**3. Data Types**

Generally, Cassandra supports [a rich set of data types](https://www.baeldung.com/cassandra-data-types). These include native types, collection types, user-defined types, and tuples, together with custom types.

**3.1. Native Types**

The native types are the built-in types and provide support to a range of constants in Cassandra.

To begin with, a string is a very popular datatype in the programming world.

CQL offers four different datatypes for strings:

|  |  |  |
| --- | --- | --- |
| **Data Type** | **Constants Supported** | **Description** |
| ascii | *string* | ASCII character string |
| inet | *string* | IPv4 or IPv6 address string |
| text | *string* | UTF8 encoded string |
| varchar | *string* | UTF8 encoded string |

A boolean has one of two possible values, either *true* or *false*:

|  |  |  |
| --- | --- | --- |
| **Data Type** | **Constants Supported** | **Description** |
| boolean | *boolean* | *true* or *false* |

Duration is a three-signed integer that represents months, days, and nanoseconds:

|  |  |  |
| --- | --- | --- |
| **Data Type** | **Constants Supported** | **Description** |
| duration | *duration* | A duration value |

Cassandra offers a wide range of data types for integer data:

|  |  |  |
| --- | --- | --- |
| **Data Type** | **Constants Supported** | **Description** |
| tinyint | *integer* | 8-bit signed int |
| smallint | *integer* | 16-bit signed int |
| int | *integer* | 32-bit signed int |
| bigint | *integer* | 64-bit signed long |
| variant | *integer* | Arbitrary-precision integer |
| counter | *integer* | Counter column (64-bit signed) |

For integer and float, we have three data types:

|  |  |  |
| --- | --- | --- |
| **Data Type** | **Constants Supported** | **Description** |
| decimal | *integer, float* | Variable precision decimal |
| double | *integer, float* | 64-bit floating-point |
| float | *integer, float* | 32-bit floating-point |

For date- and time-related needs, Cassandra provides three data types:

|  |  |  |
| --- | --- | --- |
| **Data Type** | **Constants Supported** | **Description** |
| date | *integer, string* | A date value (without time) |
| time | *integer, string* | A time value (without date) |
| timestamp | *integer, string* | A timestamp (with date & time) |

### 4.4. Basic Commands

These commands are used to read and manipulate the table values:

|  |  |  |
| --- | --- | --- |
| **Command** | **Example** | **Description** |
| INSERT | INSERT INTO table\_name (column\_name1, column\_name2) VALUES(value1, value2); | To insert a record in a table. |
| SELECT | SELECT \* FROM table\_name; | The command is used to fetch data from a specific table. |
| WHERE | SELECT \* FROM table\_name WHERE column\_name=value; | It filters out records on a predicate. |
| UPDATE | UPDATE table\_name SET column\_name2=value2 WHERE column\_name1=value1; | It is used to edit records. |
| DELETE | DELETE identifier FROM table\_name WHERE condition; | This statement deletes the value from a table. |

**In comparison, the clustering key determines the order of data within a partition key:**

|  |  |  |
| --- | --- | --- |
| **Command** | **Example** | **Description** |
| ORDER BY | SELECT \* FROM *table\_name* WHERE *column\_name1* = *value* ORDER BY *cloumn\_name2* ASC; | For this, the partition key must be defined in the WHERE clause. Also, the ORDER BY clause represents the clustering column to use for ordering. |
| GROUP BY | SELECT *column\_name* FROM *table\_name* GROUP BY *condition1*, *condition2*; | This clause only supports with Partition Key or Partition Key and Clustering Key. |
| LIMIT | SELECT \* FROM table\_name LIMIT 3; | For a large table, limit the number of rows retrieved. |