CS 186 Introduction to Database Systems
Fall 2020 Alvin Cheung, Aditya Parameswaran DIS 13

1 OLTP vs OLAP

For each workloads, choose whether it's better characterized as Online Transaction Processing (OLTP) or Online Analytical Processing (OLAP).

(a) A social media site with millions of users needs to track all the "likes" and "dislikes" that each post receives.

OLTP: OLTP workloads involve high numbers of transactions executed by many different users. Queries in these workloads involve simple lookups more often than complex joins. In this case, when a user likes or dislikes a post, the site would need to do a lookup on that post and update its likes/dislikes value.

(b) An online book store needs to aggregate and analyze its users book purchases by genre over the last eight months.

OLAP: OLAP workloads involve read-only queries and typically include lots of joins and aggregations. Often, workloads executed for analysis and decision making are OLAP workloads. In this case, the book store needs to read from data stored over the last eight months regarding book purchases, and may need to perform some joins to analyze those purchases by genre.

(c) An multiplayer online game has added updated areas to its map and now wants to assess how users behave in those areas, and how user playtime has changed as a result.

OLAP: In this case the game designers will need to read data that likely includes which players logged in after the updated areas released, their average playtimes, and any other data that may be relevant to how players behave in the new area.

2 Scaling

(a) A small startup realizes that its current database can't sustain their growing workloads. Given that these workloads involve lot of write but few reads, should it invest in more partitioning or more replication?

Partitioning: more partitioning means that more queries can execute in parallel. This is especially good for write-heavy workloads because these can often involve updates to only a few machines, meaning more of them can happen at once.

(b) A mechanical failure causes some of the startup's database machines to permanently crash, losing data in the process. If the startup wants to prevent similar losses in the future, should it invest more in partitioning or more replication?

Replication: replication means each database machine is no longer a single point of failure. Once data is replicated, the entire system is more resilient to data loss because if one machine crashes, its data can be recovered from the replicas.

3 BASE

(a) Database designer Doug is annoyed with his distributed database because for some time after issuing a write, all his reads return different values. Does this violate any of the BASE properties?

No. This is in fact the soft state and eventual consistency properties in action. As the write propagates through the system, reads may return different values because the database is inconsistent.

Which properties of BASE do these scenarios violate?

(b) All reads and writes always have the same views of data, but they sometimes respond to valid inputs with timeout errors.

Basic Availability. This database seems to prioritize consistency over availability. Since valid inputs sometimes receive an error response, it violates basic availability.

(c) Writes only propagate to 3 replicas, but the system has 5 replicas of each piece of data.

Eventual Consistency. Since writes only propagate to 3 replicas, different pieces of data will always be stale on different replicas, so reads will never be consistent.

(d) An empty database that has never been populated responds to a read query on some specific key with the message "Error: key nonexistent!"

It violates no properties. The basic availability property only guarantees that valid queries always get a non-error response. However, since the database is empty, the query was invalid, so the error response is OK.

4 Key-Value Stores

Database Doug now has the following tables:
Sales (sid, date, quantity, customer, product)
Product (pid, name, price)
Customer (cid, name, address)
Sales data is stored with Key = sid, Value = entire Sales record, partitioned on hash function h and replicated across 3 servers.

(a) Describe how the operation get(sid1) would be executed. (Assume a Sale with sid1 exists in the data).

We must first hash the key, h(sid1), to find which partition the data is stored on. Then we can retrieve the value from any of the replicas/servers.

(b) Describe how the operation put(sid2, saleRecord) would be executed.

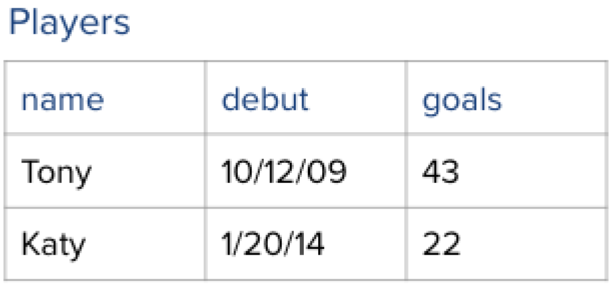
We must first hash the key, h(sid2), to find which partition the data should be stored on. Then, we will insert the record to that partition, and propagate the change to the other replicas/servers. Note: propagation of changes may not happen immediately, we only need to enforce eventual consistency.

(c) After put(sid2, saleRecord) is executed, is it guaranteed that every app will be able to access that new Sale data?

No. Since we're only enforcing eventual consistency, the changes from the put operation may not have propagated to all replicas yet. Note: It is possible to perform checks to see whether the replica an app is pulling data from is up-to-date, if the app requires non-stale data.

5 JSON

1. Convert the following relational table into a JSON document.



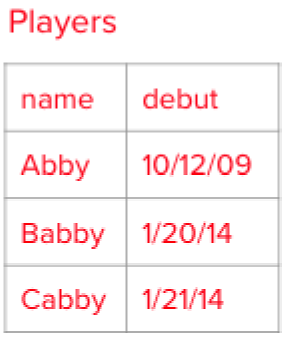
Players

name	debut	goals
Tony	10/12/09	43
Katy	1/20/14	22

```
{“players”: [
  {“name”: “Tony”, “debut”: “10/12/09”,
    “goals”: 43},
  {“name”: “Katy”, “debut”: “1/20/14”,
    “goals”: 22}
  ]
}
```

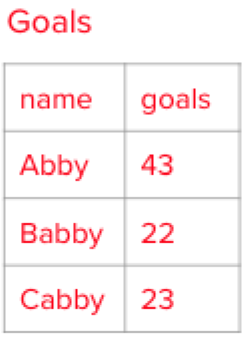
2. Convert the following JSON document into two relational tables, Players(name, debut) and Goals(name, goals).

```
{“players”: [
  {“name”: “Abby”, “debut”: “10/12/09”,
    “goals”: 43},
  {“name”: “Babby”, “debut”: “1/20/14”,
    “goals”: 22},
  {“name”: “Cabby”, “debut”: “1/21/14”,
    “goals”: 23}
  ]
}
```



Players

name	debut
Abby	10/12/09
Babby	1/20/14
Cabby	1/21/14



Goals

name	goals
Abby	43
Babby	22
Cabby	23

6 Mongo Query Language (MQL)

For the entire question, consider the MongoDB collection **teams** with the following fields:

- teamId (int)
- divisionId (int)
- stadiumCapacity (int)
- wins (int)
- losses (int)
- coach (string)
- captain (string)

1. Using MQL, write a query to fetch the following: Find the **coach** and **captain** of all teams from **division** 1 with at least 10 **wins**, sorted by **coach** DESC and ties broken by **captain** ASC.

```
db.team.aggregate([
    {$match: {
        wins: {$gte: 10},
        divisionId: 1
    }},
    {$sort: {
        "coach": -1,
        "captain": 1
    }},
    {$project: {
        "coach": 1,
        "captain": 1,
        "_id": 0
    }}
])
```

2. Translate the following SQL query into an MQL query:

```
SELECT divisionId AS div, MAX(wins) AS maxWins
FROM team WHERE stadiumCapacity >= 20000
GROUP BY divisionId
SORT BY MAX(wins), COUNT(*) DESC;
```

```
db.teams.aggregate([
    { $match: {
        stadiumCapacity: {$gte: 20000}
    }},
    { $group: {
        _id: "$divisionId",
        maxWins: {$max: "$wins"},
        count: {$sum: 1}
    }},
    { $sort: {
        "maxWins": 1,
        "count": -1
    }},
    { $project: {
        div: "$_id",
        _id: 0,
        maxWins: 1
    }}
])
```