Why Use Databases Instead of File Storage?

Storing data directly on disk (e.g., EFS, EBS, EC2 Instance Store, S3) **can be limiting** when advanced data handling is required.
In many cases, **data needs to be structured, related, and queried efficiently**—which raw storage does not support well.

Therefore, **databases are introduced as a more powerful solution** for managing, organizing, and retrieving data at scale.

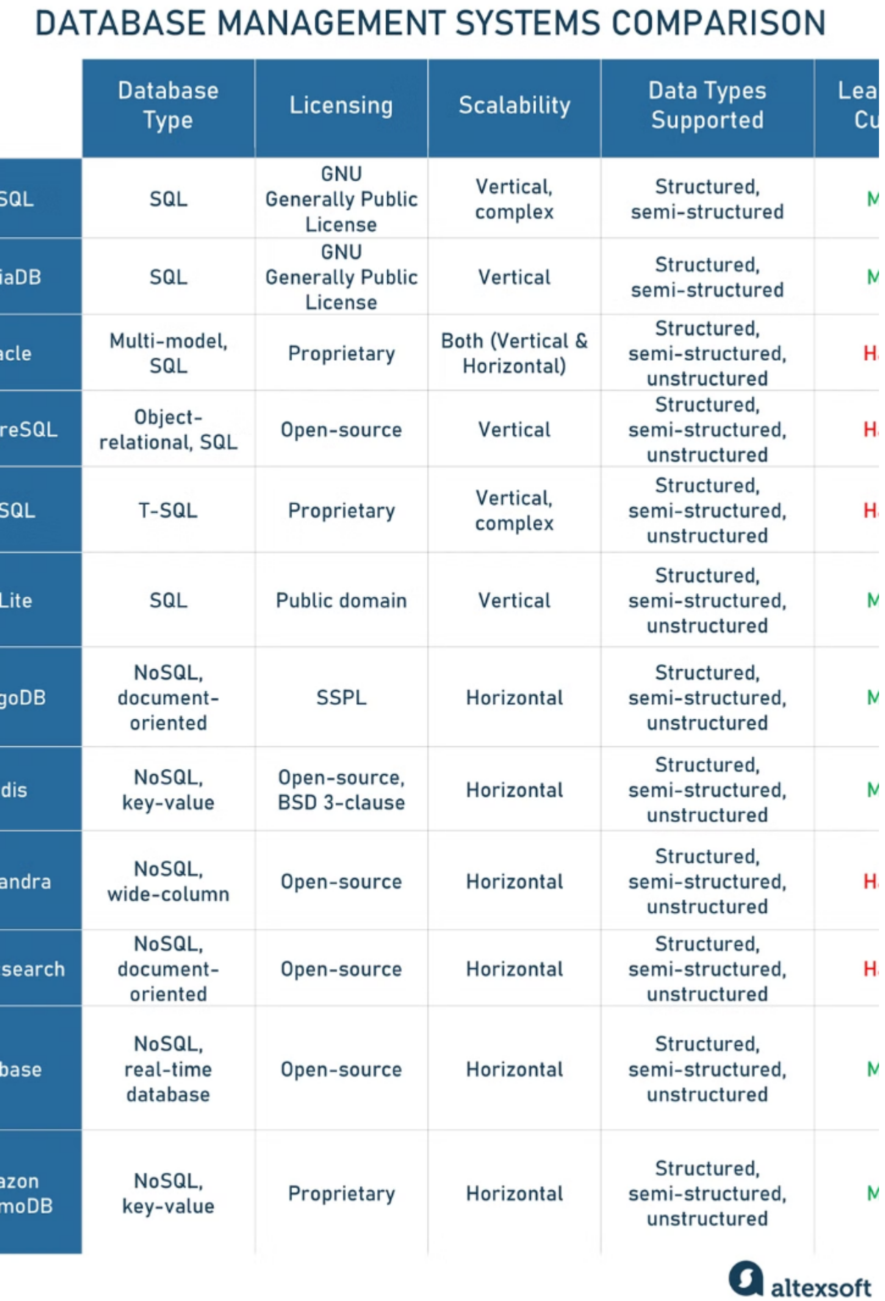
By using a database:

- The data can be **structured into tables or collections**.

- **Indexes** can be defined to enable fast searching and querying.

- **Relationships between datasets** (e.g., foreign keys in relational databases) can be modeled and maintained.

Purpose and Strength of Databases



DATABASE MANAGEMENT SYSTEMS COMPARISON

	Database Type	Licensing	Scalability	Data Types Supported	Lea Cu
SQL	SQL	GNU Generally Public License	Vertical, complex	Structured, semi-structured	M
iaDB	SQL	GNU Generally Public License	Vertical	Structured, semi-structured	M
acle	Multi-model, SQL	Proprietary	Both (Vertical & Horizontal)	Structured, semi-structured, unstructured	H
reSQL	Object-relational, SQL	Open-source	Vertical	Structured, semi-structured, unstructured	H
SQL	T-SQL	Proprietary	Vertical, complex	Structured, semi-structured, unstructured	H
Lite	SQL	Public domain	Vertical	Structured, semi-structured, unstructured	M
goDB	NoSQL, document-oriented	SSPL	Horizontal	Structured, semi-structured, unstructured	M
dis	NoSQL, key-value	Open-source, BSD 3-clause	Horizontal	Structured, semi-structured, unstructured	M
andra	NoSQL, wide-column	Open-source	Horizontal	Structured, semi-structured, unstructured	H
search	NoSQL, document-oriented	Open-source	Horizontal	Structured, semi-structured, unstructured	H
base	NoSQL, real-time database	Open-source	Horizontal	Structured, semi-structured, unstructured	M
azon moDB	NoSQL, key-value	Proprietary	Horizontal	Structured, semi-structured, unstructured	M

altexsoft

Design and Optimization

Databases are specifically **designed and optimized** to handle various types of data and access patterns.

