| **Type** | **Use Case** | **Key Features** |
| --- | --- | --- |
| Relational (SQL) | Structured enterprise data | ACID, joins, schema-based |
| NoSQL | Big data, real-time apps | Schema-less, scalable |
| In-Memory | Caching, fast access | RAM-based, ultra-fast |
| Distributed | Global apps, fault-tolerant systems | Resilient, scalable, replicated |
| Cloud | Web/mobile apps | On-demand, cost-effective, scalable |
| Graph | Social networks, fraud detection | Relationship-centric, graph traversal |

| **Purpose** | **Tech Stack** | |
| --- | --- | --- |
| SQL Databases | MySQL, PostgreSQL | |
| NoSQL Databases | MongoDB, Firebase | |
| In-memory Cache | Redis | |
| ORM for Python | SQLAlchemy, Django ORM | |
| Cloud Platforms | AWS RDS, Firebase, GCP | |
| Admin Tools | | PgAdmin, Robo 3T |

**Types of Databases with Real-time Use Cases :**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Category** | **Type** | **Data Format** | **Use Case** | **Example Technologies** | **Real-World Companies** |
| 🗃️ **Relational DB (RDBMS)** | SQL-based (Structured) | Tables (rows/columns) | Banking systems, Inventory, CRM | MySQL, PostgreSQL, Oracle, MS SQL Server | Netflix, Instagram (PostgreSQL), Amazon (Aurora) |
| 🚀 **Non-Relational DB (NoSQL)** | **Document** | JSON/BSON documents | Content management, IoT, product catalogs | MongoDB, CouchDB, Firebase Firestore | eBay, Lyft (MongoDB), The New York Times |
|  | **Key-Value** | Dictionary-like key-value pairs | Session storage, caching, rate limiting | Redis, DynamoDB, Memcached | GitHub (Redis),  Twitter (Memcached), Snapchat (DynamoDB) |
|  | **Column** | Rows + dynamic columns | Time-series data, analytics, recommendation | Apache Cassandra, HBase, ScyllaDB | Facebook (Cassandra), Uber (Cassandra), Netflix (Scylla) |
|  | **Graph** | Nodes + relationships | Social networks, fraud detection, recommendations | Neo4j, ArangoDB, Amazon Neptune | LinkedIn (graph search), Pinterest, PayPal (fraud detection) |

Comparison Table :

|  |  |  |
| --- | --- | --- |
| **Feature** | **Relational (RDBMS)** | **Non-Relational (NoSQL)** |
| Structure | Fixed schema (SQL) | Flexible schema (JSON, KV, etc.) |
| Scalability | Vertical (scale-up) | Horizontal (scale-out) |
| Best For | Complex queries, ACID | Big data, flexible or nested data |
| Example Query | SELECT \* FROM users WHERE age > 25 | db.users.find({"age": {"$gt": 25}}) |
| Transaction Support | Full ACID compliance | Varies by NoSQL type |
| Flexibility | Low | High (schema-less) |
| Performance | Slower at scale | Extremely fast for reads/writes |

Implementation by Use Case :

|  |  |  |
| --- | --- | --- |
| **1Use Case** | **Suggested DB** | **Reason** |
| User Profiles with login | **MongoDB / Firebase** | Schema flexibility, nested documents |
| Realtime Chat | **Redis / Firebase** | Low-latency pub/sub or streaming |
| Product Inventory | **MySQL / PostgreSQL** | Structured tabular data |
| Recommendation Engine | **Neo4j** | Graph traversal between users/products |
| IoT Sensor Data | **Cassandra / MongoDB** | High-write, time-series data |
| Payment Systems | **PostgreSQL / Oracle** | Strong consistency, ACID compliance |
| Session Storage | **Redis** | Fast, key-based access with TTL |
| Analytics Dashboard | **Cassandra + Spark** | Distributed, fast reads at scale |

**Mongo Atlas:**

# pip install pymongo (requirements)

import streamlit as st

from pymongo import MongoClient

from bson.objectid import ObjectId

MONGO\_URI = "mongodb+srv://<username>:<password>@cluster0.mongodb.net/myDB?retryWrites=true&w=majority"

client = MongoClient(MONGO\_URI)

db = client["user\_db"]

collection = db["users"]

st.set\_page\_config(page\_title="MongoDB CRUD with Streamlit", layout="centered")

st.title("MongoDB CRUD App using Streamlit")

def user\_form(default\_data=None, form\_key="form"):

    with st.form(key=form\_key):

        name = st.text\_input(

            "Name", value=default\_data["name"] if default\_data else "")

        email = st.text\_input(

            "Email", value=default\_data["email"] if default\_data else "")

        age = st.number\_input("Age", min\_value=0, max\_value=120,

                              value=default\_data["age"] if default\_data else 18)

        submitted = st.form\_submit\_button("Submit")

        return submitted, {"name": name, "email": email, "age": age}

st.subheader("Add New User")

submitted, user\_data = user\_form(form\_key="create\_user\_form")

if submitted:

    collection.insert\_one(user\_data)

    st.success("User added successfully!")

st.subheader("Existing Users")

users = list(collection.find())

if not users:

    st.info("No users found.")

else:

    for user in users:

        col1, col2, col3 = st.columns([3, 1, 1])

        with col1:

            st.markdown(

                f"\*\*{user['name']}\*\* ({user['email']}, Age: {user['age']})")

        with col2:

            if st.button("Edit", key=f"edit\_button\_{user['\_id']}"):

                with st.expander(f"Edit {user['name']}", expanded=True):

                    edit\_submitted, updated\_data = user\_form(

                        user, form\_key=f"edit\_form\_{user['\_id']}")

                    if edit\_submitted:

                        collection.update\_one({"\_id": user["\_id"]}, {

                                              "$set": updated\_data})

                        st.success("User updated! Please refresh.")

        with col3:

            if st.button("Delete", key=f"delete\_button\_{user['\_id']}"):

                collection.delete\_one({"\_id": user["\_id"]})

                st.warning(f"🗑️ Deleted {user['name']}. Please refresh.")

**Redis :**

**Redis (REmote DIctionary Server)** is a **NoSQL in-memory key-value database**, best known for being **super fast**.

1. Stores data in RAM for ultra-fast reads/writes
2. Like a Python dictionary — keys with values
3. Supports Strings, Lists, Sets, Hashes, Sorted Sets
4. Keys can expire (great for caching/session management)
5. Caching, leaderboards, pub/sub, session storage, queues