Slide 1: Title

- Module 1: Core Python & Data — Week 4 Lecture 18 — Integrating Python with SQL (Part 1) — Using sqlite3 to Connect and Read from blog.db.pythonAI-Enterprise-App-Development.pdf

Slide 2: Today's agenda

- Why integrate Python with SQL and when to use SQLite for local apps and learning workflows.AI-Enterprise-App-Development.pdfpython
- sqlite3 overview and DB-API 2.0 flow: connect → cursor → execute → fetch → close.pythonAI-Enterprise-App-Development.pdf
- Class example: connect to blog.db and fetch all posts with SELECT and fetchall.freecodecampAI-Enterprise-App-Development.pdf
- Hands-on: write a script that prints PostID, Title, PublishedDate, AuthorID for all posts.pynativeAI-Enterprise-App-Development.pdf
- Real-world challenges: parameterization, connection lifecycle, empty results, file paths.AI-Enterprise-App-Development.pdfpython
- Bonus exercises: filter by author, search by keyword, recent posts, headings formatting.freecodecampAI-Enterprise-App-Development.pdf

Slide 3: Where this fits (Week 4 plan)

- This session implements "Connect and Read" from the training plan (Integrating Python with SQL Part 1) to fetch all posts from the SQLite blog database in Python.AI-Enterprise-App-Development.pdf

Slide 4: Why Python + SQLite

- SQLite is lightweight, serverless, and ideal for embedded apps and education, while Python automates queries and presentation for CLI and scripts.pythonAI-Enterprise-App-Development.pdf
- The sqlite3 module is built into Python and complies with DB-API 2.0, ensuring a consistent interface for database operations.pythonAI-Enterprise-App-Development.pdf

Slide 5: DB-API flow (mental model)

- Create a Connection with sqlite3.connect('blog.db') which represents an open database handle.AI-Enterprise-App-Development.pdfpython
- Use connection.cursor() to create a Cursor that executes SQL and manages result sets.pythonAI-Enterprise-App-Development.pdf
- Execute SQL with cursor.execute(...), then fetch results via fetchone/fetchmany/fetchall and close the connection.AI-Enterprise-App-Development.pdfpython

Slide 6: sqlite3 essentials

- Connection is created by sqlite3.connect(path) and is primarily used to create cursors and manage transactions.pythonAI-Enterprise-App-Development.pdf
- Cursor executes statements and retrieves rows; after SELECT, use fetch methods to access rows as tuples.AI-Enterprise-App-Development.pdfpython
- Always close connections, or use with sqlite3.connect(...) as conn: for safe cleanup and transactional semantics.pythonAI-Enterprise-App-Development.pdf

Slide 7: Fetch methods overview

- fetchall(): returns all remaining rows as a list of tuples, commonly used after SELECT for small-to-moderate result sets.freecodecampAI-Enterprise-App-Development.pdf
- fetchone(): returns one row or None, useful for single-record queries or cursors you iterate manually.freecodecampAI-Enterprise-App-Development.pdf
- fetchmany(n): returns up to n rows, helpful for pagination or large datasets to limit memory usage.pynativeAI-Enterprise-App-Development.pdf

Slide 8: Safety with parameters (even for reads)

- Always pass values via placeholders (e.g., WHERE AuthorID = ?) instead of string concatenation to avoid SQL injection and quoting errors.AI-Enterprise-App-Development.pdfpython
- sqlite3 supports the qmark parameter style (?) per DB-API; named parameters are also supported, though paramstyle is documented as qmark.pythonAI-Enterprise-App-Development.pdf

Slide 9: Class example — connect and list posts (plan)

- Script tasks: open blog.db, run SELECT PostID, Title, Content, PublishedDate, AuthorID FROM Posts, fetchall rows, print formatted output, close connection.pythonAI-Enterprise-App-Development.pdf
- This verifies end-to-end connectivity and shows how to structure DB reads in clean, minimal code.freecodecampAI-Enterprise-App-Development.pdf

Slide 10: Code — minimal connection and fetch

- The following example reads all posts and prints selected fields for clarity, showing the DB-API flow in action.AI-Enterprise-App-Development.pdfpython

python

```
import sqlite3 conn = sqlite3.connect('blog.db') # open/create
database file cur = conn.cursor() # create cursor cur.execute('SELECT
```

```
PostID, Title, Content, PublishedDate, AuthorID FROM Posts;') #
run SELECT rows = cur.fetchall() # get all rows for r in rows:
print(f"ID={r} | Title={r[1]} | Date={r[22]} | AuthorID={r[23]}")
conn.close() # cleanly close
```

