What is a Relational Database (RDBMS)?

A relational database is a type of database that stores and provides access to data points that are related to one another. Relational databases are based on the relational model, an intuitive, straightforward way of representing data in tables. In a relational database, each row in the table is a record with a unique ID called the key. The columns of the table hold attributes of the data, and each record usually has a value for each attribute, making it easy to establish the relationships among data points.
Learn more about Oracle RDBMS Database

**A relational database example**

Here's a simple example of two tables a small business might use to process orders for its products. The first table is a customer info table, so each record includes a customer's name, address, shipping and billing information, phone number, and other contact information. Each bit of information (each attribute) is in its own column, and the database assigns a unique ID (a key) to each row. In the second table—a customer order table—each record includes the ID of the customer that placed the order, the product ordered, the quantity, the selected size and color, and so on—but not the customer's name or contact information.

These two tables have only one thing in common: the ID column (the key). But because of that common column, the relational database can create a relationship between the two tables. Then, when the company's order processing application submits an order to the database, the database can go to the customer order table, pull the correct information about the product order, and use the customer ID from that table to look up the customer's billing and shipping information in the customer info table. The warehouse can then pull the correct product, the customer can receive timely delivery of the order, and the company can get paid.

**How relational databases are structured**

The relational model means that the logical data structures—the data tables, views, and indexes—are separate from the physical storage structures. This separation means that database administrators can manage physical data storage without affecting access to that data as a logical structure. For example, renaming a database file does not rename the tables stored within it.

The distinction between logical and physical also applies to database operations, which are clearly defined actions that enable applications to manipulate the data and structures of the database. Logical operations allow an application to specify the content it needs, and physical operations determine how that data should be accessed and then carries out the task.
To ensure that data is always accurate and accessible, relational databases follow certain integrity rules. For example, an integrity rule can specify that duplicate rows are not allowed in a table in order to eliminate the potential for erroneous information entering the database.

## The relational model

In the early years of databases, every application stored data in its own unique structure. When developers wanted to build applications to use that data, they had to know a lot about the particular data structure to find the data they needed. These data structures were inefficient, hard to maintain, and hard to optimize for delivering good application performance. The relational database model was designed to solve the problem of multiple arbitrary data structures.

The relational data model provided a standard way of representing and querying data that could be used by any application. From the beginning, developers recognized that the chief strength of the relational database model was in its use of tables, which were an intuitive, efficient, and flexible way to store and access structured information.

Over time, another strength of the relational model emerged as developers began to use structured query language (SQL) to write and query data in a database. For many years, SQL has been widely used as the language for database queries. Based on relational algebra, SQL provides an internally consistent mathematical language that makes it easier to improve the performance of all database queries. In comparison, other approaches must define individual queries.

## Benefits of relational database management system

The simple yet powerful relational model is used by organizations of all types and sizes for a broad variety of information needs. Relational databases are used to track inventories, process ecommerce transactions, manage huge amounts of mission-critical customer information, and much more. A relational database can be considered for any information need in which data points relate to each other and must be managed in a secure, rules-based, consistent way.

Relational databases have been around since the 1970s. Today, the advantages of the relational model continue to make it the most widely accepted model for databases.

## Relational model and data consistency

The relational model is the best at maintaining data consistency across applications and database copies (called instances). For example, when a customer deposits money at an ATM and then looks at the account balance on a mobile phone, the customer expects to see that deposit reflected immediately in an updated account balance. Relational databases excel at this kind of data consistency, ensuring that multiple instances of a database have the same data all the time.

It’s difficult for other types of databases to maintain this level of timely consistency with large amounts of data. Some recent databases, such as NoSQL, can supply only “eventual consistency.” Under this principle, when the database is scaled or when multiple users access the same data at the same time, the data needs some time to “catch up.” Eventual consistency is acceptable for some uses, such as to maintain

listings in a product catalog, but for critical business operations such as shopping cart transactions, the relational database is still the gold standard.

### Commitment and atomicity

Relational databases handle business rules and policies at a very granular level, with strict policies about commitment (that is, making a change to the database permanent). For example, consider an inventory database that tracks three parts that are always used together. When one part is pulled from inventory, the other two must also be pulled. If one of the three parts isn't available, none of the parts should be pulled—all three parts must be available before the database makes any commitment. A relational database won't commit for one part until it knows it can commit for all three. This multifaceted commitment capability is called atomicity. Atomicity is the key to keeping data accurate in the database and ensuring that it is compliant with the rules, regulations, and policies of the business.

### ACID properties and RDBMS

Four crucial properties define relational database transactions: atomicity, consistency, isolation, and durability—typically referred to as ACID.

- **Atomicity** defines all the elements that make up a complete database transaction.
- **Consistency** defines the rules for maintaining data points in a correct state after a transaction.
- **Isolation** keeps the effect of a transaction invisible to others until it is committed, to avoid confusion.
- **Durability** ensures that data changes become permanent once the transaction is committed.