Different Support

Different database systems support di

·**Features** (e.g., transactions, replicatio

·**Shapes** of data (e.g., tabular, docume

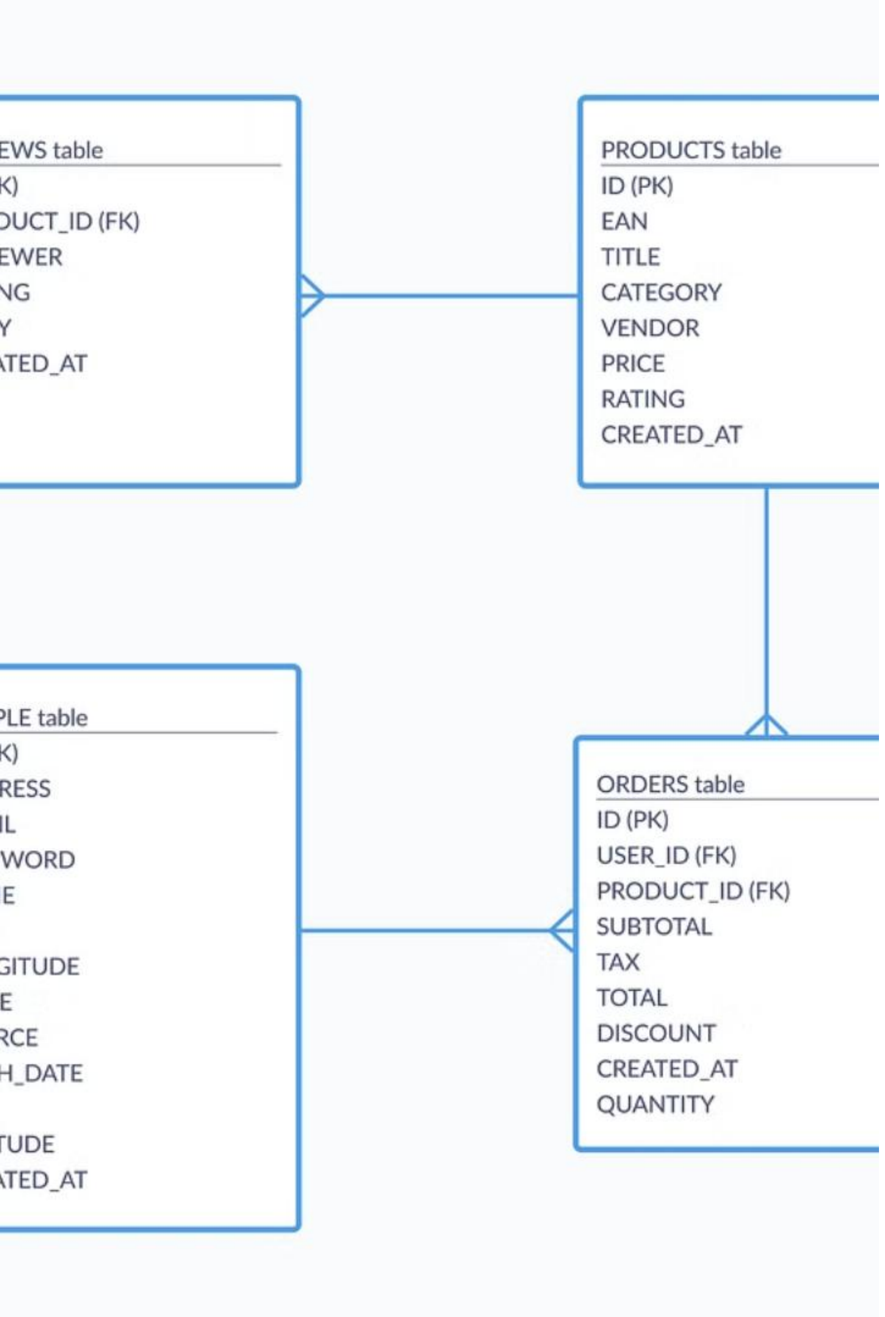
·**Constraints** (e.g., uniqueness, foreign

Selection Criteria

Databases are chosen based on:

- The **type of workload** (e.g., analytical vs transactional)
- The **structure of the data**
- The **performance requirements** and scalability goals

Understanding Relational Databases



EWS table
K)
DUCT_ID (FK)
EWER
NG
Y
ATED_AT

PRODUCTS table
ID (PK)
EAN
TITLE
CATEGORY
VENDOR
PRICE
RATING
CREATED_AT

PLE table
K)
RESS
IL
WORD
E
GITUDE
E
RCE
H_DATE
TUDE
ATED_AT

ORDERS table
ID (PK)
USER_ID (FK)
PRODUCT_ID (FK)
SUBTOTAL
TAX
TOTAL
DISCOUNT
CREATED_AT
QUANTITY

Structured Collections

Relational databases are **structured collections of data** organized into interrelated tables.

Tables and Records

Each table represents a specific **entity** (e.g., Students, Departments, Subjects) and contains **rows** (records) and **columns** (attributes).

Linked with Keys

Tables can be **linked together** using **keys**:

A **Primary Key** uniquely identifies each record in a table.

A **Foreign Key** connects a record to a related record in another table.

Benefits

These relationships allow for:

Consistent and organized data storage

Easier data maintenance and normalization

Powerful query capabilities using SQL (Structured Query Language)

Relational Database Example – Linked Tables

1

Tables Structure

Consider the following simplified relational database schema:

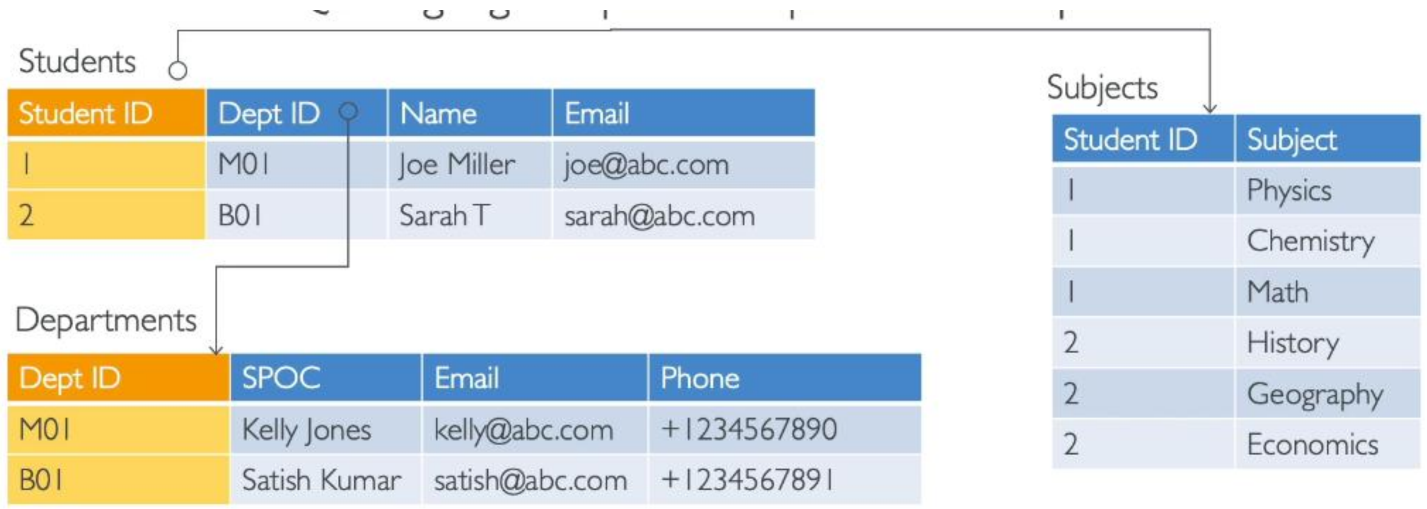
- **Students Table** contains student information and a foreign key (Dept ID) linking to the **Departments Table**.
- **Departments Table** stores department-level data, with Dept ID as the primary key.
- **Subjects Table** associates students with multiple subjects using Student ID as a foreign key.

2

Relationships

These tables demonstrate a **one-to-many relationship**:

- One department can have many students.
- One student can be enrolled in many subjects.



