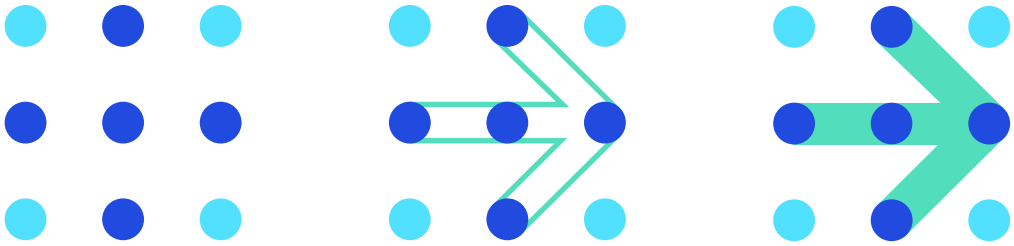
WHITE PAPER



SQL Support in DataStax Enterprise

Transitioning SQL Skills for Relational Databases to NoSQL Distributed Databases

DATASTAX

This paper serves as a guide for developers and database architects to help simplify the transition from traditional relational databases to NoSQL databases through SQL support for Apache Cassandra™.

Cassandra was first to market with a SQL-like language for NoSQL platforms, and since that time, DataStax has continued to build out additional SQL support for not only operational database tasks, but for analytic and search functions as well. SQL support in DataStax Enterprise (DSE) helps ease the transition that RDBMS developers and administrators have to make to the non-relational/NoSQL world, and makes building modern applications easier than with platforms that don't offer this familiar database syntax.

This paper describes the Structured Query Language (SQL) support in DataStax Enterprise (DSE). It covers the history behind the development of the Cassandra Query Language (CQL), how Spark SQL is used with DSE Analytics, and the extensions made to CQL to support enterprise search operations. It also briefly discusses how graph database access compares to SQL.

The term "NoSQL" was originally coined to describe database systems that organized and accessed stored data in a manner other than the relational model popularized by Codd and Date.

While such databases technically predated relational database management systems (RDBMS) and have been represented by filesystems and databases such as IBM IMS, the modern NoSQL movement began with the internet pioneers who found that relational databases didn't support their widely distributed data and scaling requirements, and thus began to build out other means to manage their data.

While NoSQL originally referred to databases that didn't use SQL, it has since been more broadly defined as "not only SQL," with a number of NoSQL vendors now including SQL or SQL-like support in their platforms. However, these platforms still differ from relational databases in their support of SQL, especially as it relates to legacy transactional operations and data relationship management.

01

SQL Support for Operational Database Tasks in DSE

History

For those who have been around Apache Cassandra for a while, the term “Thrift” may sound familiar. DataStax and the Cassandra community recognized that writing applications that used this framework presented a hurdle to those coming from the relational SQL world. For these reasons, CQL was created with the goal of forming an interface for DSE and Cassandra that shared common ground with SQL and promoted more efficient access patterns. Though the form is similar to SQL, there are important differences between how SQL and CQL interact with the database, and how common terms and concepts are defined.

Database Definition Language (DDL) Support

CREATE

Databases and Keyspaces
Starting from the top, the largest namespace unit in the relational world is the database that holds tables. It is possible to have many databases that hold many tables, and DSE and Cassandra use this same concept, keyspaces being the largest namespace unit that holds many tables. Because DSE and Cassandra are distributed, data replication is an important aspect—the fundamental difference between these units is that keyspaces in DSE also define the data replication, so there can be anywhere from one to many copies of a single row stored throughout the cluster. This is defined during keyspace creation and requires a change in mindset when going from SQL to CQL. Consult the DataStax Documentation for a thorough overview of these concepts.

Tables
The next structural unit is called the table, and keyspaces hold tables. This is where the form of the data is defined and stored. The two interfaces share a common nomenclature for creating tables, but again there are intricacies in details of the storage mechanisms that must be understood to properly leverage the system. Both SQL and CQL have the notion of a PRIMARY KEY that is used to uniquely identify each row/record in the table. PRIMARY KEYs can be comprised of either a single column or multiple columns. DSE requires at least one PRIMARY KEY to be declared when creating a table to define the partition in the distributed cluster where the data resides. SQL systems do not enforce this constraint since they are not generally distributed. Though this constraint is not enforced, these keys should be used for optimized performance for common queries in SQL systems. For more details on creating tables and data modeling, visit DataStax Documentation and DataStax Academy.

