Proprietary + Confidential

09

Code Generation

In this module, you learn to ...

01 Add code generation to your developer workflow

02 Use GenAI to document, debug, test, optimize, and better understand your code

03 Run BigQuery queries based on natural language

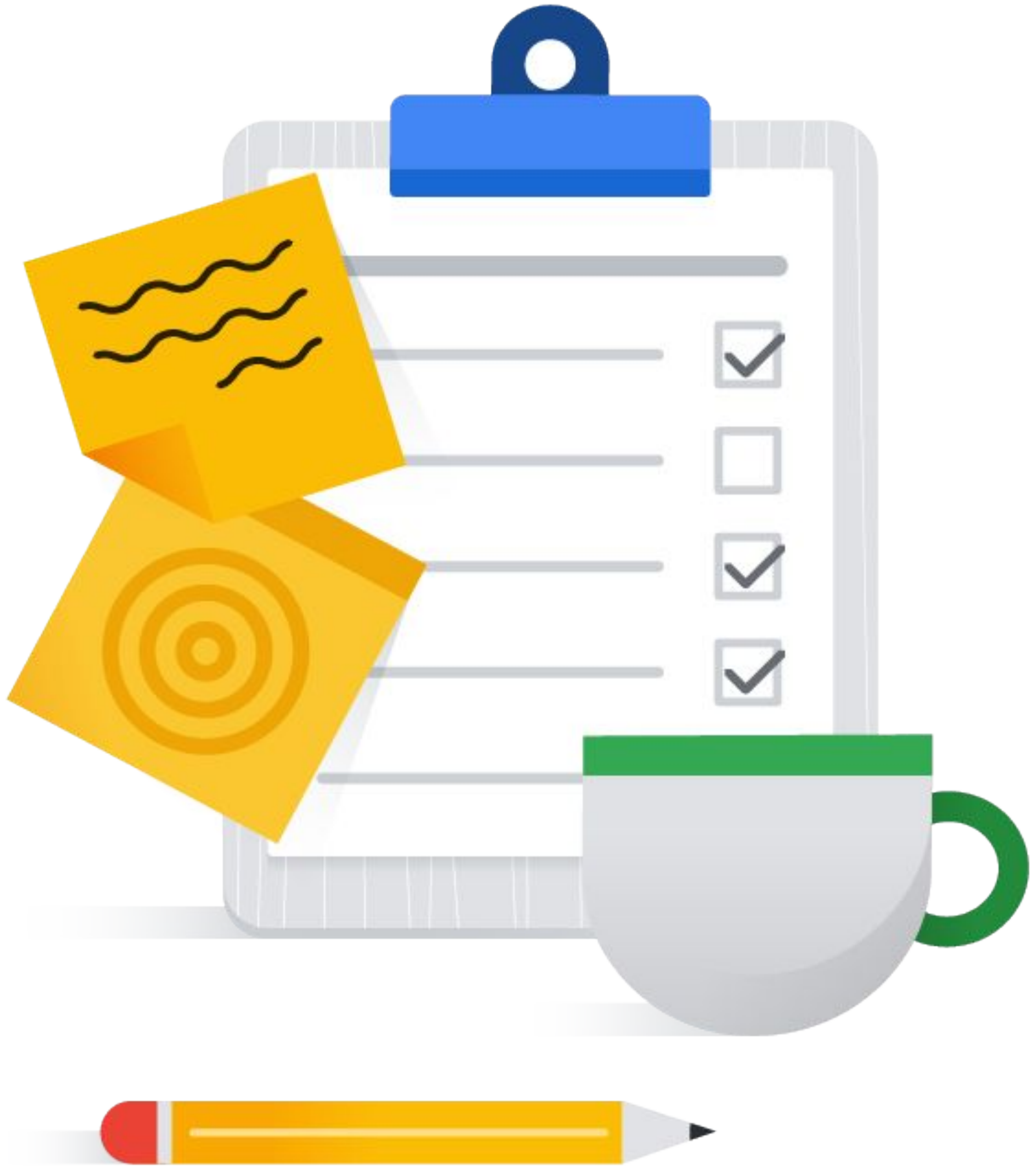
04 Use Duet AI within the Google Cloud Console to simplify your cloud projects