Students

Student ID	Dept ID	Name	Email
1	M01	Joe Miller	joe@abc.com
2	B01	Sarah T	sarah@abc.com

Departments

Dept ID	SPOC	Email	Phone
M01	Kelly Jones	kelly@abc.com	+1234567890
B01	Satish Kumar	satish@abc.com	+1234567891

Subjects

Student ID	Subject
1	Physics
1	Chemistry
1	Math
2	History
2	Geography
2	Economics

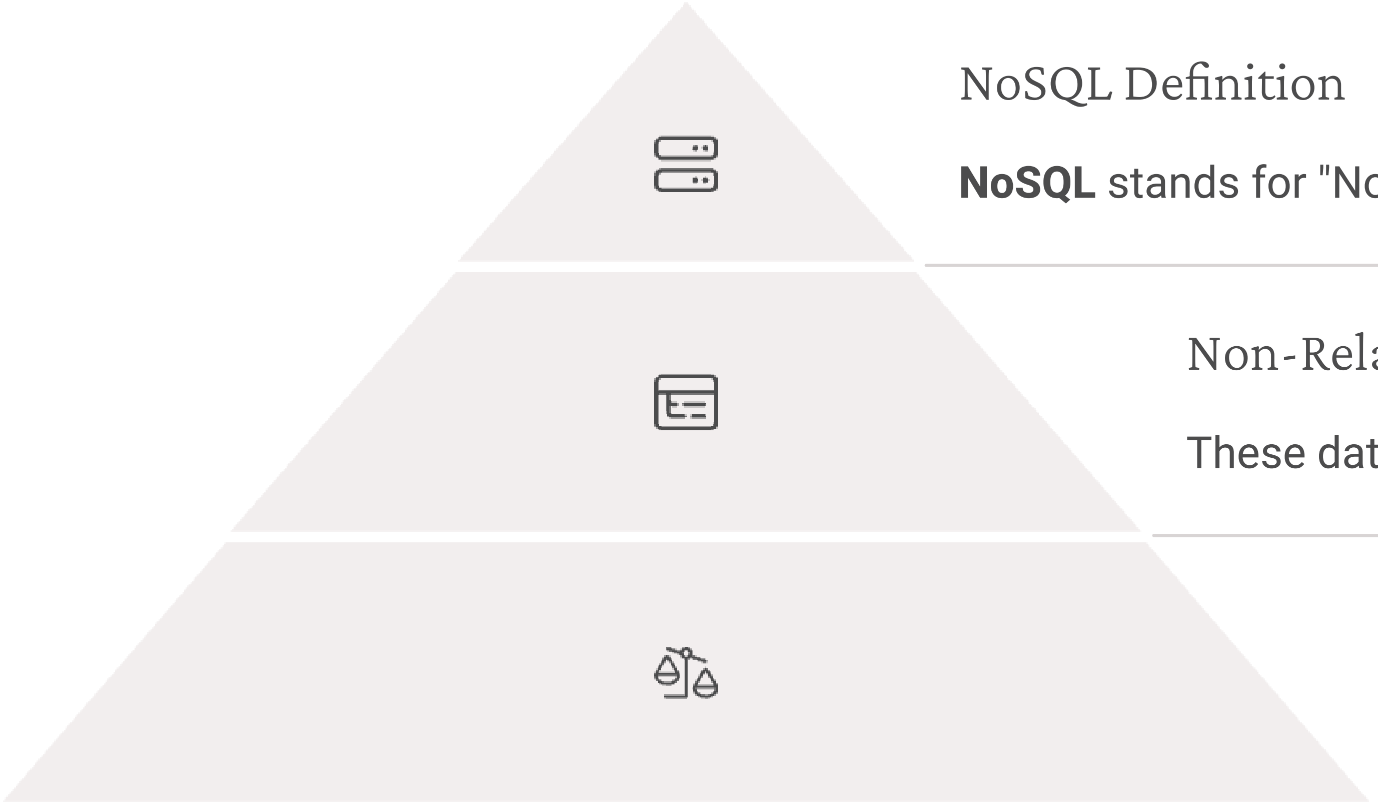
3

Query Capabilities

Such structure enables efficient querying, such as:

- "List all students in the Math department"
- "Find the subjects enrolled by Sarah T"
- "Retrieve contact details of the department SPOC for each student"

Introduction to NoSQL Databases



NoSQL Definition

NoSQL stands for "Not Only SQL"

Non-Relational Design

These databases do **not rely on fixed table schemas**

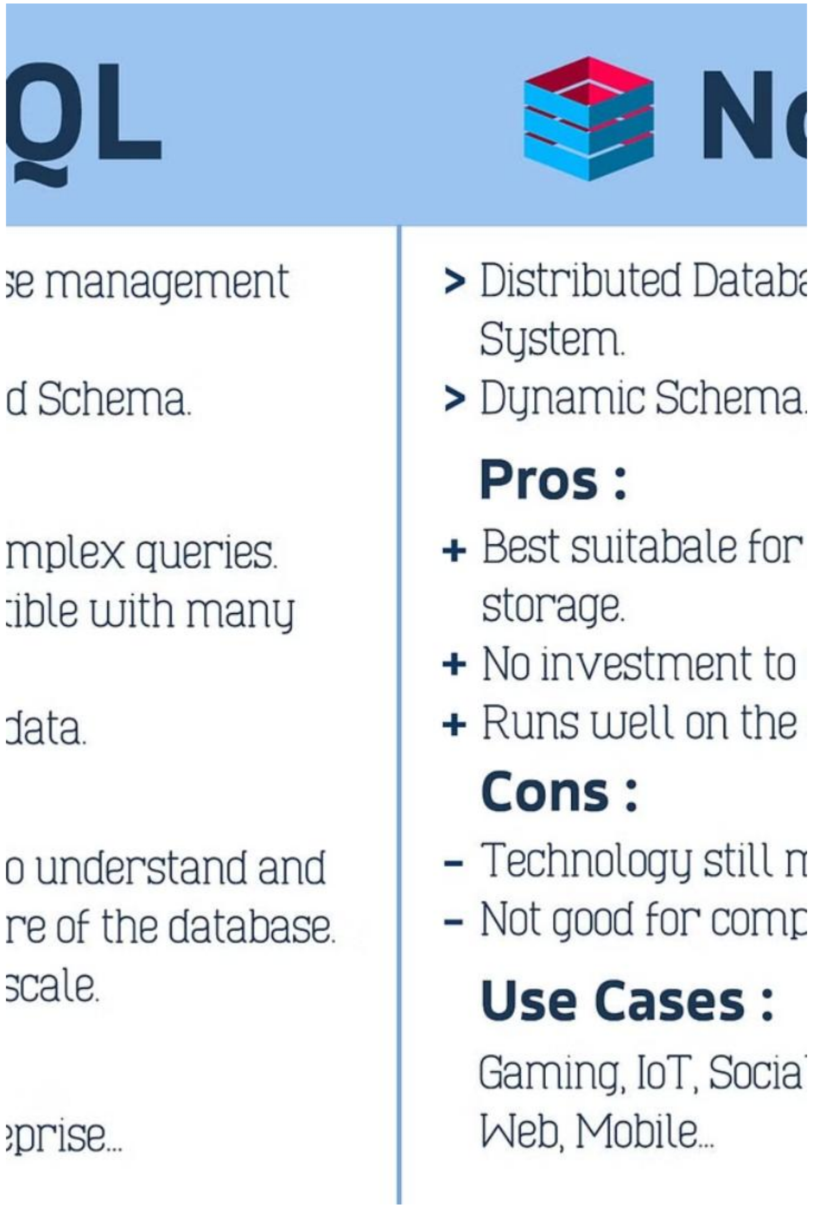
Use Cases

Semi-structured data, high scalability, real-time performance

Key Characteristics:

- **Schema flexibility**: Data models can evolve without major redesign.
- **Horizontal scalability**: Designed to scale across many distributed nodes.
- **Use-case specialization**: Each NoSQL type is suited to a specific pattern (e.g., key-based access, full-text search).

