A Collection of Demonstrations with PostgreSQL for Teaching and Learning Database System Internals*

Stefan Halfpap[1,*], Daniel Hristov[2] and Volker Markl[3]

[1]BIFOLD, TU Berlin, Germany
[2]TU Berlin, Germany
[3]BIFOLD, TU Berlin, DFKI, Germany

Abstract

Relational database systems implement fundamental concepts, such as the relational model, physical design independence, query optimization, and transaction processing. However, teaching and learning these concepts – alongside system design and implementation – can be challenging as database systems are among the most complex software systems, particularly when aiming for a practical and hands-on educational approach. A powerful way to bridge this gap is through demonstrations that showcase database internals using actual systems. PostgreSQL, in particular, provides extensive capabilities for inspecting internal structures and behavior, making it an ideal system for such demonstrations.

In this paper, we present a comprehensive collection of demonstrations designed to support the teaching and learning of database system internals using PostgreSQL. These demonstrations cover concepts from database storage representation (e.g., file organization, page structure, and tuple representation) and query processing (e.g., cardinality estimation based on statistics and transaction processing). In addition, we present a novel application for visually inspecting the structure of PostgreSQL data pages and the effects of data modifications. Finally, we discuss how we use the demonstrations in our courses at TU Berlin. Our paper, including the accompanying online artifacts, builds a novel collection of hands-on demonstration proposals for teaching and learning database system internals.

Keywords

database system education, database system internals, PostgreSQL, database page layout visualizer

1. Introduction

Database systems apply many core concepts from computer science, such as caching, concurrency control, and operating system interaction. At the same time, relational database systems implement many essential concepts, e.g., the relational model [1], including physical design independence and query optimization, as well as transaction processing [2]. Learning these concepts and their implementation is an essential skill for software engineers, as database systems are among the most complex software systems, and the concepts include timeless patterns for performant systems programming.

Traditional database courses and textbooks [3, 4, 5] cover database system concepts in depth. However, understanding internal database functionality requires more than theoretical knowledge. While some courses let students reimplement database system functionality, students may struggle to bridge the gap between theoretical concepts on the one side and practical system design [6] and implementation on the other side. Demonstrations using real database systems can enhance learning by providing visual, concrete, and interactive examples of internal mechanisms. Further, they expose students to real-world implementation details, optimizations, and trade-offs.

For example, the concept of a slotted page structure [5] is simple in theory, but understanding how to access individual fields of a specific record and what kind of metadata or optimizations are possible

DataEd'26: 5th International Workshop on Data Systems Education
* Published in the Proceedings of the Workshops of the EDBT/ICDT 2026 Joint Conference (March 24-27, 2026), Tampere, Finland
*Corresponding author.
halfpap@tu-berlin.de (S. Halfpap); d.hristov@campus.tu-berlin.de (D. Hristov); volker.markl@tu-berlin.de (V. Markl)
https://halfpap.de (S. Halfpap)
0009-0009-3989-261X (S. Halfpap); 0009-0001-9891-9108 (D. Hristov); 0009-0009-0964-026X (V. Markl)

