Big Data For Engineers

Philip J. Hartout, Gian S. Hiltbrunner

June 2020

Contents

1 Introduction

1.1 Are you capable of sketching the history of databases (ancient and modern) to a colleague in a few minutes?

Early forms of databases include natural forms of information storage such as speaking, singing and writing. These forms became more complex in time when people performed simple accounting tasks using stone inscriptions and later on when books became widespread with the invention of the book press. The computer then finally enabled us to store and manage data on much larger scales.

In the 1960s file systems were introduced allowing simple forms of data managment, in the 1970s relational table databases such as SQL were introduced allowing systematic queries. In the 2000s other databases became more popular (NoSQL), these include key-value stores, triple stores, document stores and column stores.

1.2 Do you know who Edgar Codd is?

Edgar Codd was an English computer scientist who invented the relational model for database management, the theoretical basis for relational databases.

1.3 Can you explain what Data Independence is, and why it is important?

The actual data and application programs acting on the data remain unimpaired even if the storage representation or access methods are changed. This is important when we want to store small scale data on a single hard drive and store large scale data on cluster infrastructure but essentially want to access the data in the same manner.

1.4 Do you know the rough conceptual difference between data, information, and knowledge?

- Data is conceived of as symbols or signs, representing stimuli or signals lacing any interpretation.
- Information is inferred from data", in the process of answering interrogative questions (e.g., "who", "what", "where", "how many", "when"), thereby making the data useful for "decisions and/or action".
- Knowledge represents information combined with understanding, capability and experience.

1.5 Can you cite the five fundamental shapes of data, and how structured data can be characterized?

Text, Tables, Trees, Graphs and Cubes

- Data conforms to a data model and has easily identifiable structure

- Data is well organised so, Definition, Format and Meaning of data is explicitly known
- Data resides in fixed fields within a record or file
- Similar entities are grouped together to form relations or classes
- Entities in the same group have same attributes
- Easy to access and query, So data can be easily used by other programs
- Data elements are addressable, so efficient to analyse and process

1.6 Can you briefly explain what a data model is?

Underlying Questions

- How does the data look like?
- What can be done with the data?

A data model is an abstract model that organizes elements of data and standardizes how they relate to one another and to the properties of real-world entities. For instance, a data model may specify that the data element representing a car be composed of a number of other elements which, in turn, represent the color and size of the car and define its owner.

1.7 Do you know the 8 standard prefixes of the International System of Units (when the exponent in base 10 is a positive multiple of 3)?

Prefix	Digits
kilo (k)	1,000 (3 zeros)
Mega (M)	1,000,000 (6 zeros)
Giga (G)	1,000,000,000 (9 zeros)
Tera (T)	1,000,000,000,000 (12 zeros)
Peta (P)	1,000,000,000,000,000 (15 zeros)
Exa (E)	1,000,000,000,000,000,000 (18 zeros)
Zetta (Z)	1,000,000,000,000,000,000,000 (21 zeros)
Yotta (Y)	1,000,000,000,000,000,000,000,000 (24 zeros)

1.8 Can you list the main four technologies commonly referred to as NoSQL?

- Key-value stores
- Triple stores
- Column stores
- Document stores

1.9 Do you know the three Vs?

Big Data is defined by the three Vs.

- **Volume**: High amount of data.
- **Variety**: Different types and structures.
- **Velocity**: High rate of data throughput.

1.10 Can you briefly define capacity, throughput and latency? Do you know their typical units?

- **Capacity**: Volume of the stored data. (GB)
- **Throughput**: Speed of data transmission. (kbit/sec)
- **Latency**: Time until data is being received. (sec)

1.11 Can you explain why and how the evolution of capacity, throughput and latency over the last few decades has influenced the design of modern databases?

While the capacity has increased 200,000,000,000 fold, the throughput has increased only 10,000 fold and the latency only 8 fold.

1.12 Can you name a few big players in the industry that accumulate and analyze massive amounts of data?

Google, AWS (Amazon Web Services), Microsoft Azure and many more

1.13 Do you know the difference between bit and byte?

One byte is a collection of 8 bits where a bit represents the most basic form of information storage: 1 or 0.

1.14 Can you name a few concrete examples that illustrate the various orders of magnitude of amounts of data?

2 Lessons learnt

2.1 Can you explain why it is important to take into consideration whether a use case is read-intensive, or write-intensive, or in-between?

Write Intensive $\rightarrow$ **OLTP** (On-line Transaction Processing) is involved in the operation of a particular system. OLTP is characterized by a large number of short on-line transactions (INSERT, UPDATE, DELETE). The main emphasis for OLTP systems is put on very fast query processing, maintaining data integrity in multi-access environments and an effectiveness measured by number of transactions per second. In OLTP database there is detailed and current data, and schema used to store transactional databases is the entity model (usually 3NF). It involves queries accessing individual record like updating an email record in a database.

Read Intensive $\rightarrow$ **OLAP** (On-line Analytical Processing) deals with historical data or archival data. OLAP is characterized by a relatively low volume of transactions. Queries are often very complex and involve aggregations. For OLAP systems a response time is an effectiveness measure. OLAP applications are widely used for data mining applications. In an OLAP database there is aggregated, historical data, stored in multi-dimensional schemas. Queries often access large amounts of data.

2.2 Can you explain why normal forms are important?

Normal forms are important to reduce data redundancy and improve data integrity. This is achieved by making sure that the data follows a set of rules corresponding to each of the normal forms.

2.3 Can you describe the first normal form in simple terms?

The first normal form expects the data to have a primary key ("id"), no repeating groups of entries (Not having columns such as "ChildName1" and "ChildName2") and atomicity of entries (No column such as "Children" with value e.g. "Max, Pia").

2.4 Can you describe in simple terms (that is, without knowing the details of them) how higher normal forms (like Boyce-Codd) are related to joins?

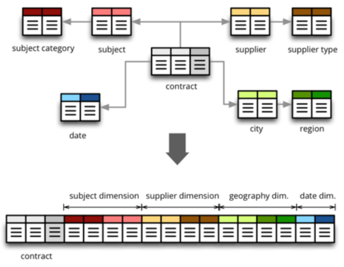
Higher order forms make sure that the data conforms to certain rules, thus when joining tables less conflicts occur.

2.5 Can you explain why it is common, for large amounts of data, to drop several levels of normal form, and denormalize data instead?

Denormalization is used to increase performance and scalability for large datasets. This leads to a increase in read performance while write performance suffers. This is due to the fact that relational databases need to perform many expensive JOIN operations when data is retrieved. These operations are prohibitively costly for very large datasets.

