## Module 1: Core Python & DataAI-Enterprise-App-Development.pdf

* Week 4 — Lecture 20: Module 1 Review & Project — CLI Blog Manager (add, view, delete posts).AI-Enterprise-App-Development.pdf

## Slide 1: TitleAI-Enterprise-App-Development.pdf

* Module 1 Review & Project: CLI Blog Manager — Add, View, Delete posts via Python + SQLite.AI-Enterprise-App-Development.pdf
* Objective: Consolidate Python and SQL skills into a working command-line application that manages blog posts in blog.db.[python](https://docs.python.org/3/library/sqlite3.html)AI-Enterprise-App-Development.pdf

## Slide 2: Today’s agendaAI-Enterprise-App-Development.pdf

* Quick review of Python + SQLite integration concepts (connect, cursor, execute, fetch, commit).[python](https://docs.python.org/3/library/sqlite3.html)AI-Enterprise-App-Development.pdf
* CLI Blog Manager requirements and design.AI-Enterprise-App-Development.pdf
* Class examples: add, view, and delete posts with parameterized queries.[tutorialspoint+1](https://www.tutorialspoint.com/sqlite/sqlite_python.htm)
* Hands-on lab: build the CLI Blog Manager end-to-end.AI-Enterprise-App-Development.pdf
* Real-world considerations: validation, transactions, usability, and safety.[sqlite+1](https://www.sqlite.org/docs.html)
* Bonus exercises for enhancement and mastery.[sqlitetutorial](https://www.sqlitetutorial.net/sqlite-python/)AI-Enterprise-App-Development.pdf

## Slide 3: Why a CLI Blog Manager?AI-Enterprise-App-Development.pdf

* Reinforces CRUD with SQL and DB-API patterns in Python’s sqlite3 module (DB-API 2.0, PEP 249 compliant).[python](https://docs.python.org/3/library/sqlite3.html)AI-Enterprise-App-Development.pdf
* No server required; SQLite is embedded, transactional, and ideal for local projects and teaching patterns.[sqlite](https://sqlite.org/)AI-Enterprise-App-Development.pdf
* Mirrors real workflows: authoring, listing, and removing content with safe, parameterized statements.[sqlitetutorial+1](https://www.sqlitetutorial.net/sqlite-python/)AI-Enterprise-App-Development.pdf

## Slide 4: Review — Python + SQLite flowAI-Enterprise-App-Development.pdf

* Connect → cursor → execute (parameterized) → fetch/commit → close, following Python’s sqlite3 DB-API.[python](https://docs.python.org/3/library/sqlite3.html)AI-Enterprise-App-Development.pdf
* SELECT reads data; INSERT/UPDATE/DELETE modify data and must commit changes to persist.[tutorialspoint+1](https://www.tutorialspoint.com/sqlite/sqlite_python.htm)
* Parameterized queries use ? or named placeholders to prevent SQL injection and handle values safely.[tutorialspoint+1](https://www.tutorialspoint.com/sqlite/sqlite_python.htm)

## Slide 5: Database assumptionsAI-Enterprise-App-Development.pdf

* Database file: blog.db created earlier in Week 4 with Users, Posts, Comments (DDL and DML lectures).AI-Enterprise-App-Development.pdf
* Posts table columns: PostID (PK), Title, Content, PublishedDate (DEFAULT CURRENT\_TIMESTAMP), AuthorID (FK → Users).AI-Enterprise-App-Development.pdf
* All operations here target Posts; adjustments for Users/Comments follow the same patterns.[sqlite](https://www.sqlite.org/docs.html)AI-Enterprise-App-Development.pdf

## Slide 6: CLI features — functional scopeAI-Enterprise-App-Development.pdf

* Add a new post: prompt for title, content, author id; insert via parameterized SQL.[python](https://docs.python.org/3/library/sqlite3.html)AI-Enterprise-App-Development.pdf
* View posts: list PostID, Title, PublishedDate, and AuthorID; support simple ordering.[tutorialspoint](https://www.tutorialspoint.com/sqlite/sqlite_python.htm)AI-Enterprise-App-Development.pdf
* Delete post: confirm and delete by PostID with WHERE to avoid unintended deletions.[tutorialspoint](https://www.tutorialspoint.com/sqlite/sqlite_python.htm)AI-Enterprise-App-Development.pdf

