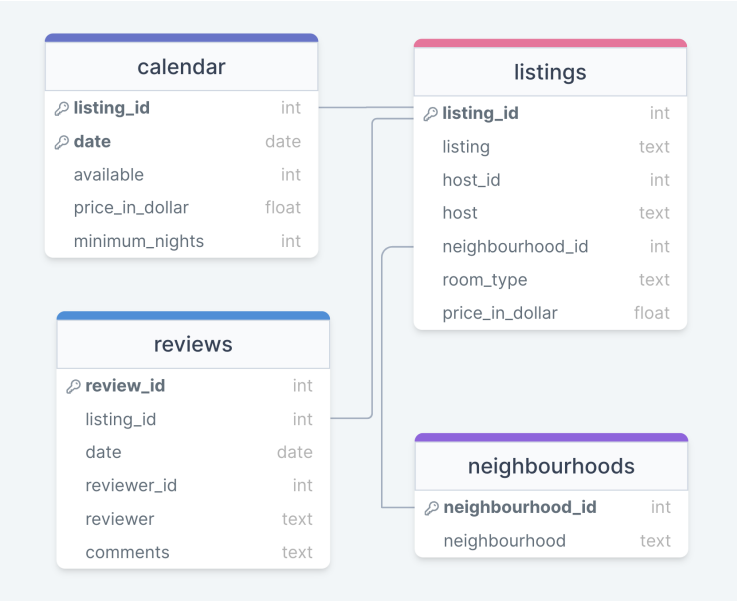

```
%%capture
!pip install git+https://github.com/sb2nov/sql-cc.git


import pandas as pd
from IPython.display import display, HTML
import sqlcc
from sqlcc import check

# Show all the rows (instead of only a few)
pd.set_option("display.max_rows", None)

# Set precision to max 2 decimals
pd.set_option('display.precision', 2)  # Use 'display.precision' instead of 'precision'

# Set CSS Style for Table
# Make it work with night & light mode
# - Alternating rows
# - th elements
# - td elements
css_style = '''
<style>
  html {
    --td-font-color: black;
    --font-color: black;
    --background-color: #e0e0e0;
  }
  html[theme=dark] {
    --td-font-color: white;
    --font-color: black;
    --background-color: #6688ff;
  }
  th {
    background: #fbd44c;
    color: var(--font-color);
    font-size: 16px;
    text-align: center;
    font-weight: bold;
  }
  tr:nth-child(even) {
    background-color: var(--background-color);
    color: var(--font-color);
  }
  td {
    font-size: 14px;
    color: var(--td-font-color);
  }
</style>
'''


def run(sql_query):
  df = sqlcc.run(sql_query)

  # Puts the scrollbar next to the DataFrame
  display(HTML(css_style +
               "<div style='max-height: 500px; overflow: auto; width: fit-content; border-style: solid;" +
               " border-width: 1px; border-color: #0139fe; font-family: GT Planar,Inter,Arial,sans-serif;'>" +
               df.style.render() +
               "</div>"))
```

1. `%%capture`: This is a Jupyter Notebook magic command that captures the output of the subsequent cell. It is typically used to suppress the output from being displayed.
2. `!pip install git+https://github.com/sb2nov/sql-cc.git`: This command installs the `sql-cc` library from a Git repository. The library provides SQL query execution capabilities.
3. `import pandas as pd`: This imports the pandas library, which is used for data manipulation and analysis.
4. `from IPython.display import display, HTML`: This imports the `display` and `HTML` classes from the `IPython.display` module. These classes are used to customize and render output in Jupyter Notebooks.
5. `import sqlcc`: This imports the `sqlcc` module, which is a part of the `sql-cc` library installed earlier. It provides functions for executing SQL queries.
6. `from sqlcc import check`: This imports the `check` function from the `sqlcc` module. This function is not used in the provided code snippet.
7. `pd.set_option("display.max_rows", None)`: This sets the pandas option `display.max_rows` to `None`, which means that pandas will display all rows of a DataFrame instead of truncating them.
8. `pd.set_option('display.precision', 2)`: This sets the pandas option `display.precision` to `2`, which ensures that floating-point numbers in the DataFrame will be displayed with a precision of two decimal places.
9. `css_style`: This is a CSS style defined as a multi-line string. It contains styling rules for customizing the appearance of the rendered table. It sets colors, font sizes, and background colors for different elements of the table.
10. `def run(sql_query)`: This defines a function named `run` that takes an SQL query as an argument. Inside the function, it executes the SQL query using the `sqlcc.run` function and stores the result in a DataFrame named `df`. Then, it uses the `display` and `HTML` classes to render the DataFrame as an HTML table with the custom CSS style applied.

