

conexus[®] AI

Validating LLM-generated SQL code



A mathematical approach

Texas Data Day 2025

How do we know that LLM-generated code is correct?

LLM-generated code is becoming more prevalent.

Having humans examine/test LLM-generated code precludes full automation.

Having LLM-generated tests/examiners “begs the question”.

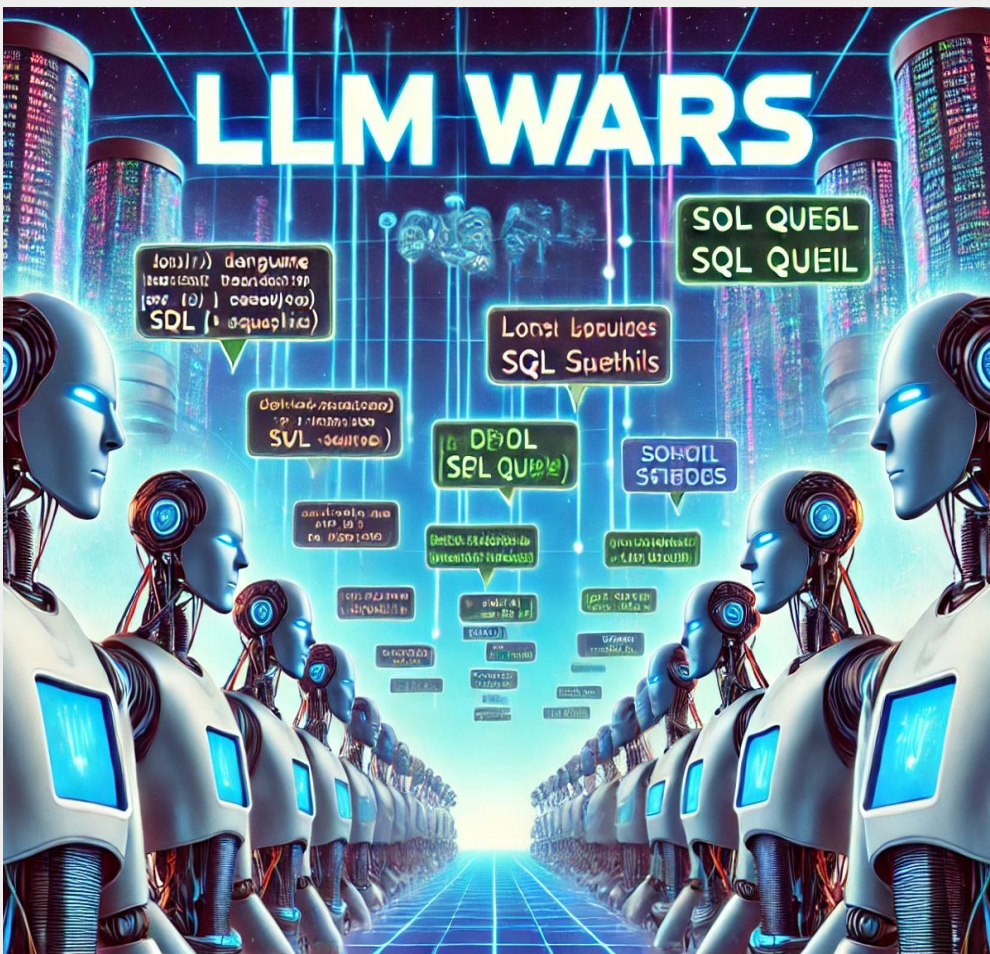
What can we do? For SQL et al, we can use algorithms to prove that SQL code will respect data integrity.

“Given a set of tables with primary and foreign keys, will this SQL code will populate a new set of tables with primary and foreign keys without throwing an error?”

Just adding more RAG is like insisting we don't need seatbelts, just safer drivers!

Outline:

- Describe results of an LLM-competition, indicating which errors are statically detectable
- Describe a new SQL verification technology, <http://sql.ai>
- Describe a fully automated LLM-generating SQL case study from the US military
- Describe query migration, a new capability for composing SQL code
- Seeking community engagement, please email me at ryan@conexus.com



Part 1: LLM Wars

Joint work with



Kato Steven Mubiru

kato.steven60@gmail.com
linkedin.com/in/katostevenmubiru/



Glorry Sibomana

glorrysibomana758@gmail.com
linkedin.com/in/glorry-sibomana/



Julius Hamilton

julius.hamilton100@gmail.com
linkedin.com/in/julius-hamilton-806944228

```
CREATE TABLE humans (
  name varchar PRIMARY KEY,
  mom varchar FOREIGN KEY ( mom)
    REFERENCES humans (name));
```

```
CREATE TABLE people (id varchar PRIMARY KEY);

CREATE TABLE parents (
  id varchar PRIMARY KEY,
  parent_id varchar NOT NULL,
  FOREIGN KEY ( parent_id) REFERENCES people (id));

CREATE TABLE gparents (
  id varchar PRIMARY KEY,
  parent_id varchar NOT NULL,
  gparent_id varchar NOT NULL,
  FOREIGN KEY ( parent_id) REFERENCES people (id),
  FOREIGN KEY ( gparent_id) REFERENCES people (id));

CREATE TABLE ggparents (
  id varchar PRIMARY KEY,
  parent_id varchar NOT NULL,
  gparent_id varchar NOT NULL,
  ggparent_id varchar NOT NULL,
  FOREIGN KEY ( parent_id) REFERENCES people (id),
  FOREIGN KEY ( gparent_id) REFERENCES people (id),
  FOREIGN KEY ( ggparent_id) REFERENCES people (id));
```