Types and Benefits of NoSQL Databases



QL

No

se management

d Schema.

mplex queries.
ible with many

data.

o understand and
re of the database.
scale.

prise...

> Distributed Databa
System.

> Dynamic Schema.

Pros :

+ Best suitabale for
storage.
+ No investment to
+ Runs well on the

Cons :

- Technology still m
- Not good for comp

Use Cases :

Gaming, IoT, Socia
Web, Mobile...

Flexibility

The data model can be changed or extended easily as applications grow.

Scalability

Distributed architecture enables seamless scaling by adding more machines.

Performance

Each type is optimized for specific access patterns and use cases.

Functionality

Rich data types and structures are supported (e.g., nested documents, graphs).

Common Types of NoSQL Databases:

**Key-Value Stores

What Are NoSQL Databases?

Definition

NoSQL databases, meaning "Not Only SQL," refer to **non-relational databases** designed for handling **semi-structured or unstructured data**.

These databases are purpose-built for specific data models and typically **do not require fixed schemas**, making them suitable for evolving applications.

Differences from Relational Databases

Unlike relational databases, NoSQL systems:

- **Do not rely on table-based design**
- Are commonly used in **modern, large-scale, real-time applications**
- Can store data in forms such as JSON, key-value pairs, or graphs

Key Advantages and Types of NoSQL Databases

✓ Benefits of NoSQL Databases:

- **Flexibility**: Data structures can evolve without schema redesign
- **Scalability**: Distributed architecture enables horizontal scaling across nodes
- **Performance**: Optimized for specific data access patterns and workloads
- **Functionality**: Specialized types support complex use cases like document search or graph traversal

📚 Types of NoSQL Databases:

- **Key-Value Stores** (e.g., Redis, DynamoDB): Fast retrieval using a unique key
- **Document Stores** (e.g., MongoDB, CouchDB): Store JSON-like documents with nested data
- **Graph Databases** (e.g., Neo4j): Designed to manage interconnected data efficiently
- **Column-Family Stores** (e.g., Cassandra, HBase): Handle wide and sparse datasets
- **Search Databases** (e.g., Elasticsearch): Optimized for full-text indexing and retrieval

JSON in NoSQL: A Flexible Data Format

JSON Definition

JSON (JavaScript Object Notation) is widely used in NoSQL databases

Schema-less Design

Allows fields to be added, removed, or modified without altering the entire dataset

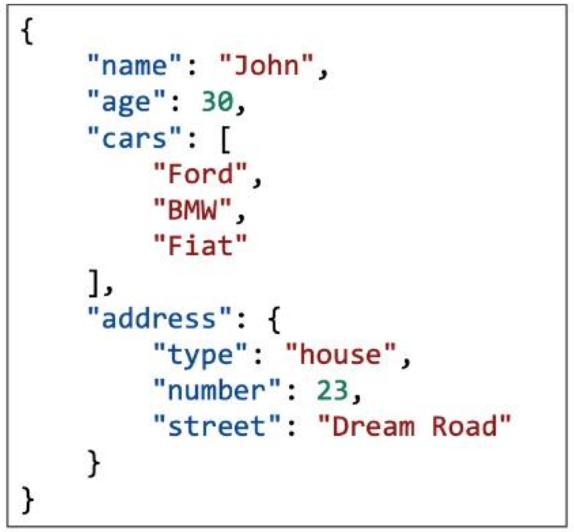
Supported Features

Nested structures, arrays, and dynamic fields

JSON supports a **schema-less design**, allowing fields to be added, removed, or modified without altering the entire dataset.

The data format supports:

- **Nested structures**: Hierarchical representation through embedded objects
- **Arrays**: Collections of values such as lists
- **Dynamic fields**: New attributes can be introduced over time without redefinition



```
{
    "name": "John",
    "age": 30,
    "cars": [
        "Ford",
        "BMW",
        "Fiat"
    ],
    "address": {
        "type": "house",
        "number": 23,
        "street": "Dream Road"
    }
}
```

Managing Databases with AWS: Shared Responsibility Model

AWS Responsibilities

AWS is responsible for infrastructure-level tasks such as:

- Operating system patching
- Automated backups and restores
- High availability and fault tolerance
- Performance monitoring and alerting

Customer Responsibilities

Customers are responsible for:

- Data management and schema design
- Access control and security configuration
- Query optimization and cost efficiency

✅ Benefits of AWS-managed databases:

- Rapid provisioning and **automated scaling** (both vertical and horizontal)
- Minimal operational overhead for **backup, patching, and upgrades**
- **Built-in monitoring and alerting** tools (e.g., Amazon CloudWatch)

⚠️ **Note**: When databases are deployed on EC2 instances directly, full responsibility is transferred to the user for managing resiliency, backup strategies, high availability, patching, and fault recovery mechanisms.

Amazon RDS: Managed Relational Database Service

What is Amazon RDS?

Amazon RDS (Relational Database Service) is a **fully managed service** that facilitates the setup, operation, and scaling of **relational databases in the cloud**.

SQL Support

This service supports **SQL-based databases**, allowing structured data to be queried using standard SQL syntax.

Automated Management

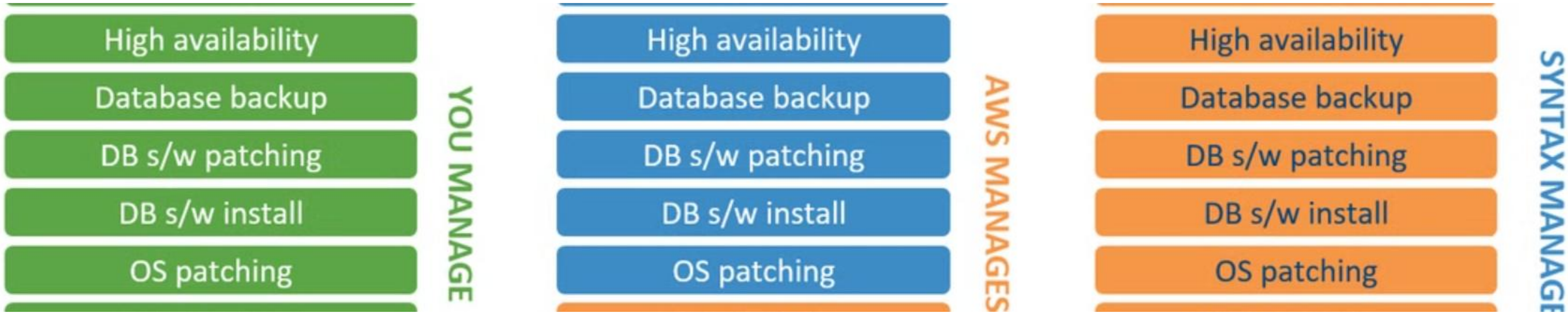
Administrative tasks such as provisioning, backups, patching, and recovery are **automatically handled by AWS**, reducing operational overhead.

