



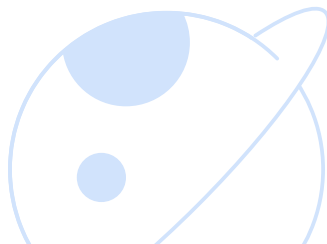
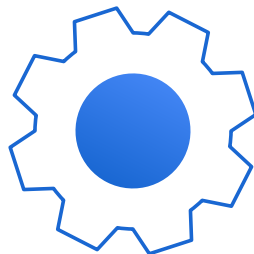
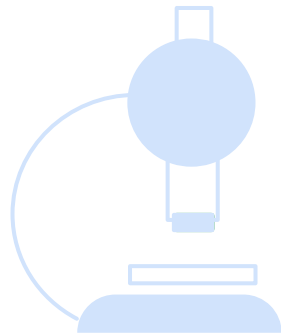
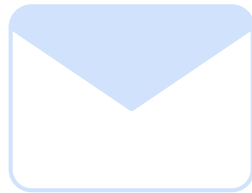
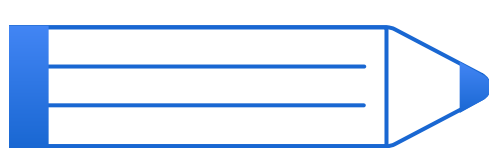
Google for Education
Higher Ed Faculty
AI Fellowship

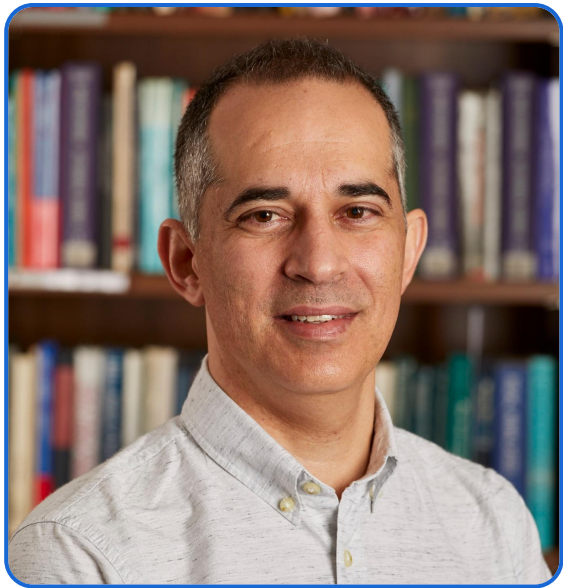


Google for Education
Higher Education Faculty AI Fellowship

Programming education in the age of AI agents

23 October 2026





Dr Marcos Barreto

Associate Professor in Data Science
London School of Economics and
Political Science (LSE)

w: <https://marcosebarreto.github.io>



Support and Funding



Anuhya Thota

BSc Data Science
LSE



Mengning Chen

BSc Mathematics, Statistics and Business
LSE



Google for Education
Higher Education Faculty AI Fellowship



Talent Development Awards 2024-2025



Department of
Statistics



Challenge

What's is left to programmers if AI can generate high-quality, complex code?

The key skills for students remain the same:

- Formulating problem requirements in detail;
- Testing/recognising problems in the code;
- Refining code; and
- Designing an entire system.

(Adapted from Michael Hansen. *Top 10 things I learned from ChatGPT*. AMSTATS NEWS, September 2023. https://magazine.amstat.org/blog/2023/09/01/hansen_chatgpt/)



Google for Education
Higher Education Faculty AI Fellowship

DOI:10.1145/3570220

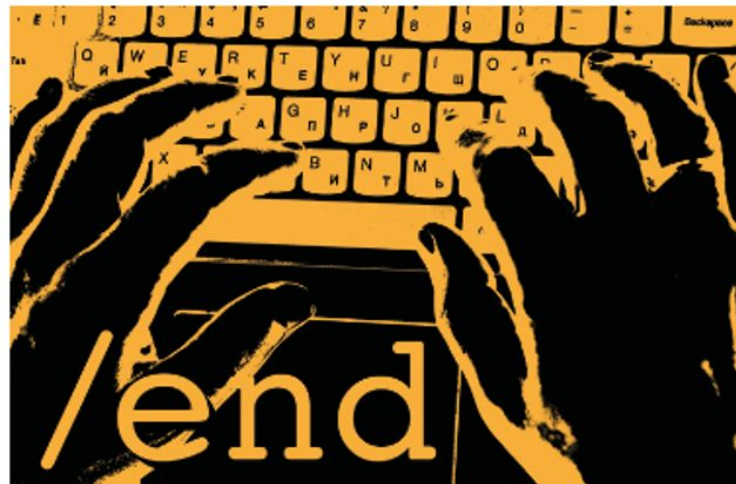
Matt Welsh

Viewpoint

The End of Programming

The end of classical computer science is coming, and most of us are dinosaurs waiting for the meteor to hit.

'I believe the conventional idea of “writing a program” is headed for extinction, and indeed, for all but very specialized applications, **most software**, as we know it, **will be replaced by AI systems** that are *trained* rather than *programmed*.'



