Verification of Relational Database Languages Codes Generated by ChatGPT

Putsadee Pornphol
Department of Digital Technology, Faculty of science and Technology, Phuket Rajabhat University
putsadee.p@pkru.ac.th

Suphamit Chittayasothorn
School of Engineering, and School of Information Technology, King Mongkut's Institute of Technology Ladkrabang
suphamit.ch@kmitl.ac.th

ABSTRACT

The potential of using large language model artificial intelligence systems to generate program codes for application development is significant. Database codes in SQL (Structured Query Language), which is the standard relational database language, can be generated by such systems. Generative AI systems know database languages syntax through their training data and the text patterns from various sources that include SQL queries and related text. Thus, the generated codes may not be perfect and need verification before usage. This paper verifies the relational completeness of the SQL codes generated by ChatGPT, one of the most widely used large language model systems. Relational algebra operators are used for the relational complete verification. An equivalent relational calculus statement is generated for each SQL and relational algebra statement. The results confirmed that ChatGPT has the ability to generate relational complete SQL, relational algebra, and relational calculus codes.

CCS CONCEPTS

• **Information systems** → Data management systems; Query languages; Relational database query languages; • **Computing methodologies** → Artificial intelligence; Natural language processing; Natural language generation.

KEYWORDS

ChatGPT, SQL, Relational Algebra, Relational Calculus

ACM Reference Format:
Putsadee Pornphol and Suphamit Chittayasothorn. 2023. Verification of Relational Database Languages Codes Generated by ChatGPT. In *2023 4th Asia Service Sciences and Software Engineering Conference (ASSE 2023), October 27–29, 2023, Aizu-Wakamatsu City, Japan.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3634814.3634817

1 INTRODUCTION

The emergence of large language models [1, 2] is considered a breakthrough that starts a new era of human-computer interaction and communication. These models, such as ChatGPT [3], are the results of many years of research and innovation. ChatGPT is one of the most well-known large language models. It is trained on a large number of textual sources and possesses the ability to generate relevant responses. ChatGPT finds various applications and assists human user interactions. One of the useful applications includes code generations, including the Structured Query Language (SQL) codes [4, 5].

ChatGPT can generate SQL code. It is a large language model that is trained on a massive dataset of text and code, including SQL codes. SQL code can be generated by the following steps: First, ChatGPT first needs to understand the query that we want it to generate. This includes understanding the tables that we want to query, the columns that we want to select, and the conditions that we want to apply. Then, it generates the SQL code. Data manipulation as well as retrieval statements can be generated.

ChatGPT knows SQL syntax through its training data and the patterns it has learned from various sources that include SQL queries and related text. During the pre-training phase of its training process, ChatGPT is exposed to a wide range of text data from the internet, books, articles, technical documentation, and more. This data includes examples of SQL queries, discussions about databases, and related content.

ChatGPT's exposure to a diverse range of SQL-related text allows it to learn how SQL queries are constructed, even though it may not possess an in-depth comprehension of the semantic meanings of individual column names or the specifics of database schemas.

Since the knowledge of generating SQL statements is based on the training sets which are supposed to be big enough to cover semantically correct cases, semantic verification of the generated SQL statements needs to be done. This paper presents the review and verification of database retrieval statements which are generated by ChatGPT. Relational algebra, relational calculus, and SQL are considered. SQL is a relational complete language and the most widely used database language. The ability to employ ChatGPT to generate SQL codes from given natural language statements is very significant. There are also several informal articles and discussions on the internet on the use of ChatGPT to generate SQL statements, but not on the relational completeness issue. This paper verifies that ChatGPT is able to generate SQL and relational calculus statements which are equivalent to essential relational algebra operators.

2 RELATIONAL COMPLETE LANGUAGE

Apart from the relational data structure, and the entity and referential integrity constraints, the third part of the relational data model is the relational complete language. For a DBMS to be considered as a Relational DBMS (RDBMS), it must support all the three parts of the relational database model.

This work is licensed under a Creative Commons Attribution International 4.0 License.

ASSE 2023, October 27–29, 2023, Aizu-Wakamatsu City, Japan
© 2023 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0853-4/23/10.
https://doi.org/10.1145/3634814.3634817

A language is a relational complete language if it is at least equivalent to relational calculus or relational algebra.

Codd published his 1972 paper "Relational Completeness of Data Base Sublanguages" [6] which introduced relational algebra and relational calculus, together with the concept of relational completeness.

