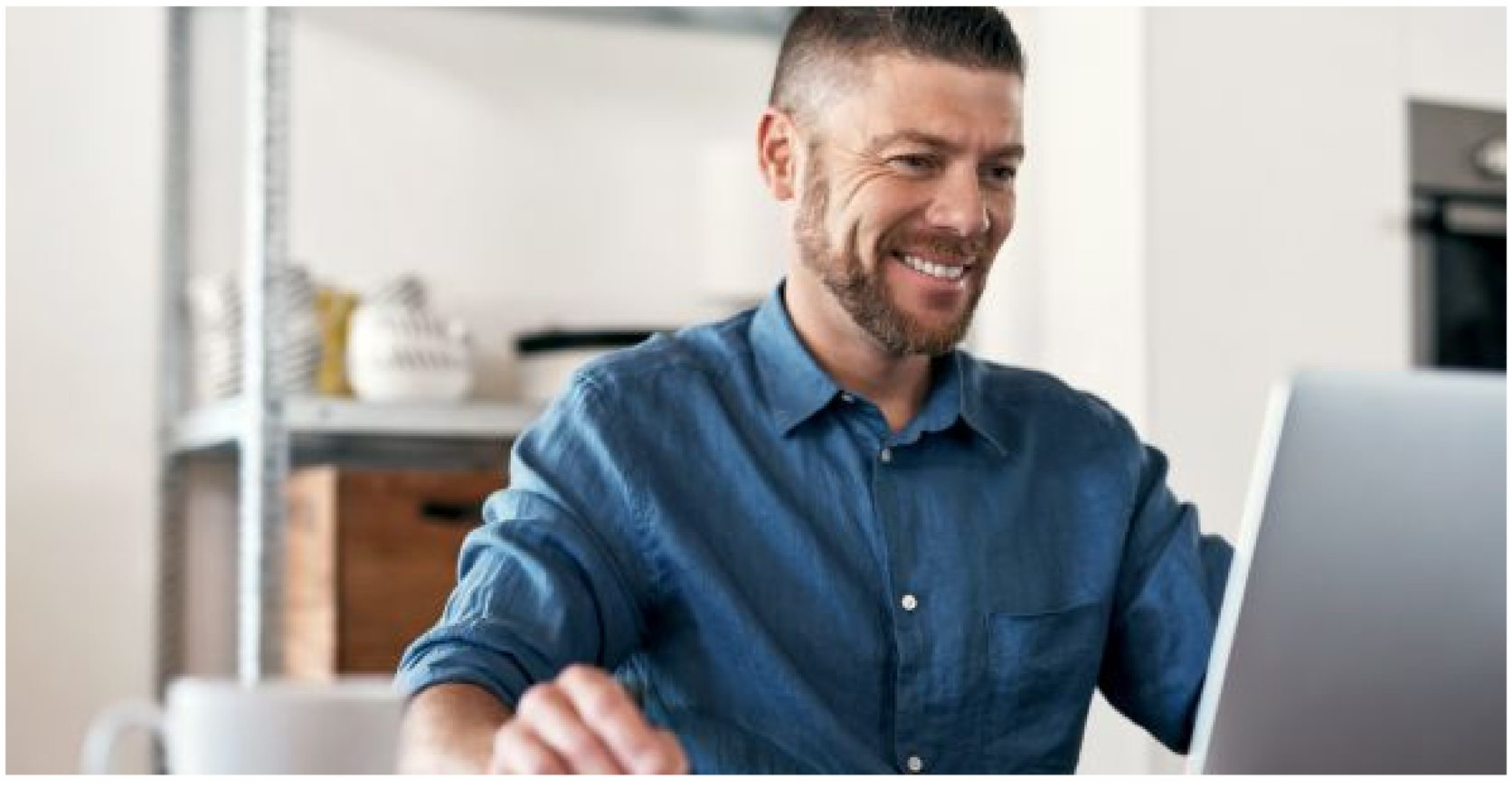
What is a relational database?



What is a relational database?

A relational database is a type of database that organizes data into rows and columns, which collectively form a table where the data points are related to each other.

Data is typically structured across multiple tables, which can be joined together via a primary key or a foreign key. These unique identifiers demonstrate the different relationships which exist between tables, and these relationships are usually illustrated through different types of data models. Analysts use SQL queries to combine different data points and summarize business performance, allowing organizations to gain insights, optimize workflows, and identify new opportunities.

For example, imagine your company maintains a database table with customer information, which contains company data at the account level. There may also be a different table, which describes all the individual transactions that align to that account. Together, these tables can provide information about the different industries that purchase a specific software product.