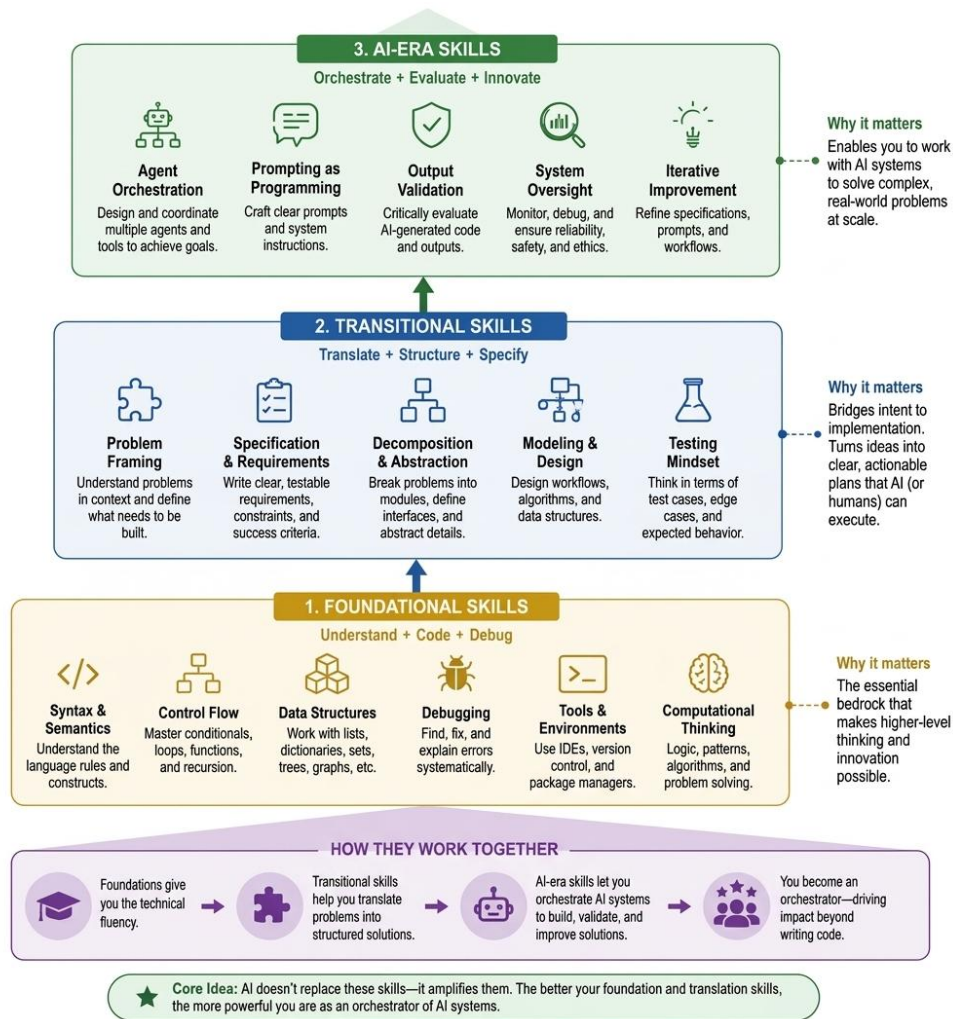
COMMUNICATIONS OF THE ACM, JANUARY 2023. VOL. 66 | NO. 1

Proprietary + Confidential

Proposed work

Define baseline **AI-assisted programming skills**:

- Foundational (*non-negotiable*)
- Transitional
- Agentic AI



Proposed work *(cont'd)*

Teaching resources for typical data science tasks

- **Database modelling and programming**

- Company (Antigravity 2.0, lecture)
- Private equity (Antigravity 2.0)
- e-Commerce (Antigravity SDK)

- **Supervised and unsupervised machine learning**

- (Regression) Prediction of loan interest rate (Antigravity 2.0)
- (Classification) Image classification (Antigravity 2.0, PyTorch)
- (Unsupervised) Customer segmentation / Market basket analysis (Antigravity 2.0/SDK)



Example: database modelling

Structure and expected
artefacts

*[text] denotes who is responsible for
producing the artefacts*

```
CompanyDB/
├── .agents/
│   └── skills/
│       ├── companydb_modelling/
│       │   └── SKILL.md
│       ├── companydb_data_insertion/
│       │   └── SKILL.md
│       └── companydb_querying/
│           └── SKILL.md
├── assets/
│   ├── data
│   └── er_diagram.png
├── db/
├── references/
│   └── technical_specification.md
├── code/
│   ├── Python scripts
│   └── Jupyter notebooks
├── docs/
│   ├── implementation_plan.md
│   ├── validation_plan.md
│   ├── task_list.md
│   ├── walkthrough.md
│   └── report.pdf
├── README.md
└── database_description.md

# [instructor/student] Project repository
# Standard folder for agent skills
# Agent skills
# [student] Translates specs to schema and verifies setup
# [student] Ingests test data and verifies rules
# [student] Writes SELECT queries and cross-checks with Python

# Project assets
# [instructor/student] Raw or synthetic data
# [student] ER database model
# [student/AI agent] Output folder for SQLite database files
# Technical specification documents
# [student] Source of truth for database schema and constraints
# [AI agent] Generated code
# [AI agent] Expected one script per skill
# [AI agent] Consolidated Python code (Google Colab compatible)
# Process-based documents
# [AI agent] Project implementation plan
# [AI agent] Project validation plan
# [AI agent] Agent's task list
# [AI agent] Project stepwise report
# [student] Reflective report
# [instructor] Exercise specification
# [instructor] Database scenario & analytical questions
```



Request prompt

User Request: 'companydb_modelling' skill to create the database data based on the referenced workspace artefacts. Do not go beyond the instructions in the skills file; more specifically, do not create any code to load data into the database or implement any query. This is the first step in the database modelling exercise where we will only create an empty database matching the provided description and schema. Before generating the database schema and any code files, generate implementation plans and validation plans and wait for my review and approval.

Skills file

[illegible]

Final artefacts

I have completed all the tasks according to the updated database modelling instructions and your feedback.

Here is a summary of the accomplishments:

- ## 1. Documentation Written:
- `decompiletranslation_plan.md`: Details the design mappings of entities, attributes, relationships, and recursive links to physical tables.
 - `decompilevalidation_plan.md`: Describes the structural testing and how to run verification tests programmatically in Google Colab.
 - `docxtask_list.md`: Tracks completed and remaining work.
- ## 2. Schema Definition Script:
- Written to `@ code/create_schema.py`.
 - This script constructs `db+company.db` and maps the correct tables with your requested adjustments:
- ```

--tbl_name is enforced as NOT NULL.
--tbl_name uses 'on' instead of 'off' for
--Relationship in the dependent table is now as VARCHAR(30) .

