<http://www.datastax.com/docs/1.1/ddl/indexes>

An index is a data structure that allows for fast, efficient lookup of data matching a given condition.

## About Primary Indexes[¶](http://www.datastax.com/docs/1.1/ddl/indexes#about-primary-indexes)

In relational database design, a primary key is the unique key used to identify each row in a table. A primary key index, like any index, speeds up random access to data in the table. The primary key also ensures record uniqueness, and may also control the order in which records are physically clustered, or stored by the database.

In Cassandra, the primary index for a column family is the index of its row keys. Each node maintains this index for the data it manages.

Rows are assigned to nodes by the cluster-configured [partitioner](http://www.datastax.com/docs/1.1/cluster_architecture/partitioning#partitioning) and the keyspace-configured [replica placement strategy](http://www.datastax.com/docs/1.1/cluster_architecture/replication#replication-strategy). The primary index in Cassandra allows looking up of rows by their row key. Since each node knows what ranges of keys each node manages, requested rows can be efficiently located by scanning the row indexes only on the relevant replicas.

With randomly partitioned row keys (the default in Cassandra), row keys are partitioned by their MD5 hash and cannot be scanned in order like traditional b-tree indexes. The CQL3 [token function](http://www.datastax.com/docs/1.1/dml/using_cql#non-ordered-partitioner-paging)introduced in Cassandra 1.1.1 now makes it possible to page through non-ordered partitioner results.

## About Secondary Indexes[¶](http://www.datastax.com/docs/1.1/ddl/indexes#about-secondary-indexes)

Secondary indexes in Cassandra refer to indexes on column values (to distinguish them from the primary row key index for a column family). Cassandra implements secondary indexes as a hidden table, separate from the table that contains the values being indexed. Using CQL, you can create a secondary index on a column after defining a column family. Using CQL 3, for example, define a static column family, and then create a secondary index on one of its named columns:

CREATE TABLE users (

userID uuid,

firstname text,

lastname text,

state text,

zip int,

PRIMARY KEY (userID)

);

CREATE INDEX users\_state

ON users (state);

Secondary index names, in this case users\_state, must be unique within the keyspace. After creating a secondary index for the state column and inserting values into the column family, you can query Cassandra directly for users who live in a given state:

SELECT \* FROM users WHERE state = 'TX';

userid | firstname | lastname | state | zip

-------------------------------------+-----------+----------+-------+-------

b70de1d0-9908-4ae3-be34-5573e5b09f14 | john | smith | TX | 77362

Secondary indexes allow for efficient querying by specific values using equality predicates (where column x = value y).

### Creating Multiple Secondary Indexes[¶](http://www.datastax.com/docs/1.1/ddl/indexes#creating-multiple-secondary-indexes)

You can create multiple secondary indexes, for example on the firstname and lastname columns of the users table, and use multiple conditions in the WHERE clause to filter the results:

CREATE INDEX users\_fname

ON users (firstname);

CREATE INDEX users\_lname

ON users (lastname);

SELECT \* FROM users WHERE users\_fname = 'bob' AND users\_lname = 'smith';

When there are multiple conditions in a WHERE clause, Cassandra's selects the least-frequent occurrence of a condition for processing first for efficiency. In this example, Cassandra queries on the last name first if there are fewer Smiths than Bobs in the database or on the first name first if there are fewer Bobs than Smiths.

### When to Use Secondary Indexes[¶](http://www.datastax.com/docs/1.1/ddl/indexes#when-to-use-secondary-indexes)

Cassandra's built-in secondary indexes are best on a column family having many rows that contain the indexed value. The more unique values that exist in a particular column, the more overhead you will have, on average, to query and maintain the index. For example, suppose you had a user table with a billion users and wanted to look up users by the state they lived in. Many users will share the same column value for state (such as CA, NY, TX, etc.). This would be a good candidate for a secondary index.

### When Not to Use Secondary Indexes[¶](http://www.datastax.com/docs/1.1/ddl/indexes#when-not-to-use-secondary-indexes)

Do not use secondary indexes to query a huge volume of records for a small number of results. For example, if you create indexes on columns that have many distinct values, a query between the fields will incur many seeks for very few results. In the column family with a billion users, looking up users by their email address (a value that is typically unique for each user) instead of by their state, is likely to be very inefficient. It would probably be more efficient to manually maintain a dynamic column family as a form of an index instead of using a secondary index. For columns containing unique data, it is sometimes fine performance-wise to use secondary indexes for convenience, as long as the query volume to the indexed column family is moderate and not under constant load.

### Building and Using Secondary Indexes[¶](http://www.datastax.com/docs/1.1/ddl/indexes#building-and-using-secondary-indexes)

An advantage of secondary indexes is the operational ease of populating and maintaining the index. Secondary indexes are built in the background automatically, without blocking reads or writes. Client-maintained column families as indexes must be created manually; for example, if the state column had been indexed by creating a column family such as users\_by\_state, your client application would have to populate the column family with data from the users column family.

### Maintaining Secondary Indexes[¶](http://www.datastax.com/docs/1.1/ddl/indexes#maintaining-secondary-indexes)

To perform a hot rebuild of a secondary index, use the nodetool utility rebuild\_index command.