Relational calculus is a non-procedural language. It is used to describe the desired query result. It is a "what" type of language based on predicate logic. The search conditions must be in well-formed formula. Existential and universal quantifiers are required; thus, the relational calculus statements look formal. In fact, there are there are two variations: The Tuple Relational Calculus, and the Domain Relational Calculus. The first one has tuple variables whose instantiated values are tuples (rows). The second one has domain variables whose instantiated values are domain (column) values.

Both of them are declarative in nature and the users define precisely what they want, not how to obtain the result. Performance has nothing to do with calculus-based languages.

Codd worked with IBM at the time of the relational model and languages invention. He proposed to IBM the commercial potential of his inventions. However, the two relational calculus languages are too formal to market as they are. An IBM team led by Chamberlin developed SEQUEL (Structured English Query Language) [7] based on the Tuple Relational Calculus. SEQUEL works on the IBM prototype DBMS System*R, which later became DB2. This language later became SQL (Structured Query Language), the standard relational database language. A user-friendly version of the Domain Relational Calculus was also developed at IBM by a team led by Zloof. The language is called Query By Examples (QBE) [8]. QBE was marketed before SQL but is much less popular.

After rigorous competitions, IBM's SQL was selected to be the standard relational database language by ANSI in 1986, and ISO in 1987 [9]. The SQL language which is a direct descendent of the Tuple Relational Calculus, is a relational complete language. SQL completely represents Tuple Relational Calculus. There is no commercial RDBMS that supports the Tuple or the Domain Relational Calculus in their original format.

3 RELATIONAL ALGEBRA

Relational algebra is a procedural language whose sole data structure is relation. The operands must be relations and the results are relations. In Codd's original paper that introduces the Relational Algebra language, it originally comprises eight operators [10]. However, it has been later extended and some operators can be derived from other operators. Different literatures may give different number of relational algebra operators. In this section, we follow Codd's original paper and therefore refer to the original eight operators.

Unlike the relational calculus languages which is now replaced by SQL and QBE, the relational algebra is still widely used internally by the DBMS. It is intended to be used as the machine language of relational database machines. All relational DBMSs translate statements in the relational calculus-based languages such as SQL and QBE, into equivalent relational algebra statements and further process them at the relational algebra level. Query optimization is also done at the relational algebra level. Some high-end database machines such as Oracle's Exadata Database Machines execute relational algebra statements at the storage server level. The relational algebra comprises two parts: a set of four operators which are traditional set operations union, intersection, difference, and Cartesian product; and four relational operators select, project, join, and divide.

4 RELATIONAL COMPLETENESS VERIFICATION OF CHATGPT-GENERATED DATABASE STATEMENTS

In this section, we ask ChatGPT to generate database queries in relational algebra, relational calculus, and SQL based on the popular sample relational database tables VENDOR, ITEM, SALE which are shown in Figure 1. For relational completeness verification purpose, five test questions are set up. Question 1 verifies that ChatGPT can correctly generate relational algebra, relational calculus, and SQL statements which are equivalent to relational algebra select and project operations. Question 2 verifies the relational algebra join operation. Question 3 verifies the relational algebra union operation. Question 4 verifies the relational algebra intersection operation, and question 5 verifies the relational algebra divide operation.

Question 1: This is a query which verifies the relational algebra select and project operations.

List vendor name of vendors who are in Bangkok.

Prompt:

Given the following schemas:
VENDOR (V#, VNAME, LOCATION, STATUS)
ITEM (I#, INAME, COLOR)
SALE (V#, I#, AMOUNT)
Write a relational algebra query: List vendor name of vendors who are in Bangkok.

ChatGPT:

The relational algebra query to list the vendor names of vendors who are in Bangkok can be expressed as follows: π(VNAME)(σ(LOCATION='Bangkok')(VENDOR))

Explanation: σ(LOCATION='Bangkok')(VENDOR): This operation selects the rows from the VENDOR relation where the LOCATION is 'Bangkok'. π(VNAME): This operation projects the VNAME attribute from the result obtained above, giving the list of vendor names.

So, the complete query reads as follows: π(VNAME)(σ(LOCATION='Bangkok')(VENDOR))

This query uses the σ (selection) operation to filter the VENDOR relation based on the condition that the LOCATION attribute is 'Bangkok', and then uses the π (projection) operation to retrieve the VNAME attribute from the filtered rows, which gives the list of vendor names of vendors who are in Bangkok.