+ Retrieving data is faster since we do fewer joins
+ Queries to retrieve can be simpler(and therefore less likely to have bugs), since we need to look at fewer tables.
- Updates and inserts are more expensive.
- Denormalization can make update and insert operations more complex.
- Data may be inconsistent.
- Data redundancy necessitates more storage.

2.6 Can you give simple examples of denormalized data?



2.7 Can you explain what a declarative language is?

Declarative languages such as SQL describe what needs to happen but do not care about how exactly something is happening. Thus we would tell the the interpreter to fetch an entry using SELECT but not really care about how this entry is selected. (e.g. looping over the entries, etc.)

2.8 Can you explain what a functional language is?

Functional programming is a programming paradigm where programs are constructed by applying and composing functions. It is a declarative programming paradigm in which function definitions are trees of expressions that each return a value.

2.9 Can you explain why it matters to design query languages that are declarative and functional?

Declarative languages make sure that the we don't "reinvent the wheel" every time we create a query. Functional languages basically have similar benefits. They use functions to perform many operations such as map instead of looping over an array manually, thereby hiding away complexity and making code more readable.

2.10 Can you describe the major relational algebra operators: select, project, group and aggregate, join, Cartesian product, union, intersection, etc?

- **SELECT**: Operation that subsets the data set given a condition. Such as SELECT all entries from the table Person whose age is larger than 34. ($\sigma_{\text{Age} \geq 34}(\text{Person})$)
- **PROJECT**: Operation that picks a subset of all columns and drops duplicate entries. From table Person (Name, Age, Weight) project the columns Age and Weight to a new table while removing the Name column and excluding duplicate entries. ($\Pi_{\text{Age,Weight}}(\text{Person})$)
- **GROUP**: Operation that groups rows sharing the grouping property such that the groups can be aggregated in a subsequent operation. (e.g. SUM)
- **AGGREGATE**: Operation that summarizes multiple rows in a summarized form. Aggregate functions include: average, sum, median, standard deviation and many others.
- **JOIN**: Operation that unites two tables by the set of tuples that match. (e.g. key)
- **CARTESIAN PRODUCT**: Operation that creates a new table based on all possible combinations.
- **UNION**: Operation that combines tables from two SELECT operations.
- **INTERSECTION**: Operation that combines rows from tables that are present in both original data sets.

2.11 Do you know what each letter stands for in ACID and CAP?

ACID

- Atomicity: Atomicity guarantees that each transaction is treated as a single "unit", which either succeeds completely, or fails completely.
- Consistency: Consistency ensures that a transaction can only bring the database from one valid state to another, maintaining database invariants.
- Isolation: Transactions are often executed concurrently (e.g., multiple transactions reading and writing to a table at the same time). Isolation ensures that concurrent execution of transactions leaves the database in the same state that would have been obtained if the transactions were executed sequentially.
- Durability: Durability guarantees that once a transaction has been committed, it will remain committed even in the case of a system failure (e.g., power outage or crash).

CAP

- Consistency: Every copy of a data set has to be modified upon completion of a transaction. Note that this is not the same as consistency in the ACID model.
- Availability: Every request receives a (non-error) response, without the guarantee that it contains the most recent write
- Partition tolerance: The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes

2.12 Can you explain why, for large amounts of data, CAP becomes relevant over ACID?

ACID addresses an individual node's data consistency
CAP addresses cluster-wide data consistency

2.13 Do you know the names of the basic components of the tabular shape at an abstract level (table, row, column, primary key) as well as the names of the most common corresponding counterparts in the NoSQL world?

SQL	**NoSQL**
database	database
table	collection
row	document
column	field
index	index
primary key	primary key

2.14 Do you know the basic SQL constructs: SELECT FROM WHERE, GROUP, HAVING, JOIN, ORDER BY, LIMIT, OFFSET, as well as nested queries?

Codecademy - SQL Commands

3 Object storage

3.1 Can you describe the limitations of traditional (local) file systems?

Local file systems are limited in capacity by the hardware capacity, i.e. they cannot contain petabytes. Local filesystems can also have local hierarchical and does not allow flexible manipulation of it.

3.2 Can you explain what object storage is?

Object storage (also known as object-based storage) is a computer data storage architecture that manages data as objects, as opposed to other storage architectures like file systems which manages data as a file hierarchy, and block storage which manages data as blocks within sectors and tracks. (Wikipedia)

It uses a flat, simple and global key value model and flexible metadata to enable retention of a massive amount of unstructured data.

3.3 Can you explain what the benefits of object storage are?

It enables the retention of massive, and diverse amounts of data on commodity hardware, with easy key-value pair access, and flexible metadata. Scaling out in terms of hardware with this approach allows also to have more performance for a lower price.

3.4 Can you contrast or relate object storage with block storage, a file system, and the key-value model?

See wiki entry again: object storage (also known as object-based storage) is a computer data storage architecture that manages data as objects, as opposed to other storage architectures like file systems which manages data as a file hierarchy, and block storage which manages data as blocks within sectors and tracks. (Wikipedia)

3.5 What is the CAP theorem?

In theoretical computer science, the CAP theorem, also named Brewer's theorem after computer scientist Eric Brewer, states that it is impossible for a distributed data store to simultaneously provide more than two out of the following three guarantees:

- Consistency: Every read receives the most recent write or an error
- Availability: Every request receives a (non-error) response, without the guarantee that it contains the most recent write
- Partition tolerance: The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes

When a network partition failure happens should we decide to

- Cancel the operation and thus decrease the availability but ensure consistency
- Proceed with the operation and thus provide availability but risk inconsistency

The CAP theorem implies that in the presence of a network partition, one has to choose between consistency and availability. Note that consistency as defined in the CAP theorem is quite different from the consistency guaranteed in ACID database transactions.

3.6 Can you explain the three different ways, on the physical level, to deal with more data (scale up, scale out, write better code)?

- Scale up - get bigger machines, price increases exponentially
- Scale out - get more commodity machines, price increases linearly
- Write better code - avoids excessive processing overhead and storage requirements, price can be substantially lower.

3.7 Can you explain why scaling out is less expensive than scaling up?

Because commodity hardware manufacturing is streamlined its price increases linearly, whereas harder to make and hence more expensive machines which are obtained by scaling up increase the cost exponentially.

3.8 Can you explain what aspects of the design of object storage enable scaling out and why?

The architecture is simple and can easily be fragmented to be stored on commodity hardware.

3.9 Do you know what a data centre is made of (racks, server nodes, storage nodes, etc.)?

Racks: vertical stack of server nodes. Storage nodes are server nodes or VMs specialized for storage.

