Kickstarter SQL Style Guide

kickstarter_sql_style_guide.md

layout	title	description	tags
default	SQL Style Guide	A guide to writing clean, clear, and consistent SQL.	data process

Purpose

Maintaining reproducibility and transparency is a core value of Kickstarter's Data team, and a SQL style guide can help us achieve that goal. Additionally, adhering to the basic rules in this style guide will improve our ability to share, maintain, and extend our research when working with SQL.

This document is written as a manual for anyone working on the Data team, but also as a guide for anyone at the company who would like to write clean and clear code that is meant to be shared.

The individual tips in this guide are based on a composite of knowledge we've gleaned from experience and our roles at previous jobs.

NOTE: This style guide is written for use with AWS Redshift/Postgres 8.0.2, but much of it can be applied to any SQL database.

Principles

- We take a disciplined and practical approach to writing code.
- We regularly check-in code to Github
- We believe consistency in style is very important.
- We demonstrate intent explicitly in code, via clear structure and comments where needed.

Rules

General stuff

- No tabs. 2 spaces per indent.
- No trailing whitespace.
- Always capitalize SQL keywords (e.g., `SELECT` or `AS`)
- Variable names should be underscore separated:

  GOOD: `SELECT COUNT(*) AS backers_count`

  BAD: `SELECT COUNT(*) AS backersCount`
- Comments should go near the top of your query, or at least near the closest `SELECT`
- Try to only comment on things that aren't obvious about the query (e.g., why a particular ID is hardcoded, etc.)
- Don't use single letter variable names be as descriptive as possible given the context:

  GOOD: `SELECT ksr.backings AS backings_with_creators`

  BAD: `SELECT ksr.backings AS b`
- Use Common Table Expressions (CTEs) early and often, and name them well.

- `HAVING` isn't supported in Redshift, so use CTEs instead. If you don't know what this means, ask a friendly Data Team member.

SELECT

Align all columns to the first column on their own line:

```
SELECT
  projects.name,
  users.email,
  projects.country,
  COUNT(backings.id) AS backings_count
FROM ...
```

`SELECT` goes on its own line:

```
SELECT
  name,
  ...
```

Always rename aggregates and function-wrapped columns:

```
SELECT
  name,
  SUM(amount) AS sum_amount
FROM ...
```

Always rename all columns when selecting with table aliases:

```
SELECT
  projects.name AS project_name,
  COUNT(backings.id) AS backings_count
FROM ksr.backings AS backings
INNER JOIN ksr.projects AS projects ON ...
```

Always use `AS` to rename columns:

GOOD:

```
SELECT
  projects.name AS project_name,
  COUNT(backings.id) AS backings_count
...
```

BAD:

```
SELECT
  projects.name project_name,
  COUNT(backings.id) backings_count
...
```

Long Window functions should be split across multiple lines: one for the `PARTITION`, `ORDER` and frame clauses, aligned to the `PARTITION` keyword. Partition keys should be one-per-line, aligned to the first, with aligned commas. Order (`ASC`, `DESC`) should always be explicit. All window functions should be aliased.

```
SUM(1) OVER (PARTITION BY category_id,
                          year
             ORDER BY pledged DESC
             ROWS UNBOUNDED PRECEDING) AS category_year
```

FROM

Only one table should be in the `FROM`. Never use `FROM`-joins:

GOOD:

```
SELECT
  projects.name AS project_name,
  COUNT(backings.id) AS backings_count
FROM ksr.projects AS projects
INNER JOIN ksr.backings AS backings ON backings.project_id = projects.id
...
```

BAD:

```
SELECT
  projects.name AS project_name,
  COUNT(backings.id) AS backings_count
FROM ksr.projects AS projects, ksr.backings AS backings
WHERE
  backings.project_id = projects.id
...
```

JOIN

Explicitly use `INNER JOIN` not just `JOIN`, making multiple lines of `INNER JOIN`s easier to scan:

GOOD:

```
SELECT
  projects.name AS project_name,
  COUNT(backings.id) AS backings_count
FROM ksr.projects AS projects
INNER JOIN ksr.backings AS backings ON ...
INNER JOIN ...
LEFT JOIN ksr.backer_rewards AS backer_rewards ON ...
LEFT JOIN ...
```

BAD:

```
SELECT
  projects.name AS project_name,
  COUNT(backings.id) AS backings_count
FROM ksr.projects AS projects
JOIN ksr.backings AS backings ON ...
LEFT JOIN ksr.backer_rewards AS backer_rewards ON ...
LEFT JOIN ...
```

