Assignment 5 Solutions

Wiam Skakri

November 18, 2025

1 Question 1

1.1 Question 1(a)

Analysis of the Options:

Option 1 represents a CP (Consistency + Partition Tolerance) system.

Explanation: By stopping acceptance of new orders during the network partition, the system prioritizes consistency over availability. This ensures that when the system does accept orders, all regions have consistent data and no conflicting updates occur. The system tolerates the partition (P) but sacrifices availability (A) by refusing service during the partition.

Option 2 represents an AP (Availability + Partition Tolerance) system.

Explanation: By continuing to accept orders independently in both regions during the partition, the system prioritizes availability over consistency. Both US-East and US-West remain available and responsive to customer requests. The system tolerates the partition (P) but sacrifices consistency (C) because the two regions may accept conflicting orders that will need to be reconciled later, leading to temporary inconsistencies.

1.2 Question 1(b)

Matching Transaction Behaviors:

(i) The inventory update (T1) and payment (T2) in a single transaction must both succeed or both fail.

Answer: **ACID** - This describes the *Atomicity* property of ACID, where a transaction is treated as a single unit that either completely succeeds or completely fails. This all-or-nothing guarantee is fundamental to ACID systems.

(ii) During the partition, a customer might see an outdated inventory count, but the system eventually corrects it.

Answer: **BASE** - This demonstrates *Eventual Consistency*, a key characteristic of BASE systems. The system remains available during the partition (Basically Available) and allows temporary inconsistencies (Soft state) that are resolved over time (Eventual consistency).

(iii) The order database allows a "soft state" where temporary inconsistencies are acceptable for higher availability.

Answer: **BASE** - This explicitly describes the *Soft state* principle of BASE systems, where the system accepts that data may be in a temporary inconsistent state to maintain availability and partition tolerance.

(iv) Once the partition heals, replicas reconcile to the same final state.

Answer: **BASE** - This describes the *Eventual Consistency* aspect of BASE systems. After network partitions are resolved, the system reconciles conflicting updates across replicas to eventually reach a consistent state, even though consistency was not guaranteed during the partition.

2 Question 2

2.1 Question 2(a)

Partitioning by Hash Function:

Given:

- Hash function: $H(key) \mod 4$
- 4 servers with IDs: 0, 1, 2, 3
- $H(\text{"product:A10"}) = 10$
- $H(\text{"user:1001:cart"}) = 5$

Calculations:

For **"user:1001:cart"**:

$$\begin{aligned} H(\text{"user:1001:cart"}) \mod 4 &= 5 \mod 4 \\ &= 1 \end{aligned}$$

Answer: **Server 1** will store "user:1001:cart"

For **"product:A10"**:

$$\begin{aligned} H(\text{"product:A10"}) \mod 4 &= 10 \mod 4 \\ &= 2 \end{aligned}$$

Answer: **Server 2** will store "product:A10"

2.2 Question 2(b)

Relational Database Schema Translation:

Based on the key-value store examples, here are the relational tables that represent the same information:

Table 1: Users

userID (PK)	**name**	**email**
1001	Alice	alice@shop.com

Table 2: Products

pid (PK)	name	price	stock
A10	Wireless Mouse	24.99	58
B25	Mechanical Keyboard	79.50	30

Table 3: Carts

cartID (PK)	userID (FK)
1	1001

Table 4: CartItems

cartID (FK)	pid (FK)	quantity
1	A10	2
1	B25	1

Primary Key: (cartID, pid)

Table 5: Orders

orderID (PK)	userID (FK)	total	status
9001	1001	129.48	shipped

Table 6: OrderItems

orderID (FK)	pid (FK)
9001	A10
9001	B25

Primary Key: (orderID, pid)

Key Differences:

- The NoSQL key-value store uses nested JSON structures to represent relationships
- The relational model normalizes data into separate tables with foreign key relationships
- The relational model enforces referential integrity through primary and foreign key constraints
- The key-value approach allows flexible schema and denormalization for faster reads

2.3 Question 2(c)

Data Lookups Required to Retrieve Shopping Cart and Compute Total Cost:

To retrieve the customer's shopping cart and compute the total cost, the following data lookups are needed:

Step 1: Retrieve the Shopping Cart

```
GET user:1001:cart
```

This returns:

```
{ "items": [ {"pid": "A10", "qty": 2}, {"pid": "B25", "qty": 1} ] }
```

Step 2: Retrieve Product Information for Each Item

For each product ID (pid) in the cart items, we need to fetch the product details to get the price:

```
GET product:A10
```

Returns: `{ "name": "Wireless Mouse", "price": 24.99, "stock": 58 }`

```
GET product:B25
```

Returns: `{ "name": "Mechanical Keyboard", "price": 79.50, "stock": 30 }`

Step 3: Compute Total Cost

Using the retrieved data:

$$\begin{aligned}\text{Total} &= (\text{price of A10} \times \text{qty of A10}) + (\text{price of B25} \times \text{qty of B25})\\ &= (24.99 \times 2) + (79.50 \times 1)\\ &= 49.98 + 79.50\\ &= 129.48\end{aligned}$$

Summary of Data Lookups:

1. `GET user:1001:cart` - Retrieve the cart to get the list of items and quantities
2. `GET product:A10` - Retrieve product A10 details (including price)
3. `GET product:B25` - Retrieve product B25 details (including price)

3 Question 3

3.1 Question 3(a)

Which statement is true about column-oriented stores?