We asked 48 LLMs to unroll a single table with a cyclic foreign key into four tables connected by foreign keys.

```
CREATE TABLE humans (
  name varchar PRIMARY KEY,
  mom varchar REFERENCES humans (mom);
);
```

<u>name</u>	<u>mom</u>
alice	barbara
brenda	claire
barbara	claire
claire	debbie
debbie	NULL



```
CREATE TABLE people (
  id varchar PRIMARY KEY;
);

CREATE TABLE parents (
  id varchar PRIMARY KEY,
  parent_id varchar NOT NULL,
  FOREIGN KEY (parent_id) REFERENCES people (id);
);

CREATE TABLE gparents (
  id varchar PRIMARY KEY,
  parent_id varchar NOT NULL,
  gparent_id varchar NOT NULL,
  FOREIGN KEY (parent_id) REFERENCES people (id),
  FOREIGN KEY (gparent_id) REFERENCES people (id);
);
```

<u>id</u>
alice
brenda
barbara
claire
debbie

<u>id</u>	<u>parent_id</u>
alice	barbara
brenda	claire
barbara	claire
claire	debbie

<u>id</u>	<u>parent_id</u>	<u>gparent_id</u>
alice	barbara	claire
brenda	claire	debbie
barbara	claire	debbie

For example...

Correct Answer

```
CREATE TABLE humans (
  name varchar PRIMARY KEY,
  mom varchar REFERENCES humans (mom)
);
```

name	mom
alice	barbara
brenda	claire
barbara	claire
claire	debbie
debbie	NULL

```
INSERT INTO people (id)
SELECT h.name
FROM humans AS h;
```

```
INSERT INTO parents (id, parent_id)
SELECT h.name, h.mom
FROM humans AS h
WHERE h.mom IS NOT NULL;
```

```
INSERT INTO gparents
(id, parent_id, gparent_id)
SELECT h1.name, h1.mom, h2.mom
FROM humans AS h1, humans AS h2
WHERE h1.mom = h2.name
AND h2.mom IS NOT NULL;
```

```
INSERT INTO ggparents
(id, parent_id, gparent_id, ggparent_id)
SELECT h1.name, h1.mom, h2.mom, h3.mom
FROM humans AS h1, humans AS h2,
humans AS h3 WHERE h1.mom = h2.name AND
h2.mom = h3.name AND h3.mom IS NOT NULL;
```

id
alice
brenda
barbara
claire
debbie

```
le (id) PRIMARY KEY);
```

```
nts (
```

id	parent_id
alice	barbara
brenda	claire
barbara	claire
claire	debbie

```
people (id));
```

```
ARI REF;
```

```
ar NOT NULL;
```

parent_id	gparent_id
barbara	claire
claire	debbie
claire	debbie

```
ple (id),
```

```
ople (id));
```

```
rents (
```

```
ARY KEY,
```

```
har NOT NULL,
```

parent_id	gparent_id	ggparent_id
barbara	claire	debbie

```
id),
```

```
parent_id) REFERENCES people (id),
```

```
parent_id) REFERENCES people (id));
```

LLM prompt specifics

- LLMs were given schema definitions
- LLMs were not given example data
- LLMs were asked not to generate comments or commentary
- LLMs were asked to use the simplest SQL possible
- LLMs were asked to populate each target table independently

The prompt and full experimental data will be made available, along with a blog post and this presentation, after Texas Data Day.

You are a helpful AI assistant tasked with generating **plain SQL code** for data migration.

Task:

Write SQL queries to **populate the destination (target) tables from the source tables**, ensuring data integrity. The SQL code must **only** pull data from source tables and insert it into target tables, without mixing source and target tables in the same query.