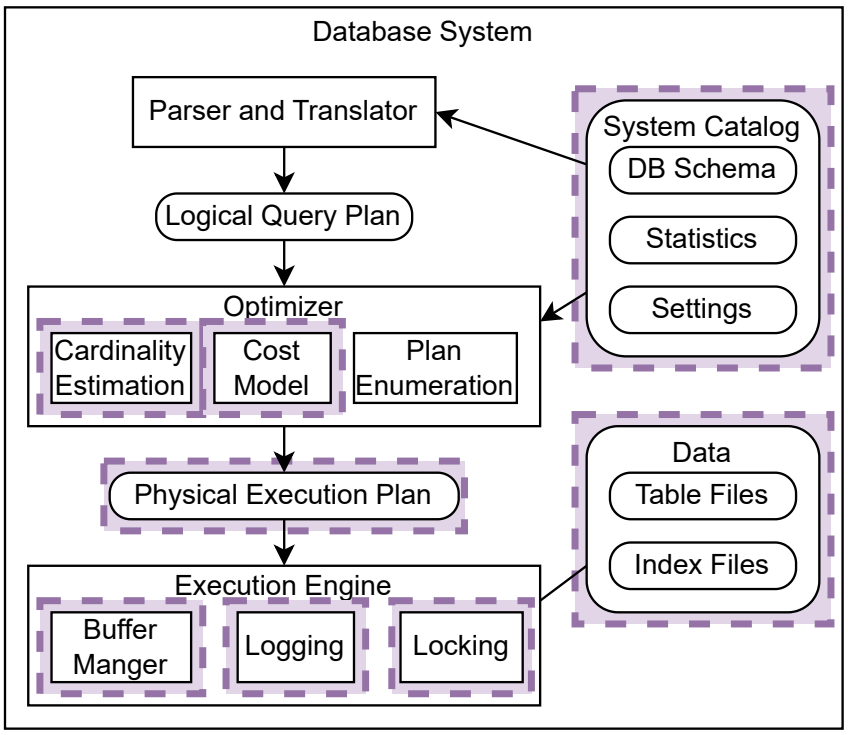

Figure 1: Database system internals and components covered by the demonstrations

requires deeper discussions. Similarly, basic cost estimation techniques, such as those introduced in System R [7] using the number of distinct values, provide a useful starting point, yet real-world cost models and statistics are more advanced. By demonstrating these concepts and implementation details within an actual system, students gain a more practical and more profound understanding, i.e., an overall better learning experience.

Challenges. Despite their benefits, preparing effective demonstrations for teaching database system internals is time-consuming. First, course instructors must think of suitable demonstrations for the concepts to be taught or implemented. Further, they must check for viable experiments as database system internals are commonly not exposed to the user. Finally, they must set up the database and commands, and test the envisioned demonstration.

Partly, practical demonstrations are already described in system documentations, or blog posts. Occasionally, system demonstrations are already part of some courses, but the material is often not publicly available or hard to reproduce, for example, if only a video recording of the lecture is available. This makes it difficult for lecturers with limited time to use these demonstrations. However, if you know which and how demonstrations can be easily performed, they could be more frequently included in courses.

Paper Idea: Demonstrating Database System Internals. This paper presents a collection of reproducible demonstrations for teaching and learning database internals using PostgreSQL. The demonstrations are aligned with core topics taught in database courses, covering: (1) the database representation on disk, including file organization, page structure, and tuple representation, and (2) query processing, including cardinality and cost estimations based on statistics and knob settings, buffer management, and transaction processing (see also Figure 1).

Reproducibility Artifacts [8]. To support adoption, we provide artifacts to easily reproduce and run the demonstrations. Further, we group and link the demonstrations to specific topics of database courses.

Why PostgreSQL. For demonstrations of database system internals, PostgreSQL is an ideal choice. First, it is a popular, commonly-used, and mature system, making it a relevant reference implementation of a row-based relational database system to know about. Second, PostgreSQL has a great documentation [9], and its internals are covered by many books [10], blog posts [11], and other resources [12]. Further, it offers extensions and corresponding documentation [13] to inspect internals out of the box. Finally, it is open-source [14], so the demonstrations provide an entry point to analyze the implementation even more deeply.

Novelty. This paper emphasizes the benefit and ease of teaching database internals using demonstrations with PostgreSQL. The paper and accompanying artifacts build a novel common collection to support the teaching of database internals throughout multiple lectures. In addition, we implemented a tool to visually inspect PostgreSQL data pages and the effects of data modifications, e.g., added tuples via INSERTs. The presented demonstrations are inspired, enabled, and partly adopted from existing write-ups and PostgreSQL extensions (see Section 4): without the previously publicly available material, collecting, designing, and engineering the demonstrations would have been more tedious.

Out of scope. Although this paper and our online repository provide many versatile demonstrations, this collection cannot be exhaustive due to the vast possibilities and complexity of PostgreSQL, SQL, and their combination. Further, this paper does not focus on performance tuning for database systems. We also do not discuss demonstrations using modifications of the PostgreSQL source code. Finally, we do not discuss alternative systems besides PostgreSQL to demonstrate database system internals.

