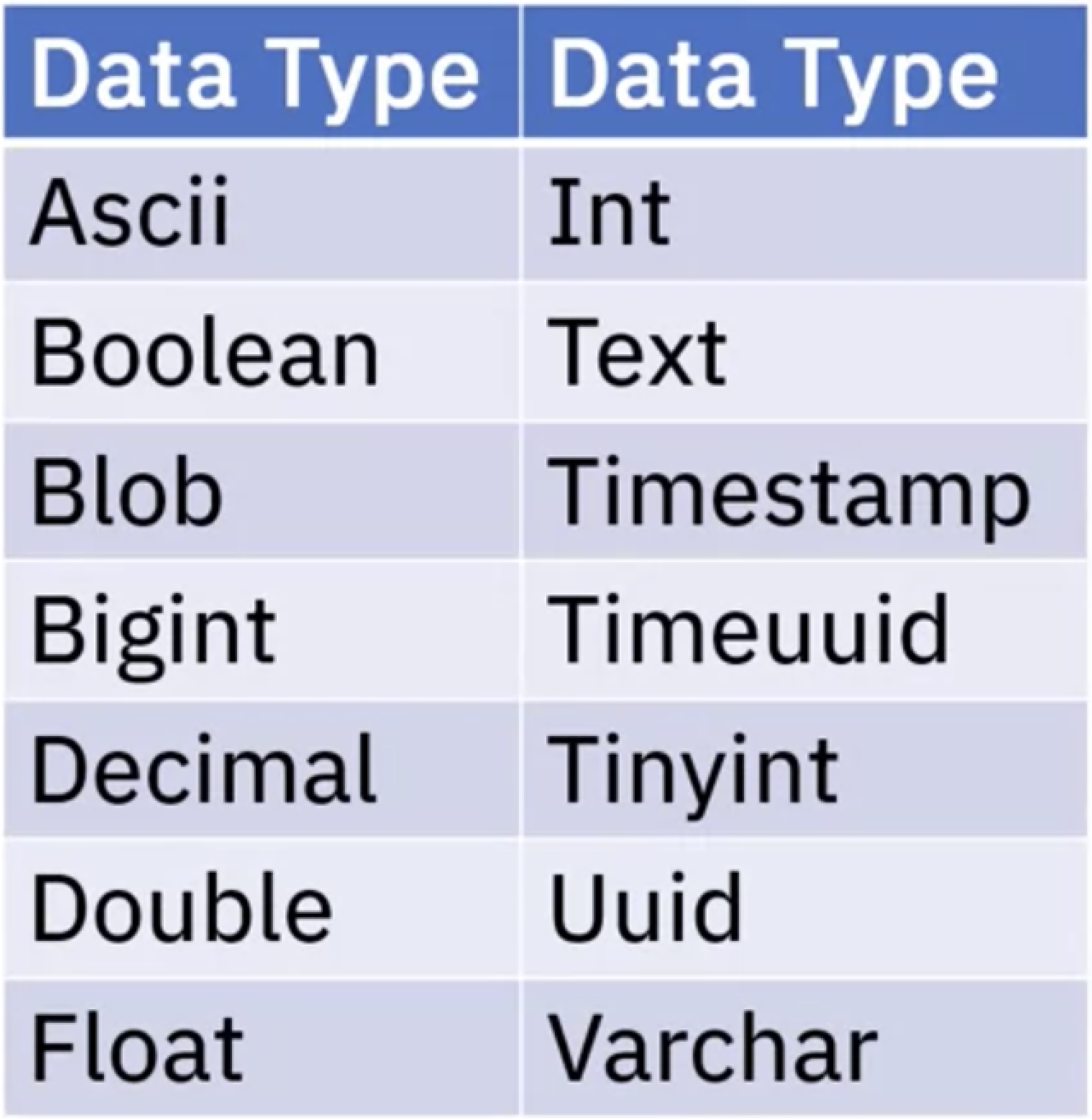
Week3_Working_with_Cassandra

CQL Data Types

Main categories:

- **Built-in Data Types**



Data Type	Data Type
Ascii	Int
Boolean	Text
Blob	Timestamp
Bigint	Timeuuid
Decimal	Tinyint
Double	Uuid
Float	Varchar

Blob: arbitraty bytes. A blob type is suitable for storing a samll image or short string (1 MB)

Bigint - used for a 64-bit signed long integer. This data type stores a higher range of integers as compared to int

Varchar - It is used for strings, and represents a utf8 encoded string

- **Collection Data Types**
- Collections are a way to group and store data together
    - Example: user has multiple email addresses
        - In relational world: a many-to-one joined relationship between `users` table and an `email` table
        - In Cassandra: no joins, we store all the data in a collection column in the `users` table
        - Data for collection storage should be limited - no unbounded growth
        - Not suitable for storing sent messages or sensor events stored every second

  Collection data types can be:

    - Lists
        - When order of the elements needs to be maintained
        - Example: entries in logs
    - Maps
        - Key:value
        - Example: entries in a journal (date:Text)
    - Sets
        - When elements are unique and do not need to be stored in a specific order
        - Example: list of email addresses
- **User-defined Data Types**
    - Collection data types for one-to-many - UDTs for one-to-one
    - Can attach multiple data fields, each named and typed to a single column
    - The fields used to create a UDT may be any valid data type, including collections and other existing UDTs
    - Once created, UDTs may be used to define a column in a table

