Máster de Formación Permanente en BIG DATA e Inteligencia Artificial

Máster en

Big Data, Inteligencia Artificial e Ingeniería de Datos

Khaos INVESTIGACIÓN

Module 4

NoSQL Databases

María del Mar Roldán – University of Málaga

Table of contents

- Cassandra

Cassandra

Definition of Cassandra

Apache Cassandra™ is a free
Distributed…
High performance…
Extremely scalable…
Fault tolerant (i.e. no single point of failure)…

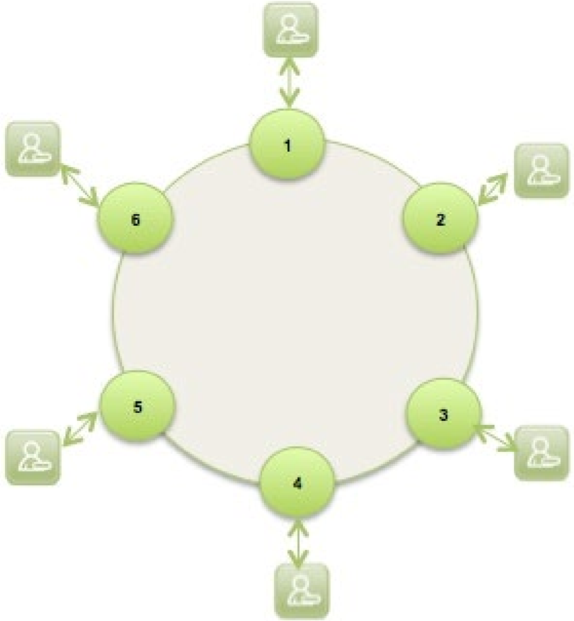
post-relational database solution. Cassandra can serve as both real-time datastore (the “system of record”) for online/transactional applications, and as a read-intensive database for business intelligence systems.

Apache Cassandra- Introducción

- Web: http://cassandra.apache.org/
- Documentación: http://www.datastax.com/docs/
- Cassandra use cases
  - Digg
  - Netflix
  - Rackspace
  - Twitter
  - …

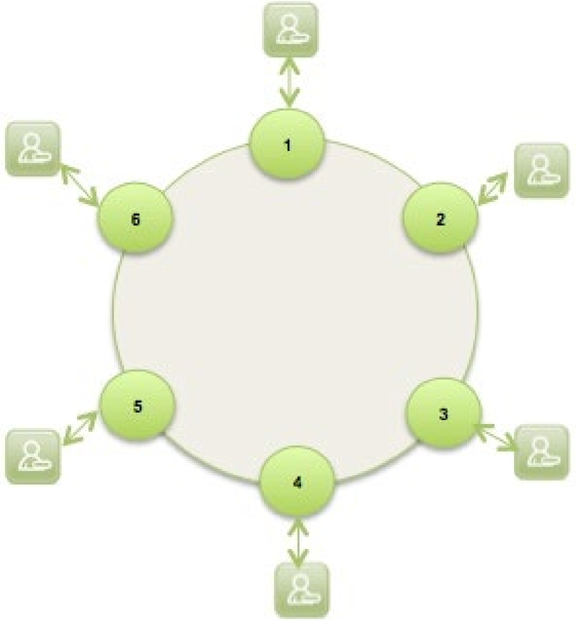
Architecture Overview

- Cassandra was designed with the understanding that system/hardware failures can and do occur
- Peer-to-peer, distributed system
- All nodes the same
- Data partitioned among all nodes in the cluster
- Custom data replication to ensure fault tolerance
- Read/Write-anywhere design



Architecture Overview

- Each node communicates with each other through the Gossip protocol, which exchanges information across the cluster every second
- A commit log is used on each node to capture write activity. Data durability is assured
- Data also written to an in-memory structure (memtable) and then to disk once the memory structure is full (an SStable)



Learning Objectives

- Understand the Cassandra data model
- Introduce cqlsh
- Understand and use the DDL subset of CQL
- Introduce DataGrip
- Understand and use the DML subset of CQL
- ~~Understand basics of data modeling~~ (Challenge)

Learning Objectives

- Understand the Cassandra data model
- Introduce cqlsh
- Understand and use the DDL subset of CQL
- Introduce DataGrip
- Understand and use the DML subset of CQL
- ~~Understand basics of data modeling~~ (Challenge)

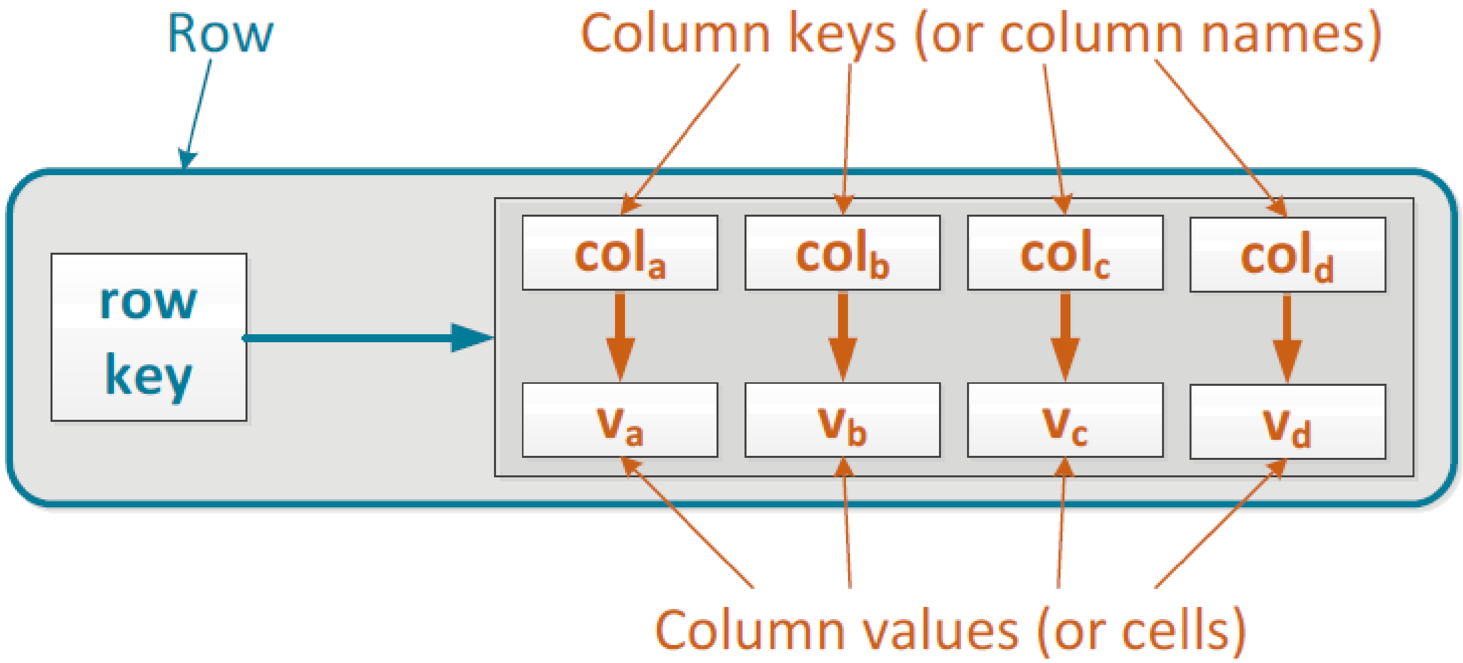
What are the essential constituents of the Cassandra data model?

- The Cassandra data model defines
  1. *Column family* as a way to store and organize data
  2. *Table* as a two-dimensional view of a multi-dimensional *column family*
  3. Operations on tables using the Cassandra Query Language (CQL)
- We cover these three constituents in the order they are listed
  - Understanding *column families* is a prerequisite to understanding *tables*
  - Understanding *tables* is a prerequisite to understanding operations

What are row, row key, column key, and column value?

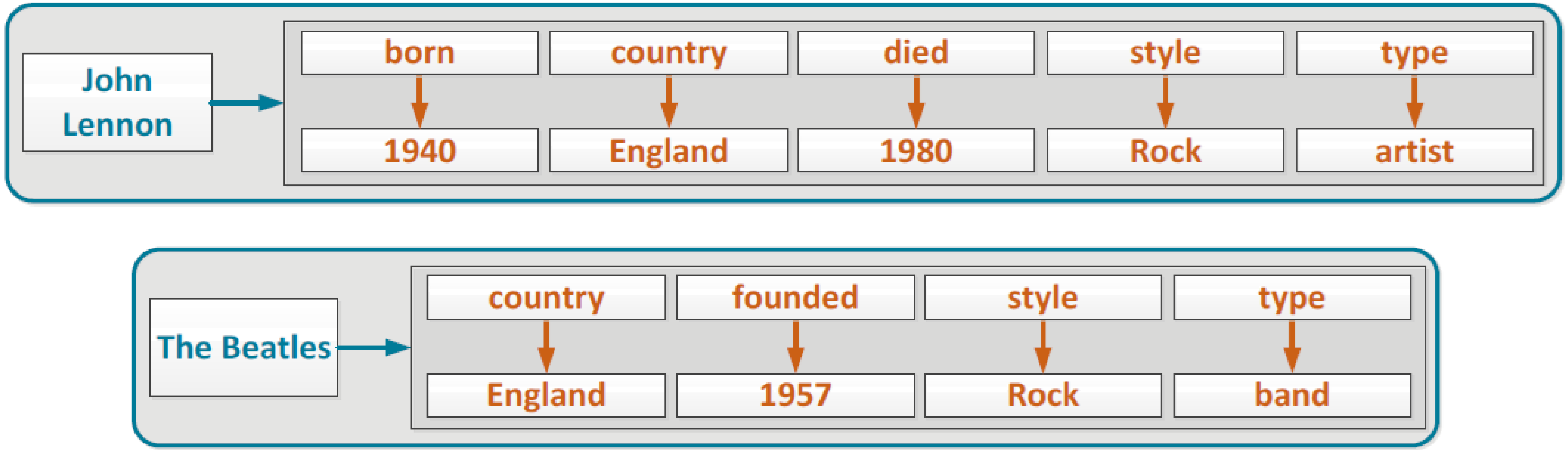
- *Row* is the smallest unit that stores related data in Cassandra
  - Rows – individual rows constitute a *column family*
  - Row key – uniquely identifies a *row* in a *column family*
  - Row – stores pairs of *column keys* and *column values*
  - Column key – uniquely identifies a *column value* in a *row*
  - Column value – stores one value or a *collection* of values

Apache Cassandra – Fila (Row)



What are row, row key, column key, and column value?

- Sample rows that describe an artist and a band
  - *Column keys* are inherently sorted
  - A *row* can be retrieved if its *row key* is known
  - A *column value* can be retrieved if its *row key* and *column key* are known

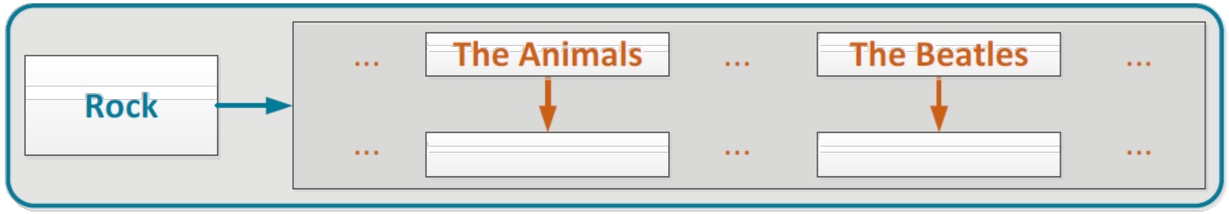


John Lennon → born → 1940; country → England; died → 1980; style → Rock; type → artist

The Beatles → country → England; founded → 1957; style → Rock; type → band

Apache Cassandra – Fila (Row)

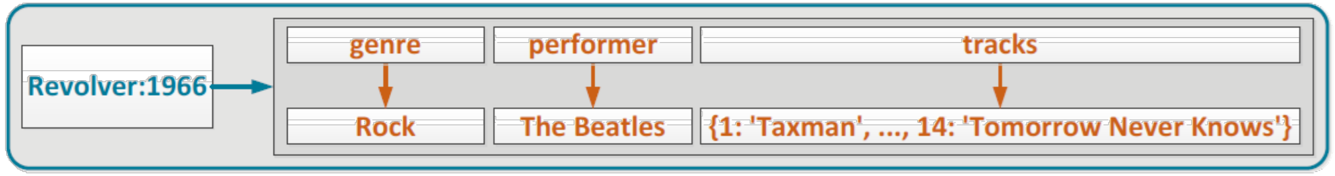
- Rows may be described as "skinny" or "wide"
  - Skinny row – has a fixed, relatively small number of *column keys*
    - Previous examples were skinny rows
  - Wide row – has a relatively large number of *column keys* (hundreds or thousands); this number may increase as new data values are inserted
    - For example, a row that stores all bands of the same style
    - The number of such bands will increase as new bands are formed



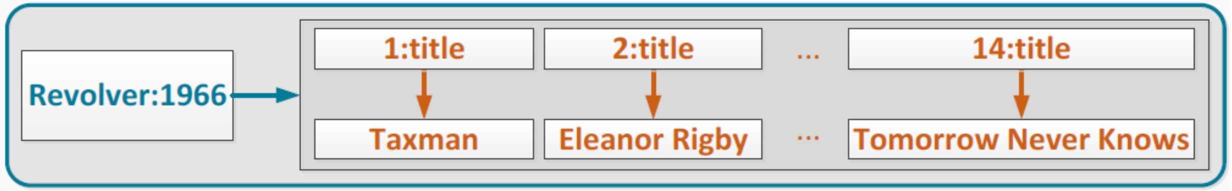
-
  -
    - Note that column values do not exist in this example
      - The column key – in this case a band name – stores all the data desired
      - Could have stored the number of albums, or year founded, etc., as column values

What are composite row key and composite column key?

