Coding Interview Problems - SDE1/2/3

In-memory Relational Data Store

	SDE-1	SDE-2	SDE-3
Design quotient	Medium	Medium	Medium
Problem solving	Medium		
Modularity/readability			
Concurrency	N/A	Medium	Medium

Implement an in-memory relational data store
Features:

1. Create table
2. Delete Table
3. Insert row
4. Update row
5. Delete row

6. Create Index on a single column

SDE1 → Create, Insert, update (no locking), id based indexing (in memory)
SDE2 → SDE1 + delete (compaction, re-indexing)
SDE3 → SDE2 + concurrency (locks), alter table or constraints

Create Table
Create a store for a specific table. We can assume all fields of type string. Table name and column names should be persisted as metadata either on the table itself or on a separate entity.

```
createTable(<table name>, [Column list])
```

Insert Row
Generate unique ID for the to-be-created row. Add the row into the table and return the newly-created-row’s ID.

```
insertRow(<table name>, [value list in the same order of table
creation])
```

Update Row
Search for the row to be updated. Perform an in-place update with the columns to be updated with the new values. If a column need not be updated

```
updateRow(<table name>, [map of column name to updated value])
```

Delete Row

`deleteRow(<table name>, [map of column name to delete value])`

Create Index

`createIndex(<table name>, <column_name>)`

Interview Question:

Ask the candidate to implement an in-memory based DBMS. Application should take in normal SQLs as inputs and perform the respective actions. In the interest of time, SQL parsing can be skipped and the respective methods can be called directly.

```
CREATE TABLE transaction
Call #createTable(tableName) method to mimic the above

INSERT INTO TABLE transaction values()
Call #insertRow(tableName, map<key,value> rowData) method to mimic insert table.
```

These methods createTable and insertRow should implement the underlying DB (In-memory).

Evaluation Matrix

Expectations	SDE1	SDE2	SDE3
Requirements + Low Level Design			
Asks Relevant Questions		✓	✓
Narrowed down scope		✓	✓
Identified Multiple solutions and reasoned one solution		✓	✓
Identified edge cases			✓
Flow walk through			✓
Coding			
Optimum solution	✓	✓	✓

Readable Code	✓	✓	✓
Modular Code		✓	✓
Handles edge cases		✓	✓
Extensible Code			✓
Quality			
Showcases happy path scenario	✓	✓	✓
Showcases few edge case scenarios		✓	✓
Identify and fix issues in quick time			✓
Extensibility			
Inject new features with minimal code changes			✓
Ease of identifying and fixing issues after new features		✓	✓

Pub/Sub Queues

Implement a persistent pub-sub queue mechanism with guaranteed delivery of every published message for all the subscribed consumers in that subscribed topic in the same order.

Features:

1. API to expose topics
2. API for publisher to push messages against a topic
3. API to subscribe and unsubscribe from topic
4. API for subscriber to consume from topic
5. Maintain the state of consumption of each message in each topic for each consumer.
6. Maintain order of message consumption for each consumer.
7. Handle cases of consumers not being available. What to do with those messages? (Write to disk and restore?)
8. Filtered message consumption in a topic by the consumer

Plus Points:

1. Separate service or component that is responsible for topics, including creation and exposing topics

2. Writing messages to disk and restoring them back when the queue is full, due to consumer unavailability.
3. Maintaining copies of messages for serving data faster.

SDE1 → Features #1-#4
SDE2 → SDE1 + Features #5-#6
SDE3 → SDE2 + Featyres #7-#8

Note: This problem will be good for SDE2 and SDE3 but very complex for SDE1. So, parking it for now

Implementing auto-sharding on a given database

Given database, list of tables and sharding criteria (number of splits, criteria for split, sharding driver), write an application to achieve the below

Features:

1. Create new database(s) with the exact schema based on the number of splits
2. Create a reference DB as a router. This should have a map of driver criteria to the shard DB.
3. Extract to-be-split data using driver defined
4. Identify and extract related table entries
5. Move the data across to the shards.
6. How to ensure uniqueness is maintained across DBs? Eg: ID in one shard should or should not be present in a different shard
7. How and where to insert new data? How to decide where to insert new data?

Document Processor / Text Editor

Features:

1. Create multi-lines content of data
2. Search and replace.
3. Cursor and its actions such as navigate up / down / left / right.
4. Page Down / Page Up.
5. Home / End

Excel sheet Implementation

Implement a single excel sheet of data

Features:

1. Print entire excel sheet
2. Update value of a cell
3. Get value of a cell
4. Add Basic formulae such as add, subtract, multiply
5. Add range operations such as sum() over range.
6. Conditional functions based on other cells
7. Formatting of each cells
8. Conditional formatting of cells

Implement Web-Server

Features:

1. Implement a subset of HTTP.
2. Should support GET, POST, PUT.
3. HTTP router which should support path based routing. (order, precedence for routes) {Regex based solution is acceptable}
4. Should be extensible so that different content-type handling can be implemented by implementing some interface.
5. Should be able to serve static content.

Note: Prior knowledge of socket not required.

Implement Photo-sharing App

Features:

1. Users should be able to post photos.
2. Method to fetch all photos of a userID.
3. Method to fetch all the followers.
4. Method to fetch all the userID user is following.
5. Photos can contain tags to other userID.

6. Feed
    a. A user can follow other users.
    b. Function to get all photos for his feed. (All photos of users this user is following)

New Problems

Simple document Service

simple-document-service [problem for machine coding round]

Rating Service

Rating Service

In memory Search Engine

In-memory Search engine

Logger Library

Design a logger library which prints log messages synchronously or asynchronously to different targets(such as file, database, some url) based on log level (debug, info, error,warn).One target can be tied to multiple log levels.The logger has to read configuration file during system startup and initialize.
Configuration can look like

```
{
        target: db,
loglevel: info,
endpoint: localhost,
Port:3306,
timestampformat: "dd-mm-yy"
},

{
        target: file,
loglevel: debug,
      filelocation: /var/lib/sample.log,
        timestampformat: "dd-mm-yy"
}
```

Printed logs will contain timestamp, string content, log level, name of application which generated the logs.*Code should be runnable.