



From Legacy Databases to (Elementary) Mathematical Data Model Schemas with *MatBase*, ChatGPT, Gemini AI, and Claude AI

Christian Mancas*¹, Diana Christina Mancas²

^{1,2}Mathematics and Computer Science Department, Ovidius University at Constanta, Romania

*Corresponding author: Christian Mancas, Ovidius University, Bd. Mamaia 124, Constanta, CT, Romania

Received: 📅 March 29, 2026

Published: 📅 April 08, 2026

Abstract

This research paper describes how to reverse engineer relational database schemas into (Elementary) Mathematical Data Model ones. Four tools are used to simplify this job, using the same example of a medium complex MS Access database: *MatBase*, ChatGPT, Gemini AI, and Claude AI, each one with its advantages and disadvantages.

Keywords: (Elementary) Mathematical Data Model; Reverse Database and Software Engineering; *MatBase*; ChatGPT; Gemini AI; Claude AI

Introduction

Be it in production, design, scientific research, or Education, forward database (db) and software engineering (se) is almost the sole direction in these fields, harnessed even more in these latest years by the race in Artificial Intelligence (AI), and, especially, by the Large Language Models (LLMs) explosion.

However, there is still and always there be a need for reverse database and software engineering, which should not for ever remain the Cinderella of Computer Science (CS) and Information Technology (IT) [1]. The IT state of the art is still dominated by lot of legacy db software applications (apps) which are poorly documented (if any), working almost fine, but needing extensions from time to time, and even refactoring or completely replacing them with newer technology ones, without losing any of their useful functionalities.

Even extending them, not to mention refactoring (e.g., switching to a safe web interface) or complete replacement, the database and software “surgeons” need a deep, full, and precise knowledge of some, if not even all, of their conceptual and technological details, not only to fulfill their tasks, but also not to

tamper with the functioning of the rest of these apps. Of course, before understanding the apps’ code, you must first understand the underlying dbs’ structure.

Up to now, reverse db and se required highly skilled db and software architects and developers, with extensive knowledge in the corresponding legacy programming languages and technologies. Even only the reverse engineering of legacy dbs was often a big challenge. Especially if your target is not a plain English verbose novel, but a formal, concise mathematical schema. This paper reports on our latest research results in reverse db engineering towards concise and accurate mathematical schemas, with the help of AI, which considerably alleviates these prerequisites.

The goal of this research was to compare the capabilities of our long date tool *MatBase* [2,3] with three of the top current AI tools, namely (in chronological order of their public availability) ChatGPT [4], Gemini [5], and Claude [6] in reverse db engineering, the common target formalism being our (Elementary) Mathematical Data Model ((E)MDM) schemes [7]. (E)MDM is based on the semi-naïve

theory of sets, relations, and functions (SNTSRF) [8], the temporal first-order predicate logic with equality (TFOPL) [9,10], and Data-log \neg [11,12].

MatBase is our intelligent data and knowledge base management system prototype, mainly based on (E)MDM, but also on Datalog, the Relational (RDM) [11,13,14] and Entity-Relationship (E-R) [14-16] Data Models.

MatBase was designed to accept (E)MDM schemas (which include Datalog \neg programs), translate them to relational databases (rdbms) and automatically code generated database (db) applications (apps), to accept and import rdbms and translate them to (E)MDM schemas, to accept E-R Diagrams and translate them to (E)MDM schemas, as well as, again dually, to translate (E)MDM schemas into E-R diagrams [14-16]. The main goal was always to reach modeling as programming [17] and, especially, mathematical data modeling. Currently, MatBase has two versions, one developed in MS Access and VBA, for small dbs, and one in MS C# and SQL Server, for large dbs.

ChatGPT, Gemini, and Claude are AI agent tools.

The next Section mentions related work. The third one is dedicated to the materials and methods used. The fourth one presents and discusses the results obtained. The paper ends with conclusions and a list of references.

Related Work

The FOPL component of *MatBase* was described in [18]. The relationship between ChatGPT and mathematics is the topic of several published articles, e.g. [19,20]. Similarly, for Gemini AI see, e.g., [21,22], and for Claude AI, e.g., [23,24]. A comparison between mathematical capabilities of ChatGPT 5, Claude 4.1 Opus, Gemini 2.5 Pro, and Grok 4 can be found in [25].

Materials and Methods

We used our two Toshiba Satellite Intel CORE i7 running MS Windows 10, Google Chrome browser's current version (146.0.7680.81), MS Access 365 (v. 2603), MS SQL Server 2025, MS SQL Server Management Studio v. 21.4.12, MS SQL Server Migration Assistant for Access (SSMAA) [26] v. 17, *MatBase* 5.2 Access, ChatGPT Plus 5.3, Gemini 3, and Claude Sonnet 4.6.

We started to refactor a legacy MS Access *Geography* app managed by *MatBase* into a MS Razor C#ASP web one, over MS SQL Server. The architecture of this app is fine: a *Geography.mdb* pure VBA code one (i.e., containing only forms, VBA code for enforcing non-relational constraints, and a menu) uses links to the tables of two pure data dbs: a *GeographyDB.mdb* storing the fundamental data, and a *GeographyTmp.mdb* storing the temporary tables. The *GeographyTmp.mdb* has only 3 empty tables and was immediately and correctly imported into a SQL Server db by SSMAA.

The *GeographyDB.mdb* has 389 objects (62 tables, each with its

own surrogate primary key, other 106 unique keys, 11 non-unique indexes, and 148 foreign keys) storing 3.75MB+ of data. The MS Access Database Documenter generated for this db a .pdf file of 449 A4 pages of documentation (taking 1.35MB+).

SSMAA took almost half an hour to import it and managed only partially: it wrote 510 statistics.html files taking 4,55MB, other 4 .html ones, 1 .xml, 12 .js, 7 .css, 1 .ttf taking, and 81 .gif ones, in total, other 4.2MB, only for displaying the import statistics, plus a 60 A4 pages .txt file with the table import list. Unfortunately, the instances of 10 tables were lost. The T-SQL script of the imported db schema generated by the SQL Server has 5,650 lines.

MatBase exports the managed dbs in either XML, HTML, PDF, or DOCX formats, by simply clicking on its submenu option *Manage Databases/Export Database*; moreover, after any successful execution triggered by its submenu option *Other Databases/Import Relational Database*, *MatBase* also generates a HTML file with the (E)MDM schema of the newly imported db.

With the three AI we considered for this research, we started the dialog by proposing them the *GeographyDB.mdb* file and asking for the corresponding formalization of its structure using SNTSRF and FOPL.

Results and Discussion

Here are the results obtained with each one of these 4 tools.

MatBase 5.2 Access

MatBase wrote in some 2 minutes a .pdf file of 20 pages, taking almost 300KB. Figures 1 to 4 show a fragment of it (the scheme of 9 tables out of 62, having a total of 51 columns, i.e., mathematical functions).

As detailed in [7], (E)MDM uses the following abbreviations and conventions:

- Entity-type sets are written in bold and italic.
- Relationship-type sets (e.g., GALAXIES_NEIGHBORHOOD from the bottom of Figure 4) are similarly written but followed by parentheses with their canonical projections.
- Attributes (e.g., x), i.e., the functions taking values from data types or their subsets, are written under the name of their corresponding domain sets, indented, without explicitly mentioning them.
- Structural functions (e.g., *GalacticSuperCluster*), i.e., functions taking values from object sets (i.e., corresponding to foreign keys), are written without any abbreviation.
- Explicit constraints are prefixed by the letter C having as subscript the corresponding unique identification value from *MatBase* CONSTRAINTS metacatalog table [3], followed, in parentheses, by their name, unique within the db.