### Stored procedures and relational databases

Data access involves many repetitive actions. For example, a simple query to get information from a data table may need to be repeated hundreds or thousands of times to produce the desired result. These data access functions require some type of code to access the database. Application developers don't want to write new code for these functions in each new application. Luckily, relational databases allow stored procedures, which are blocks of code that can be accessed with a simple application call. For example, a single stored procedure can provide consistent record tagging for users of multiple applications. Stored procedures can also help developers ensure that certain data functions in the application are implemented in a specific way.

### Database locking and concurrency

Conflicts can arise in a database when multiple users or applications attempt to change the same data at the same time. Locking and concurrency techniques reduce the potential for conflicts while maintaining the integrity of the data.

Locking prevents other users and applications from accessing data while it is being updated. In some databases, locking applies to the entire table, which creates a negative impact on application performance. Other databases, such as Oracle relational databases, apply locks at the record level, leaving the other records within the table available, helping ensure better application performance.

Concurrency manages the activity when multiple users or applications invoke queries at the same time on the same database. This capability provides the right access to users and applications according to policies defined for data control.

**What to look for when selecting a relational database**

The software used to store, manage, query, and retrieve data stored in a relational database is called a relational database management system (RDBMS). The RDBMS provides an interface between users and applications and the database, as well as administrative functions for managing data storage, access, and performance.

Several factors can guide your decision when choosing among database types and relational database products. The RDBMS you choose will depend on your business needs. Ask yourself the following questions:

- What are our data accuracy requirements? Will data storage and accuracy rely on business logic? Does our data have stringent requirements for accuracy (for example, financial data and government reports)?
- Do we need scalability? What is the scale of the data to be managed, and what is its anticipated growth? Will the database model need to support mirrored database copies (as separate instances) for scalability? If so, can it maintain data consistency across those instances?
- How important is concurrency? Will multiple users and applications need simultaneous data access? Does the database software support concurrency while protecting the data?
- What are our performance and reliability needs? Do we need a high-performance, high-reliability product? What are the requirements for query-response performance? What are the vendor's commitments for service level agreements (SLAs) or unplanned downtime?

**The relational database of the future: The self-driving database**

Over the years, relational databases have gotten better, faster, stronger, and easier to work with. But they've also gotten more complex, and administering the database has long been a full-time job. Instead of using their expertise to focus on developing innovative applications that bring value to the business, developers have had to spend most of their time on the management activity needed to optimize database performance.

Today, autonomous technology is building upon the strengths of the relational model, cloud database technology, and machine learning to deliver a new type of relational database. The self-driving database (also known as the autonomous database) maintains the power and advantages of the relational model but uses artificial intelligence (AI), machine learning, and automation to monitor and improve query performance and management tasks. For example, to improve query performance, the self-driving database can hypothesize and test indexes to make queries faster, and then push the best ones into production—all on its own. The self-driving database makes these improvements continuously, without the need for human involvement.
Autonomous technology frees up developers from the mundane tasks of managing the database. For instance, they no longer have to determine infrastructure requirements in advance. Instead, with a self-driving database, they can add storage and compute resources as needed to support database growth. With just a few steps, developers can easily create an autonomous relational database, accelerating the time for application development.

# Database

Oracle database services and products offer customers cost-optimized and high-performance versions of Oracle Database, the world's leading converged, multi-model database management system, as well as in-memory, NoSQL and MySQL databases. Oracle Autonomous Database, available on premises via Oracle Cloud@Customer or in the Oracle Cloud Infrastructure, enables customers to simplify relational database environments and reduce management workloads.

### *Why choose Oracle Database for all your data needs?*

### *IDC: Oracle Autonomous Database provides 417% ROI*

Independent analyst study shows how Oracle Autonomous Database offers significant savings and 417% ROI over five years, with only 5 months to payback.

Read the IDC report (PDF)

### Guard against data breaches

Assess, detect, and prevent data security threats with Oracle database security solutions for encryption, key management, data masking, privileged user access controls, activity monitoring, and auditing. Reduce the risk of a data breach and simplify and accelerate compliance.

### IDC: Security benefits of Oracle Autonomous Database (PDF)

### Try it—Database Security Basics Workshop

### *Use a single database for all data types*

Free application developers from complex transformations and redundant data with Oracle's converged database.

Watch the video (2:49)

Try it—Develop Apps Using JSON, XML, Spatial and Graph Workshop

### *Deploy where you need to*

Deploy Oracle Database wherever required—in your data center, public cloud, or private cloud. This offers the flexibility between deployment in your data center when residency or latency are critical, or in the cloud when you want to take advantage of scalability and the broadest set of capabilities.

**Oracle brings the Oracle Cloud to you (PDF)**

**Try it—Containerized Development with Docker on Autonomous Database Workshop**

**Cloud deployment options**