International Journal of Innovative Information Systems & Technology Research 14(3):90-93, July-Sept, 2026

© SEAHI PUBLICATIONS, 2026 www.seahipublications.org ISSN: 2467-8562

DOI:10.5281/zenodo.21381053

A Systematic Review of Record and File Manipulation Constructs in Modern Programming Languages and DBMS

Obi Ohia[1], Oruan Memoye Kepeghom PhD.[2]

Department of Computer and Robotics Education
Federal College of Education (Technical) Omoku, Rivers State, Nigeria.
opwservices@gmail.com[1], oruanmem@gmail.com[2], memoye.oruan@fcetomoku.edu.ng[2]
ORCID:0000-0001-5746-1869

Abstract
Records and files are fundamental abstractions in computing systems, appearing at multiple layers ranging from logical tuples in relational databases to physical storage units in files and pages, and further to structured objects manipulated in programming languages. This paper presents a comprehensive and critical examination of the constructs used for the creation and manipulation of records and files in modern Database Management Systems (DBMS) and programming environments. Drawing from the relational model and contemporary database architectures, the study explores SQL-based schema definition and data manipulation, transaction control mechanisms, physical storage strategies, and programming-language interfaces such as embedded SQL and database APIs. It further investigates emerging paradigms where the traditional boundaries between records and files are increasingly blurred, including DBMS integration with distributed file systems, large-scale file archiving systems supported by database indexing, and the use of relational operations as a computational substrate. The paper concludes that modern data systems are converging toward hybrid architectures in which records and files function as complementary layers, unified through advanced abstraction mechanisms and system design principles.
Keywords: embedded SQL, database APIs, Data Manipulation, File Organization, Distributed File Systems, Transaction Processing.

1. Introduction

The design of modern data systems reflects a fundamental challenge: providing simple and intuitive abstractions for users while managing complex internal processes such as storage allocation, concurrency, and fault tolerance. At the user level, operations such as inserting, updating, or retrieving records appear straightforward; however, these operations are supported by intricate mechanisms within the Database Management System (DBMS). The relational model addressed this challenge by introducing the concept of data independence, which separates logical data representation from physical storage details, thereby insulating applications from changes in storage structures (Codd, 1970).

This foundational idea led to the development of a universal data sublanguage capable of defining and manipulating data independently of the underlying system. In modern systems, SQL embodies this vision, serving as the standard language for database interaction. However, SQL alone cannot support all aspects

of application logic, necessitating integration with general-purpose programming languages that provide control flow, user interaction, and computational capabilities (Elmasri & Navathe, 2016).

As data systems evolve toward distributed and large-scale environments, the distinction between records and files becomes less rigid. Systems increasingly integrate database functionality with file system scalability, giving rise to hybrid architectures. This paper examines these constructs across multiple layers, providing a unified understanding of their roles and interactions.

2. Conceptual Foundations of Records and Files

The concept of records and files varies depending on the level of abstraction. At the logical level, the relational model represents data as relations composed of tuples, where each tuple corresponds to a record. These records are manipulated using declarative operations that define what data is required rather than how it should be retrieved (Abiteboul et al., 1995). This abstraction eliminates the need for users to manage low-level storage details and enables powerful query optimization techniques.

At the physical level, records are stored within files organized into pages and blocks on storage media. The DBMS storage engine manages these structures, ensuring efficient access and modification. The distinction between schema and instance further clarifies this abstraction: the schema defines the structure of the database, while the instance represents its current state (Elmasri & Navathe, 2016).

The principle of physical data independence ensures that changes in file organization or storage methods do not affect application programs. This separation is critical for scalability and maintainability, allowing systems to evolve without disrupting existing functionality (Silberschatz et al., 2019).

3. Constructs for Record and File Creation in DBMS

The creation of records and files in a DBMS is primarily governed by SQL Data Definition Language (DDL). These constructs enable the specification of database schemas, including relations, attributes, constraints, and storage properties. The CREATE TABLE statement defines the structure of a relation, specifying attribute types and integrity constraints such as primary keys and foreign keys, which ensure data consistency (Elmasri & Navathe, 2016).

Beyond logical definitions, DDL operations are compiled into metadata stored in the system catalog. This catalog acts as the internal representation of the database schema and plays a crucial role in query processing and storage management. It contains information about tables, indexes, constraints, and access methods, allowing the DBMS to interpret and optimize user queries (Silberschatz et al., 2019).

Thus, record creation in a DBMS involves both logical specification and physical instantiation, with the system catalog serving as the bridge between these layers.

4. Record Manipulation Constructs in DBMS

Data Manipulation Language (DML) provides the primary constructs for modifying records within a database. The core operations—INSERT, DELETE, and UPDATE—are designed to operate on sets of records, reflecting the declarative nature of SQL (Melton & Simon, 2001).

Insertion operations allow new records to be added either through explicit values or by deriving values from query results. Deletion operations remove records based on specified conditions, while update operations modify existing records. These operations must be carefully defined to ensure consistency, particularly when nested queries or dependencies exist between records (Patni, 2022).

The set-oriented nature of these constructs enables efficient processing of large datasets, as the DBMS can optimize operations internally without requiring explicit iteration by the user.