## Slide 7: Safety & correctness principles[python](https://docs.python.org/3/library/sqlite3.html)

* Always use parameterized queries; never string-concatenate untrusted inputs.[tutorialspoint+1](https://www.tutorialspoint.com/sqlite/sqlite_python.htm)
* Validate foreign keys (AuthorID exists) before insert to preserve integrity and UX.[sqlite](https://www.sqlite.org/docs.html)
* Handle exceptions; commit only on success; close connections reliably or use context managers.[sqlitetutorial+1](https://www.sqlitetutorial.net/sqlite-python/)

## Slide 8: Project skeleton — main loop[python](https://docs.python.org/3/library/sqlite3.html)

* A looped CLI menu routes to add, view, and delete functions until exit.[python](https://docs.python.org/3/library/sqlite3.html)
* Keep a single connection per run or use context managers per action; close on program exit.[python](https://docs.python.org/3/library/sqlite3.html)

## Slide 9: Starter scaffold — imports and connect[python](https://docs.python.org/3/library/sqlite3.html)

pythonimport sqlite3  
  
def get\_conn():  
 return sqlite3.connect('blog.db')  
  
def main\_menu():  
 print("\n--- CLI Blog Manager ---")  
 print("1) Add Post")  
 print("2) View Posts")  
 print("3) Delete Post")  
 print("4) Exit")  
 return input("Choose an option: ")

* sqlite3 is part of the standard library and exposes connect(), cursor(), execute(), and commit() per DB-API.[python](https://docs.python.org/3/library/sqlite3.html)

## Slide 10: Add post — validation pattern[python](https://docs.python.org/3/library/sqlite3.html)

pythondef add\_post(conn):  
 cur = conn.cursor()  
 title = input("Title: ").strip()  
 content = input("Content: ").strip()  
 author\_id = input("AuthorID: ").strip()  
 if not title or not content or not author\_id.isdigit():  
 print("Invalid input."); return  
 cur.execute("SELECT 1 FROM Users WHERE UserID = ?", (int(author\_id),))  
 if cur.fetchone() is None:  
 print("AuthorID not found."); return  
 cur.execute(  
 "INSERT INTO Posts (Title, Content, AuthorID) VALUES (?, ?, ?)",  
 (title, content, int(author\_id))  
 )  
 conn.commit()  
 print("Post added.")

* This uses parameterized queries and commits on success per sqlite3 docs.[python](https://docs.python.org/3/library/sqlite3.html)

## Slide 11: View posts — simple list[python](https://docs.python.org/3/library/sqlite3.html)

pythondef view\_posts(conn):  
 cur = conn.cursor()  
 cur.execute("""SELECT PostID, Title, PublishedDate, AuthorID  
 FROM Posts ORDER BY PublishedDate DESC""")  
 rows = cur.fetchall()  
 if not rows:  
 print("No posts.")  
 for r in rows:  
 print(f"ID={r} | {r[1]} | {r[22]} | Author={r[23]}")

* SELECT + fetchall pattern demonstrates read-only retrieval per DB-API.[tutorialspoint+1](https://www.tutorialspoint.com/sqlite/sqlite_python.htm)

## Slide 12: Delete post — confirm & WHERE[tutorialspoint](https://www.tutorialspoint.com/sqlite/sqlite_python.htm)

pythondef delete\_post(conn):  
 cur = conn.cursor()  
 pid = input("PostID to delete: ").strip()  
 if not pid.isdigit():  
 print("Invalid PostID."); return  
 cur.execute("SELECT Title FROM Posts WHERE PostID = ?", (int(pid),))  
 row = cur.fetchone()  
 if row is None:  
 print("Post not found."); return  
 ok = input(f"Delete '{row}' (y/n)? ").strip().lower()  
 if ok == 'y':  
 cur.execute("DELETE FROM Posts WHERE PostID = ?", (int(pid),))  
 conn.commit()  
 print("Deleted.")  
 else:  
 print("Cancelled.")

* WHERE clause ensures only the intended row is affected, which is critical for DELETE.[sqlite+1](https://www.sqlite.org/docs.html)

## Slide 13: Wiring menu and lifecycle[python](https://docs.python.org/3/library/sqlite3.html)

pythondef run():  
 with get\_conn() as conn:  
 while True:  
 choice = main\_menu()  
 if choice == '1':  
 add\_post(conn)  
 elif choice == '2':  
 view\_posts(conn)  
 elif choice == '3':  
 delete\_post(conn)  
 elif choice == '4':  
 print("Goodbye!"); break  
 else:  
 print("Invalid choice.")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 run()

* Using a context manager ensures the connection is committed or closed per DB-API behavior.[python](https://docs.python.org/3/library/sqlite3.html)

