

Example: online store

The **online store** will specialize in custom **furniture**, allowing users to select designs, materials, and dimensions. The platform will have various features, such as

- **Product Information,**
- **User Profiles,**
- **User Sessions,**
- **Shopping Carts,**
- **Purchase history,**
- **Activity Logs,**
- **Personalized recommendations.**

Parts of the project

- Product information – **RDBMS** - PostgreSQL (alternatively: **Document** - MongoDB),
- User accounts – **RDBMS** - PostgreSQL
- User sessions – **Key-value** - Redis,
- Shopping carts – **Key-value** - Redis,
- Purchase history – **Document** - MongoDB (alternatively: **Wide Column** - Cassandra),
- Activity logs – **Wide Column** - Cassandra,
- Personalized recommendations – **Graph** - Neo4j,
- Caching frequently accessed data – **Key-value** - Redis.

The primary product information: RDBMS

Purpose: Store the primary product information, such as product names, descriptions, prices, and stock quantities.

Tasks to Solve:

- Storing and retrieving product information.
- Managing inventory and stock levels.
- Supporting searches and filters for products based on various criteria.

Document: arguments

- **Schema flexibility:** Easily accommodates diverse furniture items with varying attributes without rigid structure.
- **Nested data:** Naturally represents complex furniture specifications and variations within a document.
- **Faster queries:** Retrieves complete product information in one operation, reducing joins.
- **Scalability:** Better handles large volumes of product data and high read loads.
- **Rapid development:** Allows quicker iterations and updates to the product catalog structure.

MongoDB might be preferred if the furniture products have highly variable attributes and the schema needs to be flexible.

RDBMS : arguments

- **Structured data:** Ideal for uniform furniture information (dimensions, materials, prices).
- **Relationships:** Efficiently represents connections between products, categories, and attributes.
- **Data integrity:** Ensures accuracy and consistency of product information.
- **Query flexibility:** Enables complex search and filtering of furniture items.
- **ACID compliance:** Guarantees reliable transactions for inventory and order management.

RDBMS with JSON: arguments

- Different types of furniture may have different customization parameters.
 - For example, a table might have a tabletop shape option, while a sofa might have an upholstery type. JSON allows storing various parameters for different products without changing the database schema.
- Combining a **constant structure** for basic data and a **variable part in JSON format** provides a balance between stability and flexibility.
- JSON Types in PostgreSQL
<https://www.postgresql.org/docs/16/datatype-json.html>

User Session Management : Key-value

Purpose: Store **user session data** for quick access and high performance.

Tasks to Solve:

- Storing user session tokens.
- Managing temporary user data (e.g., items in a shopping cart before checkout).
- Maintaining session persistence across multiple requests

Key-value: arguments

- **In-Memory Storage**

- It offers high-speed read/write operations, ideal for session management.

- **Data Expiration**

- Supports TTL (Time-to-Live), automatically expiring sessions after a certain period.

- **Scalability**

- Can handle numerous concurrent user sessions with ease

- **Quick Authentication Support**

- Enables fast user authentication processes due to low latency data access

Shopping Carts: Key-value

Purpose: To provide a temporary storage system for selected custom furniture items, facilitating the purchase process.

Tasks to Solve:

- Data Persistence:
 - Store cart items and customizations
 - Maintain cart across sessions and devices
- Cart Operations:
 - Handle rapid item modifications
 - Set expiry for inactive carts
- Performance:
 - Fast retrieval of cart contents
 - Quick calculation of cart totals
- Data Integrity:
 - Ensure consistency during simultaneous modifications .

Key-value: arguments

- Fast read/write operations for real-time cart updates
- Supports atomic operations for managing cart items
- Can easily handle temporary data with built-in expiration
- Scalable for managing multiple concurrent carts

Purchase history: Wide column/Document

Tasks to Solve:

- Record management:
 - Log, store, and access purchase history.
- Data organization:
 - Sort and filter orders by various parameters.
- Product information:
 - Link to item details and enable reordering.
- Order tracking:
 - Monitor current order status and handle returns.

Wide column : arguments

- Scalability: Excels at handling large volumes of historical data across many users.
- Efficient querying: Optimized for time-series data and range queries, standard in order histories.
- Performance: Faster reads for specific columns, beneficial for frequently accessed data like order status or dates.
- Time-based partitioning: Natural fit for historical data organized by timeframes.

Document: arguments

- Flexibility: Easily accommodate varying order structures without schema changes.
- Rich querying: Advanced query capabilities on nested data structures.
- Intuitive data model: Orders naturally fit into the document structure.
- Ease of development: Simpler to work with for many developers.

Wide-Column vs Document Database

Wide-Column Database

- Better for: High-volume time-series data, scalability, efficient range queries
- Use case: Large e-commerce platforms with millions of orders

Document Database

- Better for: Complex, varying order structures, rich querying needs
- Use case: Smaller to medium-sized platforms with frequently changing order attributes

Activity Logs: Wide-Column

Tasks to Solve:

- Event Management
 - Record and store all significant user and system actions
 - Enable quick log retrieval and analysis
- Security and Compliance
 - Protect log data and ensure adherence to legal standards
 - Set up alerts for critical events and integrate with monitoring systems
- Data Lifecycle Management
 - Manage log retention, archiving, and disposal efficiently
 - Provide tools for pattern recognition and reporting

Wide-Column : arguments

- Optimized for write-heavy workloads typical of logging
- Can handle high volumes of time-stamped events
- Allows for efficient querying of recent user activities
- Scalable for long-term storage of user behavior data

Personalized Recommendations : Graph

Tasks to Solve:

- Data storage
 - Organizing information as graphs with nodes and relationships
- Querying and analysis
 - Performing complex queries based on data relationships
- Scalability
 - Handling large volumes of data and high loads
- Integration
 - Interacting with other systems and data analysis tools

Graph : arguments

- A graph database is ideal for modeling complex relationships
- Efficient for traversing connections between users, products, and preferences
- Supports advanced recommendation algorithms
- Allows for real-time personalization based on user behavior

Parts of the project

- Product information – **RDBMS** (PostgreSQL),
- User accounts – **RDBMS** (PostgreSQL),
- User sessions – **Key-value** (Redis),
- Shopping carts – **Key-value** (Redis),
- Purchase history – **Document** (MongoDB),
- Activity logs – **Wide column** (Cassandra),
- Personalized recommendations – **Graph** (Neo4j),
- Caching frequently accessed data – **Key-value** (Redis).

System architecture

