Schema Design and Evaluation for Cassandra

by

HAJARE KAVAN VIJAYBHAI
202111007

A Thesis Submitted in Partial Fulfilment of the Requirements for the Degree of

MASTER OF TECHNOLOGY
in
INFORMATION AND COMMUNICATION TECHNOLOGY
to

DHIRUBHAI AMBANI INSTITUTE OF INFORMATION AND COMMUNICATION TECHNOLOGY

July, 2023

Declaration

I hereby declare that

i) the thesis comprises of my original work towards the degree of Master of Technology in Information and Communication Technology at Dhirubhai Ambani Institute of Information and Communication Technology and has not been submitted elsewhere for a degree,

ii) due acknowledgment has been made in the text to all the reference material used.

[illegible]

Hajare Kavan Vijaybhai

Certificate

This is to certify that the thesis work entitled Schema Design and Evaluation for Cassandra has been carried out by HAJARE KAVAN VIJAYBHAI for the degree of Master of Technology in Information and Communication Technology at *Dhirubhai Ambani Institute of Information and Communication Technology* under my/our supervision.

[illegible]

P M Jat
Thesis Supervisor

Acknowledgments

I would like to express my deepest gratitude to my supervisor, Prof. P M Jat, for their guidance, expertise, and unwavering support throughout the entire thesis work. Their insightful feedback and constructive advice have been invaluable in shaping this thesis.

I extend my sincere appreciation to the faculty members of DAIICT, for their constant encouragement and willingness to share their knowledge. Their lectures, discussions, and academic resources have significantly contributed to the development of interest in this work.

I am grateful to my family for their unconditional love, unwavering support, and belief in my abilities and God for giving me the mental strength. Their encouragement and understanding have been instrumental in providing the emotional strength necessary to complete this thesis.

Last but not the least, I want to express my gratitude to my friends and colleagues for their encouragement, insightful discussions, and moral support. Their presence has made this journey more enjoyable and less daunting.

Contents

Abstract

Cassandra is one of the widely used distributed database because of some features like fault tolerance, partition tolerance, seamless replication and scalability. However, designing the schema for Cassandra is a challenging task. This work focuses on automating the schema design procedure for Cassandra. Following are the important reasons for schema design automation. 1) Schema design for Cassandra requires query load as an input along with the conceptual model which is different from relational schema design. 2) Inclusion of a query load as an input adds more complexity to the schema design process. To automate the schema design process, we study and formulate some mapping rules and develop an algorithm capturing these rules. The algorithm takes two inputs namely ER diagram and application queries. The algorithm produces an output schema which contains the list of attributes required to answer the query, partition key attributes and clustering key attributes. Evaluation of the algorithm is done manually. We use three case studies for this purpose, out of which two are from literature namely digital library system and hotel management system and remaining one is our own application which is an E-commerce application. Results which we get on executing this algorithm on mentioned cases are turned out to be correct.

Index terms— **Apache Cassandra, data modeling, Schema Design**

List of Figures

CHAPTER 1
Introduction

Cassandra is leading No SQL database which is capable of storing huge amount of data which is distributed across multiple nodes, where multiple nodes are spread across multiple data centers, and builds a Cassandra cluster. Large data can be well maintained using Cassandra through its partitioning system where data is stored and retrieved from partition. Cassandra also provides fast writes which can store the huge amount of data which is generated by modern applications. It also provides CQL called as "Cassandra Query Language" which is almost similar to SQL for Relational databases. Apart from that Cassandra is having wide acceptance because of various features it provides. [2] [5]

1.1 Cassandra Features

- **Open-Source:** Cassandra is an open source provided by Apache for everyone to utilize.

- **Scalability:** Cassandra offers adaptable scaling because of which both scaling up or down is possible. Cassandra scales horizontally through which additional nodes can be added to store the extensive amount of data. During scaling process, it is not required to restart the cluster. Furthermore, there is no pause or downtime while scaling. As a result, throughput for both reading and writing increases. A token is given to each new node when it is joined to the system so that it can relieve a node that is overloaded. As a result, the new node splits a range that was previously controlled by another node. [6]

- **Fault tolerant:** Cassandra's capacity to replicate data is what makes it fault-tolerant. This increases its availability and system fault tolerance. As same content can exist on multiple nodes in the cluster, the failure of a single node or data centre does not bring the system to a complete stop which means zero downtime can be achieved through avoiding single point of failure. Users will be instantly transferred to the closest operational node if a certain node fails. The system will continue to operate as intended which is crucial for businesses that can never afford to lose any data or have their database go offline.

- **Seamless replication:** Because Cassandra is a peer-to-peer system, data can be swiftly replicated throughout the entire system, regardless of location. Fault tolerance feature and zero data loss are satisfied by replication done across data centres. Latency can also be improved by bringing data closer to end users. Replication of data results in extensive backup and recovery. Replication used by Cassandra to get great availability and robustness. All data item is duplicated over N hosts, where N is the replication factor per instance. Each key, k, is paired with a coordinator node that is situated higher on the ring than the position of the corresponding item. Replication of the data items is controlled by the coordinator. Keys are copied at the N-1 nodes in the circle apart from locally storage of each key inside its range. Cassandra has a number of replication policies, including "Datacenter Aware," "Rack Unaware," and "Rack Aware". The replication rule is selected by the application. If a particular application selects the "Rack Unaware" replication approach, the replicas which are non-coordinator, determined by picking the coordinator’s N-1 inheritors on the ring. For remaining two, Cassandra selects a leader from among its nodes. The leader makes a concentrated attempt to preserve the invariant that no node is accountable for more than N-1 ranges in the ring when a node contacts the leader to inform them for what ranges they are replicas. [6]

- **Column-Family:** In Cassandra, a column family is a container that holds related columns and is similar to a relational database table. Each row in a column family has a unique key, which is used to access the data in the row. The columns in a row are stored together as a unit, known as a wide row. One of the key features of column families in Cassandra is that they allow for flexible and dynamic data modeling. This means that new columns can be added to a column family without having to modify the entire schema. Overall, the column-family feature of Cassandra making it well-suited for handling large-scale, rapidly changing data sets.

1.2 Cassandra Applications

- **E-commerce:** Due to the wide usage of E-Commerce application by users, system should be always up and running which is provided by Cassandra's fault tolerance feature that allows it to keep running even if some nodes in cluster are down as data can be fetched from other nodes due to replication. E-Commerce can take this advantage of Cassandra to avoid downtime. Cassandra's scalability feature comes into picture when too many users are using the application simultaneously, because of which, the company has to increase the ability of the database to carry and store more data without causing system to restart.

- **IOT:** IOT devices in form of various sensors like weather, traffic or mobile device keeps track of and sends data on weather, traffic, energy usage, soil conditions, etc, in which data is being created at high frequency that require system which can store large volume of data and Cassandra can be very useful as it can store a large volume of data where every individual node can carry out read and write operations and supports analysis of data in real time.

- **Transportation:** Real world applications like Uber uses Cassandra to utilize its scalability and fault tolerance features which helps them in implementing their services like dispatching of trip, payment and matching of driver-rider.

1.3 Relational vs Cassandra Schema Design

In Relational schema design, only an ER diagram is required as an input, from which the relational tables can be create using concepts of relationships that exists between entities of ER diagram. Once tables are created, any queries can be answered using join, aggregation and nested queries of relational among multiple tables. Whereas in Cassandra, along with ER diagram, list of application queries that defines an application workflow is also given as an input. [4] Since Cassandra doesn't support of join operation which is the major challenge in creating schema for Cassandra. So based on given two inputs and analysis which is done on query input schema design can be done for Cassandra. Below figure shows the difference between Relational data modeling and Cassandra data modeling process.

