Introduction and Environment Setup

Snowflake user creation

Copy these SQL statements into a Snowflake Worksheet, select all and execute them (i.e. pressing the play button).

If you see a *Grant partially executed: privileges [REFERENCE_USAGE] not granted.* message when you execute `GRANT ALL ON DATABASE AIRBNB to ROLE transform`, that's just an info message and you can ignore it.

```
-- Use an admin role
USE ROLE ACCOUNTADMIN;

-- Create the `transform` role
CREATE ROLE IF NOT EXISTS transform;
GRANT ROLE TRANSFORM TO ROLE ACCOUNTADMIN;

-- Create the default warehouse if necessary
CREATE WAREHOUSE IF NOT EXISTS COMPUTE_WH;
GRANT OPERATE ON WAREHOUSE COMPUTE_WH TO ROLE TRANSFORM;

-- Create the `dbt` user and assign to role
CREATE USER IF NOT EXISTS dbt
  PASSWORD='dbtPassword123'
  LOGIN_NAME='dbt'
  MUST_CHANGE_PASSWORD=FALSE
  DEFAULT_WAREHOUSE='COMPUTE_WH'
```

```
  DEFAULT_ROLE='transform'
  DEFAULT_NAMESPACE='AIRBNB.RAW'
  COMMENT='DBT user used for data transformation';
GRANT ROLE transform to USER dbt;

-- Create our database and schemas
CREATE DATABASE IF NOT EXISTS AIRBNB;
CREATE SCHEMA IF NOT EXISTS AIRBNB.RAW;

-- Set up permissions to role `transform`
GRANT ALL ON WAREHOUSE COMPUTE_WH TO ROLE transform;
GRANT ALL ON DATABASE AIRBNB to ROLE transform;
GRANT ALL ON ALL SCHEMAS IN DATABASE AIRBNB to ROLE
transform;
GRANT ALL ON FUTURE SCHEMAS IN DATABASE AIRBNB to ROLE
transform;
GRANT ALL ON ALL TABLES IN SCHEMA AIRBNB.RAW to ROLE
transform;
GRANT ALL ON FUTURE TABLES IN SCHEMA AIRBNB.RAW to ROLE
transform;
```

Snowflake data import

Copy these SQL statements into a Snowflake Worksheet, select all and execute them (i.e. pressing the play button).

```
-- Set up the defaults
USE WAREHOUSE COMPUTE_WH;
USE DATABASE airbnb;
USE SCHEMA RAW;

-- Create our three tables and import the data from S3
```

```
CREATE OR REPLACE TABLE raw_listings
                    (id integer,
                     listing_url string,
                     name string,
                     room_type string,
                     minimum_nights integer,
                     host_id integer,
                     price string,
                     created_at datetime,
                     updated_at datetime);

COPY INTO raw_listings (id,
                        listing_url,
                        name,
                        room_type,
                        minimum_nights,
                        host_id,
                        price,
                        created_at,
                        updated_at)
                   from 's3://dbtlearn/listings.csv'
                    FILE_FORMAT = (type = 'CSV'
skip_header = 1
                    FIELD_OPTIONALLY_ENCLOSED_BY = '"');


CREATE OR REPLACE TABLE raw_reviews
                    (listing_id integer,
                     date datetime,
                     reviewer_name string,
                     comments string,
                     sentiment string);
```

```
COPY INTO raw_reviews (listing_id, date, reviewer_name,
comments, sentiment)
                    from 's3://dbtlearn/reviews.csv'
                     FILE_FORMAT = (type = 'CSV'
skip_header = 1
                     FIELD_OPTIONALLY_ENCLOSED_BY = '"');


CREATE OR REPLACE TABLE raw_hosts
                     (id integer,
                      name string,
                      is_superhost string,
                      created_at datetime,
                      updated_at datetime);

COPY INTO raw_hosts (id, name, is_superhost, created_at,
updated_at)
                    from 's3://dbtlearn/hosts.csv'
                     FILE_FORMAT = (type = 'CSV'
skip_header = 1
                     FIELD_OPTIONALLY_ENCLOSED_BY = '"');
```

-- END OF SNOWFLAKE DATA IMPORT

Python and Virtualenv setup, and dbt installation - Windows

Python

This is the Python installer you want to use:

https://www.python.org/ftp/python/3.10.7/python-3.10.7-amd64.exe

Please make sure that you work with Python 3.11 as newer versions of python might not be compatible with some of the dbt packages.