The columns (or fields) for the customer table might be *Customer ID*, *Company Name*, *Company Address*, *Industry* etc.; the columns for a transaction table might be *Transaction Date*, *Customer ID*, *Transaction Amount*, *Payment Method*, etc. The tables can be joined together with the common *Customer ID* field. You can, therefore, query the table to produce valuable reports, such as a sales reports by industry or company, which can inform messaging to prospective clients.

Relational databases are also typically associated with transactional databases, which execute commands, or transactions, collectively. A popular example that is used to illustrate this is a bank transfer. A defined amount is withdrawn from one account, and then it is deposited within another. The total amount of money is withdrawn and deposited, and this transaction cannot occur in any kind of

partial sense. Transactions have specific properties. Represented by the acronym, ACID, ACID properties are defined as:

- **Atomicity:** All changes to data are performed as if they are a single operation. That is, all the changes are performed, or none of them are.
- **Consistency:** Data remains in a consistent state from state to finish, reinforcing data integrity.
- **Isolation:** The intermediate state of a transaction is not visible to other transactions, and as a result, transactions that run concurrently appear to be serialized.
- **Durability:** After the successful completion of a transaction, changes to data persist and are not undone, even in the event of a system failure.

These properties enable reliable transaction processing.

What is a relational database management system (RDBMS)

While a relational database organizes data based off a relational data model, a relational database management system (RDBMS) is a more specific reference to the underlying database software that enables users to maintain it. These programs allow users to create, update, insert, or delete data in the system, and they provide:

- Data structure
- Multi-user access
- Privilege control
- Network access

Examples of popular RDBMS systems include MySQL, PostgreSQL, and IBM DB2. Additionally, a relational database system differs from a basic database management system (DBMS) in that it stores data in tables while a DBMS stores information as files.

What is SQL?

Invented by Don Chamberlin and Ray Boyce at IBM, Structured Query Language (SQL) is the standard programming language for interacting with relational database management systems, allowing database administrator to add, update, or delete rows of data easily. Originally known as SEQUEL, it was simplified to SQL due to a trademark issue. SQL queries also allows users to retrieve data from databases using only a few lines of code. Given this relationship, it’s easy to see why relational databases are also referred to as “SQL databases” at times.

Using the example from above, you might construct a query to find the top 10 transactions by company for a specific year with the following code:

```
SELECT COMPANY_NAME, SUM(TRANSACTION_AMOUNT)

FROM TRANSACTION_TABLE A

LEFT JOIN CUSTOMER_TABLE B

ON A.CUSTOMER_ID = B.CUSTOMER_ID

WHERE YEAR(DATE) = 2022

GROUP BY 1
```

ORDER BY 2 DESC

LIMIT 10

The ability to join data in this way helps us to reduce redundancy within our data systems, allowing data teams to maintain one master table for customers versus duplicating this information if there was another transaction in the future. To learn more, Don details more of the history of SQL in his paper here .

A brief history of relational databases

Before relational databases, companies used a hierarchical database system with a tree-like structure for the data tables. These early database management systems (DBMS) enabled users to organize large quantities of data. However, they were complex, often proprietary to a particular application, and limited in the ways in which they could uncover within the data. These limitations eventually led IBM researcher, Edgar F. Codd, to publish a paper in 1970, titled "A Relational Model of Data for Large Shared Data Banks," which theorized the relational database model. In this proposed model, information could be retrieved without specialized computer knowledge. He proposed arranging data based on meaningful relationships as tuples, or attribute-value pairs. Sets of tuples were referred to as relations, which ultimately enabled the merging of data across tables.

In 1973, the San Jose Research Laboratory—now known as the Almaden Research Center—began a program called System R (R for relational) to prove this relational theory with what it called "an industrial-strength implementation." It ultimately became a testing ground for SQL as well, enabling it to become more widely adopted in a short period of time. However, Oracle's adoption of SQL also didn't hurt its popularity with database administrators.

By 1983, IBM introduced the DB2 family of relational databases, so named because it was IBM's second family of database management software. Today, it is one of IBM's most successful products, continuing to handle billions of transactions every day on cloud infrastructure and setting the foundational layer for machine learning applications.

Relational vs. non-relational databases

While relational databases structure data into a tabular format, non-relational databases do not have as rigid of a database schema. In fact, non-relational databases organize data differently based on the type of database. Irrespective of the type of non-relational database, they all aim to solve for the flexibility and scalability issues inherent in relational models which are not ideal for unstructured data formats, like text, video, and images. These types of databases include:

- **Key-value store:** This schema-less data model is organized into a dictionary of key-value pairs, where each item has a key and a value. The key could be like something similar found in a SQL database, like a shopping cart ID, while the value is an array of data, like each individual item in that user's shopping cart. It's commonly used for caching and storing user session information, such as shopping carts. However, it's not ideal when you need to pull multiple records at a time. Redis and Memcached are examples of open-source databases with this data model.