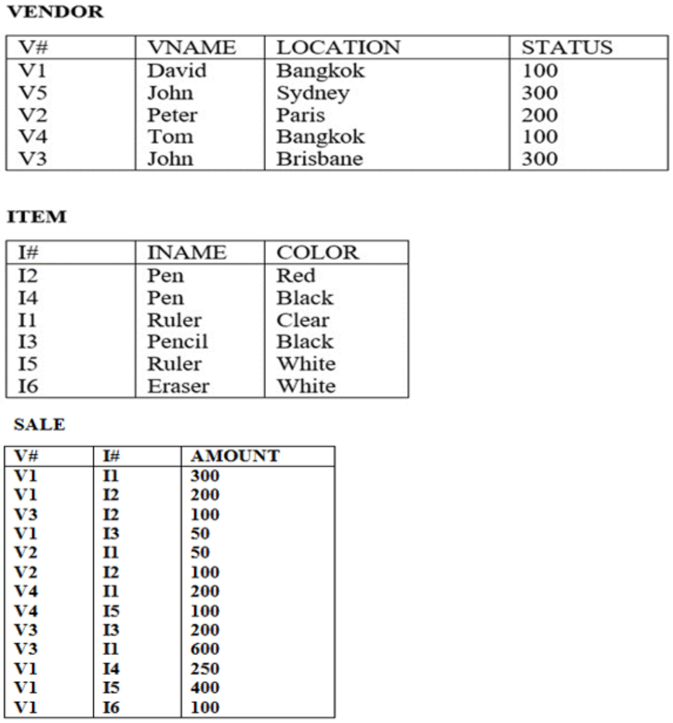
Prompt:

Write a relational calculus query: List vendor name of vendors who are in Bangkok.

ChatGPT:

Certainly, here's the relational calculus query to list the vendor names of vendors who are located in Bangkok: {v.VNAME | ∃v ∈ VENDOR (v.LOCATION = 'Bangkok')}

Prompt:



VENDOR

V#	VNAME	LOCATION	STATUS
V1	David	Bangkok	100
V5	John	Sydney	300
V2	Peter	Paris	200
V4	Tom	Bangkok	100
V3	John	Brisbane	300

ITEM

I#	INAME	COLOR
I2	Pen	Red
I4	Pen	Black
I1	Ruler	Clear
I3	Pencil	Black
I5	Ruler	White
I6	Eraser	White

SALE

V#	I#	AMOUNT
V1	I1	300
V1	I2	200
V3	I2	100
V1	I3	50
V2	I1	50
V2	I2	100
V4	I1	200
V4	I5	100
V3	I3	200
V3	I1	600
V1	I4	250
V1	I5	400
V1	I6	100

Figure 1: The sample relational database tables VENDOR, ITEM, SALE

Write a SQL query: List vendor name of vendors who are in Bangkok.

ChatGPT:

Certainly, here's the SQL query to list vendor names of vendors who are located in Bangkok:

SQL code:

SELECT VNAME FROM VENDOR WHERE LOCATION = 'Bangkok';

Question 2: This is a query which verifies the relational algebra select, project, and join operations.

List vendor name, item name, and amount supplied of vendors who are in Bangkok.

Prompt:

Given the following schemas:

VENDOR (V#, VNAME, LOCATION, STATUS)

ITEM (I#, INAME, COLOR)

SALE (V#, I#, AMOUNT)

Write a relational algebra query: List vendor name, item name, and amount supplied of vendors who are in Bangkok.

ChatGPT:

Of course, here's the relational algebra query to retrieve the vendor name, item name, and amount supplied for vendors located in Bangkok: π VNAME, INAME, AMOUNT (σ LOCATION='Bangkok' (VENDOR $\bowtie$ SALE $\bowtie$ ITEM))

In this query: π VNAME, INAME, AMOUNT specifies the projection of columns: Vendor Name, Item Name, and Amount: σ LOCATION='Bangkok' represents the selection condition where the LOCATION is 'Bangkok'.

VENDOR $\bowtie$ SALE $\bowtie$ ITEM denotes the natural join operation between the VENDOR, SALE, and ITEM relations, linking them based on the common attributes (V# and I#). This ensures that we are getting the relevant information from all three tables for the desired output.

Prompt:

Write a relational calculus query: List vendor name, item name, and amount supplied of vendors who are in Bangkok.