Contributions. We present a collection of demonstrations for teaching and learning database system internals using PostgreSQL:

- The demonstrations (see Figure 1) cover the database representation on disk and query processing (see Section 2).
- The demonstrations are easy to reproduce and run, supported by online artifacts [8] with step-by-step instructions and additional examples.
- We present a novel application for visually explaining and inspecting the slotted page structure using actual PostgreSQL pages (see Figure 2).
- We discuss how the demonstrations support education for university courses and beyond (see Section 3).

2. Demonstrations of DBS Internals

Database system internals are usually hidden from the user, who interacts with the system via the declarative query language SQL and thus does not have to know how data is physically stored on disk or processed. However, exploring these internals can deepen our understanding of general database system design and implementation details. In this section, we introduce a series of demonstrations that reveal key aspects of PostgreSQL's implementation to support teaching and learning database system internals. We organize the demonstrations into two categories: (1) database representation on disk (see Section 2.1) and (2) query processing (see Section 2.2). For both categories, we first provide an overview of the key concepts and demonstrations. Afterward, we describe details for individual demonstrations. Reproducible step-by-step instructions, including additional and more advanced demonstrations, are available online [8].

2.1. Database Representation on Disk

Ultimately, all database information is stored on disk, usually in files to use the operating system's file management capabilities. Database files are usually split into fixed-sized blocks (or pages), forming the units of data transfer for processing in memory. The blocks themselves store tuples. To efficiently access individual variable-sized tuples of a relation, tuples are stored using a slotted page structure [5].

By inspecting PostgreSQL's database representation on disk, we can better understand how database systems can organize and retrieve data. The following demonstrations cover file organization (see Section 2.1.1), page structure (see Section 2.1.2), and tuple representation (see Section 2.1.3).

2.1.1. File organization

PostgreSQL stores its configuration and data (e.g., tables, indexes) in files. We can query the corresponding directory in the file system using SQL:

```
SELECT setting FROM pg_settings WHERE name = 'data_directory';
```

```
          setting
----------------------------
 /opt/homebrew/var/postgres
```

Listing 1: PostgreSQL directory

The view `pg_settings` provides access to PostgreSQL's configuration settings, including parameter names (`name`), current values (`setting`), and units (`unit`). We can use the `SHOW` and `SET` commands to display and change individual (e.g., `SHOW name`) or all (e.g., `SHOW ALL`) parameters.

Files for specific tables (and indexes) are stored inside the database's subdirectory. Each database is assigned an OID (object identifier), which can be queried:

```
SELECT oid FROM pg_database WHERE datname = 'demo_db_internals';
```

```
  oid
-------
 16817
```

Listing 2: Object identifier for a database

We can also retrieve filenames for specific tables in this subdirectory or directly query the file path for a relation:

```
SELECT pg_relation_filepath('region');
```

```
 pg_relation_filepath
----------------------
 base/16817/16821
```

Listing 3: File path for a table

2.1.2. Page Structure

PostgreSQL organizes individual data files into fixed-size slotted pages (default 8 KB). Each page consists of:

- A 24-byte header for metadata, e.g., available free space
- An array of 4-byte tuple pointers
- The record data, containing actual table rows

We can examine the internal structure of a page using the `pageinspect` extension. After enabling this extension, we can, for example, inspect the page header of the `region` table's first block with the following SQL command:

```
CREATE EXTENSION pageinspect;
SELECT * FROM page_header(get_raw_page('region', 0));
```

```
 ... | lower | upper | ...
-----+-------+-------+-----
 ... |   44  |  7568 | ...
```