```

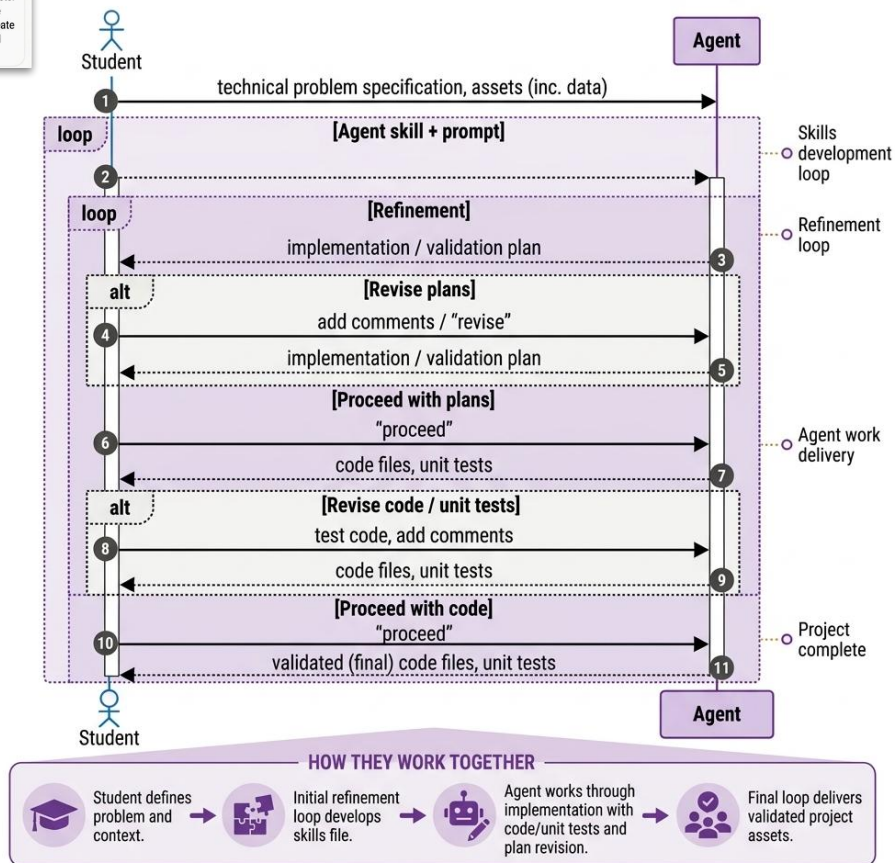
- Written to `codeverify_schema.py`.
- Uses only Python's standard `unittest` and `sqlite3` libraries and is fully prepared to be imported/called directly inside Google Colab cells without executing command-line scripts.
- Checks table structures, data types, nullabilities, and foreign key relations.

**4. Execution and Validation:**

- Ran the schema generation script successfully.
- Ran the verification test suite, which executed and passed all 9 test cases successfully.

You can view the detailed results in the [Walkthrough Report](#). The empty database schema is now fully initialized and validated, ready for the next steps!

## PROJECT DEVELOPMENT WORKFLOW: STUDENT & AGENT



## Implementation/Validation plans

### Implementation & Validation Plan: CompanyDB Database Schema

This plan describes the mapping of the Company DB E-R model (located in `assets/er_diagram.png`) and reference specifications (in `references/technical_specification.md`) to a SQL database schema. It also outlines the validation plan to verify the correctness of the generated empty database.

### 1. E-R Diagram vs. Technical Specification Review

A thorough comparison has been performed between the E-R diagram ( `assets/er_diagram.png` ), the database description ( `database_description.md` ) and the technical specifications ( `references/technical_specification.md` ).

### 1. Entities & Attributes Match

- **EMPLOYEE**: Matches primary. Composite attribute *(Name)* composed of *Forename*, *Middle*, *Surname* is correctly flattened into separate columns in the specifications. *Sex* is correctly marked as the primary key.
- **PROJECT**: Matches primary. *ProjectName* is correctly marked as the primary key. *Location* is a multi-valued attribute (represented by double oval), which maps to a separate table *dept\_locations* in the specifications.
- **NUMBER\_EMPLOYEES**: is a derived attribute (dashed oval) and is identified from primary. Persistence as per standard practices.
- **PROJECT**: Matches primary. *ProjectName* is the primary key. *Project* is unique. *Location* is simple.
- **DEPENDENT**: Matches per key. Represented as a weak entity (double rectangle) with *DependentName* and *Sex* as its keys. *DependentName* is identifying relationship-key. The composite primary key is correctly identified. This completed. This maps to a corresponding physical table with proper constraints.

- **Works\_for** (EMPLOYEE

- Mgr\_ssn is an attribute of DEPARTMENT acting as the employee's Ssn who manages the department. The employee must have a manager, so this attribute cannot be null.

- **Dependents\_of** (EMPLOYEE to DEPENDENT): Weak entity relationship. Map to foreign key **Essn** in **dependent** table, which is also part of the primary key.

**Mismatches/issues identified:** None. The physical specification in **references/technical\_specification.md** is fully consistent with the E-R diagram.

