Deep Science Publishing, 2025
https://doi.org/10.70593/978-93-7185-191-6_1

Chapter 1: Architectures in relational databases: An analytical study of SQL-based data models and ACID principles

Mohanraju Muppala

1. Introduction to Relational Databases

When implemented according to the model proposed by E. F. Codd in 1970, databases and their management systems are said to be relational, hence the term relational model. Codd showed that relational databases have significant advantages over previous hierarchical and network models. Relational modelling is the foundation of relational databases [1-3]. It provides the basis for a high-level data language that can be used both by end users and by application developers, and also supports the design of user views. The relational model is the basis of the Structured Query Language (SQL), the universal standard today used to query, manipulate, and maintain a relational database [2,4].

Routines operate on tables, or relations, as a set, or all at once. The indexes play the role of gathering related rows in a specific order along with tonguing and combining column values. Initially a product of the academic community (e.g., IBM System R and the INGRES project), relational databases quickly became a predominant mainstay of the information industry. Commercial products were supplied by Oracle, Sybase, Informix Systems, Ingres Corporation, IBM, and other companies. Now, with the large number of different database products to choose from, users can select the one that best fits their requirements, regardless of company size.

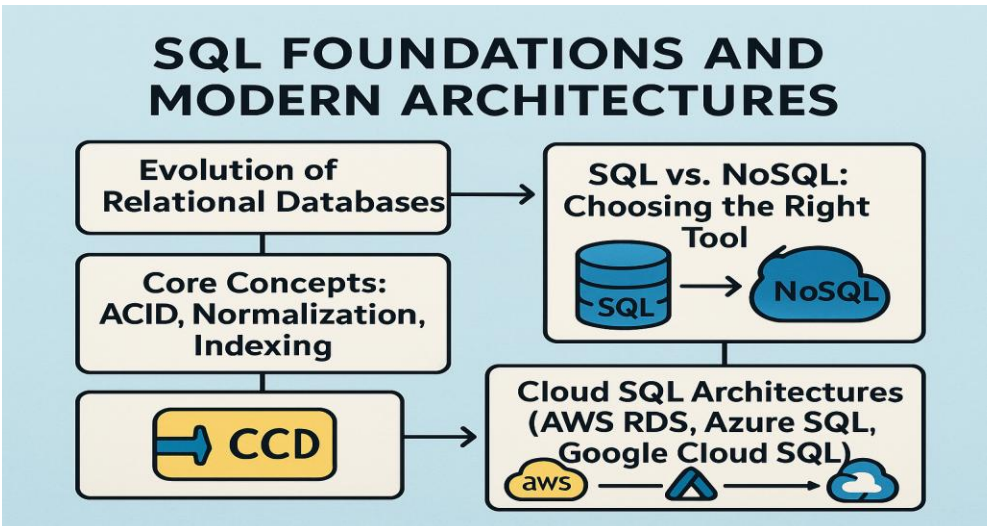
SQL FOUNDATIONS AND MODERN ARCHITECTURES
Evolution of Relational Databases
SQL vs. NoSQL: Choosing the Right Tool
SQL
NoSQL
Core Concepts: ACID, Normalization, Indexing
CCD
Cloud SQL Architectures (AWS RDS, Azure SQL, Google Cloud SQL)
aws

Fig 1. SQL Foundations and Modern Architectures

2. Core Concepts of Relational Databases

A relational database is a collection of data items organized as a set of formally described tables from which data can be accessed or reassembled in many different ways without having to reorganize the database tables [5-8]. The ANSI/SPARC Architecture for a Relational Database shows three database levels — external, conceptual, and internal — and three database schemas that correspond to these levels. Database schemas are normally stored in a data dictionary. The conceptual level deals with logical structure for the whole database. The external level interacts with users through application programs. The internal level makes physical storage decisions.

2.1. ACID Properties

ACID properties (Atomicity, Consistency, Isolation, and Durability) are a key concept in SQL-compliant relational database systems. They ensure that the database remains consistent when handling transactions, concurrent access, and machine failures [6,9]. They form the foundation of relational database systems and have shaped their design and implementation.

The ACID properties ensure that a transaction is treated as a single logical operation that either completes entirely or fails entirely (Atomicity). The database must always be consistent, going from one valid state to another (Consistency). Transactions do not interfere with each other (Isolation).

Committed transactions are permanent, even in the presence of failures (Durability) [10-12]. Although ACID properties do not appear directly in the SQL language, all SQL-compliant relational database management systems (RDBMSs) have some mechanism to support them.

2.2. Normalization Techniques

Database normalization is the process of efficiently organizing data in a database. The main aim of normalization is to add, delete, or modify fields that can be made in a single table. It also removes redundant data as much as possible to allow referring to that data through the relationship. Database normalization splits the attributes of a relation into smaller relations to achieve data integrity and reduce data redundancy.