Important Instructions:

- One 'INSERT INTO' Statement per Target Table:**
 - Write one 'INSERT INTO' statement for each target table.
 - Use a simple 'SELECT FROM WHERE' from **source tables only** to populate the target table.
- Source-Only Data for Target Tables:**
 - **Do NOT** join source tables with target tables.
 - All data for target tables must come directly from source tables without intermediary steps involving other target tables.
- Fully Qualified Column Names with Aliases:**
 - Fully qualify column names with table names in the format 'table_name.column_name'.
 - Always use table aliases (e.g., 'FROM SourceTable AS s').
- Keep SQL Simple:**
 - Avoid subqueries, or advanced SQL features. Do not use aggregation, 'GROUP BY', 'AVG', 'MIN', 'MAX', 'ORDER BY', 'CASE', 'COUNT', 'UPDATE', 'SET'
 - Use basic SQL operations: equality comparisons (=), 'AND'/'OR'/'NOT', 'IS NULL'
 - Prefer 'FROM' syntax to 'JOIN ON' syntax
- No Comments, Markdown, or Extra Formatting:**
 - **Do NOT** include any comments (e.g., no '--' or '/' ... */ lines).
 - **Do NOT** include any markdown formatting (e.g., no backticks or triple backticks).
 - Output **plain text SQL only**, without any extra symbols, explanations, or formatting.
- Output Format:**
 - Provide SQL code **ready to run** as plain text.
 - **Do NOT** include explanations, comments, or any markdown symbols.

LLM Wars Findings

Some LLMs insist on adding comments.(16%, 8/48)

- We removed comments from these responses and continued.
- Source vs target errors were ignored too

No syntax errors,

- but a tiny model did echo back the input schema definitions (1/48).

Four schema errors (bad column names) (8%, 4/48).

Thirteen data integrity errors (27%, 13/48)

- 7 NOT NULL errors, 6 UNIQUE KEY errors
- Such errors can potentially be prevented with static analysis, as described later!

The rest were correct (64%, 31/48)

CORRECT:

chatgpt-4o-latest
 claude-3-opus
 claude-3-sonnet
 claude-3.5-sonnet
 claude-3.5-sonnet-20240620
 gemini-1.5-pro
 gemini-1.5-pro-001
 gpt-4
 gpt-4-turbo
 gpt-4o
 gpt-4o-2024-05-13
 gpt-4o-2024-08-06
 llama-3.1-405b-chat
 llama-3.2-11b-chat
 mistral-large
 qwen-2-72b-instruct
 claude-3-haiku
 gemini-1.5-pro-002
 gemma-2-27b-it
 gpt-3.5-turbo
 llama-3-70b-chat
 llama-3.1-70b-chat
 llama-3.1-8b-chat
 llama-3.2-90b-chat
 mistral-7b-instruct-v0.2
 mistral-nemo
 mixtral-8x7b-instruct-v0.1
 qwen-2.5-72b-instruct

error: ambiguous column name: name

llama-3-8b-chat

error: near "FOREIGN": syntax error

llama-3.2-1b-chat

error: no such column: gparents.mom

ministral-3b

error: no such column: m.name

mistral-small

error: no such column: mom

mixtral-8x22b-instruct-v0.1

error: no such column: p.mom

gemini-1.0-pro-002

error: NOT NULL constraint failed: gparents.gparent_id

command-r-plus

error: NOT NULL constraint failed: parents.parent_id

gemini-1.5-flash

gemini-1.5-flash-001

gemini-1.5-flash-002

gemma-2-9b-it

llama-3.1-nemotron-70b-chat

llama-3.2-3b-chat

error: UNIQUE constraint failed: ggparents.id

gemini-1.0-pro

error: UNIQUE constraint failed: gparents.id

gemma-7b-it

gpt-4o-mini

mistral-7b-instruct-v0.3

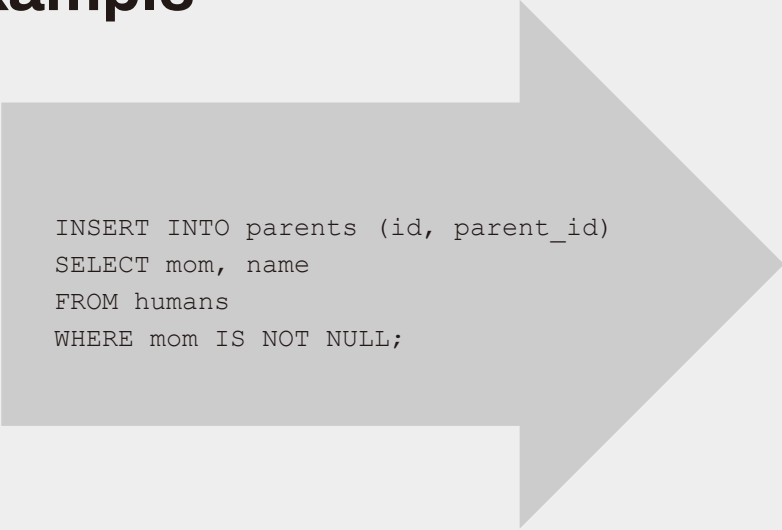
error: UNIQUE constraint failed: parents.id

gemini-1.0-pro-001

ministral-8b

Preventing Errors with Static Checking, an example

<u>name</u>	<u>mom</u>
alice	barbara
brenda	claire
barbara	claire
claire	debbie
debbie	NULL