3.10 Do you know the difference between storage, memory, CPU and network and how the three are paramount in a cluster?

Yes.

3.11 Do you know rough, typical numbers (per-node storage capacity, memory, number of cores, etc.)?

Typical data center: 1,000-100,000 machines Cores: 1-100 cores per machine Storage: 1-20 TB local storage per server RAM: 16-6TB. Network bandwidth:1-100 Gb/s

3.12 Do you know how storage and memory technologies (HDD, SDD and RAM) compare in terms of capacity and throughput?

Capacity HDD>SDD>RAM Throughput HDD<SDD<RAM

3.13 Can you sketch the (key-value) data model behind object storage? Can you spot instances of the key-value model in several other technologies seen in the lecture?

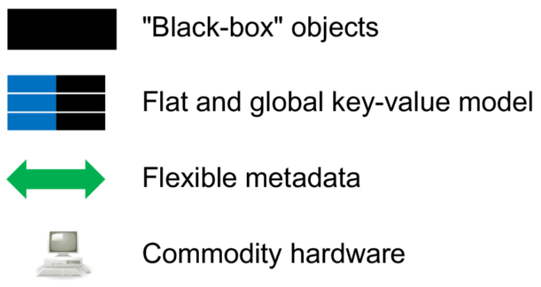


Figure 1: Key value schema and other features of object storage

Other instances: JSON, Azure cosmos DB key value paradigm, ...

	S3	Azure
Object ID	Bucket + Object	Account + container + blob
Object API	Blackbox	Block Append Page
Limit	5TB	4.78TB (block), 195TB (append), 8TB (page)

Table 1: Overall comparison Azure vs S3

SLA	Outage
99%	4 days/year
99.9%	9 hours/year
99.99%	53 minutes/year
99.999%	6 minutes/year
99.9999%	32 seconds/year
99.99999%	4 seconds/year

Table 2: SLA

3.14 Do you know the difference between data and metadata?

Metadata is data that provides information about other data. In other words, it is data about data.

3.15 Do you know the order of magnitude that can be achieved in terms of number of objects, and object size?

Object size 5TB, 100 buckets per user by default.

3.16 Can you name a few big players for cloud-based object storage (vendors, consumers)?

Google, Microsoft, Amazon.

3.17 Can you describe the features of S3 and Azure Blob Storage on a high level? Do you know what a bucket and object are? What block blob storage, append blob storage and page blob storage are and how they work?

3.18 Can you describe what the most important SLA (Service Level Agreement) parameters mean (e.g., latency, availability, durability) as well as their typical range?

Durability is loss of 1 in 10^{11} objects a year.

Response time is <10ms in 99.9%.

3.19 Can you describe a typical use case for object storage?

- Disaster recovery — Backup and Archiving.
- Static web site hosting and static content distribution.
- Document Store and file sharing.
- Big Data analytics.

3.20 Can you explain what a REST API is, what resources and methods are?

Representational state transfer (REST) is a software architectural style that defines a set of constraints to be used for creating Web services. Web services that conform to the REST architectural style, called RESTful Web services, provide interoperability between computer systems on the Internet. RESTful Web services allow the requesting systems to access and manipulate textual representations of Web resources by using a uniform and predefined set of stateless operations. Other kinds of Web services, such as SOAP Web services, expose their own arbitrary sets of operations.

REST APIs use Uniform Resource Identifiers (URIs) to address resources. REST API designers should create URIs that convey a REST API's resource model to its potential client developers. When resources are named well, an API is intuitive and easy to use.

Methods include:

- HTTP GET
- HTTP POST
- HTTP PUT
- HTTP DELETE
- HTTP PATCH

Difference between PUT and PATCH: HTTP PATCH requests are to make partial update on a resource. If you see PUT requests also modify a resource entity, so to make more clear – PATCH method is the correct choice for partially updating an existing resource, and PUT should only be used if you're replacing a resource in its entirety.

3.21 Can you explain how an object storage API looks like (get and put)?

You can address any combination of bucket and object name like so in S3:

```
http://bucket.s3.amazonaws.com/object-name
```

3.22 Are you able to upload and download objects from S3 or Azure Blob Storage?

Yes. This is also what one does when hosting a static website.

4 Distributed file systems

4.1 Can you explain the difference between block storage and object storage?

Block storage is organised as blocks, which emulate the type of behaviour seen in traditional disk or tape storage. Blocks are identified by an arbitrary and assigned identifier by which they may be stored and retrieved, but this has no obvious meaning in terms of files or documents. A filesystem must be applied on top of the block-level storage to map 'files' onto a sequence of blocks. (Replication or backups)
Object store or 'bucket store', such as Amazon S3 (Simple Storage Service) operate at a higher level of abstraction and are able to work with entities such as files, documents, images, videos or database records. Each object typically includes the data itself, a variable amount of metadata, and a globally unique identifier.

4.2 Can you explain the difference between the (logical) key-value model and a file system?

The key value model is typically built on top of an object storage system while a file system works on block storage.

4.3 Can you contrast block storage and object storage in terms of maximum number of objects/files?

- **Key-Value Model**: Object Storage: Billions of sub-terabyte files. (Many small)
- **File System**: Block Storage: Millions of sub-petabyte files. (Few big)

4.4 Can you contrast block storage and object storage in terms of the maximum size of objects/files?

- **Key-Value Model**: A terabyte
- **File System**: A petabyte

4.5 Do you know the order of magnitude of a block size for a local file system and for a distributed file system? Can you explain the rationale behind them with respect to latency and throughput?

- **Local filesystem**: 4 kB
- **Distributed file system**: 64 MB - 128 MB

On a local system we have very high trougput and low latency, but on a cluster system the machines are connected over a network infrastructure which slows down transfer speeds. In order to make transfers more efficient larger blocks are used.

4.6 Do you know where HDFS shines and why?

- Works with varied data such as text, XML, CSV and many more.
- Works on commodity hardware - Cheap
- High performance due to parallelization
- Fault-tolerant
- Scalable

4.7 Do you know that HDFS files are updated by appending atomically and why?

HDFS will always append to the last block and only one singe write operation is allowed at one time. Thus guaranteeing that there are no issues with concurrency.

4.8 Do you know how HDFS performs in terms of throughput and latency?

HDFS is optimized to access batches of data set quicker (high throughput), rather then particular records in that data set (low latency).

4.9 Do you know the main benefits of HDFS (commodity hardware, etc.)?

See "Do you know where HDFS shines and why?".