ALTER/DROP/TRUNCATE

For the remainder of the DDL commands, it is critical to understand that concurrent schema changes in a distributed environment can be hazardous to the data's integrity. Schema changes naturally take time to propagate to all nodes in the cluster depending on the size, network capacity, and load on the cluster. Caution should be top of mind when altering a schema in DSE and a general best practice is to make these changes one at a time and ensure they have fully spread throughout the cluster before moving on to subsequent alterations. For more information, see this section of the DataStax Documentation.

Data Manipulation Language (DML) Support

SELECT

```
SELECT col_0, col_1 FROM my_table;
```

Retrieving all rows in a table looks the same when comparing SQL and CQL, and is an expensive operation in both systems. Though when using a distributed system like DSE, remember that this task will fetch from multiple machines, so queries should be carefully structured to avoid adding expensive latency to the query response.

```
SELECT col_0, col_1 FROM my_table WHERE ...;
```

Restricting result sets using a qualifying WHERE clause is one of the most important differences to understand between SQL and CQL. Again, the syntax looks the same, but the columns that are allowed to be used in the WHERE clause varies construction of the WHERE clause varies.

In SQL, any column can be included in the WHERE clause; in CQL only PRIMARY KEY columns can restrict a query, in a strict order defined in the schema. This is because the distribution mechanism in DSE needs the PRIMARY KEY to compute the hash that holds the location of where the data lives in the cluster. The distribution mechanism of DSE requires that queries first define the partition where the data resides, as defined by the PRIMARY KEY, to reduce the number of machines queried. The WHERE clause can also define groups of rows to retrieve based in CLUSTERING COLUMNS.

Users coming from a SQL background often use the CQL CREATE INDEX to bypass this restriction. Though it can be used to query non-PRIMARY KEY columns in the WHERE clause, there are factors to be aware of when using these strategies. Secondary indexes (CREATE INDEX) for example, have a prescribed use and are good for searching a single partition (not for multiple partitions), and are a per-machine mechanism.

For details on the usage of this index mechanisms, see the CREATE INDEX documentation.

```
SELECT count(*) FROM my_table;
```

One of the most common questions is why the above query typically times out in DSE. Remember that this access pattern requires a scan from all nodes and places a significant load on all replicas, and without a partition all machines would have to be queried. That said, it is a common operation for database professionals and this functionality is built into the DataStax Bulk Loader.

```
dsbulk count -k my_keyspace -t my_table -h 'localhost'
```

To count all of the rows in a table, the standalone bulk loader tool shipped with DSE is recommended and above command is an example of the operation using this tool.

INSERT

```
INSERT INTO my_table (col_0, col_1) VALUES (val_0, val_1);
```

DSE and Cassandra are best in class at high-throughput writes. One difference is the ability to set DEFAULT column values in SQL. This concept does not exist in CQL and instead the values for columns must be passed explicitly in the write. This has relevance here because rows cannot be written without a value for the PRIMARY KEY columns in both SQL and CQL systems. For syntax specifics for INSERT, reference the DataStax Documentation.

Tombstones are DSE's way of managing deletes and null columns, since every INSERT in DSE is an upset that replaces the data with newer timestamped data.

A second topic to keep in mind when writing data to DSE is the concept of tombstones. Tombstones are DSE's way of managing deletes and null columns, since every INSERT in DSE is an upset that replaces the data with newer timestamped data. Take a look at the following queries in CQL. The first does not produce a tombstone, but the second does. This has relevance for the application, as queries should be written to avoid providing null values. This is something to keep in mind, especially if an Object Mapper is used, as these constructs typically assign a null value rather than leverage the UNSET capabilities that came with CQL protocol version 4.

No Tombstone

```
INSERT INTO my_table_compound_key (primary_key, clustering_key) VALUES ('pk1', 'ck1');
```

Tombstone for regular_col

```
INSERT INTO my_table_compound_key (primary_key, clustering_key, regular_col) VALUES ('pk1',
'ck1', null);
```

UPDATE

```
UPDATE my_table SET col_0=val_0 WHERE ...;
```

Similar to SELECT statements, DSE requires that all columns in the PRIMARY KEY be supplied when updating rows, to uniquely identify the location of the data in the cluster. For more, see the DataStax Documentation.

DELETE

Delete entire row(s)

```
SQL and CQL = DELETE FROM my_table WHERE ...;
```

Delete specific column from row(s)

```
SQL = UPDATE my_table SET my_col=null WHERE ...;
CQL = DELETE my_col FROM my_table WHERE ...;
```

DELETE is a bit different in SQL and CQL, as there is the ability to delete specific columns in CQL. In SQL, there is only the option to remove the entire row(s) using the DELETE syntax. The ability to delete specific columns in SQL is instead enacted through the UPDATE clause. Deleting specific columns in SQL involves inserting a NULL value, rather than deleting the column value from the table. More on DELETE in CQL can be found in the DataStax Documentation and this operation should be used with caution as it holds the same sharp edges with tombstones described above.