## Code artefact

```

1 CREATE SCHEMA
2 DROP DATABASE
3
4 # Create the database
5 CREATE DATABASE my_db
6
7 # Export the directory contents
8 cp -R /c:/path/contents my_db
9
10 # Backup and not to get confused, backup!
11 my_backup.bat
12
13 # Connecting to the database
14 use my_db
15
16 # Create the tables
17 CREATE TABLE my_table (
18 int primary key
19)
20
21 # Temporarily disable foreign keys to safely drop and recreate tables
22 alter session set disable_foreign_keys=TRUE;
23
24 # Drop the tables
25 DROP TABLE my_table;
26
27 # Creating temporary tables to ensure a clean schema...
28 CREATE TEMPORARY TABLE my_table (
29 int primary key,
30 varchar(100),
31 int,
32 varchar(100),
33 varchar(100),
34 varchar(100)
35);
36
37 # Insert data
38 INSERT INTO my_table (int, varchar(100), int, varchar(100), varchar(100), varchar(100))
39 VALUES (1, 'a', 2, 'b', 'c', 'd');
40
41 # Creating tables with primary keys and constraints...
42
43
44 #
45
46 # Make sure this is set to ON after inserting all data
47
48 Commit
49
50
51 #
52
53 #
54
55 #
56
57 #
58
59 #
60
61 #
62
63 #
64
65 #
66
67 #
68
69 #
70
71 #
72
73 #
74
75 #
76
77 #
78
79 #
80
81 #
82
83 #
84
85 #
86
87 #
88
89 #
90
91 #
92
93 #
94
95 #
96
97 #
98
99 #
100
101 #
102
103 #
104
105 #
106
107 #
108
109 #
110
111 #
112
113 #
114
115 #
116
117 #
118
119 #
120
121 #
122
123 #
124
125 #
126
127 #
128
129 #
130
131 #
132
133 #
134
135 #
136
137 #
138
139 #
140
141 #
142
143 #
144
145 #
146
147 #
148
149 #
150
151 #
152
153 #
154
155 #
156
157 #
158
159 #
160
161 #
162
163 #
164
165 #
166
167 #
168
169 #
170
171 #
172
173 #
174
175 #
176
177 #
178
179 #
180
181 #
182
183 #
184
185 #
186
187 #
188
189 #
190
191 #
192
193 #
194
195 #
196
197 #
198
199 #
200
201 #
202
203 #
204
205 #
206
207 #
208
209 #
210
211 #
212
213 #
214
215 #
216
217 #
218
219 #
220
221 #
222
223 #
224
225 #
226
227 #
228
229 #
230
231 #
232
233 #
234
235 #
236
237 #
238
239 #
240
241 #
242
243 #
244
245 #
246
247 #
248
249 #
250
251 #
252
253 #
254
255 #
256
257 #
258
259 #
260
261 #
262
263 #
264
265 #
266
267 #
268
269 #
270
271 #
272
273 #
274
275 #
276
277 #
278
279 #
280
281 #
282
283 #
284
285 #
286
287 #
288
289 #
290
291 #
292
293 #
294
295 #
296
297 #
298
299 #
300
301 #
302
303 #
304
305 #
306
307 #
308
309 #
310
311 #
312
313 #
314
315 #
316
317 #
318
319 #
320
321 #
322
323 #
324
325 #
326
327 #
328
329 #
330
331 #
332
333 #
334
335 #
336
337 #
338
339 #
340
341 #
342
343 #
344
345 #
346
347 #
348
349 #
350
351 #
352
353 #
354
355 #
356
357 #
358
359 #
360
361 #
362
363 #
364
365 #
366
367 #
368
369 #
370
371 #
372
373 #
374
375 #
376
377 #
378
379 #
380
381 #
382
383 #
384
385 #
386
387 #
388
389 #
390
391 #
392
393 #
394
395 #
396
397 #
398
399 #
400
401 #
402
403 #
404
405 #
406
407 #
408
409 #
410
411 #
412
413 #
414
415 #
416
417 #
418
419 #
420
421 #
422
423 #
424
425 #
426
427 #
428
429 #
430
431 #
432
433 #
434
435 #
436
437 #
438
439 #
440
441 #
442
443 #
444
445 #
446
447 #
448
449 #
450
451 #
452
453 #
454
455 #
456
457 #
458
459 #
460
461 #
462
463 #
464
465 #
466
467 #
468
469 #
470
471 #
472
473 #
474
475 #
476
477 #
478
479 #
480
481 #
482
483 #
484
485 #
486
487 #
488
489 #
490
491 #
492
493 #
494
495 #
496
497 #
498
499 #
500
501 #
502
503 #
504
505 #
506
507 #
508
509 #
510
511 #
512
513 #
514
515 #
516
517 #
518
519 #
520
521 #
522
523 #
524
525 #
526
527 #
528
529 #
530
531 #
532
533 #
534
535 #
536
537 #
538
539 #
540
541 #
542
543 #
544
545 #
546
547 #
548
549 #
550
551 #
552
553 #
554
555 #
556
557 #
558
559 #
560
561 #
562
563 #
564
565 #
566
567 #
568
569 #
570
571 #
572
573 #
574
575 #
576
577 #
578
579 #
580
581 #
582
583 #
584
585 #
586
587 #
588
589 #
590
591 #
592
593 #
594
595 #
596
597 #
598
599 #
600
601 #
602
603 #
604
605 #
606
607 #
608
609 #
610
611 #
612
613 #
614
615 #
616
617 #
618
619 #
620
621 #
622
623 #
624
625 #
626
627 #
628
629 #
630
631 #
632
633 #
634
635 #
636
637 #
638
639 #
640
641 #
642
643 #
644
645 #
646
647 #
648
649 #
650
651 #
652
653 #
654
655 #
656
657 #
658
659 #
660
661 #
662
663 #
664
665 #
666
667 #
668
669 #
670
671 #
672
673 #
674
675 #
676
677 #
678
679 #
680
681 #
682
683 #
684
685 #
686
687 #
688
689 #
690
691 #
692
693 #
694
695 #
696
697 #
698
699 #
700
701 #
702
703 #
704
705 #
706
707 #
708
709 #
710
711 #
712
713 #
714
715 #
716
717 #
718
719 #
720
721 #
722
723 #
724
725 #
726
727 #
728
729 #
730
731 #
732
733 #
734
735 #
736
737 #
738
739 #
740
741 #
742
743 #
744
745 #
746
747 #
748
749 #
750
751 #
752
753 #
754
755 #
756
757 #
758
759 #
760
761 #
762
763 #
764
765 #
766
767 #
768
769 #
770
771 #
772
773 #
774
775 #
776
777 #
778
779 #
780
781 #
782
783 #
784
785 #
786
787 #
788
789 #
790
791 #
792
793 #
794
795 #
796
797 #
798
799 #
800
801 #
802
803 #
804
805 #
806
807 #
808
809 #
810
811 #
812
813 #
814
815 #
816
817 #
818
819 #
820
821 #
822
823 #
824
825 #
826
827 #
828
829 #
830
831 #
832
833 #
834
835 #
836
837 #
838
839 #
840
841 #
842
843 #
844
845 #
846
847 #
848
849 #
850
851 #
852
853 #
854
855 #
856
857 #
858
859 #
860
861 #
862
863 #
864
865 #
866
867 #
868
869 #
870
871 #
872
873 #
874
875 #
876
877 #
878
879 #
880
881 #
882
883 #
884
885 #
886
887 #
888
889
```

★ **Core Idea:** This multi-layered structure ensures student control and explicit refinement of both the skill development and the technical delivery.

