

---

## Contents

### Chapter 1: Introducing the Project

1

## Chapter 1: Introducing the Project

In this chapter we're going to get into the details of the project that will help you learn the SQL Patterns. As you saw in the introduction, we're using a real-world, public dataset from StackOverflow.

StackOverflow is a popular website where users can post technical questions about any technical topic and others can post answers to these questions. They can also vote on the answers or comment on them.

Based on the quality of the answers, users gain reputation and badges which they can use as social proof both on the SO site and on other websites.

Using this dataset we're going to build a table that calculates reputation metrics for every user. This type of table is sometimes called a "feature table" and can be used in other applications in data science and analytics. You simply replace the `user_id` with a customer id or any other entity.

Since the query to build it is complex, it's the perfect tool to illustrate some of the patterns described in this book.

The schema of what it would look something like this:

column_name	type
user_id	INT64
user_name	STRING
total_posts_created	NUMERIC
total_answers_created	NUMERIC
total_answers_edited	NUMERIC
total_questions_created	NUMERIC
total_upvotes	NUMERIC
total_comments_by_user	NUMERIC
total_questions_edited	NUMERIC
streak_in_days	NUMERIC
total_comments_on_post	NUMERIC
posts_per_day	NUMERIC
edits_per_day	NUMERIC

---

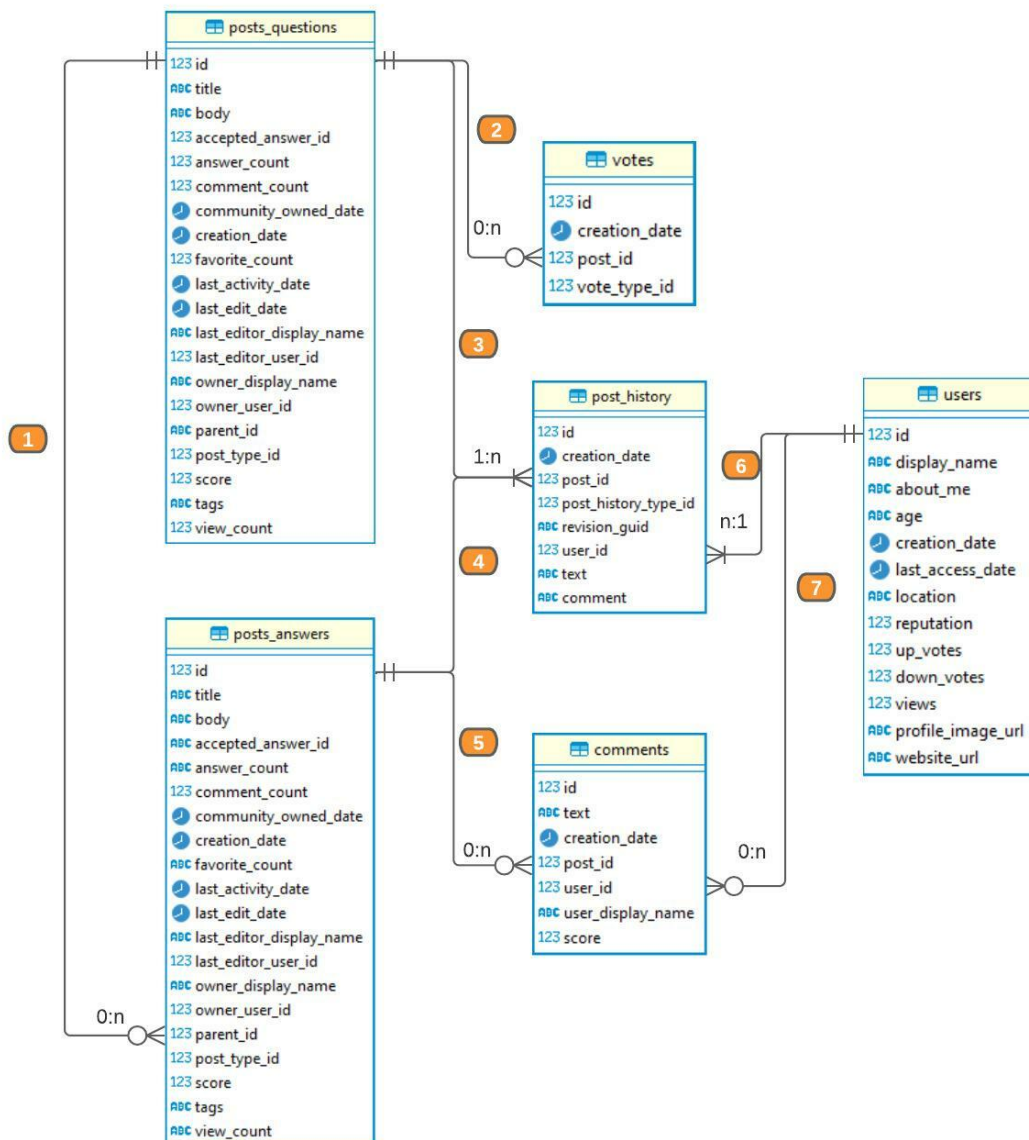
	answers_per_day		NUMERIC	
	questions_per_day		NUMERIC	
	comments_by_user_per_day		NUMERIC	
	answers_per_post		NUMERIC	
	questions_per_post		NUMERIC	
	upvotes_per_post		NUMERIC	
	downvotes_per_post		NUMERIC	
	user_comments_per_post		NUMERIC	
	comments_on_post_per_post		NUMERIC	

As you can see, we need to transform the source data model to a new model that has one row per `user_id`. Before we do that, we need to understand the source data first.

