databases

Enter the Purpose-Built Database Era:

Finding the right database type for the right job

Stepping into the purpose-built era

Data is a strategic asset for every organization. As data continues to exponentially grow, databases are becoming increasingly crucial to understanding data and converting it to valuable insights.

IT leaders need to look for ways to get more value from their data. If you're running legacy databases on-premises, you're likely finding that provisioning, operating, scaling, and managing databases is tedious, time-consuming, and expensive. You need modernized database solutions that allow you to spend time innovating and building new applications—not managing infrastructure.

Moving on-premises data to managed databases built for the cloud can help you reduce time and costs. Once your databases are in the cloud, you can innovate and build new applications faster—all while getting deeper and more valuable insights.

Migrating to the cloud is the first step toward entering the era of purpose-built databases. But once in the cloud, how do you know which types of databases to use for which functions? Read on to learn more about purpose-built database types—and how you can ensure a smooth transition into an era of innovation, performance, and business success.

Going beyond relational only

Before we begin discussing purpose-built databases, let's examine the status quo—using relational databases for just about every use case.

Relational databases were designed for tabular data with consistent structure and fixed schema. They work for problems that are well defined at the onset. Traditional applications like ERP, CRM, and e-commerce need relational databases to log transactions and store structured data, typically in GBs and occasionally TBs.

For decades, application design has been driven by relational database requirements, limiting innovation. To compete in today's market, this paradigm must reverse—with databases built to serve the needs of different kinds of applications, not the other way around.

While relational databases are still essential—in fact, they are still growing—a "relational only" approach no longer works in today's world.

With the rapid growth of data—not just in volume and velocity but also in variety, complexity, and interconnectedness—the needs of databases have changed. Many new applications that have social, mobile, IoT, and global access requirements cannot function properly on a relational database alone.

These applications need databases that can store TBs to PBs of new types of data, provide access to data with millisecond latency, process millions of requests per second, and scale to support millions of users anywhere in the world.

To create applications that meet these demands, developers must choose between a number of purpose-built database models. They must understand which database type to use when selecting the right tool for the right job.

In the following pages, we will examine a variety of database types, exploring the strengths, challenges, and primary use cases of each.

At-a-glance

Quickly jump to info on different database types

Relational

Provides high integrity, accuracy, and consistency, limitless indexing

Useful for ERP, CRM, finance, transactions, and data warehousing

Key Value

Predictable low latency regardless of scale, flexible schema, optional consistency

Useful for real-time bidding, shopping cart, product catalog, customer preferences, and sorted data collections such as metrics, messages, and order histories

Document

Flexible, semi-structured, hierarchical, evolves with application needs, powerful indexing, fast querying

Useful for catalogs, content management systems, user profiles, personalization, and mobile

In-Memory

Sub-millisecond latency, millions of operations per second, simple instruction set, support for rich commands, works with any type of database

Useful for caching, session store, leaderboards, geospatial, and real-time analytics

Graph

Create and traverse relationships within highly connected data sets

Useful for fraud detection, social networking, data lineage, and knowledge graphs

Time Series

High scalability for data that accumulates quickly

Useful for DevOps, application monitoring, industrial telemetry, and IoT applications

Ledger

Ensures accurate history, transparent, immutable, verifiable, and highly scalable

Useful for finance, manufacturing, insurance, HR and payroll, retail, and supply chains

Wide column

Provides flexibility and scalability for workloads that store massive amounts of data

Useful for storing information about users, devices, and events—for example, user profiles, IoT device data, and time series data such as log files or chat history in a messaging app

Relational databases

Description

In relational database management systems (RDBMS), data is stored in a tabular form of columns and rows and data is queried using the Structured Query Language (SQL). Each column of a table represents an attribute, each row in a table represents a record, and each field in a table represents a data value. Relational databases are so popular because 1) SQL is easy to learn and use without needing to know the underlying schema and 2) database entries can be modified without specifying the entire body.