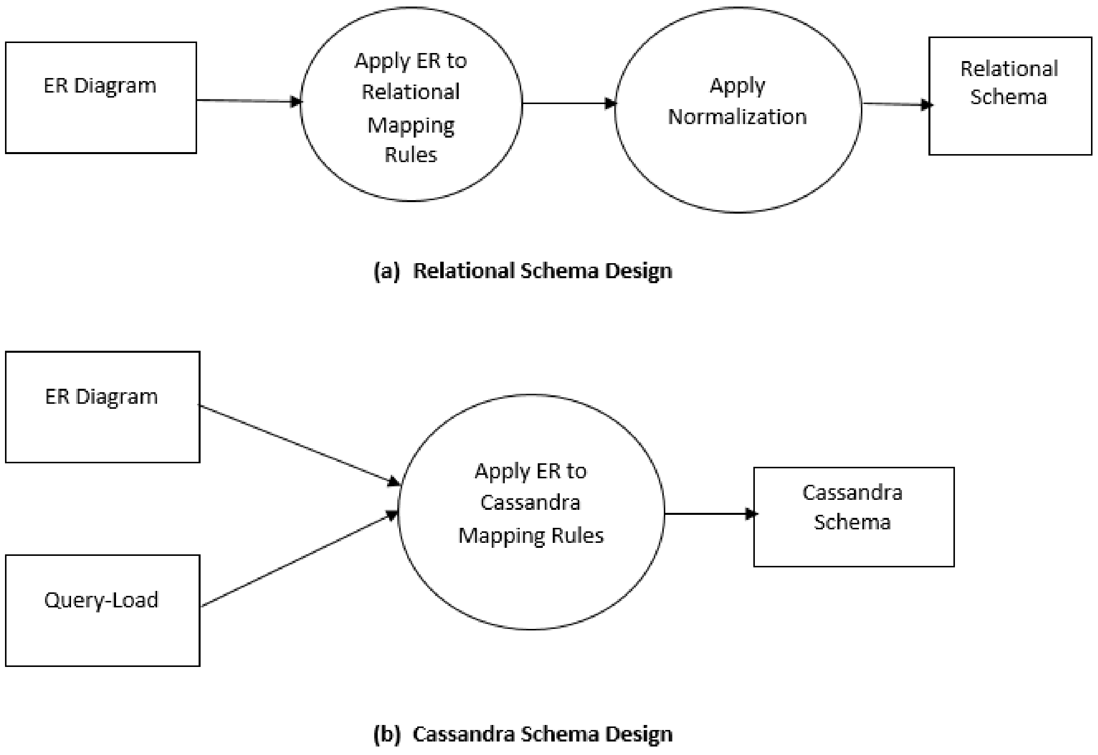

ER Diagram
Apply ER to Relational Mapping Rules
Apply Normalization
Relational Schema
(a) Relational Schema Design
ER Diagram
Query-Load
Apply ER to Cassandra Mapping Rules
Cassandra Schema
(b) Cassandra Schema Design

Figure 1.1: Schema Design Process - Relational vs Cassandra Data Model

1.4 Thesis Contribution

This thesis work focuses on devising an algorithm which automates the schema design process for Cassandra. To implement this algorithm, we study various Cassandra mapping rules which are found in literature. [4] Based on these rules, we formulate our own mapping rules and try to develop the algorithm which captures these rules. Implementation of the algorithm considers two inputs: 1) ER diagram and 2) List of application queries. These inputs are represented in the form of relational tables. The algorithm produces an output in form of schema that can answer the application query. The evaluation of Cassandra schema design is done manually. Results which we get on executing this algorithm on test cases are found to be correct. We hope that this algorithm reduces the manual effort up to a certain extent in Cassandra schema design process.

1.5 Thesis Outline

Remaining part of thesis has following structure: Chapter 2 covers the concept of Cassandra Data Model which includes the detailed description of Cassandra Data Model components and information on various data types used in Cassandra along with data model structure . Chapter 3 gives the detailed description about schema design approach for Cassandra, experiment details with input and output and brief explanation on notation that represents relationship between entities based on common attributes called as CHEBOTKO notation. Chapter 4 covers schema evaluation part by running the algorithm on 3 different applications along with results and discussions of it etc. Finally, the thesis work is concluded along with future work in Chapter 5.

CHAPTER 2
Cassandra Data Model

This chapter gives an introduction to the Cassandra Data Model. In first part components of Cassandra Data Model are covered and after that structure and data types of Cassandra are discussed. The Cassandra database is spread over a number of interconnected machines. The Cluster is the name assigned to the outermost container. Cassandra assigns data to the nodes in a cluster and arranges them in a ring pattern. Cassandra data model consists of four components namely keyspace, table (column-family), partition key and clustering key.

2.1 Cassandra Data Model Components

1. **Keyspace:** A keyspace one of the most core part of Cassandra database represents schema itself and thus acts as the top-level namespace for other sub components like tables which comes under the keyspace and are capable of storing and query the data for an application under consideration. A table in Cassandra can be considered as a set of partitions which contains rows having common structure which we called as a column-family. Each row in partition consists of partition key and optionally a clustering key, combination of these two is called as primary key.

2. **Table:** Within the keyspace, tables are defined. In Cassandra, tables are also known as Column Families where data is stored in rows and is organised in tables by columns and a primary key. Basically, table is a container for a collection of rows which are ordered and in each row, in turn is a collection of columns that are also ordered. In Cassandra, column families can be split into two categories: standard column families, which are used to store data with a fixed set of columns, each of which has a name and a value. Super column families are used to store data with a nested structure, where

each column comprises a group of sub-columns, and standard column families are great for storing simple and structured data, such as user profiles or product details. Super column families are perfect for holding information that has several different characteristics or qualities, such as user posts on social networking.

3. **Partition key:** In Cassandra, a partition key is an element of the primary key that determines how data is dispersed across the cluster. Cassandra partitions data by the hash value of the partition key, which defines the node where the data will be kept. The partition key is defined as an initial part of the primary key and uniquely identifies a row within a table. Choosing the right partition key is critical for achieving good performance in Cassandra. For example, if table stores information regarding artifacts published at particular venue, it is meaningful to partition the data based on venue, where each partition contains details of artifacts for particular venue.

4. **Clustering key:** In Cassandra, a clustering key is an element of the primary key that determines an order in which data is stored within a partition. The clustering key specifies the order in which data is sorted within a partition, as opposed to the partition key, which identifies the node where data is stored. For example, if table stores information regarding artifacts published at particular venue, it is meaningful to sort the data based on artifact's id as one venue can have multiple artifacts getting published. In that case combination of venue name and id of an artifact can uniquely define the records of the table. A clustering key is defined as part of the primary key after the partition key. One or more clustering columns may be included in a table, and the order of these columns affects how the data in a partition is sorted.

Fig. 2.1. Shows an example of Cassandra data model components for an e-commerce application.

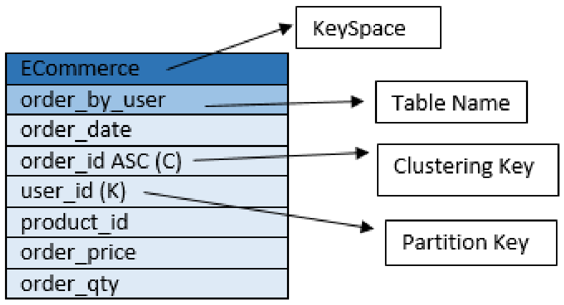


Figure 2.1: Cassandra Data Model Example

2.2 Cassandra Data Model Structure and Data Types

Partition key and clustering key both can either consists of single column or multiple column respectively that is called as simple and composite relatively. Primary key is a combination of both the key generally but out of these two components, partition key is mandatory part whereas clustering key is an optional one. Absence of clustering key suggests that there exists a partition with a single row as partition key itself sufficient to represent one partition, which is exactly opposite to other case in which clustering key is present where partition consists of multiple row partitions and rows are sorted based on given clustering key with assigned order like ascending and descending from which ascending is the default one. Properly chosen partition key can do an even distribution of the data across the nodes in the given cluster which can minimize the partition access required to answer the query which makes read operation faster for Cassandra.

There are different kinds of data types exists for attributes of Cassandra schema like regular and complex. Regular data type contains int, text etc, whereas complex consists of list, set and map. There is one special data type called counter column which keeps a distributed counter that can be incremented or decremented using synchronised operations done on cluster. Initial value of counter column is zero before it is actually updated for the first time.

A column with all the rows having same value for it in the partition is called as a static column, which is suitable in a table with numerous row separations. For showing a primary key, if partition key consists of only single column, then there is no need of an additional parenthesis to separate partition key from clustering key. Moreover, columns that are counter, static or collection type cannot be primary keys. Fig. 2.2. shows the overall structure of Cassandra Data Model and Fig. 2.3 shows the Cassandra Data Model components along with the partitions and few sample records for an E-commerce application.