Data Control Language (DCL) Support

Below is a list of all of the DCL commands available in CQL. These are used to control permissions and resources of the entities logged into the system.

- CREATE ROLE
- ALTER ROLE
- DROP ROLE
- LIST ROLES
- GRANT
- REVOKE
- RESTRICT
- UNRESTRICT
- RESTRICT ROWS
- UNRESTRICTED ROWS
- LIST PERMISSIONS

With support for the most popular programming languages, the DataStax Drivers should be the first stop for application developers looking to build modern digital experiences with DSE.

Query Support

As evidenced by the above examples, SQL and CQL share a lot of the same semantic characteristics for transactional support. For all transactional queries, the DataStax Drivers are recommended for intelligent, scalable, efficient, and extensible client libraries that have all of the tools needed to access the data stored in DSE. These drivers use the CQL protocol and include built-in synchronous and asynchronous APIs, enterprise security, load balancing, retry, and reconnection policies as well as a long list of features that foster a modern application experience backed by a distributed data store. With support for the most popular programming languages, the DataStax Drivers should be the first stop for application developers looking to build modern digital experiences with DSE.

ODBC SUPPORT

For those who are not yet ready to make the full transition from SQL to CQL, we also provide an ODBC driver for transactional workloads. We recommend restricting the use of this driver to single-partition lookups, as typical SQL operations such as joins will not meet performance expectations with this driver due to distributed access patterns that are fundamental to DSE and Cassandra's architecture. For more information on advanced operations, see this blog post and the SQL Support for DSE Analytics section of this whitepaper.

CQL Support in DataStax Studio

To support modern application development, distributed databases and DSE, DataStax Studio is designed as a tool for developers and analysts to explore and optimize queries. DataStax Studio provides a user-friendly visual interface that ships with built-in walk-throughs to help form and optimize CQL requests. The built-in Working with CQL notebook steps through the capabilities to dig into the schema, view the results from tables in various formats, and observe the tracing path used to return the data. Try this notebook to learn more.

02

SQL Support for Analytics

Direct SQL Support is exposed in a variety of flavors in DSE Analytics. The first is uniquely exposed in the AlwaysOn SQL service that premiered in DSE 6.

Whether to process streaming and historical data or to improve ongoing BI reporting, DSE Analytics delivers the flexibility to transform data into meaningful action at cloud scale.

This feature builds on top of the Spark SQL Thriftserver that is present in older DSE versions and combines enterprise security with reliability for analytical SQL queries through ODBC or JDBC drivers. DSE also allows for the ability to execute full Spark SQL applications through the use of a standard `dse spark-submit` job and direct queries through the `dse sparksql` shell.

Analytics Support in DSE

DSE Analytics, built on a production-certified version of Apache SparkTM and with enhanced capabilities like AlwaysOn SQL and a highly available Spark resource manager, enables enterprises to build real-time, contextual applications to make highly relevant, in-the-moment decisions. Whether to process streaming and historical data or to improve ongoing BI reporting, DSE Analytics delivers the flexibility to transform data into meaningful action at cloud scale.

AlwaysOn SQL

The AlwaysOn SQL service released with DSE 6 represents an important step to providing always-on SQL access to data, in support of business analysts and the tools that they use. This service enables the broad group of developers with the familiar SQL interface to analyze the data stored in DSE without having to climb the learning curve of Spark programming that the Scala, Java, and Python interfaces pose. In addition, this service was designed from the ground up to support production deployments in enterprise environments, focusing on availability and security as the two core tenets. Specifically, this service removes the need for an operator to intervene when things go wrong or nodes are brought down for maintenance while ensuring enterprise security requirements are met by integrating with DSE's Advanced Security.

ODBC/JDBC Support

DSE partners with Magnitude to ensure that users can analyze data stored in DSE via the standard ODBC and JDBC protocols. These ODBC and JDBC drivers for DSE Analytics are hosted on the DataStax website. Some are available through the repo.datastax.com and have built-in functionality to leverage the availability and security characteristics of the AlwaysOn SQL service to enable enterprise-grade BI, ETL, and other analytical tools.

SQL Analytics Support

In addition to the AlwaysOn SQL and ODBC/JDBC support, DataStax exposes programmatic access to Spark SQL through the Apache Spark libraries. These libraries can be used to build a Spark SQL app that is executed with `dse spark-submit`. This technique is a more advanced procedure for operators and builders and can be a drain on resources versus simply connecting via ODBC/JDBC and the AlwaysOn SQL service. For these reasons, it is recommended to start with ODBC/JDBC and the AlwaysOn SQL service and then move to this programmatic style as more complex Spark operations become necessary. For further details, see the supported Spark SQL syntax and programmatic access tips in the DataStax Documentation.

