

Towards Automating Database Designing

Heli Helskyaho
 Miracle Finland Oy
 Helsinki, Finland
 heli.helskyaho@miracleoy.fi

Abstract—Database designing is an important process for enabling good quality data. Without designing the database correctly, the database might contain the same data several times, or it might contain data that is not usable for decision making. The evolution of software development, programming languages, increasing amount of data, different data models, different data sources and many more have increased the importance of designing databases to provide accurate data for decision making. Designing databases manually is time consuming. If the process can be automated, it would allow faster creation of good quality databases.

The goal of this study is to investigate whether large language models could be used for designing a Data Vault 2.0 raw database to automate the designing process. In this study we introduce database designing as a process, and describe the main principles of Data Vault 2.0. We create an example data source, an example Data Vault 2.0 raw database based on the source database for reference, and then test the ChatGPTs capabilities for creating a Data Vault 2.0 raw database based on instructions given in a prompt. Finally, we analyze the results and discuss future works.

I. INTRODUCTION

The relational database management system (RDBMS) has been de facto as a data model solution for decades, and for a good reason: RDBMS is based on a solid theory and therefore gives a standardized environment for storing and retrieving data [1], [2]. The evolution of software development and programming languages has brought new ideas on storing and retrieving data to make the interaction of a programming language and the database more fluent. First the objects were introduced in programming languages giving pressure to create object-oriented database management systems (OODBMS), a database where an object can be stored as it is and called easily by an object-oriented programming language [3]. Later these features were added to a RDBMS making it an object-relational database management system (ORDBMS) [4], [5]. The next need in the software development was to be able to interchange data between applications in a standard way, and Extensible Markup Language (XML) was defined along with the XML database management systems [6]. Then, JSON was created to be a lighter version of XML and easily used by JavaScript programs or REST API Calls. Document store NoSQL databases were introduced to store JSON data natively. Later both XML and JSON were fully supported by RDBMS and eventually all the other data model needs have been implemented in RDBMS.

There are also other so-called NoSQL databases [7] created for specific data model purposes, for example key-value pair,

column-oriented, or graph. A key-value pair consists of two related data elements: a key to identify the data and a value, which is the actual data. A key-value store is considered as semi-structured data, a combination of structured and unstructured data. The key is structured and value (text) is unstructured. A column-oriented, or a columnar store, stores data in columnar structures instead of rows. A graph data model stores data as relationships. All these new features have demanded enlargements to SQL language to enable inserting and retrieving all these data models to and from the RDBMS.

Not only the evolution of software development and programming languages, but also the evolution of data has given its demands for data stores:

- growth of amount of data (volume)
- various formats of data (variety), including for example spatial or temporal data
- the speed of change of data, and the need to analyze streaming data (velocity).

Also, due to various sources of data, including sources outside the company domain, additional steps for database designing might be needed because of the changes in the nature of the data:

- the uncertainty of the data (veracity)
- the value of the data (value)
- the changing meaning of the data (variability)
- the need to validate that hypothesis before taking further action (viability)
- the need to be able to present the data in a manner that is readable and accessible (visualization).

The evolution of processing power and algorithms has made it possible to analyze large amounts of various kinds of data in a short time to give the business more information on their data. Therefore, it is even more important to be able to store all the data in a form it can be used. If the data is saved without the design process it cannot be used, because nobody knows what the data is about, how it is related to other data etc.: the data is stored but it cannot be used. With the current amount of data, different data models, different data sources etc. it is no longer possible to design databases only manually. The demand for efficiency requires machines to do as much work as possible leaving only a fraction of the work for humans to do.

To be able to use the data, there are also other requirements. All data, despite its location or format, should be able to be queried with one single language using one single frontend [8].

Without this capability we will not be able to fulfill the need of efficient data processing in retrieving the data. One key element is that the various data must be able to identify how they are related to each other, similar to a system as foreign keys in RDBMS.

The data must be identified by its label (first name = fname), and by its content (John Doe = Doe John). Sometimes the data must be transformed before saving. Some data might be missing, some data might need to be converted to another format so it will be aligned with the data from other data sources. The data types must be defined correctly before storing the data. For instance, dates are typically managed differently in different kinds of data sources.

Because all these steps are complex and varying depending on the data source and the data itself, the steps might be impossible to define in static programming, therefore machine learning could be a solution.

How to automate the database designing? One option to solve the problem would be using mathematics, for example set theory and category theory. But that approach requires the process to be defined in detail and implemented using a programming language. The process of database designing is complex, and often unstructured. When two database designers design the same database, the result is rarely exactly the same. The different solutions for the same problem can be explained for example by the knowledge and experience on database designing, the domain understanding, and the problem-solving skills of the designer. Part of the database designing process is commonsense reasoning. That part of the work is difficult to get implemented using traditional programming techniques. Since automating the database designing is important, there are several attempts to solve it. Knowledge-based approaches have been suggested to automate the design process to include the domain knowledge, the human user experience and commonsense to the process. Suggestions include for example ontologies, semantics, expert systems and forms [9], [10], [11], [12], [13], [14], [15], [16]. These approaches have not been successful since there still is no tool to automate the process.

