### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

**Table Of Contents**

### 

[Emergence of SQL](#_21tsrfy69a3) 3

[1. Who Introduced It?](#_hz9g02yxqcbh) 3

[2. Why Was It Introduced? (The Problem It Solved)](#_keynjzgh2ls9) 3

[3. How Does It Work?](#_3iw34xanq1bu) 4

[Key Components Involved:](#_d31lidk566fu) 5

[4. The Use of SQL](#_3kju46pn2h4s) 5

[5. Why Does It Work Well? (Advantages of SQL and the Relational Model)](#_334mnxwsgr3l) 6

[6. Reasons Why We Are Still Using SQL](#_n4qmc6pro61b) 6

[7. Relevant Aspects for a Software Engineer](#_4zakltyjy30e) 7

[References](#) 9

### 

# 

# 

# 

# **Emergence of SQL**

# **1. Who Introduced It?**

The relational model of data was introduced in 1970 by Dr. Edgar F. Codd, a British computer scientist working at IBM. His groundbreaking paper, “A Relational Model of Data for Large Shared Data Banks”, proposed a new and mathematically sound method for organizing data using relations (tables), which later became the foundation of SQL (Structured Query Language).

# **2. Why Was It Introduced? (The Problem It Solved)**

Before the relational model, data was managed using hierarchical and network database models like IBM's IMS. These systems were:

* Rigid and difficult to scale.
* Dependent on predefined access paths.
* Costly to maintain, especially when data structures change.
* Inefficient for complex queries.

**Codd's relational model solved these problems by:**

* Making data storage more flexible and independent of physical structure.
* Allowing easy querying without navigating complex paths.
* Enabling data independence, so applications didn’t need to be rewritten when database structures changed

# **3. How Does It Work?**

The relational model organizes data into tables (relations). Each table has:

* Columns (Attributes): Define properties or fields.
* Rows (Tuples): Represent individual records.
* Primary Key: Uniquely identifies each row.
* Foreign Key: Links related data across tables.

### **Key Components Involved:**

* Tables: Fundamental units of storage.
* Attributes (Columns): Define characteristics.
* Tuples (Rows): Actual data entries.
* Domains: Set of valid values for a column.
* Primary Keys: Unique identifiers for records.
* Foreign Keys: Create links between tables.

# **4. The Use of SQL**

SQL (Structured Query Language) was developed alongside IBM’s System R project to interact with relational databases. It has two major components:

* Data Definition Language (DDL):
  + CREATE TABLE, ALTER TABLE, DROP TABLE  
     Used to define or modify the database structure.
* Data Manipulation Language (DML):
  + SELECT, INSERT, UPDATE, DELETE  
     Used to retrieve or change data.

SQL allows users to perform queries, updates, and schema modifications in a human-readable yet powerful syntax.

# **5. Why Does It Work Well? (Advantages of SQL and the Relational Model)**

* Data Independence: Applications are loosely coupled to the database structure.
* Declarative Syntax: Users focus on *what* to retrieve, not *how* to retrieve it.
* Robust Integrity: Constraints like primary keys, foreign keys, and CHECK maintain valid and consistent data.
* Powerful Querying: SQL supports joins, subqueries, grouping, sorting, and set operations.
* ACID Compliance: Supports Atomicity, Consistency, Isolation, and Durability—essential for transactions.

# **6. Reasons Why We Are Still Using SQL**

* **Standardization:** ANSI and ISO have maintained consistent standards since SQL-86 and SQL-92 (ISO/IEC, 1987).
* **Maturity:** Decades of development have led to stable, reliable, and efficient SQL engines.
* **Compatibility and Portability:** SQL is compatible with various platforms, including MySQL, PostgreSQL, SQL Server, Oracle, and SQLite.
* **Interoperability:** Even many NoSQL platforms (like Google BigQuery or Amazon Redshift) now support SQL or SQL-like syntax (Date, 2004).  
  **Industry Dependence:** SQL powers CRM systems, e-commerce platforms, finance, education systems, mobile apps, and more.

# **7. Relevant Aspects for a Software Engineer**

* **Data modeling**

Software engineers must know how to organize real-world data into tables and define relationships. This involves creating clean database structures and applying normalization. Proper modeling improves data integrity and reduces redundancy.

* **Query Optimization**

Writing efficient SQL helps avoid slow database responses. Engineers should understand indexing, joins, and how to reduce unnecessary queries. Optimization ensures high performance in large-scale systems.

* **Database Design**

Good database design uses keys, constraints, and indexes to maintain structure and speed. Engineers create schemas that reflect real-world rules. This supports scalability and minimizes errors in application development.

* **Transaction Management**
* Transactions group database operations into a reliable unit. Using ACID principles ensures data is consistent and safe even when errors or crashes occur. This is vital in systems like banking or inventory management.
* **Integration**

Engineers connect SQL databases to backend code using raw queries or ORMs like Sequelize or Hibernate. This allows applications to interact with data easily. Integration enables features like login, saving records, or showing reports.

* **Security**

Engineers must protect databases from threats like SQL injection. Using secure queries, roles, and access control keeps user data safe. A secure database is crucial for maintaining trust and ensuring compliance.

**7. Backup, Migration, and Recovery**

It’s important to back up data regularly and manage changes with migration tools. This ensures data is not lost and updates can be controlled. Engineers must also know how to recover from failures.

# **References**

Codd, E. F. (1970). *A relational model of data for large shared databases*. Communications of the ACM, 13(6), 377–387. https://doi.org/10.1145/362384.362685

Date, C. J. (2004). *An Introduction to Database Systems* (8th ed.). Addison-Wesley.

Elmasri, R., & Navathe, S. B. (2016). *Fundamentals of Database Systems* (7th ed.). Pearson.

O’Neil, P., & O’Neil, E. (2014). *Database: Principles, Programming, and Performance* (2nd ed.). Morgan Kaufmann.

Ramakrishnan, R., & Gehrke, J. (2003). *Database Management Systems* (3rd ed.). McGraw-Hill.

Stonebraker, M., & Hellerstein, J. M. (2005). *What Goes Around Comes Around*. In J. Gray & A. Reuter (Eds.), *Readings in Database Systems* (4th ed.). MIT Press.

ISO/IEC. (1987). *Information technology – Database language – SQL (ISO/IEC 9075:1987)*. International Organization for Standardization.

Ben-Gan, I. (2016). *T-SQL Fundamentals* (3rd ed.). Microsoft Press.

Oracle. (n.d.). *Security Best Practices for Database Developers*. Retrieved from<https://www.oracle.com/database/technologies/security.html>

IBM.. (n.d.). *SQL Injection: Understanding and Prevention*. IBM Knowledge Center. https://www.ibm.com/docs/en