Normalization of the relational database occurred in the 1970s as an extension of E. F. Codd's relational model. Normalization is usually accomplished by determining a set of functional dependencies that describe the attribute behavior within a relation. Then, based on good heuristic rules, the relations involved in these dependencies can be decomposed into smaller and well-structured relations. The attribute condition that determines the normal form of the relation depends on some of the FDs that the relation satisfies.

Although the normalization aims to minimize redundancy by splitting relations, the retrieval of the relevant data would require joining of most of the relations. The relation produced by joining can be much larger than the base relations because of the properties of the join operator. Join is an expensive operation that uses a lot of resources such as CPU, memory, and network. Hence, an important measure of a relational database scheme is its ability to reduce the number of join operations required for the most common retrieval transactions.

2.3. Indexing Strategies

The ensuing quest for speed and simplicity led to the consolidation of the relational model and the SQL language as the industry standard. Both emerged in the early 1970s, shortly after E. F. Codd published his seminal paper on the relational model (Codd, 1970). However, at first, not everyone accepted the relevance and feasibility of the relational model. Edgar F. Codd himself, who was absolutely convinced of its merits, recognized this situation and took proactive steps to encourage its adoption.

In 1973, Codd published a feature list of functions for a relational database system. It included much of the functionality found today, albeit without any attempt to prioritize or even group the items. The list served as a reference to help developers evaluate their RDBMS implementations against a common framework. Meanwhile, recognition of the model's commercial potential gradually took hold in the industry. Relational database management systems offer tremendous advantages over many alternative databases, such as hierarchical and network databases. These advantages include data independence, built-in integrity constraints, powerful query faculties, and simplicity.

3. SQL vs. NoSQL: Choosing the Right Tool

The relational model proposed by Edgar Codd gained popularity through the creation of Structured Query Language (SQL) by Donald D. Chamberlin and Raymond F. Boyce in the early 1970s. The first implementation of the relational model was made by IBM. Relational databases employ a tabular structure, with each column referring to a named attribute and each row establishing a relationship between the attributes [7,13-16]. Data elements are required to be of a given data type, and values are required to be of the proper type for a given attribute; the database schema is designed and outlined before data can be entered. Each row is given a primary key that must contain a unique value for each row of the table, and rows can also be given a foreign key, referencing the primary key of another table within the database. SQL is used to create and query data elements and establish relationships between them. Transactions are organized into tables to provide data independence and the physical representation of the data.

The emergence of services offered through the Internet led to the development of web applications and database systems. The novelty of these applications was that the data exchanged belonged to the users but was stored on the service providers' servers; these applications therefore must be scalable. Systems cannot be easily modeled through the relational model, especially those systems that establish relationships of arbitrary levels between records, such as Internet telephony and Instant Messaging. Key-column value stores, graphic databases, and distributed databases have been developed to model such relationships more easily, providing better data representation and consequently more efficient processing. As a result, the SQL model has been abandoned for many applications; new database systems are usually referred to as NoSQL to differentiate themselves from traditional relational databases.

3.1. Understanding SQL

Structured Query Language (SQL, sequel) is a domain-specific language used in programming and designed for managing data held in a relational database management system (RDBMS), or for stream processing in a relational data stream management system (RDSMS) [2,17-19]. It is particularly useful for handling structured data, where there are relations between different entities/variables of the data. SQL offers two main advantages over older read–write APIs, such as ISAM or VSAM. First, it introduced the concept of accessing many records with one single command. Second, it eliminates the need to specify how to reach a record, e.g., with or without an index.

Despite being very successful language, SQL is often criticized for both some of its early design choices and its treatment of inconsistencies in the relational model [3,20-23]. It has been labeled a "non-comprehensive" and "quirky" language. As a database language, it is both an ANSI and an ISO standard, but neither the standard nor the supported language in commercial and open-source database systems completely follow the relational model. Despite these criticisms, SQL remains the most widely used database language.

3.2. Understanding NoSQL

The rise of online services that handle massive data volumes has emphasized scalability and high availability and de-emphasized atomicity. Relational tables modeled as key–value pairs are the basis for data storage in many NoSQL systems. The most famous of them is Google’s Big table, which was described in a published paper, and the Google File System. The Google File System is a loosely coupled, fault-tolerant cluster of machines that stores huge amounts of data. This white paper was followed by another, describing the Big table system. Big table stores data sorted and indexed by a row key, a column string, and a timestamp.

Many companies have built their own implementations of the concepts defined by Big table, some of them even open sourcing their code. Amazon’s Dynamo is a key–value pair database that emphasizes horizontal scalability, fault tolerance, some data availability, and eventual consistency. Dynamo was described in a published paper, and many implementations exist. Facebook developed Cassandra, which is based on ideas from both Big table and Dynamo, and released it as open-source software. It is designed to handle large amounts of data across many servers and provides high availability with no single point of failure.