AirBnB Sydney database contains tables: **listings**, **neighbourhoods**, **reviews**, and **calendar**.




```
### Question:
query = "SELECT * FROM neighbourhoods;"

run(query)
```

```
<ipython-input-5-86fbc71b6552>:55: FutureWarning: this method is deprecated in favour of `Styler.to_html()`
  df.style.render() +
```

	neighbourhood_id	neighbourhood
0	0	Ashfield
1	1	Auburn

As General manager for Airbnb Sydney, you need to make data-driven decisions about our region. We've already had many encounters with hosts calling in, and we anticipate many more scenarios that will require we to use SQL in the future. Rosa is a stellar support person who works on our team at Airbnb Sydney. She is often the first person who people speak with when they call the Airbnb Sydney office. Rosa recently got a call from a host who had experienced some issues, which she was able to successfully resolve. It's now a week later, and she would like to follow-up with the host to see if they have encountered any additional problems. Unfortunately, Rosa didn't write down the *host_id* and only remembers that the host's name started with "Jos". Upset, Rosa asks if you can write a query that matches this description?

9	9	Canterbury	

+ Code + Text

```
### Question:
# 1/ Fetch all columns from the listings table
# 2/ Make sure you filtered the data to names that start with Jos
query = """
SELECT *
  FROM listings
    WHERE host LIKE 'Jos%';
"""

run(query)
```

```
<ipython-input-5-86fbc71b6552>:55: FutureWarning: this method is deprecated in favour of `Styler.to_html()`
  df.style.render() +
```

	listing_id	listing	host_id	host	neighbourhood_id	room_type	price_in_dollar
0	22296011	Large private room on Camperdown park & Newtown	10873080	Joshua	21	Private room	40.000000

Before Rosa follows up with the host, she wants to make sure that she is prepared for the call. Information about the host:

- the neighbourhood name
- the availability of the room on the calendar
- if there are any reviews written for the room

```
### Question:
# 1/ Fetch all columns from the neighbourhoods table
# 2/ Make sure you filtered the data on the
# neighbourhood id of the host name starting with Jos...
query = """
SELECT *
  FROM neighbourhoods
    WHERE neighbourhood_id IN (
        SELECT neighbourhood_id
          FROM listings
            WHERE host LIKE 'Jos%');
"""
# also we can use WHERE neighbourhood_id = (SELECT...
run(query)
```

```
<ipython-input-5-86fbc71b6552>:55: FutureWarning: this method is deprecated in favour of `Styler.to_html()`
  df.style.render() +
```

	neighbourhood_id	neighbourhood
0	21	Marrickville

```
### Question:
# 1/ Fetch all columns from the calendar table
# 2/ Make sure you filtered the data on the
# listing id of the host name starting with Jos...

query = """
SELECT *
  FROM calendar
    WHERE listing_id = (
        SELECT listing_id
          FROM listings
            WHERE host LIKE 'Jos%');
"""
# WHERE listing_id IN (SELECT
run(query)
```

```
<ipython-input-5-86fbc71b6552>:55: FutureWarning: this method is deprecated in favour of `Styler.to_html()`
  df.style.render() +
```

```
### Question:
# 1/ Fetch all columns from the reviews table
# 2/ Make sure you filtered the data on the
# listing id of the host name starting with Jos...

query = """
SELECT *
  FROM reviews
    WHERE listing_id IN (
      SELECT listing_id
        FROM listings
          WHERE host LIKE 'Jos%');
"""
# also we can use WHERE listing_id = (SELECT...
run(query)
```

```
<ipython-input-5-86fbc71b6552>:55: FutureWarning: this method is deprecated in favour of `Styler.to_html()`
  df.style.render() +
```

listing_id	review_id	date	reviewer_id	reviewer	comments

The host's name is "Joshua." He lives in Marrickville, has no availablility all week, and no reviews have been made for his room.