- Composite row key – multiple components separated by colon
  - ‘Revolver’ and 1966 are the album title and year
  - ‘tracks’ value is a collection (map)

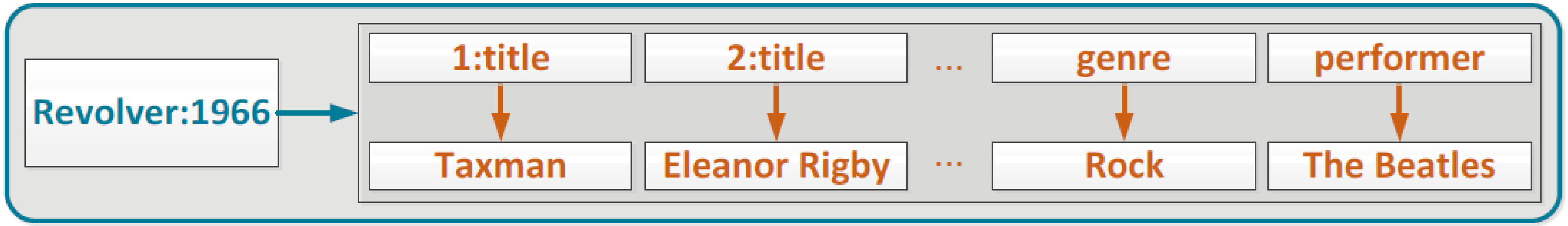


- Composite column key – multiple components separated by colon
  - Composite column keys are sorted by each component
  - 1,2, …, 14 are track numbers; ‘title’ is metadata
    - We could have stored actual title as components of composite column keys: 1:Taxman, 2:Eleanor Rigby, …, 14:Tomorrow Never Knows



Can simple and composite column keys co-exist in the same row?

- Row can contain both simple and composite column keys
  - ‘genre’ and ‘performer’ are simple column keys
  - ‘1:title’, ‘2:title’, … are composite column keys



Revolver:1966 →

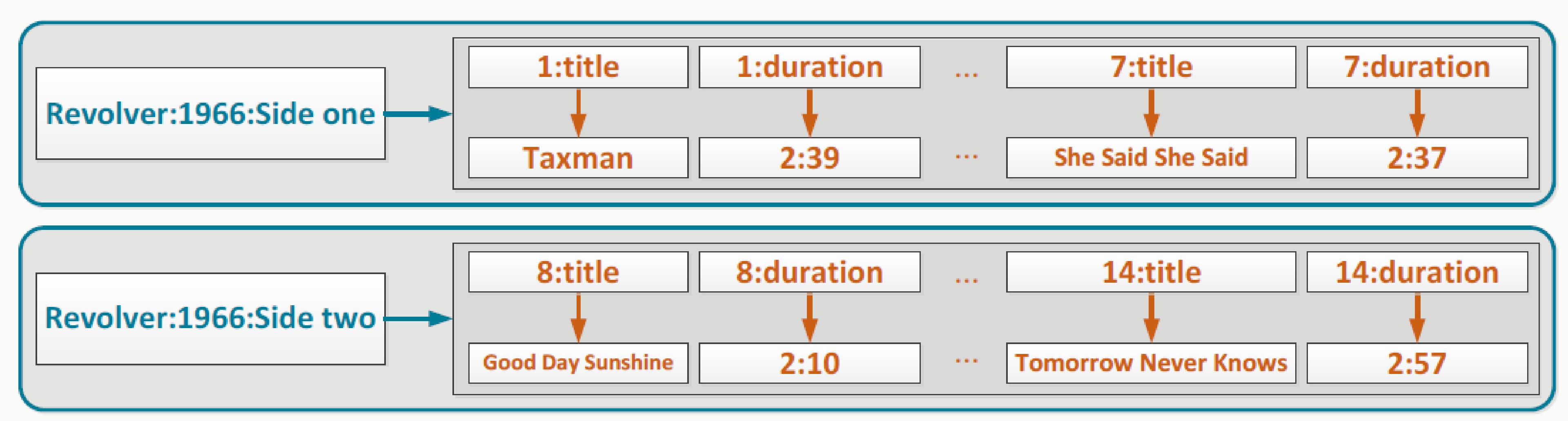
1:title	2:title	…	genre	performer
↓	↓		↓	↓
Taxman	Eleanor Rigby	…	Rock	The Beatles

What components of a row can store useful values?

- Any component of a row can store data or metadata
    - Simple or composite row keys
    - Simple or composite column keys
    - Atomic or set-valued (collection) column values
        - Metadata: ‘Side one’, ‘Side two’, ‘title’, ‘duration’
        - Data: everything else (‘Revolver’, ‘1966’, ‘She Said She Said’, etc.)

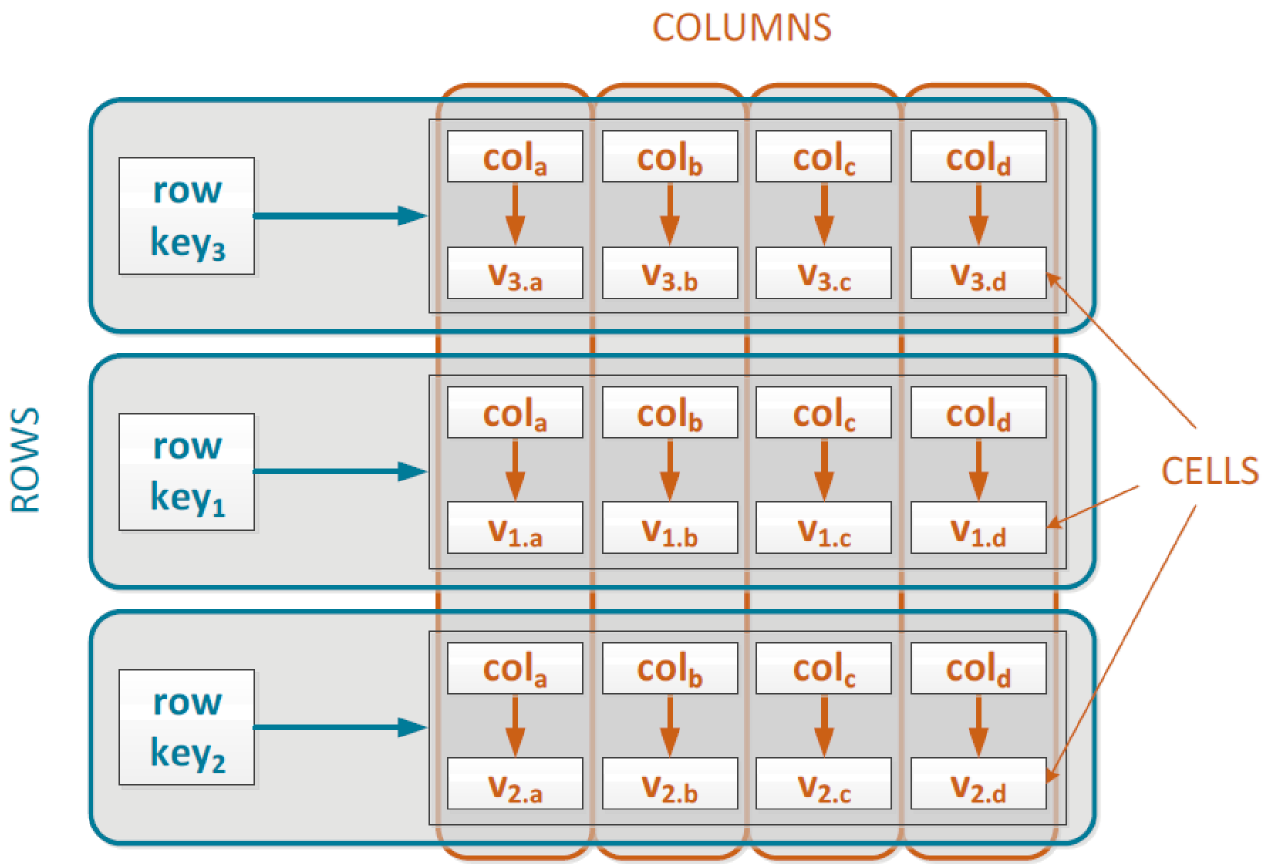
What components of a row can store useful values?

- Metadata: ‘Side one’, ‘Side two’, ‘title’, ‘duration’
- Data: everything else (‘Revolver’, ‘1966’, ‘She Said She Said’, etc.)



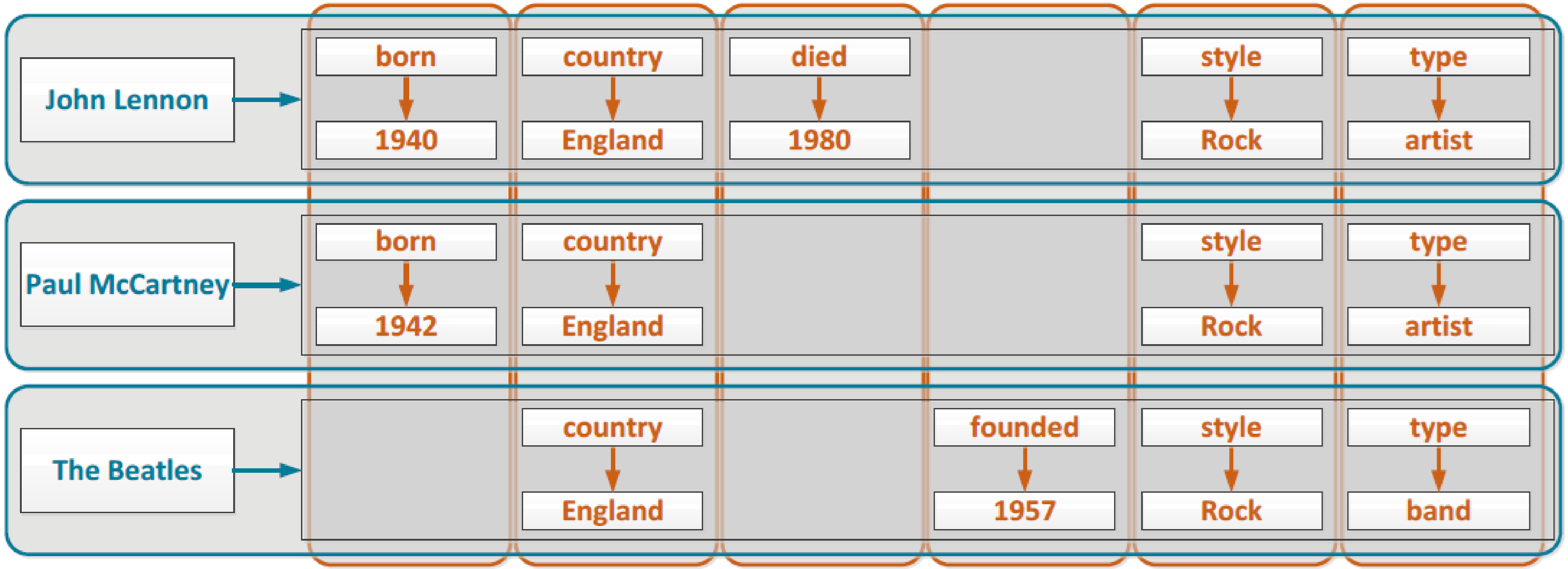
Apache Cassandra – Column Family

- Column family – set of rows with a similar structure


COLUMNS
ROWS
CELLS
row key$_3$
col$_a$ col$_b$ col$_c$ col$_d$
v$_{3.a}$ v$_{3.b}$ v$_{3.c}$ v$_{3.d}$
row key$_1$
col$_a$ col$_b$ col$_c$ col$_d$
v$_{1.a}$ v$_{1.b}$ v$_{1.c}$ v$_{1.d}$
row key$_2$
col$_a$ col$_b$ col$_c$ col$_d$
v$_{2.a}$ v$_{2.b}$ v$_{2.c}$ v$_{2.d}$

Apache Cassandra – Column Family

- Distributed
- Sparse
  - Column family that stores data about artists and bands