🛠️ Supported Database Engines:

MySQL MariaDB ORACLE Microsoft SQL Server IBM Db2

This service is widely adopted for applications requiring **high availability**, **security**, **automatic failover**, and **cost-efficient scalability**.

Why Amazon RDS Is Preferred Over EC2 for Database Management



Automated Provisioning & OS Patching

Database setup, resource allocation, and operating system maintenance are performed automatically, reducing manual work.

Continuous Backups & Point-in-Time Restore

Backups are taken automatically and can be restored to any specific point, helping recover from data loss or corruption.

Monitoring & Dashboards

Tools such as **Amazon CloudWatch** allow performance, availability, and resource metrics to be tracked and visualized without external monitoring setup.

Read Replicas for Load Distribution

Replicas can be created to serve read requests, reducing the load on the primary database and improving performance in read-heavy applications.

Additional Benefits of RDS + Limitations Compared to EC2 Hosting

🔒 Enhanced Availability & Reliability:

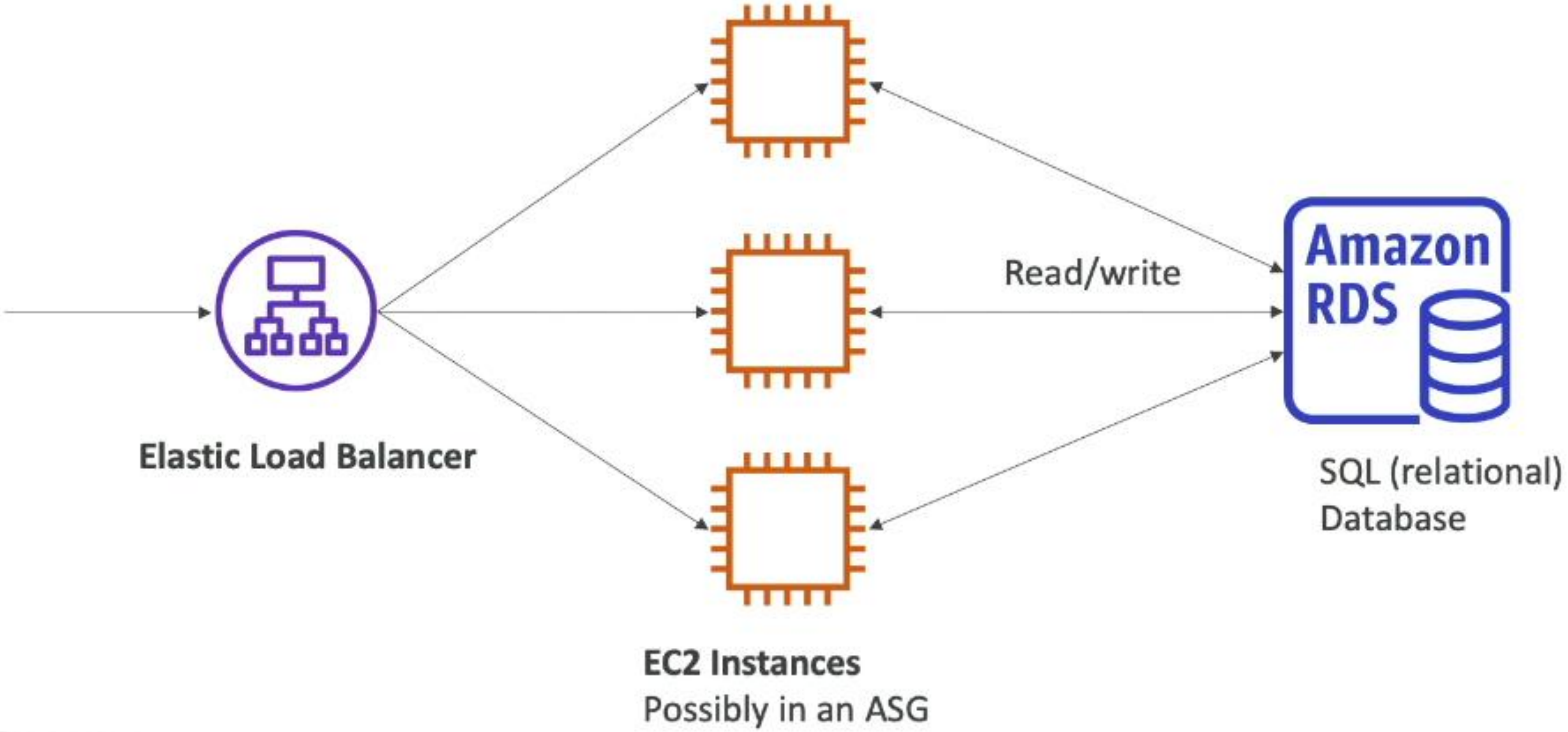
- **Multi-AZ (Availability Zone) Support for Disaster Recovery** Standby replicas in separate zones ensure database continuity during outages.
- **Maintenance Windows for Upgrades** Scheduled time slots are used for system upgrades without disrupting peak usage.
- **Scalability (Vertical and Horizontal)** Resources (CPU, memory, IOPS) can be adjusted without downtime; read replicas allow horizontal scale.
- **EBS-backed Storage** Storage is persistent, reliable, and elastic through Amazon Elastic Block Store.

⚠️ Limitation to Consider:

No Direct OS Access (No SSH) Unlike EC2-based databases, RDS does not allow access to the underlying operating system, limiting low-level customizations.

✅ **Conclusion**: RDS is preferred when managed, scalable, and resilient database infrastructure is needed without the operational overhead of server administration.

How RDS Integrates into a Scalable Cloud Architecture



How RDS Integrates into a Scalable Cloud Architecture

Elastic Load Balancer (ELB)

- Acts as the **entry point** for incoming application traffic (e.g., from users or clients).
- Distributes this traffic **evenly** across multiple EC2 instances.
- Helps ensure **high availability and fault tolerance** by rerouting requests if one instance fails.

EC2 Instances (in Auto Scaling Group – ASG)

- These are the **compute servers** that run the application code (e.g., website backend, microservices).
- Each instance **processes business logic** and then interacts with the database layer.
- Auto Scaling Groups (ASG) can be used to **automatically add or remove EC2s** based on traffic load.

