AQA Computer Science A-Level

4.11.1 Big Data

Advanced Notes

Specification:

4.11.1 Big Data:

Know that 'Big Data' is a catch-all term for data that won't fit the usual containers. Big Data can be described in terms of:

- Volume – too big to fit into a single server
- Velocity – streaming data, milliseconds to seconds to respond
- Variety – data in many forms such as structured, unstructured, text, multimedia

Know that when data sizes are so big as not to fit on to a single server:

- The processing must be distributed across more than one machine
- Functional programming is a solution, because it makes it easier to write correct and efficient distributed code

Know what features of functional programming make it easier to write:

- Correct code
- Code that can be distributed to run across more than one server

Be familiar with the:

- Fact-based model for representing data
- Graph schema for capturing the structure of the dataset
- Nodes, edges and properties in graph schema

Big Data

The name "big data" is a catch-all term for data that doesn't fit the usual containers. The three defining features of big data can be remembered as "the three Vs":

Volume

There is too much data for it all to fit on a conventional hard drive or even a server. Data has to be stored over multiple servers, each of which is composed of many hard drives.

Velocity

Data on the servers is created and modified rapidly. The servers must respond to frequently changing data within a matter of milliseconds.

Variety

The data held on the servers consists of many different types of data from binary files to multimedia files like photos and videos.

While you may be tempted to think that the volume of big data is its most challenging attribute, it turns out that big data's lack of structure causes the most trouble.

Big data's unstructured nature makes it difficult to analyse the data. Conventional databases are not suited to storing big data because they require the data to conform to a row and column structure. Furthermore, conventional databases do not scale well across multiple servers.

In order to extract useful information from big data, machine learning techniques must be used to discern patterns in the data.

Examples of big data include continuously monitored banking interactions and data from surveillance systems.

When data is stored over multiple servers, as is the case with big data, the processing associated with using the data must also be split across multiple machines. This would be incredibly difficult with conventional programming paradigms as the machines would all have to be synchronised to ensure that no data is overwritten or otherwise damaged.

Functional programming

Functional programming is a solution to the problem of processing data over multiple machines. Functional programs are stateless (meaning that they have no side effects) and make use of immutable data structures. Furthermore, the functional programming paradigm supports higher-order functions.

These attributes are what makes it easier to write correct, efficient, distributed code with functional programming than with procedural programming techniques.

Synoptic Link

Functional programming is a programming paradigm that solves problems by executing functions.

Functional programming is covered in **fundamentals of functional programming**.

Higher-order function

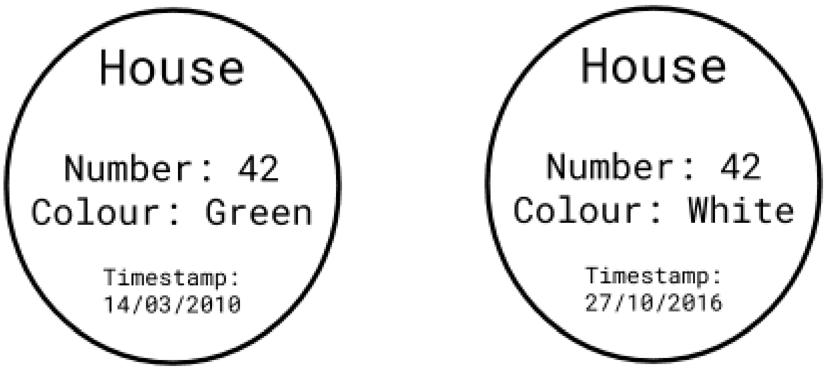
A function which takes functions as its input and / or outputs a function.

The fact-based model for representing data

Because big data doesn't conform to the row and column format typically used to represent data, it must be represented in a different manner. One way of representing big data is with the fact-based model.

In the fact-based model, each individual piece of information is stored as a fact. Facts are immutable (meaning that they never change once created) and can't be overwritten.

Stored with each fact is a timestamp, indicating the date and time at which a piece of information was recorded. Seeing as facts are never deleted or overwritten, multiple different values could be held for the same attribute. This is where timestamps come in, allowing a computer to discern which value is the most recent.