ChatGPT:

Here's the relational calculus query to retrieve the vendor name, item name, and amount supplied for vendors located in Bangkok: {v.VNAME, i.INAME, s.AMOUNT | ∃v ∈ VENDOR, ∃s ∈ SALE, ∃i ∈ ITEM ((v.LOCATION = 'Bangkok') ∧ (v.V# = s.V#) ∧ (s.I# = i.I#))}

Prompt:

Write an SQL query: List vendor name, item name, and amount supplied of vendors who are in Bangkok.

ChatGPT:

Certainly, here's the SQL query to list the vendor name, item name, and amount supplied for vendors who are located in Bangkok:

SELECT V.VNAME, I.INAME, S.AMOUNT FROM VENDOR V JOIN SALE S ON V.V# = S.V# JOIN ITEM I ON S.I# = I.I# WHERE V.LOCATION = 'Bangkok';

Question 3: This is a query which verifies the relational algebra union operation.

List rows of vendors who are in Bangkok or Paris. Use the relational algebra union.

Prompt:

Given the following schemas: VENDOR (V#, VNAME, LOCATION, STATUS) ITEM (I#, INAME, COLOR) SALE (V#, I#, AMOUNT) Write a relational algebra query: List rows of vendors who are in Bangkok or Paris. Use the relational algebra union.

ChatGPT:

Here’s the relational algebra query using the UNION operation to list rows of vendors who are in Bangkok or Paris. (π V#, VNAME, LOCATION, STATUS (σ LOCATION=’Bangkok’ (VENDOR))) ∪ (π V#, VNAME, LOCATION, STATUS (σ LOCATION=’Paris’ (VENDOR)))

Prompt:

Write a relational algebra query: List rows of vendors who are in Bangkok or Paris.

ChatGPT:

Certainly, here’s the relational algebra query to list rows of vendors who are in Bangkok or Paris: σ LOCATION=’Bangkok’ ∨ LOCATION=’Paris’ (VENDOR)

Prompt:

Write a relational calculus query: List rows of vendors who are in Bangkok or Paris. Use the relational algebra union.

ChatGPT:

Here’s the relational calculus query to list rows of vendors who are in Bangkok or Paris using the union operation: {t | ∃t ∈ VENDOR (t.LOCATION = ’Bangkok’ ∨ t.LOCATION = ’Paris’)}

Prompt:

Write an SQL query: List rows of vendors who are in Bangkok or Paris. Use the relational algebra union.

ChatGPT:

Sure, here’s the SQL query that uses the UNION operator to list rows of vendors who are in Bangkok or Paris:

```
SELECT * FROM VENDOR WHERE LOCATION = ’Bangkok’ UNION SELECT * FROM VENDOR WHERE LOCATION = ’Paris’;
```

This query retrieves all rows from the **VENDOR** table where the **LOCATION** is either ’Bangkok’ or ’Paris’ using the UNION operator to combine the results from the two separate SELECT statements.

Prompt:

Write an SQL query: List rows of vendors who are in Bangkok or Paris.

ChatGPT:

Certainly, here’s the SQL query to list rows of vendors who are in Bangkok or Paris:

```
SELECT * FROM VENDOR WHERE LOCATION IN (’Bangkok’, ’Paris’);
```

This query selects all rows from the **VENDOR** table where the **LOCATION** is either ’Bangkok’ or ’Paris’. The **IN** operator allows you to specify multiple values for the condition.

Question 4: This is a query which verifies the relational algebra intersection operation.

List sale V# of vendors who supply both I1 and I2.

Prompt:

Write a relational algebra query: List sale V# of vendors who supply both I1 and I2.

ChatGPT:

Here’s the relational algebra query to list the vendor numbers (V#) of vendors who supply both I1 and I2: π V# (σ I#=’I1’ (SALE)) ∩π V# (σ I#=’I2’ (SALE))

This query first selects the vendor numbers for sales of item I1, then intersects them with the vendor numbers for sales of item I2, resulting in the vendor numbers of vendors who supply both items.

Prompt:

Write a relational calculus query: List sale V# of vendors who supply both I1 and I2.

