NOSQL and Query Optimization

NoSQL:

A **NoSQL** (originally referring to "non-SQL" or "non-relational") database provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases. Such databases have existed since the late 1960s, but the name "NoSQL" was only coined in the early 21st century, triggered by the needs of Web 2.0 companies. NoSQL databases are increasingly used in big data and real-time web applications. NoSQL systems are also sometimes called **Not only SQL** to emphasize that they may support SQL-like query languages or sit alongside SQL databases in polyglot-persistent architectures.

Motivations for this approach include simplicity of design, simpler "horizontal" scaling to clusters of machines (which is a problem for relational databases), finer control over availability, and limiting the object-relational impedance mismatch. The data structures used by NoSQL databases (e.g. key–value pair, wide column, graph, or document) are different from those used by default in relational databases, making some operations faster in NoSQL. The particular suitability of a given NoSQL database depends on the problem it must solve. Sometimes the data structures used by NoSQL databases are also viewed as "more flexible" than relational database tables.

NoSQL databases are generally classified into four main categories:

1. **Document databases:** These databases store data as semi-structured documents, such as JSON or XML, and can be queried using document-oriented query languages.
2. **Key-value stores:** These databases store data as key-value pairs, and are optimized for simple and fast read/write operations.
3. **Column-family stores:** These databases store data as column families, which are sets of columns that are treated as a single entity. They are optimized for fast and efficient querying of large amounts of data.
4. **Graph databases:** These databases store data as nodes and edges, and are designed to handle complex relationships between data.

Query optimization

Query optimization is a feature of many relational database management systems and other databases such as NoSQL and graph databases. The **query optimizer** attempts to determine the most efficient way to execute a given query by considering the possible query plans.[1]

Generally, the query optimizer cannot be accessed directly by users: once queries are submitted to the database server, and parsed by the parser, they are then passed to the query optimizer where optimization occurs.[2][3] However, some database engines allow guiding the query optimizer with hints.

Most query optimizers represent query plans as a tree of "plan nodes". A plan node encapsulates a single operation that is required to execute the query. The nodes are arranged as a tree, in which intermediate results flow from the bottom of the tree to the top. Each node has zero or more child nodes—those are nodes whose output is fed as input to the parent node. For example, a join node will have two child nodes, which represent the two join operands, whereas a sort node would have a single child node (the input to be sorted). The leaves of the tree are nodes which produce results by scanning the disk, for example by performing an index scan or a sequential scan.

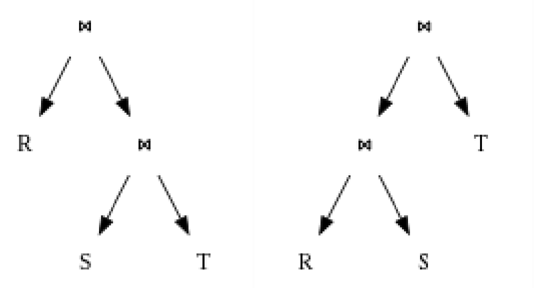
Database Optimizer

A query optimizer chooses an optimal index and access paths to execute the query. At a very high level, SQL optimizers decide the following before creating the execution tree:

1. Query rewrite based on heuristics, cost or both.
2. Index selection.
   - Selecting the optimal index(es) for each of the table (keyspaces in Couchbase N1QL, collection in case of MongoDB)
   - Depending on the index selected, choose the predicates to push down, see the query is covered or not, decide on sort and pagination strategy.
3. Join reordering
4. Join type

Implementation

Join ordering



Two possible query plans for the *triangle query* R(A, B) ⋈ S(B, C) ⋈ T(A, C); the first joins *S* and *T* first and joins the result with *R*, the second joins *R* and *S* first and joins the result with *T*