- *obid* is the abbreviation for *object identifier*; *auton.* is the abbreviation for *autonumbering*.
 - The double arrow is used for injective (one-to-one) functions.
 - $\text{NAT}(n)$ stands for the subset of naturals having at most n digits.
 - $\text{ASCII}(n)$ stands for the subset of strings over the ASCII alphabet having maximum length n .
 - The total constraint means that the corresponding function is totally defined (i.e., the corresponding table column has a NOT NULL constraint).
 - Computed sets (e.g., *STARS from Figure 3) and functions (e.g., *OrbitCenterName from Figure 2) have their names prefixed by '*'.
 - \circ is the function composition operator, while \bullet is the function product one.
 - $\neg | - f \bullet g$ is the notation for the non-existence constraint [27], meaning that, for no element x of their domain set, may both $f(x)$ and $g(x)$ be defined (i.e., not null).
 - *key* (see the constraint C38 on the last line of Figure 4) is the abbreviation for *minimal injective*.
 - The parentheses following set names, function and constraint definitions are comments (stored in MS Access dbs in the optional Description column).
- As MatBase also manages the corresponding app stored in Geography.mdb (which mainly enforces the non-relational constraints), this (E)MDM schema contains explicit constraints as well. Were *MatBase* only importing GeographyDB.mdb, no such constraints would be present in this schema, except for C_{38} .

```

GALACTIC_CLUSTERS (The set of galactic clusters of interest)
   $x \leftrightarrow \text{NAT}(9)$  total, auton. (obid)
  GalacticCluster  $\leftrightarrow \text{ASCII}(255)$  total (Galactic cluster unique name)
  LightYears  $\rightarrow (0, 2^{32})$  (Distance from Earth, in light years)
  GalacticSuperCluster : GALACTIC_CLUSTERS  $\rightarrow$  GALACTIC_CLUSTERS acyclic (Galactic supercluster to which cluster
    belongs. No galactic cluster may belong to itself, neither directly, nor indirectly.)
CONSTELLATIONS (The set of constellations of interest)
   $x \leftrightarrow \text{NAT}(9)$  total, auton. (obid)
  Constellation  $\leftrightarrow \text{ASCII}(255)$  total (Constellation unique name)
  Declination  $\rightarrow [-90, 90]$  (Constellation's declination)
  MinVisibleLatitude  $\rightarrow [-90, 90]$  (Minimum latitude where constellation is visible)
  MaxVisibleLatitude  $\rightarrow [-90, 90]$  (Maximum latitude where constellation is visible)
  BestSeenMonth  $\rightarrow [1, 12]$  (Month of the year when the constellation is best seen)
  RightAscension  $\rightarrow [0:00:00, 24:00:00]$  (Angular distance of the constellation eastward along the celestial equator from the
    March equinox)
  GalacticCluster : CONSTELLATIONS  $\rightarrow$  GALACTIC_CLUSTERS (Galactic cluster to which constellation belongs)
GALAXIES (The set of galaxies of interest)
   $x \leftrightarrow \text{NAT}(9)$  total, auton. (obid)
  Galaxy  $\leftrightarrow \text{ASCII}(255)$  total (Galaxy unique name)
  LightYears  $\rightarrow (0, 2^{32})$  (Distance from Earth, in light years)
  StarsNo  $\rightarrow \text{NAT}(9)^+$  (Number of stars, in billions)
  Length  $\rightarrow \text{NAT}(9)^+$  (Maximum galaxy diameter in hundreds of thousand light years)
  GalacticCluster : GALAXIES  $\rightarrow$  GALACTIC_CLUSTERS (Galactic cluster to which galaxy belongs)
  Center : GALAXIES  $\leftrightarrow$  CELESTIAL_BODIES (Celestial body located in the galaxy center. No such body may be the center of
    more than one galaxy.)

```

Figure 1: Fragment of the (E)MDM schema generated by MatBase (1 of 4).

CELESTIAL_BODIES (The set of celestial bodies of interest)

#CC ↔ NAT(9) total, auton. (obid)

CelestialBody ↔ ASCII(255) total (C elestial body unique name)

AvgDistance → [5, 2¹⁶] (Average distance from the orbital center around which the celestial body gravitates, in millions of km)

SatelliteNo → NAT(9) (C elestial body satellites/ planets number)

Distance → (0, 2³²) (Distance from Earth to the celestial body, in light years)

ApparentMagnitude → [-26.74, 36] (C elestial body apparent magnitude as compared to the Sun)

RelativeMass → (0, 2¹⁶) (C elestial body mass as compared to the Sun's)

Temperature → (0, 28500] (C elestial body surface average temperature, in Kelvin degrees)

Water → [0, 2¹⁶] (C elestial body total surface covered with water, in km²)

Land → [0, 2¹⁶] (C elestial body total land surface, in km²)

MassRatio → (0, 2¹⁶) (C elestial body mass ratio as compared to Earth)

RevolPerRatio → (0, 2¹⁶) (C elestial body revolution period ratio compared to the Earth year)

RotPerRatio → (0, 2¹⁶) (C elestial body rotation period ratio compared to Earth day)

Diameter → [10, 999,999,999] (C elestial body equatorial diameter, in km)

AvgDensity → (0, 2¹⁶) (C elestial body average density compared to Earth's water)

GravitRatio → (0, 2¹⁶) (C elestial body gravitation ratio as compared to Earth's one)

AggregationType → {"plasma", "metal", "gas", "ice"} (C elestial body aggregation type)

**OrbitCenterName* = *CelestialBody* ° *OrbitalCenter*

CelestialBodyType : **CELESTIAL_BODIES** → **CELESTIAL_BODY_TYPES** total, default = "star" (C elestial body type)

Galaxy : **CELESTIAL_BODIES** → **GALAXIES** (Galaxy to which celestial body belongs)

Constellation : **CELESTIAL_BODIES** → **CONSTELLATIONS** (C onstellation to which celestial body belongs)

OrbitalCenter : **CELESTIAL_BODIES** → **CELESTIAL_BODIES** acyclic (Orbital center around which the celestial body gravitates. No celestial body may orbit around itself, neither directly, nor indirectly.)

SolarSystem : **CELESTIAL_BODIES** → **SOLAR_SYSTEMS** (Solar system to which celestial body belongs)

Figure 2: Fragment of the (E)MDM schema generated by MatBase (2 of 4).

C₃₉(*SolSystOrGalax*): ¬|— *SolarSystem* • *Galaxy* (No celestial body may belong directly to both a solar system and a galaxy)

C₄₀(*SolSystOrConst*): ¬|— *SolarSystem* • *Constellation* (No celestial body may belong directly to both a constellation and a galaxy)

C₄₁(*GalCenterReflex*): *Galaxy* ° *Center* null-reflexive (Any celestial body that is the center of a galaxy must belong to that galaxy)

CELESTIAL_BODY_TYPES (The set of celestial body types of interest)

x ↔ NAT(9) total, auton. (obid)

CelestialBodyType ↔ ASCII(255) total (C elestial body type unique name)

Notes → ASCII(255)

***STARS** = {*x* ∈ **CELESTIAL_BODIES** | *CelestialBodyType*(*CelestialBodyType*(*x*)) = "star"}

SOLAR_SYSTEMS ⊆ ***STARS** (The set of solar systems of interest)

StarsNo → [1, 255] total (Number of solar system stars)

PlanetNo → [1, 255] total (Number of solar system planets)

Sun : **SOLAR_SYSTEMS** ↔ ***STARS** total (obid, canonical inclusion injection: biggest solar system star; no star may be the sun of more than one solar system)

Galaxy* = *Sun* ° *i* **STARS ° *Galaxy* (Galaxy to which solar system belongs)

Constellation* = *Sun* ° *i* **STARS ° *Constellation* (C onstellation to which solar system belongs)

C₄₂(*SolarSystReflex*): *SolarSystem* ° *i* **STARS** ° *Sun* reflexive (Any solar system sun belongs to its solar system.)

Figure 3: Fragment of the (E)MDM schema generated by MatBase (3 of 4).