# Ongoing work

- **Pilot experimentation**
  - Database modelling and programming: 78 students, 6-23 October
  - Machine learning (supervised): 57 students, late Autumn Term
- **Implementing unsupervised machine learning**
  - Customer segmentation / Market basket analysis (Antigravity 2.0/SDK)
- **Improving AI usage declaration and analysis of students' reports**
- **CONTACT: Dr Marcos Barreto ([m.e.barreto@lse.ac.uk](mailto:m.e.barreto@lse.ac.uk))**





# End of presentation



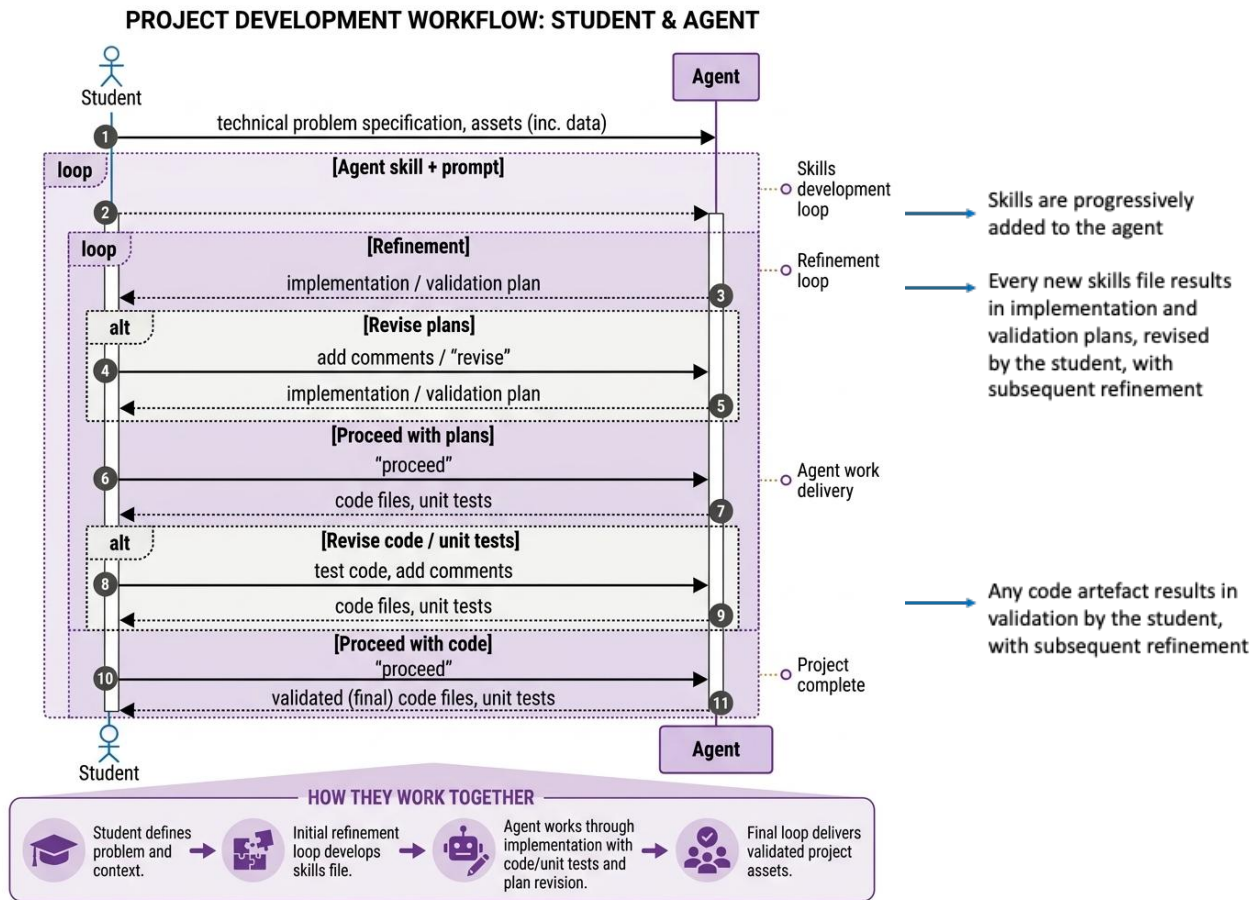


# Extras - demo



# Example: database modelling

Student-agent  
collaboration



# Example: database modelling

Skills file



Google for Education  
Higher Education Faculty AI Fellowship

|             |                                                                                                                                                                                                                          |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| name        | companydb_modelling                                                                                                                                                                                                      |
| description | Reviews an E-R model against a technical specification and generates a SQLite schema implementation plan, scripts, and verification tests. Use this skill when the user asks for help creating or generating a database. |

## Goal

Validate the database E-R model stored as `assets/er_diagram.png` against `references/technical_specification.md` and produce a schema implementation plan.

## Instructions

1. **Read Specifications:** Read `references/technical_specification.md` at the project root. This is the source of truth for entities, attributes, relationships, cardinalities, and business rules.
2. **Review E-R Diagram:** Review `assets/er_diagram.png` (or `.mmd` source) at the project root. If no diagram is found, STOP, and ask the user to produce a valid diagram matching the technical specification document.
3. **Compare:** Compare the diagram against the technical specification. Identify any missing or extra entities, attributes, relationships, cardinalities, or specialisations. Do NOT add anything not explicitly defined.
4. **Mismatches:** If there are mismatches, STOP. Report them and wait for user input before proceeding. Do not silently modify the model.
5. **Implementation Plan:** Generate an implementation plan stating how the E-R model/diagram and the technical specification will be mapped into a database schema. Present the implementation plan to the user and wait for their approval before moving forward. The implementation plan may contemplate the following mappings:
  - Entities to tables
  - 1:N relationships to foreign keys
  - N:M relationships to junction tables
  - Subtypes (specialisation) to tables (explain the strategy: single table vs. one per subtype)
  - Any other mapping operations, following the E-R model/diagram and technical specification document.
6. **SQLite Schema Scripts:** Once the implementation plan is approved, generate Python scripts (not raw SQL files) to create the schema (primary keys, foreign keys, CHECK constraints). Target database is SQLite. Check if files exist at the target path first; do not overwrite without user confirmation. Place all generated Python scripts in the `code` folder, and store the SQLite database file in a `db` folder.
7. **Verification Script:** Generate a validation plan to data types, primary and foreign keys. Present the approval. Once approved, create a Python script for the following statements:
  - `SELECT name FROM sqlite_master WHERE`
  - `PRAGMA foreign_key_list(table)`
  - `PRAGMA foreign_keys` (to verify enforcement of the schema and re-run.