3.3. Use Cases for SQL

A wide array of data-management problems can be addressed with the SQL language and its relational-database environment [9,24-26]. Designers of information systems use SQL when modeling the real world of their domain area as a collection of relations. Web developers use forms and CGI script to build transactional Web services on top of a database engine. For example, a Web site might allow someone to query the name and location of a restaurant throughout the world or set up a personal calendar that reminds the customer of airline flights and visits to the dentist. Businesses rely on SQL databases to store information about employees, customers, and the production process. SQL is also the preferred tool for analyzing business growth and market-share trends. Data miners rely on SQL to query vast amounts of information to guide strategies in sales, production, and other domains. With the advent of software as a service, a customer can rent access to a database engine on the Internet rather than installing the database software.

Many of today's telephone services—such as voice mail, call waiting, call forwarding, and 800-number routing—rely on databases that are frequently updated and accessed in real time [27-29]. Document indexes that track keyword entries in a supercomputer cluster require a high level of concurrency. Warehouse managers profit from hardware components designed to fit together like Legos, added easily as the need for more data storage arises. Database designers have also found the concept of views to be quite effective: Here, views serve like virtual tables often that can be based on the instruction style as well as physical hierarchy of the queries need derived from one or more base tables. This technology is very useful in building a modern Web-service architecture.

3.4. Use Cases for NoSQL

Relational databases have been the dominant form of databases for more than 40 years. Structured Query Language (SQL) is the most widely used programming language for manipulating them. Amazon's Relational Database Service (RDS) supports six popular commercial and open source databases: Microsoft SQL Server, Oracle, MySQL, MariaDB, PostgreSQL, and Amazon Aurora. Amazon Aurora, a new open source database, is more cost-efficient and delivers performance comparable to high end commercial databases. On the other hand, several prototypes of different commercial NoSQL databases have been implemented recently, California Modernized Automated Delivery System (CalMars) being one example. In their implementation, data were stored in Accumulo, a key-value pair store.

Relational databases have been used largely in individual business applications that must handle various sales and distribution transaction data. These databases are required to enforce the ACID properties: Atomicity, Consistency, Isolation, and Durability. However, supporting applications such as Web 2.0 social networking, building cyber-physical networks, and the Internet of Things (IoT) require very large datasets with ultra-high- availability and disaster-recovery capabilities, as well as the ability to provide fast response and answer ad hoc queries. Both RDBMS and NoSQL, the two major classes of database management systems today, fail in one or more of these requirements. Specifically, an RDBMS is very expensive in storage, fails to provide very high availability in the face of disasters, and answers ad hoc queries slowly.

3.5. Comparison of SQL and NoSQL

The short summary was mentioned above: SQL enforces a rigid schema, while NoSQL databases allow for a flexible schema and support horizontal scaling. The list can certainly be expanded, using information from Crosman [196] and applicants who make such a comparison, for example, Hossain et al. [197]. The write operations are always served by a single node in an ideal "SQL environment," which uses a B-Tree structure to index the records of a table. Index creation and shard key design should be well thought out, because as soon as the responsibility of serving writes has been shared by multiple nodes, the implementation steps to allow growth can be very complex. Vertical scaling is used to store more data on a single server, but the impacts of the write lock need to be considered. This lock prevents more than one write operations from being served concurrently. Based on the data required to be stored in the database, the design of the schema also requires detailed attention. Imposing a rigid schema logic does not allow any room for growth without a significant schema alteration.

NoSQL databases that support horizontal growth are called "distributed," which means that each insert operation is served by the assigned node and the responsibility of the write serving grows with the number of nodes [30-32]. These systems address the write lock restriction that currently exists with SQL. A flexible schema allows anything to be inserted in any collection at any time. Horizontal scaling also requires data duplication among the nodes, and this replication process yields the challenges of consistent read and delayed update. Consistent read means reading the same copy that was updated, and delayed update means that the modification operation has not yet propagated to all nodes. These two situations may be fulfilled by either the CAP theorem or the

BASE properties. All of these properties need to be balanced when designing the system.

4. Cloud SQL Architectures

Cloud SQL refers to the implementation of SQL relational databases within public cloud environments. Offering services such as Amazon RDS, Azure SQL Database, and Google Cloud SQL, cloud-hosted database engines build upon traditional foundations while addressing the demands of modern use cases. This exposure reveals the diversity of SQL database implementations and their key roles in database management systems.

Amazon RDS (Relational Database Service) supports several database engines, including MySQL, MariaDB, Oracle, Microsoft SQL Server, PostgreSQL, and the proprietary Amazon Aurora. The service combines multiple versions and editions in a fully managed, multi-AZ deployment. Azure SQL Database, a modular cloud-service based on Microsoft SQL Server, offers both single and pooled databases within elastic database pools. Meanwhile, Google Cloud SQL provides a fully managed service for relational databases on the Google Cloud Platform, currently encompassing MySQL and PostgreSQL.