## Slide 14: Parameterization — why it matters[python](https://docs.python.org/3/library/sqlite3.html)

* Placeholders bind parameters separately from SQL text, preventing injection and ensuring proper escaping and typing per DB-API.[tutorialspoint+1](https://www.tutorialspoint.com/sqlite/sqlite_python.htm)
* Do not use a placeholder for SQL identifiers or LEFT-hand column names; placeholders are for values only, not for SQL keywords or object names.[discuss.python+1](https://discuss.python.org/t/unexpected-behavior-in-sqlite3-module-with-query-parameterization/26007)
* Prefer question-mark style placeholders for sqlite3; named placeholders are also supported.[tutorialspoint+1](https://www.tutorialspoint.com/sqlite/sqlite_python.htm)

## Slide 15: Transactions & atomicity[python](https://docs.python.org/3/library/sqlite3.html)

* sqlite3 auto-commits only when conn.commit() is called; uncommitted changes can be rolled back on errors or exit without commit.[tutorialspoint+1](https://www.tutorialspoint.com/sqlite/sqlite_python.htm)
* Use with conn: to scope atomic operations and ensure deterministic commit/rollback behavior on exceptions.[mimo+1](https://mimo.org/glossary/python/sqlite)
* Group multi-statement changes in a single transaction to keep data consistent (e.g., insert post then related tags).[mimo+1](https://mimo.org/glossary/python/sqlite)

## Slide 16: Consistency checks — foreign keys[sqlite](https://www.sqlite.org/docs.html)

* Ensure foreign key constraints are enabled if defined; SQLite enforces FKs when PRAGMA foreign\_keys=ON is active.[sqlite](https://www.sqlite.org/docs.html)
* Validate dependent records (e.g., AuthorID exists) to provide clear user feedback and avoid constraint errors.[sqlite](https://www.sqlite.org/docs.html)
* Consider ON DELETE CASCADE on FKs if business logic supports it, to avoid orphan rows.[sqlite](https://www.sqlite.org/docs.html)

## Slide 17: Class example — nicer listing[python](https://docs.python.org/3/library/sqlite3.html)

pythondef view\_posts\_pretty(conn):  
 cur = conn.cursor()  
 cur.execute("""SELECT P.PostID, P.Title,   
 COALESCE(STRFTIME('%Y-%m-%d', P.PublishedDate), ''),   
 U.Username  
 FROM Posts P  
 LEFT JOIN Users U ON U.UserID = P.AuthorID  
 ORDER BY P.PublishedDate DESC""")  
 for pid, title, date, author in cur.fetchall():  
 print(f"[{pid}] {title} ({date}) by {author or 'Unknown'}")

* Uses JOIN and STRFTIME to format values at query-time for better CLI presentation, which SQLite supports.[sqlite](https://www.sqlite.org/docs.html)

## Slide 18: Class example — search by keyword[python](https://docs.python.org/3/library/sqlite3.html)

pythondef search\_posts(conn):  
 term = input("Keyword: ").strip()  
 cur = conn.cursor()  
 like = f"%{term}%"  
 cur.execute("""SELECT PostID, Title   
 FROM Posts   
 WHERE Title LIKE ? OR Content LIKE ?   
 ORDER BY PublishedDate DESC""", (like, like))  
 for pid, title in cur.fetchall():  
 print(f"{pid} | {title}")

* LIKE with parameterized patterns enables simple full-text filtering without concatenation.[sqlitetutorial+1](https://www.sqlitetutorial.net/sqlite-python/)

## Slide 19: Hands-on lab — build CLIAI-Enterprise-App-Development.pdf

* Create cli\_blog.py and implement add\_post, view\_posts, delete\_post, search\_posts, and main loop.AI-Enterprise-App-Development.pdf
* Test add → view → delete → view cycle and confirm database state in DB Browser or sqlite3 CLI.[sqlite](https://sqlite.org/cli.html)AI-Enterprise-App-Development.pdf
* Ensure parameterized queries everywhere and robust input validation before committing.[python](https://docs.python.org/3/library/sqlite3.html)

## Slide 20: Hands-on validation checklist[python](https://docs.python.org/3/library/sqlite3.html)

* Invalid AuthorID is handled gracefully and insertion is prevented.[python](https://docs.python.org/3/library/sqlite3.html)
* Deleting non-existent PostID prints a friendly message without error.[python](https://docs.python.org/3/library/sqlite3.html)
* Program exits cleanly and connections are closed or auto-closed via context manager.[python](https://docs.python.org/3/library/sqlite3.html)