Virtualenv setup

Here are the commands we executed in this lesson:

```
cd Desktop
mkdir course
cd course

virtualenv venv
venv\Scripts\activate
```

Virtualenv setup and dbt installation - Mac

iTerm2

We suggest you to use *iTerm2* instead of the built-in Terminal application.

https://iterm2.com/

Homebrew

Homebrew is a widely popular application manager for the Mac. This is what we use in the class for installing a virtualenv.

https://brew.sh/

dbt installation

Here are the commands we execute in this lesson:

```
create course
cd course
virtualenv venv
. venv/bin/activate
pip install dbt-snowflake==1.5.0
which dbt
```

dbt setup

Initialize the dbt profiles folder on Mac/Linux:

```
mkdir ~/.dbt
```

Initialize the dbt profiles folder on Windows:

```
mkdir %userprofile%\.dbt
```

Create a dbt project (all platforms):

```
dbt init dbtlearn
```

Models

Code used in the lesson

SRC Listings

`models/src/src_listings.sql`:

```
WITH raw_listings AS (
    SELECT
        *
    FROM
        AIRBNB.RAW.RAW_LISTINGS
)
SELECT
    id AS listing_id,
    name AS listing_name,
    listing_url,
    room_type,
    minimum_nights,
    host_id,
    price AS price_str,
    created_at,
    updated_at
FROM
    raw_listings
```

SRC Reviews

`models/src/src_reviews.sql`:

```
WITH raw_reviews AS (
    SELECT
```

```
        *
    FROM
        AIRBNB.RAW.RAW_REVIEWS
)
SELECT
    listing_id,
    date AS review_date,
    reviewer_name,
    comments AS review_text,
    sentiment AS review_sentiment
FROM
    raw_reviews
```

Exercise

Create a model which builds on top of our `raw_hosts` table.

1. Call the model `models/src/src_hosts.sql`
2. Use a CTE (common table expression) to define an alias called `raw_hosts`. This CTE select every column from the raw hosts table `AIRBNB.RAW.RAW_HOSTS`
3. In your final `SELECT`, select every column and record from `raw_hosts` and rename the following columns:
    - `id` to `host_id`
    - `name` to `host_name`

Solution

```
WITH raw_hosts AS (
    SELECT
        *
    FROM
       AIRBNB.RAW.RAW_HOSTS
)
SELECT
    id AS host_id,
    NAME AS host_name,
    is_superhost,
    created_at,
    updated_at
FROM
    raw_hosts
```

Models

Code used in the lesson

DIM Listings

`models/dim/dim_listings_cleansed.sql`:

```
WITH src_listings AS (
  SELECT
    *
  FROM
    {{ ref('src_listings') }}
)
```

```
SELECT
  listing_id,
  listing_name,
  room_type,
  CASE
    WHEN minimum_nights = 0 THEN 1
    ELSE minimum_nights
  END AS minimum_nights,
  host_id,
  REPLACE(
    price_str,
    '$'
  ) :: NUMBER(
    10,
    2
  ) AS price,
  created_at,
  updated_at
FROM
  src_listings
```

DIM hosts

`models/dim/dim_hosts_cleansed.sql`:

```
{{
  config(
    materialized = 'view'
    )
}}

WITH src_hosts AS (
    SELECT
```

```
        *
    FROM
        {{ ref('src_hosts') }}
)
SELECT
    host_id,
    NVL(
        host_name,
        'Anonymous'
    ) AS host_name,
    is_superhost,
    created_at,
    updated_at
FROM
    src_hosts
```

Exercise

Create a new model in the `models/dim/` folder called `dim_hosts_cleansed.sql`.