Could Large Language Models (LLMs) be the solution for automating the database designing process? LLMs, as being Generative AI, produce innovative outcomes that are similar to the creative part of a designer's work. Could LLMs be used to automate the designing process of a Data Vault 2.0 database?

In our study, we conducted tests using ChatGPT 3.5 to provide DDLs for creating a Data Vault 2.0 database. Our study consisted of the following:

- We defined a source database.
- We manually created a Data Vault raw database as a reference.
- We used ChatGPT to generate the DDLs for creating a Data Vault database.
- We investigate the results against the reference database and the Data Vault 2.0 methodology.

The focus of this paper is the conceptual and logical design process for relational data and how that can be automated.

There are several tools that are able to generate the physical models based on the logical model, therefore automating the creation of a logical model is more vital.

The remainder of this paper is organized as follows. Section II introduces the process of database designing and its complexity. Section III introduces Data Vault 2.0 methodology and the principles of a Data Vault 2.0 raw database. Section IV explains the tests made in understanding the potential of ChatGPT 3.5 in automating Data Vault 2.0 raw database designing process. In section V we discuss the validity of the chosen approach. Finally, in section VI we conclude the paper and discuss the future work.

II. DATABASE DESIGNING

A. The process of designing a database

In database design it is determined what data will be stored, where, and using what data model. There are several different data models and the correct choice depends on the nature and use of the data. The data models available depend on the Database Management System (DBMS). Examples of data models are relational, columnar, document, spatial, or graph. A DBMS supporting several data models is called a multi-model database [8].

A typical database designing process consists of four phases: requirement analysis, conceptual design, logical design, and physical design [17]. In requirement analysis the main target is finding and analyzing the requirements the future end users have for the data and the system. The result of this phase is specification of user requirements for data, functionality, security, performance and so on. The next phase is conceptual design during which those requirements will be interpreted into a formal presentation: the conceptual model and process models. In logical design the conceptual model is transformed into a logical data model that the target DBMS understands. For instance, in RDBMS that would be a logical schema, or in NoSQL maybe a JSON schema, a Key-value pair definition, column store definition, or the graph structures: the nodes, edges, and properties. This is the phase to understand which data model would fit the best and to design a logical model for it. In relational data model the tool for logical design is an Entity-Relationship (ER) diagram. In the ER diagram the goal is to identify entities and how they are related to each other. It is also important to discover attributes of these entities and define the correct data types for them. If an attribute can be defined to be mandatory, the best practice is to do so. Missing values or wrong data types in the attributes will lead to poor data quality. The physical design depends on the selected database technology, and requires excellent knowledge on that. The physical design includes, for example, how the schema objects are implemented on disks, or what kind of indexes are used for data retrieval optimization.

B. The process of adding new data sources to an existing database

Designing a new database is a complex process but adding new data sources to an existing database is even more

complicated because the process should also include analyzing and comparing the existing data structures to the new ones. There are two high level questions when adding a new data source: does the structure for storing the data exist, and does the data or part of it already exist in the database.

Data mapping is the process of mapping data from one dataset to another dataset. Data mapping can be divided into two categories: mapping the metadata (Entity matching) and mapping the data (Content mapping). Entity matching includes mapping the entity types and data labels. Mapping of entity types includes discovering if the entity type already exists in the database. To do that, we identify the entity type and discover if that entity type (for instance a Customer, or an Order) exists in the database.

If the entity type exists in the database:

- We need to decide if the new data will be stored in the existing structure, or if we need to use a different data model and somehow link it to the existing data.
- If an existing structure is chosen, we need to verify it has all attributes needed. If all attributes do not exist, we need to decide how the new attributes will be stored.

If the entity type does not exist in the database:

- We need to decide how and in what format the data will be stored. For instance, if the new data arrives in JSON, will we store it as JSON, relational or graph.
- We need to discover how this entity type is related to other entity types and how we want to implement a relationship between these entity types.

Mapping of the data labels identifies the same data being labeled differently in data sources. For instance, in one data source the first name of the customer might be called FirstName, in another First Name, and in the third data source it might be FName. When storing the data, the system should understand that these are labels for the same data.

The next step is content matching. We identify each entity on the new dataset to understand if that particular entity (Customer John Doe) already exists in the database. Identifying an entity is not an easy task because not all datasets have a natural identifier that could be used for identifying. The identifying process can be difficult and might include data transformations, for example J. Doe -> John Doe, Doe John -> John Doe.

A dataset can include categorical data. Categories can be ordinal or nominal. The difference is that ordinal categories can be ordered, whereas nominal categories do not have ordering. An example of an ordinal category is school evaluations from A to D, and examples of nominal categories are colors or zip codes. Each data source might have its own categories for the same data and storing those different

categories as they are would cause a mapping challenge when trying to use the data. If we store data as it is, we need to do the mapping of different categories each time we query data, instead of doing it only once when storing it. Harmonizing the categories while saving the data is recommended.

