PURSUIT

Presented by Gordon Jaja

Data Engineering Take-Home (DETA) Exercise

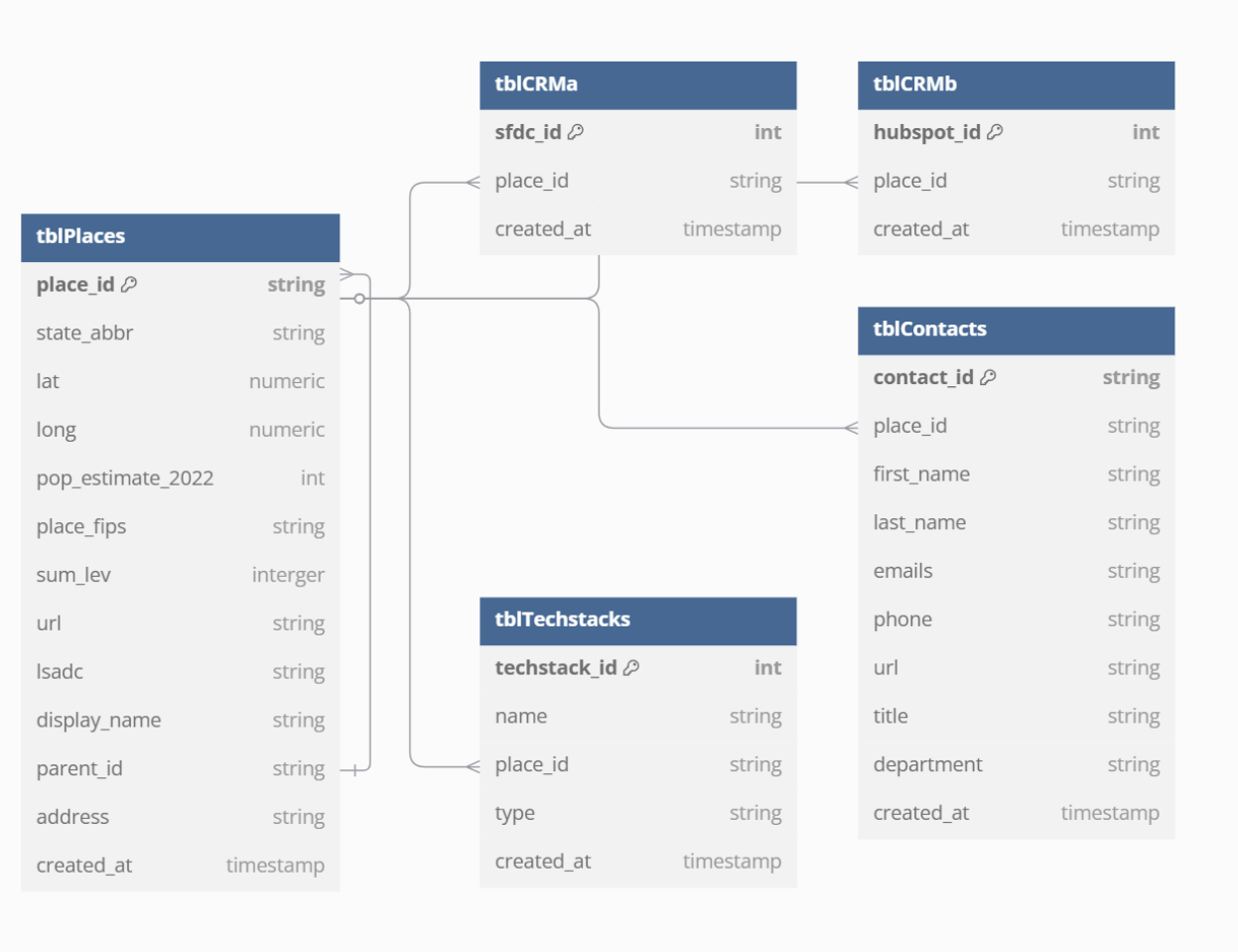
Overview

For review simplicity, this exercise has been scripted in Python and SQLite.

Note that this has limitations in implementation, such as scripting of sequences, indexes, materialized views and functions. Nonetheless, you'll still be able to perform all the tests.