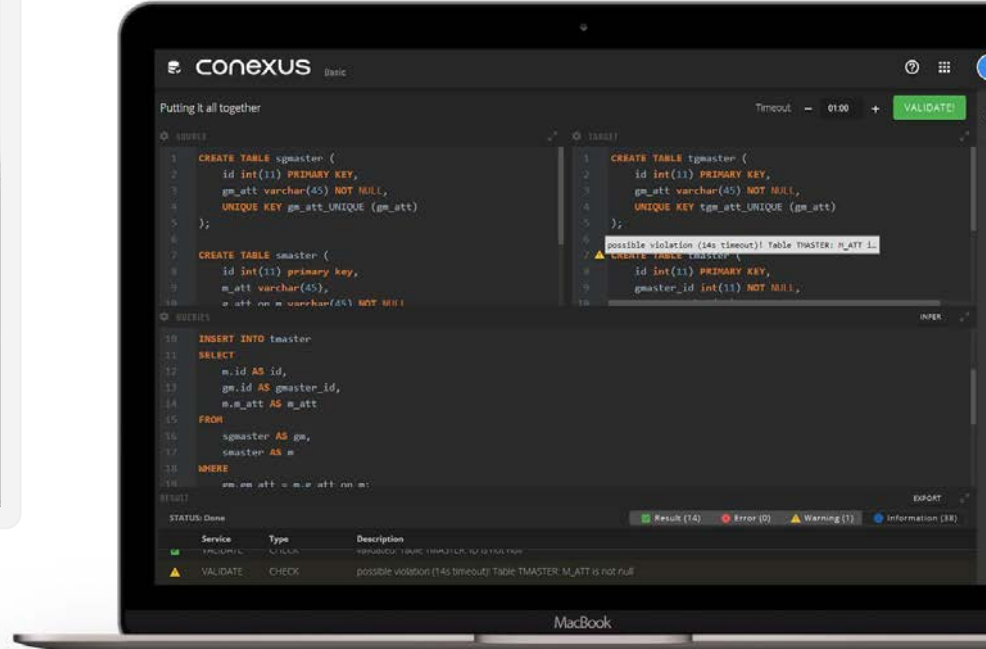
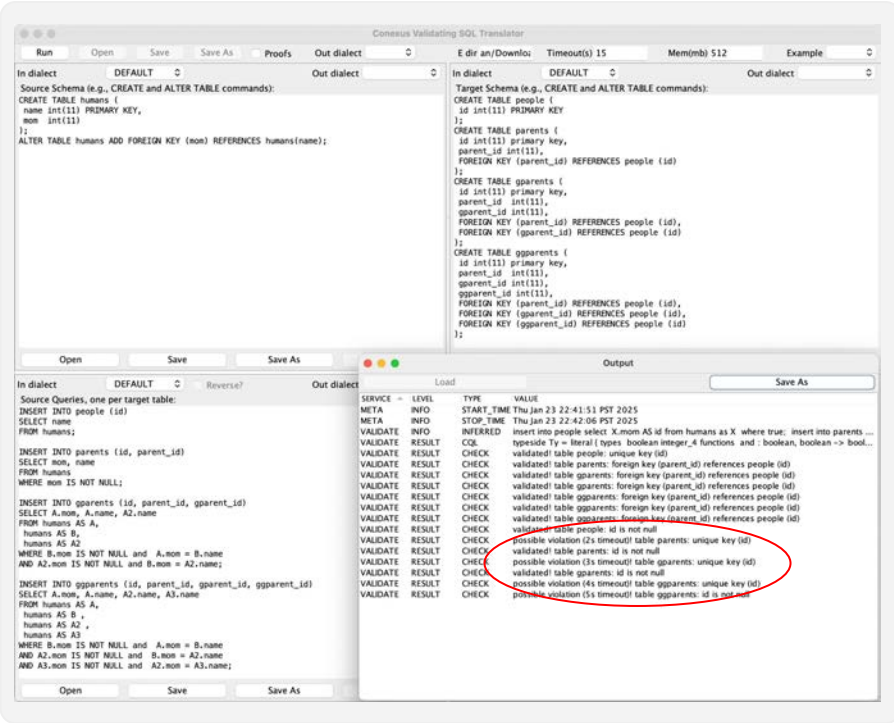
```
INSERT INTO parents (id, parent_id)
SELECT mom, name
FROM humans
WHERE mom IS NOT NULL;
```

<u>id</u>	<u>parent_id</u>
barbara	alice
claire	brenda
claire	barbara

Correct answer:

<u>id</u>	<u>parent_id</u>
alice	barbara
brenda	claire
barbara	claire
claire	debbie

Claim: this query does not guarantee the creation of a primary key of the humans table, and we can check this automatically, without data!



Compatible with



SQL Correctness Verifier

Problem: LLM-generated SQL code may be wrong; it may corrupt data integrity.

Analysis: Verifying correctness of SQL code by hand takes time, and verifying correctness of SQL code with LLMs introduces error. And testing only demonstrates absence of individual errors, not correctness.

Solution: Use Conexus to verify the correctness of SQL code with respect to data integrity using mathematical algorithms to get 100% confidence.

Benefit: LLM-generated SQL code can now be trusted to run, or fed back to the LLM to try again.