- Use a CTE to reference the `src_hosts` model
- SELECT every column and every record, and add a cleansing step to host_name:
    - If host_name is not null, keep the original value
    - If host_name is null, replace it with the value 'Anonymous'
    - Use the NVL(column_name, default_null_value) function Execute `dbt run` and verify that your model has been created

Solution

```
WITH src_hosts AS (
    SELECT
        *
    FROM
        {{ ref('src_hosts') }}
)
SELECT
    host_id,
    NVL(
        host_name,
        'Anonymous'
    ) AS host_name,
    is_superhost,
    created_at,
    updated_at
FROM
    src_hosts
```

Incremental Models

The `fct/fct_reviews.sql` model:

```
{{
  config(
    materialized = 'incremental',
    on_schema_change='fail'
    )
}}
WITH src_reviews AS (
  SELECT * FROM {{ ref('src_reviews') }}
```

```
)
SELECT * FROM src_reviews
WHERE review_text is not null

{% if is_incremental() %}
  AND review_date > (select max(review_date) from {{
this }})
{% endif %}
```

Get every review for listing *3176*:

```
SELECT * FROM "AIRBNB"."DEV"."FCT_REVIEWS" WHERE
listing_id=3176;
```

Add a new record to the table:

```
INSERT INTO "AIRBNB"."RAW"."RAW_REVIEWS"
VALUES (3176, CURRENT_TIMESTAMP(), 'Zoltan', 'excellent
stay!', 'positive');
```

Making a full-refresh:

```
dbt run --full-refresh
```

DIM listings with hosts

The contents of `dim/dim_listings_w_hosts.sql`:

```
WITH
l AS (
    SELECT
        *
```

```
    FROM
        {{ ref('dim_listings_cleansed') }}
),
h AS (
    SELECT *
    FROM {{ ref('dim_hosts_cleansed') }}
)

SELECT
    l.listing_id,
    l.listing_name,
    l.room_type,
    l.minimum_nights,
    l.price,
    l.host_id,
    h.host_name,
    h.is_superhost as host_is_superhost,
    l.created_at,
    GREATEST(l.updated_at, h.updated_at) as updated_at
FROM l
LEFT JOIN h ON (h.host_id = l.host_id)
```

Dropping the views after ephemeral materialization

```
DROP VIEW AIRBNB.DEV.SRC_HOSTS;
DROP VIEW AIRBNB.DEV.SRC_LISTINGS;
DROP VIEW AIRBNB.DEV.SRC_REVIEWS;
```

Sources and Seeds

Full Moon Dates CSV

Download the CSV from the lesson's *Resources* section, or download it from the following S3 location: https://dbtlearn.s3.us-east-2.amazonaws.com/seed_full_moon_dates.csv

Then place it to the `seeds` folder

If you download from S3 on a Mac/Linux, can you import the csv straight to your seed folder by executing this command:

```
curl https://dbtlearn.s3.us-east-2.amazonaws.com/seed_full_moon_dates.csv -o seeds/seed_full_moon_dates.csv
```

Contents of models/sources.yml

```yaml
version: 2

sources:
  - name: airbnb
    schema: raw
    tables:
      - name: listings
        identifier: raw_listings

      - name: hosts
        identifier: raw_hosts

      - name: reviews
        identifier: raw_reviews
```

```
        loaded_at_field: date
        freshness:
          warn_after: {count: 1, period: hour}
          error_after: {count: 24, period: hour}
```

Contents of models/mart/full_moon_reviews.sql

```
{{ config(
  materialized = 'table',
) }}

WITH fct_reviews AS (
    SELECT * FROM {{ ref('fct_reviews') }}
),
full_moon_dates AS (
    SELECT * FROM {{ ref('seed_full_moon_dates') }}
)

SELECT
  r.*,
  CASE
    WHEN fm.full_moon_date IS NULL THEN 'not full moon'
    ELSE 'full moon'
  END AS is_full_moon
FROM
  fct_reviews
  r
  LEFT JOIN full_moon_dates
  fm
  ON (TO_DATE(r.review_date) = DATEADD(DAY, 1,
fm.full_moon_date))
```

Snapshots

Snapshots for listing

The contents of `snapshots/scd_raw_listings.sql`:

```
{% snapshot scd_raw_listings %}

{{
   config(
       target_schema='dev',
       unique_key='id',
       strategy='timestamp',
       updated_at='updated_at',
       invalidate_hard_deletes=True
   )
}}

select * FROM {{ source('airbnb', 'listings') }}

{% endsnapshot %}
```

Updating the table

```
UPDATE AIRBNB.RAW.RAW_LISTINGS SET MINIMUM_NIGHTS=30,
    updated_at=CURRENT_TIMESTAMP() WHERE ID=3176;

SELECT * FROM AIRBNB.DEV.SCD_RAW_LISTINGS WHERE ID=3176;
```