Answer: **C. They are well-suited for analytical queries that aggregate over a few columns.**

Explanation: Column-oriented stores excel at analytical queries that need to aggregate over a few columns (e.g., SUM, AVG, COUNT) because they only need to read the relevant columns from disk, not entire rows. This minimizes I/O and improves query performance for OLAP workloads.

3.2 Question 3(b)

How are values of a column typically stored in a columnar store?

Answer: **B. Together, often compressed, as a contiguous block**

Explanation:

In column-oriented stores, all values for a single column are stored together in a contiguous block on disk. This storage pattern enables:

3.3 Question 3(c)

Which query benefits most from a column-oriented store?

Answer: **C. SELECT AVG(Salary) FROM Employees WHERE Department = 'Engineering';**

Explanation:

This query benefits most from column-oriented storage because:

- It only needs to access **two columns**: Department and Salary
- It performs an **aggregation** (AVG) over a subset of rows
- The column store can read only the Department and Salary columns, ignoring all other employee attributes
- Column compression can significantly reduce the amount of data read from disk

3.4 Question 3(d)

Data Read from Disk Calculation:

Given:

- Table: Sales(Date, CustomerID, ProductID, Amount)
- Each field: 8 bytes
- Total rows: 1 million
- Query: SELECT SUM(Amount) FROM Sales;

(1) Row-oriented store:

In a row-oriented store, all fields of each row are stored together. To compute SUM(Amount), we must read entire rows:

$$\begin{aligned}\text{Bytes per row} &= 4 \text{ fields} \times 8 \text{ bytes/field} = 32 \text{ bytes}\\ \text{Total rows} &= 1,000,000\\ \text{Total data read} &= 32 \times 1,000,000 = 32,000,000 \text{ bytes}\\ &= 32 \text{ MB}\end{aligned}$$

Answer: **32 MB must be read from disk**

(2) Column-oriented store:

In a column-oriented store, each column is stored separately. We only need to read the Amount column:

$$\begin{aligned}\text{Bytes per Amount value} &= 8 \text{ bytes}\\ \text{Total rows} &= 1,000,000\\ \text{Total data read} &= 8 \times 1,000,000 = 8,000,000 \text{ bytes}\\ &= 8 \text{ MB}\end{aligned}$$

Answer: **8 MB must be read from disk**

Performance Advantage:

The column-oriented store reads **4 times less data** (8 MB vs 32 MB), demonstrating the significant I/O advantage of columnar storage for analytical queries that access only a subset of columns.

4 Question 4

4.1 Question 4(a)

(i) All "users" documents in a collection users must have the exact same fields and types.

Answer: **FALSE**

Explanation: Document databases like MongoDB and CouchDB support **schema flexibility** (also called "schema-less" or "dynamic schema"), meaning documents within the same collection can have different fields and field types. This allows new fields to be added to some documents without affecting others, which is a key advantage over traditional relational databases with rigid schemas.

(ii) The cart field inside the user document shows that document databases support nested structures within a single record.

Answer: **TRUE**

Explanation: Document databases excel at storing hierarchical and nested data structures. The user document contains a `cart` field that is itself a complex structure (an array of objects), where each cart item has nested fields like `pid` and `qty`. This demonstrates **embedding** or **denormalization**, where related data is stored together in a single document rather than requiring separate tables and joins as in relational databases.

(iii) When the price of product A10 changes, all historical orders automatically reflect the new price.

Answer: **FALSE**

Explanation: Document databases do not automatically update embedded data when the source changes. The `orders` collection stores the price (24.99) embedded within each order document, so when the price in the `products` collection changes, existing orders retain their original embedded price values. This is actually desirable behavior for orders since it preserves the historical price at the time of purchase rather than updating it retroactively.

(iv) A MongoDB-style query like `db.users.find({ "cart.pid": "B25" })` will return all users who have product B25 in their cart.

Answer: **TRUE**

Explanation: MongoDB and similar document databases support querying nested fields using **dot notation**. The query `"cart.pid"` allows MongoDB to search through the `cart` array in each user document and match any document where at least one item in the cart array has `pid: "B25"`. This query would return the user with id 1001, since their cart contains `{"pid": "B25", "qty": 1}`, demonstrating the power of document databases to query complex nested structures without requiring joins.