When the database has been designed and all the mappings are done, it is time to insert data into the database. It makes no sense to store poor quality data, because that would not be usable for any decision making; in the worst case it could result in wrong decisions. Missing data and null values have always been a problem with data. If the value is null, you do not know what it means: the data does not exist, it is not available at the moment, the user did not enter it for some reason, you do not have access to that data, or something else. If the data has no meaning at all, it is possible we will not store it at all. If we can guess the value for the missing data using defaults or estimation algorithms, we might add the value using imputation. If we do not know what to do with it, we might store it as it is. The decision depends on the data and the insight we have on the data.

Multicollinearity means that one or several attribute/-s in the dataset is/are dependent on another attribute/-s on the dataset. This might affect the way we treat those attributes in data discovery and it is a feature that must be recognized by the process. There are several other challenges with the data. For example:

- The data has inconsistent values. For instance, a phone number in the field that should have the email address. Some of these can be checked using data type checking: all values in that column should be of type something, for example number. This technique does not reveal all the problematic data.
- The dataset contains duplicate values and duplicate data. That can usually be detected by defining a meaningful identifier that has the same value for all equal data rows and would reveal the duplicate data.
- The data is too detailed. It might be the data is restricted, confidential, or private and because of that nature it can never be used as it is. In these cases, the data needs to be aggregated or anonymized before saving.
- The data in the dataset is stored in a way that it will not be usable in the future and it needs to be transformed to something more useful. For example, Age would not be the same after a year or two, so instead of storing the age, it might be useful to convert it to the year of birth.
- The data type used in the dataset is completely wrong or not in the same format as the other data in the target database. Typically, the data type for dates is something that needs to be checked. Also, decimals, character sets, and data length are worth checking.

The nature of the data structure affects how it can be discovered. For instance, structured data is handled differently from semi-structured or unstructured data, and time series are handled in its own way.

III. A DATA VAULT 2.0 DATABASE

Data Vault 2.0 is a methodology for implementing large scale analytic solutions and data warehouses. Part of the methodology defines standards and best practices for the logical and physical database design [18], [17]. One of the foundational principles of the Data Vault 2.0 methodology is that new data sources can be added with minimal to zero impact on the existing solution and the applications using it. A Data Vault 2.0 logical data architecture may consist of several different physical or virtual databases or layers. It includes a staging area for highly performant source-to-stage loading processes, a raw data warehouse layer for all the auditable, historical data stored over time, and layers for business users to see and use the data. In this paper our focus is on the raw data vault layer.

In the raw data vault layer, there are three different kinds of entities: Hubs, Links, and Satellites. A Hub is defined to be a unique list of business keys. A business key is the identifier the business uses for selecting a specific row in an entity. Preferably the business key is not a surrogate key. Each Hub entity represents a business concept, such as Customer, Employee, Facility or Product. A Link is defined to be a unique list of relationships, associations, events, or transactions between two or more business keys. Satellites include delta-driven descriptive information; the data that changes over time. A raw data vault layer is designed primarily using these three elements. There might also be some special entities, for example Stand-Alone tables, or Reference tables. Data Vault 2.0 documentation covers the guidelines and recommendations for all these entities, also including solutions for several modeling problems. Let's observe some guidelines for the three main entity types: Hubs, Links, and Satellites.

A. A Hub

1) *The business key*: a Hub must have at least one business key driven attribute.

2) *Relationship to Satellites*: The description, slowly changing historical data about the Hub's business key.

3) *Technical attributes*: A Hub must include some technical attributes for the loading process and for the auditability. Examples of these attributes are the Load Date Time Stamp and the Record Source.

B. A Link

1) *The intersection*: A Link is a m:n relationship between two or more Hub business keys. A Link can never depend on another Link, and a Link never includes any temporality regarding the start or end dates of the relationship that exists between the business keys

2) *Technical attributes*: A Link must include some technical attributes, for example Load Date and Record Source.

C. A Satellite

1) *Satellites are associated with*: Hubs or Links.

2) *A Satellite hosts the descriptive, time variant content*: the data about a Hub or a Link.

3) *Technical attributes*: A Satellite must include some technical attributes for the loading process and for the auditability. Examples of these attributes are Load Date Time Stamp and Record Source. Optional attributes include for example Hash Difference which helps in detecting if the descriptive data has changed and determining if a new row should be inserted.

IV. DESIGNING A DATA VAULT 2.0 DATABASE USING CHATGPT

We chose Data Vault 2.0 methodology and its raw database as the test database to be designed. The reason for this is that Data Vault 2.0 has commonly known principles that have been documented and are available on the internet. We chose LLMs to generate Data Definition Language (DDL) commands for creating a Data Vault 2.0 raw database based on DDL commands of the source database and other guidance given in a prompt. A prompt is the input to the LLM to stimulate the generation of a response. The process of designing and optimizing prompts to achieve better outcomes is called Prompt engineering. ReAct (Reasoning and Acting), as a prompting framework, allows LLMs to generate reasoning traces and actions specific to the given task. Once an LLM generates an action, the action can be carried out externally, and the output of the action can be included in the next prompt as further information to the LLM. A token is the atomic unit for LLM to represent its input and output. Tokens are enumerations of words, characters, or other segments of text or programming code.