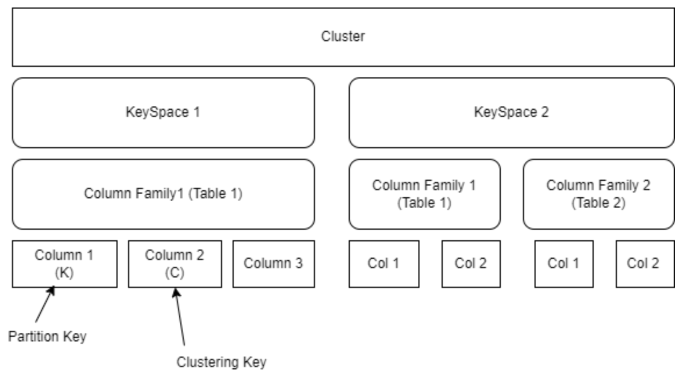
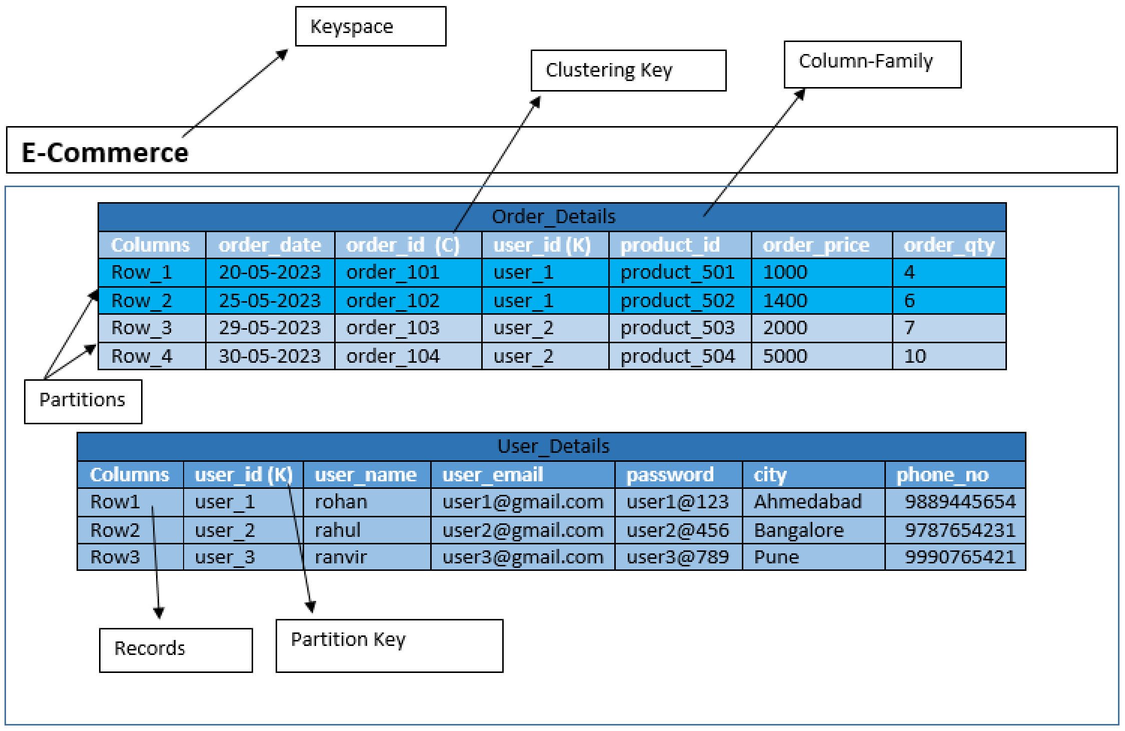


Figure 2.2: Cassandra Data Model Structure



Order_Details

Columns	order_date	order_id (C)	user_id (K)	product_id	order_price	order_qty
Row_1	20-05-2023	order_101	user_1	product_501	1000	4
Row_2	25-05-2023	order_102	user_1	product_502	1400	6
Row_3	29-05-2023	order_103	user_2	product_503	2000	7
Row_4	30-05-2023	order_104	user_2	product_504	5000	10

User_Details

Columns	user_id (K)	user_name	user_email	password	city	phone_no
Row1	user_1	rohan	user1@gmail.com	user1@123	Ahmedabad	9889445654
Row2	user_2	rahul	user2@gmail.com	user2@456	Bangalore	9787654231
Row3	user_3	ranvir	user3@gmail.com	user3@789	Pune	9990765421

Figure 2.3: Cassandra Data Model For Ecommerce Application

CHAPTER 3

Cassandra Schema Design

This chapter includes chebotko notation, an approach to schema design for Cassandra database along with input and output representation and at last the algorithm for schema generation from the given input which are an ER diagram and application queries.

3.1 CHEBOTKO Notation

Extended entity-relationship (EER) diagrams, commonly referred to as Chebotko diagrams, are a type of graphical notation used in data modelling to represent the relationship between entities based on common attributes between them. A visual representation of the logical and physical data model of Cassandra is offered via Chebotko diagrams. It represents a schema design as a combination of unique tables and the transition of an application queries. It helps with readability and better expressivity. Below figure shows an example through couple of queries from Case 1 of Chapter 4.

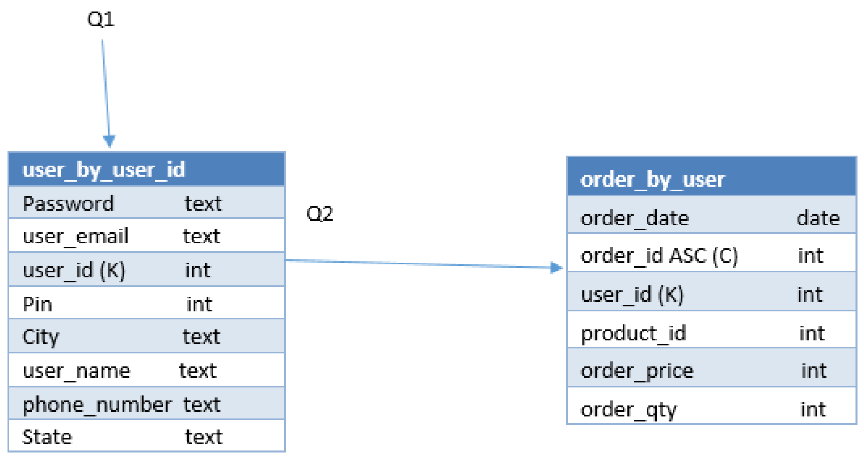


Figure 3.1: CHEBOTKO Notation Example

3.2 The Design Approach

For efficient Cassandra schema design, following steps are important:

1. **Understanding the conceptual model:** It helps in an identification of entities, relationships and attributes required to answer the application queries.
2. **Knowledge about an application:** Understanding the overall flow of an application under consideration helps in identifying the application queries for which the schema design is done.
3. **Organizing of an entities:** Organizing of entities participating in answering the query is important which is also called as nesting of one entity into another.

Below mentioned details gives the information on an input and output of the Cassandra schema design process along with the approach to the algorithm.

- **Input:** For designing the Cassandra schema using the devised algorithm, two inputs are taken: 1) ER diagram and 2) Application queries. Both the input are then represented using tables on which algorithm is applied.

- **Output:** Cassandra schema is the output of the algorithm when it is applied to the given two inputs which consists of attributes required to answer the query along with partition key and clustering key attributes.

- **Algorithmic Approach:** The algorithm which is devised to output the Cassandra schema by taking inputs in form of an ER diagram and application query. Inputs are represented in relational table form and mapping rules of Cassandra are applied on it. Representation of inputs is described in upcoming section. To apply mapping rules on given input, predicate part of query is analysed. Analysis done on predicate part of query outputs three components namely list of attributes required to answer the query, partition key attributes and clustering key attributes. These three components makes the overall schema when combined.

  As described in Algorithm 1, we need to apply mapping rules to get the final schema which can answer the particular query which can also be referred as query oriented conversion of conceptual model to logical data model. For that, following step by step procedure is applied:

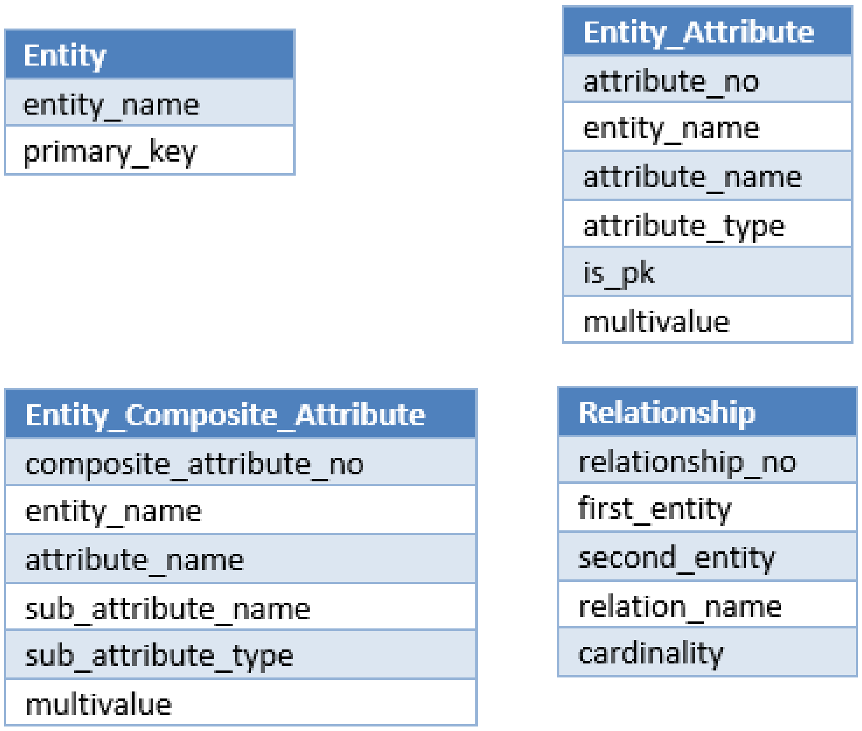
1. **Identification of queried entities and attributes:** As an initial step, we need to identify the entities and attributes required to answer the query. Entities required are defined as "get_entity" and "predicate_entity" in our input representation of an ER diagram as shown in below section. All the attributes from entity present in"get_entity"column are fetched which can answer the particular query.
2. **Mapping of Partition Key Attributes:** Attributes which are restricted with an equal sign (=) in query are mapped to list of partition key attributes. These attributes can be fetched from "equality_attributes" column of query table as shown in section 3.4.
3. **Mapping of Clustering Key Attributes:** Attributes which are restricted with not equal sign (>,<,>=,<= etc) in query are mapped to list of clustering key attributes. These attributes can be fetched from "range_attributes" column of query table along with "order" column to represent the sorted order of those columns as shown in section 3.4.