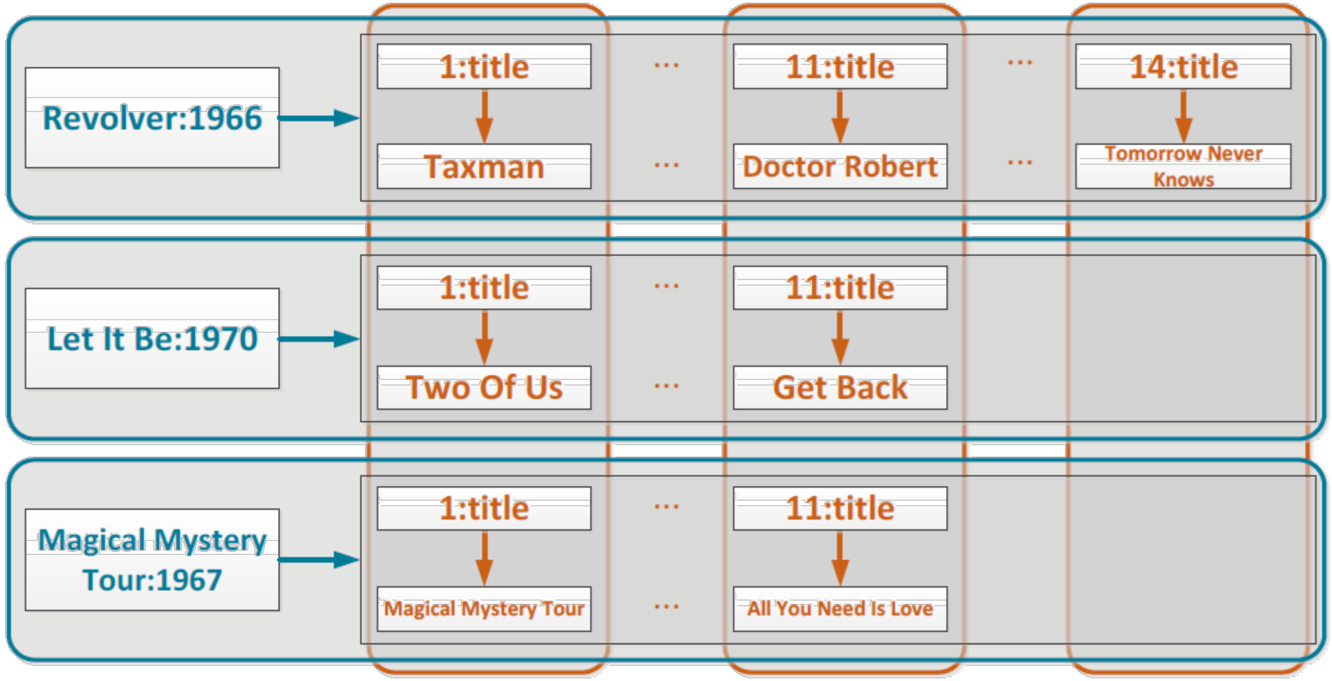
John Lennon → born → 1940; country → England; died → 1980; style → Rock; type → artist

Paul McCartney → born → 1942; country → England; style → Rock; type → artist

The Beatles → country → England; founded → 1957; style → Rock; type → band

Apache Cassandra – Column Family

- Sorted columns
- Multidimensional
  - Column family that stores albums and their tracks



Apache Cassandra – Column Family

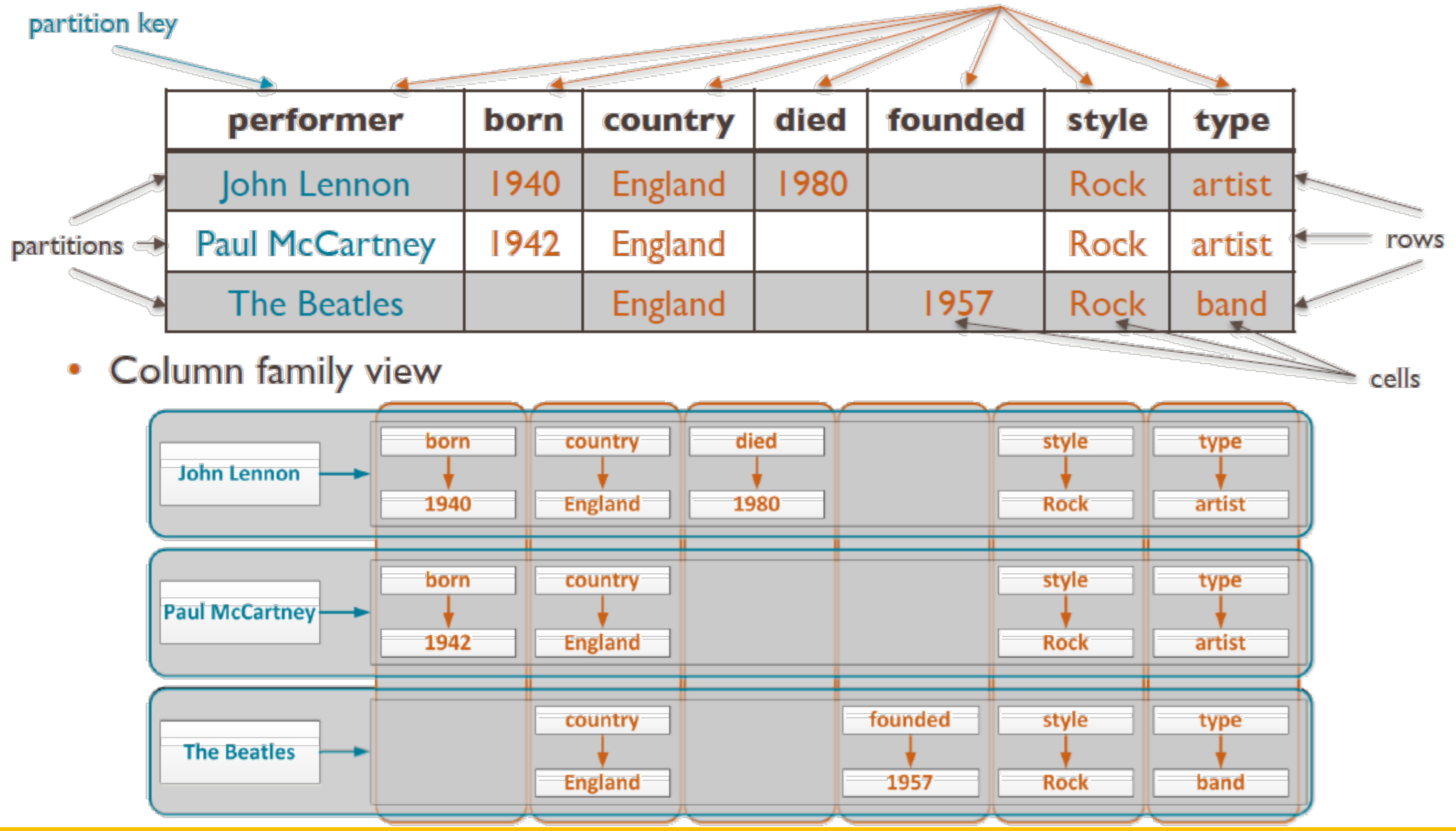
- Size of a column family is only limited to the size of a cluster
  - Linear scalability
  - **Rows are distributed among the nodes in a cluster**
- Column family component size considerations
  - Data from a one row must fit on one node
    - Data from any given row never spans multiple nodes
  - Maximum columns per row is 2 billion
    - In practice – Up to 100 thousand
  - Maximum data size per cell (column value) is 2 GB
    - In practice – Up to 100 MB

Apache Cassandra – CQL Table

- What is a CQL table and how is it related to a column family?
  - **A CQL table is a column family**
    - CQL tables provide two-dimensional views of a column family, which contains potentially multi-dimensional data, due to composite keys and collections
  - CQL table and column family are largely interchangeable terms
    - Not surprising when you recall tables and relations, columns and attributes, rows and tuples in relational databases
  - Supported by declarative language Cassandra Query Language
    - Data Definition Language (DDL), subset of CQL
    - SQL-like syntax, but with somewhat different semantics
    - Convenient for defining and expressing Cassandra database schemas

What are partition, partition key, row, column, and cell?

- Table with single-row partitions



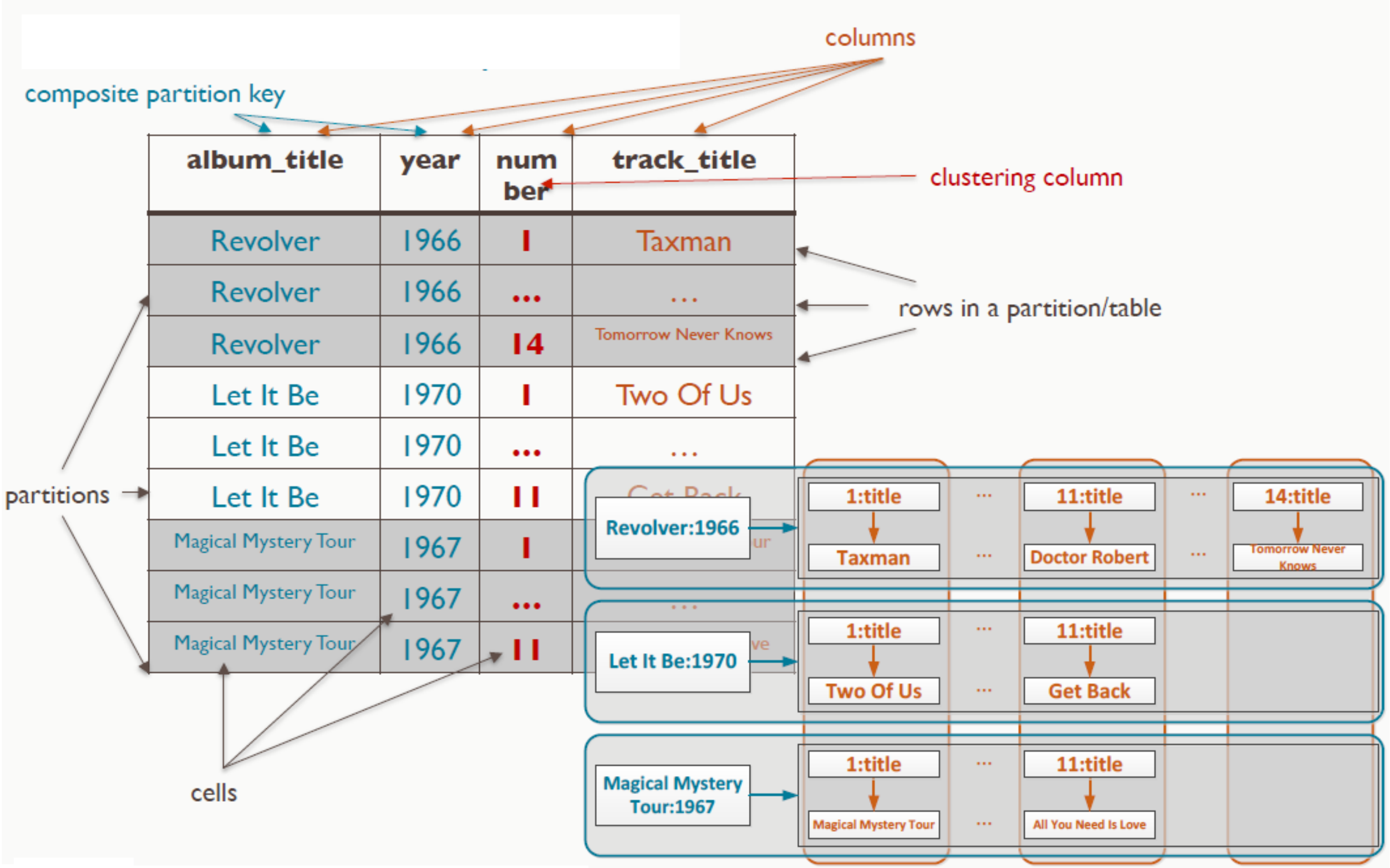
performer	born	country	died	founded	style	type
John Lennon	1940	England	1980		Rock	artist
Paul McCartney	1942	England			Rock	artist
The Beatles		England		1957	Rock	band

- Column family view

John Lennon	born 1940	country England	died 1980		style Rock	type artist
Paul McCartney	born 1942	country England			style Rock	type artist
The Beatles		country England		founded 1957	style Rock	type band

What are composite partition key and clustering column?