Airbnb Melbourne has been in operation longer than our office in Sydney. They called to gauge the prices that we have for our listings. They wanted to know the maximum, minimum, and average price for our listings to compare with what they're seeing in their region.

```
### Question:
# 1/ Fetch price_in_dollar from the listings table
# 2/ Get the "Average" price across all listings

query = """
SELECT AVG(price_in_dollar)
  FROM listings;
"""

run(query)
```

```
<ipython-input-5-86fbc71b6552>:55: FutureWarning: this method is deprecated in favour of `Styler.to_html()`
  df.style.render() +
```

	AVG(price_in_dollar)
0	703.571429

```
### Question:
# 1/ Fetch price_in_dollar from the listings table
# 2/ Get the "Maximum" price across all listings

query = """
SELECT MAX(price_in_dollar)
  FROM listings;
"""

run(query)
```

```
<ipython-input-5-86fbc71b6552>:55: FutureWarning: this method is deprecated in favour of `Styler.to_html()`
  df.style.render() +
```

	MAX(price_in_dollar)
0	28613.000000

```
### Question:
# 1/ Fetch price_in_dollar from the listings table
# 2/ Get the "Minimum" price across all listings

query = """
SELECT MIN(price_in_dollar)
  FROM listings;
"""

run(query)
```

```
<ipython-input-5-86fbc71b6552>:55: FutureWarning: this method is deprecated in favour of `Styler.to_html()`
  df.style.render() +
```

	MIN(price_in_dollar)
0	30.000000

Monthly growth goals.

We need to expand as quickly as possible all across Sydney. Some mentors tell me it's better to focus on places where we are already quite present. Let's see if we can get an overview of which neighbourhoods have the most listings. Make use of GROUP BY where we focus on neighbourhood_id and its count.

```
### Question:
# 1/ Fetch neighbourhood_id from the listings table and
# 2/ the number of listings in each neighbourhood -- Please count the neighbourhood_id column.

query = """
SELECT neighbourhood_id, COUNT(neighbourhood_id)
  FROM listings
    GROUP BY neighbourhood_id;
"""

run(query)
```

```
<ipython-input-5-86fbc71b6552>:55: FutureWarning: this method is deprecated in favour of `Styler.to_html()`
  df.style.render() +
```

	neighbourhood_id	COUNT(neighbourhood_id)
0	0	1
1	1	1
2	4	1
3	8	2
4	11	2
5	17	1
6	18	2
7	20	5
8	21	2
9	23	3
10	24	1
11	26	3
12	27	4
13	28	1
14	29	3
15	31	2
16	32	10
17	33	1
18	35	13
19	37	5

```
### Question:
# 1/ Fetch neighbourhood_id from the listings table
# 2/ For the second column get the number of listings in each neighbourhood
# 3/ Make sure to order by count descendingly.

query = """
SELECT neighbourhood_id, COUNT(neighbourhood_id)
  FROM listings
    GROUP BY neighbourhood_id
      ORDER BY COUNT(neighbourhood_id) DESC;
"""
# also we can use ORDER BY 2 DESC
run(query)
```

```
<ipython-input-5-86fbc71b6552>:55: FutureWarning: this method is deprecated in favour of `Styler.to_html()`
  df.style.render() +
```

	neighbourhood_id	COUNT(neighbourhood_id)
0	35	13
1	32	10
2	37	5
3	20	5
4	27	4
5	29	3
6	26	3
7	23	3
8	31	2
9	21	2
10	18	2
11	11	2
12	8	2
13	33	1
14	28	1
15	24	1
16	17	1
17	4	1
18	1	1
19	0	1

Looks like we've 20 neighbourhoods in the listings table but quiet a few with really few listings.

Are there neighbourhoods with no Airbnb listings? How do we get all the neighbourhood_id's from neighbourhoods?

```
### Question:
# 1/ Fetch neighbourhood_id from the neighbourhoods table

query = """
SELECT neighbourhood_id
  FROM neighbourhoods;
"""

run(query)
```

```
<ipython-input-5-86fbc71b6552>:55: FutureWarning: this method is deprecated in favour of `Styler.to_html()`
  df.style.render() +
```

	neighbourhood_id
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
11	11
12	12
13	13
14	14
15	15
16	16
17	17
18	18
19	19

We see about 38 neighbourhoods in the neighbourhoods table compared to the 20 in listings. So our analysis shows that some neighbourhoods have few listings or none at all. These might offer interesting new opportunities to explore.