Bonus: This capability can be used with arbitrary SQL code, not just code generated from LLMs.

How the SQL validator works: good old fashioned symbolic AI

LLM

Example Prompt given to LLM

"Write SQL code to populate

from

```
CREATE TABLE tgmaster (id int(11) PRIMARY KEY, gm_att
varchar(45) NOT NULL, UNIQUE KEY (gm_att));

CREATE TABLE tmaster (id int(11) PRIMARY KEY, gmaster_id
int(11) NOT NULL, m_att varchar(45) NOT NULL, FOREIGN KEY
(gmaster_id) REFERENCES tgmaster (id) );

CREATE TABLE tdetail (id int(11) PRIMARY KEY, master_id int(11)
NOT NULL, d_att varchar(45) NOT NULL, FOREIGN KEY
(master_id) REFERENCES tmaster (id));
```

```
CREATE TABLE sgmaster (id int(11) PRIMARY
KEY, gm_att varchar(45) NOT NULL, UNIQUE
KEY (gm_att));

CREATE TABLE smaster (id int(11) primary key,
m_att varchar(45) NOT NULL, g_att_on_m
varchar(45) NOT NULL);

CREATE TABLE sdetail (id int(11) primary key,
master_id int(11) NOT NULL, d_att varchar(45)
NOT NULL);"
```

Example Response from LLM

```
"insert into tgmaster
select g.id as id, g.gm_att as gm_att
from sgmaster as g;

insert into tmaster
select m.id as id, gm.id as gmaster_id, m.m_att as m_att
from sgmaster as gm, smaster as m
where gm.gm_att = m.g_att_on_m;

insert into tdetail
select d.id as id, m.id as master_id, d.d_att as d_att
from sgmaster as gm, smaster as m, sdetail as d
where d.master_id = m.id and gm.gm_att = m.g_att_on_m;"
```

Individual correctness proofs / refutations of data integrity

Correctness Verifier

SERVICE	LEVEL	TYPE	VALUE
META	INFO	START TIME	Fri Feb 03 09:20:51 PST 2023
META	INFO	STOP TIME	Fri Feb 03 09:20:51 PST 2023
MICRODATE	RESULT	MICRODATE	select v2.id as id, v2.d as master_id from sdetail v2, smaster v2 where v2...
VALIDATE	INFO	INFERRED	insert into tgmaster select X.gm_att as gm_att, X.id as id from sgmaster as X where true...
VALIDATE	RESULT	SQL	Upside: Tr = literal 1 type: Boolean string_4 string_45 functions and : boolean, boolea...
VALIDATE	RESULT	CHECK	validated! table sgmaster: unique key (id)
VALIDATE	RESULT	CHECK	validated! table tmaster: foreign key (gmaster_id) references tgmaster (id)
VALIDATE	RESULT	CHECK	validated! table sgmaster: unique key (gm_att)
VALIDATE	RESULT	CHECK	validated! table tgmaster: id is not null
VALIDATE	RESULT	CHECK	validated! table sgmaster: gm_att is not null
VALIDATE	RESULT	CHECK	validated! table tmaster: unique key (id)
VALIDATE	RESULT	CHECK	validated! table tdetail: foreign key (master_id) references tmaster (id)
VALIDATE	RESULT	CHECK	validated! table smaster: id is not null
VALIDATE	RESULT	CHECK	validated! table tmaster: gmaster_id is not null
VALIDATE	RESULT	CHECK	validated! table smaster: m_att is not null
VALIDATE	RESULT	CHECK	validated! table tdetail: unique key (id)
VALIDATE	RESULT	CHECK	validated! table tdetail: id is not null
VALIDATE	RESULT	CHECK	validated! table smaster: master_id is not null
VALIDATE	RESULT	CHECK	validated! table tdetail: d_att is not null

LLM Wars Conclusions

Negation is still problematic for LLMs

- “Following instructions” still problematic for LLMs

Correctness needs to improve – 66% on a basic example!

Static analysis provides a way to do so

- We don’t just think so, we know so!
- Now for a case study.

Read more at <http://sql.ai>

Fun fact:

Follow-up experiments on more complex examples led to much larger variance in LLM responses. More to come!

Help build community!
Email ryan@conexus.com

PART 2

A fully automated case study

The DARPA BRASS IMMORTALS project (2017) sought to build long-lived, resource-aware, self-updating software using LLM precursors.

During drone flight, an “LLM” would create SQL code to update avionics.

This code needs to be checked by a computer within in seconds to be safe to run.

