dbt coding conventions

Model configuration

- Model-specific attributes (like sort/dist keys) should be specified in the model.
- If a particular configuration applies to all models in a directory, it should be specified in the `dbt_project.yml` file.
- In-model configurations should be specified like this:

```
{{
  config(
    materialized = 'table',
    sort = 'id',
    dist = 'id'
  )
}}
```

dbt conventions

- Only `stg_` models (or `base_` models if your project requires them) should select from `source`s.
- All other models should only select from other models.

Testing

- Every subdirectory should contain a `schema.yml` file, in which each model in the subdirectory is tested.
- At a minimum, unique and not_null tests should be applied to the primary key of each model.

Naming and field conventions

- Schema, table and column names should be in `snake_case`.
- Use names based on the *business* terminology, rather than the source terminology.
- Table names should be plurals, e.g. `accounts`.
- Each model should have a primary key.
- The primary key of a model should be named `<object>_id`, e.g. `account_id` – this makes it easier to know what `id` is being referenced in downstream joined models.
- Timestamp columns should be named `<event>_at`, e.g. `created_at`, and should be in UTC. If a different timezone is being used, this should be indicated with a suffix, e.g `created_at_pt`.
- Booleans should be prefixed with `is_` or `has_`.
- Price/revenue fields should be in decimal currency (e.g. `19.99` for $19.99; many app databases store prices as integers in cents). If non-decimal currency is used, indicate this with suffix, e.g. `price_in_cents`.
- Avoid reserved words as column names
- Consistency is key! Use the same field names across models where possible, e.g. a key to the `customers` table should be named `customer_id` rather than `user_id`.

CTEs

- All `{{ ref('...') }}` statements should be placed in CTEs at the top of the file
- Where performance permits, CTEs should perform a single, logical unit of work.
- CTE names should be as verbose as needed to convey what they do
- CTEs with confusing or noteable logic should be commented
- CTEs that are duplicated across models should be pulled out into their own models
- CTEs should be formatted like this:

```
with

events as (

    ...

),

-- CTE comments go here
filtered_events as (

    ...

)

select * from filtered_events
```

SQL style guide

- Indents should be four spaces (except for predicates, which should line up with the `where` keyword)
- Lines of SQL should be no longer than 80 characters
- Field names and function names should all be lowercase
- The `as` keyword should be used when aliasing a field or table
- Fields should be stated before aggregates / window functions
- Ordering and grouping by a number (eg. group by 1, 2) is preferred. Note that if you are grouping by more than a few columns, it may be worth revisiting your model design.
- Specify join keys - do not use `using`. Certain warehouses have inconsistencies in `using` results (specifically Snowflake).
- Prefer `union all` to `union` *
- Avoid table aliases in join conditions (especially initialisms) – it's harder to understand what the table called "c" is compared to "customers".
- If joining two or more tables, *always* prefix your column names with the table alias. If only selecting from one table, prefixes are not needed.
- Be explicit about your join (i.e. write `inner join` instead of `join`). `left joins` are normally the most useful, `right joins` often indicate that you should change which table you select `from` and which one you `join` to.
- *DO NOT OPTIMIZE FOR A SMALLER NUMBER OF LINES OF CODE. NEWLINES ARE CHEAP, BRAIN TIME IS EXPENSIVE*

Example SQL

```
with

my_data as (

    select * from {{ ref('my_data') }}

),

some_cte as (

    select * from {{ ref('some_cte') }}

),

final as (

    select [distinct]
        my_data.field_1,
        my_data.field_2,
        my_data.field_3,

        -- use line breaks to visually separate calculations into blocks
        case
            when my_data.cancellation_date is null and my_data.expiration_date is not null then expiration_date
            when my_data.cancellation_date is null then my_data.start_date + 7
            else my_data.cancellation_date
        end as cancellation_date,

        -- use a line break before aggregations
        sum(some_cte.field_4),
        max(some_cte.field_5)

    from my_data

    left join some_cte
        on my_data.id = some_cte.id

    where my_data.field_1 = 'abc'
      and (
          my_data.field_2 = 'def' or
          my_data.field_2 = 'ghi'
      )

    group by 1, 2, 3, 4
    having count(*) > 1

)

select * from final
```

- Your join should list the "left" table first (i.e. the table you are selecting `from`):

```
select
    trips.*,
    drivers.rating as driver_rating,
    riders.rating as rider_rating

from trips

left join users as drivers
    on trips.driver_id = drivers.user_id

left join users as riders
    on trips.rider_id = riders.user_id
```

YAML style guide

- Indents should be two spaces
- List items should be indented
- Use a new line to separate list items that are dictionaries where appropriate
- Lines of YAML should be no longer than 80 characters.

Example YAML

```
version: 2

models:
  - name: events
    columns:
      - name: event_id
        description: This is a unique identifier for the event
        tests:
          - unique
          - not_null

      - name: event_time
        description: "When the event occurred in UTC (eg. 2018-01-01 12:00:00)"
        tests:
          - not_null

      - name: user_id
        description: The ID of the user who recorded the event
        tests:
          - not_null
          - relationships:
              to: ref('users')
              field: id
```

Jinja style guide

- When using Jinja delimiters, use spaces on the inside of your delimiter, like `{{ this }}` instead of `{{this}}`
- Use newlines to visually indicate logical blocks of Jinja