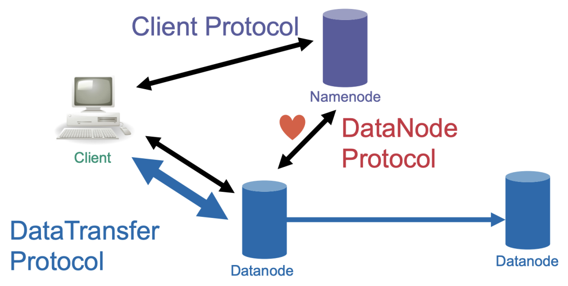
4.10 Can you contrast master-slave architectures to peer-to-peer architectures?

n the master-slave model, one node is in charge (master). When there's no single node with a special role in taking charge, you have a peer-to-peer distribution model.

4.11 Can you explain the HDFS architecture, what a namenode is and what a datanode is, how blocks are replicated?

HDFS works in a master-slave configuration. There is a master called the "Namenode" and multiple slaves called "Datanodes".

- **Namenode**: Manages the file namespace, acess control file to block mapping and keeps track of the block locations.
- **Datanode**: System that stores blocks on the local disk.
- **Replication**: Blocks are replicated on multiple Datanodes for fault tolerance.



4.12 Can you sketch how the various components communicate with each other (client, namenode, datanode)?

- **Client Protocol**: Manages metadata operations, DataNode locations and block IDs.
- **DataNode Protocol**: Block operations such as replication managment and registration of the data node, heartbeat pings, block reports and block recieved confirmations. The DataNode always initiates the connection.
- **Data Transfer Protocol**: Data blocks are transferred from a DataNode to a client or back. The datanode then initiates replication pipelining.

Add precise data read and write protocol?

4.13 Can you point to the single points of failure of HDFS and explain how they can be addressed?

The NameNode is a single point of failure.
We resolve this using:

- **Persistance**: We store the inital namespace file and append an edit log with the changes that have been made.
- **Backup**: We back this file up.

4.14 Can you explain how the namenode stores the file system namespace, in memory and on disk? In particular, can you explain how the namespace file and the edit log work together at startup time and how they get modified once the system is up and running?

4.15 Can you explain what a standby namenode is, what it is and what it is not?

4.16 Are you able to use the HDFS shell commands (creating directories, reading, uploading and downloading files, etc.)?

5 Syntax

5.1 Can you describe why syntax is relevant to data management and to Big Data, for all data shapes?

Syntax is crucial for efficient data management, and should be adapted to the data type is represents, e.g. if the data is in first normal form (i.e. there is atomic integrity) then csv is good, but if there is nestedness you will probably have data duplication, in this case JSON is more appropriate and avoids redundancy.

5.2 What are the system demands for normalized vs. denormalized data?

Normalized data is write intensive, e.g. if there is nestedness. Denormalized data is read intensive.

5.3 What do normalized vs. denormalized data avoid?

Normalized: avoid update anomalies Denormalized: avoids joins.

5.4 Can you give examples of syntax for trees? For tables? (The two data shapes extensively covered in the lecture)

CSV for tables and JSON for trees.

5.5 Can you explain what well-formedness is with respect to syntax?

well-formedness in the case of JSON for instance is when all the rules of the JSON syntax are respected by a JSON string.

5.6 Can you list the XML basic building blocks (we covered document, element, attribute, text) and do you know what they look like?

They look like this:

```xml
<element attribute="value">
Some text
</element>
```

Listing 1: XML building blocks

5.7 Can you tell whether a given XML document is well-formed, with a software (oXygen) as well as with your own eyes?

Well formed json string

```
{
  "tissue_directories": [
        "gtex/brain/brain_cerebellar_hemisphere.tpm.csv"
  ],
  "tissue_names": [
        "Brain Cerebellar Hemisphere"
  ],
  "sample_strategy": [
        "mean",
        "mean"
  ],
  "expression_selection": [
        [
              "smaller",
              100
        ],
        [
              "smaller",
              100
        ]
  ]
}
```

Listing 2: JSON well formed

Another example of not well-formed JSON.

5.8 Do you know the five fundamental pre-defined entities (used to escape characters: & < > ' ") and their syntax?

See Table 3.

5.9 Do you know when some characters must be escaped (e.g., < in text)?

Characters must be escaped in text. In JSON, done with "\". In XML, follow the conve ntions shown in Table 3.

```
"tissue_directories": [
        "gtex/brain/brain_cerebellar_hemisphere.tpm.csv"
],
"tissue_names" [
        "Brain Cerebellar Hemisphere"
],
sample_strategy: [
        "mean"
        "mean"
],
"expression_selection": [
        [
                "smaller",
                100
        ],
        [
                smaller,
                100
        ]
]
```

Listing 3: JSON not well formed

5.10 Can you list the JSON basic building blocks (object, array, string, number, boolean,null) and do you know what they look like?

Shown in Listing 2.

5.11 Can you tell whether a given JSON document is well-formed, also with your own eyes?

Try with Listings 2 and 3.

Name	Character	Unicode code point (decimal)	Standard	Name
quot	"	U+0022 (34)	XML 1.0	quotation mark
amp	&	U+0026 (38)	XML 1.0	ampersand
apos	'	U+0027 (39)	XML 1.0	apostrophe
lt	<	U+003C (60)	XML 1.0	less-than sign
gt	>	U+003E (62)	XML 1.0	greater-than sign

Table 3: Predefined entities.

5.12 What is the difference between validation and well-formedness

validation comes after well-formedness, i.e. a well formed string is validated against a schema, which is a well-formed JSON string. Example of a schema is shown in:

```
 {"type": "object",
"properties": {
    "tissue_directories": {
        "type": "array",
        "items": {
            "type": "string"
        }
    },
    "tissue_names": {
        "type": "array",
        "items": {
            "type": "string"
        }
    },
    "sample_strategy": {
        "type": "array",
        "items": {
            "oneOf": [
                {
                    "type": "string",
                    "enum": ["mean", "max", "min", "median"
                    ]
                },
                {
                    "type": "number", "minimum": 0, "maximum": 1
                }
            ]
        }
    },
    "expression_selection": {
        "type": "array",
        "items": {
            "type": "array",
            "items": [
                {
                    "oneOf": [
                        {
                            "type": "string",
                            "enum": ["greater", "smaller"
                            ]
                        },
                        {
                            "type": "number", "minimum": 0
                        }
                    ]
                },
                {
                    "type": "number","minimum": 0
                }
                ]
              }
            }
          }
```

s

Listing 4: Example of schema (done to be compatible for `https://python-jsonschema.readthedocs.io/en/stable/`)

6 Wide column stores

6.1 Can you explain the limitations of the traditional relational model?

6.2 Can you explain the differences and similarities between a wide column store and the traditional relational model?

6.3 Can you explain why wide column stores are called wide column stores, in particular on the storage level?

6.4 Do you know and can you contrast the two ways data can be distributed(partitioning, replicating)?

6.5 Can you explain the data model behind wide column stores, in particular, rows,columns, column families and cells?

6.6 Can you explain the motivation behind data getting denormalized into more columnfamilies?

6.7 Can you explain what aspects of an instance (table) in a column store must betypically known in advance, and which can be changed on the fly?