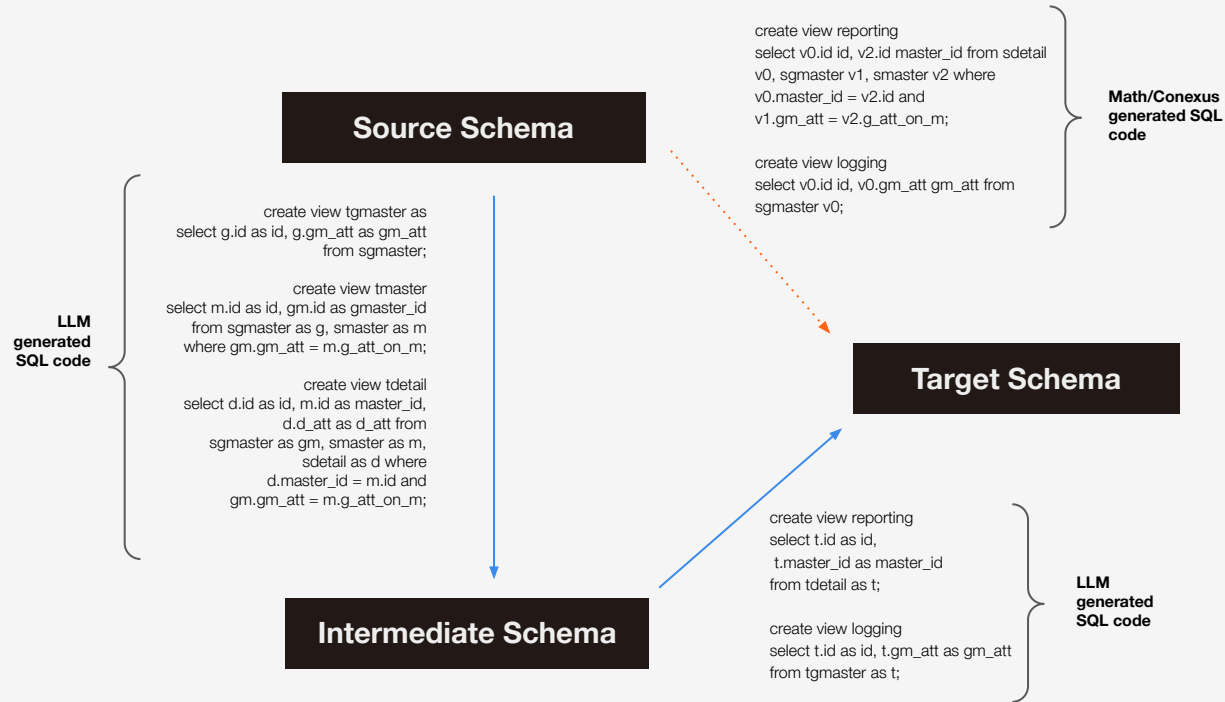
It is too expensive for the LLM to constantly generate new code, so the project also involved composing SQL code (part 3).

Conclusion: fully automated, computer-checked LLM code generation possible in real time, life and death situations.

Part 3

Given that LLM-generated SQL code requires work to verify, let's try to compose correct SQL code so we don't have to verify over and over.

API: 2 sets of SQL queries in, 1 set of SQL queries out



SQL Composition

Problem: To generate good SQL code, LLM prompts must be “chunked; split into pieces and the results re-assembled.”

Analysis: Re-assembling the results by hand take time, and re-assembling the results by LLM introduces error.

Solution: Use Conexus to compose SQL code perfectly and automatically, without introducing biases, errors, or assumptions, using mathematical algorithms.

Benefit: LLM-generated SQL code can be used once chunking is done.

Bonus: This capability can be used with arbitrary SQL code, not just code generated from LLMs.

Example Fortune 50 Deployment

WellHeadSystem_V1 (10)					
Row	Class	Com...	ID	Idan...	IDJob
0	?0	?1	148F6...	0099...	?2
1	?13	?14	B8C6...	A3E8...	?15
2	?26	?27	44F80...	22AC...	F45FF..
3	?40	?41	FDD3...	842FB...	?42

New Source

WellHead (10)			
Row	BusinessU...	CreatedBy	CreationD...
0	ignorebuf	tipt	7/1/20...
1	ignorebuf	ubhi	12/20/20...
2	ignorebuf	mpjgg	2/17/20...
3	ignorebuf	tugp	12/19/20...

WV100CalcUnitSus_WVWellHead (10)					
Row	Com	Dep...	DTT...	DTT...	DTT...
0	?0	7045	?1	?2	6/27...
1	?12	4372	?13	?14	12/1...
2	?24	?25	?26	?27	2/16...
3	?37	6888.8	?38	?39	5/21...