- **Document store:** As suggested by the name, document databases store data as documents. They can be helpful in managing semi-structured data, and data are typically stored in JSON, XML, or BSON formats. This keeps the data together when it is used in applications, reducing the amount of translation needed to use the data. Developers also gain more flexibility since data schemas do not need to match across documents (e.g. name vs. first_name). However, this can be problematic for complex transactions, leading to data corruption. Popular use cases of document databases include content management systems and user profiles. An example of a document-oriented database is MongoDB, the database component of the MEAN stack.

- **Wide-column store:** These databases store information in columns, enabling users to access only the specific columns they need without allocating additional memory on irrelevant data. This

database tries to solve for the shortcomings of key-value and document stores, but since it can be a more complex system to manage, it is not recommended for use for newer teams and projects. Apache HBase and Apache Cassandra are examples of open-source, wide-column databases. Apache HBase is built on top of Hadoop Distributed Files System that provides a way of storing sparse data sets, which is commonly used in many big data applications. Apache Cassandra, on the other hand, has been designed to manage large amounts of data across multiple servers and clustering that spans multiple data centers. It's been used for a variety of use cases, such as social networking websites and real-time data analytics.

- **Graph store:** This type of database typically houses data from a knowledge graph. Data elements are stored as nodes, edges and properties. Any object, place, or person can be a node. An edge defines the relationship between the nodes. Graph databases are used for storing and managing a network of connections between elements within the graph. Neo4j (link resides outside IBM), a graph-based database service based on Java with an open-source community edition where users can purchase licenses for online backup and high availability extensions, or pre-package licensed version with backup and extensions included.

NoSQL databases also prioritize availability over consistency.

When computers run over a network, they invariably need to decide to prioritize consistent results (where every answer is always the same) or high uptime, called "availability." This is called the "CAP Theory," which stands for Consistency, Availability, or Partition Tolerance. Relational databases ensure the information is always in-sync and consistent. Some NoSQL databases, like Redis, prefer to always provide a response. That means the information you receive from a query may be incorrect by a few seconds—perhaps up to half a minute. On social media sites, this means seeing an old profile picture when the newest one is only a few moments old. The alternative could be a timeout or error. On the other hand, in banking and financial transactions, an error and resubmit may be better than old, incorrect information.

For a full rundown of the differences between SQL and NoSQL, see "SQL vs. NoSQL Databases: What's the Difference?"

Benefits of relational databases

The primary benefit of the relational database approach is the ability to create meaningful information by joining the tables. Joining tables allows you to understand the *relations* between the data, or how the tables connect. SQL includes the ability to count, add, group, and also combine queries. SQL can perform basic math and subtotal functions and logical transformations. Analysts can order the results by date, name, or any column. These features make the relational approach the single most popular query tool in business today.

Relational databases have several advantages compared to other database formats:

Ease of Use

By virtue of its product lifespan, there is more of a community around relational databases, which partially perpetuates its continued use. SQL also makes it easy to retrieve datasets from multiple tables and perform simple transformations such as filtering and aggregation. The use of indices within relational databases also allows them to locate this information quickly without searching each row in the selected table.

While relational databases have historically been viewed as a more rigid and inflexible data storage option, advances in technology and DBaaS options are changing that perception. While there is still more overhead to develop schemas compared to NoSQL database offerings, relational databases are becoming more flexible as they migrate to cloud environments.

Reduced redundancy

Relational databases can eliminate redundancy in two ways. The relational model itself reduces data redundancy via a process known as normalization. As noted earlier, a customer table should only log unique records of customer information versus duplicating this information for multiple transactions.

Stored procedures also help to reduce repetitive work. For example, if database access is restricted to certain roles, functions or teams, a stored procedure can help to manage access-control. These reusable functions free up coveted application developer time to tackle high impact work.

Ease of backup and disaster recovery

Relational databases are transactional—they guarantee the state of the entire system is consistent at any moment. Most relational databases offer easy export and import options, making backup and restore trivial. These exports can happen even while the database is running, making restore on failure easy. Modern, cloud-based relational databases can do continuous mirroring, making the loss of data on restore measured in seconds or less. Most cloud-managed services allow you to create Read Replicas, like in IBM Cloud® Databases for PostgreSQL. These Read Replicas enable you to store a read-only copy of your data in a cloud data center. Replicas can be promoted to Read/Write instances for disaster recovery as well.