Listing 4: Page header with free space information

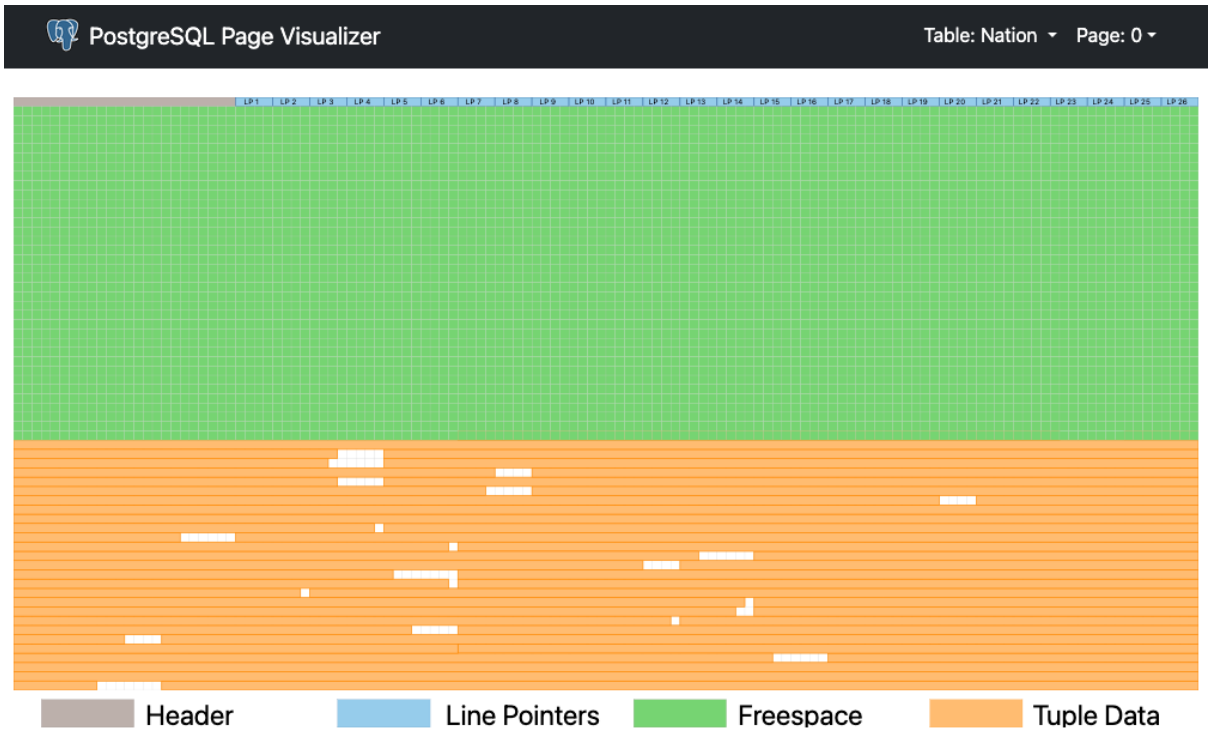

Figure 2: Screenshot of our web application for visualizing PostgreSQL pages with page header (grey), line pointers (blue), free space (green), and tuple data (orange)

2.1.3. Tuple Representation

The `pageinspect` extension also allows us to inspect individual tuples stored per page. Listing 5 shows a query that retrieves the line pointers (`lp`), byte offsets (`lp_off`), and MVCC information (`t_xmin`, `t_xmax`, `t_ctid`) of all tuples that are stored on the `user` table's first block.

```
SELECT lp, lp_off, t_xmin, t_xmax, t_ctid, t_data
FROM heap_page_items(get_raw_page('user', 0));
```

```
 lp | lp_off | t_xmin | t_xmax | t_ctid |  t_data
----+--------+--------+--------+--------+----------
  1 |   8160 | 146456 | 146457 | (0,2)  | \x010...
  2 |   8128 | 146457 |      0 | (0,2)  | \x010...
```

Listing 5: Byte offsets and MVCC information per page tuple

As part of the demonstrations, we implemented an application to visually inspect individual PostgreSQL data pages (see Figure 2). Besides providing a general visual overview of a slotted page implementation, users can also inspect how pages change as a result of data modification operations.