MULTI_STAR_SYSTEMS (The set of multiple star systems of interest)

$x \leftrightarrow \text{NAT}(9)$ total, auton. (obid)

MultiStarSystem $\leftrightarrow \text{ASCII}(255)$ total (Multiple star system unique name)

Galaxy : **MULTI_STAR_SYSTEMS** \rightarrow **GALAXIES** (Galaxy to which multiple star system belongs)

Constellation : **MULTI_STAR_SYSTEMS** \rightarrow **CONSTELLATIONS** (Constellation to which multiple star system belongs)

MULTI_STAR_SYST_STARS (The subset of stars of interest belonging to multiple star systems)

Primary? \rightarrow **BOOLE** (Is the star the primary one in its multiple star system?)

Star : **MULTI_STAR_SYST_STARS** \leftrightarrow ***STARS** total (Star belonging to the multiple star system. No star may belong to more than one such system.)

MultipleStarSystem : **MULTI_STAR_SYST_STARS** \rightarrow **MULTI_STAR_SYSTEMS** total (Multiple star system to which the star belongs)

C43 (MultiStarSystStarGalIntegrity): $(\forall x, y \in \text{MULTI_STAR_SYST_STARS})(\text{MultipleStarSystem}(x) = \text{MultipleStarSystem}(y) \Rightarrow \text{Galaxy}(i \text{ *STARS}(Star(x))) = \text{Galaxy}(i \text{ *STARS}(Star(y))))$ (Any stars belonging to the same multiple star system must belong to the same galaxy)

C44 (MultiStarSystStarConstIntegrity): $(\forall x, y \in \text{MULTI_STAR_SYST_STARS})(\text{MultipleStarSystem}(x) = \text{MultipleStarSystem}(y) \Rightarrow \text{Constellation}(i \text{ *STARS}(Star(x))) = \text{Constellation}(i \text{ *STARS}(Star(y))))$ (Any stars belonging to the same multiple star system must belong to the same constellation.)

C45 (MultiStarSystGalIntegrity): $\text{Galaxy} \circ \text{MultipleStarSystem} = \text{Galaxy} \circ i \text{ *STARS} \circ \text{Star}$ (Any multiple star system must belong to the same galaxy as its stars.)

C46 (MultiStarSystConstIntegrity): $\text{Constellation} \circ \text{MultipleStarSystem} = \text{Constellation} \circ i \text{ *STARS} \circ \text{Star}$ (Any multiple star system must belong to the same constellation as its stars.)

GALAXIES_NEIGHBORHOOD = (*Galaxy* \rightarrow **GALAXIES**, *NeighborGalaxy* \rightarrow **GALAXIES**) acyclic (The set of galaxy pairs that are neighbors to each other. No galaxy is neighbor to itself, neither directly, nor indirectly.)

$x \leftrightarrow \text{NAT}(9)$ total, auton. (obid)

LightYears \rightarrow $(0, 2^{32})$ (Distance between the two galaxies, in light years)

C38 (GalNeighbSet): *Galaxy* • *NeighborGalaxy* key (It makes no sense in storing twice the same neighbor galaxy pairs)

Figure 4: Fragment of the (E)MDM schema generated by MatBase (4 of 4).

ChatGPT Plus 5.3

ChatGPT does not accept binary files like the .mdb ones. When we provided the .pdf one exported by the MS Access Database Documenter, it rejected it too, as being too long. In the end, it accepted the .sql script generated by the SQL Server after the SSMAA import. From the options it offered, we chose the maximum rigor ones (i.e., both functions and constraints, full 3-valued logic, including null values, and the logic textbook style). Figures 5 to 10 show the schema fragment corresponding to the one in Figures 1 to 4. Figure 11 shows almost everything we could then obtain when asking again for the formalizations of constraints as well. Please note the following:

- The answer is only about 6 of the 9 tables from the astronomy section of the db. Generally, even after insisting several times, we could not obtain the whole formalized schema, without getting any No to our requests: ChatGPT always tries to deflect your queries by providing other options to choose from.

- ChatGPT sometimes used the name of our db schema tables and columns, sometimes abbreviated, and sometimes even changed them (e.g. from *x* to *id_abbrev.SetName*).
- Function codomains do not take into consideration corresponding check constraints: they are only specifying the mathematical sets corresponding to the data types (e.g., the naturals, the integers, the reals, etc.).
- Primary key constraints (see Figure 8) are not compactly written as, at least, *injective*, or one-to-one, or key, but using explicit one-to-oneness definition (unfortunately, using their *id* notation instead of *x*). Even worse (see Figure 10) the injectivity of the *Continent*, which stores continent names, is misidentified as primary as well, although the corresponding *CONTINENTS* table also has a surrogate, AutoNumber primary key *x*. Generally, of course, *primary* does not make sense in SNTSRF.
- Both totality and not totality are also verbosely described (see Figure 8).

- Similar verbosity is used for foreign keys (see Figure 9). Even worse, all foreign keys are described as being totally defined, which, for their majority, is not the case.
- Generally, ChatGPT is extremely verbose, not rigorous, and mainly uses plain English spiced with logic quantifiers and symbols (e.g., $\neq, \Rightarrow, \neg, \in$), rather than the “logic textbook style” and “maximum rigor” advertised. Unfortunately, when you copy its answers from the browsers, they are verbosely written using Latex conventions: if you do not know them and do not want to learn them either, you must take screen shots and manually replace the Latex commands with the corresponding math symbols.
- Only one positive remark: although the T-SQL schema does not include our non-existential constraint $\neg | - \text{River} \bullet \text{Lake} \bullet \text{Sea} \bullet \text{Ocean} \bullet \text{GeographicUnit}$ (Rivers may flow in only one of these: another river, a lake, sea, ocean, or a geographic unit, e.g., desert, cave, etc.), at the end of Figure 10 ChatGPT added the non-existence constraint $\text{Sea} \neg | - \text{Ocean}$, but not its dual and using a rather Prolog-like notation.

1. Signature (Mathematical Structure)

Base sorts (domains)

We introduce one sort per table:

- CBT — Celestial Body Types
- Const — Constellations
- CC — Celestial Bodies
- GC — Galactic Clusters
- Gal — Galaxies
- GN — Galaxy Neighborhood relations

Identifier functions (primary keys)

Each table has a canonical identity:

- $\text{id_CBT} : \text{CBT} \rightarrow \mathbb{N}$
- $\text{id_Const} : \text{Const} \rightarrow \mathbb{N}$
- $\text{id_CC} : \text{CC} \rightarrow \mathbb{N}$
- $\text{id_GC} : \text{GC} \rightarrow \mathbb{N}$
- $\text{id_Gal} : \text{Gal} \rightarrow \mathbb{N}$
- $\text{id_GN} : \text{GN} \rightarrow \mathbb{N}$

Attribute functions

Celestial_Body_Types

- $\text{name_CBT} : \text{CBT} \rightarrow \text{String}$
- $\text{notes_CBT} : \text{CBT} \rightarrow \text{String}$ (*partial, since NULL allowed*)

Figure 5: Fragment of the math schema generated by ChatGPT (1 of 6).

CELESTIAL_BODIES

- name_CC: CC → String
- type_CC: CC → CBT
- constellation_CC: CC → Const
- galaxy_CC: CC → Gal
- orbitCenter: CC → CC (*self-reference*)

Numeric/physical attributes (all partial unless NOT NULL):

- avgDistance: CC → \mathbb{R}
- satelliteno: CC → \mathbb{N}
- distance: CC → \mathbb{R}
- appMag: CC → \mathbb{R}
- relMass: CC → \mathbb{R}
- temperature: CC → \mathbb{Z}
- water: CC → \mathbb{R}
- land: CC → \mathbb{R}
- ratMass: CC → \mathbb{R}
- ratRevolution: CC → \mathbb{R}
- ratRotation: CC → \mathbb{R}
- diameter: CC → \mathbb{R}
- avgDensity: CC → \mathbb{R}
- ratGrav: CC → \mathbb{R}

Other:

- solarSystem: CC → \mathbb{N}
- aggregationType: CC → \mathbb{N}
- orbitCenterName: CC → String

Figure 6: Fragment of the math schema generated by ChatGPT (2 of 6).