Old Source

Target

Query (upgrade)

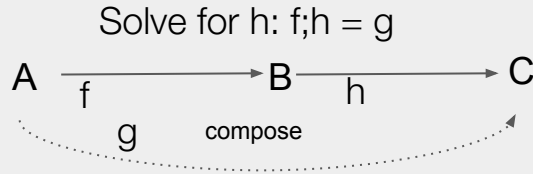
Query (new)

Query (old)

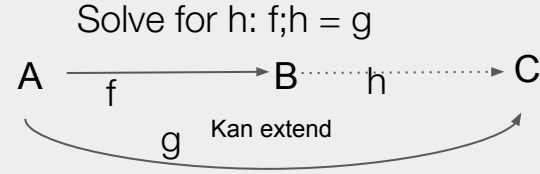
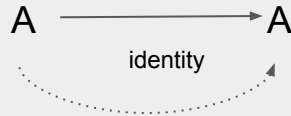
We want to mathematically migrate Query (old) to Query (new)

Four ways to compose SQL queries

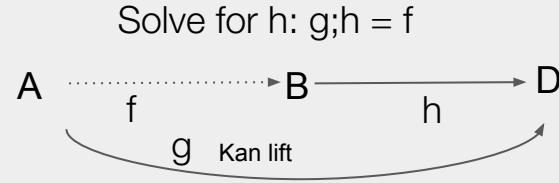
The mathematics of category theory tells us that there are four ways to compose SQL queries. This lets us build virtual data fabrics/meshes.



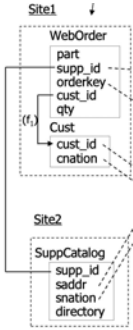
EASY



HARD



Source



Row	CUST_ID	CNATION
0	cust1	usa
1	cust2	uk
2	cust3	jpy

Row	SUPP_ID	SADDR	SNATION	DIRECTORY
3	supp1	s1addr	usa	dir1
4	supp2	s2addr	india	dir2
5	supp3	s3addr	dir3	dir3
6	supp4	s4addr	mexico	dir4

Row	PART	SUPP_ID	ORDERKEY	CUST_ID	QTY
7	part1	supp1	order1	cust1	1
8	part2	supp2	order2	cust1	2

```
select w.CUST_ID as col1,
       w.SUPP_ID as col2,
       sc.SADDR as col3,
       sc.SNATION as col4
from WEBORDER as w,
     SUPPCATALOG as sc
where w.SUPP_ID = sc.SUPP_ID;
```

```
forall c1 c2 : "CUST" where c1.CUST_ID = c2.CUST_ID -> where c1 = c2 #pk
forall w : "WEBORDER" -> exists c : "CUST" where w.CUST_ID = c.CUST_ID #fk
forall c1 c2 : MasterCust where c1.cust_id = c2.cust_id -> where c1 = c2 #pk
forall s1 s2 : MasterSupp where s1.supp_id = s2.supp_id -> where s1 = s2 #pk
forall sc : Supp2Cust -> exists ms : MasterSupp where sc.supp_id = ms.supp_id #fk
forall sc : Supp2Cust -> exists mc : MasterCust where sc.cust_id = mc.cust_id #fk

#source to target mapping rule
forall w : "WEBORDER"
  c : "CUST"
  sc : "SUPPCATALOG"
where w.CUST_ID = c.CUST_ID w.SUPP_ID = sc.SUPP_ID
-> exists ms : MasterSupp
      s2c : Supp2Cust
      mc : MasterCust where
  ms.supp_id = sc.SUPP_ID ms.saddr = sc.SADDR ms.snation = sc.SNATION
  s2c.supp_id = sc.SUPP_ID s2c.orderkey = w.ORDERKEY s2c.cust_id = c.CUST_ID
  mc.cust_id = c.CUST_ID mc.cnation = c.CNATION

forall ms : MasterSupp -> #target to source mapping rule
exists sc : SUPPCATALOG where
  ms.supp_id = sc.SUPP_ID ms.saddr = sc.SADDR ms.snation = sc.SNATION

forall c : "CUST" mc : MasterCust where c.CUST_ID = mc.cust_id ->
where c.CNATION = mc.cnation #target and source mapping rule (not pk or fk)
```

Rules describing how third party related the Source and Target data models (human or machine curated)

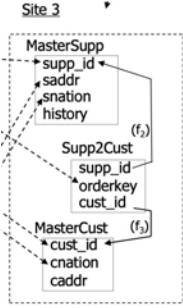
DATA MIGRATION

Row	col1	col2	col4	col3
0	cust1	supp1	usa	s1addr
1	cust1	supp2	india	s2addr

The validation report shows that the translated SQL gives same answer on old and new data.

SQL TRANSLATION

Target



Row	CUST_ID	CNATION
0	cust1	usa
1	cust3	jpy
2	cust2	uk

Row	supp_id	cnation	caddr
3	supp1	usa	dir1

Row	supp_id	saddr	snation	history
4	supp1	s1addr	usa	71
5	supp2	s2addr	india	72

Row	SUPP_ID	SADDR	SNATION	DIRECTORY
6	supp1	s1addr	usa	dir1
7	supp2	s2addr	india	dir2
8	supp4	s4addr	mexico	dir4
9	supp3	s3addr	en	dir3

Row	supp_id	orderkey	cust_id
10	supp1	order1	cust1
11	supp2	order2	cust1

Row	PART	SUPP_ID	ORDERKEY	CUST_ID	QTY
12	part1	supp1	order1	cust1	1
13	part2	supp2	order2	cust1	2

```
select v11.CUST_ID as col1,
       v11.SUPP_ID as col2,
       v3.saddr as col3,
       v3.snation as col4
from WEBORDER as v11,
     MasterSupp as v3
where v3.supp_id = v11.SUPP_ID;
```