Topics

01 Using Codey for Code Generation

02 Case Study: Generating SQL for BigQuery

03 Duet AI



Codey API for Text-to-Code

Python
Node.js
Java
Go
PHP
Ruby
.NET (C#)
SQL

Prompt

# Write a function that checks if a year is a leap year.

Response

```
def is_leap_year(year):
   “””Returns True if year is a leap year,
False otherwise.”””
   if year % 4 == 0:
      Return True
```

Codey helps developers work faster,
and close skills gaps.

Proprietary + Confidential

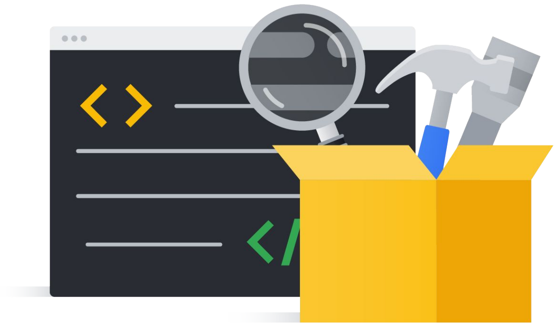
Codey is a tuned version of PaLM optimized for code

- Includes both code generation and chat version
    - code-bison for generation
    - codechat-bison for chat
    - 32k versions are available when larger inputs are required
- Used just like their text counterparts
    - Keep the temperature low

```
import vertexai
from vertexai.language_models import CodeChatModel

vertexai.init(project="vertext-ai-dar",
location="us-central1")
chat_model =
CodeChatModel.from_pretrained("codechat-bison")
parameters = {
    "candidate_count": 1,
    "max_output_tokens": 1024,
    "temperature": 0.2
}
chat = chat_model.start_chat()
```

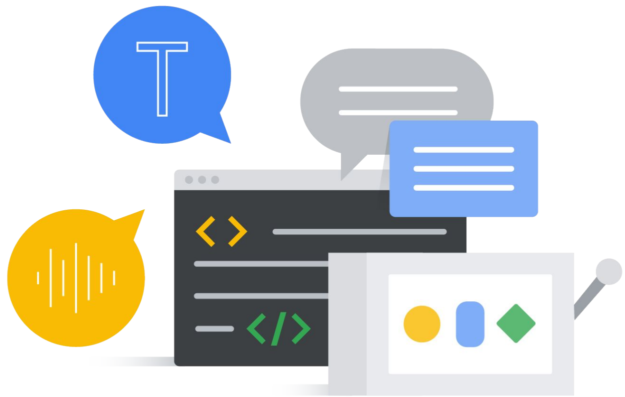
Vertex AI Codey APIs

Code Generation API



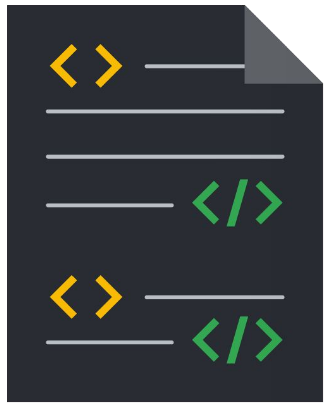
`code-bison`

Code Chat API



`codechat-bison`

Code Completion API



`code-gecko`

Proprietary + Confidential

Use cases for Codey

- Code Generation
- Writing test code
- Code explanations
- Documentation and comments
- Debugging
- Code Optimization
- Code Conversion
- Text to SQL

