Q.) Write a list of what you would do differently if this were run to collect data on 500 million repositories instead of 100,000.

The current design handles 100,000 repositories efficiently. At 500M scale, the following changes would be made:

1. Distributed workers with a message queue

A scheduler publishes tasks (“fetch repos 0 to 10,000”) to a queue (SQS/RabbitMQ). Hundreds of workers consume tasks in parallel. If one crashes, others continue unaffected.

2. Partitioned PostgreSQL tables

Partition the repositories table by ID range or creation date
E.g we split repositories table into
repositories_1000 (all repos containing 1000 stars)
repositories_2000 (all repos containing 1000 stars)
So we only search relevant tables when filtering data during collecting.

3. Incremental crawling

Use GitHub’s Events API to detect which repositories changed since the last crawl. Only re-fetch those and not all 500M every day.

Q.) How will the schema evolve if you want to gather more metadata in the future, for example, issues, pull requests, commits inside pull requests, comments inside PRs and issues, reviews on a PR, CI checks, etc.? A PR can get 10 comments today and then 20 comments tomorrow. Updating the DB with this new information should be an efficient operation (minimal rows affected) ?

The schema is designed to grow without breaking changes:

Adding new entities/tables (issues, PRs, comments) in the schema :
- Each entity gets its own table - never add all info to repositories
E.g

```
CREATE TABLE pull_requests (
    node_id      TEXT PRIMARY KEY,
    repo_node_id TEXT REFERENCES repositories(node_id),
    number       INTEGER,
    state        TEXT,
    ...
);

CREATE TABLE comments (
```

```
    node_id        TEXT PRIMARY KEY,
    parent_node_id TEXT,
    parent_type    TEXT,
    ...
);
`
```

For Efficient comment updates (PR gets 10 comments today, 20 tomorrow):

- Each comment has its own `node_id`. So New comments = new rows inserted. Existing comments never touched unless their content changed. Only new rows are written - minimal rows affected. (I use this logic for storing my repositories as well).

Rules for schema evolution:

- New columns are always nullable so it never break existing queries
- New entity types get new tables so it never bloats ‘repositories’ table.
- Use `node_id` as FK everywhere because it is stable across repo renames.