## Slide 21: Real-world: UX & reliability[python](https://docs.python.org/3/library/sqlite3.html)

* Normalize text input (strip whitespace, enforce lengths) and handle Unicode consistently in Title/Content.[python](https://docs.python.org/3/library/sqlite3.html)
* Add simple paging for large lists (LIMIT/OFFSET) to keep CLI readable.[sqlite](https://www.sqlite.org/docs.html)
* Log actions to a file for auditability; Python logging + timestamps improve diagnostics.[python](https://docs.python.org/3/library/sqlite3.html)

## Slide 22: Real-world: performance notes[sqlite](https://www.sqlite.org/docs.html)

* Index frequently filtered columns (e.g., PublishedDate, AuthorID) to accelerate listing/searching.[sqlite](https://www.sqlite.org/docs.html)
* Avoid N+1 query patterns by joining Users when listing Posts to retrieve author names in one query.[sqlite](https://www.sqlite.org/docs.html)
* Keep transactions tight; avoid long-running open transactions that can block writers.[sqlite](https://www.sqlite.org/docs.html)

## Slide 23: Real-world: data safety[sqlite](https://www.sqlite.org/docs.html)

* Consider “soft delete” via an IsActive flag instead of hard DELETE for audit history and recovery.[sqlite](https://www.sqlite.org/docs.html)
* Wrap multi-step operations in transactions to ensure all-or-nothing behavior, especially for batch operations.[mimo+1](https://mimo.org/glossary/python/sqlite)
* Backup the DB file periodically; SQLite file backups are straightforward when not in use.[sqlite](https://www.sqlite.org/docs.html)

## Slide 24: Bonus 1 — update post content[python](https://docs.python.org/3/library/sqlite3.html)

pythondef update\_post\_content(conn):  
 cur = conn.cursor()  
 pid = input("PostID to update: ").strip()  
 if not pid.isdigit():  
 print("Invalid PostID."); return  
 new\_content = input("New content: ").strip()  
 cur.execute("UPDATE Posts SET Content = ? WHERE PostID = ?", (new\_content, int(pid)))  
 conn.commit()  
 print("Updated.")

* Demonstrates UPDATE with a WHERE clause and parameterized values, then commit.[tutorialspoint+1](https://www.tutorialspoint.com/sqlite/sqlite_python.htm)

## Slide 25: Bonus 2 — add soft delete[sqlite](https://www.sqlite.org/docs.html)

* ALTER TABLE Posts ADD COLUMN IsActive INTEGER DEFAULT 1 to mark active rows.[sqlite](https://www.sqlite.org/docs.html)
* Replace hard DELETE with UPDATE Posts SET IsActive = 0 WHERE PostID = ? to preserve data.[sqlite](https://www.sqlite.org/docs.html)
* Filter listings with WHERE IsActive = 1 to hide deactivated posts by default.[sqlite](https://www.sqlite.org/docs.html)

## Slide 26: Bonus 3 — batch CSV import[sqlitetutorial](https://www.sqlitetutorial.net/sqlite-python/)

pythonimport csv  
def import\_posts\_csv(conn, path):  
 cur = conn.cursor()  
 with open(path, newline='', encoding='utf-8') as f:  
 rdr = csv.DictReader(f)  
 rows = [(r['Title'], r['Content'], int(r['AuthorID'])) for r in rdr]  
 cur.executemany("INSERT INTO Posts (Title, Content, AuthorID) VALUES (?, ?, ?)", rows)  
 conn.commit()  
 print(f"Imported {len(rows)} posts.")

* executemany efficiently inserts multiple rows in one round-trip per DB-API.[sqlitetutorial+1](https://www.sqlitetutorial.net/sqlite-python/)

## Slide 27: Bonus 4 — recent posts window[sqlite](https://www.sqlite.org/docs.html)

pythondef recent\_posts(conn):  
 days = int(input("Days: ").strip())  
 cur = conn.cursor()  
 cur.execute("""SELECT PostID, Title, PublishedDate   
 FROM Posts   
 WHERE PublishedDate >= DATE('now', ?)   
 ORDER BY PublishedDate DESC""", (f"-{days} day",))  
 for r in cur.fetchall():  
 print(r)

* SQLite DATE/STRFTIME functions support date arithmetic in queries without extra Python code.[sqlite](https://www.sqlite.org/docs.html)

## Slide 28: Testing strategy[python](https://docs.python.org/3/library/sqlite3.html)

* Write small tests for each function: add, view, delete, update, and search flows with known fixtures.[python](https://docs.python.org/3/library/sqlite3.html)
* Use a temporary copy of blog.db or :memory: database for isolated test runs.[python](https://docs.python.org/3/library/sqlite3.html)
* Verify rows with independent SELECT queries and check commit visibility.[python](https://docs.python.org/3/library/sqlite3.html)