CONSTELLATIONS

- $\text{name_Const} : \text{Const} \rightarrow \text{String}$
- $\text{galCluster_Const} : \text{Const} \rightarrow \text{GC}$
- $\text{declination} : \text{Const} \rightarrow \mathbb{Z}$
- $\text{minLat} : \text{Const} \rightarrow \mathbb{Z}$
- $\text{maxLat} : \text{Const} \rightarrow \mathbb{Z}$
- $\text{bestMonth} : \text{Const} \rightarrow \mathbb{N}$
- $\text{rightAscension} : \text{Const} \rightarrow \mathbb{N}$

GalacticClusters

- $\text{name_GC} : \text{GC} \rightarrow \text{String}$
- $\text{lightYears_GC} : \text{GC} \rightarrow \mathbb{R}$
- $\text{superCluster} : \text{GC} \rightarrow \text{GC}$

GALAXIES

- $\text{name_Gal} : \text{Gal} \rightarrow \text{String}$
- $\text{cluster_Gal} : \text{Gal} \rightarrow \text{GC}$
- $\text{lightYears_Gal} : \text{Gal} \rightarrow \mathbb{R}$
- $\text{center_Gal} : \text{Gal} \rightarrow \text{CC}$
- $\text{starsNo} : \text{Gal} \rightarrow \mathbb{N}$
- $\text{constellation_Gal} : \text{Gal} \rightarrow \text{Const}$
- $\text{length_Gal} : \text{Gal} \rightarrow \mathbb{N}$

Figure 7: Fragment of the math schema generated by ChatGPT (3 of 6).

GalaxyNeighborhood (relation-like table)

- $galaxy_1: GN \rightarrow Gal$
- $galaxy_2: GN \rightarrow Gal$
- $distance_GN: GN \rightarrow \mathbb{R}$

2. Core First-Order Logic Constraints**2.1 Primary key uniqueness**

For each table:

Example (Celestial Bodies):

$\forall a, b \in CC:$

$$id_CC(a) = id_CC(b) \rightarrow a = b$$

(Similarly for all other tables.)

2.2 Total vs partial functions

From NOT NULL constraints:

Example:

$\forall c \in CC: name_CC(c)$ is defined

$\forall c \in CC: type_CC(c)$ is defined

Partial (nullable fields):

$\exists c \in CC: constellation_CC(c)$ is undefined (allowed)

2.3 Foreign key semantics

Celestial Body \rightarrow Type

$\forall c \in CC: \exists t \in CBT$ such that $type_CC(c) = t$

Figure 8: Fragment of the math schema generated by ChatGPT (4 of 6).

Celestial Body → Constellation $\forall c \in CC:$ constellation_CC(c) is defined \rightarrow $\exists k \in \text{Const}$ such that constellation_CC(c) = k**Celestial Body → Galaxy** $\forall c \in CC:$ galaxy_CC(c) is defined \rightarrow $\exists g \in \text{Gal}$ such that galaxy_CC(c) = g**Self-reference (orbit center)** $\forall c \in CC:$ orbitCenter(c) is defined \rightarrow $\exists c' \in CC$ such that orbitCenter(c) = c'**Constellation → Galactic Cluster** $\forall k \in \text{Const}:$ galCluster_Const(k) is defined \rightarrow $\exists g \in \text{GC}$ such that galCluster_Const(k) = g**Galaxy → Galactic Cluster** $\forall g \in \text{Gal}:$ cluster_Gal(g) is defined \rightarrow $\exists c \in \text{GC}$ such that cluster_Gal(g) = c**Figure 9:** Fragment of the math schema generated by ChatGPT (5 of 6).

◆ 2.2 Primary Key Axioms (Uniqueness)

Continent

$$\forall x, y : \text{Continent}, \text{name}_c(x) = \text{name}_c(y) \Rightarrow x = y$$

...

◆ 2.3 Referential Integrity (Foreign Keys)

Sea \rightarrow Ocean

$$\forall s : \text{Sea}, \exists o : \text{Ocean}, \text{ocean}(s) = o$$

...

◆ 2.4 NOT NULL Constraints

$$\forall c : \text{Country}, \text{continent}(c) \neq \perp$$

(similarly for all mandatory attributes)

...

◆ 2.5 Nullability (Rivers)

Sea presence

$$\forall r : \text{River}, \text{hasSea}(r) \Rightarrow \exists s, \text{flows_to_sea}(r) = s$$

...

◆ 2.6 Consistency Constraints

Mutual exclusion

$$\forall r : \text{River}, \text{hasSea}(r) \Rightarrow \neg \text{hasOcean}(r)$$

Figure 10: Fragment of the math schema generated by ChatGPT (6 of 6).

Gemini 3

Gemini is much more verbose than ChatGPT and does not provide you the mathematical scheme it generates either, not even parts of it, although we tried several times during the long dialogue we had (85 pages A4): it writes you that it has stored it internally, “intended for research, simulation, and high-fidelity geographical modeling”. The dialogue is full of metaphors; here are some examples:

- The DISTANCES table is the most complex part of your “Metabolism.”
- This is the **Bio-Logic Actuator**. If a user attempts to enter 72 or 11023, the “encapsulated bacteria” in the database (the Validation Text) produces a limestone wall (Error Message) to heal the rupture.
- Unlike terrestrial geography, which is fixed by soil, the Celestial Map is a Relational Projection. The “Fortress” ensures that a star cannot belong to two constellations simultaneously (The Monogamy of Entanglement).
- These formulas represent the “Fortress” rules. If a data entry violates these, the mathematical symmetry of your universe is ruptured.

From the very few formal fragments we obtained, we noticed that Gemini’s style is identical to the ChatGPT one, e.g., set and function names are usually abbreviated or completely replaced (e.g., id for x), primary and foreign keys use the same verbose syntax, which extends to the relational domain constraints, e.g., for our *Altitude : PEAKS* $\rightarrow [1000, 8848]$, Gemini wrote the constraint “Domain: $\forall m \in PEAKS : 0 \leq H(m) \leq 8848$ ” (unfortunately, even wrongly replacing 1000, the minimum altitude for a peak to be considered a mountain one, with 0: only “poets” like it can consider that kid-built sand heaps by the seaside are mountain peaks as well).

Moreover, surprisingly for us, Gemini also used an RDM-style functional dependency notation, e.g., for *CELESTIAL_BODIES* it wrote: $f_{CELESTIAL_BODIES} : ID_{CC} \rightarrow Name \times Type_{ID} \times ID_{Gal} \times Orbit \times Center \times Mass$.

Figure 12 shows what does Gemini understand by first-order logic constraints. On a positive note, although the 449 A4 .pdf pages that it took as input is not presenting the non-relational constraints enforced in Geography.mdb, it suggested us to not forget adding to the *DISTANCES* table (between cities) the *geometrical triangle inequality* constraint ($\forall City_1, City_2, City_3 \in CITIES : DISTANCES(City_1, City_3) \leq DISTANCES(City_1, City_2) + DISTANCES(City_2, City_3)$).