Snapshots for hosts

The contents of `snapshots/scd_raw_hosts.sql`:

```
{% snapshot scd_raw_hosts %}

{{
   config(
       target_schema='dev',
       unique_key='id',
       strategy='timestamp',
       updated_at='updated_at',
       invalidate_hard_deletes=True
   )
}}

select * FROM {{ source('airbnb', 'hosts') }}

{% endsnapshot %}
```

Tests

Generic Tests

The contents of `models/schema.yml`:

```
version: 2

models:
  - name: dim_listings_cleansed
    columns:

      - name: listing_id
        tests:
```

```
          - unique
          - not_null

      - name: host_id
        tests:
          - not_null
          - relationships:
              to: ref('dim_hosts_cleansed')
              field: host_id

      - name: room_type
        tests:
          - accepted_values:
              values: ['Entire home/apt',
                       'Private room',
                       'Shared room',
                       'Hotel room']
```

Generic test for minimum nights check

The contents of `tests/dim_listings_minumum_nights.sql`:

```
SELECT
    *
FROM
    {{ ref('dim_listings_cleansed') }}
WHERE minimum_nights < 1
LIMIT 10
```

Restricting test execution to a model

```
dbt test --select dim_listings_cleansed
```

Exercise

Create a singular test in `tests/consistent_created_at.sql` that checks that there is no review date that is submitted before its listing was created: Make sure that every `review_date` in `fct_reviews` is more recent than the associated `created_at` in `dim_listings_cleansed`.

Solution

```
SELECT * FROM {{ ref('dim_listings_cleansed') }} l
INNER JOIN {{ ref('fct_reviews') }} r
USING (listing_id)
WHERE l.created_at >= r.review_date
```

Marcos, Custom Tests and Packages

Macros

The contents of `macros/no_nulls_in_columns.sql`:

```
{% macro no_nulls_in_columns(model) %}
    SELECT * FROM {{ model }} WHERE
    {% for col in adapter.get_columns_in_relation(model)
-%}
        {{ col.column }} IS NULL OR
    {% endfor %}
    FALSE
{% endmacro %}
```

The contents of `tests/no_nulls_in_dim_listings.sql`

```
{{ no_nulls_in_columns(ref('dim_listings_cleansed')) }}
```

Custom Generic Tests

The contents of `macros/positive_value.sql`

```
{% test positive_value(model, column_name) %}
SELECT
    *
FROM
    {{ model }}
WHERE
    {{ column_name}} < 1
{% endtest %}
```

Packages

The contents of `packages.yml`:

```
packages:
  - package: dbt-labs/dbt_utils
    version: 0.8.0
```

The contents of `models/fct_reviews.sql`:

```
{{
  config(
    materialized = 'incremental',
    on_schema_change='fail'
    )
}}
WITH src_reviews AS (
  SELECT * FROM {{ ref('src_reviews') }}
)
SELECT
  {{ dbt_utils.surrogate_key(['listing_id',
'review_date', 'reviewer_name', 'review_text']) }}
    AS review_id,
  *
  FROM src_reviews
WHERE review_text is not null
{% if is_incremental() %}
  AND review_date > (select max(review_date) from {{
this }})
{% endif %}
```

Documentation

The `models/schema.yml` after adding the documentation:

```
version: 2
```

```
models:
  - name: dim_listings_cleansed
    description: Cleansed table which contains Airbnb
listings.
    columns:

      - name: listing_id
        description: Primary key for the listing
        tests:
          - unique
          - not_null

      - name: host_id
        description: The hosts's id. References the host
table.
        tests:
          - not_null
          - relationships:
              to: ref('dim_hosts_cleansed')
              field: host_id

      - name: room_type
        description: Type of the apartment / room
        tests:
          - accepted_values:
              values: ['Entire home/apt', 'Private
room', 'Shared room', 'Hotel room']

      - name: minimum_nights
        description: '{{
doc("dim_listing_cleansed__minimum_nights") }}'
        tests:
```

```
          - positive_value

  - name: dim_hosts_cleansed
    columns:
      - name: host_id
        tests:
          - not_null
          - unique

      - name: host_name
        tests:
          - not_null

      - name: is_superhost
        tests:
          - accepted_values:
              values: ['t', 'f']

  - name: fct_reviews
    columns:
      - name: listing_id
        tests:
          - relationships:
              to: ref('dim_listings_cleansed')
              field: listing_id

      - name: reviewer_name
        tests:
          - not_null

      - name: review_sentiment
        tests:
          - accepted_values:
```

```
                values: ['positive', 'neutral',
'negative']
```