Amazon RDS (Relational Database Service)

- This is the **central data storage component** where structured data is **stored, managed, and queried**.
- All EC2 instances are connected to RDS and **perform read/write operations** using SQL.
- Data is stored **in tables**, and features like **automatic backups, replication, and scaling** are provided by RDS.

Amazon Aurora: A High-Performance Cloud-Native Database

5x

MySQL Performance

Performance improvement over traditional RDS MySQL

3x

PostgreSQL Performance

Performance improvement over traditional RDS PostgreSQL

128 TB

Maximum Storage

Auto-scaled in 10 GB increments

20%

Cost Premium

More expensive than standard RDS

Amazon Aurora is a **proprietary relational database engine** developed by AWS, designed for **cloud-native optimization**.

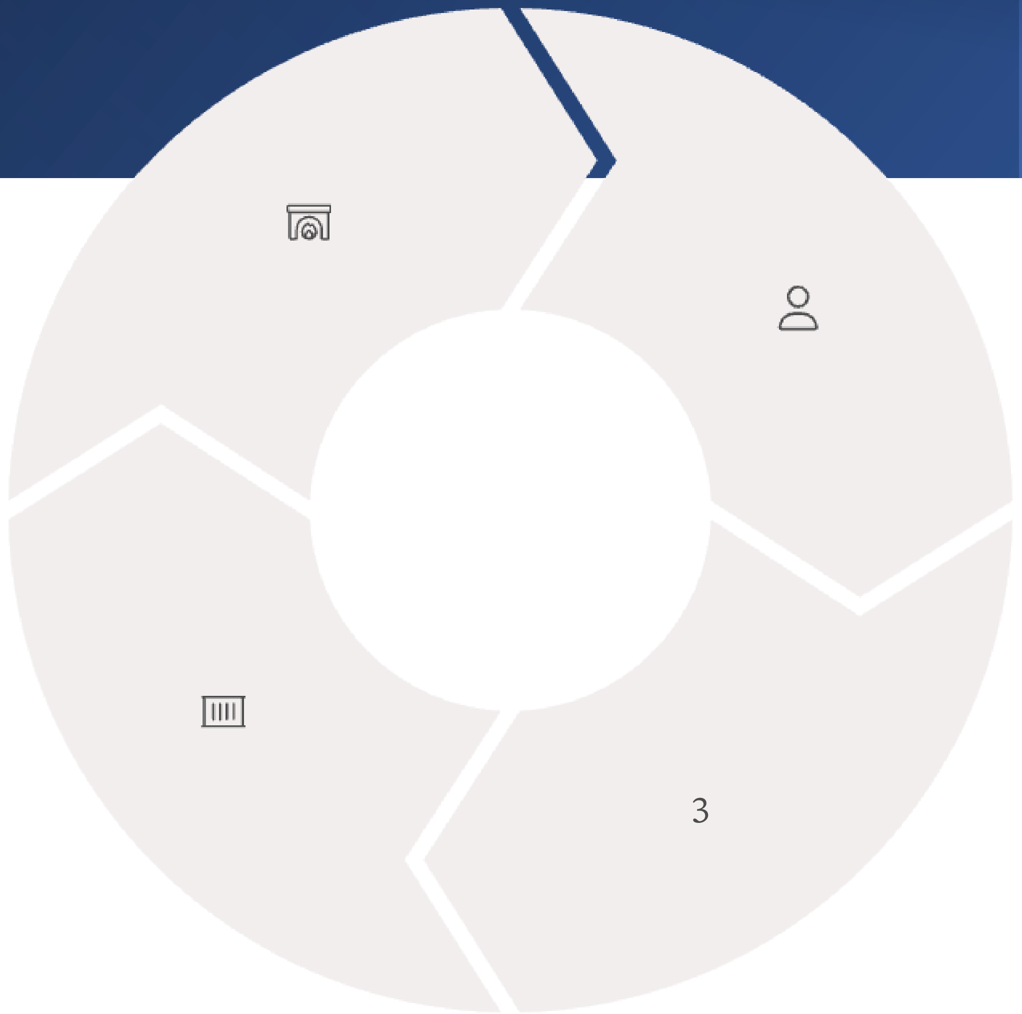
Aurora is compatible with both **MySQL** and **PostgreSQL**, allowing applications built on those engines to migrate with minimal changes.

Storage is auto-scaled, growing in **10 GB increments** up to **128 TB**, which helps eliminate manual storage provisioning.

Aurora is **not included in AWS Free Tier**, and charges begin immediately upon use.

✅ Aurora is recommended for high-throughput, low-latency applications requiring enterprise-grade performance with managed scalability.

Amazon Aurora Serverless: Auto-Scaling Database for Dynamic Workloads



Client Application

Connects to the proxy fleet

Proxy Fleet

Managed by Aurora, routes requests intelligently

Shared Storage

Automatically grows as more data is stored

Aurora Instances

Scale automatically based on workload

Aurora Serverless is a version of Amazon Aurora that **automatically starts up, shuts down, and scales** the database based on actual application usage.

It supports both **MySQL** and **PostgreSQL**, but does **not require manual provisioning** of servers.

Instead of running all the time, the database **activates only when needed**—saving cost and effort.

✅ Key Benefits:

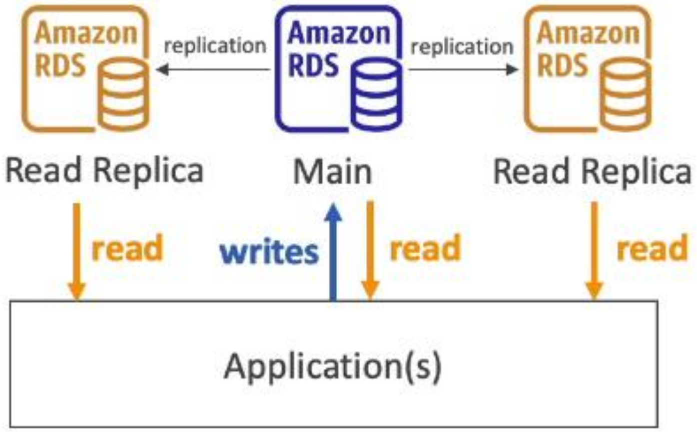
- No need to plan for CPU or memory—Aurora adjusts resources for you.
- **Pay per second** of usage—ideal for apps that run **occasionally** or at **unpredictable times**.

RDS Deployment Options: Read Replicas vs. Multi-AZ

🔁 Read Replicas – For Performance Scaling:

- Designed to **reduce the load on the main database** by allowing multiple replicas to serve read-only traffic.
- Applications can **send read queries** to any replica while all **write operations go only to the main database**.
- Up to **15 read replicas** can be created for one RDS instance.
- Data changes in the main DB are **automatically copied (replicated)** to each read replica.