The performance of a query plan is determined largely by the order in which the tables are joined. For example, when joining 3 tables A, B, C of size 10 rows, 10,000 rows, and 1,000,000 rows, respectively, a query plan that joins B and C first can take several orders-of-magnitude more time to execute than one that joins A and C first. Most query optimizers determine join order via a dynamic programming algorithm pioneered by IBM's System R database project. This algorithm works in two stages:

1. First, all ways to access each relation in the query are computed. Every relation in the query can be accessed via a sequential scan. If there is an index on a relation that can be used to answer

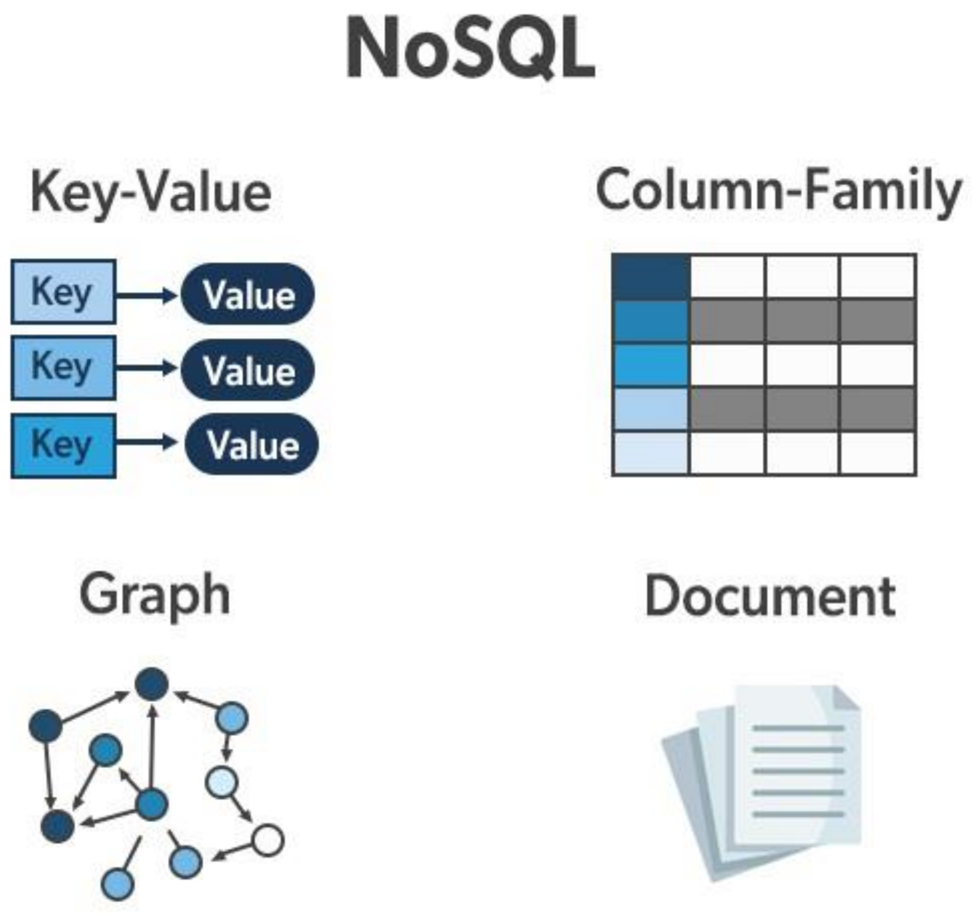
a predicate in the query, an index scan can also be used. For each relation, the optimizer records the cheapest way to scan the relation, as well as the cheapest way to scan the relation that produces records in a particular sorted order.

2. The optimizer then considers combining each pair of relations for which a join condition exists. For each pair, the optimizer will consider the available join algorithms implemented by the DBMS. It will preserve the cheapest way to join each pair of relations, in addition to the cheapest way to join each pair of relations that produces its output according to a particular sort order.
3. Then all three-relation query plans are computed, by joining each two-relation plan produced by the previous phase with the remaining relations in the query.