As an example, we can consider one query like "order details by user" from an E-Commerce application of Case 1 of Chapter 4. Here we need two entities namely order and user, where order is an entity for which details are required (called as get_entity) and user is entity which is considered as a predicate entity. In this, order details are organized with user details which shows the order details are nested per user. Now from query part, we need to identify the attributes that are equally restricted and range restricted and assigned them to partition key and clustering key details respectively. For this query, it happens to be a "user_id" and "order_id" respectively as order details are equally restricted by "user_id" and for each partition done based on "user_id" is sorted based on "order_id". Combination of both partition and clustering key attributes defines the primary key of the table. Finally the table name is given as "order_by_user" based on the naming convention "get_entity_by_predicate_entity" which is followed by the devised algorithm. All the steps followed in the algorithm are mentioned in Algorithm 1. Which is covered in section 3.5.

3.3 ER Diagram Representation

For Cassandra schema design, the algorithm represents inputs in form of tables as discussed previously. For ER diagram representation, which is the first input of the algorithm, we need four tables. Structure for all the tables are shown below:

1. **Entity**(entity_name, primary_key),
2. **Entity_Attribute**(attribute_no,entity_name,attribute_name, attribute_type, is_pk-,multivalue)
3. **Entity_Composite_Attribute**(composite_attribute_no, entity_name, attribute_name, sub_attribute_name, sub_attribute_type, multivalue)
4. **Relationship**(relationship_no,first_entity,second_entity, relation_name, cardinality)



Entity
entity_name
primary_key

Entity_Attribute
attribute_no
entity_name
attribute_name
attribute_type
is_pk
multivalue

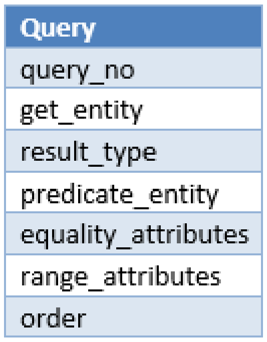
Entity_Composite_Attribute
composite_attribute_no
entity_name
attribute_name
sub_attribute_name
sub_attribute_type
multivalue

Relationship
relationship_no
first_entity
second_entity
relation_name
cardinality

Figure 3.2: Relational representation of an ER (input)

3.4 Query Load Representation

Finally, the second input of the algorithm which is an application query, is also represented in form of table and structure for the same is as show below:
Query(query_no,get_entity,result_type,predicate_entity,equality_attributes,range_attributes, order)



Query
query_no
get_entity
result_type
predicate_entity
equality_attributes
range_attributes
order

Figure 3.3: Relational representation of Query (input)

3.5 Algorithm

Algorithm 1. describes the overall procedure to derive the schema for Cassandra based on the inputs given to it and mapping rules of Cassandra applied. Apart from mapping rules, some adaptions like fetching an attributes for entity of "get_entity" column, representing the "predicate_entity" column as a join path [7], try to give table name as meaningful as possible, multivalue attribute should appear twice in schema, "get_entity_by_predicate_entity" exists then fetch attributes from in memory structure in place of making the database call. To represent the final output schema, five variables are used in Algorithm 1. and these are tablename, attributeSet, partitionKeySet, clusteringKeySet and finalSchema which represents name of column-family for every query of workload, set of attributes required to be fetched for schema, partition attributes, clustering attributes and output variable to store overall attributes for schema respectively. [1]

Algorithm 1: Cassandra_Schema_Design

```
input  : ER diagram and query from workload
output: Schema required to answer the query
1  Take ER diagram and query as an input.
2  Fetch all attributes for the entity type stored in “get_entity” column of
     “Query” table using “Entity_Attribute” table and prime attributes of
     entites present in "predicate_entity" column and store it in “attributeSet”
     variable.
     attributeSet = attributes of (“get_entity”) U attributes present in
     (“primary_key” column of entities present in “predicate_entity” column
     of query table)
3  if “equality_attributes” column is not empty then
4      partitionKeySet = attributes of (“equality_attributes”)
5  else
6      partitionKeySet = attributes present in (“primary_key” column of
         entity/s present in “predicate_entity” column of query table)
7  if attribute A from "equality_attributes" ∈ attributeSet and A is multivalue
     then
8      partitionKeySet = partitionKeySet U A
         attributeSet = attributeSet U A (complex data type)
9  else
10     partitionKeySet = partitionKeySet U A
         attributeSet = attributeSet - A (complex data type)
11 clusteringKeySet = attributes of (“range_attributes”) U primary attributes
     of (“get_entity”)
12 if tablename already exists then
13     if leading attribute of (“equality_attributes”) ∈ "get_entity" then
14         tablename = leading attribute of (“equality_attributes”)_by_”
             predicate_entity”
15     else
16         tablename = “get_entity”_by_leading attribute of
             (“range_attributes”)
17 else
18     tablename = value of (“get_entity”_by_”predicate_entity”)
19 finalSchema = attributeSet U partitionKeySet U clusteringKeySet
```

3.6 Enhancement of the Algorithm

Algorithm 1 defined in previous section is optimised such that if a new query from an application queries can be answered using existing schema which was derived for particular query, no new schema would be constructed. Instead of new schema, our algorithm outputs the query no, whose schema can answer the new query. To implement this, if for current query, "get_entity" and "predicate_entity" column is same as any previous query, then partitionKeySet and clusteringKeySet of current query is compared with that respective query and if both of these are same then current query can be answered using that previous query and thus it's query number is returned.
For example, if one query needs all the details of an artifact published at particular venue which is query number 9 from Case 2 of Chapter 4, and other query which is query number 10, that comes later needs only artifact_title of artifact published at particular venue, then both of these query can be answered using similar schema as "get_entity"and"predicate_entity" column as well as partitionKeySet and clusteringKeySet are same for both the queries. So our result returns the query number from which the current query can be answered. In current example, query number 10 can be answered using the schema of query number 9.

CHAPTER 4
Schema Evaluation

This chapter covers the evaluation of schema design algorithm for which the 3 different types of applications are used for testing of the algorithm along with the results and its respective discussion.

4.1 Cases

The devised algorithm is executed for below mentioned 3 different types of applications and evaluated manually. For evaluation of second and third application, specific reference is used while first application is evaluated manually. E-commerce application is for online shopping done by users, Digital library contains information on set of artifacts published at venues along with user's review and Hotel management system keeps information of user's reservations of hotel rooms with amenities of room. Following context represents the ER diagram and list of application queries for the respective systems.

1. E-Commerce application
2. Digital Library System
3. Hotel Management System

Case 1: E-Commerce Application

i) ER Diagram:

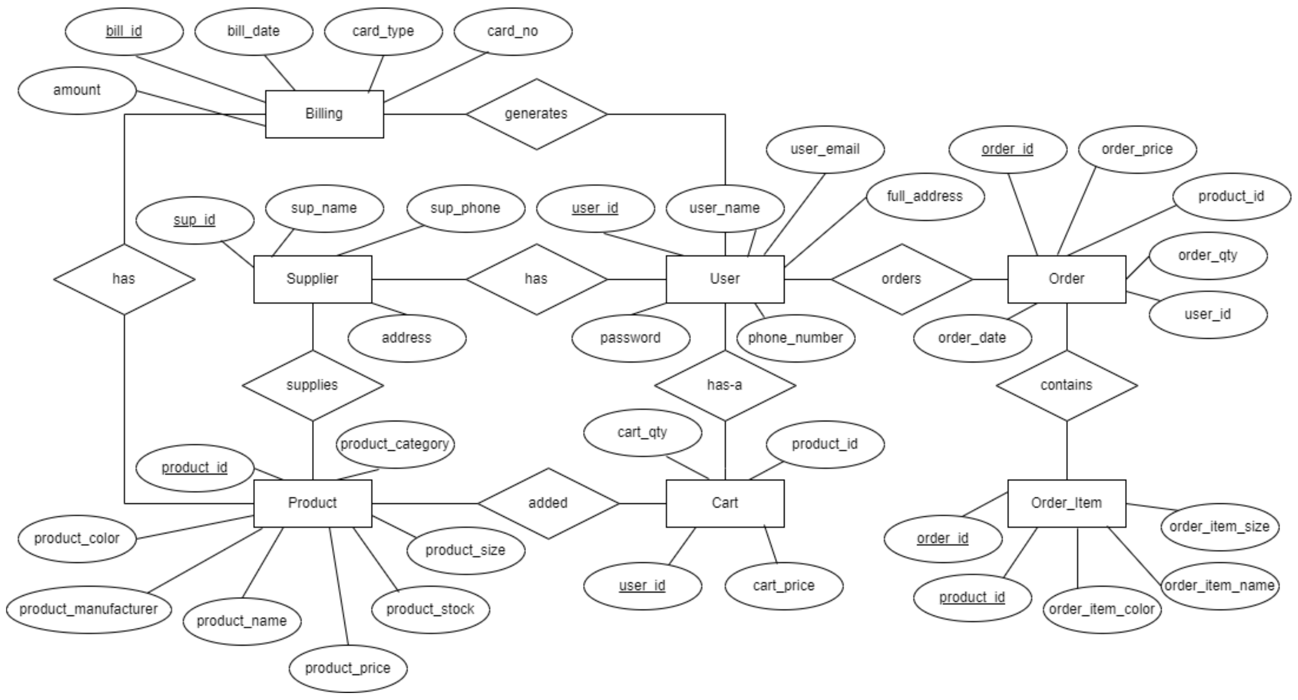


Figure 4.1: ER Diagram For E-Commerce Application

ii) List of Application Queries:

1. Find Order details by user.
2. Find User details by user id.
3. Find Cart details for particular product category and with given minimum product price and range of size in descending order.
4. Find Order Item details for given user name with order date recent to old.
5. List Order Item for particular order price.
6. Find Supplier details by given supplier name.
7. List product name and product category for given supplier name and product stock is more than one.
8. List billing details for given user name and product manufacturer with given minimum product size.

Case 2: Digital Library System [4]

i) ER Diagram:

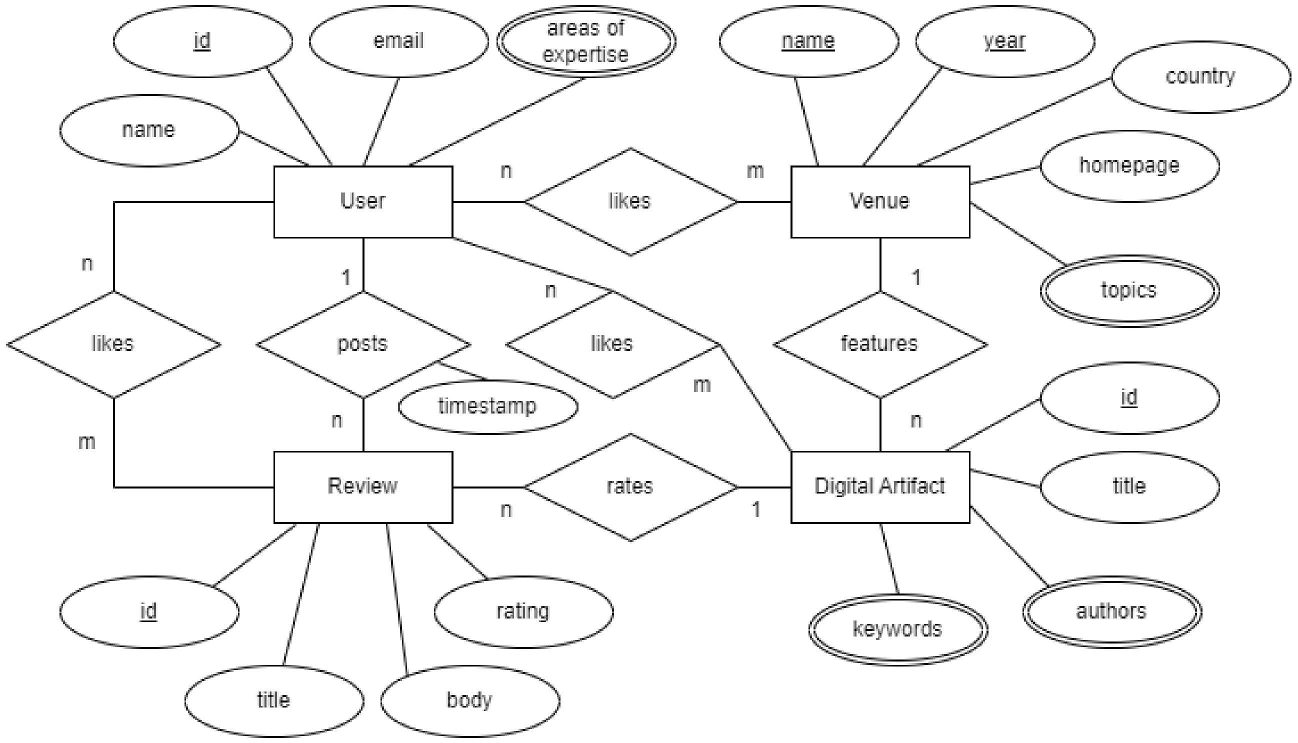


Figure 4.2: ER Diagram For Digital Library Use Case

ii) List of Application Queries:

1. Find artifacts published in a venue with a given name after a given year. Order results by year (DESC).
2. Find artifacts published by a given author. Order results by year (DESC).
3. Find users who liked a given artifact.
4. Find users who liked a given artifact and who have expertise in a certain area.
5. Find an average rating of a given artifact.
6. Find venues that a given user liked.
7. Find artifacts published after a certain year that a given user liked. Order results by year (DESC).
8. Find reviews posted by a given user with a rating >= x. Order results by rating (DESC).

9. Find information about an artifact with a given id.

Case 3: Hotel Management System [3]

i) ER Diagram:

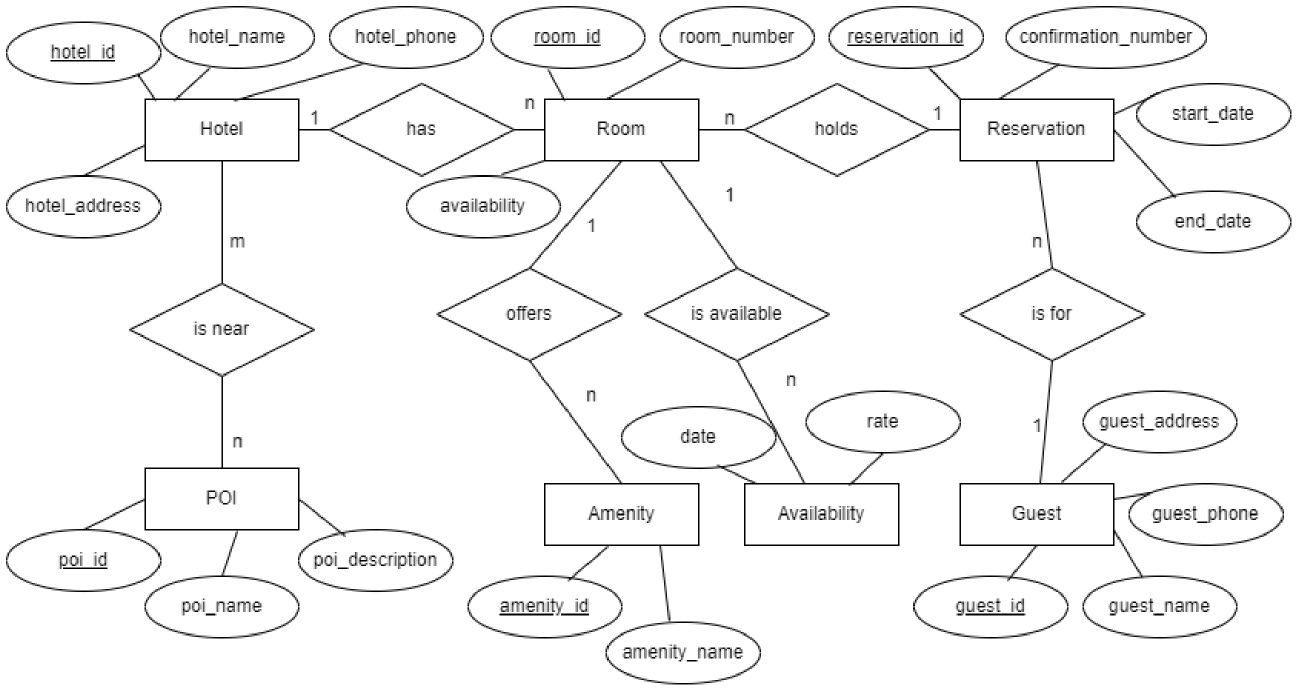


Figure 4.3: ER Diagram For Hotel Management System

ii) List of Application Queries:

1. Find hotels near a given point of interest.

2. Find information about a given hotel, such as its name and location.

3. Find points of interest near a given hotel.

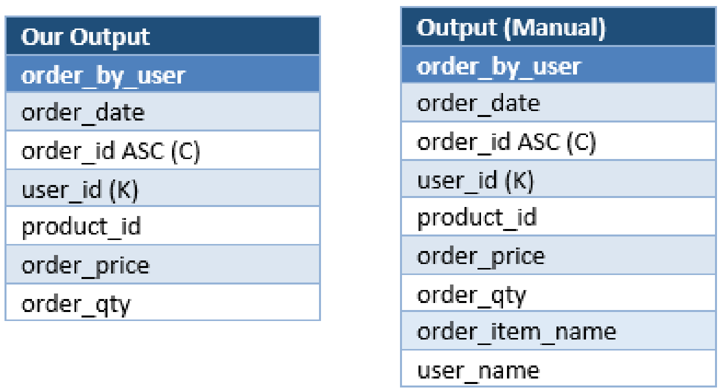
4. Find an available room in a given date range.