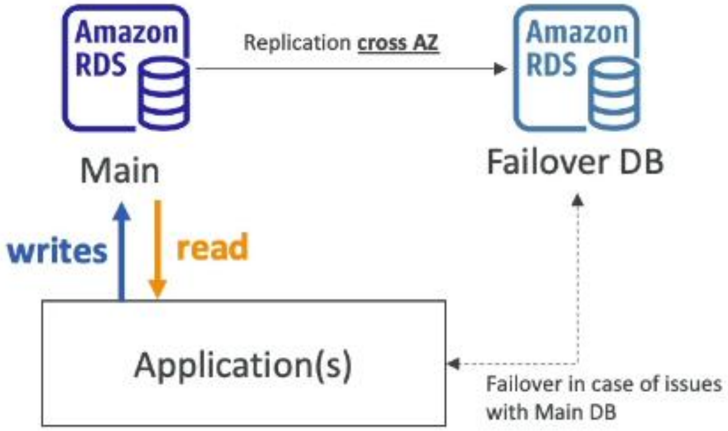
🧠 **Use Case**: Best for apps with high read traffic (e.g., dashboards, analytics, reports).



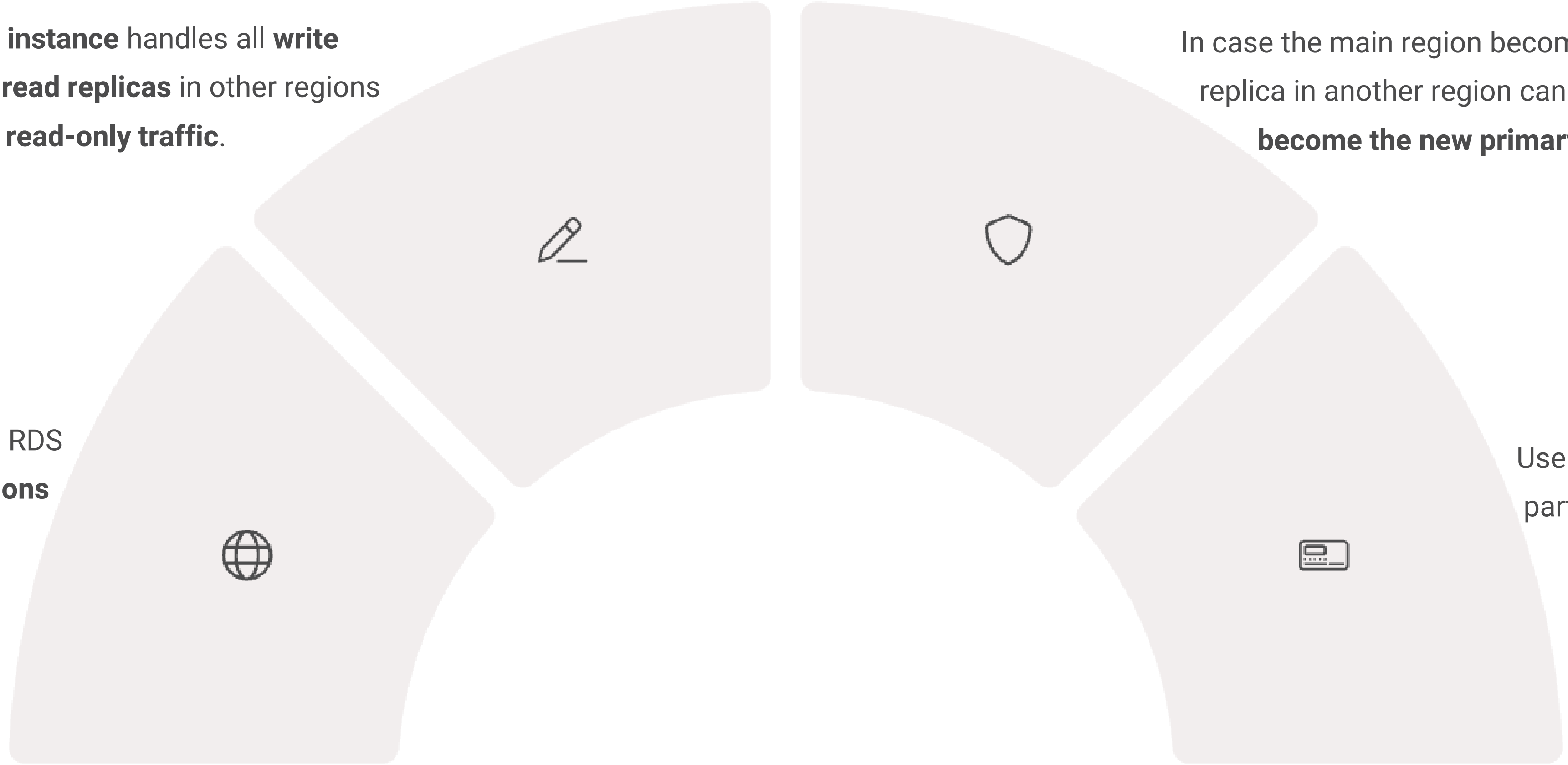
🛡 Multi-AZ Deployment – For High Availability:

- Provides **automatic failover** in case the main database becomes unavailable (e.g., due to an outage in one Availability Zone).
- The standby (failover) database is **not used for reads or writes** under normal conditions—it only activates during failure.
- Replication occurs **synchronously** across Availability Zones to ensure real-time backup.

🧠 **Use Case**: Ideal for critical production workloads requiring **continuous availability** and **disaster recovery**.



RDS Multi-Region Deployment: Global Read Replicas



Write Operations

The **main RDS instance** handles all **write operations**, while **read replicas** in other regions serve **read-only traffic**.

Disaster Recovery

In case the main region becomes unavailable, a replica in another region can be **promoted to become the new primary** database.

Multi-Region Setup

A **Multi-Region RDS setup** allows Amazon RDS to be deployed across **different AWS regions** around the world using **read replicas**.

Local Performance

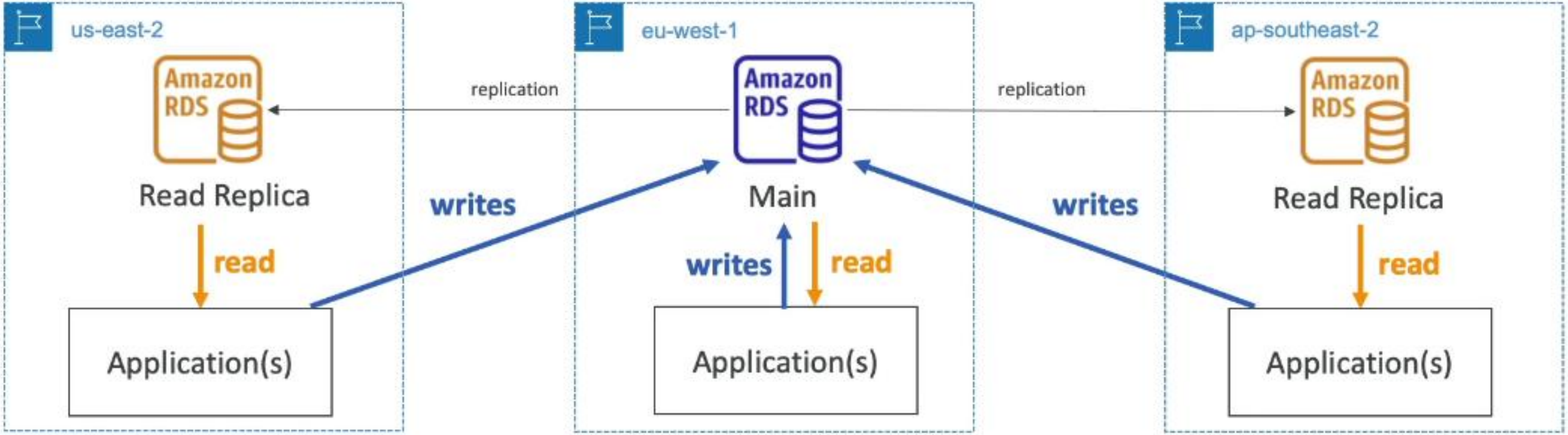
Users and applications located in different parts of the world can **read data from the nearest replica**, reducing latency.