2.2. Query Processing

Query processing involves multiple steps within the database system. The declarative SQL query is first parsed and transformed into a logical query plan. The logical query plan is converted into a physical execution plan. To select the most efficient execution plan, the optimizer evaluates multiple equivalent plans using data statistics, a cost model, and an enumeration strategy.

Using demonstrations with PostgreSQL, we can inspect some of the internal query processing for a deeper understanding. First, we can inspect the currently stored table statistics (see Section 2.2.1). Second, we can examine the chosen execution plan (see Section 2.2.2). Additionally, we can inspect some execution internal behavior, e.g., the number of actually accessed buffers and the state of the buffer cache (see Section 2.2.3). Further, we can explore transaction processing (see Section 2.2.4). Finally, we can observe logging, running processes and the messages exchanged between the user client and the database server (see accompanying artifacts [8]).

2.2.1. Statistics and Cardinality Estimations

PostgreSQL stores relation-level statistics, such as the estimated number of tuples (`reltuples`) and the relation size in pages (`relpages`) in the catalog table `pg_class`:

```
SELECT reltuples, relpages FROM pg_class WHERE relname = 'nation';
```

```
 reltuples | relpages
-----------+----------
        25 |        1
```

Listing 6: Estimated table cardinality and number of pages

These statistics serve as the basis for cardinality estimations. Since these estimates are based on sampling, they may be inaccurate. Their (re)creation can be manually and automatically triggered.

PostgreSQL also maintains attribute-level statistics, which can be queried using the view `pg_stats` (see Listing 7), for example:

- The number of distinct values (`ndistinct`); a negative value means that PostgreSQL assumes the number of distinct values increases with the table cardinality, estimating it as `−reltuples/ndistinct`
- The most common values (`most_common_vals`) and their frequencies (`most_common_freqs`)
- The fraction of NULL values (`null_frac`)
- Histogram-based data distributions

```
SELECT n_distinct, most_common_vals, most_common_freqs
FROM pg_stats
WHERE tablename = 'nation' and attname = 'n_regionkey';
```

```
 n_distinct | most_common_vals |   most_common_freqs
------------+------------------+-----------------------
       -0.2 | {0,1,2,3,4}      | {0.2,0.2,0.2,0.2,0.2}
```

Listing 7: Attribute-level statistics

2.2.2. Query Execution Plan and Cost Model

The PostgreSQL optimizer selects an execution plan by estimating costs for different equivalent plans. We can inspect the chosen plan for a given SQL query using the EXPLAIN command (see Listing 8). The command also exposes total execution costs (`cost`), estimated cardinalities (`rows`), and information per operator:

```
EXPLAIN SELECT * FROM nation WHERE n_regionkey = 4;
```

```
                       QUERY PLAN
-------------------------------------------------------
 Seq Scan on nation  (cost=0.00..1.31 rows=5 width=109)
   Filter: (n_regionkey = 4)
```

Listing 8: Query plan, expected costs, and cardinalities

2.2.3. Query Execution Information

The chosen execution plan can be further analyzed by actually running the query using the ANALYZE option (see online artifacts [8]). The resulting output includes the optimization and run time, as well as actual cardinalities, which can be compared to estimated cardinalities for each operator.

The BUFFERS option in EXPLAIN ANALYZE reveals further execution details (see online artifacts [8]). We can, for example, inspect how many pages were already present in the buffer cache and how many were read from disk or the operating system cache. Further, we can check temporary file usage for

pipeline breakers, such as hash table builds or sorts. Finally, we can inspect how many pages were modified in the buffer and written back to disk.

The PostgreSQL extension `pg_buffercache` enables an inspection of shared buffer states. This extension provides the `pg_buffercache` view, where each row represents a page in the shared buffer. For every buffer slot, we can derive the buffered page (i.e., which block of which relation), whether the page is dirty (`isdirty`), the current access counter for the clock-sweep replacement algorithm (`usagecount`), and how many processes the page currently pin (`pinning_backends`):

```
CREATE EXTENSION pg_buffercache;
SELECT * FROM pg_buffercache LIMIT 2;
```

```
 bufferid | ... | isdirty | usagecount | pinning_backends
----------+-----+---------+------------+------------------
        1 | ... | f       |          5 |                0
        2 | ... | f       |          5 |                0
```

Listing 9: Buffer cache information

Using the `pg_buffercache` view, we can also query statistics (1) about used, dirty, pinned pages and (2) for which relation there are currently the most cached pages [8].