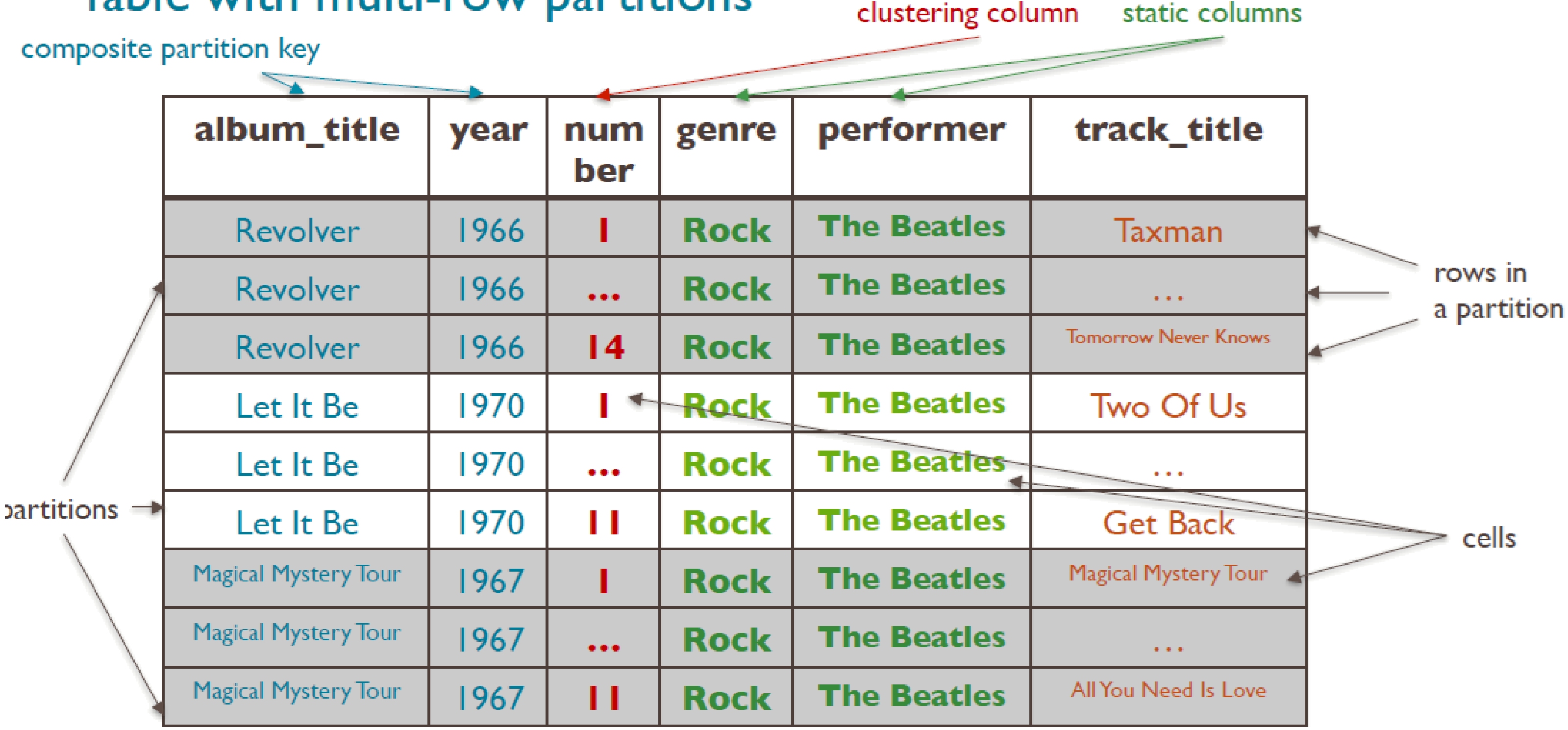
- Table with multi-row partitions



album_title	year	number	track_title
Revolver	1966	1	Taxman
Revolver	1966	...	...
Revolver	1966	14	Tomorrow Never Knows
Let It Be	1970	1	Two Of Us
Let It Be	1970	...	...
Let It Be	1970	11	Get Back
Magical Mystery Tour	1967	1	
Magical Mystery Tour	1967	...	...
Magical Mystery Tour	1967	11	

What are static columns?

- Table with multi-row partitions



clustering column

static columns

composite partition key

album_title	year	number	genre	performer	track_title
Revolver	1966	1	Rock	The Beatles	Taxman
Revolver	1966	...	Rock	The Beatles	...
Revolver	1966	14	Rock	The Beatles	Tomorrow Never Knows
Let It Be	1970	1	Rock	The Beatles	Two Of Us
Let It Be	1970	...	Rock	The Beatles	...
Let It Be	1970	11	Rock	The Beatles	Get Back
Magical Mystery Tour	1967	1	Rock	The Beatles	Magical Mystery Tour
Magical Mystery Tour	1967	...	Rock	The Beatles	...
Magical Mystery Tour	1967	11	Rock	The Beatles	All You Need Is Love

rows in a partition

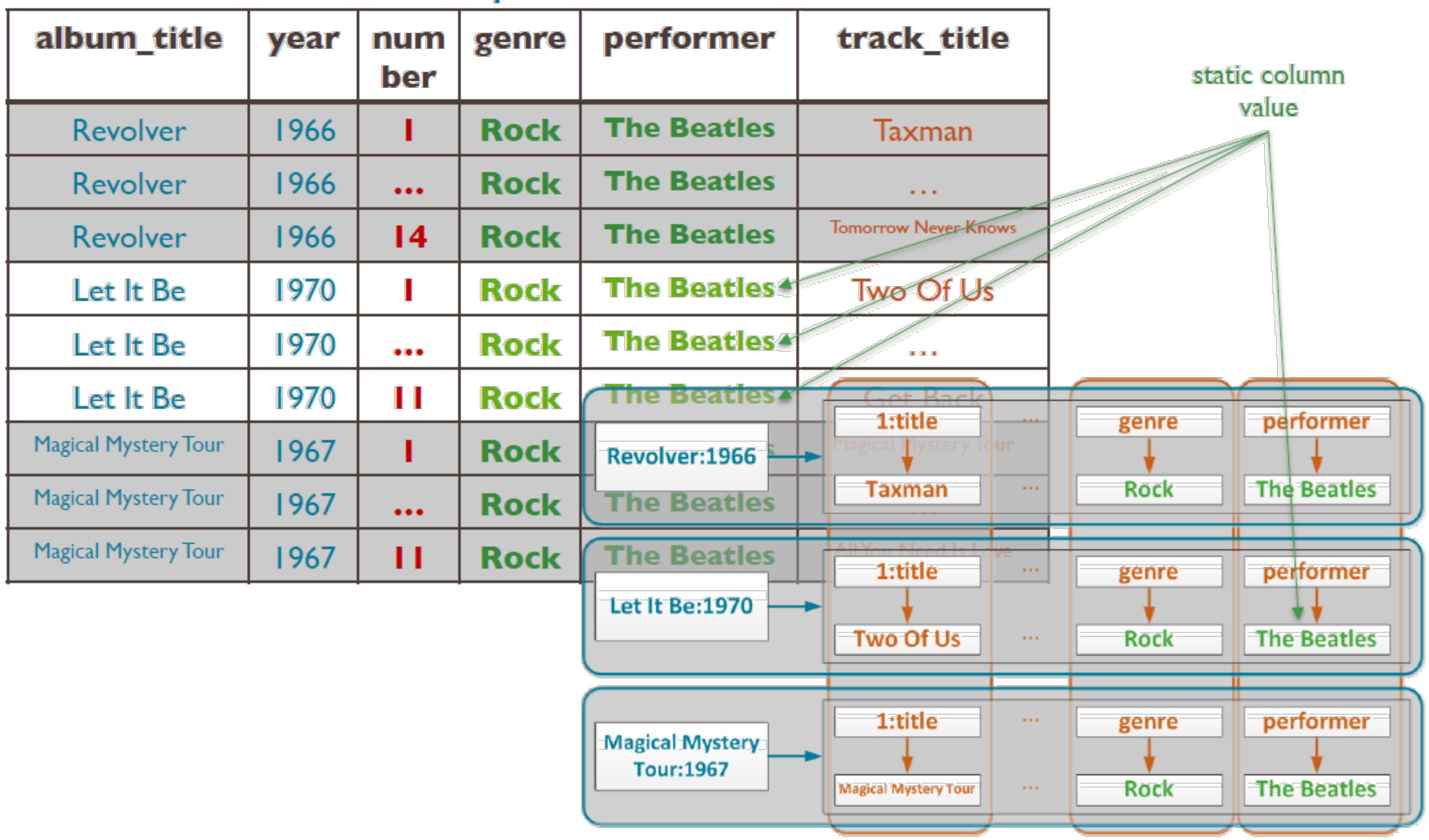
partitions

cells

- Static column values are shared for all rows in a multi-row partition

What are static columns?

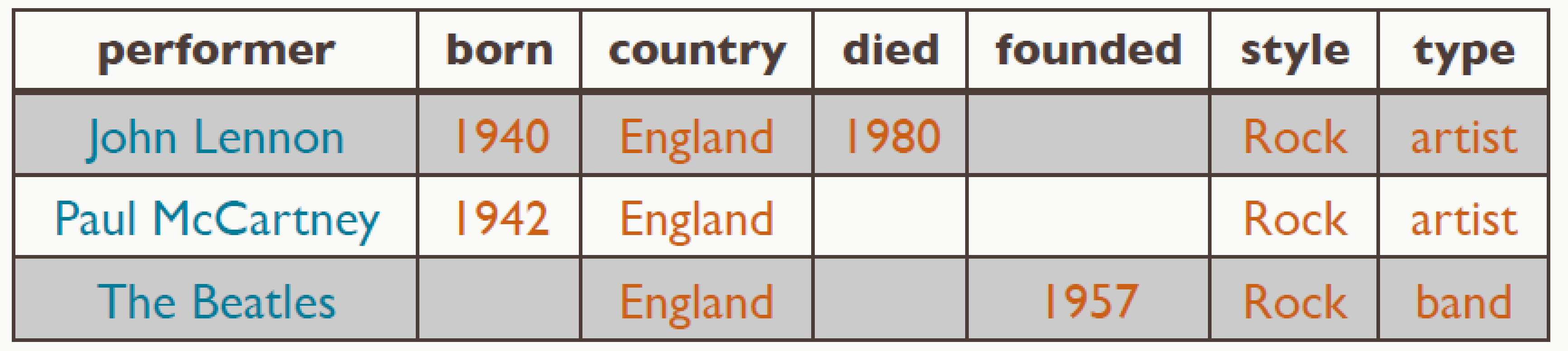
- Table with multi-row partitions



album_title	year	num ber	genre	performer	track_title
Revolver	1966	1	Rock	The Beatles	Taxman
Revolver	1966	...	Rock	The Beatles	...
Revolver	1966	14	Rock	The Beatles	Tomorrow Never Knows
Let It Be	1970	1	Rock	The Beatles	Two Of Us
Let It Be	1970	...	Rock	The Beatles	...
Let It Be	1970	11	Rock	The Beatles	Get Back
Magical Mystery Tour	1967	1	Rock	The Beatles	Magical Mystery Tour
Magical Mystery Tour	1967	...	Rock	The Beatles	...
Magical Mystery Tour	1967	11	Rock	The Beatles	All You Need Is Love

What is a primary key?

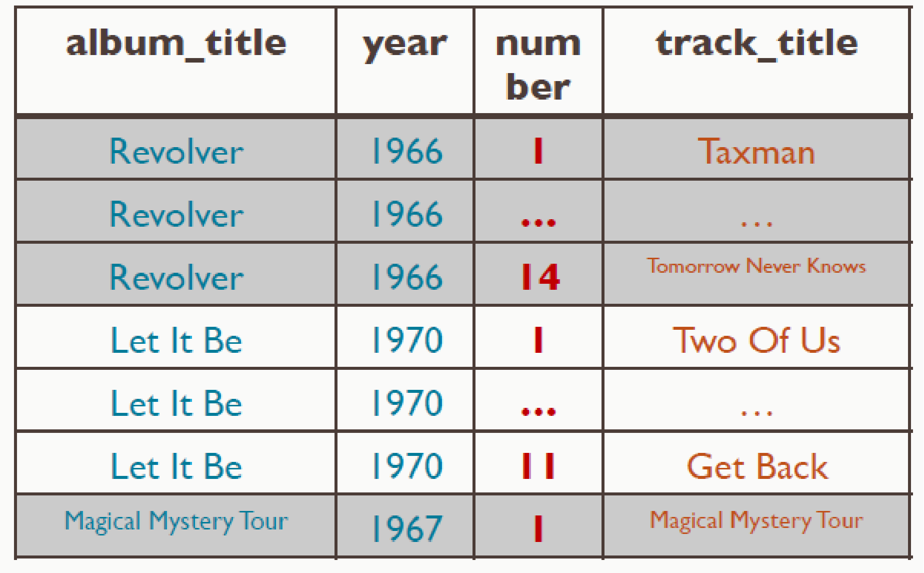
- Primary key uniquely identifies a row in a table
  - Simple or composite partition key and all clustering columns (if present)
  - Primary key
    - performer



performer	born	country	died	founded	style	type
John Lennon	1940	England	1980		Rock	artist
Paul McCartney	1942	England			Rock	artist
The Beatles		England		1957	Rock	band

What is a primary key?

- Primary key uniquely identifies a row in a table
  - Simple or composite partition key and all clustering columns (if present)
  - Primary key (table above)
    - album, year, number
  - Static columns cannot be part



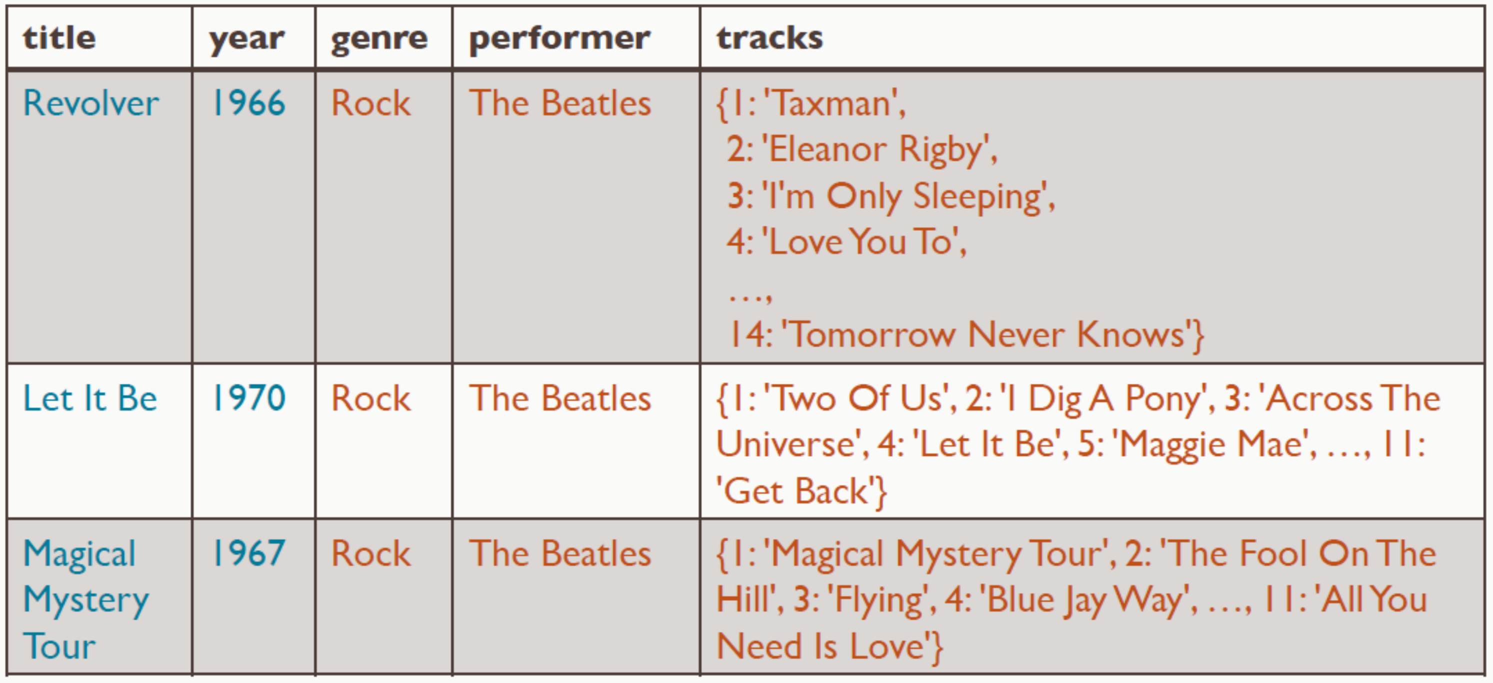
album_title	year	number	track_title
Revolver	1966	1	Taxman
Revolver	1966	...	...
Revolver	1966	14	Tomorrow Never Knows
Let It Be	1970	1	Two Of Us
Let It Be	1970	...	...
Let It Be	1970	11	Get Back
Magical Mystery Tour	1967	1	Magical Mystery Tour

What are collection columns?