We tested creating a Data Vault 2.0 database with the free LLM based tool called ChatGPT that is using GPT-3. The first test is to create a Data Vault 2.0 raw database using the DDLs of the source database. The simple source database is shown in Fig. 1. It consists of four tables: Customer, Product, Order and Orderline. The database is designed using Oracle SQL Developer Data Modeler, and the DDLs are generated based on the design.

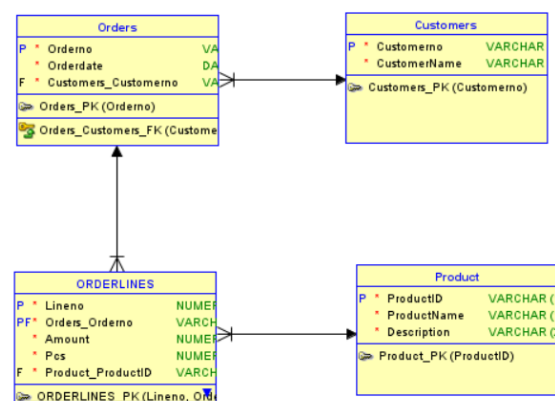


Fig. 1. The source database

We manually created a reference Data Vault 2.0 raw database for the source database. The expected result of a Data Vault 2.0 raw database is shown in Fig. 2.

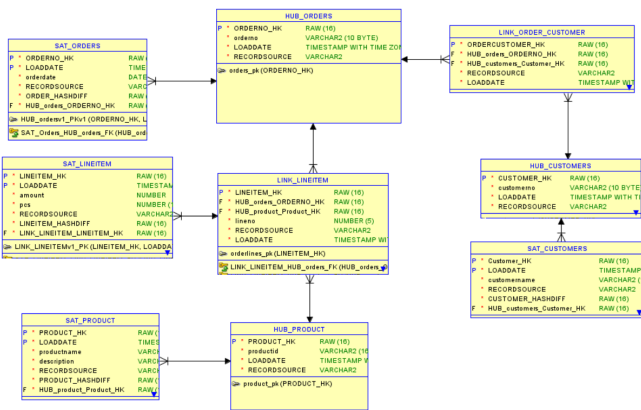


Fig. 2. An expected Data Vault 2.0 raw database design for the data source

To generate the DDLs for creating the Data Vault 2.0 raw database we prompted ChatGPT with the DDLs of the source database.

The DDLs for the database are following:

```
CREATE TABLE Customers (customerno VARCHAR2 (10) NOT NULL,
customername VARCHAR2 (100) NOT NULL)
TABLESPACE datal LOGGING;

ALTER TABLE Customers ADD CONSTRAINT customers_pk
PRIMARY KEY (customerno);

CREATE TABLE orderlines (lineno NUMBER (5) NOT NULL,
orders_orderno VARCHAR2 (10) NOT NULL, amount NUMBER
(10, 2) NOT NULL, pcs NUMBER (10) NOT NULL,
product_productid VARCHAR2 (16) NOT NULL) LOGGING;

ALTER TABLE orderlines ADD CONSTRAINT orderlines_pk
PRIMARY KEY (lineno, orders_orderno);

CREATE TABLE orders (orderno VARCHAR2 (10) NOT NULL,
orderdate DATE NOT NULL, customers_customerno
VARCHAR2 (10) NOT NULL) LOGGING;

ALTER TABLE orders ADD CONSTRAINT orders_pk PRIMARY
KEY (orderno);

CREATE TABLE product (productid VARCHAR2 (16)
NOT NULL, productname VARCHAR2 (100) NOT NULL,
description VARCHAR2 (200) NOT NULL) LOGGING;

ALTER TABLE product ADD CONSTRAINT product_pk
PRIMARY KEY (productid);

ALTER TABLE orderliness ADD CONSTRAINT
orderlines_orders_fk FOREIGN KEY (orders_orderno)
REFERENCES orders (orderno) NOT DEFERRABLE;

ALTER TABLE orderliness ADD CONSTRAINT
orderlines_product_fk FOREIGN KEY (product_productid)
REFERENCES product (productid)
NOT DEFERRABLE;

ALTER TABLE orders ADD CONSTRAINT
orders_customers_fk FOREIGN KEY (customers_customerno)
REFERENCES customers
(customerno) NOT DEFERRABLE;

ALTER TABLE orderliness ADD CONSTRAINT
orderlines_orders_fk FOREIGN KEY (orders_orderno)
REFERENCES orders (orderno) NOT DEFERRABLE;

ALTER TABLE orderliness ADD CONSTRAINT
orderlines_product_fk FOREIGN KEY (product_productid)
REFERENCES product (productid)
NOT DEFERRABLE;

ALTER TABLE orders ADD CONSTRAINT
orders_customers_fk FOREIGN KEY
```

```
(customers_customerno) REFERENCES customers
(customerno) NOT DEFERRABLE;
```

When this set of DDLs was given to ChatGPT in May 2023, with a prompt “Please generate the DDLs for the target following the Data Vault 2.0 methodology.”, the outcome was a set of DDLs that did not have correct column names; they were called attribute1, attribute2 etc. When prompting ChatGPT to correct the column naming, the result was better, but not good. With further prompting the column names were created correctly but many other flaws still existed. The diagram created based on the DDLs ChatGPT provided is shown in Fig. 3.