4.1. AWS RDS Overview

Amazon Web Services (AWS) is a leading cloud computing platform that offers a wide range of computing services including infrastructure as a service (IaaS), platform as a service (PaaS), and Software as a Service (SaaS). Offering more than 80 services, AWS boasts a client list including Netflix, Siemens, Samsung, NASA, Hearst Corporation, 21st Century Fox, Turner, British Airways, Comcast, Virgin Atlantic, and more. These services serve all kinds of businesses, from government agencies to startups. The AWS cloud offers flexible, reliable, scalable, easy to use, and cost-effective cloud computing services.

Amazon Relational Database Service (RDS) is part of Amazon’s cloud computing platform. It is a distributed relational database service introduced in 2009. It is a web service that makes it easier to set up, operate, and scale a relational database in the cloud. It provides cost-efficient and resizable capacity while managing time-consuming database administration tasks such as hardware provisioning, database setup, patching, and backups. It fosters greater than 99.95% availability for Multi-AZ database instances. Its purpose is to simplify the provisioning, operation, and scalability of a relational database for use in applications.

4.2. Azure SQL Overview

Microsoft Azure SQL is a family of Azure database services related to SQL. It includes the Azure SQL Database managed database service, Azure Managed Instance managed instance service, and SQL Server Stretch Database service.

Azure SQL Database is a managed cloud database provided as a service offering of Microsoft Azure. It is based on the Microsoft SQL Server Engine, provided as a service that abstracts and handles most database management functions such as upgrading, patching, backups, and monitoring without user involvement. Microsoft manages all aspects of a SQL Database except data and objects stored in the database. Users are free to create databases on the SQL Database server as needed, and two service tiers for each database – Basic and Standard – provide different levels of cost and performance.

4.3. Google Cloud SQL Overview

In the realm of cloud SQL architectures, Google Cloud SQL stands out by delivering fully managed MySQL, PostgreSQL, and SQL Server databases in the cloud. Cloud SQL handles replication, patch management, and failover, minimizing operational overhead while ensuring rapid response times for both OLTP and OLAP workloads. It can be used as the primary data repository for moving an on-premises relational database to the cloud or as a backend for modern serverless and mobile applications. The fully managed nature of Cloud SQL allows developers to focus on building applications rather than managing databases.

All three relational database engines are available on Google Cloud Platform as Cloud SQL instances. Google Cloud SQL offers enterprise-grade networking and security features such as private IP, customer-managed encryption keys, and database flags. The platform balances scalability and high availability, enabling organizations to scale their infrastructure seamlessly without downtime. Although Cloud SQL primarily caters to OLTP workloads, it can also support OLAP queries, offering a degree of versatility for various analytical tasks.

4.4. Comparative Analysis of Cloud SQL Solutions

Relational databases are engineered to ensure the ACID properties, a feature lacking in NoSQL implementations. Normalization techniques guarantee that data conforms to a defined schema, thereby avoiding duplication and enhancing storage efficiency—attributes not prioritized by NoSQL databases. Moreover, indexing capabilities facilitate rapid record searches, a function not available in NoSQL architectures.

Cloud SQL offerings from the three major cloud providers—AWS RDS for PostgreSQL, Azure SQL Database, and Google Cloud SQL—exemplify how foundational principles are implemented within new database architectures. The following table summarizes the key features of these managed relational database services to provide a clearer comparative perspective.

5. Future Trends in Relational Databases

Relational databases represent the longest-lived of all modern software constructs. For many years advancing hardware capabilities, especially improvements in processor speed, disk speed, and disk capacity, outpaced the capabilities of their software. The performance delivered by these hardware improvements seemed to mask the presence of software limitations, making all that more important the design of the underlying data structures. As a result, architects of database systems could focus on the logical issues of data management without worrying excessively about the physical aspects. The past decade, however, has seen a reversal of that balance of power. One can no longer assume that hardware advances are sufficient. That shift has made database architects reexamine fundamental algorithmic and data-structure design choices.

The project described here supports the movement toward a new architecture for a relational database system. Its goal is twofold. First, it focuses on the design of indexing mechanisms that are tuned for modern processors. Second, the overarching organization of the component algorithms is reconsidered. A novel implementation strategy is proposed that decouples the architecture of the algorithms from the schema of the relations on which the operations are being performed. This decoupling enables an array of additional optimization techniques capable, in turn, of delivering increased performance.

5.1. Emerging Technologies

The idea of tables also emerged early in the history of computer science, notably in the creation of Charles Bachman's Integrated Data Store, albeit without the rigor of a formal mathematical foundation. Relational Models were proposed independently by Hugh Darwen, who credits Chris Date with many of the formal foundations. Moreover, tables supported by Codd’s Relational Model have become a standard component of the overwhelming majority of existing mission-critical operational and analytical data management systems.

