Module 1: Core Python & Data

AI-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: Title AI-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

AI-Enterprise-App-Development.pdf

Slide 2: Today’s agenda

AI-Enterprise-App-Development.pdf

- Quick review of Python + SQLite integration concepts (connect, cursor, execute, fetch, commit).python

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

- 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
- Bonus exercises for enhancement and mastery.sqlitetutorial

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

AI-Enterprise-App-Development.pdf

- No server required; SQLite is embedded, transactional, and ideal for local projects and teaching patterns.sqlite

AI-Enterprise-App-Development.pdf

- Mirrors real workflows: authoring, listing, and removing content with safe, parameterized statements.sqlitetutorial+1

AI-Enterprise-App-Development.pdf

Slide 4: Review — Python + SQLite flow

AI-Enterprise-App-Development.pdf

- Connect → cursor → execute (parameterized) → fetch/commit → close, following Python’s sqlite3 DB-API.python

AI-Enterprise-App-Development.pdf

- SELECT reads data; INSERT/UPDATE/DELETE modify data and must commit changes to persist.tutorialspoint+1
- Parameterized queries use ? or named placeholders to prevent SQL injection and handle values safely.tutorialspoint+1

Slide 5: Database assumptions

AI-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

AI-Enterprise-App-Development.pdf

Slide 6: CLI features — functional scope

AI-Enterprise-App-Development.pdf

- Add a new post: prompt for title, content, author id; insert via parameterized

SQL.python AI-Enterprise-App-Development.pdf

- View posts: list PostID, Title, PublishedDate, and AuthorID; support simple ordering.tutorialspoint

AI-Enterprise-App-Development.pdf

- Delete post: confirm and delete by PostID with WHERE to avoid unintended deletions.tutorialspoint

AI-Enterprise-App-Development.pdf

Slide 7: Safety & correctness principlespython

- Always use parameterized queries; never string-concatenate untrusted inputs.tutorialspoint+1
- Validate foreign keys (AuthorID exists) before insert to preserve integrity and UX.sqlite
- Handle exceptions; commit only on success; close connections reliably or use context managers.sqlitetutorial+1

Slide 8: Project skeleton — main looppython

- A looped CLI menu routes to add, view, and delete functions until exit.python
- Keep a single connection per run or use context managers per action; close on program exit.python

Slide 9: Starter scaffold — imports and connectpython

```
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

Slide 10: Add post — validation patternpython

```
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

Slide 11: View posts — simple listpython

```
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

Slide 12: Delete post — confirm & WHERETutorialspoint

```
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

Slide 13: Wiring menu and lifecyclepython

```
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

Slide 14: Parameterization — why it matterspython

- Placeholders bind parameters separately from SQL text, preventing injection and ensuring proper escaping and typing per DB-API.tutorialspoint+1
- 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
- Prefer question-mark style placeholders for sqlite3; named placeholders are also supported.tutorialspoint+1

Slide 15: Transactions & atomicitypython

- sqlite3 auto-commits only when conn.commit() is called; uncommitted changes can be rolled back on errors or exit without commit.tutorialspoint+1
- Use with conn: to scope atomic operations and ensure deterministic commit/rollback behavior on exceptions.mimo+1

- Group multi-statement changes in a single transaction to keep data consistent (e.g., insert post then related tags).mimo+1

Slide 16: Consistency checks — foreign keyssqlite

- Ensure foreign key constraints are enabled if defined; SQLite enforces FKs when PRAGMA foreign_keys=ON is active.sqlite
- Validate dependent records (e.g., AuthorID exists) to provide clear user feedback and avoid constraint errors.sqlite
- Consider ON DELETE CASCADE on FKs if business logic supports it, to avoid orphan rows.sqlite

Slide 17: Class example — nicer listingpython

```
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

Slide 18: Class example — search by keywordpython

```
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

Slide 19: Hands-on lab — build CLI

AI-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

AI-Enterprise-App-Development.pdf

- Ensure parameterized queries everywhere and robust input validation before committing.python

Slide 20: Hands-on validation checklistpython

- Invalid AuthorID is handled gracefully and insertion is prevented.python
- Deleting non-existent PostID prints a friendly message without error.python
- Program exits cleanly and connections are closed or auto-closed via context manager.python

Slide 21: Real-world: UX & reliabilitypython

- Normalize text input (strip whitespace, enforce lengths) and handle Unicode consistently in Title/Content.python
- Add simple paging for large lists (LIMIT/OFFSET) to keep CLI readable.sqlite
- Log actions to a file for auditability; Python logging + timestamps improve diagnostics.python

Slide 22: Real-world: performance notessqlite

- Index frequently filtered columns (e.g., PublishedDate, AuthorID) to accelerate listing/searching.sqlite
- Avoid N+1 query patterns by joining Users when listing Posts to retrieve author names in one query.sqlite

- Keep transactions tight; avoid long-running open transactions that can block writers.sqlite

Slide 23: Real-world: data safetysqlite

- Consider “soft delete” via an IsActive flag instead of hard DELETE for audit history and recovery.sqlite
- Wrap multi-step operations in transactions to ensure all-or-nothing behavior, especially for batch operations.mimo+1
- Backup the DB file periodically; SQLite file backups are straightforward when not in use.sqlite

Slide 24: Bonus 1 — update post contentpython

```
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

Slide 25: Bonus 2 — add soft deletesqlite

- ALTER TABLE Posts ADD COLUMN IsActive INTEGER DEFAULT 1 to mark active rows.sqlite
- Replace hard DELETE with UPDATE Posts SET IsActive = 0 WHERE PostID = ? to preserve data.sqlite
- Filter listings with WHERE IsActive = 1 to hide deactivated posts by default.sqlite

Slide 26: Bonus 3 — batch CSV importsqlitetutorial

```
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

Slide 27: Bonus 4 — recent posts windowsqlite

```
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

Slide 28: Testing strategypython

- Write small tests for each function: add, view, delete, update, and search flows with known fixtures.python
- Use a temporary copy of blog.db or :memory: database for isolated test runs.python
- Verify rows with independent SELECT queries and check commit visibility.python

Slide 29: Debugging tipspython

- Catch sqlite3.Error exceptions and print both message and failing path to diagnose issues quickly.python

- When queries misbehave, print the SQL and parameters and test in sqlite3 CLI for faster iteration.sqlite
- Confirm schema with PRAGMA table_info(Posts) to verify expected columns and types.sqlite

Slide 30: Documentation & referencespython

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

Slide 31: Recap — what was built

AI-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.pdfpython

- Proper lifecycle: connect, operate, commit on change, and close with robust input validation and confirmations.tutorialspoint+1
- A foundation for further features like users, comments, tags, and soft deletion policies. AI-Enterprise-App-Development.pdfsqlite

Slide 32: Deliverables for submission

AI-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
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