Example: Relational Database Schema

	Patient
x	Patient ID
	First Name
	Last Name
	Gender
	DOB
x	Doctor ID

	Doctor
x	Doctor ID
	First Name
	Last Name
	Medical Specialty
x	Hospital Affiliation

	Hospital
x	Hospital ID
	Name
	Address
	Rating

	Visit
x	Visit ID
x	Patient ID
x	Hospital ID
	Date
x	Treatment ID

	Medical Treatment
x	Treatment ID
	Procedure
	How Performed
	Adverse Outcome
	Contraindication

Advantages

Works well with structured data

Supports ACID transactional consistency and support "joins"

Comes with built-in data integrity

Ensures data accuracy and consistency

Constrains relationships in this system

Equipped with limitless indexing

Not designed for

Semi-structured or sparse data

Use cases

ERP apps

CRM

Finance

Transactions

Data warehousing

Key value databases

Description

A key-value database stores data as a collection of key-value pairs in which one or more keys serve as a unique identifier for the index. Schema is flexible for pairs which are not indexed, and may even be sparse. Values can be anything, ranging from simple numbers to compound strings or complete JSON documents. Key-value stores lend themselves well to sharded horizontal scaling, allowing for consistent read and write operation free from the constraints of vertically scaling of any single node. For operational needs where the access patterns are known and fully indexed, key value databases can provide consistent low latency performance at any scale.

Example: Key-value Table

<table>
<tr><th colspan="2">Primary Key</th><th rowspan="2">Attributes</th></tr>
<tr><th>Partition key: GameID</th><th>PSort key: Podium</th></tr>
<tr><td rowspan="6">Armadillos Exploring Space</td><td rowspan="2">1. Gold</td><td>PlayerID</td></tr>
<tr><td>Hammer57</td></tr>
<tr><td rowspan="2">2. Silver</td><td>PlayerID</td></tr>
<tr><td>Lol777313</td></tr>
<tr><td rowspan="2">3. Bronze</td><td>PlayerID</td></tr>
<tr><td>x Jam22Jam</td></tr>
</table>

Advantages

Scaling decoupled from CPU load of any single node, resulting in consistent low latency regardless of throughput requirements

Schema flexibility allows for sparse storage and direct developer ownership

Not designed for

Ad-hoc or analytical access patterns

Synchronous aggregations or referential integrity

Use cases

Real-time bidding

Shopping cart

Product catalog

Customer preferences

IoT data collection

Sorted order or activity histories

Document databases

Description

In document databases, data is stored in JSON-like documents and JSON documents are first-class objects within the database. These databases make it easier for developers to store and query data by using the same document-model format developers use in their application code.

Example: JSON Document

```
[
 {
  "year" : 2013,
  "title" : "Turn It Down, Or Else!",
  "info" : {
      "directors" : [ "Alice Smith", "Bob Jones"],
      "release_date" : "2013-01-18T00:00:00Z",
      "rating" : 6.2,
      "genres" : ["Comedy", "Drama"],
      "image_url" : "http://ia.media-imdb.com/images/N/O9ERWAU7FS797AJ7LU8HN09AMUP908RLlo5JF90EWR7LJKQ7@@._V1_SX400_.jpg",
      "plot" : "A rock band plays their music at high volumes, annoying the neighbors.",
      "actors" : ["David Matthewman", "Jonathan G. Neff"]
    }
 },v
{
  "year": 2015,
  "title": "The Big New Movie",
   "info": {
      "plot": "Nothing happens at all.",
      "rating": 0
}
}
]
```