One reason for this success is the power of the Structured Query Language (SQL) in instructing the DBMS to perform at-scale data transformations. These queries often take the form of database views, which render the data in a different way to the user, without physically regenerating the data [9,33-34]. They are widely used across several “user-facing” relational systems for a variety of usage scenarios such as: (i) security and access control; (ii) simplification of queries or derived attribute calculation; and (iii) materialization of complex queries for performance reasons. However, modern DBMS architectures do not support permanent, shared, splitting of query definitions.

5.2. Impact of AI on Databases

The concept of relational databases was first proposed by Edgar Codd in a 1969 paper published when he was still working at IBM. Although IBM was remarkably fast in implementing the concept, it took a while for the industry and users to realize the importance of the technology in evolving the database architecture. Today, most databases use what is known as a structured query language, or SQL. SQL is simply used to interact with the relational database by writing queries such as "SELECT," "UPDATE," or "DELETE." These commands in SQL can fetch data from multiple tables in the database and organize it in the desired format.

The term curriculum vitae (CV) is Latin for "life course," which distills the purpose of a CV: it's a comprehensive record of one's professional career and achievements. It offers an extensive review of professional experience gathered over many years and serves as background information for a comprehensive job application, often necessitating adaptations for specific positions or institutions. Therefore, alongside education and training, the CV should highlight the most relevant skills and knowledge pertinent to the applied position, laying the foundation for the forthcoming letter of motivation. As the Artificial Intelligence (AI) revolution clearly demonstrates, AI cannot fully replace human beings. Despite the fact that ChatGPT produces perfect SQL queries, the developer must possess sufficient knowledge and skills to review the generated query adequately.

5.3. Scalability Challenges

With the increasing size and importance of databases, scalability started to pose problems. Oracle 6 supported binding a SQL statement to a cursor so that it was precompiled on the database server before execution and could be parsed, and optimized only once, and made it possible for a program to open multiple cursors. These features were part of Oracle's application program interface called Pro*C; later, similar APIs, called Embedded SQL, were also supported by other database companies. Oracle 7 added bind variables, which are variables bound to the SELECT-LIST and WHERE-LOCATION of a cursor.

A database is scalable if an increase in data volume or in concurrent users does not cause an equal degradation in performance. Lack of scalability can be caused by any number of factors. Compaq, Intel, and Microsoft designed a database optimized for warehouses of all sizes, ultimately handling multi-terabyte–multi-user warehouses, running on affordable industry-standard platforms. When implementing scalability, they looked at factors such as the following: efficiency of basic SQL and Transaction processing, archive and backup operations while the system is still running, auto reorganization, sharing of I/O resources for multiple database operations, the automatic use of new CPUs when added, and the ability to spread the table data across multiple disks.

6. Case Studies in Database Evolution

Relational or SQL database engines compile query statements into an execution plan made up of a tree of operators. Each operator accepts one or more input tables and produces an output table to be pushed to the next operator. Rows within the table can be processed in any order as long as the final answer is faithfully produced. The execution strategy can be either iterator mode or batched mode—the latter often leads to better instruction cache locality, can exploit SIMD instructions, and leads to improved CPU register locality. Relational approaches have also been explored in functional programming languages, and the same idea applies to any list-processing language such as LINQ.

Despite the ORM hype around GraphQL, REST, and NoSQL, the industry is slowly adopting a near-standard meta-framework for describing and querying software metadata. Traditional complex domain objects are being broken down into flat relational structures, with data access described in a REST datamodel. Functions are added to the datamodel, with batch and endless mode streaming queries expressed as metaSQL. The table/functions scheme aligns naturally with ORM and has already been implemented for at least Oracle, MSSQL, Greenplum, and Snowflake—any of which can be used as the meta-model repository.

6.1. Successful SQL Implementations

SQL's success can be measured by the ease with which a new relational DBMS can incorporate the language and product-oriented features into existing products or products being developed. One of the reasons for the success of SQL is the commitment of vendors and users to the language. Users demanded it and vendors eventually recognized the wisdom and agreed to support it. It was a laborious process that eventually proved productive.

The SQL language was first implemented by Oracle in 1979; IBM then supported SQL after the language became a formal standard. Most other vendors have followed suit and now offer SQL support of some kind. IBM developed a product, System R, to support SQL and other features and used that implementation as the basis for future development of DB2. Other implementations have also been created with different architectures for DBMSs for both relational and nonrelational databases. Oracle, Ingres, and Sybase are full function commercial relational DBMSs. DB2 is the first commercial relational DBMS for mainframes; for a mainframe relational DBMS, SQL is essential. The SQL/DS product is designed to run on a lower-end mainframe.