5. Find the rate and amenities for a room.

4.2 Results and Discussion

When the algorithm is executed by considering the cases given in above section, we get an output schema for the application queries of given respective applications as shown in left part of every result image given below.

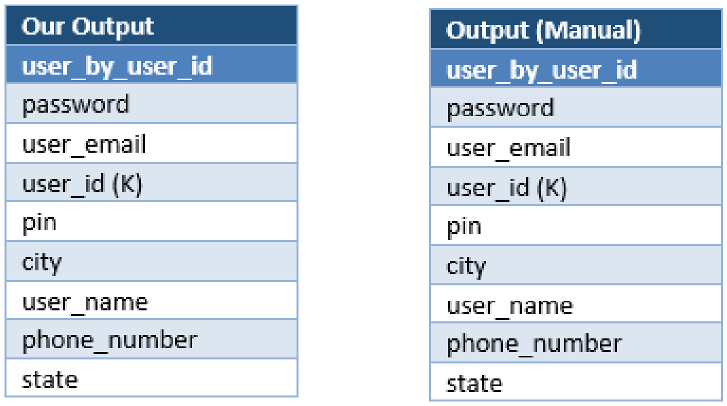
1) Results for E-Commerce Application:



Our Output
order_by_user
order_date
order_id ASC (C)
user_id (K)
product_id
order_price
order_qty

Output (Manual)
order_by_user
order_date
order_id ASC (C)
user_id (K)
product_id
order_price
order_qty
order_item_name
user_name

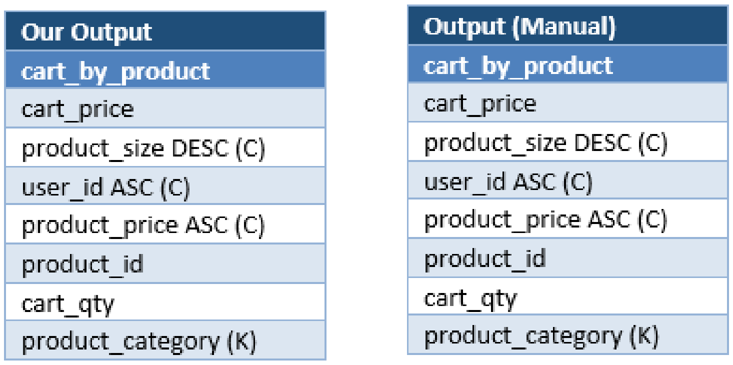
Figure 4.4: Case 1 - Query 1



Our Output
user_by_user_id
password
user_email
user_id (K)
pin
city
user_name
phone_number
state

Output (Manual)
user_by_user_id
password
user_email
user_id (K)
pin
city
user_name
phone_number
state

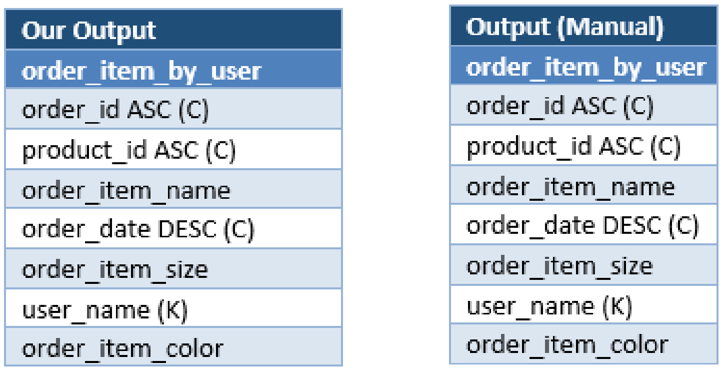
Figure 4.5: Case 1 - Query 2



Our Output
cart_by_product
cart_price
product_size DESC (C)
user_id ASC (C)
product_price ASC (C)
product_id
cart_qty
product_category (K)

Output (Manual)
cart_by_product
cart_price
product_size DESC (C)
user_id ASC (C)
product_price ASC (C)
product_id
cart_qty
product_category (K)

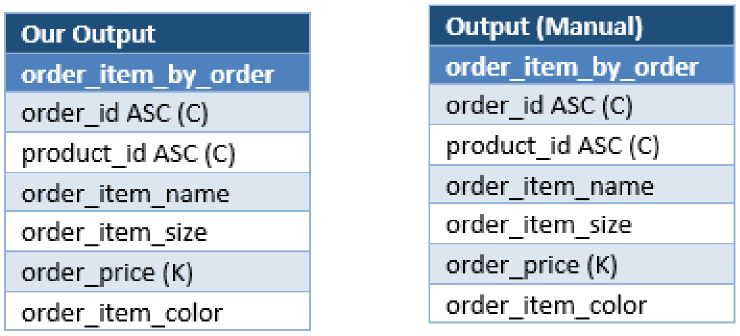
Figure 4.6: Case 1 - Query 3



Our Output
order_item_by_user
order_id ASC (C)
product_id ASC (C)
order_item_name
order_date DESC (C)
order_item_size
user_name (K)
order_item_color

Output (Manual)
order_item_by_user
order_id ASC (C)
product_id ASC (C)
order_item_name
order_date DESC (C)
order_item_size
user_name (K)
order_item_color

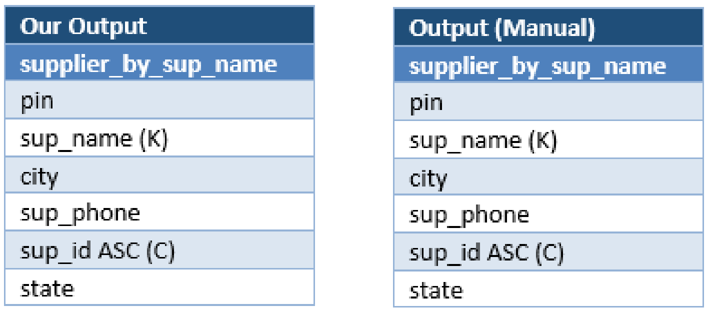
Figure 4.7: Case 1 - Query 4



Our Output
order_item_by_order
order_id ASC (C)
product_id ASC (C)
order_item_name
order_item_size
order_price (K)
order_item_color

Output (Manual)
order_item_by_order
order_id ASC (C)
product_id ASC (C)
order_item_name
order_item_size
order_price (K)
order_item_color

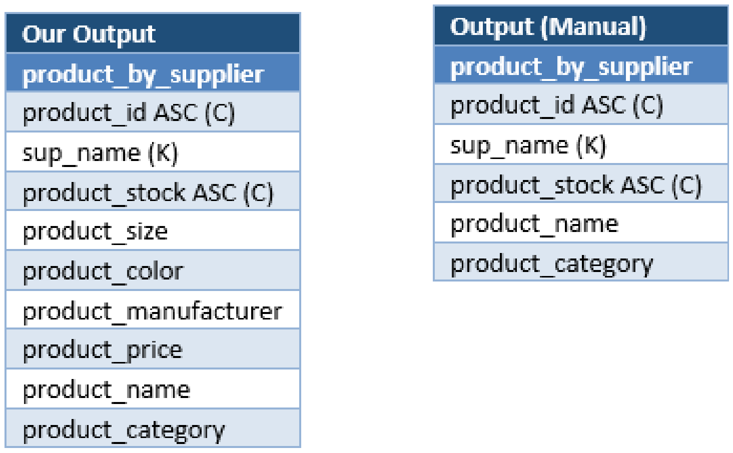
Figure 4.8: Case 1 - Query 5



Our Output
supplier_by_sup_name
pin
sup_name (K)
city
sup_phone
sup_id ASC (C)
state

Output (Manual)
supplier_by_sup_name
pin
sup_name (K)
city
sup_phone
sup_id ASC (C)
state