The contents of `models/docs.md`:

```
{% docs dim_listing_cleansed__minimum_nights %}
Minimum number of nights required to rent this property.

Keep in mind that old listings might have
`minimum_nights` set
to 0 in the source tables. Our cleansing algorithm
updates this to `1`.

{% enddocs %}
```

The contents of `models/overview.md`:

```
{% docs __overview__ %}
# Airbnb pipeline

Hey, welcome to our Airbnb pipeline documentation!

Here is the schema of our input data:
![input schema](https://dbtlearn.s3.us-east-
2.amazonaws.com/input_schema.png)

{% enddocs %}
```

Analyses, Hooks and Exposures

Create the REPORTER role and PRESET user in Snowflake

```
USE ROLE ACCOUNTADMIN;
CREATE ROLE IF NOT EXISTS REPORTER;
CREATE USER IF NOT EXISTS PRESET
 PASSWORD='presetPassword123'
 LOGIN_NAME='preset'
 MUST_CHANGE_PASSWORD=FALSE
 DEFAULT_WAREHOUSE='COMPUTE_WH'
 DEFAULT_ROLE='REPORTER'
 DEFAULT_NAMESPACE='AIRBNB.DEV'
 COMMENT='Preset user for creating reports';

GRANT ROLE REPORTER TO USER PRESET;
GRANT ROLE REPORTER TO ROLE ACCOUNTADMIN;
GRANT ALL ON WAREHOUSE COMPUTE_WH TO ROLE REPORTER;
GRANT USAGE ON DATABASE AIRBNB TO ROLE REPORTER;
GRANT USAGE ON SCHEMA AIRBNB.DEV TO ROLE REPORTER;

-- We don't want to grant select rights here; we'll do
this through hooks:
-- GRANT SELECT ON ALL TABLES IN SCHEMA AIRBNB.DEV TO
ROLE REPORTER;
-- GRANT SELECT ON ALL VIEWS IN SCHEMA AIRBNB.DEV TO
ROLE REPORTER;
-- GRANT SELECT ON FUTURE TABLES IN SCHEMA AIRBNB.DEV TO
ROLE REPORTER;
-- GRANT SELECT ON FUTURE VIEWS IN SCHEMA AIRBNB.DEV TO
ROLE REPORTER;
```

Analyses

The contents of `analyses/full_moon_no_sleep.sql`:

```
WITH mart_fullmoon_reviews AS (
    SELECT * FROM {{ ref('mart_fullmoon_reviews') }}
)
SELECT
    is_full_moon,
    review_sentiment,
    COUNT(*) as reviews
FROM
    mart_fullmoon_reviews
GROUP BY
    is_full_moon,
    review_sentiment
ORDER BY
    is_full_moon,
    review_sentiment
```

Exposures

The contents of `models/dashboard.yml`:

```
version: 2

exposures:
  - name: Executive Dashboard
    type: dashboard
    maturity: low
    url: https://7e942fbd.us2a.app.preset.io:443/r/2
```

```
    description: Executive Dashboard about Airbnb
listings and hosts


    depends_on:
      - ref('dim_listings_w_hosts')
      - ref('mart_fullmoon_reviews')

    owner:
      name: Zoltan C. Toth
      email: hello@learndbt.com
```

Post-hook

Add this to your `dbt_project.yml`:

```
+post-hook:
      - "GRANT SELECT ON {{ this }} TO ROLE REPORTER"
```

Debugging Tests and Testing with dbt-expectations

- The original Great Expectations project on GitHub: https://github.com/great-expectations/great_expectations
- dbt-expectations: https://github.com/calogica/dbt-expectations

For the final code in *packages.yml*, *models/schema.yml* and *models/sources.yml*, please refer to the course's Github repo: https://github.com/nordquant/complete-dbt-bootcamp-zero-to-hero

Testing a single model

```
dbt test --select dim_listings_w_hosts
```

Testing individual sources:

```
dbt test --select source:airbnb.listings
```

Debugging dbt

```
dbt --debug test --select dim_listings_w_hosts
```

Keep in mind that in the lecture we didn't use the *--debug* flag after all as taking a look at the compiled sql file is the better way of debugging tests.