- This pattern uses connect → cursor → execute → fetchall → close as recommended by the sqlite3 documentation.pythonAI-Enterprise-App-Development.pdf

Slide 11: Using a context manager (recommended)

- Using with sqlite3.connect('blog.db') as conn: ensures commit on success and rollback on exceptions for write ops, and always closes the connection.AI-Enterprise-App-Development.pdfpython

python

```
import sqlite3 with sqlite3.connect('blog.db') as conn: cur =
conn.cursor() cur.execute('SELECT PostID, Title, PublishedDate,
AuthorID FROM Posts ORDER BY PublishedDate DESC;') for pid,
title, date, author in cur.fetchall(): print(f"[{pid}] {title}
({date}) Author={author}")
```

- This idiom simplifies lifecycle and avoids leaking connections in larger scripts.pythonAI-Enterprise-App-Development.pdf

Slide 12: Class example — fetch one vs many

- fetchone(): efficient when expecting a single row like the latest post, while fetchmany(n) is useful for a preview listing.freecodecampAI-Enterprise-App-Development.pdf

python

```
cur.execute('SELECT PostID, Title FROM Posts ORDER BY PublishedDate
DESC;') first = cur.fetchone() # single row top3 = cur.fetchmany(3)
# next 3 rows print("Latest:", first) print("Top3:", top3)
```

- Choose the fetch method that fits the use case and dataset size for clarity and memory efficiency.pynativeAI-Enterprise-App-Development.pdf

Slide 13: Formatting output for CLI readability

- Use f-strings and fixed widths or simple separators for readable terminal output when iterating rows.AI-Enterprise-App-Development.pdfpython
- Consider truncating long content fields for listings and optionally show Content length or a preview snippet.pythonAI-Enterprise-App-Development.pdf

Slide 14: Handling empty results

- If fetchall returns an empty list, print “No posts found” to avoid silent failures and provide user feedback.pynativeAI-Enterprise-App-Development.pdf
- For robust behavior, treat None from fetchone as a normal case (e.g., when a table exists but is empty).freecodecampAI-Enterprise-App-Development.pdf

Slide 15: Verifying schema quickly

- Use PRAGMA table_info(Posts) to confirm column names and types during debugging and before formatting output.sqliteAI-Enterprise-App-Development.pdf

python

```
cur.execute("PRAGMA table_info(Posts);") print(cur.fetchall())
```

- This is especially helpful when lecture or project code evolves over time and schemas change.sqliteAI-Enterprise-App-Development.pdf

Slide 16: Paths and environment notes

- The working directory matters: ensure blog.db exists at the path used by sqlite3.connect, or pass an absolute path.AI-Enterprise-App-Development.pdfpython
- For classroom setups, agree on a standard project folder so scripts resolve blog.db consistently across machines.pythonAI-Enterprise-App-Development.pdf

Slide 17: Class example — fetch by author (parameterized)

- Demonstrates safe query with a WHERE clause using a placeholder to filter by AuthorID.AI-Enterprise-App-Development.pdfpython

python

```
author_id = int(input("AuthorID: ").strip()) cur.execute('SELECT
PostID, Title FROM Posts WHERE AuthorID = ? ORDER BY PublishedDate
DESC;', (author_id,)) for pid, title in cur.fetchall(): print(pid,
title)
```

- Parameterized queries prevent injection and ensure correct typing without manual quoting.pythonAI-Enterprise-App-Development.pdf

Slide 18: Class example — search by keyword (LIKE)

- Use LIKE with parameters for case-insensitive contains search depending on collation; for simple demo, wrap with % on both sides.freecodecampAI-Enterprise-App-Development.pdf

python

```
term = input("Search term: ").strip() pattern = f"%{term}%"
cur.execute('SELECT PostID, Title FROM Posts WHERE Title LIKE
? OR Content LIKE ?;', (pattern, pattern)) for pid, title in
cur.fetchall(): print(pid, title)
```

- This shows dynamic filtering while preserving parameter safety and avoiding string concatenation.AI-Enterprise-App-Development.pdfpython

