FAKULTA ELEKTROTECHNICKÁ

České vysoké učení technické v Praze

B4M36DS2 – Database Systems 2

Practical Class 3

NoSQl: Basic Principles

Sharding, Replication, CAP theorem

Yuliia Prokop

prokoyul@fel.cvut.cz, Telegram **@Yulia_Prokop**

ČVUT FEL CourseWare Wiki https://cw.fel.cvut.cz/b241/courses/b4m36ds2/start

Exercise 1 - Data Sharding

Split data across multiple computers for improved performance.

Use the cz_users.csv file with online store user data.

1. Count users per city, total users, and identify the most popular cities.

2. Divide users into three groups using two methods:

Method A (Geographic): Group the 12 cities into **3 shards** by **geographic proximity** (nearby cities together). Goal**:** roughly equal number of users in each shard.

Method B (Range-based): Group by user_id ranges (1-8, 9-16, 17-24)

3. Create a table showing user counts per group and compare distribution evenness and query efficiency.

Exercise 1 - Data Sharding - Solution

1) Explore the data

- **Total users:** 24
- **Users per city (city_code → count):**
  PRG **5**, BRN **4**, OST **3**, PLZ **3**, LIB **2**, OLO **1**, CBU **1**, HKR **1**, PAR **1**, ZLN **1**, UST **1**, KVA **1**
- **Most popular cities:** PRG (5), BRN (4), then OST/PLZ (3 each)

2) Make two 3-way splits

Method A — Geographic (nearby cities together)

Choose three regional shards (West / Central–North / Moravia–East):

- **Shard A (West/NW):** PRG, PLZ, KVA, UST, ZLN → **11** users (5+3+1+1+1)
- **Shard B (Central/North):** LIB, HKR, CBU, PAR → **5** users (2+1+1+1)
- **Shard C (Moravia/East):** BRN, OLO, OST → **8** users (4+1+3)

Method B — Range-based (by user_id)

Use equal ranges: **1–8**, **9–16**, **17–24**.

Because user_id runs 1…24 without gaps, each shard has **8** users.

Exercise 1 - Data Sharding – Solution - 2

3) Compare results

Method	Shard / Rule	Users
Geographic	A: PRG, PLZ, KVA, UST, ZLN	11
Geographic	B: LIB, HKR, CBU, PAR	5
Geographic	C: BRN, OLO, OST	8
Range-based	1: user_id 1–8	8
Range-based	2: user_id 9–16	8
Range-based	3: user_id 17–24	8

Evenness

- **Geographic:** imbalanced (**11 / 5 / 8**) because PRG is heavy and grouped with nearby cities.
- **Range-based:** perfectly even (**8 / 8 / 8**) by construction (equal ID ranges).

Exercise 1 - Data Sharding – Methods Comparison

Method A: Geographic Sharding

Shard 1 (West/NW)
PRG (5), PLZ (3)
KVA (1), UST (1), ZLN (1)
11 users

Shard 2 (Central/North)
LIB (2), HKR (1), CBU (1), PAR (1)
5 users

Shard 3 (Moravia/East)
BRN (4), OLO (1), OST (3)
8 users

Method B: Range-based Sharding

Shard 1
User IDs: 1-8
8 users (33%)

Shard 2
User IDs: 9-16
8 users (33%)

Shard 3
User IDs: 17-24
8 users (33%)

Load Distribution Visualization

Analysis & Trade-offs

Geographic Sharding:

- ✓ Great for city-based queries
- ✓ Data locality (related data together)
- ✱ Can be balanced with load-aware regional grouping
- ✱ Requires monitoring & rebalancing as populations shift

Range-based Sharding:

- ✓ Even buckets (with equal ranges)
- ✓ Easy user lookups by ID
- × City queries need all shards
- ✱ No geo locality (higher cross-shard latency)

Recommendation: Range-based as a default; choose Geographic for latency-sensitive, city-scoped workloads.

Exercise 1 - Data Sharding – Solution - 3

Query efficiency (who is better for which queries?)

- **City-based queries** (e.g., "all users in PRG"):

  Geographic wins – PRG lives entirely in **Shard A**, so the query hits **one shard**.

  Range-based loses – a city's users are spread across ranges, so the query fans out to **several shards**.

- **Key lookups by `user_id`:**

  Range-based is trivial – each ID range maps to **one shard**.

  Geographic also hits one shard **if you know the user's city**; otherwise, you need a tiny directory (city → shard).

Exercise 1 - Data Sharding – Solution - 4

Two common ways to place new users

A) Geographic sharding (by city)

- The app keeps a tiny **map**: `city → shard`.
- New user from **PRG**? Put them in the shard for **PRG**.
- If one city grows too fast, we either:
  - **Move a nearby city** to a neighboring shard (to even things out), or
  - **Split the big city** across two shards using a simple rule like "`hash(user_id) % 2`".

B) Range / ID-based sharding

- We split **user_id** into ranges (or small blocks called "chunks").
- New user with `id = 1034` goes to the shard that holds that range/block.
- If the "last range" keeps getting all new users, we:
  - **Split the range** into two smaller ones and **move one** to a lighter shard, or
  - Use **many small blocks** from the start and spread them evenly.