- Multiple values can be stored in a column
  - Set – typed collection of unique values (e.g., genres)
    - Ordered by values
    - No duplicates
      - {"Blues", "Jazz", "Rock"}
  - List – typed collection of non-unique values (e.g., artists)
    - Ordered by position
    - Duplicates are allowed
      - ["Lennon", "Lennon", "McCartney"]
  - Map – typed collection of key-value pairs (e.g., tracks)
    - Ordered by keys
    - Unique keys but not values
      - {1:"Taxman", 2:"Eleanor Rigby", 3:"I'm Only Sleeping"}

Apache Cassandra - Map

- Map example
  - Collection column tracks holds a map of album tracks

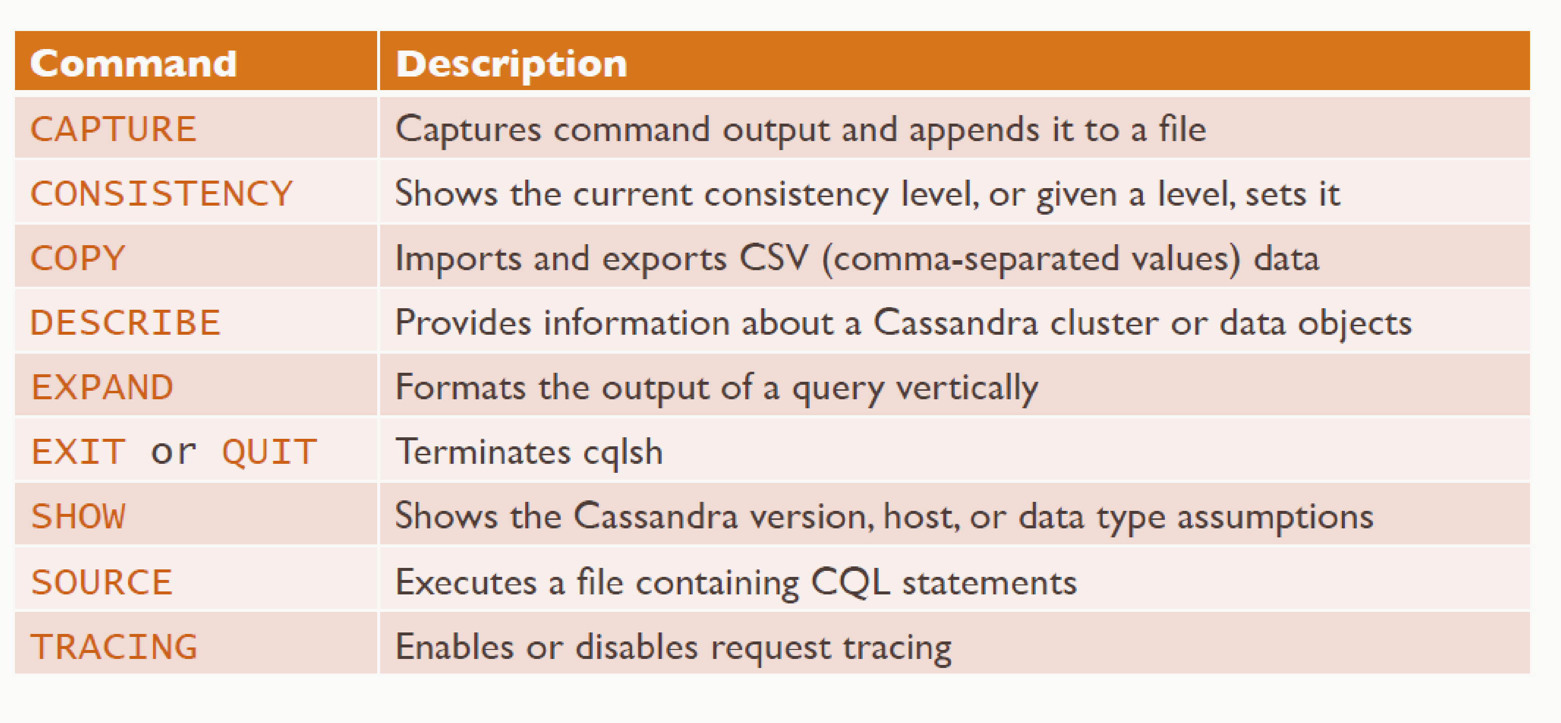


title	year	genre	performer	tracks
Revolver	1966	Rock	The Beatles	{1: 'Taxman', 2: 'Eleanor Rigby', 3: 'I'm Only Sleeping', 4: 'Love You To', …, 14: 'Tomorrow Never Knows'}
Let It Be	1970	Rock	The Beatles	{1: 'Two Of Us', 2: 'I Dig A Pony', 3: 'Across The Universe', 4: 'Let It Be', 5: 'Maggie Mae', …, 11: 'Get Back'}
Magical Mystery Tour	1967	Rock	The Beatles	{1: 'Magical Mystery Tour', 2: 'The Fool On The Hill', 3: 'Flying', 4: 'Blue Jay Way', …, 11: 'All You Need Is Love'}

Learning Objectives

- Understand the Cassandra data model
- Introduce cqlsh
- Understand and use the DDL subset of CQL
- Introduce DataGrip
- Understand and use the DML subset of CQL
- ~~Understand basics of data modeling~~ (Challenge)

cqlsh Shell commands



Command	Description
CAPTURE	Captures command output and appends it to a file
CONSISTENCY	Shows the current consistency level, or given a level, sets it
COPY	Imports and exports CSV (comma-separated values) data
DESCRIBE	Provides information about a Cassandra cluster or data objects
EXPAND	Formats the output of a query vertically
EXIT or QUIT	Terminates cqlsh
SHOW	Shows the Cassandra version, host, or data type assumptions
SOURCE	Executes a file containing CQL statements
TRACING	Enables or disables request tracing

Learning Objectives

- Understand the Cassandra data model
- Introduce cqlsh
- Understand and use the DDL subset of CQL
- Introduce DataGrip
- Understand and use the DML subset of CQL
- ~~Understand basics of data modeling~~ (Challenge)

What is a keyspace or schema?

- Keyspace – a top-level namespace for a CQL table schema
  - Defines the replication strategy for a set of tables
    - Keyspace per application is a good idea
  - Data objects (e.g., tables) belong to a single keyspace
- Replication strategy – the number and pattern by which partitions are copied among nodes in a cluster
  - Two strategies available
    - Simple Strategy (used for prototyping)
    - Network Topology Strategy (production)

How to create, use and drop keyspaces/schemas?

- To create a keyspace

```
CREATE KEYSPACE musicdb
WITH replication = {
'class': 'SimpleStrategy',
'replication_factor' : 3
};
```

- To assign the working default keyspace for a cqlsh session

```
USE musicdb;
```

- To delete a keyspace and all internal data objects

```
DROP KEYSPACE musicdb;
```

What is the syntax of the CREATE TABLE statement?

- Primary key declared inline

```
CREATE TABLE performer (
name VARCHAR PRIMARY KEY,
type VARCHAR,
country VARCHAR,
style VARCHAR,
founded INT,
born INT,
died INT
);
```

What is the syntax of the CREATE TABLE statement?

- Primary key declared in separate clause

```
CREATE TABLE performer (
name VARCHAR,
type VARCHAR,
country VARCHAR,
style VARCHAR,
founded INT,
born INT,
died INT,
PRIMARY KEY (name)
);
```

How are primary key, partition key, and clustering columns defined?

- Simple partition key, no clustering columns

  PRIMARY KEY (partition_key_column)

- Composite partition key, no clustering columns

  PRIMARY KEY (**(** partition_key_col1, …, partition_key_colN **)**)

- Simple partition key and clustering columns

  PRIMARY KEY (partition_key_column,
  clustering_column1, …, clustering_columnM)

- Composite partition key and clustering columns

  PRIMARY KEY ((partition_key_col1, …, partition_key_colN),
  clustering_column1, …, clustering_columnM)

Apache Cassandra

- Example: Can find all performers and albums for a given track title

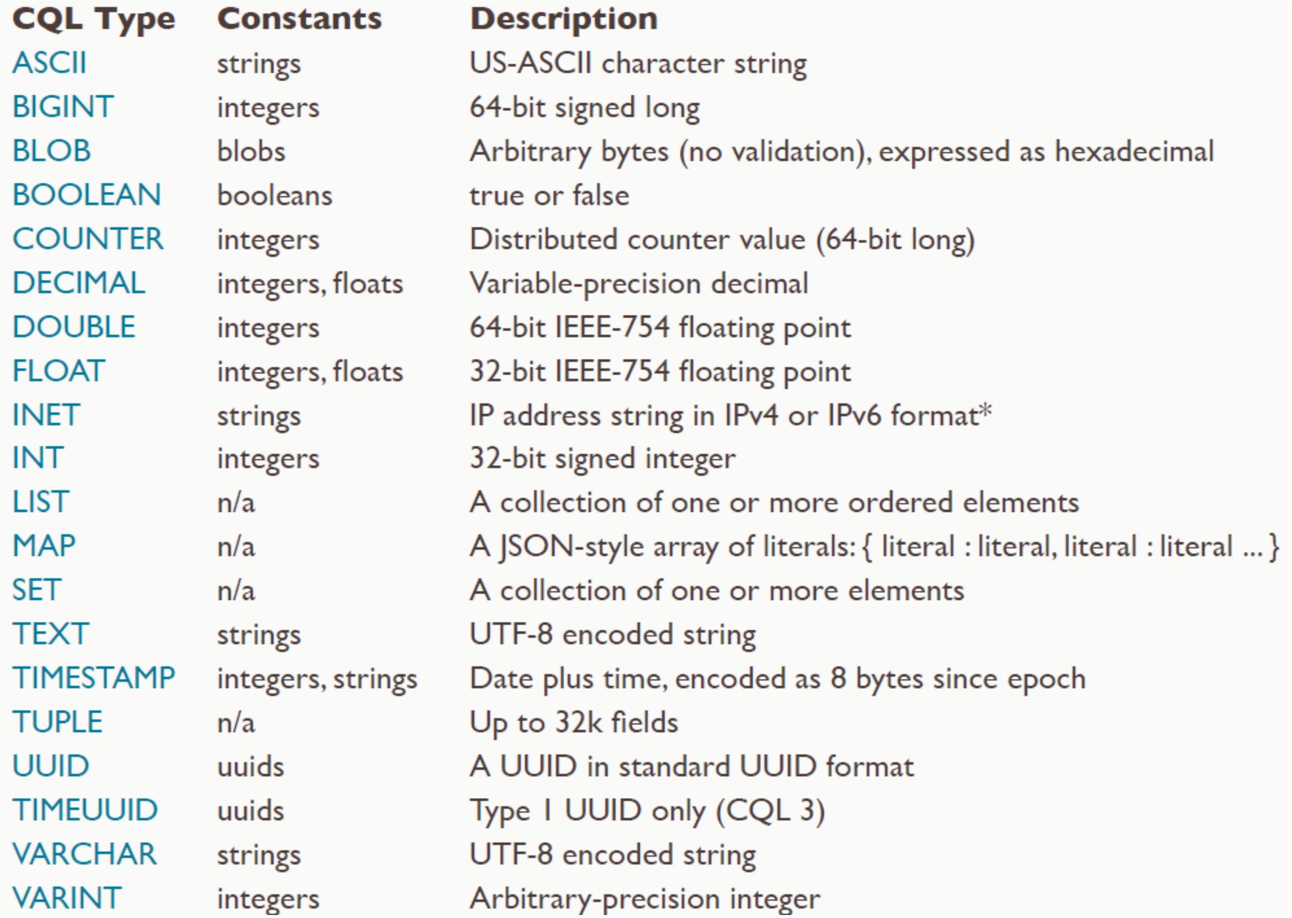
```
CREATE TABLE albums_by_track (
track_title VARCHAR,
performer VARCHAR,
year INT,
album_title VARCHAR,
PRIMARY KEY
(track_title, performer, year, album_title)
);
```