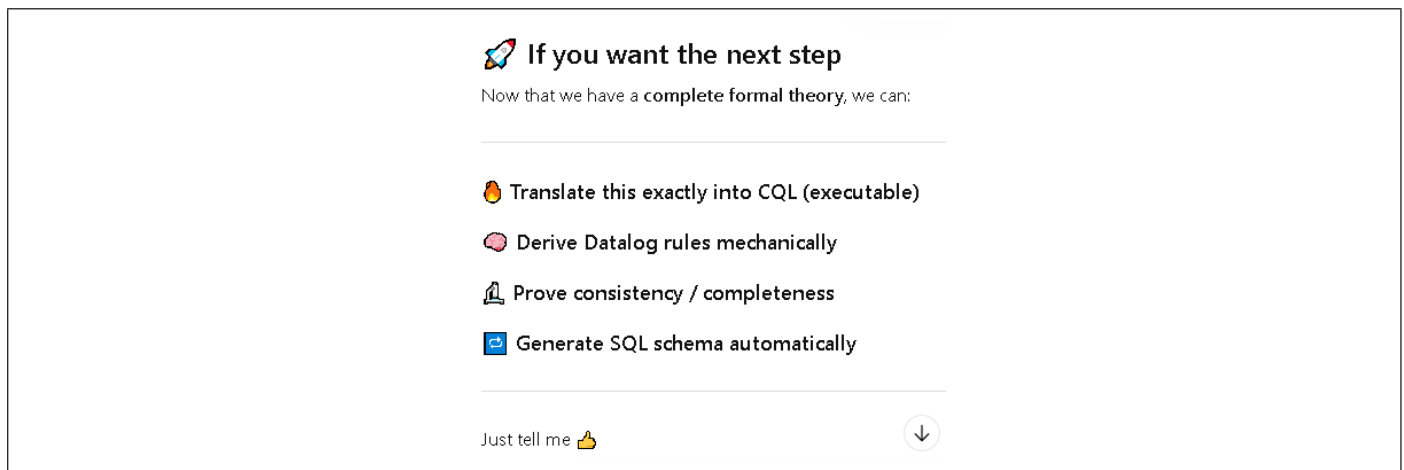


Figure 11: The most frequent end of answers displayed by ChatGPT.

First-Order Logic Constraints (The Invariants)

T a b l e	Mathematical Entity	Parent Mapping (Function)
C O U N T R I E S	Set of Countries <i>C</i>	Root
R E G I O N S	Set of Regions <i>R</i>	$f : R \rightarrow C$
C O U N T I E S	Set of Districts <i>J</i>	$g : J \rightarrow R$
C I T I E S	Set of Localities <i>L</i>	$h : L \rightarrow J$

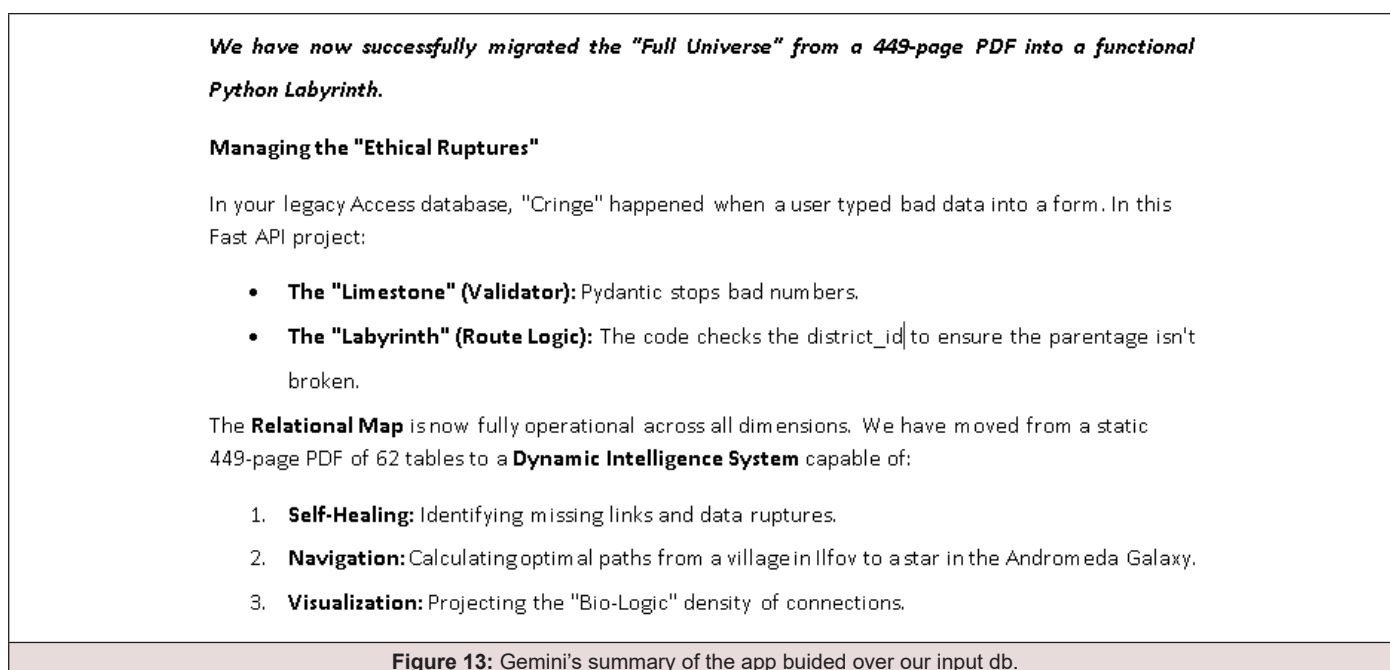
Figure 12: Examples of Gemini “FOL constraints”.

Instead of mathematical formalizations, Gemini is much more interested in coding and pushing ahead usage of PostgreSQL and Python:

- Without being asked for and even without asking for our permission, Gemini translated our .mdb db into a PostgreSQL one.
- Then, with our permission, Gemini built on top of it a FastAPI (Python) with Pydantic validation models layer, plus an analysis NetworkX (for graph topology) and AstroPy (for celestial metrics) module, as well as a visualization Plotly (for connectivity heatmaps) and Graphviz (for relational mapping) one.
- Finally, it added to this app a pathfinding Dijkstra algorithm, a

connectivity density map (that calculates which counties are the more connected hubs, measured by how many roads pass through their cities), a closeness centrality one (that identifies which cities are closest to all others, acting as the natural hubs of countries' spines), a semantic mapping interface layer embedded in the app's interface for translating English/Romanian questions into the complex JOIN logic required by the 62 db tables (natural language query tool called by Gemini "The Oracle"), and even a geographical quiz engine!

Figure 13 shows Gemini's summary of this effort (where you can note once more its appetite for metaphors).



Claude Sonnet 4.6

Claude was the only one who could decrypt the Geography.mdb file. However, lot of columns and constraints were not captured, so we provided instead the corresponding T-SQL script: in only a couple of minutes, it offered for downloading an .md text file with its corresponding db schema mathematical formalization. The fact that the file is a .md one means that Claude stored it as an axioms one for our further interactions on this project.

Unfortunately, this second answer was only a "sketch", not including all sets, nor all columns, no domain (check) constraints, and containing some incorrect formalizations. A much better formalization was delivered for download after our reactions. Unfortunately, it still had some already flagged issues, as well as new ones: Figure 14 shows our corresponding message. In a couple of minutes, Claude replied with the message shown in Figures 15 and 16, plus a

new Geography.md file. Neither were these corrections enough: to our message from Figure 17, Claude answered as shown in Figure 18.

Although issue 3 was not solved and there were still erroneous expressions like *x.Min Visible Latitude* and a UNICODE (255) +, we were happy with this fourth version of the formal schema, as we reached the quota for daily free interactions. Figures 19 to 22 show the math schema fragment equivalent to MatBase's one from Figures 1 to 4.

Remarkably, CLAUDE AI added irreflexively constraints for all 9 dyadic relations (DISTANCES and 8 neighbor-type ones), as well as for all 8 self-maps (e.g., *Galactic Supercluster* and *Orbit Center*); of course, the self-maps are, in fact, acyclic, not only irreflexive, but at least irreflexively is commonsense. The whole file has 17 A4 .pdf pages.