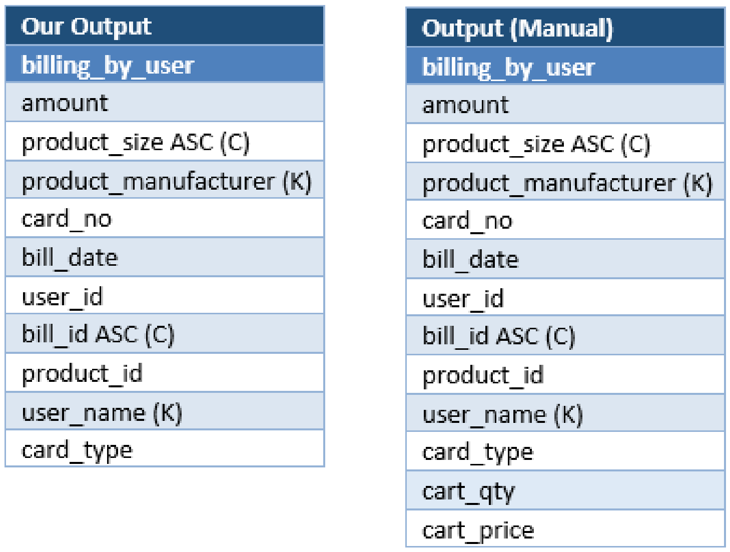
Figure 4.9: Case 1 - Query 6



Our Output
product_by_supplier
product_id ASC (C)
sup_name (K)
product_stock ASC (C)
product_size
product_color
product_manufacturer
product_price
product_name
product_category

Output (Manual)
product_by_supplier
product_id ASC (C)
sup_name (K)
product_stock ASC (C)
product_name
product_category

Figure 4.10: Case 1 - Query 7

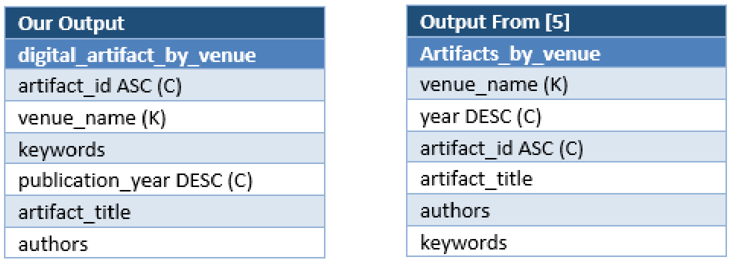


Our Output
billing_by_user
amount
product_size ASC (C)
product_manufacturer (K)
card_no
bill_date
user_id
bill_id ASC (C)
product_id
user_name (K)
card_type

Output (Manual)
billing_by_user
amount
product_size ASC (C)
product_manufacturer (K)
card_no
bill_date
user_id
bill_id ASC (C)
product_id
user_name (K)
card_type
cart_qty
cart_price

Figure 4.11: Case 1 - Query 8

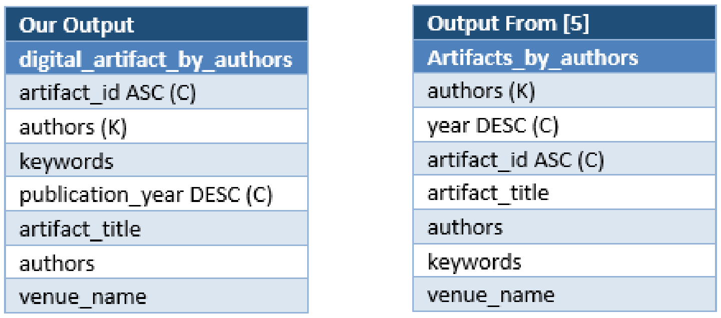
2) Results for Digital Library System:



Our Output
digital_artifact_by_venue
artifact_id ASC (C)
venue_name (K)
keywords
publication_year DESC (C)
artifact_title
authors

Output From [5]
Artifacts_by_venue
venue_name (K)
year DESC (C)
artifact_id ASC (C)
artifact_title
authors
keywords

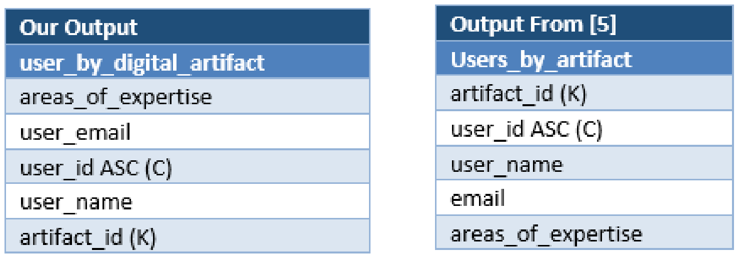
Figure 4.12: Case 2 - Query 1



Our Output
digital_artifact_by_authors
artifact_id ASC (C)
authors (K)
keywords
publication_year DESC (C)
artifact_title
authors
venue_name

Output From [5]
Artifacts_by_authors
authors (K)
year DESC (C)
artifact_id ASC (C)
artifact_title
authors
keywords
venue_name

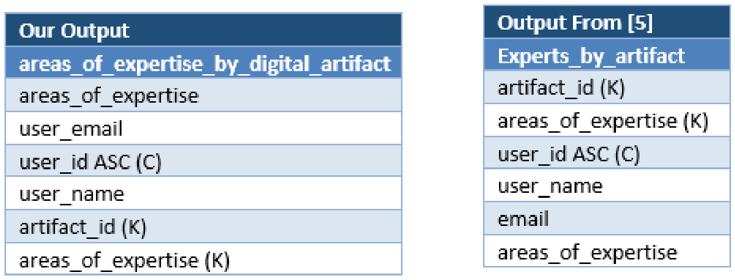
Figure 4.13: Case 2 - Query 2



Our Output
user_by_digital_artifact
areas_of_expertise
user_email
user_id ASC (C)
user_name
artifact_id (K)

Output From [5]
Users_by_artifact
artifact_id (K)
user_id ASC (C)
user_name
email
areas_of_expertise

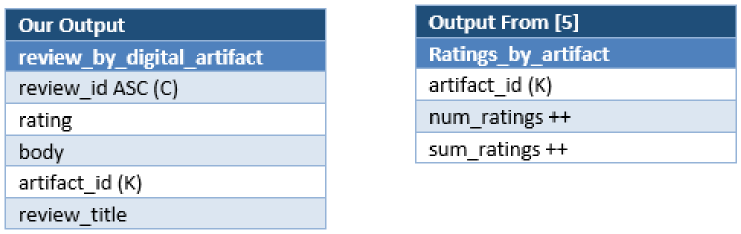
Figure 4.14: Case 2 - Query 3



Our Output
areas_of_expertise_by_digital_artifact
areas_of_expertise
user_email
user_id ASC (C)
user_name
artifact_id (K)
areas_of_expertise (K)

Output From [5]
Experts_by_artifact
artifact_id (K)
areas_of_expertise (K)
user_id ASC (C)
user_name
email
areas_of_expertise

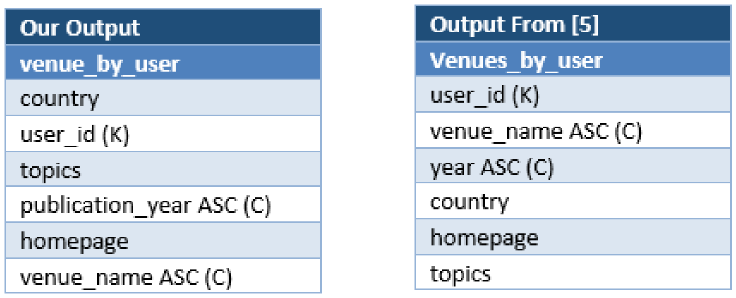
Figure 4.15: Case 2 - Query 4



Our Output
review_by_digital_artifact
review_id ASC (C)
rating
body
artifact_id (K)
review_title

Output From [5]
Ratings_by_artifact
artifact_id (K)
num_ratings ++
sum_ratings ++

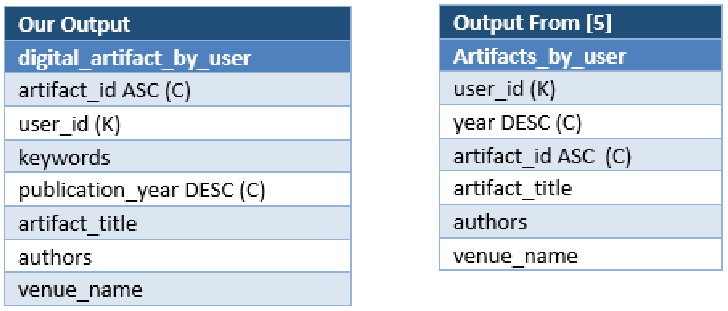
Figure 4.16: Case 2 - Query 5



Our Output
venue_by_user
country
user_id (K)
topics
publication_year ASC (C)
homepage
venue_name ASC (C)

Output From [5]
Venues_by_user
user_id (K)
venue_name ASC (C)
year ASC (C)
country
homepage
topics

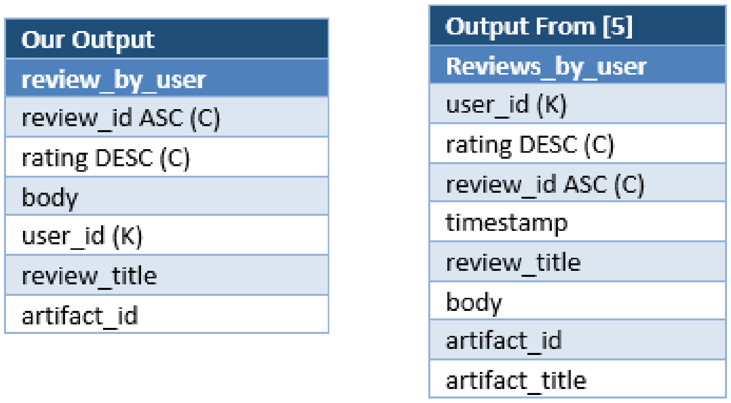
Figure 4.17: Case 2 - Query 6



Our Output
digital_artifact_by_user
artifact_id ASC (C)
user_id (K)
keywords
publication_year DESC (C)
artifact_title
authors
venue_name