2.2.4. Transaction Processing

PostgreSQL also offers ways to demonstrate transaction processing (see online artifacts [8]): First, we can inspect the results of transactions regarding MVCC meta information, including visibility information and the version chain of records (see also Section 2.1.3). We can also inspect currently running transactions using the `pg_stat_activity` view and inspect held locks using the `pgrowlocks` module. The following query shows the row, transaction, and mode for the currently held locks on the `orders` table:

```
CREATE EXTENSION pgrowlocks;
SELECT locked_row, locker, modes FROM pgrowlocks('orders');
```

```
 locked_row | locker |       modes
------------+--------+-------------------
 (0,3)      | 146453 | {"No Key Update"}
```

Listing 10: Locked rows of a table

3. Usage for Education

In this section, we describe and discuss the use of our demonstrations for educational purposes. First, we provide an overview of possible usage scenarios within and beyond the university context (see Section 3.1). In Section 3.2, we explain how the demonstrations map to common database system course topics and illustrate, using a concrete example, how they enhance the teaching of database internals. Finally, we describe how the demonstrations can be run during courses (see Section 3.3) and report on our teaching experiences (see Section 3.4).

3.1. Usage Possibilities

Our hands-on demonstrations offer versatile applications for teaching and learning database system internals in both academic and non-academic settings. We primarily developed the collection for our university lecture on database system internals, which is a common and fundamental course in computer science.

In addition to lectures, the demonstrations can also support other database-related courses. At TU Berlin, we offer a practical, hands-on database systems course that includes performance analysis and tuning. In this course, we have used our PostgreSQL setup and selected demonstrations, such as inspecting configuration parameters and query execution plans. Furthermore, our demonstrations can support implementation-oriented database system courses, as they introduce (i) existing implementations of

Table 1
Suitable demonstrations per course topic

Course Topic	Suitable Demonstrations
Database Representation	File organization (Slotted) Page structure Tuple representation
Indexing	Index utilization Index representation on disk
Query Optimization	Statistics Cardinality estimation Query plan
Query Execution	Query execution plan Buffer usage
Concurrency Control	Running transactions MVCC columns Row locking
Recovery	Running transactions Write-ahead log

features that students should implement on their own or (ii) PostgreSQL's implementation, which shall be adapted or extended [15].

The educational use of our demonstrations is not limited to university settings. This paper and its accompanying artifacts also enable self-education and learning outside of universities. For this purpose, our online materials [8] include short descriptions of the underlying concepts, along with links to additional documentation and resources.

3.2. Demonstrations per Course Topic

We collected our demonstrations to cover a broad range of fundamental topics commonly taught in courses on database system internals. Table 1 maps individual demonstrations to typical course topics. All listed demonstrations can be performed independently.

Example: File Organization and Page Structure. In the following, we illustrate how specific demonstrations support teaching how database systems store tables. We teach our students that all database information is usually stored in files, which are divided into fixed-sized blocks (or pages). Furthermore, tuples are organized within these pages using a slotted page structure.

While these principles are general, the concrete file organization (e.g., directory structure and naming) and file format (i.e., byte layout) of slotted pages are system-specific. We discuss exemplary implementation choices and their trade-offs in our lectures. Using our demonstrations, we can then directly show how PostgreSQL organizes table files and implements slotted pages.

In this process, we switch from a slide presentation to demonstrations. We query the file name of an existing table and show the directory structure to explain PostgreSQL's file organization. Likewise, we inspect individual files to illustrate how PostgreSQL implements slotted pages. At this point, instructors may return to the slide presentation to introduce additional theoretical concepts. Alternatively, they may continue with further demonstrations showing how data modification operations affect files.