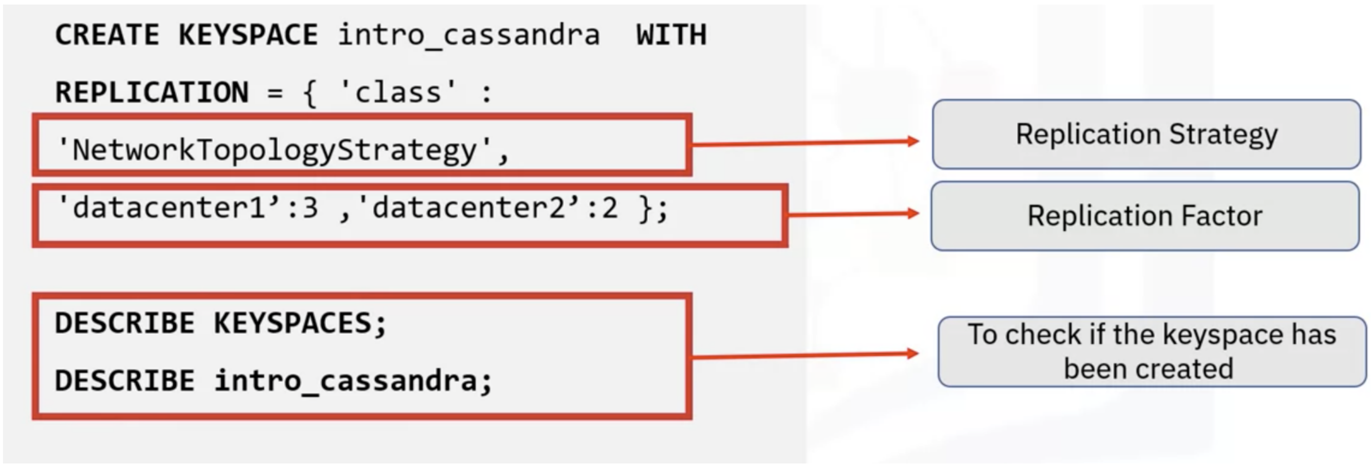
In summary:

- Cassandra supports built-in, collection, and user-defined data types;
- Both collection and user-defined data types offer a way to group and store data together;
- Collection data types can emulate one-to-many relationships;
- There are three types of collection data types: lists, maps, and sets;
- UDTs can emulate one-to-one relationships;
- UDTs allow users to attach multiple data fields to a column;

Keyspace Operations

- Keyspace needs to be defined before creating tables

- Keuspace can contain any number of tables, and a table belongs to only one keyspace
- Replication is specified at the keyspace level
- Youy need to specify the replicatioon factor during the ceration of keyspace - which can be modified later



```
CREATE KEYSPACE intro_cassandra  WITH
REPLICATION = { 'class' :
'NetworkTopologyStrategy',
'datacenter1':3 ,'datacenter2':2 };

DESCRIBE KEYSPACES;
DESCRIBE intro_cassandra;
```

- Replication Factor
  - Number of replicas placed on different nodes in the cluster
- Replication Strategy
  - Which nodes are going to house the replicas
- Replicas
  - All replicas are equally important - no primary or secondary replicas
  - Replication factor should not exceed the number of cluster nodes

Examples:

The below command creates a keyspace called training, using SimpleStrategy and a replication_factor of 3.

SimpleStrategy is used when all the nodes in your cassandra cluster exist in a single data center.

On cqlsh run the below command.

- Create

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

`describe keyspaces`
`describe training`

- Alter
  In a previous exercise you created a keyspace named training using SimpleStrategy.
  Let us change that to use NetworkTopologyStrategy.

NetworkTopologyStrategy is used when all the nodes in your cassandra cluster are spread across multiple data centers.

Alter a keyspace.

```
ALTER KEYSPACE training
WITH replication = {'class': 'NetworkTopologyStrategy'};
```

`describe training`

`use training;`
`describe tables`

- Drop

```
drop keyspace training;
```

```
use system;
describe keyspaces
```

Table Operations

- Data in Cassandra is organized logically in tables
- A table's metadata specifies the primary key - instructing Cassandra how to distribute the table data at cluster and node level
- You can add Time to Live at table level - meaning that you can expire (delete) all data that has surpassed the TTL
- You can modify the columns and column names but only for regular columns
- Primary key, once defined at table creatin, cannot be modified
- You can either drop a table or truncate its data

Examples
The below command creates a table named movies, in the training keyspace.
The movies table has three columns:

- ‘movie_id’ is an integer and is the primary key.
- ‘movie_name’ is a text column.
- ‘year_of_release’ is an integer.

```
use training;
CREATE TABLE movies(
movie_id int PRIMARY KEY,
movie_name text,
year_of_release int
);
```

- Alter

```
ALTER TABLE movies
ADD genre text;
```

- Drop
  `drop table movies;`

Summary

1. Cassandra supports built-in, collection, and user-defined data types.
2. Both collection and user-defined data types offer a way to group and store data together.
3. Keyspaces are defined before creating tables, and a keyspace can contain any number of tables.
4. Common keyspace operations are CREATE KEYSPACE, ALTER KEYSPACE, and DROP KEYSPACE.
5. Cassandra organizes data logically in tables.
6. A table's metadata specifies the primary key – instructing Cassandra how to distribute the table data at the cluster and node level.
7. Cluster level writes are sent to all the partition's replicas, irrespective of the consistency factor.
8. By default, Cassandra doesn't perform a read before writes, therefore INSERT and UPDATE operations behave similarly.
9. Reads at cluster level are sent only to the number of replicas according to the consistency setting.
10. Reads should follow the Primary Key columns order for best performance.