Proprietary + Confidential

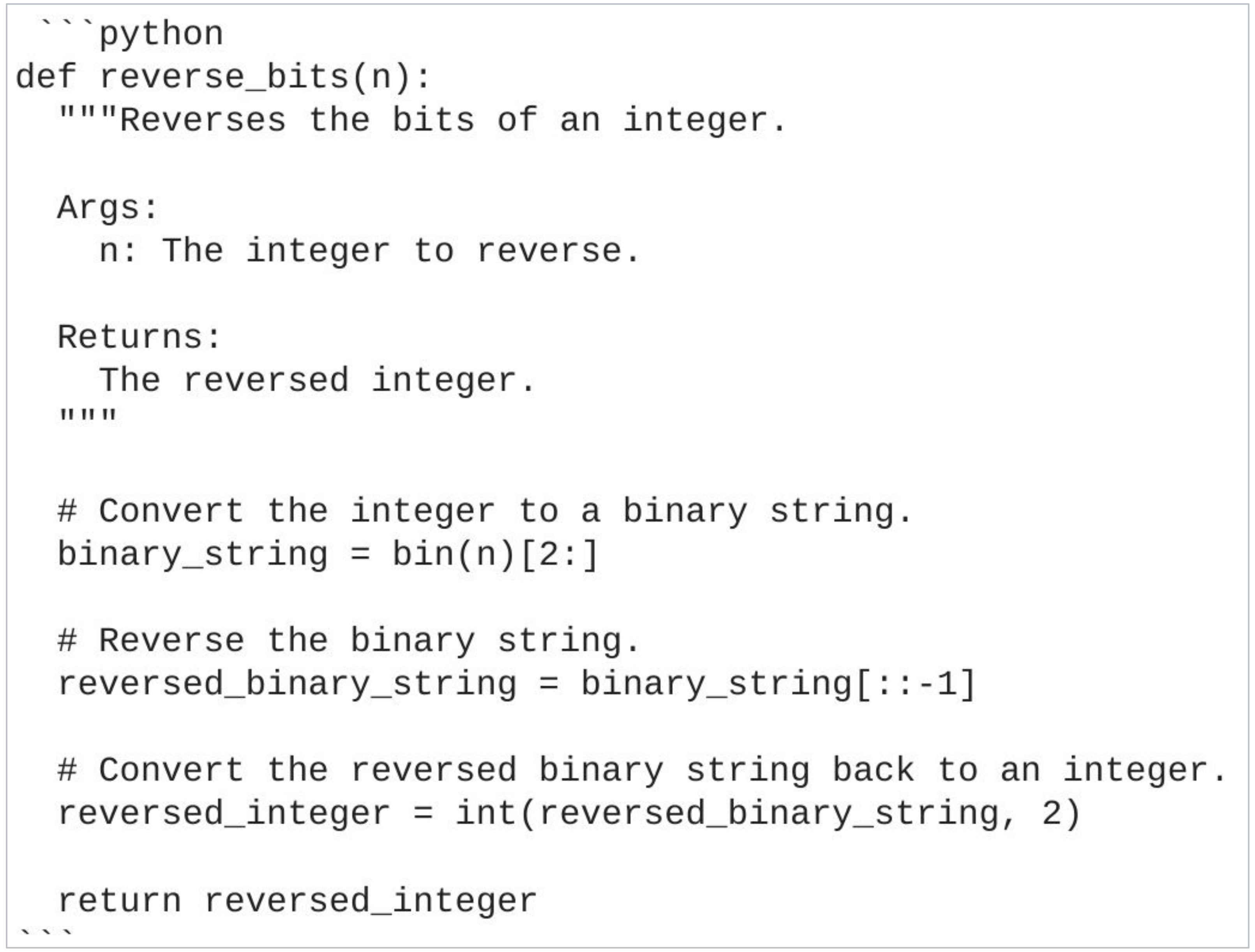
Improving Codey’s accuracy

- Add context to prompts that tell Codey how you want it to code
- Give the model explicit instructions
- Add examples if needed

```
prompt = '''
Context: You are a Coding Chatbot. You answer coding questions.
         You generate code in Python unless you are told otherwise.
         You follow Python code best practices as defined by the
         PEP 8 – Style Guide for Python Code

Question: {0}
Code:
'''
```

