Fall 2025
Y.Wu

CSDS 341: Introduction to Databases
Assignment #5

Due: Nov 18, 2025 (11:59 PM ET)

All assignments in this course are to be digitally produced and submitted to the canvas as pdf.

This assignment contains some practice that are optional (*), not counted as credits – for you to try and play with installed SQL systems.

1. [NoSQL Concepts] (25).

(a) [CAP] Recall the three components of CAP theorem: Consistency, Availability, and Partition Tolerance. Consider an online retail database system that runs across multiple data centers. The system is replicated across two regions: US-East, and US-West. The system experiences a temporary **network partition**, where servers in US-East cannot communicate with those in US-West temporarily. There are two options:

- Option 1: System stops accepting any new orders until the communication is restored.
- Option 2: Both systems continue to accept new orders independently, syncing data later.

Which option represents a CP system, and which represents an AP system? Explain why.

(b) [ACID vs. BASE]. The retail company considers replacing its traditional ACID-compliant SQL system with a BASE-style NoSQL system to improve scalability. Match the following transaction behaviors to ACID or BASE principles. An example is shown for the first question.

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

(ii) During the partition, a customer might see an outdated inventory count, but the system eventually corrects it.

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

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

2. [NoSQL Key-value Store] (25). As an intern at this online retailer, you are designing a small shopping cart and inventory tracking system for their online shopping service. You used a key-value store such as Redis. Some examples are provided below.

Key	***Value (JSON-like structure)***
user:1001:cart	{ "items": [{"pid": "A10", "qty": 2}, {"pid": "B25", "qty": 1}] }
product:A10	{ "name": "Wireless Mouse", "price": 24.99, "stock": 58 }
product:B25	{ "name": "Mechanical Keyboard", "price": 79.50, "stock": 30 }
order:9001	{ "user": 1001, "items": ["A10", "B25"], "total": 129.48, "status": "shipped" }

(a) Suppose the database is partitioned by hashing the key to server IDs. Let the hash function be H(key) mod 4 --- there are 4 servers with IDs 0,1,2,3.
Assume the hash values "H(product:A10)" = 10, H("user:1001:cart")=5. Which server would store the value of "user:1001:cart" and "product:A10", respectively?

(b) Translate this key–value data into a set of relational tables (with table names, columns, and tuples) that would represent the same information in a relational database system. Useful schema are provided: Users (userID, *other attribute*); Products (pid, name, price, stock); Carts (cartID, userID); CartItems (cartID, pid, quantity); Orders (orderID, pid); OrderItems (orderID, pid). You can enrich Users with any relevant attributes and fill in any value that makes sense.

(c) *[optional practice]. Install Redis (command line, insights, or use Redis Cloud). Try create a key-value database with the above data. To retrieve the customer's shopping cart and compute total cost, what data lookups would you need to perform? Try the following script and observe the results.* ***You don't need to submit anything for this (c).***

GET user:1001:cart
For each pid in items:
GET product:A10
GET product:B25

3. [NoSQL Column Store] (25). Answer the following questions.

(a) Which of the following statements is true about column-oriented stores?
A. They store all attributes of a single record together on disk.
B. They are optimized for reading and writing entire rows at once.
C. They are well-suited for analytical queries that aggregate over a few columns.
D. They must use relational schemas.

(b) In a columnar store, how are values of a column typically stored?
A. Sequentially across rows
B. Together, often compressed, as a contiguous block
C. Randomly across multiple pages
D. As key-value pairs per tuple

(c) Which type of query benefits most from a column-oriented store?
A. SELECT * FROM Sales WHERE ID = 100;
B. UPDATE Users SET Password='new' WHERE ID = 5;
C. SELECT AVG(Salary) FROM Employees WHERE Department = 'Engineering';

(d) Suppose we have a table Sales (Date, CustomerID, ProductID, Amount). Each field is 8 bytes and there are 1 million rows. You run: SELECT SUM(Amount) FROM Sales; Approximately how much data (in bytes) must be read from disk in: (1) a row-oriented store, and (2) a column-oriented store?

4. [NoSQL Document Databases] (25). Your online retail system now uses a document-oriented database like MongoDB or CouchDB. Each document is a collection, similar to a table in relational systems. Some example documents:

```
Collection: users:
 {
  "_id": 1001,
  "name": "Alice",
  "email": "alice@shop.com",
 "cart": [
{ "pid": "A10", "qty": 2 },
 { "pid": "B25", "qty": 1 }
  ]
}
```

```
Collection: products
 {
  "_id": "A10",
  "name": "Wireless Mouse",
  "price": 24.99,
  "stock": 58,
  "category": "electronics"
}
```

```
Collection: orders
{
  "_id": 9001,
  "user": 1001,
  "items": [
    { "pid": "A10", "price": 24.99, "qty": 2 },
    { "pid": "B25", "price": 79.50, "qty": 1 }
  ],
  "total": 129.48,
  "status": "shipped"
}
```

(a) True or False? Please explain your answer.

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

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

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

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

(b)* *[optional practice]. Install MongoDB and practice the following queries that answer the questions and try run them in your MongoDB database. Observe the results.* ***You don't need to submit anything for this (b).***

- **Write a MongoDB-style query to find all orders where the total amount is greater than 100.**

  db.orders.find({ total: { $gt: 100 } })

- **Write an update to increase the stock of product A10 by 10.**

  db.products.updateOne({ _id: "A10" }, { $inc: { stock: 10 } })

- **Write a query to find all users who have pid: "B25" in their shopping cart.**

  db.users.find({ "cart.pid": "B25" })