For example, the two facts above contain the number and colour of a house at two different points in time. The house was green in 2010 but re-painted white in 2016. If this information were stored in a dataset and the colour of house number 42 was queried, the two timestamps would be compared and the most recent information (`Colour: White`) would be returned.

Thanks to facts being immutable (and therefore not overwritable), using the fact-based model for storing big data reduces the risk of accidentally losing data due to human error. Moreover, the model does away with an index for the data and instead simply appends new data to the dataset as it is created.

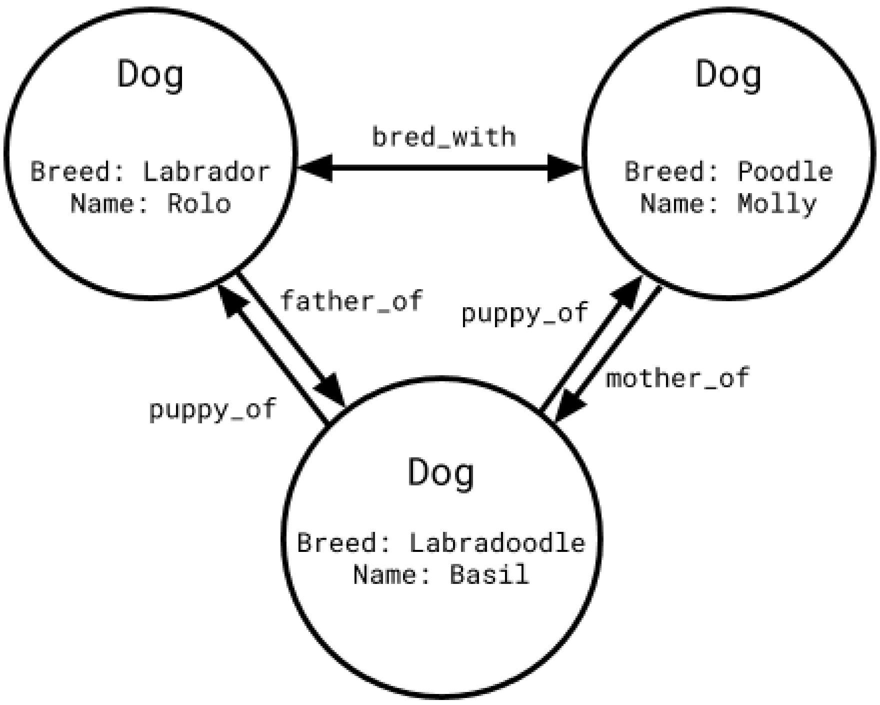
Representing big data using graph schema

Graph schema uses graphs consisting of nodes and edges to graphically represent the structure of a dataset. Nodes in a graph represent entities and can contain the properties of the entity.

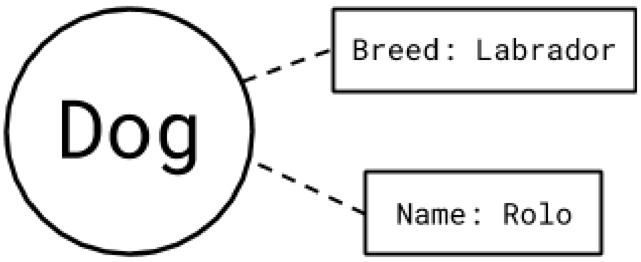
Edges are used to represent relationships between entities and are labelled with a brief description of the relationship.

The example on the right shows three entities represented in a graph schema as three circles. The properties of each entity (`breed` and `name`) are listed inside of the circles. Arrows linking the circles represent the relationships between the nodes.

Timestamps are rarely included in graph schema diagrams, instead you should assume that each node contains the most recent information available.



An accepted alternative to the method of representing properties used in the example above is to list an entity's properties inside rectangles joined to entities with a dashed line. The dashed lines do not represent relationships like arrows do, just that the property belongs to the entity.


Dog
Breed: Labrador
Name: Rolo