1. What is indexing and when should you use it?

Indexing creates additional data structures, called indexes, that contain a subset of the data from the collection based on a selected field. They are used to store pointers for values contained in records so that reading entire records and collections is not necessary for queries which contain the indexed field(s). It is ideal to index fields which are used by frequently performed queries, or for fields which are present in many queries. They become particularly useful in large datasets and operations which involve joins and aggregates of data, since they can save a large portion of read time in these situations. It does come with additional overhead with write or remove operations, since the indexes will need to be updated after these actions.

2. What is index cardinality and why is it important?

A number which represents the portion of the values being indexed which are unique. A high index indicates the values as being very unique, and a low index indicates more duplicate values. When aiming to create efficient indexes it is typically better to use fields with higher cardinality, and for those fields to be searched against earlier in the query, since this will lead to less total iterations of data reading being performed to obtain the desired dataset.

3. What are capped collections? And, when should you use them?

They are collections with a fixed/maximum size. The size is measured by either storage/file size, or the number of documents. Once the limit is reached, the older records are erased first. In broad terms, it is good to use in situations where more recent data is more useful, and chronology is a natural aspect of the data being recorded. This is common in ‘real-time’ data recording like logging, or data analysis. In a situation of this nature, collection sizes could quickly become unreasonably enormous, to the extent of becoming unusable even.

4. What are the differences between sparse and traditional indexes?

Sparse indexes are run on a field for which a value is not present in every record. The records which do not contain a value for the indexed field, or lack the field entirely, are not indexed; this tends to lead to more optimized storage. A traditional index is one in which a value is present for the indexed field in each record contained in a collection. Because the value is present in all records, every record is indexed; this tends to lead to more optimized queries.

Sources:

*Indexes*. MongoDB. (n.d.). <https://www.mongodb.com/docs/manual/indexes/>

Garcia-Molina et al., (2013, January 1) *Database Systems: The Complete Book*. Pearson

Winand, M. (2012, January 1) *SQL Performance Explained*. Winand