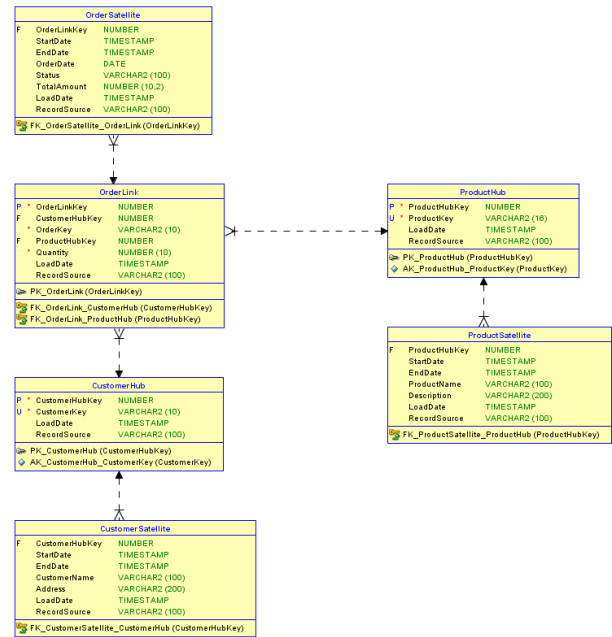


Fig. 3. The Data Vault 2.0 raw database created in May 2023 using DDLs generated by ChatGPT

Another attempt was made in September 2023. Then, ChatGPT was able to perform better. We prompted it with the same prompt as before.

This time ChatGPT replies:

“Data Vault 2.0 methodology involves modeling the data warehouse in a specific way to enable better flexibility and scalability in managing data. To generate DDLs for a target database following the Data Vault 2.0 methodology, you would typically create the following structures: Hubs, Links, and Satellites. Below, I'll provide DDLs for tables representing these structures based on your existing tables.

Hubs - These are business key entities.

```
-- Hubs
CREATE TABLE hub_customers (hub_customerno
VARCHAR2 (10) BYTE) NOT NULL, record_source
VARCHAR2 (100), load_date DATE, record_hash
VARCHAR2 (64), CONSTRAINT hub_customers_pk PRIMARY
KEY (hub_customerno));

CREATE TABLE hub_orders (hub_orderno
VARCHAR2 (10) BYTE) NOT NULL, record_source
```

```

VARCHAR2(100),    load_date    DATE,    record_hash
VARCHAR2(64), CONSTRAINT hub_orders_pk PRIMARY KEY
(hub_orderno));

```

```

CREATE TABLE hub_product (hub_productid
VARCHAR2(16) NOT NULL, record_source
VARCHAR2(100), load_date DATE, record_hash
VARCHAR2(64), CONSTRAINT hub_product_pk PRIMARY KEY
(hub_productid));

```

Links - These represent relationships between hubs.

```

-- Links
CREATE TABLE link_customer_order (link_id
NUMBER GENERATED BY DEFAULT AS IDENTITY,
hub_customerno VARCHAR2(10 BYTE) NOT NULL,
hub_orderno VARCHAR2(10 BYTE) NOT NULL,
record_source VARCHAR2(100), load_date DATE,
record_hash VARCHAR2(64), CONSTRAINT
link_customer_order_pk PRIMARY KEY (link_id),
CONSTRAINT link_customer_order_customer_fk FOREIGN
KEY(hub_customerno) REFERENCES
hub_customers(hub_customerno), CONSTRAINT
link_customer_order_order_fk FOREIGN KEY
(hub_orderno) REFERENCES hub_orders(hub_orderno));

```

```

CREATE TABLE link_order_product (link_id
NUMBER GENERATED BY DEFAULT AS IDENTITY,
hub_orderno VARCHAR2(10 BYTE) NOT NULL,
hub_productid VARCHAR2(16) NOT NULL,
record_source VARCHAR2(100), load_date DATE,
record_hash VARCHAR2(64), CONSTRAINT
link_order_product_pk PRIMARY KEY (link_id),
CONSTRAINT link_order_product_order_fk FOREIGN KEY
(hub_orderno) REFERENCES hub_orders(hub_orderno),
CONSTRAINT link_order_product_product_fk FOREIGN KEY
(hub_productid) REFERENCES
hub_product(hub_productid));

