occurs.
- Clever database administration can minimise problem, but the trade-off remains a key factor when choosing DBMS type.

DBMS “Chemistry”: ACID vs BASE

ACID	BASE
Strong consistency	Weak consistency
Isolation	Availability first; last write wins
Pessimistic (uses locks))	Optimistic
Nested transactions	Best effort: approximate answers
Available/Consistent	Available/Partition-tolerant
Robust Database/Simple Code	Simple Database/Complex Code

- Atomicity, Consistency, Isolation, Durable: strong guarantees on Consistency - transactions must complete or roll back to a consistent state; transactions do not interfere with each other.
- Basic Availability, Soft state, Eventual consistency: strong guarantees on Availability - transaction handling is approximate, so Consistency might suffer in the short term and might not ever be fully achieved.

Pandas and tabular data: input and output

Pandas provides a wide range of input and output options* for tabular formats, not just CSV files.

SQL-based

- SQLalchemy[a] is a python-friendly wrapper for ODBC.
- Using a SQLalchemy connector object, it is easy to connect to SQL databases.
- To read from table or query into a dataframe, use `pd.read_sql(...)`.
- To write dataframe `df` to a database, use `df.to_sql(...)`.

[a] https://www.sqlalchemy.org/

Not SQL-based

- There is a wide variety of "tabular" formats that do not use SQL for access.
- They include generic formats like CSV, parquet and even the clipboard.
- Read using `pd.read_XXX(...)` and write using `df.to_XXX(...)`.
- For proprietary formats like SAS, only `pd.read_XXX(...)` is available.

Tabular data and dataframes can be exchanged easily because they have the same "shape".

* https://pandas.pydata.org/pandas-docs/stable/reference/io.html

Pandas and nontabular data: input and output

- XML and JSON can store hierarchical (nested) data.
- This needs to be flattened to interoperate with dataframes.
- `pd.read_xml(...)` has the option to apply an XSLT stylesheet when reading.
- `pd.read_json(...)` has an `orient` parameter for a similar purpose.
- Databases like mongodb and redis generally have specific connection libraries like `mongodbclient`
- The developer is responsible for parsing and transforming the data to/from the database.

Summary of Data Storage

- Data flows through data mining workflows, so its management and provenance is critical
- Operational databases are a common source of such data
- Data structure and relationships need to be considered in any workflow
- Many data mining tools prefer data structured as *dataframes* (generalised tables)
  - Denormalised data is preferred by ML tools
  - Operational data from relational and graph databases tend to be normalised
  - Other data stores tend to be denormalised, but need code to "unpick" the data elements
- BASE might not be good enough for anomaly detection systems: higher probability of false positives!
- Pandas provides functions to exchange data between dataframes and databases.

References

Learning Spark: Lightning-Fast Big Data Analysis, Holden Karau, Andy Konwinski, Patrick Wendell and Matei Zaharia. 2015 O'Reilly Media.

High Performance Spark: Best practices for scaling & optimizing Apache Spark Holden Karau and Rachel Warren. 2017 O'Reilly Media.

Graph Databases: New opportunities for connected data, Ian Robinson, Jim Webber and Emil Eifrem. 2015 O'Reilly Media.