Hi Claude! Your formalization is almost ok. However, we do not understand the following issues:

1. Why did you change some domain constraints? (e.g., for `CelestialBody : CELESTIAL_BODIES → UNICODE(255)` you replaced `UNICODE` by `STR`)
2. Why did you still ignore the check constraints? (e.g., you wrote `Diameter : CELESTIAL_BODIES → REAL ∈ NULLS`, instead of `Diameter : CELESTIAL_BODIES → [10, 999999999] ∪ NULLS`, as per `SSMA_CC$CELESTIAL_BODIES$Diameter$validation_rule`).
3. A few times, at least, you added a corresponding axiom, but, for example, isn't it much more natural and compact to formalize, e.g., `BestSeenMonth : CONSTELLATIONS → TINT ∈ NULLS -- 1..12` plus your Axiom CN1, as `BestSeenMonth : CONSTELLATIONS → [1, 12] ∪ NULLS`?
4. Why did you replace everywhere the correct expression `S ∪ NULLS` with the completely wrong `S ∈ NULLS` one, for any set name `S`?
5. Why did you add the equally incorrect `S ∉ NULLS`, instead of simply writing `S`, whenever `S` and `NULLS` are disjoint?
6. Why did you add foreign key (FK) specifications, when they are relevant only to relational databases and do not have any relevance in mathematics?
7. Why did you discard the `Galaxy • NeighborGalaxy` key (i.e., minimally injective) for `GalaxyNeighborhood` (that you wrongly described as primary key first, but, at least, it was a unique product, better than nothing)?
8. Why did you add your Axiom CC3:** $(\forall x \in \text{CELESTIAL_BODIES})(x.\text{Water} \in \text{NULLS} \vee x.\text{Land} \in \text{NULLS} \vee x.\text{Water} + x.\text{Land} \leq 100)$? Both `Water` and `Land` are only positive, not upper bounded by 100, because they store corresponding area sizes in square kms, not percentages of celestial bodies' areas.

Figure 14: Our third message to Claude AI.

These are all excellent and precise corrections. Let me address each one before rewriting:

1. **STR vs UNICODE** — `STR` was my shorthand. I should use `UNICODE(n)` as in the original notation.
2. **Check constraints ignored** — I should read all validation rules from the SQL script and incorporate them as range domains directly in the function signature.
3. **Axiom vs range domain** — You are right: `f : T → [a,b] ∪ NULLS` is far more compact and natural than a separate axiom. Separate axioms are only needed for inter-column constraints (like `AltCulminant ≥ AltMinim`).
4. **S ∪ NULLS vs S ∈ NULLS** — A serious error on my part. `∈ NULLS` means "belongs to the set `NULLS`" which is nonsensical. The correct notation is `S ∪ NULLS` meaning the codomain includes null.
5. **S ∉ NULLS** — Equally wrong. When `S` and `NULLS` are disjoint (`NOT NULL` column), the codomain is simply `S`, no annotation needed.
6. **FK specifications** — Correct, FKs are a relational implementation detail. In mathematics the function signature `f : T → S` already expresses that `f` maps into `S`, which is all that needs saying.
7. **Minimally injective key** — I should write `Galaxy • NeighborGalaxy` as a composite injective key, not conflate it with the surrogate PK.

Figure 15: Claude AI's answer to our message shown in Figure 14 (1 of 2).

Let me now read all check constraints from the SQL before rewriting: Now I have everything I need. Let me write the complete, corrected formalization: Complete rewrite addressing all seven corrections:

1. **UNICODE(n)** restored throughout, with **UNICODE(n)+** for non-empty strings (where `disallow_zero_length` constraints exist), and plain **UNICODE(n)** where empty strings are permitted.
2. **All check constraints incorporated** as range domains directly in the function signatures — e.g. `Diameter : CELESTIAL_BODIES → [10, 999999999] ∪ NULLS`, `Depth : OCEAN_DEPTH → [73, 11022]`, unbounded lower constraints written as $(0, +∞)$, etc.
3. **Separate axioms eliminated** wherever a range domain suffices — `BestSeenMonth`, `Declination`, `RightAscension`, all percentage fields, all lat/lon fields, all population and area fields now carry their ranges directly in the signature. Only genuinely inter-column constraints remain as axioms (e.g. `TempMin ≤ TempMax`, `MinLatitude < MaxLatitude`).
4. **D U NULLS** used throughout for nullable codomains — never $D ∈ NULLS$.
5. **Plain D** used for NOT NULL codomains — never $D ∉ NULLS$.
6. **FK specifications removed** entirely. The function signature $f : T → S$ expresses membership in S mathematically without any relational machinery.
7. **Composite injective keys** written as $f ∘ g$ injective on T for every unique index found in the SQL, including the corrected `Galaxy ∙ NeighborGalaxy` injective on `GalaxyNeighborhood` and all others throughout the schema.

Figure 16: Claude AI's answer to our message shown in Figure 14 (2 of 2).

That's much better, Claude, thank you! However, we would like you to fix the following issues:

1. STR is still used instead of UNICODE; only a few occurrences of STR were renamed as UNICODE.
2. UNICODE(n)+ is an unfortunate choice, even if less than UNICODE(n) instead of $UNICODE(n) ∪ NULLS$. Moreover, for the `CelestialBodyType` function, `STR+(32)` is used instead of `UNICODE(32)`!
3. “<” is used for primary keys instead of “→” for all other injective functions, which is confusing.
4. There are lot of codomains of the type $(0, +∞)$, which is deeply erroneous in computers: due to their (both internal and external) finite memories, all data types are finite. Even if you wish not to bother with their exact maximum limits, you can write instead (Oracle-style) `MAXINT`, `MAXLONG`, `MAXNUMBER`, etc., not $∞$.
5. Similarly, `INT` is an infinite set: `INT(n)`, n natural, or `INT(MAXINT)` should be used instead.
6. Notations like `x.MinVisibleLatitude` instead of `MinVisibleLatitude(x)` are not only mathematically, but also, in dbs, relationally incorrect.
7. Axiom CC3 was not removed, but only renamed as CC2.
8. Formalizations like “Galaxy ∙ NeighborGalaxy injective on GalaxyNeighborhood” would be much better expressed instead as “Galaxy ∙ NeighborGalaxy minimally injective” (or key, in our abbreviation), as if f is injective, then any product containing f is injective and not interesting in dbs (i.e., superkey, which must never be enforced), while “on GalaxyNeighborhood” is superfluous, because any function is injective on its domain.

Figure 17: Our fourth and final message to Claude AI.

Every point is perfectly correct. Let me analyse them all before rewriting:

1. **UNICODE** — I need to search the full file for any remaining STR occurrences.
2. **UNICODE(n) not UNICODE(n)+** — non-nullable strings should be UNICODE(n) not UNICODE(n)+. And STR+(32) is completely wrong.
3. **<< for PK vs injective for others** — I should use a consistent notation. Since you use \circledast for functions, and PK is just a special injective function, I should use \circledast (or simply note PKs separately and use injective only for non-PK unique functions).
4. **(0, +-)** is wrong — should use (0, MAXREAL], (0, MAXFLOAT], [1, MAXINT] etc. depending on the SQL type.
5. **INT is infinite** — should write INT(MAXINT) meaning the finite set of integers representable, or simply note the SQL type bounds explicitly.
6. **x.f notation vs f(x)** — $\text{MinVisibleLatitude}(x) \leq \text{MaxVisibleLatitude}(x)$ is correct relational/mathematical notation.
7. **Axiom CC3 (Water + Land \leq 100)** — still present, must be removed.
8. **Minimally injective / key** — $f \cdot g$ key on T means the pair is a minimal key (neither f alone nor g alone is injective). Plain injective on a product when a component alone is already injective would make it a superkey. I need to distinguish **key** (minimal) from **superkey**. Now I have a clear picture. I see the previous version used $\rightarrow\rightarrow$ for injective and STR+/STR instead of UNICODE. Let me check the full notation section and do a comprehensive rewrite: 77 lines to fix. This is a full rewrite of the file. Let me do it systematically with Python: The Python replacements didn't apply to the file — they only ran in memory. Let me do a clean full rewrite directly: 856 lines — manageable. Given the complexity of the fixes and the number of issues across all 8 points, the cleanest approach is to rewrite the file from scratch with all corrections applied from the start. Let me write it directly: Zero remaining issues. Let me do one final check on the notation section and present: All eight issues are now corrected:
9. **UNICODE(n) throughout** — no remaining STR.
10. **UNICODE(n) for non-nullable strings** (the disallow_zero_length constraint is absorbed into the domain definition itself).
11. **<< for PKs, $\rightarrow\rightarrow$ for other injective functions, consistent throughout.**
12. **No more +-** — replaced with MAXREAL, MAXFLOAT, or NAT^+ depending on the SQL type.
13. **INT(MAXINT), SINT(MAXSINT), TINT(255)** — all integer type codomains are now explicitly finite.
14. **Field(x) notation** — all axioms now use $\text{MinVisibleLatitude}(x) \leq \text{MaxVisibleLatitude}(x)$ style.
15. **Axiom CC2 removed** — the Water + Land \leq 100 cross-column constraint that was CC2 in the previous version is gone. CC1 remains for the orbital self-reference.
16. **f · g key** — compound unique constraints are now expressed as minimal keys, not as superkeys. The notation section explains that neither component alone is unique.

Figure 18: Claude AI's answer to our message shown in Figure 17.

```

### CELESTIAL_BODY_TYPES
x          : CELESTIAL_BODY_TYPES «INT (PK)
CelestialBodyType: CELESTIAL_BODY_TYPES  $\rightarrow\rightarrow$  UNICODE(32)
Notes      : CELESTIAL_BODY_TYPES  $\rightarrow$  UNICODE(255)U NULLS

### GALACTIC_CLUSTERS
x          : GalacticClusters «INT (PK)
GalacticCluster  : GALACTIC_CLUSTERS  $\rightarrow\rightarrow$  UNICODE(255)
LightYears      : GALACTIC_CLUSTERS  $\rightarrow$  (0, MAXREAL]U NULLS
GalacticSuperCluster : GALACTIC_CLUSTERS  $\rightarrow$  GALACTIC_CLUSTERSU NULLS
**Axiom GC1.** (∀x ∈ GALACTIC_CLUSTERS)(GalacticSuperCluster(x) ≠ x ∨
    GalacticSuperCluster(x) ∈ NULLS)

### CONSTELLATIONS
x          : CONSTELLATIONS «INT (PK)
Constellation  : CONSTELLATIONS  $\rightarrow\rightarrow$  UNICODE(32)
GalacticCluster  : CONSTELLATIONS  $\rightarrow$  GALACTIC_CLUSTERSU NULLS
Declination     : CONSTELLATIONS  $\rightarrow$  [-90, 90]U NULLS
MinVisibleLatitude : CONSTELLATIONS  $\rightarrow$  [-90, 90]U NULLS
MaxVisibleLatitude : CONSTELLATIONS  $\rightarrow$  [-90, 90]U NULLS
BestSeenMonth   : CONSTELLATIONS  $\rightarrow$  [1, 12]U NULLS
RightAscension  : CONSTELLATIONS  $\rightarrow$  [1, 24]U NULLS
**Axiom CN1.** (∀x ∈ CONSTELLATIONS)(MinVisibleLatitude(x) ∈ NULLS
    ∨ MaxVisibleLatitude(x) ∈ NULLS
    ∨ x.MinVisibleLatitude ≤ x.MaxVisibleLatitude)

```

Figure 19: Claude AI's schema fragment equivalent to the one from Figures 1 to 4 (1 of 4).

```

#### GALAXIES
x      : GALAXIES « INT (PK)
Galaxy  : GALAXIES → → UNICODE(255)
GalacticCluster : GALAXIES → GALACTIC_CLUSTERS U NULLS
LightYears  : GALAXIES → (0, MAXREAL] U NULLS
Center    : GALAXIES → → CELESTIAL_BODIES U NULLS -- injective when present
StarsNo   : GALAXIES → (0, MAXREAL] U NULLS
Constellation : GALAXIES → CONSTELLATIONS U NULLS
Length    : GALAXIES → (0, MAXREAL] U NULLS

#### GALAXY_NEIGHBORHOOD
x      : GALAXY_NEIGHBORHOOD « INT (PK)
Galaxy  : GALAXY_NEIGHBORHOOD → GALAXIES
NeighborGalaxy : GALAXY_NEIGHBORHOOD → GALAXIES
LightYears  : GALAXY_NEIGHBORHOOD → (0, MAXREAL] U NULLS
Galaxy · NeighborGalaxy key
**Axiom GN1.** (∀x ∈ GALAXY_NEIGHBORHOOD)(Galaxy(x) ≠ NeighborGalaxy(x))

#### SOLAR_SYSTEMS
Sun     : SOLAR_SYSTEMS « CELESTIAL_BODIES (PK — shared with CELESTIAL_BODIES)
Galaxy  : SOLAR_SYSTEMS → GALAXIES U NULLS
Constellation : SOLAR_SYSTEMS → CONSTELLATIONS U NULLS
PlanetNo  : SOLAR_SYSTEMS → (0, MAXREAL]
StarsNo   : SOLAR_SYSTEMS → (0, MAXREAL]

```

Figure 20: Claude AI's schema fragment equivalent to the one from Figures 1 to 4 (2 of 4).

```

#### CELESTIAL_BODIES
#CC     : CELESTIAL_BODIES « INT (PK)
CelestialBody : CELESTIAL_BODIES → → UNICODE(255)+
CelestialBodyType : CELESTIAL_BODIES → Celestial_Body_Types
Constellation : CELESTIAL_BODIES → CONSTELLATIONS U NULLS
Galaxy      : CELESTIAL_BODIES → GALAXIES U NULLS
OrbitalCenter : CELESTIAL_BODIES → CELESTIAL_BODIES U NULLS
AvgDistance  : CELESTIAL_BODIES → [5, MAXREAL] U NULLS
SatellitesNo : CELESTIAL_BODIES → NAT U NULLS
Distance     : CELESTIAL_BODIES → (0, MAXREAL] U NULLS
ApparentMagnitude : CELESTIAL_BODIES → [-26.74, 36] U NULLS
RelativeMass  : CELESTIAL_BODIES → (0, MAXREAL] U NULLS
Temperature   : CELESTIAL_BODIES → (0, 28500] U NULLS
Water         : CELESTIAL_BODIES → (0, MAXREAL] U NULLS
Land          : CELESTIAL_BODIES → (0, MAXREAL] U NULLS
RatMass       : CELESTIAL_BODIES → (0, MAXREAL] U NULLS
RatPerRevolution : CELESTIAL_BODIES → (0, MAXREAL] U NULLS
RatPerRotation : CELESTIAL_BODIES → (0, MAXREAL] U NULLS
Diameter      : CELESTIAL_BODIES → [10, 999999999] U NULLS
AvgDensity    : CELESTIAL_BODIES → (0, MAXREAL] U NULLS
RatGravitation : CELESTIAL_BODIES → (0, MAXREAL] U NULLS
SolarSystem   : CELESTIAL_BODIES → SOLAR_SYSTEMS U NULLS
AggregationType : CELESTIAL_BODIES → {0, 1, 2, 3} U NULLS
*OrbitCenterName : CELESTIAL_BODIES → UNICODE(255) U NULLS -- computed
**Axiom CC1.** (∀x ∈ CELESTIAL_BODIES)(OrbitalCenter(x) ≠ #CC(x) ∨ OrbitalCenter(x) ∈ NULLS)