```

Satellites - These tables store descriptive and historical data.

```

-- Satellites
CREATE TABLE sat_customer (hub_customerno
VARCHAR2(10 BYTE) NOT NULL, sat_record_source
VARCHAR2(100), sat_load_date DATE, sat_record_hash
VARCHAR2(64), customername VARCHAR2(100) NOT NULL,
CONSTRAINT sat_customer_pk PRIMARY KEY
(hub_customerno), CONSTRAINT sat_customer_hub_fk
FOREIGN KEY (hub_customerno) REFERENCES
hub_customers(hub_customerno));

```

```

CREATE TABLE sat_order (hub_orderno
VARCHAR2(10 BYTE) NOT NULL, sat_record_source
VARCHAR2(100), sat_load_date DATE,
sat_record_hash VARCHAR2(64), orderdate DATE NOT
NULL, customers_customerno VARCHAR2(10 BYTE) NOT
NULL, CONSTRAINT sat_order_pk PRIMARY KEY
(hub_orderno), CONSTRAINT sat_order_hub_fk FOREIGN
KEY (hub_orderno) REFERENCES
hub_orders(hub_orderno), CONSTRAINT
sat_order_customer_fk FOREIGN KEY
(customers_customerno) REFERENCES
hub_customers(hub_customerno));

```

```

CREATE TABLE sat_product (hub_productid
VARCHAR2(16) NOT NULL, sat_record_source
VARCHAR2(100), sat_load_date DATE, sat_record_hash
VARCHAR2(64), productname VARCHAR2(100) NOT NULL,
description VARCHAR2(200) NOT NULL, CONSTRAINT
sat_product_pk PRIMARY KEY (hub_productid),
CONSTRAINT sat_product_hub_fk FOREIGN KEY
(hub_productid) REFERENCES
hub_product(hub_productid));

```

Please note that Data Vault modeling can vary depending on your specific requirements and business

rules. These DDLs represent a basic structure following the Data Vault 2.0 methodology. You may need to adapt them to your specific needs, and consider adding additional columns for tracking lineage, data quality, and more, as required by your organization.”

The design created by ChatGPT in September 2023 is shown in Fig. 4.

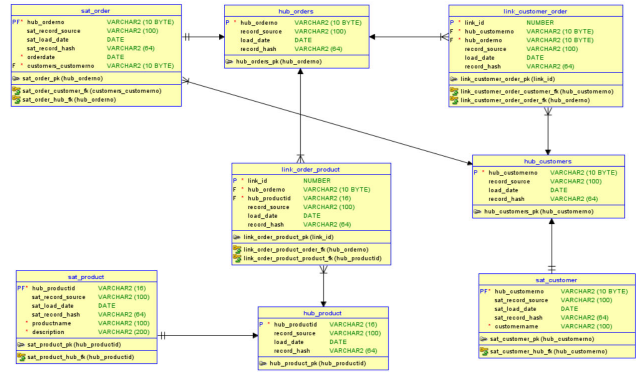


Fig. 4. A Data Vault 2.0 raw database designed by ChatGPT in September 2023

ChatGPT did not create an identical database to the reference database (Fig. 2). In Data Vault 2.0 there is a concept called a Dependent Child. This concept is for treating weak entities, for example Orderline in our example. ChatGPT did not handle this situation correctly: all the data about a weak entity Orderline is missing from the design. Instead, there is a foreign key reference between Hub_customer and Hub_order which is against the methodology. Also, the design is missing hash keys as primary keys.

For Hubs ChatGPT names the hash keys wrongly and they are of the wrong data type. A hash key should be of type binary. The technical columns should not be optional, and load date should be stored with timestamp, not only date. In Oracle the data type date holds the time element, but in general it is better to define load date including the time. All Hubs have a column called record_hash. If that is the hash difference (hashdiff) defined in the Data Vault methodology, it should not be in a hub and it should be of type binary.

For Links ChatGPT has similar issues: naming conventions, technical columns defined as optional, and load date defined as date, not as timestamp. Links should not have a column record_hash. The definition for the primary key (link_id NUMBER GENERATED BY DEFAULT AS IDENTITY) is wrong. The primary key should be a hash-based value of the business key of each hub.

For Satellites the most crucial mistake is the primary key. It has been defined wrongly and would not allow storing history data as Satellites should. The primary key of a Satellite should be a combination of hash key and load date, the minimum. The sat_record_hash column that assumably is the hash difference, should be of type binary, not varchar. The technical columns should be mandatory, and the load date should include the timestamp.

The design for the Data Vault database ChatGPT generated was not correct. ChatGPT advises: “Please note that Data Vault modeling can vary depending on your specific requirements and business rules. These DDLs represent a basic structure following the Data Vault 2.0 methodology. You may need to adapt them to your specific needs, and consider adding additional columns for tracking lineage, data quality, and more, as required by your organization.”. Based on our tests the adaptation required is more demanding than creating the design manually. Therefore, it is important to get ChatGPT to generate as correct DDLs as possible.