Code generation example

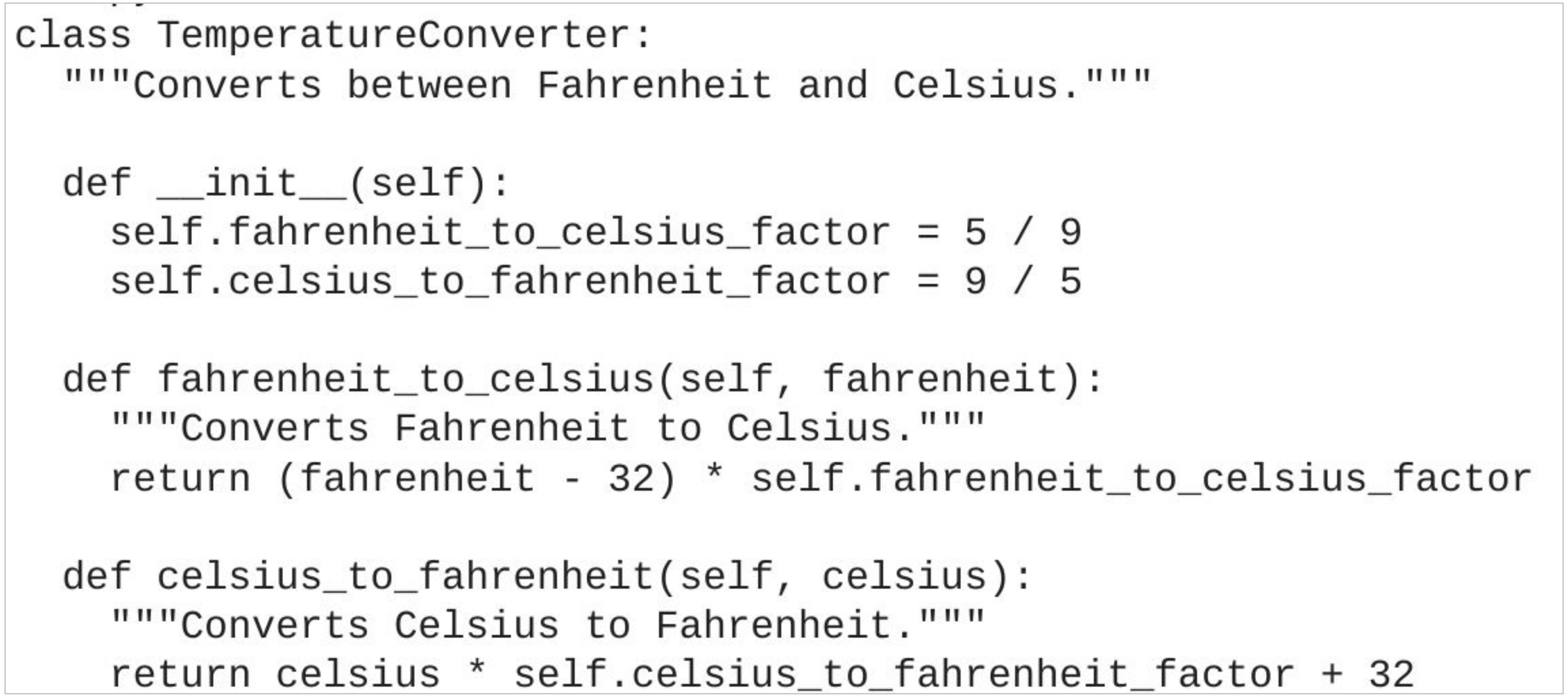
```
question = "Write a function that reverses the bits of an integer."

response = chat.send_message(prompt.format(question), **parameters)
print(response.text)
```



~~~
 ```python
def reverse_bits(n):
  """Reverses the bits of an integer.

  Args:
    n: The integer to reverse.