Conclusions

Commercial LLMs still not safe for unsupervised use

Negation and instruction following still problematic

Symbolic AI / computer algebra lets us check LLM-generated SQL code

Deployed for US military drones

“Spray and Pray” is now a feasible “neuro-symbolic” AI use case

We expect an explosion of domain-specific LLM-enabled schema mappers

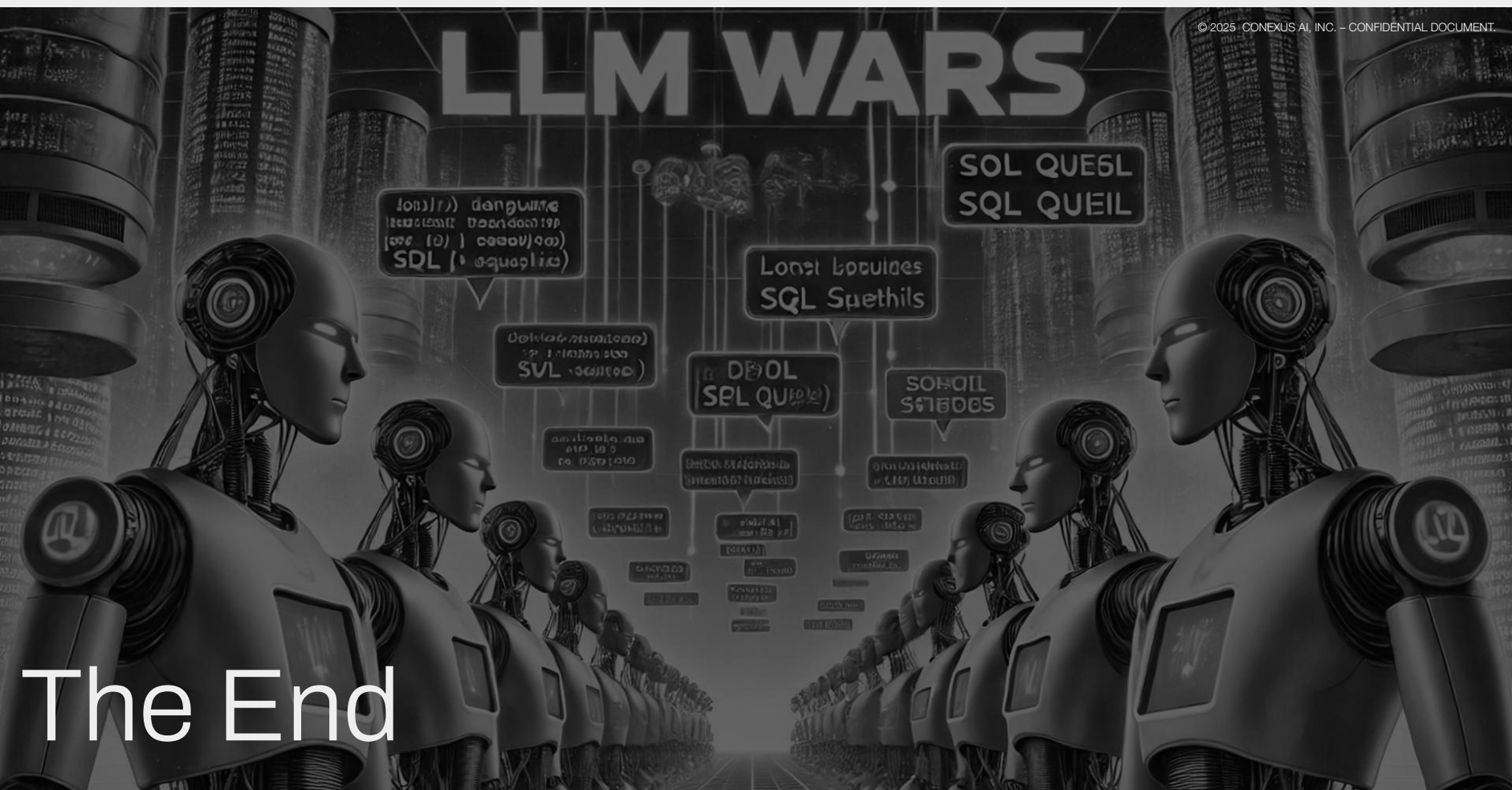
Safe AI enabled by constraining problem space (to e.g. SQL integrity)

We expect more validators too!

We are looking for community engagement!

Connect with Ryan at ryan@conexus.com

LLM WARS



SQL (SQL)
SQL (SQL)
SQL (SQL)

SOL QUE6L
SQL QUEIL

SQL Spethils

SQL (SQL)

SQL QU()

SQL (SQL)

SQL (SQL)

SQL (SQL)

SQL (SQL)

SQL (SQL)

SQL (SQL)

SQL (SQL)

SQL (SQL)

SQL (SQL)

SQL (SQL)

The End