Sort order can avoid a redundant sort operation later on in processing the query. Second, a particular sort order can speed up a subsequent join because it clusters the data in a particular way.

Types of NoSQL Database

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



Document-Based Database:

The document-based database is a nonrelational database. Instead of storing the data in rows and columns (tables), it uses the documents to store the data in the database. A document database stores data in JSON, BSON, or XML documents.

Documents can be stored and retrieved in a form that is much closer to the data objects used in applications which means less translation is required to use these data in the applications. In the Document database, the particular elements can be accessed by using the index value that is assigned for faster querying.

Collections are the group of documents that store documents that have similar contents. Not all the documents are in any collection as they require a similar schema because document databases have a flexible schema.

Key features of documents database:

- Flexible schema: Documents in the database has a flexible schema. It means the documents in the database need not be the same schema.
- Faster creation and maintenance: the creation of documents is easy and minimal maintenance is required once we create the document.
- No foreign keys: There is no dynamic relationship between two documents so documents can be independent of one another. So, there is no requirement for a foreign key in a document database.
- Open formats: To build a document we use XML, JSON, and others.

Key-Value Stores:

A key-value store is a nonrelational database. The simplest form of a NoSQL database is a key-value store. Every data element in the database is stored in key-value pairs. The data can be retrieved by using a unique key allotted to each element in the database. The values can be simple data types like strings and numbers or complex objects.

A key-value store is like a relational database with only two columns which is the key and the value.

Key features of the key-value store:

- Simplicity.
- Scalability.
- Speed.

Column Oriented Databases:

A column-oriented database is a non-relational database that stores the data in columns instead of rows. That means when we want to run analytics on a small number of columns, you can read those columns directly without consuming memory with the unwanted data.

Columnar databases are designed to read data more efficiently and retrieve the data with greater speed. A columnar database is used to store a large amount of data. Key features of columnar oriented database:

- Scalability.
- Compression.
- Very responsive.

Graph-Based databases:

Graph-based databases focus on the relationship between the elements. It stores the data in the form of nodes in the database. The connections between the nodes are called links or relationships.

Key features of graph database:

- In a graph-based database, it is easy to identify the relationship between the data by using the links.
- The Query’s output is real-time results.
- The speed depends upon the number of relationships among the database elements.
- Updating data is also easy, as adding a new node or edge to a graph database is a straightforward task that does not require significant schema changes.

Querying in NoSQL

Querying in NoSQL:

Suppose we want to get specific results on the **transport** database.

```
{
"Brand":"Benz"
"Max_Speed":250
"Color":"Green"
}
```

1. **To display the vehicles which have a speed greater than 100.**

Query:

```
>db.transport.find({Max_speed:
 {$gt:100}}).pretty()
```

Output:

```
{
"Brand":"Benz"
"Max_Speed":250
"Color":"Green"
}
```

2. To display the vehicles which have a speed equal to 250.

Query:

```
>db.transport.find({Max_speed:
 {$eq:250}}}.pretty()
```

Output:

```
{
"Brand":"Benz"
"Max_Speed":250
"Color":"Green"
}
```

$eq – This operator is used to check 2 values and returns the data which is equal to the specified value. So like this, we have $gte (greater than or equal to), $lte (lesser than or equal to), $lt(less than), $ne(Not equal) in NoSQL.

Indexing in MongoDB

Indexing in MongoDB :
MongoDB uses indexing in order to make the query processing more efficient. If there is no indexing, then the MongoDB must scan every document in the collection and retrieve only those documents that match the query. Indexes are special data structures that stores some information related to the documents such that it becomes easy for MongoDB to find the right data file. The indexes are order by the value of the field specified in the index.
Creating an Index :
MongoDB provides a method called createIndex() that allows user to create an index.
Syntax –

```
db.COLLECTION_NAME.createIndex({KEY:1})
```

The key determines the field on the basis of which you want to create an index and 1 (or -1) determines the order in which these indexes will be arranged(ascending or descending).