Output From [5]
Artifacts_by_user
user_id (K)
year DESC (C)
artifact_id ASC (C)
artifact_title
authors
venue_name

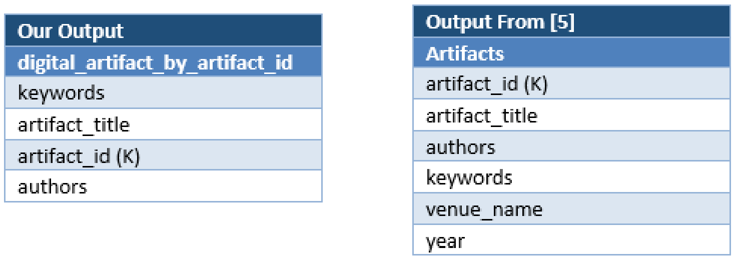
Figure 4.18: Case 2 - Query 7



Our Output
review_by_user
review_id ASC (C)
rating DESC (C)
body
user_id (K)
review_title
artifact_id

Output From [5]
Reviews_by_user
user_id (K)
rating DESC (C)
review_id ASC (C)
timestamp
review_title
body
artifact_id
artifact_title

Figure 4.19: Case 2 - Query 8

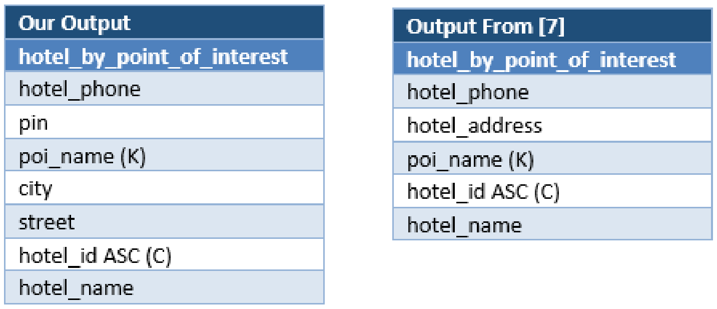


Our Output
digital_artifact_by_artifact_id
keywords
artifact_title
artifact_id (K)
authors

Output From [5]
Artifacts
artifact_id (K)
artifact_title
authors
keywords
venue_name
year

Figure 4.20: Case 2 - Query 9

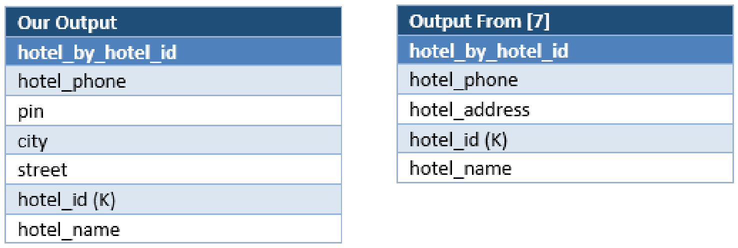
3) Results for Hotel Management System:



Our Output
hotel_by_point_of_interest
hotel_phone
pin
poi_name (K)
city
street
hotel_id ASC (C)
hotel_name

Output From [7]
hotel_by_point_of_interest
hotel_phone
hotel_address
poi_name (K)
hotel_id ASC (C)
hotel_name

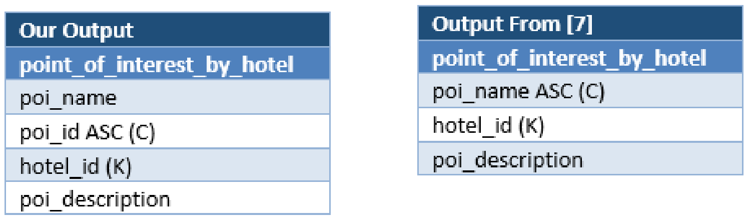
Figure 4.21: Case 3 - Query 1



Our Output
hotel_by_hotel_id
hotel_phone
pin
city
street
hotel_id (K)
hotel_name

Output From [7]
hotel_by_hotel_id
hotel_phone
hotel_address
hotel_id (K)
hotel_name

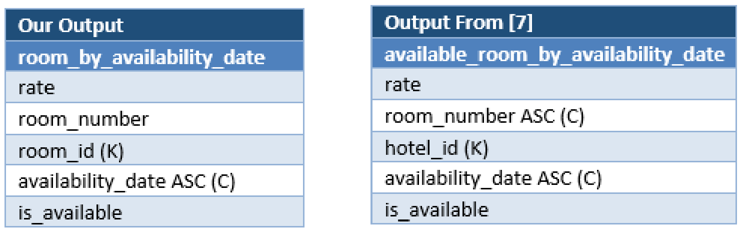
Figure 4.22: Case 3 - Query 2



Our Output
point_of_interest_by_hotel
poi_name
poi_id ASC (C)
hotel_id (K)
poi_description

Output From [7]
point_of_interest_by_hotel
poi_name ASC (C)
hotel_id (K)
poi_description

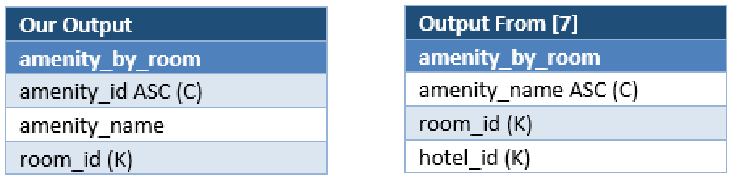
Figure 4.23: Case 3 - Query 3



Our Output
room_by_availability_date
rate
room_number
room_id (K)
availability_date ASC (C)
is_available

Output From [7]
available_room_by_availability_date
rate
room_number ASC (C)
hotel_id (K)
availability_date ASC (C)
is_available

Figure 4.24: Case 3 - Query 4



Our Output
amenity_by_room
amenity_id ASC (C)
amenity_name
room_id (K)

Output From [7]
amenity_by_room
amenity_name ASC (C)
room_id (K)
hotel_id (K)

Figure 4.25: Case 3 - Query 5

When the algorithm is executed for the above mentioned three applications, we get an output schema as shown in the result part of every application. In output, left part of every image represents outcome of the algorithm and right part represents an actual output. There are some differences in the actual output and algorithm's output schema for few queries and reason for them are as follows:

- An actual output considers an attribute of relationship and some more attributes of entity/s which come in join path of source and destination entity. our result considers only prime attributes of it.

- An actual output may consider some attributes based on human intelligence which algorithm doesn't consider.

- For actual output, some entities are considered as an weak entity from an ER diagram, but we have considered it as an independent entity because of which attributes list may vary.

Chapter 5
Conclusion And Future Work

Through this work, we devise and implement an algorithm that can be used to automate the schema design for Cassandra database. The algorithm takes two inputs ER diagram and application queries. The inputs are represented and stored in form of relational tables. Devised algorithm is executed and tested for three different applications. Results which we get on executing this algorithm on test cases are found to be correct. We hope, this algorithm is able to reduce the human work required to derive a schema for Cassandra database up to a certain extent by providing schema in an automated way for the application queries of particular application. As a part of future work, one can expand the scope of this work to automate the evaluation process of schema generated for the application queries. Apart from that, inputs to the algorithm can be captured from GUI where ER can be read from diagram and query can be read in SQL form.

References

[1] Datastax documentation. https://docs.datastax.com/en/home/docs/index.html. Accessed: 2023-06-02.

[2] What is cassandra? https://www.spiceworks.com/tech/bigdata/articles/what-is-cassandra/. Accessed: 2023-05-29.

[3] Conceptual data modeling. https://cassandra.apache.org/doc/latest/cassandra/data_modeling/data_modeling_conceptual.html, 2009. Accessed: 2023-05-29.

[4] A. Chebotko, A. Kashlev, and S. Lu. A big data modeling methodology for apache cassandra. In *2015 IEEE International Congress on Big Data*, pages 238–245, New York, NY, USA, 2015.

[5] R. Edwards. Exploring common apache cassandra use cases. https://www.datastax.com/blog/exploring-common-apache-cassandra-use-cases. Accessed: 2023-05-29.

[6] A. Lakshman and P. Malik. Cassandra: A decentralized structured storage system. *Operating Sys. Review*, 44(2):35–40, 2010.

[7] M. J. Mior, K. Salem, A. Aboulnaga, and R. Liu. Nose: Schema design for nosql applications. *IEEE Transactions on Knowledge and Data Engineering*, 29(10):2275–2289, Oct. 2017.