  Returns:
    The reversed integer.
  """

  # Convert the integer to a binary string.
  binary_string = bin(n)[2:]

  # Reverse the binary string.
  reversed_binary_string = binary_string[::-1]

  # Convert the reversed binary string back to an integer.
  reversed_integer = int(reversed_binary_string, 2)

  return reversed_integer
```
~~~

Code generation example 2

```
question = '''Write a Python class that can be used to convert Fahrenheit to Celsius
              and visa versa
'''
```



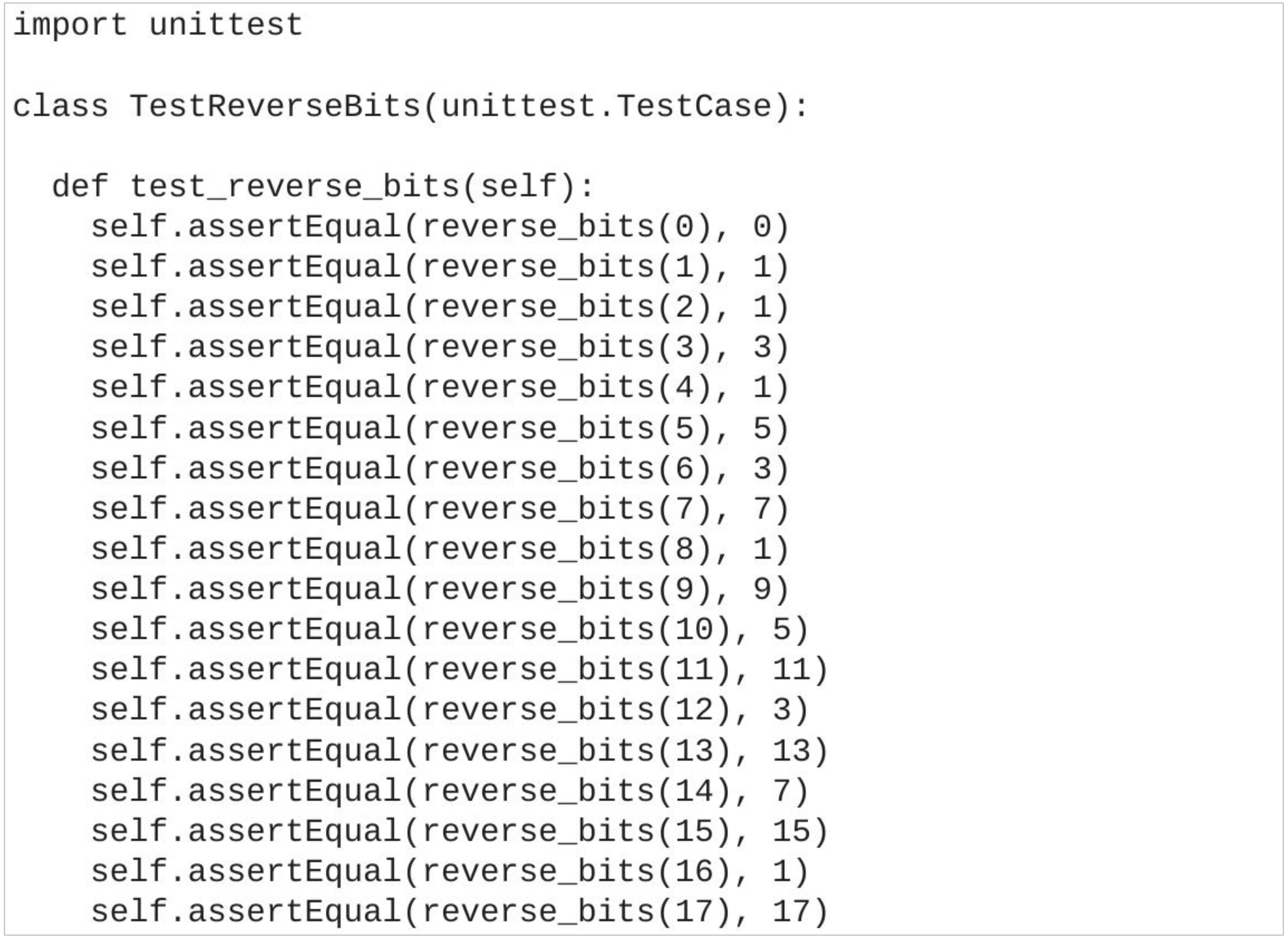
```
class TemperatureConverter:
  """Converts between Fahrenheit and Celsius."""

  def __init__(self):
    self.fahrenheit_to_celsius_factor = 5 / 9
    self.celsius_to_fahrenheit_factor = 9 / 5

  def fahrenheit_to_celsius(self, fahrenheit):
    """Converts Fahrenheit to Celsius."""
    return (fahrenheit - 32) * self.fahrenheit_to_celsius_factor

  def celsius_to_fahrenheit(self, celsius):
    """Converts Celsius to Fahrenheit."""
    return celsius * self.celsius_to_fahrenheit_factor + 32
```

