A static analysis of the supported data models and query languages

Antonio José González Castillo

Charles University / Universidad de Málaga

NDBI040 - Modern Database Systems

Irena Holubová

CRATEDB OVERVIEW

CrateDB is an open-source SQL database management system that's designed to handle a variety of data types including structured and unstructured data, making it suitable for applications like IoT, machine data analysis, and large-scale applications. It combines the scalability of NoSQL with the ease of use of SQL.

Supported Data Models in CrateDB

- Document-Oriented Data Model: CrateDB primarily implements a document-oriented data model. It stores data in JSON-like structures (objects), which can be nested. Each document is schema-less, which means you can have different structures for each document if needed. However, CrateDB also supports defining a strict schema to enforce data integrity. Details: Documents are stored in tables, which are analogous to traditional relational database tables but each row in the table can be a complex document with nested and varying structures.

- Relational Data Model: Despite its NoSQL capabilities, CrateDB also supports the relational data model. It allows the definition of table schemas with standard SQL data types and supports SQL joins, aggregates, and transactions. Details: This model is implemented using traditional table structures where data can be inserted, queried, updated, and deleted using standard SQL syntax. CrateDB enhances this model by automatically distributing data across multiple nodes for scalability and resilience.

- Blob Storage: CrateDB includes support for unstructured Blob (Binary Large Object) storage, which allows it to store and manage large objects such as multimedia files. Details: Blobs are stored in containers called "blob tables", which provide efficient, distributed storage and retrieval of binary data.

Supported Query Languages in CrateDB

- SQL: Details: CrateDB uses SQL as its primary query language, which is enhanced to handle its NoSQL features. It supports a broad subset of standard SQL with extensions for handling dynamic objects (JSON) and full-text search. This includes aggregation, joins, indexing, and more.

- Dynamic SQL Capabilities: CrateDB supports dynamic schema evolution. Columns can be added to existing tables without downtime, and the schema of JSON objects can evolve as new fields are added in the queries.

ANALYTICAL QUESTIONS

How is the Data Model Implemented in CrateDB?

Underlying Storage and Indexing: CrateDB uses a columnar storage format, which allows it to efficiently store and query large datasets. This is particularly useful for analytical queries that need to scan large amounts of data.

- Distributed Architecture: Data in CrateDB is automatically sharded and distributed across multiple nodes. This ensures high availability and scalability. The distributed nature also allows for parallel processing of queries, enhancing performance.

- Handling of Data Models: The integration of different data models is seamless in CrateDB. The document model is implemented over the relational structure, allowing users to query and manipulate JSON documents using SQL. The blob storage utilizes a distributed file storage system integrated within the database.

How can different models be linked and worked with?

- Linking Document and Relational Models
    - Using SQL on JSON Documents: CrateDB allows you to store JSON documents in regular SQL tables and query them using SQL syntax. This integration means you can perform relational operations on document data. Example: Suppose you have a table businesses with a JSON column details that stores additional information about each business (like reviews, hours, etc.). You can query this JSON data alongside relational data using SQL queries.

- Joining Document and Relational Data: CrateDB supports SQL joins between tables, even if one or more of the tables contain JSON documents. This allows you to perform complex queries that integrate structured and semi-structured data. Example: Joining a document-stored table businesses with a relational table categories.

- Linking Blob Storage with Relational and Document Models

  Accessing Blob Data via SQL: In CrateDB, blobs are stored in special blob tables, which can be queried using their unique hash keys. Example: If you store images related to business reviews in a blob table, you can reference these blobs using a key in your main reviews table.

- Utilising SQL Features for Complex Queries
  - Built-In Full-Text Search: CrateDB supports full-text indexing and search capabilities, particularly useful for document data stored in JSON columns. Example: Searching text within a JSON document
  - Aggregation and Analysis: You can perform SQL aggregations not only on traditional numeric fields but also on data extracted from JSON documents. Example: Aggregate operations on document attributes

References: https://cratedb.com/docs/crate/reference/en/latest/

Are indexes over attributes represented in a particular data model supported?

CrateDB does not support index creation in the same way as other relational databases. Instead, CrateDB uses an automatic indexing architecture where indexes are automatically created and maintained as needed to optimise query performance.

That said, in CrateDB you don't need to create indexes manually as in other databases. The CrateDB engine will automatically index relevant columns based on the nature of your queries and data access patterns.

https://cratedb.com/docs/crate/reference/en/latest/general/ddl/fulltext-indices.html

What querying means are supported over the data expressed by the different supported models?

Type of construct	Relational	Document (JSON)	Blob Storage
Projection	`SELECT`	`SELECT` (supports nested fields with dot notation)	Not applicable
Source	`FROM`	`FROM`	Not applicable
Selection	`WHERE`	`WHERE` (supports querying nested JSON fields)	Not applicable
Aggregation/aggregation functions	`GROUP BY ... HAVING`	`GROUP BY ... HAVING` (supports aggregation on JSON fields)	Not applicable
Join	`JOIN`	`JOIN` (supports joins using JSON fields)	Not applicable
Graph Traversal (JOIN)	Not supported	Not supported	Not applicable
Unlimited Traversal	`WITH RECURSIVE`	Not supported	Not applicable
Optional	`OUTER JOIN`	`OUTER JOIN`	Not applicable
Union	`UNION`	`UNION`	Not applicable
Intersection	`INTERSECTION`	Not supported	Not applicable
The Difference	`EXCEPT`	Not supported	Not applicable
Sorting (on non/indexed columns)	`ORDER BY`	`ORDER BY` (supports JSON fields)	Not applicable
Skipping	`OFFSET`	`OFFSET`	Not applicable
Limitation	`LIMIT`	`LIMIT`	Not applicable
Distinct	`DISTINCT`	`DISTINCT` (supports distinct on JSON fields)	Not applicable
Aliasing	`AS`	`AS`	Not applicable
Nesting	`(SELECT ...)`	`(SELECT ...)` (nested queries including subselects on JSON arrays)	Not applicable
MapReduce	Not directly supported	Not directly supported	Not applicable