6.8 Can you name a few big players, in particular the one behind the initial founding paper?

6.9 Can you explain why HBase is based on HDFS, yet low-latency?

6.10 Can you explain what regions are for wide column stores?

6.11 Do you know how to identify a region based on the content of the wide column store?

6.12 Do you know the four basic kinds of (low-level) queries in HBase?

6.13 Can you describe the physical architecture of a wide column store like HBase, and compare it with that of a distributed file system like HDFS?

6.14 Do you know the physical layers of HBase (table, region, Store, Memstore, HFile,HBase block, Key Value)?

6.15 Do you know the difference between an HDFS block and an HBase block as well astheir typical sizes?

6.16 Can you explain how new cells are written to HBase via the cell store?

6.17 Can you explain how cells are read from HBase via both the stored HFiles and the Memstore?

6.18 Can you explain what compaction and flushing is?

6.19 Can you sketch the contents of an HFile and know what it corresponds to in anHBase table?

6.20 Can you explain why it is crucial not to have too long column family names?

6.21 Can you populate data into, and execute queries on top of HBase?

7 Data models

7.1 Can you explain the difference between syntax and a data model?

The data model is the manifestation of the data from a logical standpoint, whereas syntax is the manifestation of the data from a physical standpoint. E.g. a JSON string could potentially be modelled in approximately the same way in XML, only the syntax will obviously differ.

One such difference is that the labels of nodes are on edges for JSON, whereas they are on te nodes themselves in XML.

7.2 Can you explain why and how trees can be used to model data that is de-normalized(i.e., not in first normal form)?

Why? Trees can represent nested structure of data, thereby avoiding duplicates. How? JSON and XML contain data types which can be recursively defined to enable the representation of nested structure. a valid JSON string can be the value of a key in another JSON string.

7.3 Can you relate syntaxes to data models (e.g., CSV to tables, XML/JSON to trees, ...)?

Answer is in the question.

7.4 Can you explain why trees modeling XML (infoset) have labels on the nodes, while trees modeling JSON have labels on edges?

Because they are element names and keys, respectively.

7.5 Can you name a few data models for XML?

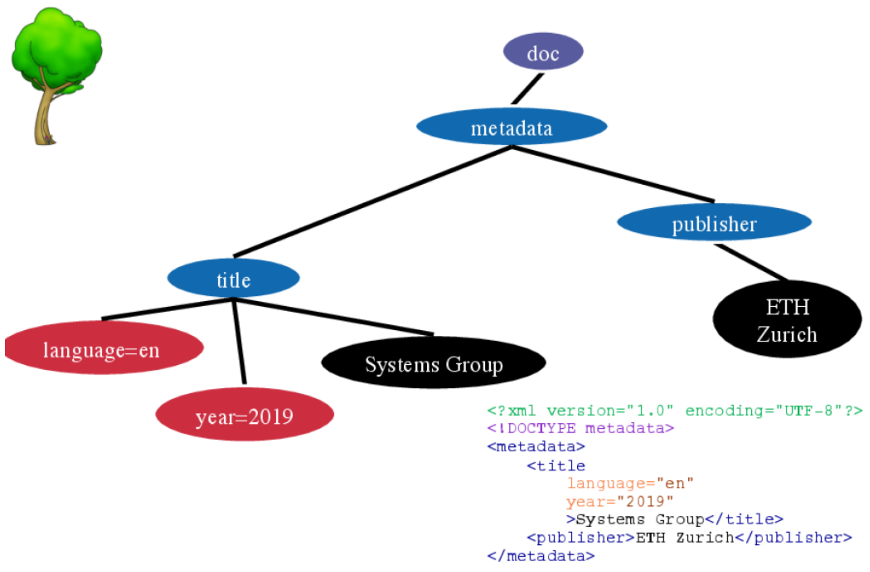
(Infoset, PSVI, XDM)

7.6 Are you able, given an XML document, to sketch a tree representing that data (with a document node, elements, attributes, text, ...)?

See Figure 2.

7.7 Do you know the different between an atomic type/value, and a structured type/value (regardless of the exact data model)?

Atomic: cannot be divided into other atomic types, structured type can.



```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE metadata>
<metadata>
    <title
        language="en"
        year="2019"
        >Systems Group</title>
    <publisher>ETH Zurich</publisher>
</metadata>
```

Figure 2: Tree from document.

7.8 Can you name a few atomic types found across a broad number of data models?

Strings, numbers, booleans, dates and times, tieme intervals, binaries, null.

7.9 Do you know the difference between the lexical space and the value space of anatomic type?

Value space is the range of values for a given type, while lexical space is the range of representations.

7.10 Can you tell the difference between structured types based on lists (e.g., JSON array) and maps (e.g., JSON object)?

Maps: key value model, whereas lists don't have such a model. Difference is also in syntax: [] vs. {} for arrays and objects, respectively.

7.11 Can you give examples of type cardinalities, and their associated symbols?

See table 4.

How many?	Common sign	Common adjective
One		required
Zero or more	/	repeated
Zero or one	?	optional
One or more	+	

Table 4: Cardinality

7.12 Can you explain the difference between well-formedness and validity? Do you know what extra information you need to assess validity?

See section 5.12.

7.13 Are you able to design simple XML Schemas to validate XML? Can you restrict simpletypes to allow specific values or value fulfilling specific criteria (length, pattern...)? Can you explain how to declare elements with simple types? Can you explain how to declare elements with complex types and various content models (empty, simplecontent, complex content, mixed content)?

Example XML schema shown in listing 5.

7.14 Can you tell, given an XML Schema, whether an XML document is valid against it,with software (oXygen) but also with your own eyes?

Go through a couple of examples.

7.15 Are you able to design simple JSON Schemas to validate JSON?

See Listing 4.

7.16 Can you name (without details) further data modeling technologies for tree-likedata?

Avro, protocol buffers, ...