Writing test code example

```
question = '''Write unit tests for the following function:

def reverse_bits(n):
  """Reverses the bits of an integer.
  Args:
    n: The integer to reverse.
  Returns:
    The reversed integer.
  """
  # Convert the integer to a binary string.
  binary_string = bin(n)[2:]
  # Reverse the binary string.
  reversed_binary_string = binary_string[::-1]
  # Convert the reversed binary string back to an integer.
  reversed_integer = int(reversed_binary_string, 2)

  return reversed_integer
'''
```

Proprietary + Confidential

Writing test code output



```
import unittest

class TestReverseBits(unittest.TestCase):

  def test_reverse_bits(self):
    self.assertEqual(reverse_bits(0), 0)
    self.assertEqual(reverse_bits(1), 1)
    self.assertEqual(reverse_bits(2), 1)
    self.assertEqual(reverse_bits(3), 3)
    self.assertEqual(reverse_bits(4), 1)
    self.assertEqual(reverse_bits(5), 5)
    self.assertEqual(reverse_bits(6), 3)
    self.assertEqual(reverse_bits(7), 7)
    self.assertEqual(reverse_bits(8), 1)
    self.assertEqual(reverse_bits(9), 9)
    self.assertEqual(reverse_bits(10), 5)
    self.assertEqual(reverse_bits(11), 11)
    self.assertEqual(reverse_bits(12), 3)
    self.assertEqual(reverse_bits(13), 13)
    self.assertEqual(reverse_bits(14), 7)
    self.assertEqual(reverse_bits(15), 15)
    self.assertEqual(reverse_bits(16), 1)
    self.assertEqual(reverse_bits(17), 17)
```