Additional filters in the `INNER JOIN` go on new indented lines:

```
SELECT
  projects.name AS project_name,
  COUNT(backings.id) AS backings_count
FROM ksr.projects AS projects
INNER JOIN ksr.backings AS backings ON projects.id = backings.project_id
  AND backings.project_country != 'US'
...
```

The `ON` keyword and condition goes on the `INNER JOIN` line:

```
SELECT
  projects.name AS project_name,
  COUNT(backings.id) AS backings_count
FROM ksr.projects AS projects
INNER JOIN ksr.backings AS backings ON projects.id = backings.project_id
...
```

Begin with `INNER JOIN`s and then list `LEFT JOIN`s, order them semantically, and do not intermingle `LEFT JOIN`s with `INNER JOIN`s unless necessary:

GOOD:

```
INNER JOIN ksr.backings AS backings ON ...
INNER JOIN ksr.users AS users ON ...
INNER JOIN ksr.locations AS locations ON ...
LEFT JOIN ksr.backer_rewards AS backer_rewards ON ...
LEFT JOIN ...
```

BAD:

```
LEFT JOIN ksr.backer_rewards AS backer_rewards ON backings
INNER JOIN ksr.users AS users ON ...
LEFT JOIN ...
INNER JOIN ksr.locations AS locations ON ...
```

WHERE

Multiple `WHERE` clauses should go on different lines and begin with the SQL operator:

```
SELECT
  name,
  goal
FROM ksr.projects AS projects
WHERE
  country = 'US'
  AND deadline >= '2015-01-01'
...
```

CASE

`CASE` statements aren't always easy to format but try to align `WHEN`, `THEN`, and `ELSE` together inside `CASE` and `END`:

```
CASE WHEN category = 'Art'
     THEN backer_id
     ELSE NULL
END
```

Common Table Expressions (CTEs)

From AWS:

> `WITH` clause subqueries are an efficient way of defining tables that can be used throughout the execution of a single query. In all cases, the same results can be achieved by using subqueries in the main body of the `SELECT` statement, but `WITH` clause subqueries may be simpler to write and read.

The body of a CTE must be one indent further than the `WITH` keyword. Open them at the end of a line and close them on a new line:

```
WITH backings_per_category AS (
  SELECT
    category_id,
    deadline,
    ...
)
```

Multiple CTEs should be formatted accordingly:

```
WITH backings_per_category AS (
  SELECT
    ...
```

```
), backers AS (
  SELECT
    ...
), backers_and_creators AS (
  ...
)
SELECT * FROM backers;
```

If possible, `JOIN` CTEs inside subsequent CTEs, not in the main clause:

GOOD:

```
WITH backings_per_category AS (
  SELECT
    ...
), backers AS (
  SELECT
    backer_id,
    COUNT(backings_per_category.id) AS projects_backed_per_category
  INNER JOIN ksr.users AS users ON users.id = backings_per_category.backer_id
), backers_and_creators AS (
  ...
)
SELECT * FROM backers_and_creators;
```

BAD:

```
WITH backings_per_category AS (
  SELECT
    ...
), backers AS (
  SELECT
    backer_id,
    COUNT(backings_per_category.id) AS projects_backed_per_category
), backers_and_creators AS (
  ...
)
SELECT * FROM backers_and_creators
INNER JOIN backers ON backers_and_creators ON backers.backer_id = backers_and_creators.backer_id
```

Always use CTEs over inlined subqueries.

Tips

- Sublime is your friend. Configure it to use soft tabs (e.g. 2 spaces), and trim trailing whitespace
- Helpful Sublime packages include Githubinator, SendText, and Package Control.
- Check code into github early and often.
- Always provide a Githubinator permalink in Trello cards where any code is used.

evansolomon commented on Jan 30, 2016

I always felt like window function style was tricky to get right, but personally I don't love the mid-line alignment. Feels kinda like objective-c to me, which is never good in a style guid :) I tended to use style them more like subqueries, i.e.

```
  SUM(1) OVER (
    PARTITION BY
      category_id,
      year
    ORDER BY pledged DESC
    ROWS UNBOUNDED PRECEDING
  ) AS category_year
```

evansolomon commented on Jan 30, 2016

I personally like indenting `THEN` one level past its parent `WHEN`, with the whole thing indented from its `CASE` / `END`. I know some people feel like this is too much nesting though.

```
CASE
  WHEN category = 'Art'
    THEN backer_id
  WHEN category = 'Whatever'
    THEN backer_something_else
  ELSE NULL
END
```

fredbenenson commented on Jan 30, 2016 Owner

@evansolomon Great tips ... `CASE` and Window functions are always mushy in terms of formatting for me. Sometimes it just feels like I'm trying to keep it under 80 chars :)