Analytics Support in DataStax Studio

Similar to CQL support in DataStax Studio, there is also the ability to execute Spark SQL queries in this visual interface for developers. This feature is new to DataStax Studio 6 and provides the ability to interactively perform Spark SQL queries against a DSE cluster that has analytics and the AlwaysOn SQL service enabled. In addition, there is schema-aware assist and syntax validation to foster faster prototyping. For more, see the DataStax Documentation and you can try it out by downloading DataStax Studio.

03

SQL Support for Search

There are capabilities in SQL to search for patterns using wildcards and the LIKE clause in queries. This same functionality is expressed in DSE through CQL, and is powered by DSE Search indexes.

Search Support in DSE

The search support in DSE extends far beyond that of the traditional relational database and leverages the advanced techniques of Apache Solr, powered by Lucene™, to enable matching on phrases, wildcards, joins, and grouping across an extended range of data types. Though many of these more complex techniques manifest in the Solr_query syntax in CQL, as of DSE 6, the familiar search functionality from the SQL world has been integrated with the core CQL semantics.

BOOLEAN OPERATORS

```
SELECT * FROM my_table WHERE primary_key boolean operator 'some_value';
```

DSE supports boolean operators =, <=, >=, < and > through normal CQL syntax where the columns in the PRIMARY KEY are used to filter the query. These operators can also be used on non-PRIMARY KEY columns by creating a DSE Search index on the regular columns.

```
CREATE SEARCH INDEX ON my_table WITH COLUMNS regular_col;
SELECT * FROM my_table WHERE regular_col!='some_value';
SELECT * FROM my_table WHERE regular_col IS NOT NULL;
```

LIKE

```
CREATE SEARCH INDEX ON my_table WITH COLUMNS regular_col;
SELECT * FROM my_table WHERE regular_col LIKE '%some_value%';
```

In the SQL world, LIKE can be used in the WHERE clause on any column to retrieve text results that match the supplied pattern. This functionality is mimicked in CQL with DSE Search indexes.

CONTAINS

```
SELECT * FROM my_table WHERE collection_col CONTAINS ‘some_value’;
```

The ability to search collections is exposed in the CQL world when the collection column is part of the PRIMARY KEY or when there is a DSE Search index on the collection column. The above returns all rows that have ‘some_value’ in the collection specified.

IN

```
SELECT * FROM my_table WHERE primary_key IN (‘some_value_0’, ‘some_value_1’);
```

To retrieve rows for a set of qualifiers, the IN clause can be used on columns that are part of the PRIMARY KEY. It is advised to limit the number of columns used in the IN clause as the access efficiency will decrease with the number of columns added. For more, see the Datastax Documentation.

GEOSPATIAL

DSE has specific types, namely Point, LineString and Polygon for geospatial data. This feature is incredibly useful for location based services.

DATE RANGES

```
CREATE TABLE my_table (key int PRIMARY KEY, daterange_col ‘DateRangeType’);
CREATE SEARCH INDEX ON my_table WITH COLUMNS daterange_col;
SELECT * FROM my_table WHERE solr_query = ‘daterange_col:[2018-01-01 TO *]’;
SELECT * FROM my_table WHERE solr_query = ‘daterange_col:2018-04-12’;
SELECT * FROM my_table WHERE solr_query = ‘daterange_col:[2018-04-12 TO 2018-04-13]’;
```

DSE Search extends powerful utilities for date ranges and filtering by single points in time or open bound lookups. This DateRangeType offers microsecond precision and time zone customization.

For adequate performance, the DSE Search indexes must be stored in memory, so planning ahead for your data set will go a long way in maximizing the utility of the available resources.

Do’s and Don’ts

DO’S

Before any of the above operations can be performed, DSE Search must be enabled on the node. Also, it is important to remember to index only the columns that require advanced search functionality. For adequate performance, the DSE Search indexes must be stored in memory, so planning ahead for your data set will go a long way in maximizing the utility of the available resources.

For behavior in line with ANSI SQL, use the StringField type for columns included in the DSE Search index and TextField types for more complex search engine functionality.

DON’TS

DSE Search does not have the same concept of consistency that is fundamental for regular transactional queries so do not rely on search lookups if strong consistency is needed. DSE Search queries are only directed toward data centers that have DSE Search enabled and an index for that data created, or the queries will be rejected. As mentioned in the Do’s section above, be smart with the DSE Search indexes as they are not free of cost in terms of latency. Additional strain is induced on the machines by indexing and large indexes will impact performance.