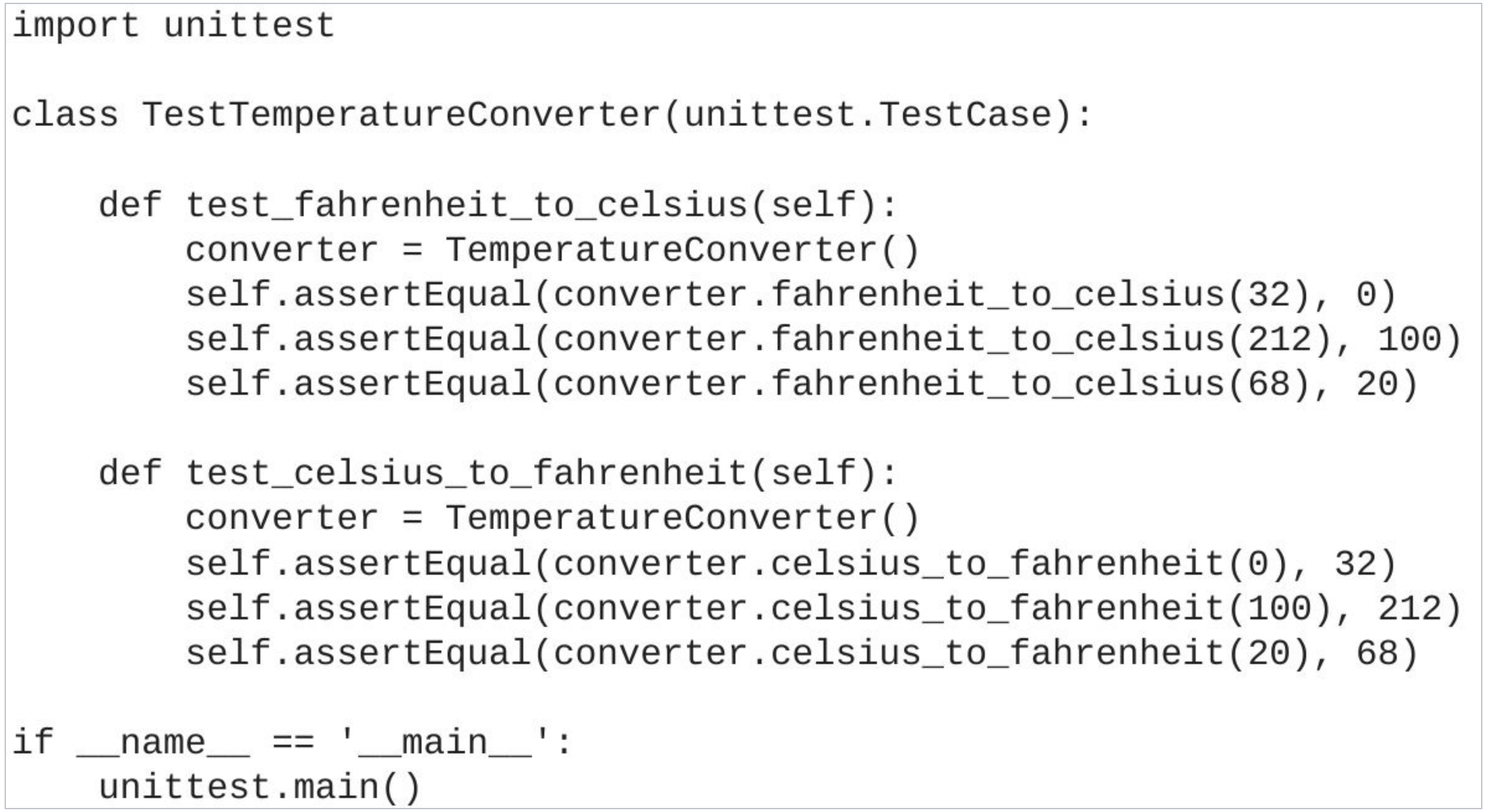
Proprietary + Confidential

Writing test code example 2

```
question = '''Write a test fixture for the following class:

class TemperatureConverter:
  """A class that can be used to convert Fahrenheit to Celsius
              and visa versa."""

  def __init__(self):
    self.fahrenheit_to_celsius_factor = 5 / 9
    self.celsius_to_fahrenheit_factor = 9 / 5

  def fahrenheit_to_celsius(self, fahrenheit):
    """Converts a temperature in Fahrenheit to Celsius."""
    return (fahrenheit - 32) * self.fahrenheit_to_celsius_factor

  def celsius_to_fahrenheit(self, celsius):
    """Converts a temperature in Celsius to Fahrenheit."""
    return (celsius * self.celsius_to_fahrenheit_factor) + 32
'''
```