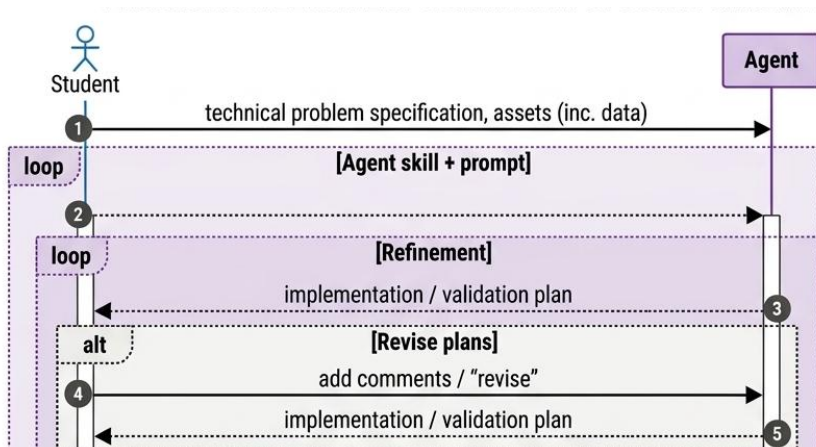
## Constraints

- Enforce business rules from `references/technical_specification.md` only.
- For single-column/single-row constraints, use SQLite `CHECK` constraints.
- For multi-row or multi-table constraints, delegate to application logic or database triggers; call these out in the plan.
- If the diagram and technical specification disagree, clarify with the user.

User Request: `companydb\_modelling` skill to create the database data based on the referenced workspace artefacts. Do not go beyond the instructions in the skills file; more specifically, do not create any code to load data into the database or implement any query. This is the first step in the database modelling exercise where we will only create an empty database matching the provided description and schema. Before generating the database schema and any code files, generate implementation plans and validation plans and wait for my review and approval.

[illegible]

Google for Education  
Higher Education Faculty AI Fellowship



This plan describes the mapping of the Company DB E-R model (located in `assets/er_diagram.png`) and reference specifications (in `references/technical_specification.md`) to a SQLite database schema. It also outlines the validation plan to verify the correctness of the generated empty database.

A thorough comparison has been performed between the E-R diagram ( `assets/er_diagram.png` ), the database description ( `database_description.md` ), and the technical specifications ( `references/technical_specification.md` ):

- **EMPLOYEE** : Matches perfectly. Composite attribute (**Name**) composed of **Fname**, **Minit**, **Lname** is correctly flattened into separate columns in the specifications. **Ssn** is correctly marked as the primary key.
- **DEPARTMENT** : Matches perfectly. **Dnumber** is the primary key. **Dname** is unique. **Locations** is a multi-valued attribute (represented by double oval), which maps to a separate table **dept\_locations** in the specifications. **Number\_Employees** is a derived attribute (dashed oval) and is omitted from physical persistence as per standard practices.
- **PROJECT** : Matches perfectly. **Pnumber** is the primary key. **Pname** is unique. **Plocation** is simple.
- **DEPENDENT** : Matches perfectly. Represented as a weak entity (double rectangle) with **Dependent\_name** and **Essn** as its key. **Dependent\_name** is the identifying relationship-key. The composite primary key is correctly identified.

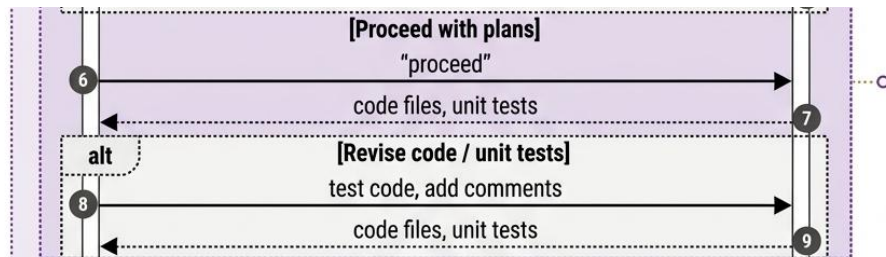
proper cons.

Mgr\_ssn is an attribute of DEPARTMENT acting as a foreign key

- Works\_on** (EMPLOYEE to PROJECT): N:M relationship. Maps to a junction table `works_on` with composite primary key (`Essn`, `Pno1`).
- Dependents\_of** (EMPLOYEE to DEPENDENT): Weak entity relationship. Maps to foreign key `Essn` in `dependent` table, which is also part of the primary key.

**Mismatches/Issues identified:** None. The physical specification in `references/technical_specification.md` is fully consistent with the E-R diagram.

# Code test / refinement loop

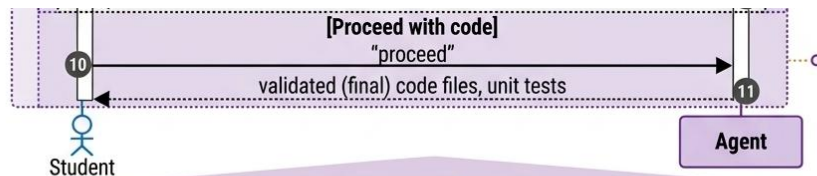


CompanyDB\_v1 > code > create\_schema.py (all agent edits)

```
1 import os
2 import sqlite3
3
4 def create_database_schema(db_path="db/company.db"):
5 # Ensure the directory exists
6 db_dir = os.path.dirname(db_path)
7 if db_dir and not os.path.exists(db_dir):
8 os.makedirs(db_dir)
9
10 print(f"Connecting to database at: {db_path}")
11 conn = sqlite3.connect(db_path)
12 cursor = conn.cursor()
13
14 # Temporarily disable foreign keys to safely drop and recreate tables
15 cursor.execute("PRAGMA foreign_keys = OFF;")
16
17 pr
18 ta
19 fo
20
21 Make sure this is set to ON after inserting all data
22 Cancel Add Comment
23
24 print("Creating tables with correct schemas and constraints...")
25
26 # 1. Create Department table (referencing employee)
27 cursor.execute("""
28 CREATE TABLE department (
29 Dname VARCHAR(30) NOT NULL UNIQUE,
30 Dnumber INT PRIMARY KEY,
31 Mgr_ssn CHAR(11) NOT NULL,
32 Mgr_start_date DATE,
33 FOREIGN KEY (Mgr_ssn) REFERENCES employee(Ssn) ON DELETE RESTRICT ON UPDATE CASCADE
34);
35 """)
```



# Final artefacts



I have completed all the tasks according to the updated database modelling instructions and your feedback.