5. Transaction Management and Data Integrity

Transaction management is essential for ensuring the correctness of record manipulation. A transaction is a sequence of operations that must be executed atomically, maintaining database consistency even in the presence of failures or concurrent access. The ACID properties—atomicity, consistency, isolation, and durability—provide the foundation for reliable transaction processing (Gray & Reuter, 1992).

SQL provides constructs such as COMMIT and ROLLBACK to define transaction boundaries. While many systems implement autocommit behavior for simplicity, explicit transaction control is necessary for complex operations involving multiple steps. Isolation levels further allow systems to balance consistency and performance by controlling how transactions interact with one another (Patni, 2022).

6. Physical Storage: Record Layout and File Organization

The physical realization of records involves their storage within files using various organizational strategies. Records may be stored in fixed-length or variable-length formats, each with distinct advantages. Variable-length records require additional mechanisms such as offset tables and null indicators to manage attribute variability (Silberschatz et al., 2019).

The slotted-page structure is commonly used to manage records within a page, allowing records to be moved without invalidating references. At a higher level, file organization methods such as heap files, sequential files, hash-based files, and B+-tree indexed files determine how records are stored and accessed. These methods significantly influence the performance of database operations (Elmasri & Navathe, 2016).

Large objects introduce additional complexity, as they may be stored externally in file systems or internally within the database. This demonstrates the interaction between file system constructs and database storage mechanisms.

7. Programming-Language Constructs for Record and File Manipulation

Programming languages play a critical role in extending database functionality beyond what SQL can provide. They offer constructs for control flow, user interaction, and integration with external systems. Database APIs such as JDBC and ODBC enable applications to connect to databases, execute queries, and process results (Silberschatz et al., 2019).

Embedded SQL provides a tighter integration, allowing SQL statements to be written directly within host-language code. Constructs such as host variables and cursors facilitate data exchange and sequential processing of query results. These mechanisms enable applications to manipulate records efficiently while leveraging the DBMS for storage and consistency management (Elmasri & Navathe, 2016).

Transaction control is also exposed through programming interfaces, enabling developers to manage complex operations explicitly (Patni, 2022).

8. Convergence of DBMS and File Systems

Modern data systems increasingly integrate DBMS functionality with file system scalability. One example is the integration of DBMS with distributed file systems, where database features such as transactions and indexing are layered on top of scalable storage infrastructures. The concept of meta DFS files enables efficient data updates in systems that traditionally support only append operations (Kim et al., 2014).

Large-scale archival systems further illustrate this convergence by combining file storage with database indexing. Such systems manage millions of files while providing efficient retrieval through database queries (Liu et al., 2016).

Additionally, DBMS are increasingly used as computational platforms, where algorithms are expressed as sequences of relational operations. This approach leverages the optimization capabilities of database systems to handle large datasets efficiently (Fichte et al., 2020).

9. Conclusion

The constructs for creating and manipulating records and files in modern systems reflect a layered and evolving architecture. DBMS provide powerful declarative mechanisms for defining and manipulating data, while programming languages enable flexible interaction and computation. At the same time, physical storage mechanisms ensure efficient data organization and access.

Emerging architectures demonstrate that records and files are no longer separate paradigms but complementary components of a unified data management framework. Records provide semantic structure and query capabilities, while files provide scalable storage. The future of data systems lies in the seamless integration of these layers, driven by the need to manage increasingly complex and large-scale data environments.

References

1. Abiteboul, S., Hull, R., & Vianu, V. (1995). *Foundations of databases*. Addison-Wesley. https://doi.org/10.5555/551350
2. Codd, E. F. (1970). A relational model of data for large shared data banks. *Communications of the ACM, 13*(6), 377–387. https://doi.org/10.1145/362384.362685
3. Elmasri, R., & Navathe, S. B. (2016). *Fundamentals of database systems* (7th ed.). Pearson. https://doi.org/10.5555/554501
4. Fichte, J. K., Hecher, M., Thier, P., & Woltran, S. (2020). Exploiting database management systems and treewidth for counting. *arXiv*. https://doi.org/10.48550/arXiv.2001.04191
5. Gray, J., & Reuter, A. (1992). *Transaction processing: Concepts and techniques*. Morgan Kaufmann. https://doi.org/10.5555/573304
6. Kim, J.-S., Whang, K.-Y., Kwon, H.-Y., & Song, I.-Y. (2014). Odysseus/DFS: Integration of DBMS and distributed file system for transaction processing of big data. *arXiv*. https://doi.org/10.48550/arXiv.1406.0435
7. Liu, Y., Wang, F., Ji, K., Deng, H., Dai, W., & Liang, B. (2016). NVST data archiving system based on Fastbit NoSQL database. *arXiv*. https://doi.org/10.48550/arXiv.1612.07587
8. Melton, J., & Simon, A. R. (2001). *SQL: 1999 understanding relational language components*. Morgan Kaufmann. https://doi.org/10.5555/377430
9. Patni, J. C. (2022). *A comprehensive study of SQL: Practice and implementation*. CRC Press. https://doi.org/10.1201/9781003324690
10. Silberschatz, A., Korth, H. F., & Sudarshan, S. (2019). *Database system concepts* (7th ed.). McGraw-Hill. https://www.db-book.com
11. Date, C. J. (2004). *An introduction to database systems* (8th ed.). Addison-Wesley. https://doi.org/10.5555/861613