Overall, each demonstration consists of only a small number of SQL statements and can therefore be presented often within five minutes, including the accompanying explanations. Lecturers can decide how many and which demonstrations to present during a lecture. Using this approach, theoretical concepts introduced through slides can be complemented with practical demonstrations of implementation details in a real system.

3.3. Running the Demonstrations

The demonstrations can be executed on existing PostgreSQL installations. In addition, we provide a Docker-based setup that includes a script for loading exemplary datasets [8]. The demonstrations can be run directly during course sessions, for example, using PostgreSQL's command-line tool `psql`. Overall, instructors can choose their preferred tools for running the queries, including environments based on Jupyter notebooks [16], which enable the structured preparation of query workflows and the integration of additional material alongside the SQL statements. Most demonstrations can also be performed using PGLite [17], a lightweight PostgreSQL variant that runs entirely in the browser. As such, it requires no additional PostgreSQL installation and only minimal setup, such as creating exemplary tables. PGLite currently supports the extensions `pageinspect` and `pg_buffercache`, but it provides only a limited subset of extensions, which may restrict desired demonstrations.

Course instructors can also choose which demonstrations to run and combine (see Section 3.2). They may also adapt the demonstrations, for example, by using custom datasets or deciding which PostgreSQL information to expose. Furthermore, instructors can develop new demonstrations or present entire lecture units using Jupyter notebooks.

3.4. Experience Report

Overall, we like a teaching approach that combines fundamental concepts with insights into concrete implementation details. In our experience, live demonstrations of system internals make lectures more engaging and tend to attract greater student attention than slide-based explanations alone. Furthermore, the demonstrations provide a starting point for interested students to explore system implementations in greater depth.

When preparing lectures, we identify suitable demonstrations in advance (see Table 1) and integrate them into the presentation flow. To date, we have primarily focused on short demonstrations using `psql` during lectures. For our practical and exercise sessions with longer, more complex demonstrations, we prefer Jupyter notebooks, in which all queries and their order can be prepared. In general, careful preparation, such as selecting an appropriate setup, preloading datasets, and preparing the required commands (see online artifacts [8]), helps to ensure smooth execution.

4. Related Work and Source Material

The topic "Teaching Database System Internals using Demonstrations" falls into the broader category of database system education. Database system courses are often based on textbooks [3, 4, 5], which primarily focus on foundational concepts. While some textbooks discuss specific database systems [5], they do so only to a limited extent and do not provide structured demonstration proposals. Courses often use also more recent database research ideas and discuss system implementations, but to a limited extent. Demonstrations are occasionally included in lectures, as is evident from some publicly available lecture recordings (e.g., for courses of the CMU database group [18]). However, these demonstrations are usually not publicly available or reproducible using the publicly available material (e.g., slides and video stream recordings).

For PostgreSQL, a wealth of educational material is available. However, no centralized repository of demonstration proposals specifically aimed at teaching and learning database system internals exists. Potentially closest to our work is the website *The Internals of PostgreSQL* [11], which served as an early inspiration for our collection of demonstrations. This website offers a visual documentation of PostgreSQL, explaining many concepts including some demonstrations of the database representation on disk. However, its primary focus is documentation rather than designing or structuring demonstrations for course usage. Similarly, the official PostgreSQL documentation [9] and the book *PostgreSQL 14 Internals* [10] provide detailed descriptions of PostgreSQL's implementation. Although these resources contain valuable technical insights, their comprehensiveness makes it tedious to extract structured

demonstration proposals directly. Additional valuable sources include the PostgreSQL Weekly newsletter [12] and contributions of various PostgreSQL events [19, 20]. Furthermore, many blog posts cover PostgreSQL details, frequently including code snipped for demonstration purposes.

In this paper, we design and collect a series of structured demonstration proposals to aid database system education. In this process, we inspected the mentioned related work, which inspired us and taught us about PostgreSQL's possibilities, enabling viable demonstrations. We focus on practical, easy-to-use, and directly applicable demonstrations, and also provide reproducibility artifacts [8].