Here is a summary of the accomplishments:

## 1. Documentation Written:

- [docs/implementation\\_plan.md](#): Details the design mappings of entities, attributes, relationships, and recursive links to physical tables.
- [docs/validation\\_plan.md](#): Describes the structural testing and how to run verification tests programmatically in Google Colab.
- [docs/task\\_list.md](#): Tracks completed and remaining work.

## 2. Schema Definition Script:

- Written to [code/create\\_schema.py](#).
- This script constructs [db/company.db](#) and maps the correct tables with your requested adjustments:
  - [Mgr\\_ssn](#) is enforced as [NOT NULL](#).
  - [works\\_on](#) uses [Ssn](#) instead of [Essn](#).
  - [Relationship](#) in the [dependent](#) table is typed as [VARCHAR\(30\)](#).

## 3. Validation Suite:

- Written to [code/verify\\_schema.py](#).
- Uses only Python's standard [unittest](#) and [sqlite3](#) libraries and is fully prepared to be imported/called directly inside Google Colab cells without executing command-line scripts.
- Checks table structures, data types, nullabilities, and foreign key relations.

## 4. Execution and Validation:

- Ran the schema generation script successfully.
- Ran the verification test suite, which executed and passed all 9 test cases successfully.

You can view the detailed results in the [Walkthrough Report](#). The empty database schema is now fully initialized and validated, ready for the next steps!



# Human agency + guardrails

## Implementation & Validation Plan: Collaborative SQL Querying

This plan outlines the approach to collaboratively write, validate, and programmatically cross-check SQL queries for the CompanyDB analytical questions, in strict adherence to the `companydb_querying` skill.

### 1. Collaborative Query Writing Strategy

As mandated by the `companydb_querying` skill, we will treat each query as a collaborative process. For every analytical objective:

1. **Identify the Objective:** State the question clearly.
2. **Interactive Design:** We will ask you to guide us on:

- Which tables need to be queried ( `FROM` clause).
- How to structure any necessary `JOIN` operation.
- Any conditions or filtering rules to apply ( `WHERE` clause).
- Which columns to project ( `SELECT` clause).

3. **Query Construction:** We will synthesize the read-only SQL query based on your inputs and our verified SQLite schema.

4. **Interactive Log:** We will capture this collaborative design implementation plan stored in the `docs/` folder.

### 2. Programmatic Verification Strategy

To guarantee the correctness of each generated SQL, to prevent hallucination, we will implement a cross-checking script `code/verify_queries.py`. For each query, this script will:

1. **Execute the SQL Query:** Run the proposed SQL query against the database and retrieve the result set.
2. **Independent Python Logic Calculation:** Implement the same logic in Python (using lists, dicts, or other standard data structures) to perform the equivalent filtering/aggregating/joining and return the expected result.
3. **Assert Equality:** Verify that the SQL result set matches the Python result set exactly (structure and values).
4. **Output Report:** Save a verification report in the `docs/query_verification_report.md`.

### 3. Immediate Action: Waiting for Your Analytical Questions

Since the current workspace files (such as `database_description.md` and `README.md`) contain instructions but do **not** enumerate the specific analytical questions to be answered, we need you to provide them.

To help you formulate the queries and guide the process, here is a quick reference of our validated schema:

- **employee** : `Fname`, `Minit`, `Lname`, `Ssn` (PK), `Bdate`, `Address`, `Gender`, `Salary`, `Super_ssn` (FK to employee), `Dno` (FK to department).
- **department** : `Dname` (Unique), `Dnumber` (PK), `Mgr_ssn` (FK to employee, NOT NULL), `Mgr_start_date`.
- **dept\_locations** : `Dnumber` (FK to department), `Dlocation` (PK, composite).
- **project** : `Pname` (Unique), `Pnumber` (PK), `Plocation`, `Dnum` (FK to department, NOT NULL).
- **works\_on** : `Ssn` (FK to employee), `Pno` (FK to project), `Hours` (PK, composite).
- **dependent** : `Essn` (FK to employee), `Dependent_name` (PK, composite), `Gender`, `Bdate`, `Relationship`.

### Ready for Your Instructions

Please provide the analytical questions you would like us to solve. For each question, feel free to give us your thoughts on the tables, joins, filters, and projections, or simply list the questions and we will work through them one by one!

# Human agency + guardrails (cont'd)

I see the updated section in [database\\_description.md](#) now! Excellent.

Here are the three analytical questions we need to answer. To make this a truly collaborative effort, let's break down each question.

For each question below, I have provided a quick hint or visual of the tables involved, and a hint:

1. Which tables we should query ( **FROM** clause)?
2. Which conditions to apply ( **WHERE** or **ON** clause)?
3. How we should structure the **JOIN** operations (e.g. **INNER JOIN**, **LEFT JOIN**)?
4. Which columns we should project in the output ( **SELECT** clause)?

Question 1: Group projects per department and list all employees working in

- **Objective:** Show the departmental organization of projects, along with the employee count.
- **Key entities involved:** Departments, Projects, Employees, and the assignments between them.
- **Tables to think about:** `department`, `project`, `employee`, and `works_on`.

## NOTE

How would you structure the **FROM** and **JOIN** clauses to connect Departments with Projects with the Employees assigned to them? Which fields should we project to make the list clear?