Exercise 1 - Data Sharding – Solution - 5

Advise:

- **Sharding = buckets.** Keep buckets from getting too full.
- **Geographic sharding** is great for city questions, but you may need to **rebalance cities**.
- **Range/ID sharding** is great for evenly sized buckets, but watch out for the **“new IDs all go here”** problem.
- **Small, frequent moves** are better than big, rare ones.
- You can rebalance **without downtime**: copy → double-write → switch.

Exercise 2 - Replication & Synchronization Issues

Identify issues that arise when storing identical data on multiple computers.

Scenario: Main database + read replica with synchronization delays.

1. Model Sync Problem: User #5 changes city from LIB to PRG at 12:00, and another user reads at 12:01.

- Fill the table showing what the user sees with different sync delays.

2. Assess Data Staleness Impact: Rate criticality (High/Medium/Low) for scenarios:

- User viewing own profile
- City statistics calculation
- Product recommendations
- Bank balance checking

3. When is stale data acceptable vs. requiring fresh data?

Exercise 2 - Solution

Model the synchronization problem (main DB + read replica)

Scenario: User #5 changes city **LIB → PRG** at **12:00**. Another user reads User #5 at **12:01**.

Time to copy changes	What will the reader see at 12:01?	Is there a problem?	Why?
Instantly (0 s)	city = PRG	**No**	Replica is updated immediately
After 30 s	city = PRG	**No**	Update reached replica by **12:00:30**; the 12:01 read sees the new value
After 2 min	city = LIB (stale)	**Yes**	Replica lags until **12:02**; the 12:01 read still returns the old value

Exercise 2 – Solution - 2

Assess the impact of staleness

Situation	Criticality	Rationale
User views their own profile	High	Users expect read-your-writes: after they update something, the next page should reflect it. Otherwise, the product feels broken
Calculating general statistics by cities	Low	Aggregates change slowly; a single late update has a negligible effect. Batch jobs tolerate short lags
System recommends products	Medium	A stale city might reduce relevance (wrong geo-targeting), but it's usually tolerable for a short time
Checking bank account balance	High	Financial data must be accurate and current; stale reads risk overdrafts and user trust

Rule of thumb: if staleness can **confuse a user** or **cost money/compliance**, treat it as **High**.

Exercise 2 – Solution - 3

When stale data is OK vs. when fresh data is required

Stale data is acceptable (short delays are fine):

- Analytics and reporting
- Content/product recommendations
- Search results and ranking
- General statistics and dashboards

Fresh data is required (must reflect the latest state):

- Financial transactions and balances
- Authentication, session, and security decisions
- **User's own profile changes** (read-your-writes experience)
- Real-time inventory/seat/room availability

Exercise 3 - Consistency vs. Availability Trade-offs

Make decisions when perfect accuracy and constant availability conflict

Scenario: Two offices, same data copies, connection lost for 1 hour.

1. Identify read vs. write challenges during network partition.

2. Approach Comparison: Evaluate for different use cases:

- **Approach A:** Block all writes until the connection is restored
- **Approach B:** Allow writes, resolve conflicts later
- **Use cases:** E-commerce peak hours, bank transfers, social media

3. Develop criteria for approach selection

Exercise 3 - Solution

Problem Understanding (partition lasts)

- **Read operations:**
  - ✓ **Possible** (data exists locally) **but may be stale or divergent** between offices.
- **Write operations:**
  - ✓ **Risky** – concurrent writes can conflict.
- **Core dilemma:**
  - ✓ Block writes to keep data consistent (**CP**) vs. allow writes to stay online (**AP**) and fix conflicts later.

Exercise 3 – Solution - 2

Approach Comparison

Approach A — "Safe" (Block writes until link is back)

- ✓ Pros: Single source of truth; no divergent updates.
- ✓ Cons: Users can't change data; lost revenue / poor UX.

Approach B - " Available" (Allow writes; reconcile later)

- ✓ Pros: Keep business running; capture user intent and orders.
- ✓ Cons: Conflicts to resolve; temporary inaccuracies.

Exercise 3 – Solution - 3

Situation	Approach A "Safe"	Approach B "Available"	Recommended & Why
E-commerce during peak hours	Pros: No inventory mismatch Cons: Users can't checkout → lost sales	Pros: Stay open, accept orders Cons: Oversell risk; stock divergence	**B**, with guardrails. **Why:** revenue/UX wins short-term; add **reservations**, **idempotent order IDs**, **post-partition reconciliation** (cancel/refund backorders)
Bank transfers	Pros: No double-spend/incorrect balances Cons: Service unavailable	Pros: Service looks up Cons: Risk of overdrafts, regulatory issues	**A. Why:** financial correctness > availability; legal/compliance risk
Social media (posts/comments)	Pros: No dupes/out-of-order. Cons: Users can't post.	Pros: Users stay engaged Cons: Duplicates/order issues	**B**, plus **client-generated IDs**, **timestamps**, simple **merge rules** (e.g., last-writer-wins for likes; merge lists for comments)