5. Conclusion

This paper presents a collection of demonstrations using PostgreSQL to enhance the teaching and learning of database system internals. The demonstrations cover a broad range of fundamental topics commonly taught in university database courses. In addition, we present a novel application for visualizing and explaining the slotted page structure using actual PostgreSQL pages. By exposing real-world implementation details, optimizations, and trade-offs, our demonstrations can deepen the understanding of database system internals. Beyond traditional classroom settings, they can serve as a valuable resource for self-study and hands-on exploration.

Declaration on Generative AI

During the preparation of this work, the authors used ChatGPT and Grammarly in order to: Grammar and spelling check, Paraphrase and reword. After using these tools/services, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] E. F. Codd, A relational model of data for large shared data banks, Commun. ACM 13 (1970) 377–387. URL: https://doi.org/10.1145/362384.362685.

[2] J. Gray, The transaction concept: Virtues and limitations, in: Proceedings of the International Conference on Very Large Databases (VLDB), 1981, pp. 144–154.

[3] H. Garcia-Molina, J. D. Ullman, J. Widom, Database Systems: The Complete Book, 2nd ed., Pearson Education, 2009.

[4] R. Ramakrishnan, J. Gehrke, Database Management Systems, 3rd ed., McGraw-Hill, 2003.

[5] A. Silberschatz, H. F. Korth, S. Sudarshan, Database System Concepts, 7th ed., McGraw-Hill, 2020.

[6] J. M. Hellerstein, M. Stonebraker, J. R. Hamilton, Architecture of a database system, Found. Trends Databases 1 (2007) 141–259. URL: https://doi.org/10.1561/1900000002.

[7] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, T. G. Price, Access path selection in a relational database management system, in: Proceedings of the International Conference on Management of Data (SIGMOD), 1979, pp. 23–34. URL: https://doi.org/10.1145/582095.582099.

[8] S. Halfpap, A Collection of Demonstrations with PostgreSQL for Teaching and Learning Database System Internals, https://github.com/klauck/demo_dbs_internals, 2026. Accessed: 2026-02-20.

[9] The PostgreSQL Global Development Group, PostgreSQL Documentation, URL: https://www.postgresql.org/docs/, 2026. Accessed: 2026-02-20.

[10] E. Rogov, PostgreSQL 14 Internals, URL: https://postgrespro.com/community/books/internals, 2023. Accessed: 2026-02-20.

[11] H. Suzuki, The Internals of PostgreSQL, URL: https://www.interdb.jp/pg/, 2015. Accessed: 2026-02-20.

[12] P. Cooper, Postgres Weekly, 2026. URL: https://postgresweekly.com/, accessed: 2026-02-20.

[13] PostgreSQL Global Development Group, PostgreSQL Documentation: Additional Supplied Modules and Extensions, 2026. URL: https://www.postgresql.org/docs/current/contrib.html, accessed: 2026-02-20.

[14] PostgreSQL Global Development Group, PostgreSQL Source Code, 2026. URL: https://git.postgresql.org/gitweb/?p=postgresql.git, accessed: 2026-02-20.
[15] A. Ailamaki, J. M. Hellerstein, Exposing undergraduate students to database system internals, SIGMOD Rec. 32 (2003) 18–20. URL: https://doi.org/10.1145/945721.945725.
[16] Project Jupyter, Jupyter Notebooks, 2026. URL: https://jupyter.org/, accessed: 2026-02-20.
[17] ElectricSQL, PGlite: Embeddable Postgres in WebAssembly, 2026. URL: https://pglite.dev/, accessed: 2026-02-20.
[18] Carnegie Mellon University Database Group, Database Courses at CMU, 2026. URL: https://db.cs.cmu.edu/courses/, accessed: 2026-02-20.
[19] PostgreSQL Global Development Group, PostgreSQL Events, 2026. URL: https://www.postgresql.org/about/events/, accessed: 2026-02-20.
[20] PostgreSQL Europe, PostgreSQL Europe Events, 2026. URL: https://www.postgresql.eu/events/, accessed: 2026-02-20.