Advantages

Flexible, semi-structured, and hierarchical

Adjustable to application needs as databases evolve

Flexible schema

Simple hierarchical and semi-structured data

Powerfully index for fast querying

Naturally map documents to object-oriented programming

Easily flows data to persistent layer

Expressive query languages built for documents

Capable of ad-hoc queries and aggregations across documents

Not designed for

Explicitly defined relations between different pieces of data

Use Cases

Catalogs

Content management systems

User profiles / personalization

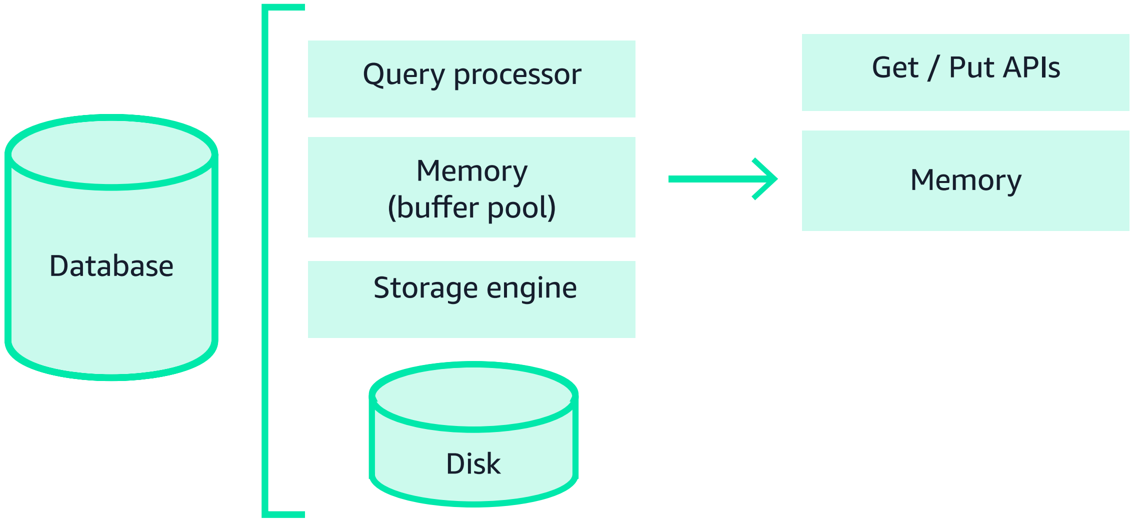
Mobile

In-memory databases

Description

With the rise of real-time applications, in-memory databases are growing in popularity. In-memory databases predominantly rely on main memory for data storage, management, and manipulation. In-memory has been popularized by open-source software for memory caching, which can speed up dynamic databases by caching data to decrease access latency, increase throughput, and ease the load off the main databases.

Example: In-memory Database Architecture



Advantages

Sub-millisecond latency

Can perform millions of operations per second

Significant performance gains when compared to disk-based alternatives

Simpler instruction set

Support for rich command set (Redis)

Works with any type of database, relational or non-relational, or even storage services

Not designed for

Persisting data to disk all the time

Use Cases

Caching

Session store

Leaderboards

Geospatial services

Pub/sub

Real-time streaming

Graph databases

Description

Graph databases are a type of NoSQL database designed to make it easy to build and run applications that work with highly connected datasets. In a graph data model, relationships are first class citizens, i.e. they are represented directly. Using specialized graph languages, like SPARQL or Gremlin, allows you to easily build queries that efficiently navigate highly connected datasets.

In graph databases, data is stored in the form of nodes, edges, and properties:

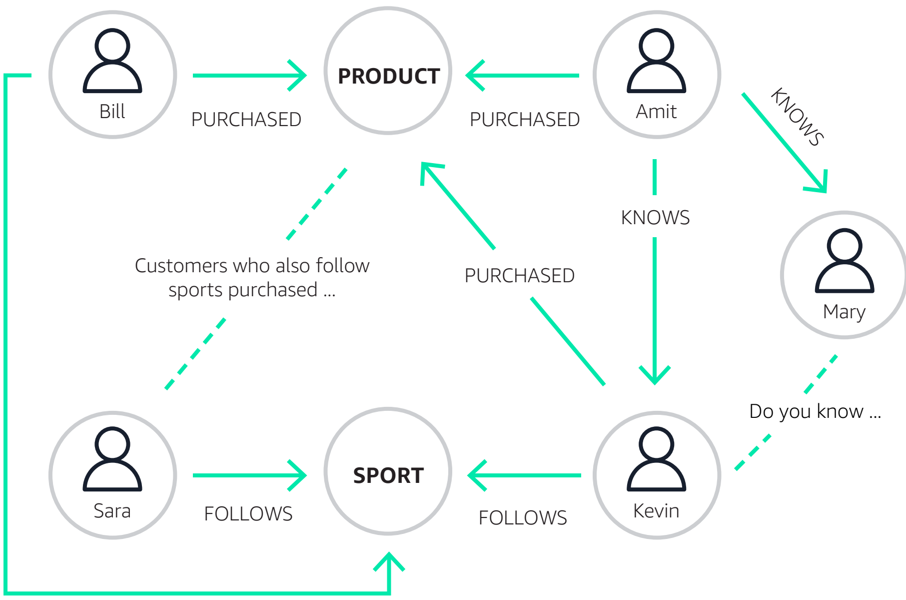
Nodes are equivalent to records in a relational database system

Edges represent relationships that connect nodes

Properties are additional information added to the nodes

In RDF graphs, the concepts of Nodes, Edges, and Properties are represented as Resources with Internationalized Resource Identifiers (IRIs)