Slide 19: Hands-on exercise — Fetch all posts

- Build a script fetch_posts.py that connects to blog.db, selects PostID, Title, PublishedDate, AuthorID, prints all rows sorted by PublishedDate DESC, and closes safely.pythonAI-Enterprise-App-Development.pdf
- Test with empty and non-empty databases to verify "No posts found" message and normal output.pynativeAI-Enterprise-App-Development.pdf

Slide 20: Hands-on extension — Author filter

- Extend fetch_posts.py to accept an optional author ID via input and filter results with WHERE AuthorID = ? when provided, else list all posts.AI-Enterprise-App-Development.pdfpython
- Validate input is digit and handle invalid inputs gracefully with clear messaging.pythonAI-Enterprise-App-Development.pdf

Slide 21: Real-world challenge — connection management

- Prefer context managers to ensure deterministic cleanup; for larger apps, centralize connection creation and pass cursors or repository functions.AI-Enterprise-App-Development.pdfpython
- For read-only scripts, keep transactions short and avoid holding locks while printing or formatting output.pythonAI-Enterprise-App-Development.pdf

Slide 22: Real-world challenge — unicode and length

- Ensure terminal and file encoding is UTF-8 so titles and content display correctly, especially for non-ASCII characters.AI-Enterprise-App-Development.pdfpython
- Consider truncation for long content fields in list views, and provide a detail view that retrieves the full content on demand.pythonAI-Enterprise-App-Development.pdf

Slide 23: Real-world challenge — performance basics

- Add ORDER BY on PublishedDate with an index if datasets grow to keep sorting performant in listings.sqliteAI-Enterprise-App-Development.pdf
- Use fetchmany for paging large results to reduce memory footprint and improve responsiveness in CLIs.pynativeAI-Enterprise-App-Development.pdf

Slide 24: Bonus — recent posts (date filtering)

- Use SQLite date/time functions to fetch posts newer than N days directly in SQL for a simple “recent posts” view.sqliteAI-Enterprise-App-Development.pdf

python

```
days = int(input("Show posts from last N days: ")) cur.execute("SELECT
PostID, Title, PublishedDate FROM Posts WHERE PublishedDate >=
DATE('now', ?) ORDER BY PublishedDate DESC;", (f'-{days} day',))
print(cur.fetchall())
```

- SQLite’s built-in date functions allow serverless date filtering without external libraries for basics.sqliteAI-Enterprise-App-Development.pdf

Slide 25: Bonus — selecting subsets of columns

- Select only fields needed for the view (e.g., PostID, Title) to reduce I/O and simplify printing, reserving Content for detail views.geeksforgeeksAI-Enterprise-App-Development.pdf
- This helps keep listings fast and uncluttered while protecting the terminal from overly long lines.geeksforgeeksAI-Enterprise-App-Development.pdf

Slide 26: Bonus — introspection and column names

- Use cursor.description after execute to derive column headers dynamically for generic printers.pymotwAI-Enterprise-App-Development.pdf
- This enables reusable table printers in CLIs without hardcoding column names for every query.pymotwAI-Enterprise-App-Development.pdf

Slide 27: Debugging tips

- If reading fails, first verify the database path and ensure the Posts table exists using PRAGMA table_info(Posts).sqliteAI-Enterprise-App-Development.pdf
- Print the number of rows returned and sample values to validate assumptions before formatting deeply.pynativeAI-Enterprise-App-Development.pdf

Slide 28: Testing checklist

- Works when there are zero, some, and many posts; filters correctly by author; search returns expected matches; handles invalid inputs gracefully.AI-Enterprise-App-Development.pdfpython
- Connection is closed in success and failure scenarios; no stack traces leak to the user in normal invalid input paths.pythonAI-Enterprise-App-Development.pdf

Slide 29: Minimal “gold” solution outline

- Use with sqlite3.connect('blog.db') as conn, build a parameterized SELECT with optional WHERE, order results, fetch and print, then exit.AI-Enterprise-App-Development.pdfpython
- Keep functions small: get_connection(), fetch_all_posts(), fetch_posts_by_author(), and a simple main guard for script entry.pythonAI-Enterprise-App-Development.pdf

Slide 30: Code — consolidated demo (read-only)