ChatGPT:

Relational calculus has its limitations in expressing complex operations like intersection directly. However, you can achieve the desired result using nested quantifiers: { s.V# | ∃s ∈ SALE, ∃v1 ∈ VENDOR, ∃v2 ∈ VENDOR ((s.I# = ’I1’ ∧ v1.V# = s.V#) ∧ (s.I# = ’I2’ ∧ v2.V# = s.V#) ∧ (v1.V# = v2.V#)) }

This query retrieves the vendor numbers (V#) of vendors who supply both I1 and I2 by ensuring there exist sales records for both items with the same vendor number.

Please note that relational calculus might become more complex for set operations like intersection compared to relational algebra.

Prompt:

Write an SQL query: List sale V# of vendors who supply both I1 and I2.

ChatGPT:

Certainly, here’s the SQL query to list the vendor numbers (V#) of vendors who supply both I1 and I2:

```
SELECT DISTINCT s1.V#
FROM SALE s1 WHERE s1.I# = ’I1’
AND EXISTS ( SELECT 1
FROM SALE s2
WHERE s2.V# = s1.V# AND s2.I# = ’I2’ );
```

This query first selects the vendor numbers for sales of item I1 (**’I1’**), then checks if there exists another sale record for the same vendor number (**V#**) but with item I2 (**’I2’**). The **EXISTS** subquery ensures that vendors who supply both I1 and I2 are included in the result.

Question 5: This is a query which verifies the relational algebra difference operation.

List sale V# of vendors who supply I1 but not I2.

Prompt:

Write a relational algebra query: List sale V# of vendors who supply I1 but not I2.

ChatGPT:

Here’s the relational algebra query to list the vendor numbers (V#) of vendors who supply I1 but not I2: π V# ((σ I#=’I1’ (SALE)) - (π V# (σ I#=’I2’ (SALE)))).

In this query: **(σ I#=’I1’ (SALE))** selects the rows of sales where the item is I1 and **(π V# (σ I#=’I2’ (SALE)))** selects the vendor numbers for sales of item I2.

The set difference operator - subtracts the set of vendor numbers who supply I2 from the set of vendor numbers who supply I1, resulting in the vendor numbers who supply I1 but not I2.

Prompt:

Write a relational calculus query: List sale V# of vendors who supply I1 but not I2.

ChatGPT:

Here's the relational calculus query to list the vendor numbers (V#) of vendors who supply I1 but not I2: { s.V# | ∃s ∈ SALE, ∃v ∈ VENDOR ((s.I# = 'I1' ∧ v.V# = s.V#) ∧ ¬ (∃s' ∈ SALE (s'.I# = 'I2' ∧ s'.V# = s.V#))) }

This query retrieves the vendor numbers (V#) of vendors who supply I1 and then checks if there is no sale record for the same vendor number that supplies I2, using the negation (¬) operator. This ensures that only vendors who supply I1 but not I2 are included in the result.

Prompt:

Write an SQL query: List sale V# of vendors who supply I1 but not I2.

ChatGPT:

Certainly, here's the SQL query to list the vendor numbers (V#) of vendors who supply I1 but not I2:

```
SELECT DISTINCT s1.V#
FROM SALE s1
WHERE s1.I# = 'I1' AND s1.V# NOT IN (SELECT s2.V#
FROM SALE s2
WHERE s2.I# = 'I2' );
```

This query selects the distinct vendor numbers for sales of item I1 (**'I1'**), and then excludes the vendor numbers that have sales records for item I2 (**'I2'**) using a subquery with the **NOT IN** condition. The result will contain the vendor numbers who supply I1 but not I2.

Question 6: This is a query which verifies the relational algebra divide operation.

List sale V# of vendors who supply all I# from item using the relational algebra divide.

Prompt:

Write a relational algebra query: List sale V# of vendors who supply all I# from item using the relational algebra divide.

ChatGPT:

To list the sale V# of vendors who supply all I# from the item using the relational algebra divide operation, you can use the following query: π V# ((π V#, I# (SALE)) ÷π I# (ITEM))

In this query: **(π V#, I# (SALE))** selects the V# and I# columns from the SALE relation, representing the set of vendor-item combinations in the sales: **(π I# (ITEM))** selects the I# column from the ITEM relation, representing the set of all items.

The divide operation ÷ finds those vendors whose sale combinations cover all items in the ITEM relation.

This query retrieves the vendor numbers (V#) of vendors who supply all items from the ITEM relation.