Since LLMs are guided by prompting, we tested prompting ChatGPT with these details:

“Please generate the DDLs for the target database following the Data Vault 2.0 methodology. Please note that ordelines is a dependent child in Data Vault 2.0. Also note that some of the technical columns in Data Vault 2.0 must be mandatory and load_date needs to be datetime. Also note that hash diff columns only exist in satellites. Hash keys must be of type binary. Hubs should not include descriptive data, that should be in a satellite. Hubs should include the mandatory technical columns, the business key and the hash key.”

With this prompt ChatGPT is able to define Hubs and Satellites correctly, but the hash keys are missing from Links and all foreign keys are missing. The dependent child has been solved almost correctly: there is only an extra link (link_orderline_orders) that should not exist.

We changed the prompting by adding a hint about foreign keys:

“Please generate the DDLs for the target database following the Data Vault 2.0 methodology. Please note that ordelines is a dependent child in Data Vault 2.0. Also note that some of the technical columns in Data Vault 2.0 must be mandatory and load_date needs to be datetime. Also note that hash diff columns only exist in satellites. Hash keys must be of type binary. Hubs should not include descriptive data, that should be in a satellite. Hubs should include the mandatory technical columns, the business key and the hash key. All tables should have foreign keys as defined in Data Vault 2.0”

As a result, ChatGPT was able to create the foreign keys and the dependent child structures correctly, but “forgot” that the Hubs should have both the hash key and a business key creating only one column as a mixture of those. It did not create a link for Orders and Customers.

Based on these tests, ChatGPT was able to generate DDLs that are syntactically correct and mostly follow the Data Vault 2.0 methodology, but it seems there are too many rules in Data Vault 2.0 for ChatGPT 3.5 to manage. Therefore, it does not follow all the rules given in the prompt.

V. DISCUSSION

ChatGPT was able to create syntactically correct DDLs for Data Vault 2.0 database generation but in every test round the DDLs were missing something. Since a model is as good as the data used to train it, one explanation for this is that the data from the internet that was used to train the LLM model has

enough correct data on DDLs but does not have enough data about Data Vault 2.0 methodology and/or has plenty of wrong practices of Data Vault 2.0. To get the model more accurate on DDLs for Data Vault 2.0 raw database designing we could use for example fine-tuning or Retrieval-Augmented Generation (RAG). Fine-tuning is a process to further train the model using smaller, more specific datasets to improve its performance on the specific target task or domain. On the other hand, RAG provides a way to optimize the output of an LLM with targeted, domain-specific information without modifying the LLM model by allowing it to use additional data resources without retraining. These additional data resources can be databases, documents (for example naming conventions or Data Vault 2.0 best practices), or any kind of data available. RAG uses a vector database that would bring an additional benefit: semantic search. Traditional search finds keywords while semantic search uses the meaning of a question and source data to retrieve more accurate results. The source of the information in the RAG’s vector database can be identified. Therefore, incorrect information in the RAG can be corrected or deleted.

Due to the nature of an LLM the same prompt can result in different results and the flaws of the DDLs cannot be predicted and corrected automatically. It would be possible to define the Temperature parameter, that affects the randomness of the generated content, to zero to limit the randomness and get the results nearly deterministic, but that would disable the creativity of the process. One of the reasons we chose LLM was that it can create new content in a similar way as a human designer does. Creativity is good but hallucination is a pervasive problem for LLMs. Hallucination means that the LLM can create fictitious output that is not based on any facts. Hallucination can be partly tackled with fine-tuning and RAG but it cannot be guaranteed there will be no hallucination. Because humans also make mistakes the manual process of database designing includes a verification process to guarantee the quality of the model. Similar process would be needed for automated design. This process could be manual or automated. We could use reinforcement learning from human feedback or from existing reference databases against which to benchmark output from the process and use the result to train the model further. Also, any automated testing techniques might be used to evaluate the result. Techniques for understanding what the model knows and how the prompt should be created [19] might be beneficial to improve the accuracy. There is a lot of research on LLM verification going on. Examples of the latest ideas are for example chain-or-verification [20] and self-verification [21].

The DDLs generated by ChatGPT do not include instructions on using the columns. For example, what should be stored in the hashdiff column? The methodology defines that the hashdiff should not include the business key column and it should always be generated using the same hash function and the same datatype. How can data and process governance be implemented in the process? For example, how can we be sure the same hash function is always used? Could the process governance be guided using RAG? The information for the process could be in tables or documents and the LLM model could read that. Could also the problem

with business keys and any business-related data be managed using RAG?

VI. CONCLUSION AND FUTURE WORK

Based on the tests, large language models (LLMs) seem to be a possible solution for automating Data Vault 2.0 raw database designing. There are several open questions before ChatGPT, or any LLM based tool, can be used to automate Data Vault database designing work. Examples of these questions are:

- How can a model be trained to follow **all** the best practices of Data Vault 2.0 methodology?
- Could using fine-tuning and/or Retrieval-Augmented Generation (RAG) guarantee a correct, or at least acceptable result?
- How to validate and measure the result of the process?
- Can LLMs solve the problem of existing data structures and existing data when adding a new data source? For example, would LLMs be able to split Satellites and integrate new data sources to an existing model?
- Can data and process governance be added to the process? For example, can business keys be generated correctly without assuming the primary key is always the business key, or could LLM also produce the instructions on using technical columns? Or could LLM automatically assure the same hash function is always used for the hash attributes?