```

Figure 21: Claude AI's schema fragment equivalent to the one from Figures 1 to 4 (3 of 4).

```

### MULTI_STAR_SYSTEMS
x      : MULTI_STAR_SYSTEMS «INT (PK)
MultiStarSystem : MULTI_STAR_SYSTEMS → → UNICODE(255)+
*Constellation : MULTI_STAR_SYSTEMS → CONSTELLATIONS U NULLS -- computed
*Galaxy       : MULTI_STAR_SYSTEMS → GALAXIES U NULLS -- computed
*StarsNo      : MULTI_STAR_SYSTEMS → TINT(255) U NULLS -- computed

### MULTI_STAR_SYST_STARS
x      : MULTI_STAR_SYST_STARS «INT (PK)
MultiStarSystem : MULTI_STAR_SYST_STARS → MULTI_STAR_SYSTEMS
Star        : MULTI_STAR_SYST_STARS → → CELESTIAL_BODIES -- injective: each star in at most one system
Primary?    : MULTI_STAR_SYST_STARS → BIT U NULLS
**Axiom MS1.***( $\forall s \in \text{MULTI\_STAR\_SYSTEMS}$ )
|{ $x \in \text{MULTI\_STAR\_SYST\_STARS} \mid \text{MultiStarSystem}(x)=s.x \wedge \text{Primary?}(x)=1$  }|  $\leq 1$ 

```

Figure 22: Claude AI's schema fragment equivalent to the one from Figures 1 to 4 (4 of 4).

Conclusions

As expected, *MatBase* is still the best one of these four tools, both as accuracy and speed. Its great disadvantage, however, is that it may reverse engineer only MS Access and SQL Server dbs. The 2nd best is, by far, Claude AI, both as speed and as almost perfect accuracy. Moreover, it is not limited to only MS Access and SQL Server dbs, very probably just like ChatGPT and Gemini. Unfortunately, almost after any answer, you must wait some 10h before you get the right to ask another free of charge question. The 3rd best is ChatGPT. Unfortunately, it takes extremely long to get a full schema, which, moreover, is not that accurate. Fortunately, it is also providing the possibility to formalize db schemas using Category Theory formalism as well, but this is beyond the scope of this research: it will be our next further work topic. Finally, Gemini is almost not useful at all for this endeavor but is very interesting for developing data intelligence apps. Future work will evaluate Claude AI, ChatGPT, and Gemini for legacy database reverse engineering.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Acknowledgements

This research was not sponsored by anybody and nobody other than its authors contributed to it. The corresponding author always recalls with pleasure the contributions made by some of his outstanding former students: Lavinia Crasovschi for the (E)MDM, Adrian Mocanu and Sabina-Maria Motoc for *MatBase*.

References

- Mancas C (2013) Should Reverse Engineering Remain a Computer Science Cinderella? J. Inform. Tech. Soft. Eng pp. 1-3.
- Mancas C (2019) *MatBase – A Tool for Transparent Programming while Modeling Data at Conceptual Levels*. In: Proc. 5th Int. Conf. on Comp. Sci. & Inf. Techn. (CSITEC 2019), AIRCC Pub. Corp. Chennai, India 15-27.
- Mancas C (2020) *MatBase Metadata Catalog Management*. Acta Scientific Computer Sciences 2(4): 25-29.
- OpenAI. ChatGPT (2026). <https://chatgpt.com/>
- Google LLC. Gemini AI (2026). https://chat.aichatapp.ai/onboarding?landing=gemini&utm_source=google&utm_medium=cpc&utm_campaign=22724736430&utm_term=gemini%20ai&utm_adgroup=187207088087&utm_network=g&utm_matchtype=b&utm_loc_physical_ms=9193633&gad_source=1&gad_campaignid=22724736430&gbraid=0AAAAA1RoX3ghVrd_XhWfc3aChdH-GmeG&gclid=Cj0KCQjwj470BhCmARIsAF5wUEFhqT w91s549fLjsz2A5jxFk1M0qNSfdgCFjjsMfltAsbY1p1TnLXgaAvzcEA Lw_wcb
- Anthropic Claude AI (2026) <https://claude.ai>
- Mancas C (2024) The (Elementary) Mathematical Data Model revisited. PriMera Scientific Engineering 5(4): 78-91.
- Pinter C C (2014) *A Book of Set Theory*. Dover Pub. Inc., Mineola, NY.
- Heil J (2021) *First-Order Logic. A Concise Introduction*. 2nd Edition. Hackett Pub. Co. Inc., Indianapolis / Cambridge, U.S.A.
- Kroger F and Merz S (2008) *Temporal Logic and State Systems*. Springer-Verlag, Berlin Heidelberg, Germany.
- Abiteboul S, Hull R, Vianu V (1995) *Foundations of Databases*. Addison-Wesley, Reading, MA.
- Mancas C (2021) On Computing Transitive Closures in *MatBase* (Does HRH Charles Prince of Wales Also Have Cuman Blood?). JMEST 8(8): 14499-14513.
- Codd EF (1970) A relational model for large shared data banks. CACM 13(6): 377-387.
- Mancas C (2015) *Conceptual Data Modeling and Database Design: A Completely Algorithmic Approach*. Volume 1: The Shortest Advisable Path." Apple Academic Press, Waretown, NJ.

15. Chen PP (1976) The entity-relationship model. Toward a unified view of data. *ACM TODS* 1(1): 9-36.
16. Thalheim B (2000) *Entity-Relationship Modeling: Foundations of Database Technology*. Springer-Verlag, Berlin, Germany.
17. Mancas C (2020) On Modelware as the 5th Generation of Programming Languages. *Acta Scientific Computer Sciences* 2(9): 24-26.
18. Mancas C, Dragomir S, Crasovschi L (2003) On modeling First Order Predicate Calculus using the Elementary Mathematical Data Model in MatBase DBMS. In Proc. IASTED AI 2003 MIT Conf. on Applied Informatics, 1197-1202, Innsbruck, Austria. Acta Press: Calgary, Canada.
19. Almarashdi Hanan Shaher, Adeeb M Jarrah, Othman Abu Khurma, Serigne Mbaye Gningue (2024) Unveiling the potential: A systematic review of ChatGPT in transforming mathematics teaching and learning. *Eurasia Journal of Mathematics, Science and Technology Education* 20(12): 2555.
20. Pepin B, Buchholtz N, Salinas Hernández U (2025) A Scoping Survey of ChatGPT in Mathematics Education. *Digit Exp Math Educ* 11(1): 9-41.
21. Feng T, Trieu Trinh, Garrett Bingham, Jiwon Kang, Shengtong Zhang, et al. (2026) Semi-Autonomous Mathematics Discovery with Gemini: A Case Study on the Erdős Problems.
22. Woodruff D P et al. (2026) Accelerating Scientific Research with Gemini: Case Studies and Common Techniques.
23. Anthropic. Education Report: How university students use Claude (2025)
<https://www.anthropic.com/news/anthropic-education-report-how-university-students-use-claude>
24. Mancas C and Mancas D C (2026) From (Elementary) Mathematical Data Model Schemas to Safe Blazor Web Applications with Claude AI. *PriMera Scientific Engineering J.* 8.4: 16-37. DOI: 10.56831/PSEN-08-261
25. Hart A (2025) Can LLMs Write Mathematics Papers? A Case Study in Reservoir Computing pp. 1-10.
26. Microsoft Corp. SQL Server Migration Assistant for Access (AccessToSQL) (2026).
<https://learn.microsoft.com/en-us/sql/ssma/access/sql-server-migration-assistant-for-access-accesssql?view=sql-server-ver17>
27. Mancas C (2024) On Enforcing Existence and Non-Existence Constraints in MatBase. *PriMera Scientific Engineering* 4(6):4-12.