For question 1, join department and project tables to retrieve the department name, all project names, and the employee count per project. Let's do question 1 first and add

Thought for 11s ^

This is excellent guidance! Based on your

1. Join `department` and `project` by the `department_id` foreign key.
2. Retrieve employees working on each project via the `works_on` table.
3. Retrieve the department name, project name, and employee count per project.

Since we want to "group projects per department" and list all employees working on them, we have two main ways to structure the results:

## Option A: Tabular Layout (Grouping by Department)

This structure groups the results by department, listing all projects and their respective employee count. It's the standard way to present this data.

```
sql
SELECT
 d.Dname AS Department_Name,
 p.Pname AS Project_Name,
 COUNT(w.Ssn) AS Employee_Count
FROM department d
JOIN project p ON d.Dnumber = p.Dnumber
LEFT JOIN works_on w ON p.Pnumber = w.Pnumber
GROUP BY d.Dnumber, p.Pnumber
ORDER BY d.Dname, p.Pname;
```

I have successfully implemented, tested, and programmatically verified **Question 1!**

## 1. Results for Question 1 (Projects per Department with Employee Count)

The SQL query was run against the database and cross-checked against independent Python aggregation logic. Both matches were 100% identical. Here is the verified database output:

| Department Name | Project Name     | Employee Count |
|-----------------|------------------|----------------|
| Administration  | Computerization  | 3              |
| Administration  | Newbenefits      | 3              |
| Hardware        | InkjetPrinters   | 8              |
| Hardware        | LaserPrinters    | 3              |
| Headquarters    | Reorganization   | 3              |
| Research        | ProductX         | 2              |
| Research        | ProductY         | 3              |
| Research        | ProductZ         | 2              |
| Software        | DatabaseSystems  | 8              |
| Software        | Middleware       | 4              |
| Software        | OperatingSystems | 9              |

The SQL query and design notes have been captured in [docs/query\\_implementation\\_plan.md](#), and the test suite has been saved in [code/verify\\_queries.py](#).



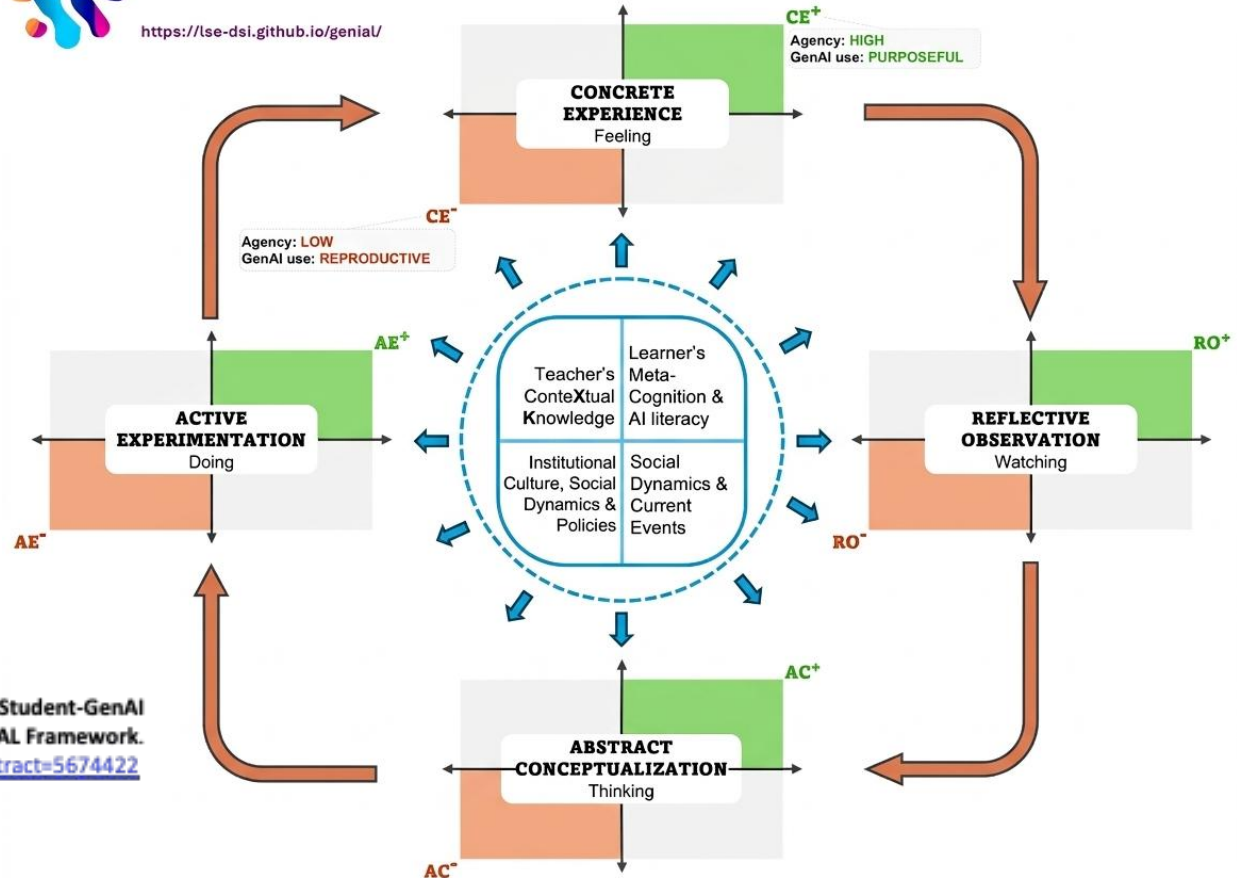
GENIAL: **GEN**erative **AI** Tools as  
a Catalyst for **Learning**

<https://lse-dsi.github.io/genial/>

# Validation

- **GENIAL framework**  
applied to student-agent  
interaction
- Analysis of skills files  
against reference  
implementation

Marcos Barreto et al. (GENIAL team). **Mapping Student-GenAI  
Interactions onto Experiential Learning: The GENIAL Framework.**  
October 2025. Available at SSRN: <https://ssrn.com/abstract=5674422>



Google for Education  
Higher Education Faculty AI Fellowship



# End of extras

