**PHYSICAL DATA MODELING IN CASSANDRA**

Creating the physical model is a reasonably simple task once the logical data model has been developed.

You go through each of the logical model tables, giving each object a type. Any valid CQL data type, including the basic kinds, collections, and user-defined types, can be used. Additional user-defined types that can be generated to ease your design may be discovered.

After assigning data types, you analyse the model by performing size calculations and checking its functionality. Based on your results, you may make some changes. Let's go over the data modelling process in further depth again by using an example.

Before we begin, let's have a look at a few modifications to the Chebotko notation for physical data models. You must be able to add the type information for each column in order to draw physical models. This illustration depicts the insertion of a type to each column in a sample table.



The picture includes a keyspace indication for each table as well as visual hints for columns represented by collections and user-defined types. Take note of the distinction between static columns and secondary index columns. There are no restrictions on including these in a logical model; nevertheless, they are often more of a physical data modelling concern.

**HOTEL PHYSICAL DATA MODEL :**

Let's get started on the physical model. To begin, keyspaces are required to contain the tables. To keep the architecture as simple as possible, make a hotel keyspace with tables for hotel and availability data and a reservation keyspace with tables for reservation and guest data. In a practical system, the tables might be divided across even more keyspaces to segregate concerns.

For the hotels table, use Cassandra’s text type to represent the hotel’s id. For the address, create an address user defined type. Use the text type to represent the phone number, as there is considerable variance in the formatting of numbers between countries.

While the uuid type makes sense for attributes like the hotel\_id, this document generally utilises text attributes as identifiers to keep the samples simple and legible. In the hospitality business, for example, it is common practice to refer to properties using short codes such as "LN3477" or "DL7734." This example employs these values for hotel\_ids while realising that they are not always globally unique.

You'll discover that using unique IDs to uniquely reference elements and using these uuids as references in tables representing other things is frequently useful. This helps to reduce coupling between entity kinds. This may be especially useful if your application employs a microservice architecture style, with different services in charge of each entity type.

You take the similar method for creating physical representations of various tables in the logical hotel data model. The resulting design is depicted in the figure:



It is worth noting that the address type is also included in the design. It is marked with an asterisk to indicate that it is a user-defined type with no recognised primary key columns. This type is used in the tables hotels and hotels\_by\_poi.

As with the address user-defined type, user-defined types are widely used to assist reduce duplication of non-primary key columns. This can help to reduce design complexity.

Remember that the keyspace in which a UDT is defined determines its scope. You must declare address again to utilise it in the reservation keyspace defined below design. This is simply one of many trade-offs to consider when designing a data model.

**RESERVATION PHYSICAL DATA MODEL :**

Let's have a look at the reservation tables in the design now. Remember that the logical model had three denormalized tables to handle reservations queries by confirmation number, guest, hotel, and date. Assume you'll manage this denormalization manually in the initial iteration of your physical data model design. It should be noted that this design might be updated to make use of Cassandra's (experimental) materialized view capability

 

It is worth noting that the address type is replicated in this keyspace, and guest\_id is modelled as an uuid type in all tables.