Example: Graph Nodes and Relationships



Advantages

Ability to make frequent schema changes

Quickly make relationships between many different types of data

Real-time query response time

Superior performance for querying related data–big or small

Meets more intelligent data activation requirements

Explicit semantics for each query—no hidden assumptions

Flexible online schema environment

Not designed for

Applications that do not traverse or query relationships

Processing high volumes of transactions

Handling queries that span the entire database

Use Cases

Fraud detection

Social networking

Recommendation engines

Knowledge graphs

Data lineage

Time series databases

Description

Time series databases are optimized for time-stamped or time series data. Time series data is very different from other data workloads in that it typically arrives in time-order form, the data is append-only, and queries are always over a time interval. Examples of such data include server metrics, application performance monitoring, network data, sensor data, events, clicks, trades in a market, and many other types of analytics.

Advantages

Ideal for measurements or events that are tracked, monitored, and aggregated over time

High scalability for quickly accumulating time series data

Robust usability for many functions, including:

- Data retention policies
- Continuous queries
- Flexible time aggregations

Not designed for

Data not in time order form, such as:

- Documents
- Catalogs
- Customer profiles

Use Cases

DevOps

Application monitoring

Industrial telemetry

IoT applications

Ledger databases

Description

Ledger databases provide a transparent, immutable, and cryptographically verifiable transaction log owned by a central trusted authority. Many organizations build applications with ledger-like functionality because they want to maintain an accurate history of their applications' data—for example, tracking the history of credits and debits in banking transactions, verifying the data lineage of an insurance claim, or tracking movement of an item in a supply chain network.

Advantages

Maintain accurate history of application data

Immutable and transparent

Cryptographically verifiable

Highly scalable

Not designed for

Decentralized use case

i.e., multiple entities need to read/write on data independently

Use Cases

Finance – Keep track of ledger data such as credits and debits

Manufacturing – Reconcile data between supply chain systems to track full manufacturing history

Insurance – Track claim transaction histories

HR and payroll – Track and maintain a record of employee details

Retail and supply chain – Access info on every supply chain stage

Wide column databases

Description

Wide column databases are massively scalable. They are good for applications that require fast performance and storing large amounts of data. In a wide column database, tables have schemas, and you should define your table schema to match your query patterns (i.e., no joins).

Advantages

Scalable

Flexible

Not designed for

Running queries across multiple tables by using joins

Use Cases

High scale industrial applications for:

- Equipment maintenance
- Fleet management
- Route optimization

Data logs

Geographic data

Getting the most from purpose-built databases

The world has changed, and the one-size-fits-all approach of using relational databases as the only store for your applications no longer works. Today's leading developers are breaking complex applications into microservices and then picking the right purpose-built databases for the right jobs. This ensures that their applications are well architected and scale effectively. Relational databases still play an important role in application design and functionality, but purpose-built database models are designed from the ground up to perform the specific functions modern applications require—quickly and efficiently.

Benefits of purpose-built databases:

Right tool for the right job
Better performance
Cloud scale
More functionality
Easier to debug and monitor
Independence between teams
Faster time to market
Lower TCO
Reduced operation

Today's developers need diverse data models that match a variety of use cases. Finding the right tool for the right job can be challenging, but we hope this document helps you simplify the process.

To get the most out of these different database types, however, you'll need to first migrate your data, databases, and applications to the cloud. And remember, not all cloud providers are created equal. You'll want a provider that offers the performance, scale, and availability of commercial databases and also the cost-effectiveness, freedom, and flexibility of open-source databases.

Customer success stories

Airbnb modernizes vacations without taking a day off

A year after Airbnb launched, the company decided to migrate nearly all of its cloud computing functions to AWS. As part of that migration, Airbnb moved its primary MySQL database to the Amazon Relational Database Service (RDS) for MySQL. The community-based vacation rental service was able to complete the database migration with only 15 minutes of downtime.

By running on Amazon RDS, Airbnb has reduced overhead associated with administrative tasks like replication and scaling, which can now be triggered with a simple API call or through the AWS Management Console. Airbnb further migrated from RDS MySQL to Amazon Aurora to improve read and write performance and scalability, cut down on replica creation and lag time, and improve failover and recovery time.

Airbnb also uses Amazon DynamoDB to store user search history, and Amazon ElastiCache to store session state in memory for faster (sub-millisecond) site rendering.

Customer success stories

Duolingo translates databases into business success

An online provider of free language training with a global audience of 300 million, Duolingo relies on AWS to store and serve up over 31 billion items for 80 different language courses. Duolingo requires elastically scalable, high-performance, high-concurrency database services—and that's exactly what it gets with AWS.