Apache Cassandra

- Example: Can find a performer, genre, and all track numbers and titles for a given album title and year

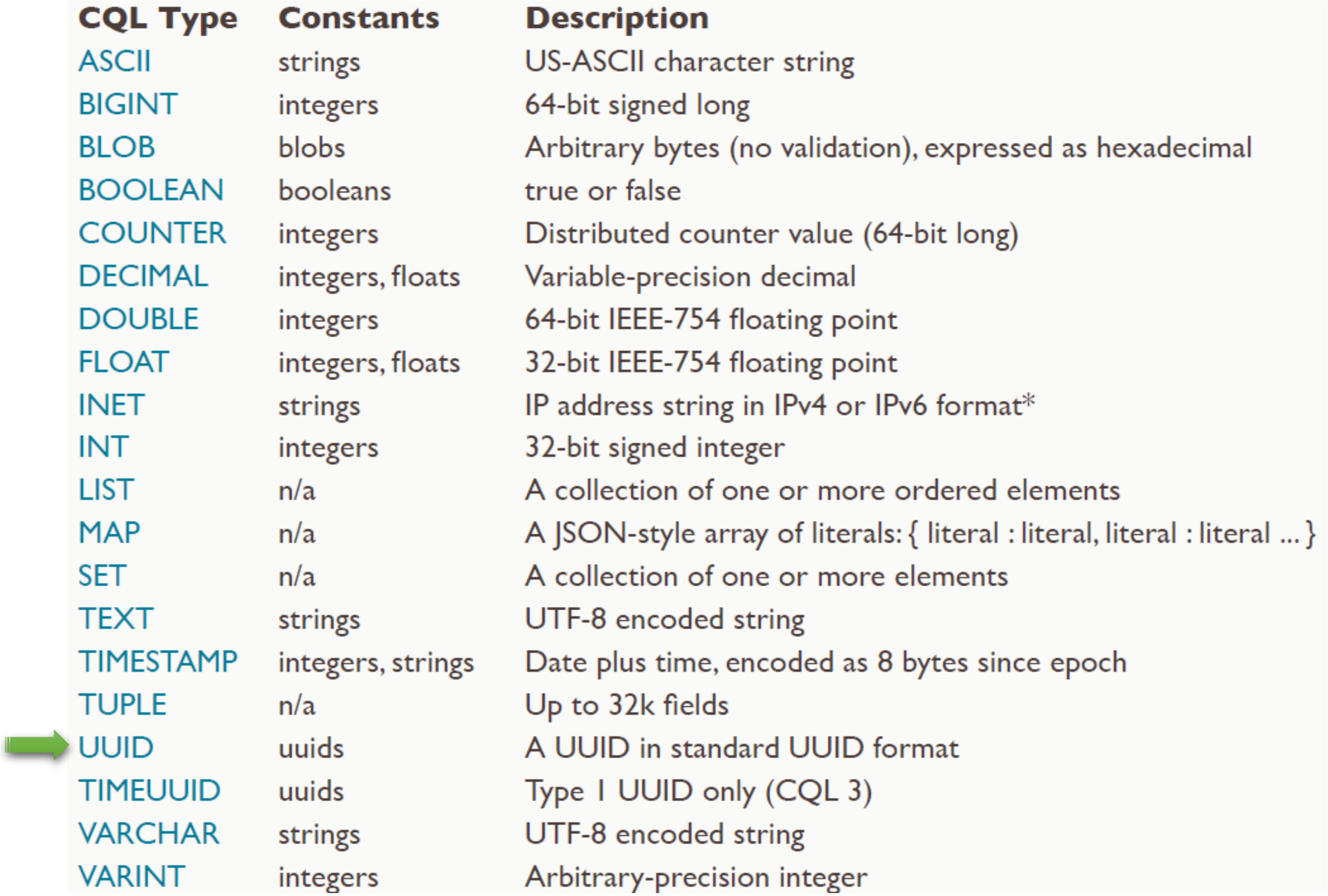
```
CREATE TABLE tracks_by_album (
album_title VARCHAR,
year INT,
performer VARCHAR STATIC,
genre VARCHAR STATIC,
number INT,
track_title VARCHAR,
PRIMARY KEY
((album_title, year), number)
);
```

What CQL data types are available?



CQL Type	Constants	Description
ASCII	strings	US-ASCII character string
BIGINT	integers	64-bit signed long
BLOB	blobs	Arbitrary bytes (no validation), expressed as hexadecimal
BOOLEAN	booleans	true or false
COUNTER	integers	Distributed counter value (64-bit long)
DECIMAL	integers, floats	Variable-precision decimal
DOUBLE	integers	64-bit IEEE-754 floating point
FLOAT	integers, floats	32-bit IEEE-754 floating point
INET	strings	IP address string in IPv4 or IPv6 format*
INT	integers	32-bit signed integer
LIST	n/a	A collection of one or more ordered elements
MAP	n/a	A JSON-style array of literals: { literal : literal, literal : literal ... }
SET	n/a	A collection of one or more elements
TEXT	strings	UTF-8 encoded string
TIMESTAMP	integers, strings	Date plus time, encoded as 8 bytes since epoch
TUPLE	n/a	Up to 32k fields
UUID	uuids	A UUID in standard UUID format
TIMEUUID	uuids	Type 1 UUID only (CQL 3)
VARCHAR	strings	UTF-8 encoded string
VARINT	integers	Arbitrary-precision integer

What CQL data types are available?

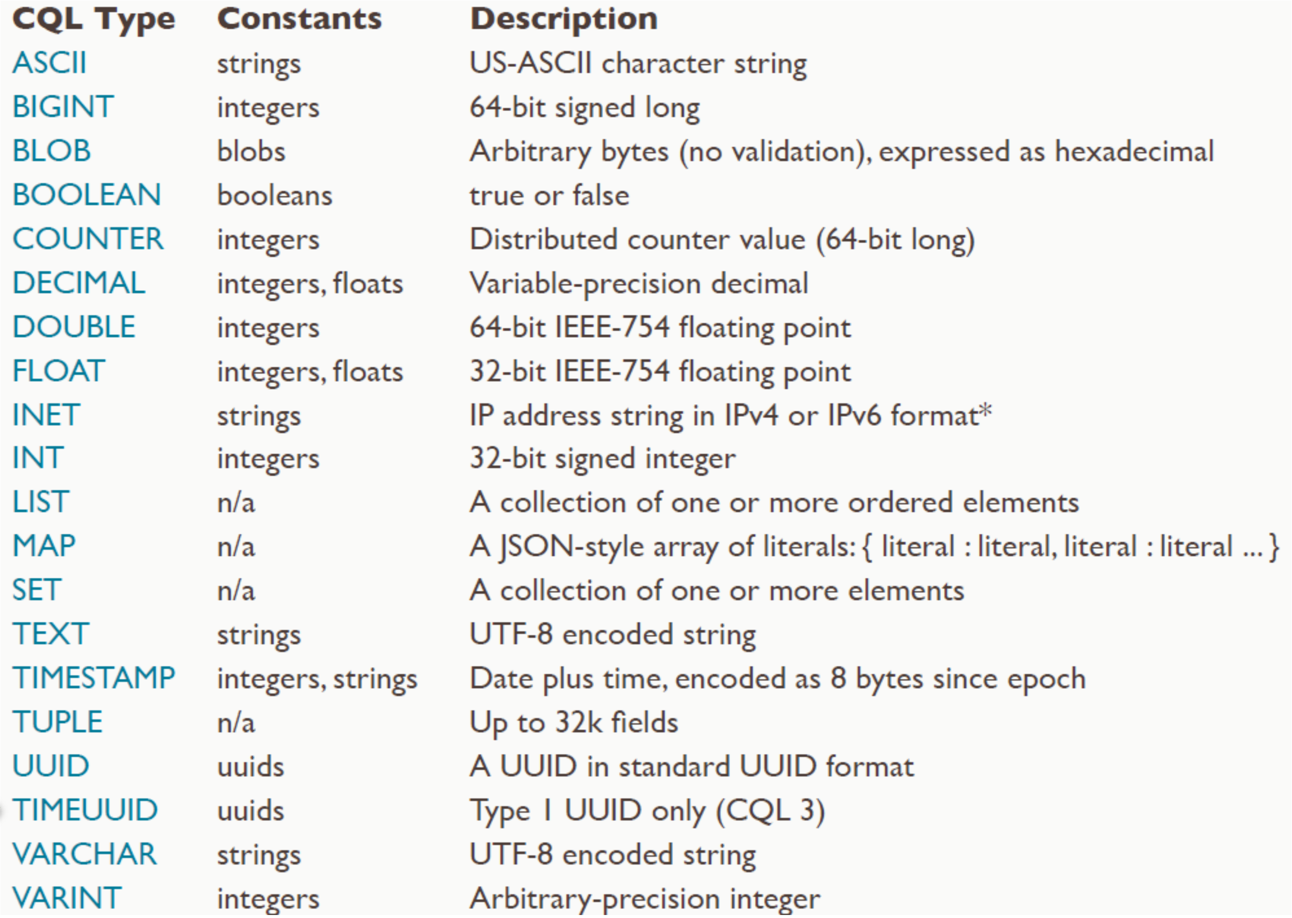


CQL Type	Constants	Description
ASCII	strings	US-ASCII character string
BIGINT	integers	64-bit signed long
BLOB	blobs	Arbitrary bytes (no validation), expressed as hexadecimal
BOOLEAN	booleans	true or false
COUNTER	integers	Distributed counter value (64-bit long)
DECIMAL	integers, floats	Variable-precision decimal
DOUBLE	integers	64-bit IEEE-754 floating point
FLOAT	integers, floats	32-bit IEEE-754 floating point
INET	strings	IP address string in IPv4 or IPv6 format*
INT	integers	32-bit signed integer
LIST	n/a	A collection of one or more ordered elements
MAP	n/a	A JSON-style array of literals: { literal : literal, literal : literal ... }
SET	n/a	A collection of one or more elements
TEXT	strings	UTF-8 encoded string
TIMESTAMP	integers, strings	Date plus time, encoded as 8 bytes since epoch
TUPLE	n/a	Up to 32k fields
UUID	uuids	A UUID in standard UUID format
TIMEUUID	uuids	Type 1 UUID only (CQL 3)
VARCHAR	strings	UTF-8 encoded string
VARINT	integers	Arbitrary-precision integer

Apache Cassandra – UUID y TIMEUUID

- Ids
  - UUID and TIMEUUID are universally unique identifiers
    - Generated programmatically
  - UUID
    - Format
      - hex{8}-hex{4}-hex{4}-hex{4}-hex{12}
      - 52b11d6d-16e2-4ee2-b2a9-5ef1e9589328
    - Used to assign conflict-free (unique) identifiers to data objects
    - Numeric range so vast that duplication is statistically all but imposible
    - CQL function **uuid()** generates a new UUID
  - UUID data type supports Version 4 UUIDs
    - Randomly generated sequence of 32 hex digits separated by dashes
    - 52b11d6d-16e2-4ee2-b2a9-5ef1e9589328

What CQL data types are available?



CQL Type	Constants	Description
ASCII	strings	US-ASCII character string
BIGINT	integers	64-bit signed long
BLOB	blobs	Arbitrary bytes (no validation), expressed as hexadecimal
BOOLEAN	booleans	true or false
COUNTER	integers	Distributed counter value (64-bit long)
DECIMAL	integers, floats	Variable-precision decimal
DOUBLE	integers	64-bit IEEE-754 floating point
FLOAT	integers, floats	32-bit IEEE-754 floating point
INET	strings	IP address string in IPv4 or IPv6 format*
INT	integers	32-bit signed integer
LIST	n/a	A collection of one or more ordered elements
MAP	n/a	A JSON-style array of literals: { literal : literal, literal : literal ... }
SET	n/a	A collection of one or more elements
TEXT	strings	UTF-8 encoded string
TIMESTAMP	integers, strings	Date plus time, encoded as 8 bytes since epoch
TUPLE	n/a	Up to 32k fields
UUID	uuids	A UUID in standard UUID format
➡ TIMEUUID	uuids	Type 1 UUID only (CQL 3)
VARCHAR	strings	UTF-8 encoded string
VARINT	integers	Arbitrary-precision integer