6.2. Successful NoSQL Implementations

The rise in relational database sizes and demand for quicker responses led to the distribution of databases. While relational databases can be distributed, there are limitations. The relational model's properties restrict database distribution. For instance, relations don't define the distribution structure, and relations don't guarantee that records in tables of the same database will be physically stored in the same machine, so operations defined in the relational algebra are not optimized for the distribution aspect. Throughout history, most databases used in large multinational companies have been relational due to the robustness and maturity of the model. The big drawback for relational models at that time was the following. In relational databases, a lot of calculations were done internally. On the other hand, in the early stages of the internet, databases were growing and shrinking in very short amounts of time. This was a big drawback, so experts began thinking about ways to speed up database processing.

The first NoSQL database was called Change Set, which was developed by Carlo Strozzi in 1998. Subsequently, in 2003, Amazon developed the Dynamo database, focusing on availability and partition tolerance. A year later, Google developed BigTable. Also, in 2008, Google released Google GFS. In the same year, Facebook started development of the Cassandra database in C++ and Java. After these developments, many new databases emerged, supporting various storage types such as Document, File, Graph, and Key-Value.

6.3. Lessons Learned from Failures

SQL was designed in the 1970s by personal preferences of its designers and the current trends in Artificial Intelligence research, that suggested the usage of predicates for querying and rules for inferencing. There are very serious reasons why the creators of the Relational Model did not use logic for querying or inference.

Consider for example the query Which suppliers supply some red part? The English words some nowhere appear in the relational query. It is a recurring pattern in Relational Algebra that seemingly simple English statements have to be specified in a much lengthier, more complicated form. This is because the formalism of first-order predicate logic satisfies the condition that every sub-formula with free variables correspond to one defined relation. Indeed, it requires that every well-formed formula contain no free variables: either it corresponds to truth values, or it is syntactically ill-formed. Also, it cannot express the operation of closure that rules in deductive reasoning require, and that allows—for example—queries such as Which suppliers supply directly or indirectly every part manufactured in London.

Contrast the formalism of SQL with the natural expression of the same query English. If SS is the supply relation and SP is the ship relation

select S.sname from SS as S where exists(select * from SP as T where T.sno = S.sno and color = 'red')

no existing apples of the same variety. Watermelons have a black interior color. All existing apricots have a yellow skin. Eight varieties of apples have the same color in the peel and in the inside. No sandy apple exists.

7. Best Practices for Database Management

Every database tends to develop a certain style or approach, in particular because of the rapidly increasing number of features supported by modern database systems and the demands of different application areas. However, some basic principles still hold for all systems.

Normalizing the information stored in a relational database—each item of information should be stored in precisely the right place—preventing multiple potential sources for updates and inconsistencies introduced as a result. The objective should be to try to ensure that every piece of information in the database is in exactly one place and that every fact stored in the database has its own unique place. Database normalization, a process that is supported by a comprehensive set of theoretical results, fulfills these requirements.

Use SQL programming within the database rather than manual programming outside it—this conserves the load on the communication system and makes use of the indexing available within the system. It also tends to make the final system easier to maintain and enhances the security of processing.

7.1. Performance Optimization

Database Management Systems (DBMSs) and Database Applications have existed for almost half a century, yet performance optimization remains an indispensable topic in the field [35-38]. The first relational DBMS based on SQL, System R, was developed at IBM in the 1970s. The implementation of System R took approximately five years and involved about fifty people. SQL is a declarative language, akin to logic programming: the user formulates a query but does not specify an execution method.

SQL queries undergo transformation, optimization, and compilation into execution plans that access the base tables, producing the desired query answers. Today, DBMSs are omnipresent, catering to government agencies, banks, insurance companies, manufacturing firms, among others. Modern DBMSs accommodate thousands of concurrent users, achieving response times measured in seconds and transaction throughputs in the tens of thousands per second. Such performance is attained through a confluence of factors, including Moore's Law, High-Performance Computing, High-Performance Architectures, High-Performance Software Systems, and High-Performance Algorithms.

7.2. Security Considerations

SQL fulfills the access-control requirements of relational databases through three types of security mechanisms: (i) to regulate the visibility of relations (and of columns of a relation), SQL provides views, authorization ratings, and a GRANT operation; (ii) triggers allow the database manager to respond to events; and (iii) integrity constraints enforce conditions on data that SQL's type system cannot specify.

In the relational model, a table's name is meaningful and therefore must be unique. Different users can be permitted to see separate subsets of the columns of a relation. The combination of these two guarantees denies unauthorized users any guess as to the semantics of the columns they cannot see. In SQL, these objectives are accomplished by views: a view is a standing query, stored as a virtual table. Like all queries, it has a name, and its columns can be named. Authorization ratings are associated with views, as are the base tables. The GRANT/DENY statement determines the rating.