Finding answers to these questions would be potential future work. Automating the database designing process would be important to allow efficient database designing processes and good quality data for decision making.

ACKNOWLEDGMENT

I would like to thank Cynthia Meyersohn from Data Vault Alliance for her support on verifying the best practices on Data Vault 2.0 in this paper.

I would also like to that professors Tomi Männistö and Laura Ruotsalainen from University of Helsinki for their help and guidance while writing this paper and in general.

REFERENCES

- [1] E.F. Codd, "A Relational Model of Data for Large Shared Data Banks", *Communications of the ACM*, Volume 13, Number 6, June, 1970.
- [2] C.J. Date, *Introduction to Database Systems*. 8th Edition, Pearson, 2003.
- [3] T. Atwood, "An Object-Oriented DBMS for Design Support Applications", *Proceedings of the IEEE COMPINT 85*, September 1985 pp. 299-307.
- [4] C.J Date, H. Darwen, *Databases, Types, and the Relational Model - The Third Manifesto* (Third Edition). Addison-Wesley, 2007
- [5] M. Stonebraker, D. Moore, *Object-Relational DBMSs: The Next Great Wave*. Morgan Kaufmann Publishers, 1996.
- [6] Jagadish, H., Al-Khalifa, S., Chapman, A. et al. TIMBER: "A native XML database", *VLDB 11*, 2002, pp 274–291.
- [7] A. Nayak, A. Poriya, D. Poojary, "Type of NOSQL Databases and its Comparison with Relational Databases", *International Journal of Applied Information Systems (IJ AIS)*, Foundation of Computer Science FCS, New York, USA, Volume 5– No.4, March 2013.
- [8] Z.H. Liu, J. Lu, D. Gawlick, H. Helskyaho, G. Pogossiants, Z. Wu, "Multi-model Database Management Systems - A Look Forward" *Heterogeneous Data Management, Polystores, and Analytics for Healthcare, DMAH Poly 2018*. Lecture Notes in Computer Science(), vol 11470. Springer, Cham.
- [9] K.L. Rouff, "CODES: A Database Design Expert System Prototype", *Proceedings of the 1st International Conference on Artificial Intelligence*, pp. 439–445, 1984.
- [10] A. Dogac, B. Yuruten, S. Spaccapietra, S., "A Generalized Expert System for Database Design", *IEEE Transactions on Software Engineering*, 15: 479–491, 1989.
- [11] M. Bouzeghoub, "Using Expert Systems in Schema Design", *Conceptual Modelling Databases and CASE*, pp. 465-487, 1992.
- [12] M. Lloyd-Williams, P. Beynon-Davis, "Expert Systems for Database Design: A Comparative Review", *Artificial Intelligence Rev.*, vol. 6, no. 3, pp. 263-283, 1992.
- [13] V.C. Storey R.C. Goldstein, "Knowledge-Based Approaches to Database Design", *Management Information Systems Quarterly*, vol. 17, no. 1, pp. 25-46, March 1993.
- [14] V.C. Storey, H. Ullrich, S. Sundaresan, "An Ontology to Support Automated Database Design", *Proc. 16th Int'l Conf. Conceptual Modeling (ER '97)*, 1997.
- [15] V.C. Storey, D. Dey, H. Ullrich, S. Sundaresan, "An Ontology-Based Expert System for Database Design", *Data and Knowledge Eng.*, vol. 28, no. 1, pp. 31-46, 1998.
- [16] V. C. Storey, R. C. Goldstein, H. Ullrich, "Naive semantics to support automated database design", *IEEE Transactions on Knowledge and Data Engineering*, vol. 14, no. 1, pp. 1-12, Jan.-Feb. 2002.
- [17] H. Helskyaho, *Oracle SQL Developer Data Modeler for Database Design Mastery*. McGraw Hill, 2015.
- [18] Data Vault Alliance official website, Data Vault 2.0 Data Modeling Specification v2.0.4, Web: <https://datavaultalliance.com/news/data-vault-2-0-data-modeling-specification-v2-0-4/>
- [19] Z. Jiang, F. F. Xu, J. Araki, G. Neubig, "How Can We Know What Language Models Know?", *Transactions of the Association for Computational Linguistics 8*, 2020, pp 423–438.
- [20] S. Dhuliawala, M. Komeili, J. Xu, R. Raileanu, X. Li, A. Celikyilmaz, J. Weston, "Chain-of-Verification Reduces Hallucination in Large Language Models", September 2023.
- [21] Y. Weng, M. Zhu, F. Xia, B. Li, S. He, K. Liu, J. Zhao, "Large Language Models are Better Reasoners with Self-Verification", May 2023.