Apache Cassandra – TIMEUUID

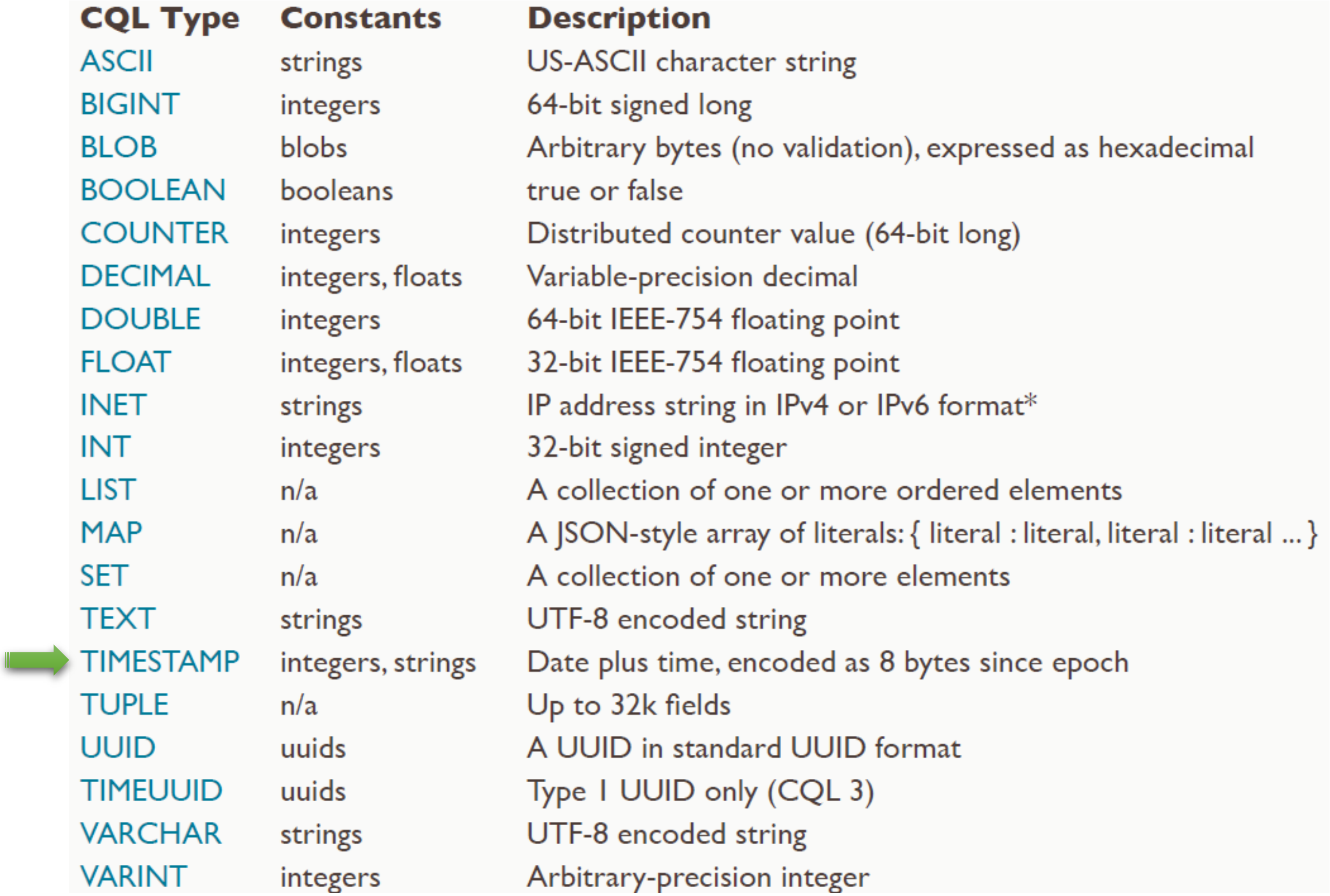
- Time Ids
  - TIMEUUID data type supports Version 1 UUIDs
    - Embeds a time value within a UUID
    - Generated using time (60 bits), a clock sequence number (14 bits), and MAC address (48 bits)
      - 1be43390-9fe4-11e3-8d05-425861b86ab6
    - CQL function **now()** generates a new TIMEUUID
  - Time can be extracted from TIMEUUID
    - CQL function dateOf() extracts the embedded timestamp as a date
  - TIMEUUID values in clustering columns or in column names are ordered based on time
    - DESC order on TIMEUUID lists most recent data first

Apache Cassandra – UUID y TIMEUUID

- Ids, Example
  - Users are identified by UUID
  - User activities (i.e., rating a track) are identified by TIMEUUID
    - A user may rate the same track multiple times
    - Activities are ordered by the time component of TIMEUUID

```
CREATE TABLE track_ratings_by_user (
user UUID,
activity TIMEUUID,
rating INT,
album_title VARCHAR,
album_year INT,
track_title VARCHAR,
PRIMARY KEY (user, activity)
) WITH CLUSTERING ORDER BY (activity DESC);
```

What CQL data types are available?

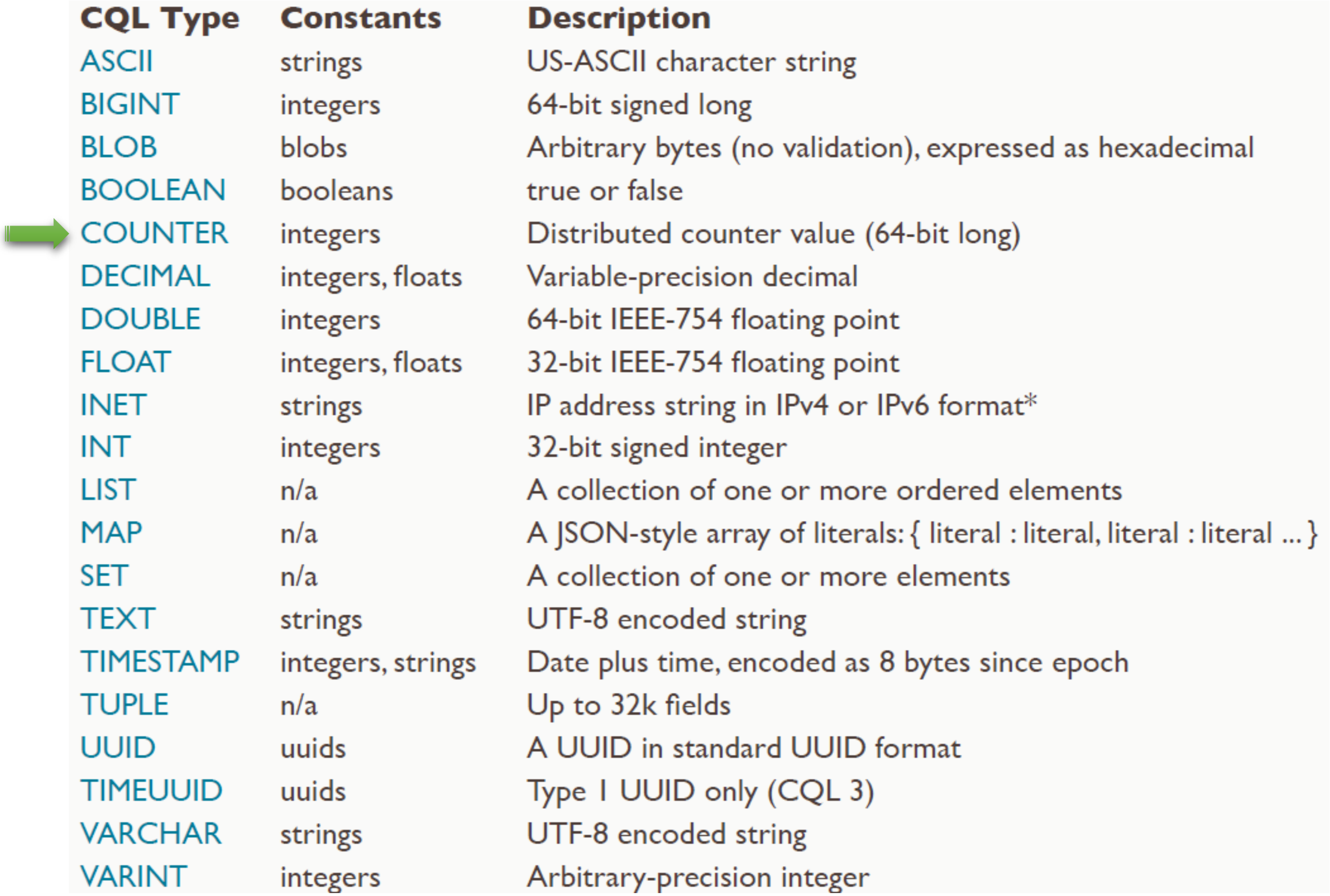


CQL Type	Constants	Description
ASCII	strings	US-ASCII character string
BIGINT	integers	64-bit signed long
BLOB	blobs	Arbitrary bytes (no validation), expressed as hexadecimal
BOOLEAN	booleans	true or false
COUNTER	integers	Distributed counter value (64-bit long)
DECIMAL	integers, floats	Variable-precision decimal
DOUBLE	integers	64-bit IEEE-754 floating point
FLOAT	integers, floats	32-bit IEEE-754 floating point
INET	strings	IP address string in IPv4 or IPv6 format*
INT	integers	32-bit signed integer
LIST	n/a	A collection of one or more ordered elements
MAP	n/a	A JSON-style array of literals: { literal : literal, literal : literal ... }
SET	n/a	A collection of one or more elements
TEXT	strings	UTF-8 encoded string
TIMESTAMP	integers, strings	Date plus time, encoded as 8 bytes since epoch
TUPLE	n/a	Up to 32k fields
UUID	uuids	A UUID in standard UUID format
TIMEUUID	uuids	Type 1 UUID only (CQL 3)
VARCHAR	strings	UTF-8 encoded string
VARINT	integers	Arbitrary-precision integer

Apache Cassandra - TIMESTAMP

- TIMESTAMP holds date and time
  - 64-bit integer representing a number of milliseconds since January 1 1970 at 00:00:00 GMT
  - Entered as
    - 64-bit integer
    - String literal in the ISO 8601 format
      - 1979-12-18 08:12:51-0400
      - 2014-02-27
      - Other variations are allowed
  - Displayed in cqlsh as
    - yyyy-mm-dd HH:mm:ssZ

What CQL data types are available?



CQL Type	Constants	Description
ASCII	strings	US-ASCII character string
BIGINT	integers	64-bit signed long
BLOB	blobs	Arbitrary bytes (no validation), expressed as hexadecimal
BOOLEAN	booleans	true or false
COUNTER	integers	Distributed counter value (64-bit long)
DECIMAL	integers, floats	Variable-precision decimal
DOUBLE	integers	64-bit IEEE-754 floating point
FLOAT	integers, floats	32-bit IEEE-754 floating point
INET	strings	IP address string in IPv4 or IPv6 format*
INT	integers	32-bit signed integer
LIST	n/a	A collection of one or more ordered elements
MAP	n/a	A JSON-style array of literals: { literal : literal, literal : literal ... }
SET	n/a	A collection of one or more elements
TEXT	strings	UTF-8 encoded string
TIMESTAMP	integers, strings	Date plus time, encoded as 8 bytes since epoch
TUPLE	n/a	Up to 32k fields
UUID	uuids	A UUID in standard UUID format
TIMEUUID	uuids	Type 1 UUID only (CQL 3)
VARCHAR	strings	UTF-8 encoded string
VARINT	integers	Arbitrary-precision integer

Apache Cassandra - COUNTER

- Cassandra supports distributed counters
  - Useful for tracking a count
  - Counter column stores a number that can only be updated
    - Incremented or decremented
    - Cannot assign an initial value to a counter (initial value is 0)
  - Counter column **cannot be part of a primary key**
  - If a table has a counter column, **all non-counter columns must be part of a primary key**

Apache Cassandra - COUNTER

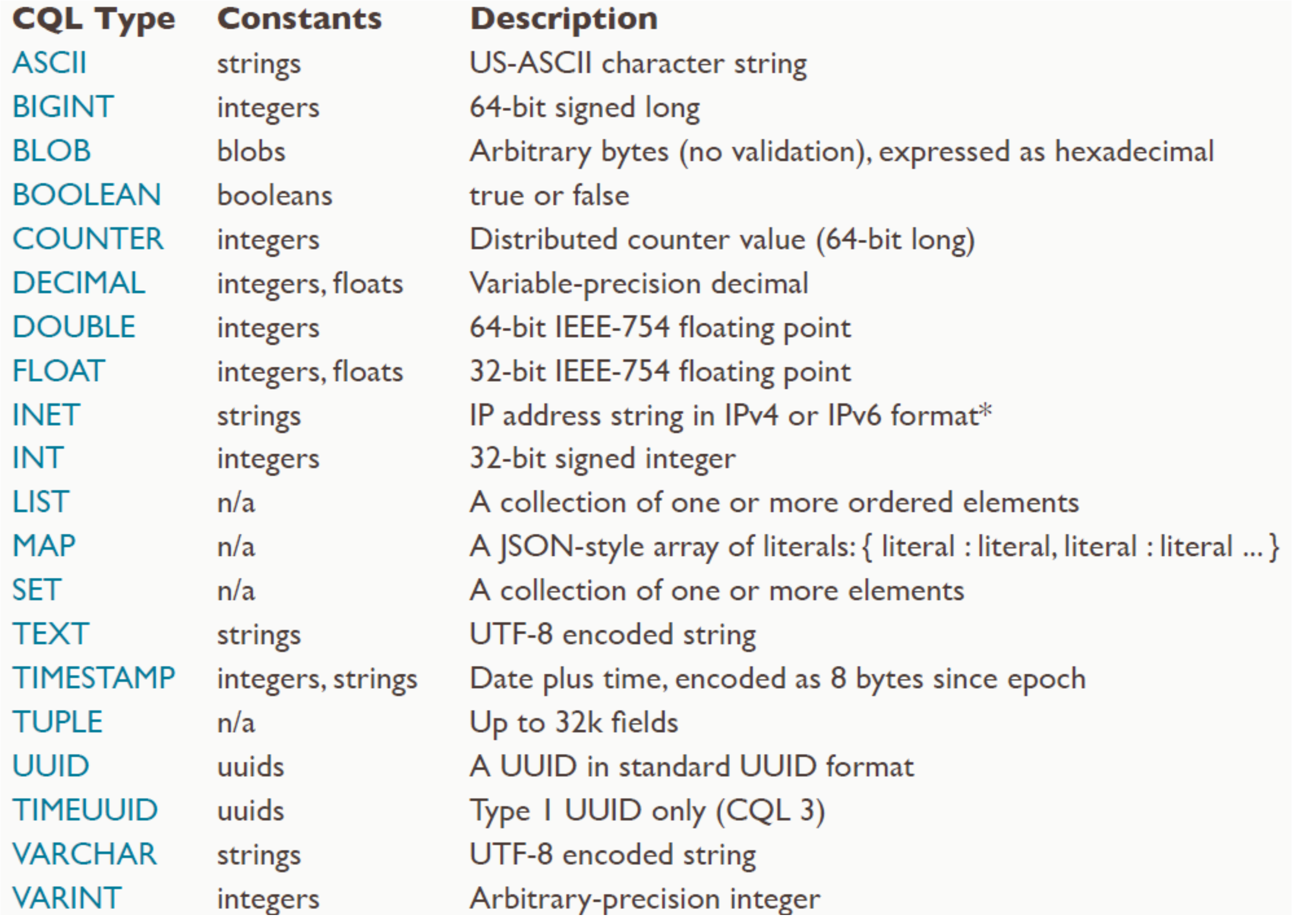
- COUNTER example:

```
CREATE TABLE ratings_by_track (
album_title VARCHAR,
album_year INT,
track_title VARCHAR,
num_ratings COUNTER,
sum_ratings COUNTER,
PRIMARY KEY (album_title, album_year, track_title)
);
```

What are special properties of the COUNTER data type?