Creating a new (simplified) column

Now that we've recruited hosts and have some listings, we want to improve the experience of our customers. We want to make it faster for them to go through listings that might be of interest to them. Some of our customers have a big pockets, and others don't. So we want to make three kinds of classes:

- Affordable (0 - 70 dollar per night)
- Mid-range (70 - 300 dollar per night)
- Expensive (300+ dollar per night)

Let's make a SQL case statement to create a new column called "price_range", which includes the *listing*, *listing_id*, *host*, and *host_id*.

```
### Question:
# 1/ Fetch listing, listing_id, host, host_id
# from the listings table
# 2/ Create a case on price_in_dollar which takes into account:
# - Affordable < 70
# - Expensive > 300
# - Else Mid-range
# This we call price_range

query = """
SELECT listing, listing_id, host, host_id,
    CASE
      WHEN price_in_dollar < 70 THEN 'Affordable'
      WHEN price_in_dollar > 300 THEN 'Expensive'
      ELSE 'Mid-range'
    END AS price_range
FROM listings;
"""

run(query)
```

```
<ipython-input-5-86fbc71b6552>:55: FutureWarning: this method is deprecated in favour of `Styler.to_html()`
  df.style.render() +
```

	listing	listing_id	host	host_id	price_range
0	Peaceful 1 Bedroom Apartment in Bondi Beach	574105250645758912	Andrew	109067745	Mid-range
1	Bondi Vibes - Funky Designer Studio	7874902	Alex White	41506490	Mid-range
2	Nice studio close to the beach!	4575789	María	22980172	Mid-range
3	Just bring your beach towel	23077495	Gladys	1305312	Mid-range
4	Stylish lite 2b+2bth mod secure Beach apt with pkg	657377039990074112	Rick	285488167	Expensive
5	Spacious and clean 2-bedroom apartment in Bondi	53798702	Tiina	244604436	Mid-range
6	Bondi Beach Apartment 50m to beach	4344478	Stephen	22553304	Mid-range
7	Calm & Coastal: Bronte Beach Studio with Parking	48699778	Annie	185783910	Mid-range
8	Minutes to Bronte beach Ocean views	12072720	Angelika	11745874	Mid-range
9	Terrace in heart of Bondi Junction	20255786	Astrid And Nick	18901875	Expensive
10	Bright stylish home in a great location	22973428	Sian	2537559	Mid-range
11	Double room available in open and airy Bronte home	22199388	Audrey	14228436	Affordable
12	Bondi Beachy - 2 bedroom in the heart of Bondi!	705145817561041408	Bondi Beach Holiday Homes	113874	Mid-range
13	House in Central Sydney	8925052	Barry	43312353	Mid-range
14	Central and Sunny Room	11970913	Hannah	37772146	Affordable
15	Central&Spacious 3Bed Apt at Zetland | MHS	672903848532202624	Evan	185126628	Expensive
16	Bayside Studio close to City with amazing view	22234562	Jackie	43502287	Mid-range
17	Centrally located, spacious, cute, sunny studio!	16411678	Alzbeta	1043937	Mid-range

We're now interested in tracking all neighbourhoods in which we are "over-represented". Let's first count all the occurences of each neighbourhood in our `listings`-table.

```
### Question:
# 1/ Fetch neighbourhood_id from the listings table
# 2/ For the second column get the number of listings in each neighbourhood

query = """
SELECT neighbourhood_id, COUNT(*)
  FROM listings
    GROUP BY neighbourhood_id;
"""

run(query)
```

```
<ipython-input-5-86fbc71b6552>:55: FutureWarning: this method is deprecated in favour of `Styler.to_html()`
  df.style.render() +
```

	neighbourhood_id	COUNT(*)
0	0	1
1	1	1
2	4	1
3	8	2
4	11	2
5	17	1
6	18	2
7	20	5
8	21	2
9	23	3
10	24	1
11	26	3
12	27	4
13	28	1
14	29	3
15	31	2
16	32	10
17	33	1
18	35	13
19	37	5

Let's limit the results to only neighbourhoods that have a count higher than 4.

```
### Question:
# 1/ Fetch neighbourhood_id from the listings table
# 2/ For the second column get the number of listings in each neighbourhood
# 3/ Filter this summarized count > 4 (greater than).

query = """
SELECT neighbourhood_id, COUNT(*)
  FROM listings
    GROUP BY neighbourhood_id
      HAVING COUNT(*) > 4;
"""
```

```
run(query)
```

```
<ipython-input-5-86fbc71b6552>:55: FutureWarning: this method is deprecated in favour of `Styler.to_html()`
  df.style.render() +
```

	neighbourhood_id	COUNT(*)
0	20	5
1	32	10
2	35	13
3	37	5