⚠️ Things to Consider:

- **Writes still go to one region only**—this means **multi-region write support is not provided** in standard RDS (not multi-master).
- **Replication latency** and **extra costs** are involved due to data being copied across regions.

✅ Best for global apps needing fast reads from anywhere and protection against regional outages.

RDS Multi-Region Deployment: Global Read Replicas



Amazon ElastiCache: High-Speed In-Memory Caching

What is ElastiCache?

ElastiCache is a fully managed caching service provided by AWS, similar to how **Amazon RDS manages relational databases**.

It supports popular in-memory caching engines like **Redis** and **Memcached**.

Caches store data in **memory (RAM)** instead of on disk, providing **very fast data access** with **low latency**.

🚀 Key Benefits

- **Boosts application performance** by serving frequently requested data from memory.
- **Reduces pressure on primary databases**, especially for **read-heavy workloads** like search, analytics, and user sessions.
- Ideal for **temporary, frequently accessed** information (e.g., shopping cart data, leaderboard scores).

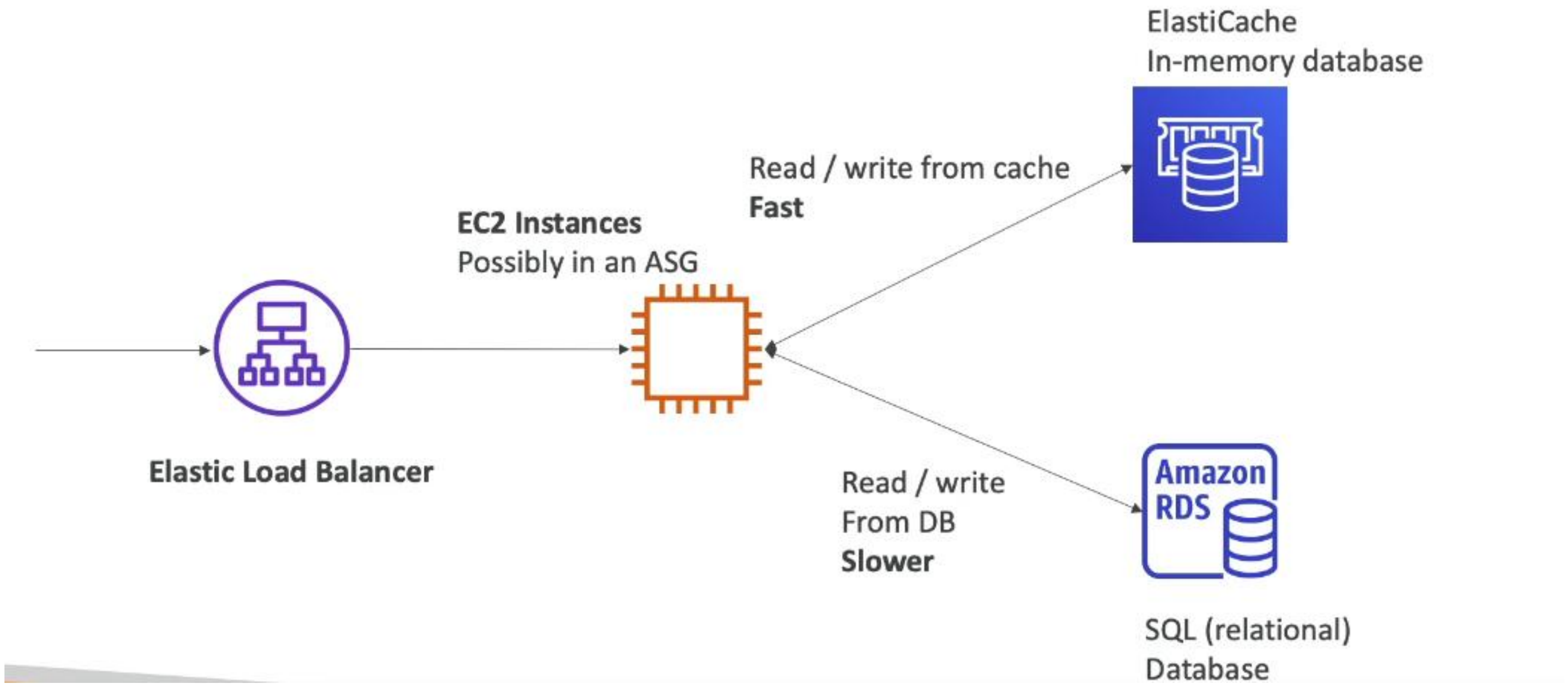
🛠️ Managed by AWS

AWS handles:

- Operating system patching and maintenance
- Configuration and monitoring
- Performance tuning
- Failure recovery and automatic backups

✅ ElastiCache is recommended when low-latency, high-throughput data access is needed to support scalable applications.

Amazon ElastiCache: High-Speed In-Memory Caching



ElastiCache
In-memory database

Read / write from cache
Fast

EC2 Instances
Possibly in an ASG

Elastic Load Balancer

Read / write
From DB
Slower

SQL (relational)
Database

How ElastiCache Works with RDS in Application Architecture

1

Elastic Load Balancer

Distributes user traffic to multiple **EC2 instances** (application servers).

2

EC2 Instances

These EC2 instances handle application logic and make decisions on where to fetch or write data.

3

ElastiCache

If the data is **frequently accessed**, it is retrieved from **ElastiCache** (an **in-memory database**) for **faster performance**.

4

Amazon RDS

If the data is **not cached** or needs to be persisted, it is accessed from **Amazon RDS** (a **relational SQL database**), which is **slower** due to disk-based storage.

🧠 Why Use ElastiCache in This Setup?

- **Speed**: Memory is significantly faster than disk, so accessing ElastiCache results in **low latency**.
- **Efficiency**: Reduces the number of queries sent to RDS, helping it handle more users with less load.
- **Scalability**: Ideal for high-throughput applications where data is read repeatedly (e.g., user sessions, leaderboard scores).

✅ **Summary**: ElastiCache improves speed, reduces load on the main database, and increases scalability in high-demand environments.