Exercise 3 – Solution - 4

Selection Principles (no universal right answer)

Decide per feature using these criteria (top→down):

1. **Risk & impact of inconsistency vs. downtime** (money/compliance/safety > social > analytics)
2. **User expectation** (does the user expect changes to appear immediately – read-your-writes?)
3. **Regulatory / legal constraints** (finance/healthcare generally requires CP during partitions)
4. **Business model sensitivity to outages** (checkout/ordering often favors AP with guardrails)
5. **Conflict resolution feasibility** (can you merge safely? if not, favor CP)
6. **Operational reality** (how often/long are partitions? do you have monitoring & reconciliation tools?)
7. **Cost of rollback / correction** (cheap to fix → AP; expensive/irreversible → CP)

One-line rule of thumb:

- If inconsistency can **lose money, break law, or break trust**, choose **A (CP)**.
- If downtime is worse and conflicts are **cheap to fix**, choose **B (AP)** with guardrails.

Exercise 3 – Solution - 5

Network Partition: Two Offices Disconnected for 1 Hour

Office 1 (Prague) - - - ⚠ CONNECTION LOST ⚡/⛓⚒ - - - Office 2 (Brno)

During partition: reads may be stale/divergent; the choice affects writes

Approach A: "Safe" (Choose Consistency)

Strategy: Block writes until link is restored (CP)

Use Cases Analysis:

Bank Transfers
✓ No risk of double spending or incorrect balances

E-commerce Peak
× Lost sales, customer frustration
× Lost checkout; consider temporary maintenance/ read-only page

Social Network
⚠ Users can read but not post (read-only mode)

Approach B: "Available" (Choose Availability)

Strategy: Allow writes, reconcile later (AP)

Use Cases Analysis:

Bank Transfers
× Risk of overdrafts, accounting errors
× Regulatory risk - avoid this approach

E-commerce Peak
✓ Business continuity, no lost sales
💡 Use inventory reservation; cancel/refund backorders after heal

Social Network
✓ Users stay engaged, minor duplicates OK
💡 Client-generated IDs; merge rules

CAP During network partitions, you must choose between Consistency and Availability

✓ good • × risk • ⚠ requires caution
💡 = recommended practice / implementation tip

Exercise 4 - Distributed System Design

Design a fault-tolerant system applying all learned principles

Scenario: Online store system handling 1000 users, 10K orders/day, growing to 10K users/month. Must survive a single computer failure.

1. **Architecture Decisions**:
   - Choose computer count (1, 3, or 10) – evaluate pros/cons
   - Select data partitioning strategy (functional, geographic, range-based, or custom)
2. **Fault Tolerance**:
   - Identify single-point-of-failure risks
   - Design replication strategy (number of copies needed)
3. **Verification**: Test system with queries:
   - User viewing own orders
   - Cross-city statistics
   - Single computer failure scenario

Exercise 4 - Solution

Architecture Decisions

Computer Count Evaluation

Quantity	Pros	Cons	Suitable?
1 computer	Simple, cheap	Single point of failure	**No**
3 computers	Fault tolerant (survive 1 failure), manageable complexity, cost-effective	Limited headroom vs 10 nodes	**Yes**
10 computers	High scalability	Over-engineered for current needs	No

Choice: 3 computers — exactly 3 machines total. Meets "survive one failure" while keeping ops simple.

Exercise 4 – Solution - 2

Data Partitioning Strategy

Chosen Method: Range-based sharding

Thus, the user and their orders always live on the same shard.

- **Shard 1:** Users **1–333** (+ their orders)
- **Shard 2:** Users **334–666** (+ their orders)
- **Shard 3:** Users **667–1000** (+ their orders)

Routing rule: Load balancer routes by **user_id range** (not hash).

Rationale: Keeps user + orders together, one-hop lookups by user_id, easy to explain.

Exercise 4 – Solution - 3

Ensuring Reliability: Any one computer can fail with **no data loss** and **service stays up**.

Replication Model (Leader–Follower with quorums)

- **Replication factor N = 3** (each shard on **all three machines**).
- For each shard, one **Leader** (on a different machine per shard); the other two are **Followers**.
- **Write quorum W = 2**, **Read quorum R = 1 (or 2 for stronger reads) ⇒ R + W > N**.
  Writes are acknowledged after **the majority** persists; reads hit leader (R=1) or majority (R=2) when needed.

This is "synchronous to **the majority",** not "to all nodes".

Failure Handling

- **Automatic leader election** (consensus) per shard if a leader dies.
- **Target recovery time:** *SLO* **< 30s** (not a hard guarantee).

Remove Single Points of Failure

- **Load balancer:** 2 instances (active/active or active/standby) + health checks.
- Shared config/metadata store (if used): run redundant.

Exercise 4 – Solution - 4

Solution Verification

Query	Computers Queried	Performance
User views their orders	**1 shard** (range → leader)	**Fast** — single-shard lookup
Statistics for all cities	**3 shards**	**Fan-out** + aggregation (slower, but parallelizable)
One computer broke	**Still works** (majority alive)	Leader fails over; reads/writes continue with W=2/R=1(2)