Duolingo uses Amazon DynamoDB as one of its primary database solutions. Each second, Duolingo's DynamoDB implementation supports 24,000 reads and 3,000 writes, personalizing lessons for users taking 6 billion exercises per month. And Amazon DynamoDB provides autoscaling, which intelligently adjusts performance based on user demand—ensuring high availability and minimizing wasted costs due to over-provisioning.

Duolingo also uses Amazon ElastiCache to provide instant access to common words and phrases, Amazon Aurora as the transactional database for maintaining user data, and Amazon Redshift for data analytics. With this database backbone, Duolingo teaches more language students than the entire US school system.

A portfolio purpose-built for success

AWS offers a broad range of database services that are purpose-built for every major use case. These services are fully managed and allow you to build applications that scale easily. All of these services are proven to provide deep functionality—so you get the availability, performance, reliability, and security required for production workloads.

AWS has a comprehensive portfolio of purpose-built databases, supports diverse data models, and allows your teams to build use case driven, highly scalable, distributed applications. By picking the best database to solve a specific problem or a group of problems, you can break away from restrictive one-size-fits-all monolithic databases and focus on building enterprise-class applications to meet the needs of your business.

Go to the next page for a more detailed look at these AWS database solutions.

Learn more about purpose-built databases on AWS >>

AWS purpose-built databases

Relational

Amazon Aurora: MySQL and PostgreSQL-compatible relational database built for the cloud. Combines the performance and availability of traditional enterprise databases with the simplicity and cost-effectiveness of open source databases.

Amazon Relational Database Service (RDS): Service that makes it easy to set up, operate, and scale relational databases in the cloud. Provides cost-efficient and resizable capacity while automating time-consuming administration tasks. Offers seven familiar database engines, including Amazon Aurora, PostgreSQL, MySQL, MariaDB, Oracle Database, and SQL Server.

Amazon Redshift: Cloud data warehouse that extends data warehouse queries to your data lake, with no loading required. Run analytic queries against petabytes of data stored locally and directly against exabytes of data stored in Amazon S3.

Key value

Amazon DynamoDB: Fully managed (serverless) key value database that delivers single-digit millisecond performance at any scale. Multi-region, multi-master database with built-in security, eventual and strong consistency of reads, ACID-compliant for transactional operations across one or more rows, backup and restore, and in-memory caching. Highest levels of availability and scaling elasticity.

Document

Amazon DocumentDB: Fast, scalable, highly available, and fully managed document database service that supports MongoDB workloads. Designed from the ground up for mission-critical performance, scalability, and availability.

In-memory

Amazon ElastiCache for Redis: Blazing fast, fully managed in-memory data store compatible with Redis. Provides sub-millisecond latency to power internet-scale, real-time applications.

Amazon ElastiCache for Memcached: Fully managed, in-memory key-value store service compatible with Memcached. Can be used as a cache or a data store. Delivers the performance, ease-of-use, and simplicity of Memcached.

Graph

Amazon Neptune: Fast, reliable, fully managed graph database service that makes it easy to build and run applications that work with highly connected datasets.

Time Series

Amazon Timestream: Scalable, fully managed, fast time series database service for IoT and operational applications. Enables storage and analysis of trillions of events per day at 1/10th the cost of relational databases.

Wide Column

Amazon Keyspaces (for Apache Cassandra): Scalable, highly available, and managed Apache Cassandra–compatible database service that allows you to use your existing Cassandra Query Language (CQL) code and tools. Serverless solution for building apps that can serve thousands of requests per second with virtually unlimited throughput and storage.

ABOUT AWS

For 14 years, Amazon Web Services has been the world's most comprehensive and broadly adopted cloud platform. AWS offers over 175 fully featured services for compute, storage, databases, networking, analytics, robotics, machine learning (ML), and artificial intelligence (AI), Internet of Things (IoT), mobile, security, hybrid, virtual and augmented reality (VR and AR), media, and application development, deployment, and management from 73 Availability Zones (AZs) within 23 geographic regions, spanning the U.S., Australia, Brazil, Canada, China, France, Germany, India, Ireland, Japan, Korea, Singapore, Sweden, and the U.K. Millions of customers, including the fastest-growing startups, largest enterprises, and leading government agencies, trust AWS to power their infrastructure, become more agile, and lower costs. To learn more about AWS, visit aws.amazon.com.