7.3. Backup and Recovery Strategies

A backup system protects a database against media failure. If the storage medium of a database becomes faulty, the data stored on the medium is lost. The backup database is a copy of the database that is written to a different storage medium, so that the backup information survives the media failure. The database generally uses an off-site copy of the backup to recover the original database. The basic backup procedure involves making a copy of the database which is transferred to a backup location. In addition to the backup of database files, the database needs to make backups of logs. The log records the modification operations that were performed on the database. When a database is recovered, the modifications are again applied to the data, until it reaches the state before the failure.

Recovery of a database is a procedure that reconstructs the database. After a database manager detects a deadlock, and rolls back one or more transactions, it recovers the database. Recovery is also necessary after a system failure or media failure. Often one of several processes of a transaction updates the database so that changes made by the earlier steps of the transaction are written to the database. If the transaction fails in between, the changes may persist in the database. Recovery resets such changes made by incomplete transactions and restores the database to a consistent state. Recovery ensures Atomicity and Durability.

8. Conclusion

The stable nature of the relational model contributes significantly to the success of relational database management systems. Users can continue to issue the same SQL queries issued many years ago; however, physical relational database management systems can and do change, adapting to technology changes and usage scenarios.

Object-relational features have experienced little adoption due to an inadequate match with application-typical object-oriented programming languages and styles. Relational extensions of the kind represented by database web integration and OLAP (on-line analytic processing) received much more attention. OLAP has matured into an interesting subfield of research, with dedicated DBMS products and work on SQL extensions.

References:

[1] Ionescu SA, Diaconita V, Radu AO. Engineering Sustainable Data Architectures for Modern Financial Institutions. Electronics. 2025 Apr 19;14(8):1650.

[2] Anshelevich D, Katz G. Synthetic tabular data generation using a VAE-GAN architecture. Knowledge-Based Systems. 2025 Jul 10:113997.

[3] Khan W, Kumar T, Zhang C, Raj K, Roy AM, Luo B. SQL and NoSQL database software architecture performance analysis and assessments—a systematic literature review. Big Data and Cognitive Computing. 2023 May 12;7(2):97.

[4] Hong Z, Yuan Z, Zhang Q, Chen H, Dong J, Huang F, Huang X. Next-generation database interfaces: A survey of llm-based text-to-sql. arXiv preprint arXiv:2406.08426. 2024 Jun 12.

[5] Choi H, Jeong J. Domain-Specific Manufacturing Analytics Framework: An Integrated Architecture with Retrieval-Augmented Generation and Ollama-Based Models for Manufacturing Execution Systems Environments. Processes. 2025 Feb 27;13(3):670.

[6] Islam S. Future trends in SQL databases and big data analytics: Impact of machine learning and artificial intelligence. Available at SSRN 5064781. 2024 Aug 6.

[7] de Oliveira VF, Pessoa MA, Junqueira F, Miyagi PE. SQL and NoSQL Databases in the Context of Industry 4.0. Machines. 2021 Dec 27;10(1):20.

[8] Rockoff L. The language of SQL. Addison-Wesley Professional; 2021 Nov 4.

[9] Shivadekar S, Halem M, Yeah Y, Vibhute S. Edge AI cosmos blockchain distributed network for precise ablh detection. Multimedia tools and applications. 2024 Aug;83(27):69083-109.

[10] Fotache M, Munteanu A, Strîmbei C, Hrubaru I. Framework for the assessment of data masking performance penalties in SQL database servers. Case Study: Oracle. IEEE Access. 2023 Feb 22;11:18520-41.

[11] Panda SP. Augmented and Virtual Reality in Intelligent Systems. Available at SSRN. 2021 Apr 16.

[12] Karwin B. SQL Antipatterns, Volume 1: Avoiding the Pitfalls of Database Programming. The Pragmatic Programmers LLC; 2022 Oct 24.

[13] Nasereddin M, ALKhamaiseh A, Qasaimeh M, Al-Qassas R. A systematic review of detection and prevention techniques of SQL injection attacks. Information Security Journal: A Global Perspective. 2023 Jul 4;32(4):252-65.

[14] Chakraborty S, Aithal PS. CRUD Operation on WordPress Database Using C# SQL Client. International Journal of Case Studies in Business, IT, and Education (IJCSBE). 2023 Nov 28;7(4):138-49.

[15] Panda SP. The Evolution and Defense Against Social Engineering and Phishing Attacks. International Journal of Science and Research (IJSR). 2025 Jan 1.

[16] Choi H, Lee S, Jeong D. Forensic recovery of SQL server database: Practical approach. IEEE Access. 2021 Jan 18;9:14564-75.

[17] Thalji N, Raza A, Islam MS, Samee NA, Jamjoom MM. Ae-net: Novel autoencoder-based deep features for sql injection attack detection. IEEE access. 2023 Nov 28;11:135507-16.