Prompt:

Write a relational calculus equivalent of the following relational algebra statement: π V# ((π V#, I# (SALE)) ÷π I# (ITEM))

ChatGPT:

The relational algebra division operation is not directly expressible in relational calculus due to its complexity. However, I can provide a relational calculus expression that captures the essence of the given relational algebra statement using nested quantifiers: { v.V# | ∀v ∈ VENDOR, ∀i ∈ ITEM, ¬ (∃s ∈ SALE ((s.V# = v.V#) ∧ ¬ (∃s' ∈ SALE ((s'.V# = v.V#) ∧ (s'.I# = i.I#))))) }

This relational calculus expression checks for each vendor and each item whether there does not exist any sale record for which a vendor supplied the item but did not supply any other items. This captures the idea of dividing the set of sales by the set of items to find vendors who supply all items.

Please note that while this relational calculus expression captures the essence of the division operation, the actual division operation is more intricate and may require different structures or approaches.

Prompt:

Write an SQL equivalent of the following relational algebra statement: π V# ((π V#, I# (SALE)) ÷π I# (ITEM))

ChatGPT:

SQL does not have a direct equivalent to the relational algebra division operation. However, you can achieve a similar result using a combination of SQL operations. Here's an approximation of the given relational algebra statement:

```
SELECT DISTINCT s.V#
FROM SALE s
WHERE NOT EXISTS (SELECT i.I#
FROM ITEM i
WHERE NOT EXISTS ( SELECT s2.I#
FROM SALE s2
WHERE s2.V# = s.V# AND s2.I# = i.I# ) );
```

This SQL query approximates the division operation by selecting the distinct vendor numbers from the **SALE** table where there does not exist an item from the **ITEM** table that is not supplied by the vendor. It checks if for each item in the **ITEM** table, there is a corresponding sale record for the same vendor in the **SALE** table.

Please note that while this query approximates the division operation, the actual division operation is more intricate and may require different structures or approaches.

5 CONCLUSION

Five natural language test questions which cover all the essential relational operators are designed and submitted to ChatGPT for the generation of relational algebra, relational calculus, and SQL statements. We verified each of them and found that ChatGPT generates correct queries for all test questions. The ability to generate relational algebra and relational calculus statements is remarkable. It should be noted that the prompt should refer to relevant table names to avoid confusion. We also test more complex queries on more detailed database tables and obtain satisfactory results. It can be concluded that ChatGPT is a useful tool which has the potential to be employed as a database programming assistant.

REFERENCES

[1] A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I. (2019). Language Models are Unsupervised Multitask Learners. arXiv:1901.02860

[2] Veres and Csaba. (2022). Large Language Models are Not Models of Natural Language: They are Corpus Models.10, 61970-61979.

[3] OpenAI. (2023). ChatGPT (Aug 01 version) [Large language model]. https://chat.openai.com/chat.

[4] Immanuel Trummer. (2022). Generating Code for Processing SQL Queries using GPT-3 Codex. 2204.08941. arXiv. cs.DB

[5] Raul Castro Fernandez and Aaron J. Elmore and Michael J. Franklin and Sanjay Krishnan and Chenhao Tan.(2023). How Large Language Models Will Disrupt Data Management. Proc. VLDB Endow.16. 3302-3309. https://api.semanticscholar.org/CorpusID:261193780

[6] Codd, E.F. (1972). Relational Completeness of Data Base Sublanguages. Research report // San Jos{é} Research Laboratory: Computer sciences. IBM Corporation. https://books.google.co.th/books?id$=$vwJrHAAACAAJ

[7] Chamberlin, D.D. and Boyce, R.F. (1974). SEQUEL: A Structured English Query Language. In: Proceedings of the 1974 ACM SIGFIDET (Now SIGMOD) Workshop

on Data Description, Access and Control (SIGFIDET' 74).ACM Press. Ann Aarbor. 249-264.
[8] Moshé M. Zloof. (1975). Query-by-Example: the Invocation and Definition of Tables and Forms. Proceedings of the International Conference on Very Large Data Bases, September 22-24, 1975, Framingham, Massachusetts, USA. Douglas S. Kerr. ACM. Page 1-24. https://researchr.org/publication/Zloof75
[9] ISO/IEC 9075-2:2016. Information technology — database languages — SQL — part 2: Foundation (SQL/Foundation). Standard, ISO/IEC, August 2021. https://www.iso.org/standard/63556.html.
[10] E. F. Codd. (1979). Extending the Database Relational Model to Capture More Meaning. ACM} Trans. Database Syst. Volum4. Issue 4. Pages 397-434. https://doi.org/10.1145/320107.320109