A- Data Structure

Assess the data structure and table fields. Identify relationships, normalize-able data and indexing opportunities. Visualize and review results using modeling tool (dbdiagram.io):



Primary Keys & Notable Fields

Identify unique identifiers for each table, where available, otherwise create one.

- `tblPlaces`
  - `place_id` set as primary key
  - `parent_id` self-references to `place_id`

- `tblContacts`
  - Added `contact_id` as primary key
  - Alternatively, a composed primary key consisting of `first_name`, `last_name`, `email` and `phone` could be used, however it could be unreliable since some values are null.

- `tblTechstacks`
  - Added `techstack_id` as primary key for simplicity, since all 3 existing fields are required to have a primary key.
  - Alternatively, depending on size of table and desired complexity, consider further normalization of name and type into separate dedicated tables.

- Added `created_at` timestamp field to all tables without timestamp field. This is an important addition for numerous reasons, including materialized views, syncing and data warehousing.

Performance Indexing

Select fields that are most likely to be searched by, that are not a primary key.

- `tblContacts`
  - `first_name`
  - `last_name`
  - `department`
  - `title`

- `tblPlaces`
  - `display_name`
  - `state_abbrev`
  - `address`

- `tblTechstacks:`
  - `name`
  - `place_id`
  - `type`

Relationships

Identify relationships based on field names and data similarity.

- `tblContacts.place_id > tblPlaces.place_id` // many-to-one
- `tblTechstacks.place_id > tblPlaces.place_id` // many-to-one
- `tblPlaces.parent_id < tblPlaces.place_id` // one-to-many
- `tblCRMa.place_id > tblPlaces.place_id` // many-to-one
- `tblCRMb.place_id > tblPlaces.place_id` // many-to-one

Search Architecture

- Materialized Views (MV) based on most queried and desired data. In this exercise, I've created a fake MV table (mvEntConCRM.sql) based on the exercise queries.
- Search Functions / Stored Procedures for fast and efficient execution.
- Hybrid search using both Materialized View and table for data after last MV refresh

Performance & Tradeoff considerations

- Materialized views are excellent for searching and showing popular data, the tradeoff being stale data and expensive refreshes.
- Indexing, great to locate, slows-down writes.
- ACID vs Base (NoSQL), consistency vs speed
- Normalization vs. Denormalization, quality & complexity vs. redundancy & simplicity

Performance at Scale

- Indexing / No over-indexing (slows down writes)
- Query Optimization / Analyze / Stored Procs / Best Practices / Batch Updates
- Partitioning based on dates or column
- Caching DB and/or application
- Read Replicas
- Distributed DB / Load Balancers
- Maintenance / Logging / Tracking
- Throttling (application-side)

Other

- Rename fields that are reserved words such as name, type, long. Standardize naming conventions, including pluralization of fields and table names.

B- Pipeline

Script execution (pipedb.py)

- Read all CSV files
- Eliminate/handle/log corrupt data
- Connect to DB
- Create tables (if non-existent)
- Import mapped Data

C- ETL

DB Processing

- Receive imported data
- Validate duplicates/corrupt data (using triggers)
- Format data into desired format
- Log failures
- Sync / flatten data for searching, based on scheduled and/or trigger, depending on requirements.
- Refresh materialized views.

D- Exercise Queries

Q2.a- Give me all entities where a contact has a title contains finance

```
SELECT
      te.*
FROM
      tblContacts tc,
      tblEntities te
WHERE
      tc.place_id=te.place_id and
      tc.title like '%finance%'
```

Q2.b- Show me all contacts where an email contains bob and entity has technology “Accela” and population > 10k

```
SELECT
      tc.*
FROM
      tblContacts tc,
      tblTechstacks tt,
      tblEntities te
WHERE
      tc.place_id=te.place_id and
      tc.place_id=tt.place_id and
      te.pop_estimate_2022>100000 and
      tc.emails like '%Bob%' and
      tt.name like 'Accela%'
```

Q2.c- Give me all entities that are synced w/ Customer A’s CRM.

```
SELECT
      te.*
FROM
      tblEntities te,
      tblCRMa ta
WHERE
      te.place_id=ta.place_id
```