[18] Chakraborty S, Paul S, Hasan KA. Performance comparison for data retrieval from nosql and sql databases: a case study for covid-19 genome sequence dataset. In2021 2nd International Conference on Robotics, electrical and signal processing techniques (ICREST) 2021 Jan 5 (pp. 324-328). IEEE.

[19] Crespo-Martínez IS, Campazas-Vega A, Guerrero-Higueras ÁM, Riego-DelCastillo V, Álvarez-Aparicio C, Fernández-Llamas C. SQL injection attack detection in network flow data. Computers & Security. 2023 Apr 1;127:103093.

[20] Antas J, Rocha Silva R, Bernardino J. Assessment of SQL and NoSQL systems to store and mine COVID-19 data. Computers. 2022 Feb 21;11(2):29.

[21] Shivadekar S, Kataria DB, Hundekar S, Wanjale K, Balpande VP, Suryawanshi R. Deep learning based image classification of lungs radiography for detecting covid-19 using a deep cnn and resnet 50. International Journal of Intelligent Systems and Applications in Engineering. 2023;11:241-50.

[22] Ashlam AA, Badii A, Stahl F. Multi-phase algorithmic framework to prevent SQL injection attacks using improved machine learning and deep learning to enhance database security in real-time. In2022 15th International Conference on Security of Information and Networks (SIN) 2022 Nov 11 (pp. 01-04). IEEE.

[23] Tanimura C. SQL for Data Analysis: Advanced Techniques for Transforming Data Into Insights. " O'Reilly Media, Inc."; 2021 Sep 9.

[24] Brunner U, Stockinger K. Valuenet: A natural language-to-sql system that learns from database information. In2021 IEEE 37th International Conference on Data Engineering (ICDE) 2021 Apr 19 (pp. 2177-2182). IEEE.

[25] Panda SP. Relational, NoSQL, and Artificial Intelligence-Integrated Database Architectures: Foundations, Cloud Platforms, and Regulatory-Compliant Systems. Deep Science Publishing; 2025 Jun 22.

[26] Lawson JG, Street DA. Detecting dirty data using SQL: Rigorous house insurance case. Journal of Accounting Education. 2021 Jun 1;55:100714.

[27] Zhang B, Ren R, Liu J, Jiang M, Ren J, Li J. SQLPsdem: A proxy-based mechanism towards detecting, locating and preventing second-order SQL injections. IEEE Transactions on Software Engineering. 2024 May 14;50(7):1807-26.

[28] Panda SP. Artificial Intelligence Across Borders: Transforming Industries Through Intelligent Innovation. Deep Science Publishing; 2025 Jun 6.

[29] Gandhi N, Patel J, Sisodiya R, Doshi N, Mishra S. A CNN-BiLSTM based approach for detection of SQL injection attacks. In2021 International conference on computational intelligence and knowledge economy (ICCIKE) 2021 Mar 17 (pp. 378-383). IEEE.

[30] Dhanaraj RK, Ramakrishnan V, Poongodi M, Krishnasamy L, Hamdi M, Kotecha K, Vijayakumar V. Random forest bagging and x-means clustered antipattern detection from SQL query log for accessing secure mobile data. Wireless communications and mobile computing. 2021;2021(1):2730246.

[31] Panda SP, Muppala M, Koneti SB. The Contribution of AI in Climate Modeling and Sustainable Decision-Making. Available at SSRN 5283619. 2025 Jun 1.

[32] Shivadekar S. Artificial Intelligence for Cognitive Systems: Deep Learning, Neuro-symbolic Integration, and Human-Centric Intelligence. Deep Science Publishing; 2025 Jun 30.

[33] Davidson L. Pro SQL Server Relational Database Design and Implementation: Best Practices for Scalability and Performance. Apress; 2021.

[34] Zhang W, Li Y, Li X, Shao M, Mi Y, Zhang H, Zhi G. Deep Neural Network-Based SQL Injection Detection Method. Security and Communication Networks. 2022;2022(1):4836289.

[35] Roy P, Kumar R, Rani P. SQL injection attack detection by machine learning classifier. In2022 International conference on applied artificial intelligence and computing (ICAAIC) 2022 May 9 (pp. 394-400). IEEE.

[36] Katsogiannis-Meimarakis G, Koutrika G. A survey on deep learning approaches for text-to-SQL. The VLDB Journal. 2023 Jul;32(4):905-36.

[37] Mo X, Pang J, Wan B, Tang R, Liu H, Jiang S. Multi-relational graph contrastive learning with learnable graph augmentation. Neural Networks. 2025 Jan 1;181:106757.

[38] Birch K, Komljenovic J, Sellar S. Architectures of assetization: Legacy infrastructures and the configuration of datafication in UK higher education. new media & society. 2025 Apr;27(4):1868-87.