## Understanding the Data Model

Writing accurate and efficient SQL begins with understanding the data model we're starting with. This may already exist in the form of documentation and diagrams but more often than not you'll have to learn it as you go.

The original StackOverflow (SO) data model is different from the one loaded in BigQuery. When the engineers loaded it, they modified the model somewhat. For example the SO model contains a single `Posts` table for all the different post types whereas BigQuery split each one into a separate table.



**Figure 1.1 - StackOverflow ER diagram**

There are 8 tables that represent the various post types. You can get this result by using the INFORMATION\_SCHEMA views in BigQuery like this:

```
SELECT table_name
FROM bigquery-public-data.stackoverflow.INFORMATION_SCHEMA.TABLES
WHERE table_name like 'posts_'
```

Here's the result of the query

---

table_name	
posts_answers	
posts_orphaned_tag_wiki	
posts_tag_wiki	
posts_questions	
posts_tag_wiki_excerpt	
posts_wiki_placeholder	
posts_privilege_wiki	
posts_moderator_nomination	

We'll be focusing on just two of them for our project so I've left the other ones out: 1. `posts_questions` contains all the question posts 2. `posts_answers` contains all the answer posts

They both have the same schema:

```
SELECT column_name, data_type
FROM bigquery-public-data.stackoverflow.INFORMATION_SCHEMA.COLUMNS
WHERE table_name = 'posts_answers'
```

Here's the result of the query

column_name	data_type
id	INT64
title	STRING
body	STRING
accepted_answer_id	STRING
answer_count	STRING
comment_count	INT64
community_owned_date	TIMESTAMP
creation_date	TIMESTAMP
favorite_count	STRING
last_activity_date	TIMESTAMP
last_edit_date	TIMESTAMP
last_editor_display_name	STRING
last_editor_user_id	INT64
owner_display_name	STRING
owner_user_id	INT64

---

---

parent_id	INT64	
post_type_id	INT64	
score	INT64	
tags	STRING	
view_count	STRING	

Both tables have an `id` column that identifies a single post, `creation_date` that identifies the timestamp when the post was created and a few other attributes like `score` for the upvotes and downvotes.

Note the `parent_id` column which signifies a hierarchical structure. The `parent_id` is a one-to-many relationship that links up an answer to the corresponding question. A single question can have multiple answers but an answer belongs to one and only one question. This is relation 1 in the **Figure 1.1** above.

Both post types (question and answer) have a one-to-many relationship to the `post_history`. These are relations 3 and 4 in the diagram above.

```
SELECT column_name, data_type
FROM bigquery-public-data.stackoverflow.INFORMATION_SCHEMA.COLUMNS
WHERE table_name = 'post_history'
```

Here's the result of the query

column_name	data_type
-----	-----
id	INT64
creation_date	TIMESTAMP
post_id	INT64
post_history_type_id	INT64
revision_guid	STRING
user_id	INT64
text	STRING
comment	STRING

A single post can have many types of activities identified by the `post_history_type_id` column. This id indicates the different types of activities a user can perform on the site. We're only concerned with the first 6. You can see the rest of them here if you're curious.

1. Initial Title - initial title (*questions only*)

- 
2. Initial Body - initial post (*raw body text*)
  3. Initial Tags - initial list of tags (*questions only*)
  4. Edit Title - modified title (*questions only*)
  5. Edit Body - modified post body (*raw markdown*)
  6. Edit Tags - modified list of tags (*questions only*)

The first 3 indicate when a post is first submitted and the next 3 when a post is edited.

The `post_history` table also connects to the `users` table via the `user_id` in a one-to-many relationship shown in the diagram as number 6. A single user can perform multiple activities on a post.

In database lingo this is known as a bridge table because it connects two tables (user and posts) that have a many-to-many relationship which cannot be modeled otherwise.

The `users` table has one row per user and contains user attributes such as name, reputation, etc. We'll use some of these attributes in our final table.

```
SELECT column_name, data_type
FROM bigquery-public-data.stackoverflow.INFORMATION_SCHEMA.COLUMNS
WHERE table_name = 'users'
```

Here's the result of the query

column_name	data_type
-----	-----
id	INT64
display_name	STRING
about_me	STRING
age	STRING
creation_date	TIMESTAMP
last_access_date	TIMESTAMP
location	STRING
reputation	INT64
up_votes	INT64
down_votes	INT64
views	INT64
profile_image_url	STRING
website_url	STRING

---

Next we take a look at the comments table. It has a zero-to-many relationship with posts and with users shown in the diagram as number 5 and number 7, since both a user or a post could have 0 or many comments.

```
SELECT column_name, data_type
FROM bigquery-public-data.stackoverflow.INFORMATION_SCHEMA.COLUMNS
WHERE table_name = 'comments'
```

Here's the result of the query

column_name	data_type
id	INT64
text	STRING
creation_date	TIMESTAMP
post_id	INT64
user_id	INT64
user_display_name	STRING
score	INT64

Finally the votes table represents the upvotes and downvotes on a post. We'll need this to compute the total vote count on a user's post which will indicate how good the question or the answer is. This table has a granularity of one row per vote per post per date.

```
SELECT column_name, data_type
FROM bigquery-public-data.stackoverflow.INFORMATION_SCHEMA.COLUMNS
WHERE table_name = 'votes'
```

Here's the result of the query

column_name	data_type
id	INT64
creation_date	TIMESTAMP
post_id	INT64
vote_type_id	INT64

The votes table is connected to a post in a 0-to-many relationship shown in the diagram as number 2. In order for us to get upvotes and downvotes on a user's post, we'll need to join it with the users table.

---

Alright, now that we've familiarized ourselves with the source data model, next let's look at some core concepts.