LAB: Helpdesk Ticketing System with Redis (Cloud + Insight + Python)

Problem Statement

You are building a Helpdesk Ticketing System for an e-commerce brand. Customers create support tickets (order tracking, returns, technical issues). Agents are assigned based on skills and workload. Each ticket has a priority, status, an SLA due time, and an activity log (events like created, assigned, updated, closed).

You must design this system on **Redis** to achieve real-time performance and simple, scalable data modeling. You will:

1. **Host Redis in the cloud** using Redis Cloud and connect to it via **RedisInsight** (GUI).
2. **Choose correct Redis data structures** for each entity:
   - **HASH** for structured records (users, agents, tickets),
   - **SET** for agent skills and per-agent open ticket lists,
   - **ZSET** for a global priority queue of tickets,
   - **LIST** for append-only ticket activity logs,
   - **STRING** for counters and email→user indexes.
3. **Seed a realistic dataset** (users, agents, tickets with mixed priorities/statuses, logs).
4. Perform **CRUD** both manually in RedisInsight and programmatically with Python **(redis-py)**.
5. Enforce consistency rules:
   - If a ticket is assigned to an agent, it must appear in that agent's open_tickets set and increment their load.
   - Closed tickets must be removed from the global priority queue.
   - Email lookup must resolve to the correct user.

Your deliverables include screenshots (connection, keyspace, CLI outputs, Python run), and a short write-up explaining why you chose each Redis datatype and how you preserved consistency.

Part A — Setup (Cloud → Insight)

1. **Create a Redis Cloud database**

- Sign up at redis.com, create a free database.
- Note the Public Endpoint (host:port)**,** Username (often default), and Password. TLS is enabled by default.

2. **Connect with RedisInsight**

- Open RedisInsight → Add database → fill:
  - Host: YOUR_REDIS_HOST

    - Port: YOUR_REDIS_PORT
    - Username: default (or yours)
    - Password: YOUR_REDIS_PASSWORD
    - Enable TLS/SSL
- Save & connect. You should see Browser/CLI/Workbench.

Instruction #1 (mandatory): Take a screenshot of the successful RedisInsight connection and include it in your submission

Part B — Scenario & Data Model

Entities & Chosen Data Types

Entity	Key Pattern	Type	Fields / Members	Rationale
User	help:user:{id}	**HASH**	name, email, phone, joined_at	Structured attributes; partial updates
Email → UserId	help:idx:user_email:{email}	**STRING**	user_id	Fast lookup by email
Agent	help:agent:{id}	**HASH**	name, email, load	Update individual fields easily
Agent Skills	help:agent:{id}:skills	**SET**	e.g., returns, billing	Membership tests & intersections
Ticket	help:ticket:{id}	**HASH**	user_id, subject, status, priority, created_at, assigned_agent, sla_due_at	Ticket record
Ticket Log	help:ticket:{id}:log	**LIST**	strings like timestamp: action	Append-only event timeline
Global Priority Queue	help:queue:priority	**ZSET**	member=ticket:{id}, score=priority	Sort & pop highest priority
Agent's Open Tickets	help:agent:{id}:open_tickets	**SET**	ticket:{id}	Fast membership/listing
ID Counters	`help:seq:users	agents	tickets`	**STRING (int)**

Instruction #2 (mandatory): Use these data types. If you add more, include a short justification.

Part C — Required Scenarios

CREATE (5 tasks)

1. Create **5 users** + their email index keys.
2. Create **3 agents** + their **skills** sets.
3. Create **8 tickets** with priority mix (1/5/10/20).
4. Assign at least **3 tickets** to agents (update open_tickets + load).
5. Push **≥2 log entries** per ticket.

READ (6 tasks)

1. Resolve a user by email (index → user_id) and fetch their HASH.
2. Fetch **top-3 highest priority** tickets (ZREVRANGE or ZREVRANGE ... WITHSCORES).
3. List an agent's open tickets (SET) and show each ticket's subject.
4. Display a ticket's log timeline (LIST).
5. Compute counts of tickets by status (client-side tally).
6. Identify agents who cover **"returns"** by inspecting each skills SET.

UPDATE (6 tasks)

1. open → in_progress and log it.
2. in_progress → on_hold and log it.
3. Reassign a ticket A202 → A201 (adjust sets + load).
4. Extend sla_due_at (HSET).
5. Keep load consistent on every assign/unassign.
6. Fix a typo in subject.

DELETE (4 tasks)

1. **Business close**: set status=closed, ZREM from queue, remove from agent open_tickets, log "closed".
2. **Hard delete** one demo ticket (DEL ticket + log + ZREM).
3. Delete a user and its email index key.
4. Cleanup: use SCAN + pattern to remove lab keys (careful—no KEYS in production).

Instruction #6 (mandatory): For each CRUD category, show all the commands .

Part D — Validation / Integrity Checks

- **Urgent list:**
  ZREVRANGE help:queue:priority 0 4 WITHSCORES

- **Agent workload check:**
  HGET help:agent:A201 load (compare across agents client-side)
- **Consistency rules:**
  - Assigned ticket must appear in the agent’s open_tickets set.
  - Closed tickets must **not** appear in the ZSET queue.
  - Email index must resolve to the correct user.

Part E — Submission Checklist

1. RedisInsight connection screenshot
2. Key Browser snapshot showing your key patterns
3. CLI outputs for CRUD (screenshots)
4. Python run output screenshots
5. Short write-up (½–1 page):
   - Why HASH/SET/ZSET/LIST were the right choices
   - Trade-offs (e.g., ZSET vs LIST for priority)
   - How you kept data consistent (load, sets, queue)