- Performance considerations
  - **Read** is **as efficient** as for **non-counter columns**
  - **Update** is fast but **slightly slower** than an update for **non-counter columns**
    - A read is required before a write can be performed
- Accuracy considerations
  - If a counter update is timed out, a client application cannot simply retry a “failed” counter update as the timed-out update may have been persisted
    - Counter update is not an idempotent operation
    - Running an increment twice is not the same as running it once

What CQL data types are available?



CQL Type	Constants	Description
ASCII	strings	US-ASCII character string
BIGINT	integers	64-bit signed long
BLOB	blobs	Arbitrary bytes (no validation), expressed as hexadecimal
BOOLEAN	booleans	true or false
COUNTER	integers	Distributed counter value (64-bit long)
DECIMAL	integers, floats	Variable-precision decimal
DOUBLE	integers	64-bit IEEE-754 floating point
FLOAT	integers, floats	32-bit IEEE-754 floating point
INET	strings	IP address string in IPv4 or IPv6 format*
INT	integers	32-bit signed integer
➡ LIST	n/a	A collection of one or more ordered elements
➡ MAP	n/a	A JSON-style array of literals: { literal : literal, literal : literal ... }
➡ SET	n/a	A collection of one or more elements
TEXT	strings	UTF-8 encoded string
TIMESTAMP	integers, strings	Date plus time, encoded as 8 bytes since epoch
TUPLE	n/a	Up to 32k fields
UUID	uuids	A UUID in standard UUID format
TIMEUUID	uuids	Type 1 UUID only (CQL 3)
VARCHAR	strings	UTF-8 encoded string
VARINT	integers	Arbitrary-precision integer

Apache Cassandra – Collection Columns

- Collection columns are multi-valued columns
  - Designed to store discrete sets of data (e.g., tags for a blog post)
    - A collection is retrieved in its entirety
  - 64,000 - maximum number of elements in a collection
    - In practice – dozens or hundreds
  - 64 KB - maximum size of each collection element
    - In practice – much smaller
  - Collection columns
    - cannot be part of a primary key
    - cannot be part of a partition key
    - cannot be used as a clustering column
    - cannot nest inside of another collection

Apache Cassandra – Collection Columns

- Set – typed collection of unique values
    - keywords SET<VARCHAR>
        - Ordered by values
        - No duplicates
- List – typed collection of non-unique values
    - songwriters LIST<VARCHAR>
        - Ordered by position
        - Duplicates are allowed
- Map – typed collection of key-value pairs
    - tracks MAP<INT,VARCHAR>
        - Ordered by keys
        - Unique keys but not values

Apache Cassandra – User-defined types

- User-defined types group related fields of information
  - Represents related data in a single table, instead of multiple, separate tables
  - Uses any data type, including collections and other user-defined types
  - Reserved words cannot be used as a name for a user-defined type
    - byte
    - smallint
    - complex
    - enum
    - date
    - interval
    - macaddr
    - bitstring

Apache Cassandra - UDF

```
CREATE TYPE track (
album_title VARCHAR,
album_year INT,
track_title VARCHAR,
);
```

Apache Cassandra - UDF

- Table columns can be user-defined types
  - Requires the use of the **frozen keyword** in C* 2.1
  - A user-defined type can be used as a data type for a collection

```
CREATE TABLE musicdb.track_ratings_by_user (
user UUID,
activity TIMEUUID,
rating INT,
song frozen <track>, -- To force the update of the full record
PRIMARY KEY (user, activity)
) WITH CLUSTERING ORDER BY (activity DESC);
```

Apache Cassandra – ALTER TYPE

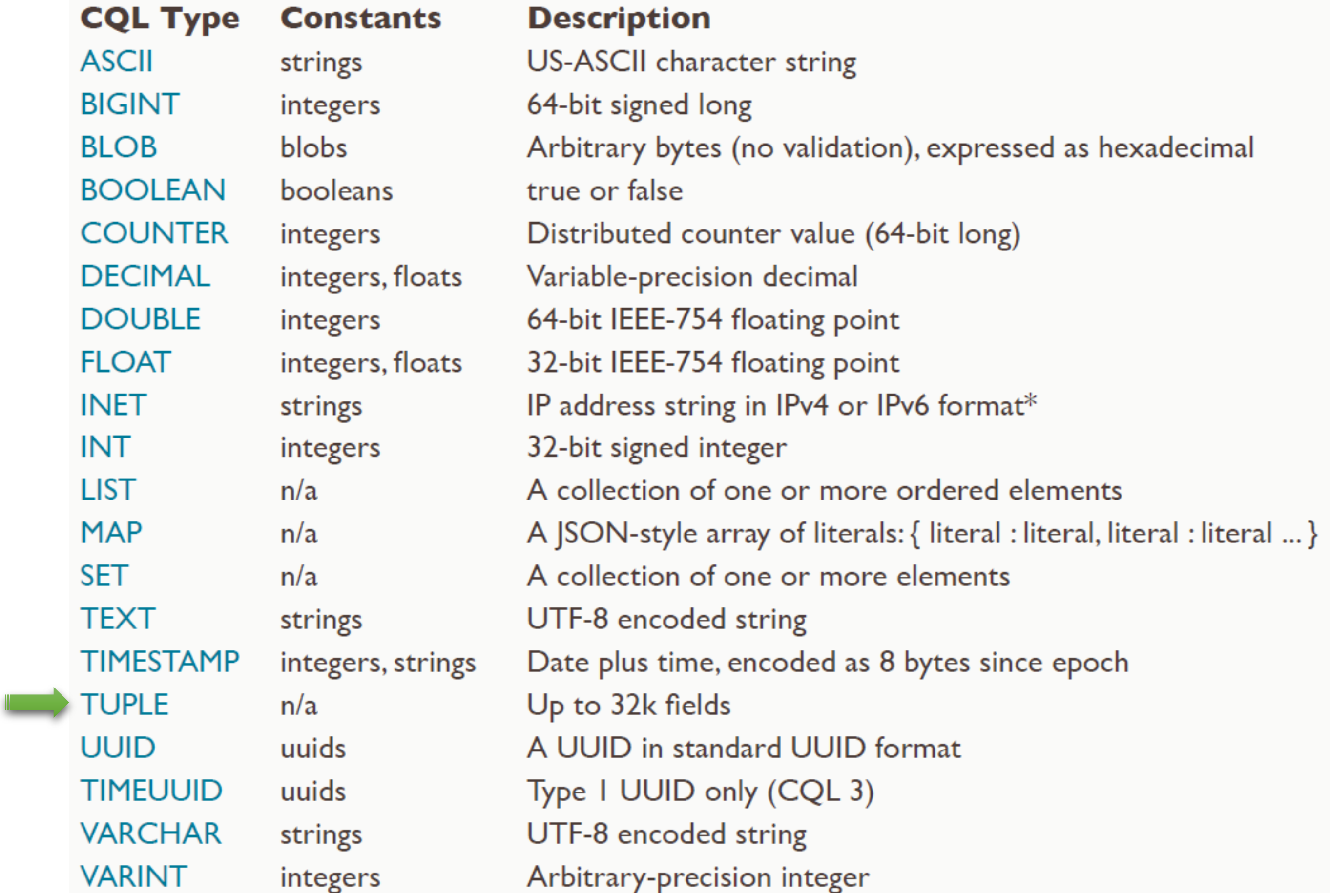
- ALTER TYPE can change a user-defined type
  - Change the type of a field
    - ALTER TYPE track ALTER album_title TYPE BLOB;
      - Types must be compatible
  - Add a field to a type
    - ALTER TYPE track ADD track_number INT;
  - Rename a field of a type
    - ALTER TYPE track RENAME album_year TO year;
  - Rename a user-defined type
    - ALTER TYPE track RENAME TO song;

Apache Cassandra – DROP TYPE

- DROP TYPE removes a user-defined type
  - Cannot drop a user-defined type that is in use by a table or another type

```
DROP TYPE track;
```

What CQL data types are available?



CQL Type	Constants	Description
ASCII	strings	US-ASCII character string
BIGINT	integers	64-bit signed long
BLOB	blobs	Arbitrary bytes (no validation), expressed as hexadecimal
BOOLEAN	booleans	true or false
COUNTER	integers	Distributed counter value (64-bit long)
DECIMAL	integers, floats	Variable-precision decimal
DOUBLE	integers	64-bit IEEE-754 floating point
FLOAT	integers, floats	32-bit IEEE-754 floating point
INET	strings	IP address string in IPv4 or IPv6 format*
INT	integers	32-bit signed integer
LIST	n/a	A collection of one or more ordered elements
MAP	n/a	A JSON-style array of literals: { literal : literal, literal : literal ... }
SET	n/a	A collection of one or more elements
TEXT	strings	UTF-8 encoded string
TIMESTAMP	integers, strings	Date plus time, encoded as 8 bytes since epoch
TUPLE	n/a	Up to 32k fields
UUID	uuids	A UUID in standard UUID format
TIMEUUID	uuids	Type 1 UUID only (CQL 3)
VARCHAR	strings	UTF-8 encoded string
VARINT	integers	Arbitrary-precision integer

Apache Cassandra - TUPLE

- Tuples hold fixed-length sets of typed positional fields
  - Convenient alternative to creating a user-defined type
  - Accommodates up to 32768 fields, but generally only use a few
  - **Useful when prototyping**
  - Must use the **frozen keyword** in C* 2.1
  - Tuples **can be nested** in other tuples

Apache Cassandra - TUPLE

```
CREATE TABLE user (
id UUID PRIMARY KEY,
email text,
equalizer frozen<tuple<float,float,float,float,float,
float,float,float,float,float>>,
name text,
preferences set<text>
);
```

Apache Cassandra – ORDER BY

- CLUSTERING ORDER BY defines how data values in clustering columns are ordered (ASC or DESC) in a table
  - ASC is the default order for all clustering columns
  - When retrieving data, the default order or the order specified by a CLUSTERING ORDER BY clause is used
- **The order can be reversed in a query using the ORDER BY clause**

Apache Cassandra – ALTER TABLE

- ALTER TABLE manipulates the table metadata
  - Adding a column
    - ALTER TABLE album ADD cover_image VARCHAR;
  - Changing a column data type
    - Types must be compatible
    - **Clustering and indexed columns are not supported**
      - ALTER TABLE album ALTER cover_image TYPE BLOB;
  - Dropping a column
    - **PRIMARY KEY columns are not supported**
      - ALTER TABLE album DROP cover_image;

Apache Cassandra – DROP TABLE

- DROP TABLE removes a table (all data in the table is lost)
  - DROP TABLE album;

What is a secondary index?

- Tables are indexed on columns in a primary key
  - Search on a partition key is very efficient
  - Search on a partition key and clustering columns is very efficient
  - **Search on other columns is not supported**
    - **Last version of Cassandra relax this constraint**
- Secondary indexes
  - Can index additional columns to enable searching by those columns
    - one column per index
  - Cannot be created for
    - counter columns
    - static columns

What is a secondary index?

```
CREATE TABLE performer (
name VARCHAR,
type VARCHAR,
country VARCHAR,
style VARCHAR,
founded INT,
born INT,
died INT,
PRIMARY KEY (name)
);
CREATE INDEX performer_style_key ON performer (style);
```

What is a secondary index?

- DROP INDEX performer_style_key;

When do you want to use a secondary index?

- Secondary indexes are for searching convenience
  - Use with low-cardinality columns
    - **Columns that may contain a relatively small set of distinct values**
      - For example, there are many artists but only a few dozen music styles
      - Allows searching for all artists for a specified style (a potentially expensive query because it may return a large result set)
  - Use with smaller datasets or when prototyping
- Do not use
  - On high-cardinality columns
  - On counter column tables
  - On a frequently updated or deleted columns
  - To look for a row in a large partition unless narrowly queried
    - e.g., search on both a partition key and an indexed column

Time for Exercises