7.17 Do you understand that a table in first normal form can be seen as a homogeneous collection of flat trees (one per row and all with the same attributes)?

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="shiporder">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="orderperson" type="xs:string"/>
        <xs:element name="shipto">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="name" type="xs:string"/>
              <xs:element name="address" type="xs:string"/>
              <xs:element name="city" type="xs:string"/>
              <xs:element name="country" type="xs:string"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element maxOccurs="unbounded" name="item">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="title" type="xs:string"/>
              <xs:element minOccurs="0" name="note" type="xs:string"/>
              <xs:element name="quantity" type="xs:positiveInteger"/>
              <xs:element name="price" type="xs:decimal"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="orderid" type="xs:string" use="required"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Listing 5: XML Schema example from here

8 Massive parallel processing (MapReduce)

8.1 Can you explain the patterns that appear in large-scale data processing: map, shuffle?

Map: each worker node applies the map function to the local data, and writes the output to a temporary storage. A master node ensures that only one copy of the redundant input data is processed.

Shuffle: worker nodes redistribute data based on the output keys (produced by the map function), such that all data belonging to one key is located on the same worker node.

Reduce: worker nodes now process each group of output data, per key, in parallel.

8.2 Can you explain how the mastery of these patterns can bring signficant improvements in performance?

8.3 Can you explain the MapReduce model, also in terms of the patterns depicted above (map, shuffle, reduce)?

8.4 Can you describe the physical architecture of MapReduce (map tasks, reduce tasks as well as data flow)? Can you explain version 1 of the architecture (JobTracker, TaskTrackers) and its limits?

8.5 Can you outline what a map function looks like, and what a reduce function looks like?

8.6 Do you know the main input and output types of MapReduce?

8.7 Can you explain how combining improve MapReduce's performance?

8.8 Can you state the assumptions behind reusing the reduce function as a combinefunction?

8.9 Do you understand, in some simple cases, how to design a combine function thatwould make a MapReduce job faster, even if the combine function is not the exact same as the reduce function? (Example: computing an average, which requires keeping track of the weights in the output of the combine function).

8.10 Can you explain why MapReduce shines on top of a distributed file system: "Bring the query to the data"?

8.11 Can you explain why MapReduce especially makes sense when the bottleneck is the speed of reading and writing data from the disk (as opposed to other bottlenecks such as storage capacity or CPU usage)?

8.12 Can you explain how MapReduce splits differ from HDFS blocks, what impedance mismatches arise and how they are addressed?

8.13 Do you know what the Java API of MapReduce looks like on a high level (Version 2 of the API that we covered, not to be confused with Version 2 of MapReduce running on YARN)?

9 Resource Management

9.1 Can you explain how YARN works, and how it can be used to improve MapReduce, and to support other technologies like Spark?

The responsibilities of the jobtracker in mapreduce are manifold:

1. Resource management
2. Scheduling
3. Monitoring
4. Job lifecycle
5. Fault-tolerance

YARN solves multiple issues that arise with vanilla mapreduce and DAG processing:

1. Scalability
2. The jobtracker bottleneck
3. the fact that the jobtracker has to take care of both scheduling and monitoring.
4. The fixed size and static allocation of jobs.
5. Fungibility, i.e. the ability of a good or asset to be interchanged with other individual goods or assets of the same type.

YARN divides scheduling from monitoring.

9.2 Can you describe the YARN components?

- Resource Manager: does not monitor tasks or restart upon failure.
- Node Manager
- Container

9.3 Do you know what a ResourceManager does?

Pure scheduling

9.4 Do you know what a NodeManager does?

Conceptually, the NodeManager is more of a generic and efficient version of TaskTracker (of Hadoop1 architecture) which is more flexible than TaskTracker. In contrast to fixed number of slots for map and reduce tasks in MRV1, the NodeManager of MRV2 has a number of dynamically created resource containers. There is no hard code split available into Map and Reduce slots as in MRV1. The container refers to a collection of resources such as memory, CPU, disk and network IO. The number of containers on a node is the product of configuration parameter and the total amount of node resources. Node manager is the slave daemon of Yarn.

9.5 Do you know what and where a Container is?

As previously described, ResourceManager (RM) is the master that arbitrates all the available cluster resources and thus helps manage the distributed applications running on the YARN system. It works together with the per-node NodeManagers (NMs) and the per-application ApplicationMasters (AMs).

9.6 Do you know what an ApplicationMaster is and does?

ApplicationMasters are responsible for negotiating resources with the ResourceManager and for working with the NodeManagers to start the containers.

9.7 Can you list the main resources that are managed in a cluster? .

Disk storage,memory, CPU, network I/O: all are dynamically allocated for each application.

9.8 Can you explain, in simpler words, what the added value of YARN is? Can you explainwhat it is an improvement over the first version of MapReduce, which was taking care of resource management on its own, and had issues with this?

Essentially, YARN allows for a more flexible and dynamic allocation of tasks on a cluster, which also allows to prioritize certain tasks and users while wasting a minimum of space.

9.9 From HADOOP website

The fundamental idea of YARN is to split up the functionalities of resource management and job scheduling/monitoring into separate daemons. The idea is to have a global ResourceManager (RM) and per-application ApplicationMaster (AM). An application is either a single job or a DAG of jobs.

The ResourceManager and the NodeManager form the data-computation framework. The ResourceManager is the ultimate authority that arbitrates resources among all the applications in the system. The NodeManager is the per-machine framework agent who is responsible for containers, monitoring their resource usage (cpu, memory, disk, network) and reporting the same to the ResourceManager/Scheduler.

The per-application ApplicationMaster is, in effect, a framework specific library and is tasked with negotiating resources from the ResourceManager and working with the NodeManager(s) to execute and monitor the tasks.

The ResourceManager has two main components: Scheduler and ApplicationsManager.

The Scheduler is responsible for allocating resources to the various running applications subject to familiar constraints of capacities, queues etc. The Scheduler is pure scheduler in the sense that it performs no monitoring or tracking of status for the application. Also, it offers no guarantees about restarting failed tasks either due to application failure or hardware failures. The Scheduler performs its scheduling function based the resource requirements of the applications;

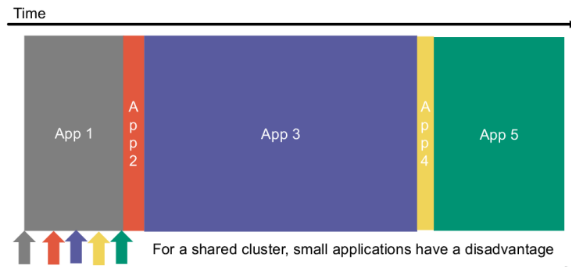

Time
App 1
App 2
App 3
App 4
App 5
For a shared cluster, small applications have a disadvantage

Figure 3: FIFO: least efficient, whole cluster devoted to task.