- This sample ties together optional filtering and safe output for immediate use in class exercises.AI-Enterprise-App-Development.pdfpython

python

```
import sqlite3 def list_posts(author_id=None): with sqlite3.connect('blog.db')
as conn: cur = conn.cursor() if author_id is None: cur.execute("""SELECT
PostID, Title, PublishedDate, AuthorID FROM Posts ORDER BY PublishedDate
DESC;""") else: cur.execute("""SELECT PostID, Title, PublishedDate,
AuthorID FROM Posts WHERE AuthorID = ? ORDER BY PublishedDate
DESC;""", (author_id,)) rows = cur.fetchall() if not rows: print("No
posts found."); return for pid, title, date, aid in rows: print(f"[{pid}]
{title} ({date}) Author={aid}") if __name__ == "__main__": ans =
input("Filter by AuthorID? (blank = all): ").strip() list_posts(int(ans)
if ans.isdigit() else None)
```

- This pattern mirrors the DB-API workflow and prepares the class for Part 2's write operations.pythonAI-Enterprise-App-Development.pdf

Slide 31: Quality checklist before moving on

- Uses parameterized queries, handles empty results, formats output readably, and closes connections reliably with context manager.AI-Enterprise-App-Development.pdfpython
- Script lives in the project directory so blog.db resolves correctly, and PRAGMA usage is only needed when enforcing FKs in write flows later.pythonAI-Enterprise-App-Development.pdf

Slide 32: Wrap-up and next steps (within Module 1)

- This lecture completes "Connect and Read" and sets up for Part 2, where DML writes (INSERT) will be performed dynamically from Python input in the next session.AI-Enterprise-App-Development.pdfpython

Slide 33: Bonus exercise A — print "top N" posts

- Prompt for an integer N and show only the N most recent posts with ORDER BY and LIMIT to practice controlling result sizes.sqliteAI-Enterprise-App-Development.pdf

Slide 34: Bonus exercise B — post detail view

- Prompt for a PostID and print a full detail view including Content, falling back to "not found" if ID doesn't exist to practice single-row fetchone.freecodecampAI-Enterprise-App-Development.pdf

Slide 35: References

- Training Plan mapping (Week 4 Wednesday), ensuring alignment with curriculum scope and deliverables.AI-Enterprise-App-Development.pdf
- Python sqlite3 (DB-API 2.0) documentation: connect, cursor, execute, fetch methods, and parameter styles.python
- Tutorials on SELECT and fetch patterns with fetchall, fetchone, and fetchmany to reinforce hands-on practice.pynative+1

If a single, copy-paste-ready "gold" script is desired for distribution to learners, the consolidated demo in Slide 30 can be provided as a downloadable file during class.

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.freecodecamp.org/news/work-with-sqlite-in-python-handbook/
4. https://pynative.com/python-sqlite-select-from-table/
5. https://docs.python.org/tr/3.9/library/sqlite3.html
6. https://www.sqlite.org/docs.html
7. https://sqlite.org/lang_datefunc.html
8. https://www.geeksforgeeks.org/python/python-sqlite-select-data-from-table/
9. https://pymotw.com/2/sqlite3/
10. https://docs.python.org/3.9/library/sqlite3.html
11. https://datacarpentry.github.io/python-ecology-lesson/instructor/09-working-with-sql.html
12. https://stackoverflow.com/questions/62340498/open-database-files-db-using-python
13. https://stackoverflow.com/questions/7831371/is-there-a-way-to-get-a-list-of-column-names-in-sqlite
14. https://realpython.com/ref/stdlib/sqlite3/
15. https://www.sqlitetutorial.net/sqlite-python/sqlite-python-select/
16. https://cewing.github.io/training.codefellows/lectures/day21/intro_to_dbapi2.html
17. https://flask.palletsprojects.com/en/stable/patterns/sqlite3/
18. https://www.youtube.com/watch?v=Hyo9rIuYlFc
19. https://stackoverflow.com/questions/55067006/using-sqlite3-db-api-multiple-parameter-substitution-in-select-statements
20. https://www.digitalocean.com/community/tutorials/how-to-use-the-sqlite3-module-in-python-3

21. https://www.tutorialspoint.com/sqlite/sqlite_python.htm
22. https://duckdb.org/docs/stable/clients/python/dbapi.html
23. https://carpentry.library.ucsb.edu/2021-08-23-ucsb-python-online/09-working-with-sql/index.html
24. 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
25. https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/attachments/77906941/3a281410-3233-4813-b42b-a5659c1a1224/MD_1_Core_python_-*data_W_4_L_16*-Date_08_Sept_2025.ipynb