Example –

```
db.mycol.createIndex({“age”:1})

{

“createdCollectionAutomatically” : false,

“numIndexesBefore” : 1,

“numIndexesAfter” : 2,

“ok” : 1

}
```

The createIndex() method also has a number of optional parameters.
These include:

- background (Boolean)
- unique (Boolean)
- name (string)
- sparse (Boolean)
- expireAfterSeconds (integer)
- hidden (Boolean)
- storageEngine (Document)

Drop an index:
In order to drop an index, MongoDB provides the dropIndex() method.
Syntax –

```
db.NAME_OF_COLLECTION.dropIndex({KEY:1})
```

The dropIndex() methods can only delete one index at a time. In order to delete (or drop) multiple indexes from the collection, MongoDB provides the dropIndexes() method that takes multiple indexes as its parameters.

Syntax –

```
db.NAME_OF_COLLECTION.dropIndexes({KEY1:1, KEY2, 1})
```

The dropIndex() methods can only delete one index at a time. In order to delete (or drop) multiple indexes from the collection, MongoDB provides the dropIndexes() method that takes multiple indexes as its parameters.

Get description of all indexes :
The getIndexes() method in MongoDB gives a description of all the indexes that exists in the given collection.
Syntax –

```
db.NAME_OF_COLLECTION.getIndexes()
```

It will retrieve all the description of the indexes created within the collection.

ordering data sets in NoSQL databases.

- One way to create indexes and order data sets in NoSQL is to use the **clustering order** option, which allows you to specify the sort order of the data within a partition.
- Another way is to use a **time bucketing** technique, which limits the number of records per partition by grouping them into time intervals.
- You can also use different types of NoSQL databases, such as document, key-value, column-family or graph, depending on your data model and query needs.
- You should also consider the size of your data sets and the performance of your NoSQL system, as they may affect the query speed and efficiency.

NOSQL in Cloud

NoSQL Cloud Database Services are cloud-based database services that provide scalable, high-performance, and cost-effective solutions for storing and retrieving data. NoSQL (Not Only SQL) databases are designed to handle large volumes of unstructured, semi-structured, and structured data, and can easily scale horizontally to accommodate increased data volumes.

Cloud-based NoSQL databases offer several advantages over traditional on-premise databases. These include:

1. Scalability: Cloud-based NoSQL databases can easily scale horizontally by adding more servers to the cluster. This allows for seamless scalability as data volumes increase.
2. High availability: NoSQL cloud databases are designed to be highly available and can provide reliable uptime and performance, which is critical for many applications.
3. Reduced cost: Cloud-based NoSQL databases can be more cost-effective than traditional on-premise databases because they eliminate the need for expensive hardware and infrastructure. This can be particularly beneficial for small to medium-sized businesses that do not have the resources to invest in expensive hardware.
4. Improved performance: Cloud-based NoSQL databases can provide high performance and low latency, making them well-suited for applications that require fast and efficient data access.
5. Flexibility: Cloud-based NoSQL databases are designed to handle unstructured, semi-structured, and structured data, making them a flexible solution for a wide range of applications.

Some popular NoSQL Cloud Database Services include:

1. Amazon DynamoDB: A fully managed NoSQL database service offered by Amazon Web Services (AWS) that provides fast and predictable performance with seamless scalability.
2. Google Cloud Datastore: A NoSQL document database service that is fully managed and offers automatic scaling, high availability, and low latency.
3. Microsoft Azure Cosmos DB: A globally distributed, multi-model database service that provides high availability, low latency, and flexible data modeling.
4. MongoDB Atlas: A fully managed global cloud database service for MongoDB that provides automated backups, advanced security, and easy scalability.
5. Overall, NoSQL Cloud Database Services provide a flexible, scalable, and cost-effective solution for storing and retrieving data in the cloud. They offer several advantages over traditional on-premise databases and

can be an excellent choice for businesses of all sizes that need to store and manage large volumes of data.