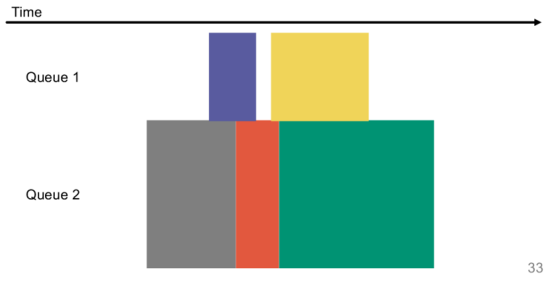


Figure 4: Capacity scheduling increases the amount of queues but does not allow optimal ressource allocation.

it does so based on the abstract notion of a resource Container which incorporates elements such as memory, cpu, disk, network etc.

The Scheduler has a pluggable policy which is responsible for partitioning the cluster resources among the various queues, applications etc. The current schedulers such as the CapacityScheduler and the FairScheduler would be some examples of plug-ins.

The ApplicationsManager is responsible for accepting job-submissions, negotiating the first container for executing the application specific ApplicationMaster and provides the service for restarting the ApplicationMaster container on failure. The per-application ApplicationMaster has the responsibility of negotiating appropriate resource containers from the Scheduler, tracking their status and monitoring for progress.

MapReduce in hadoop-2.x maintains API compatibility with previous stable release (hadoop-1.x). This means that all MapReduce jobs should still run unchanged on top of YARN with just a recompile.

9.10 What are the pitfalls and advantages of various types of scheduling?

Figure 5: FAIR scheduling, most optimal scheduling of ressources.

10 Massive parallel processing (Spark)

10.1 Can you explain how Spark is more powerful than MapReduce on a data model level?

10.2 Can you explain what a Resilient Distributed Dataset (RDD) is?

10.3 Do you know the difference between an action and a transformation?

10.4 Do you know the main actions and transformations available in Spark?

10.5 Are you able to classify transformations and actions in various buckets (those thatwork on any value, those that work on key-value pairs, unary binary transformations, ...)?

10.6 Can you describe how transformations run physically (tasks, stages...)?

10.7 Can you explain when and how a series of transformations can be optimized by keeping the same set of machines with no network communication between the transformations?

10.8 Can you explain what a stage is? Can you relate it to transformations? To tasks? To Jobs?

10.9 Can you tell why and when shuffling is needed? In other words, can say whether atransformation has a narrow dependency or a wide dependency?

10.10 Can you easily draw a directly acyclic graph for a Spark job, and mark the stages?

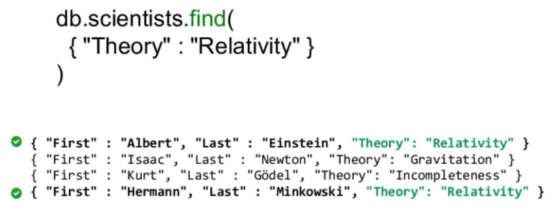
10.11 Do you understand why keeping an RDD persisted can be useful and improve performance?

10.12 Can you explain how controlling the way data is partitioned can make executionfaster because we can influence stages?

10.13 Can you explain what a DataFrame is and what its benefits are?

10.14 Can you describe the limitations of DataFrames?

10.15 Can you write queries on DataFrames, both in Spark SQL or using DataFrames transformations?



```
db.scientists.find(
  { "Theory" : "Relativity" }
)
```

```
✓ { "First" : "Albert", "Last" : "Einstein", "Theory": "Relativity" }
  { "First" : "Isaac", "Last" : "Newton", "Theory": "Gravitation" }
  { "First" : "Kurt", "Last" : "Gödel", "Theory": "Incompleteness" }
✓ { "First" : "Hermann", "Last" : "Minkowski", "Theory": "Relativity" }
```

Figure 6: Find query result.

11 Document stores

11.1 Do you understand how collections in documents store generalize the concept of a relational table?

Yes, essentially document stores generalize the idea of a relational table by encompassing trees. A document store is essentially a collection of trees, and a relational table is essentially a special case of document where the tree structure is flat.

11.2 Can you explain what documents store can do that relational databases cannot (e.g., heterogeneous collections, schema-less collections, data denormalized into trees...)?

It can:

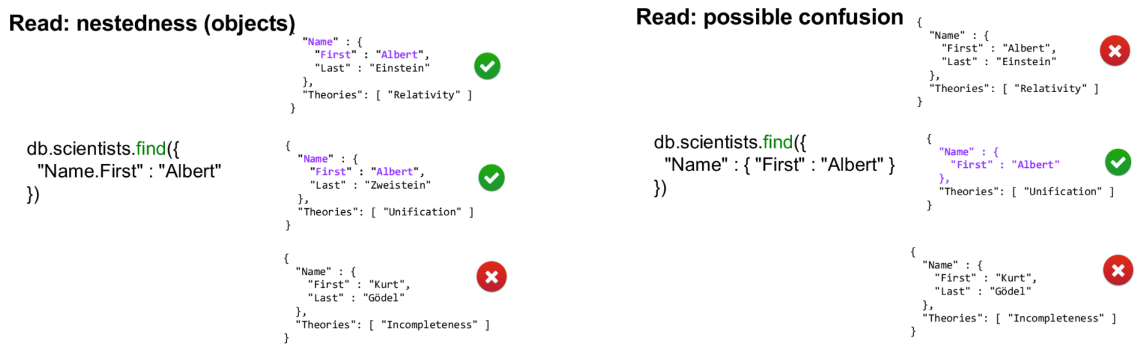
- It can collect and store heteogeneous collections of unstructured or semi structured data.
- It also does not require normalized data
- It does not require a schema to collect the data.

11.3 Do you know how to issue MongoDB queries on a low level? Do you know the parameters of the find() function? (query, then projection, then sorting)?