SQL Support for Graph

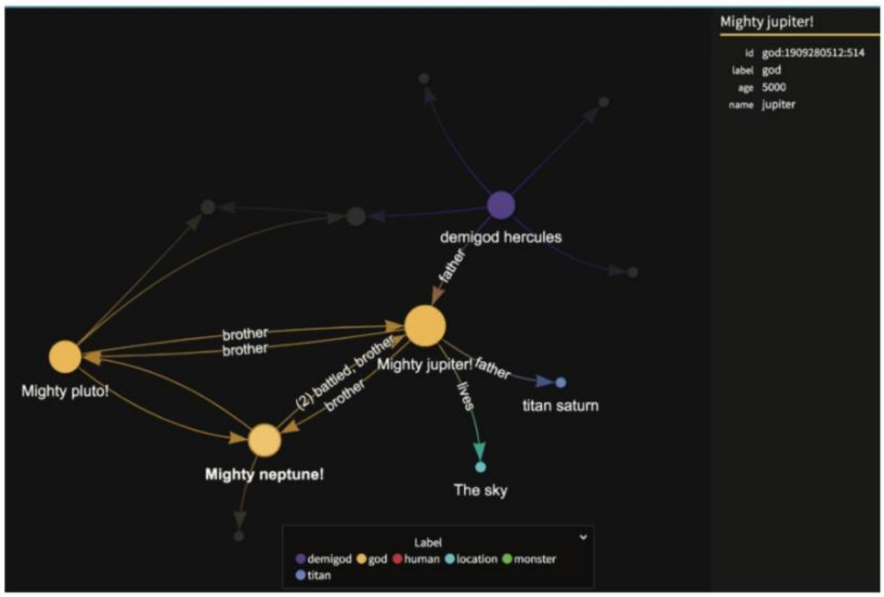
Graph solutions identify connections and relationships that are essential for solving modern fraud detection and personalization use cases. These problems require a new way of storing and accessing data, and the same expressions in SQL do not fit nicely into this new paradigm. To adequately address the emerging demand of the graph database, DataStax invests heavily into the Apache Tinkerpop™ graph computing framework that leverages Gremlin as its property graph query language and core API. Gremlin is purpose-built for graph expression and provides an intuitive language for property graph databases that SQL can not offer. More comparisons between DSE Graph and relational database systems can be found in the DataStax Documentation as well as the website SQL2Gremlin.

GRAPH DATABASE SUPPORT IN DSE

DSE Graph is a key component of the DSE multi-model stack. Built on the best distribution of Apache Cassandra, DSE Graph is constructed from the ground up to handle large, enterprise-grade graph applications.

DSE GRAPH DEVELOPER SOLUTIONS

Learning new technologies can be a daunting task, especially when learning one as cutting edge as DSE Graph. Because of this learning curve, DataStax has focused on DSE Graph developer solutions to make this transition as seamless as possible. To get a better understanding and overview of graph technology, take a look at the DSE Graph Quickstart guide. DataStax Studio is the tool for those getting started with DSE Graph, offering built-in content assist, interactive visual display of graphs, and tools to analyze query paths. For application development, the DataStax Drivers offer a DSE Graph Fluent API that combines all of the advanced features of the DSE Drivers with a programmatic interface for constructing Gremlin traversals.



Example of the Interactive Graph feature in DSE Graph

Conclusion

SQL support in DSE helps ease the transition for developers and administrators that are familiar with SQL for relational databases and moving to a Cassandra/NoSQL database platform.

Cassandra was first to market with a SQL-like language for NoSQL platforms, and DataStax continues to review and maintain SQL support for operational as well as mixed workload database tasks, including analytic and search functions.

For more information on the SQL functionality contained in DSE, see the DataStax Documentation and visit the DataStax downloads page to try out DSE in your own environment.

DataStax helps companies compete in a rapidly changing world where expectations are high and new innovations happen daily. DataStax is an experienced partner in on-premises, hybrid, and multi-cloud deployments and offers a suite of distributed data management products and cloud services. We make it easy for enterprises to deliver killer apps that crush the competition.

More than 400 of the world's leading enterprises including Capital One, Cisco, Comcast, Delta Airlines, eBay, Macy's, McDonald's, Safeway, Sony, and Walmart use DataStax to build modern applications that can be deployed across any cloud. For more information, visit www.DataStax.com and follow us on Twitter @DataStax.

Learn More

datastax.com

DATASTAX