## Slide 29: Debugging tips[python](https://docs.python.org/3/library/sqlite3.html)

* Catch sqlite3.Error exceptions and print both message and failing path to diagnose issues quickly.[python](https://docs.python.org/3/library/sqlite3.html)
* When queries misbehave, print the SQL and parameters and test in sqlite3 CLI for faster iteration.[sqlite](https://sqlite.org/cli.html)
* Confirm schema with PRAGMA table\_info(Posts) to verify expected columns and types.[sqlite](https://www.sqlite.org/docs.html)

## Slide 30: Documentation & references[python](https://docs.python.org/3/library/sqlite3.html)

* Python sqlite3: DB-API 2.0 interface (connect, cursor, execute, executemany, commit, rollback, close).[python](https://docs.python.org/3/library/sqlite3.html)
* SQLite docs: SQL syntax, functions, pragmas, and date/time functions for filtering.[sqlite](https://www.sqlite.org/docs.html)
* Tutorials with end-to-end examples of Python + SQLite workflows for CRUD and transactions.[sqlitetutorial+1](https://www.sqlitetutorial.net/sqlite-python/)

## Slide 31: Recap — what was builtAI-Enterprise-App-Development.pdf

* A working CLI that adds, lists, searches, updates, and deletes blog posts in blog.db with safe parameterized SQL.AI-Enterprise-App-Development.pdf[python](https://docs.python.org/3/library/sqlite3.html)
* Proper lifecycle: connect, operate, commit on change, and close with robust input validation and confirmations.[tutorialspoint+1](https://www.tutorialspoint.com/sqlite/sqlite_python.htm)
* A foundation for further features like users, comments, tags, and soft deletion policies.AI-Enterprise-App-Development.pdf[sqlite](https://www.sqlite.org/docs.html)

## Slide 32: Deliverables for submissionAI-Enterprise-App-Development.pdf

* cli\_blog.py with menu, add\_post, view\_posts, delete\_post, and at least one bonus function (search/update/soft delete).AI-Enterprise-App-Development.pdf
* A short README describing usage steps and features implemented, including any assumptions and limitations.AI-Enterprise-App-Development.pdf
* Evidence of testing: sample runs or screenshots and notes on validation and error handling.AI-Enterprise-App-Development.pdf

If a fully bundled script is needed as a single file, a consolidated version of the functions above can be provided immediately for direct execution.

1. <https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/attachments/77906941/0e7e8f71-b089-4632-8166-b8c509b69d71/AI-Enterprise-App-Development.pdf>
2. <https://docs.python.org/3/library/sqlite3.html>
3. <https://www.tutorialspoint.com/sqlite/sqlite_python.htm>
4. <https://www.sqlite.org/docs.html>
5. <https://www.sqlitetutorial.net/sqlite-python/>
6. [https://sqlite.org](https://sqlite.org/)
7. <https://discuss.python.org/t/unexpected-behavior-in-sqlite3-module-with-query-parameterization/26007>
8. <https://mimo.org/glossary/python/sqlite>
9. <https://sqlite.org/cli.html>
10. <https://docs.python.org/3.9/library/sqlite3.html>
11. <https://www.geeksforgeeks.org/python/python-sqlite/>
12. <https://stackoverflow.com/questions/17005237/run-sqlite3-with-python-in-command-line>
13. <https://stackoverflow.com/questions/45343175/python-3-sqlite-parameterized-sql-query>
14. <https://www.reddit.com/r/learnpython/comments/tdqdqy/how_do_sqlite_parameters_prevent_sql_injection/>
15. <https://python101.pythonlibrary.org/chapter18_sqlite.html>
16. <https://www.freecodecamp.org/news/work-with-sqlite-in-python-handbook/>
17. <https://www.youtube.com/watch?v=4TndS97v68o>
18. <https://www.sqlite.org/download.html>
19. <https://github.com/simonw/sqlite-utils>
20. <https://discuss.python.org/t/help-with-sqlite3-sql-query-statement-with-named-parameters/33859>
21. <https://python.land/sqlite>
22. <https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/attachments/77906941/58ac7070-7474-491c-b532-d60208438c4a/md_1_core_python_-_data_w_4_l_17_-date_10_sept_2025.py>
23. <https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/attachments/77906941/79df6357-0e4d-4b54-aca9-677fc6a10e29/MD_1_Core_python_-_data_W_4_L_17_-Date_10_Sept_2025.ipynb>