Couple of examples:

- The following example returns all fields from all documents in the inventory collection where the status equals "A": `db.inventory.find( { status: "A" } )`
- Sorting: `db.orders.find().sort( { amount: -1 } )`

Further examples shown in Figure 6. Note subtleties in 7.



```
db.scientists.find({
  "Name.First" : "Albert"
})
```

```
{
  "Name" : {
    "First" : "Albert",
    "Last" : "Einstein"
  },
  "Theories": [ "Relativity" ]
}
```

✓

```
{
  "Name" : {
    "First" : "Albert",
    "Last" : "Zweistein"
  },
  "Theories": [ "Unification" ]
}
```

✓

```
{
  "Name" : {
    "First" : "Kurt",
    "Last" : "Gödel"
  },
  "Theories": [ "Incompleteness" ]
}
```

✗

```
db.scientists.find({
  "Name" : { "First" : "Albert" }
})
```

```
{
  "Name" : {
    "First" : "Albert",
    "Last" : "Einstein"
  },
  "Theories": [ "Relativity" ]
}
```

✗

```
{
  "Name" : {
    "First" : "Albert"
  },
  "Theories": [ "Unification" ]
}
```

✓

```
{
  "Name" : {
    "First" : "Kurt",
    "Last" : "Gödel"
  },
  "Theories": [ "Incompleteness" ]
}
```

✗

Figure 7: Nestedness

11.4 What is the one operations document stores cannot do that RDBMS can and vice versa?

Document stores cannot perform join operations, RDBMS don't validate the data once populated.

11.5 Can you explain how, in a document store, the documents can be sharded and replicated?

From MongoDB on sharding: Sharding is a method for distributing data across multiple machines. MongoDB uses sharding to support deployments with very large data sets and high throughput operations.

Database systems with large data sets or high throughput applications can challenge the capacity of a single server. For example, high query rates can exhaust the CPU capacity of the server. Working set sizes larger than the system's RAM stress the I/O capacity of disk drives.

There are two methods for addressing system growth: vertical and horizontal scaling.

Vertical Scaling involves increasing the capacity of a single server, such as using a more powerful CPU, adding more RAM, or increasing the amount of storage space. Limitations in available technology may restrict a single machine from being sufficiently powerful for a given workload. Additionally, Cloud-based providers have hard ceilings based on available hardware configurations. As a result, there is a practical maximum for vertical scaling.

Horizontal Scaling involves dividing the system dataset and load over multiple servers, adding additional servers to increase capacity as required. While the overall speed or capacity of a single machine may not be high, each machine handles a subset of the overall workload, potentially providing better efficiency than a single high-speed high-capacity server. Expanding the capacity of the deployment only requires adding additional servers as needed, which can be a lower overall cost than high-end hardware for a single machine. The trade off is increased complexity in infrastructure and maintenance for the deployment.

For replication: often a primary copy of the data is kept, and updated first. Then, secondary members of a replica sets are updated. Each shard contains a replica sets, composed of primary vs. secondary copies of the data.

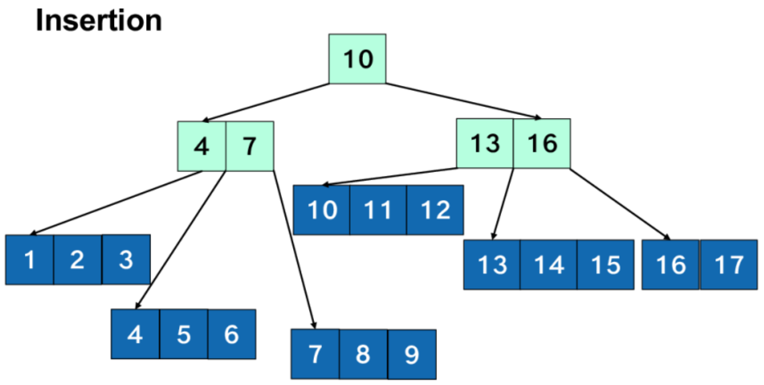


Figure 8: Tree as index.

11.6 Do you understand how indices can make queries faster, like in relational databases?

Hash indices are a fast way of performing selections of data in a document store. It's essentially a table containing the value and the row id of the document.

11.7 What are some of the limitations of hash indices?

- No support for range queries
- Hash function not perfect in real life
- Space requirements for collision avoidance. (pigeonhole principle).

11.8 Do you know what kinds of indices there are (hash, B-trees)?

Hash tables are tables as described two questions earlier. B trees are trees are balanced trees which result in block access of the data. Some properties of these trees:

- All leaves are at the same depth.
- All non-leaf nodes have between 3 and 5 children.
- The values are only at the leaves.

Example shown in Figure

11.9 Are you capable of telling if an index is useful to a given query, for simple settings(for example, an index on a single field and a query that selects on that field)?

11.10 Do you know that a compound index (e.g., on keys a and b) can also be used as anindex on any prefix of the compound key (e.g., on key a only) "for free"?

12 Querying trees

12.1 Can you explain why a language such as JSONiq provides, in the context of document stores, exactly the same functionality as SQL in a relational database?

12.2 Can you name the first-class citizen of the JDM (sequence of items)?

12.3 Can you name various kinds of items in the JDM?

12.4 Can you name a few query languages in the XML/JSON ecosystem?

12.5 Are you able, in JSONiq, to construct items (atomic items, elements, etc.)?

12.6 Are you able, in JSONiq, to perform logical operations? Do you understand what theEffective Boolean Value of a sequence and how it relates to logical operations?

12.7 Are you able, in JSONiq, to perform arithmetic operations (addition, etc.)? Do youunderstand the constraints on the input sequences of such operations? Can you explain the behavior of these operations on empty sequences? Can you explain what happens if one of the two operands is a node and not an atomic item?

12.8 Are you able, in JSONiq, to perform comparisons (lt, ge, etc.)? Do you understand the constraints on the input sequences of such operations? Can you explain the behavior of these operations on empty sequences? Can you explain what happens if one of the two operands is a node and not an atomic item?

12.9 Do you understand how general comparisons (<, >=, etc.) work on sequences with more than one item, and implicitly use an existential quantifier)?

12.10 Do you understand how FLWOR expressions work and describe what they return? (for clause, let clause, where clause, order by clause, etc.)

12.11 Are you able to use further expressions (if-then-else, switch, ...)?

12.12 Do you understand how to dynamically build JSON content with object and array constructors?

12.13 Do you understand that expressions can be combined at will, as any expression takesand returns sequences of items? Do you know how to use parentheses to make precedence clear, like you did in primary school with + and *?

12.14 Do you know the JSONiq type syntax (atomic types taken from XML Schema, syntaxfor object and array types, as well as cardinality symbols), and how to use typechecking (instance of, cast as, etc.)?

12.15 Given a collection of JSON objects (for example JSON Lines on HDFS), are you able touse Rumble to write JSONiq queries (FLWOR) that do projection? selection? (weleave grouping, join, ... aside in the context of this lecture, however you need to know that this can be done with JSONiq as well).