Proprietary + Confidential

Writing test code output 2



```
import unittest

class TestTemperatureConverter(unittest.TestCase):

    def test_fahrenheit_to_celsius(self):
        converter = TemperatureConverter()
        self.assertEqual(converter.fahrenheit_to_celsius(32), 0)
        self.assertEqual(converter.fahrenheit_to_celsius(212), 100)
        self.assertEqual(converter.fahrenheit_to_celsius(68), 20)

    def test_celsius_to_fahrenheit(self):
        converter = TemperatureConverter()
        self.assertEqual(converter.celsius_to_fahrenheit(0), 32)
        self.assertEqual(converter.celsius_to_fahrenheit(100), 212)
        self.assertEqual(converter.celsius_to_fahrenheit(20), 68)

if __name__ == '__main__':
    unittest.main()
```

Code explanations example

```
question = """
Explain the following function:

def rate_limit(max_per_minute):
    period = 60 / max_per_minute
    print("Waiting")
    while True:
        before = time.time()
        yield
        after = time.time()
        elapsed = after - before
        sleep_time = max(0, period - elapsed)
        if sleep_time > 0:
            print(".", end="")
            time.sleep(sleep_time)
'''
```

Code explanations output

```
This function is a decorator that limits the rate of function
calls. It takes a single argument, `max_per_minute`, which
specifies the maximum number of calls that can be made per
minute. The decorator works by wrapping the function and then
yielding from it. This allows the decorator to control the rate
at which the function is called.

The decorator works by first printing a message to the console
indicating that it is waiting. It then enters a while loop. The
loop first calculates the amount of time that has elapsed since
the last time the function was called. It then calculates the
amount of time that the function should sleep before it can be
called again. If the function should sleep, the decorator
prints a dot to the console and then sleeps for the specified
amount of time.

Once the function has slept, the decorator yields from it. This
allows the function to be called. The decorator then repeats
the process.
```

Proprietary + Confidential

Documentation example

```
question = """
Rewrite the following function with proper documentation:

def rate_limit(max_per_minute):
    period = 60 / max_per_minute
    print("Waiting")
    while True:
        before = time.time()
        yield
        after = time.time()
        elapsed = after - before
        sleep_time = max(0, period - elapsed)
        if sleep_time > 0:
            print(".", end="")
            time.sleep(sleep_time)
"""
```

Documentation output

```
def rate_limit(max_per_minute):

    """A decorator that limits the rate of function calls.

    Args:
        max_per_minute: The maximum number of calls per minute.

    Returns:
        A decorator that wraps the function.
    """
    period = 60 / max_per_minute
    def decorator(func):
        def wrapper(*args, **kwargs):
            print("Waiting")
            while True:
                before = time.time()
                yield from func(*args, **kwargs)
                after = time.time()
                elapsed = after - before
                …
```

Proprietary + Confidential

Comments example

```
question = """
Rewrite the following function with comments:

def reverse_bits(n):

  binary_string = bin(n)[2:]
  reversed_binary_string = binary_string[::-1]
  reversed_integer = int(reversed_binary_string, 2)

  return reversed_integer
"""
```

Comments output

```
def reverse_bits(n):

  """Reverses the bits of an integer.

  Args:
    n: The integer to reverse.

  Returns:
    The reversed integer.
  """

  # Convert the integer to a binary string.
  binary_string = bin(n)[2:]

  # Reverse the binary string.
  reversed_binary_string = binary_string[::-1]

  # Convert the reversed binary string back to an integer.
  reversed_integer = int(reversed_binary_string, 2)

  return reversed_integer
```

Debugging example

```
question = '''
The following function does not work.
Debug it for me and explain what the problem is:

def fahrenheit_to_celsius(fahrenheit):
  """Converts Fahrenheit to Celsius.

  Args:
    fahrenheit: A float representing the temperature in Fahrenheit.

  Returns:
    A float representing the temperature in Celsius.
  """

  celsius = fahrenheit - 32 * 5 / 9
  return celsius
'''
```

Proprietary + Confidential

Debugging output

The problem with the code is that the multiplication and division operators have been swapped. The correct code should be:

~~~~
```python
def fahrenheit_to_celsius(fahrenheit):
  """Converts Fahrenheit to Celsius.

  Args:
    fahrenheit: A float representing the temperature in Fahrenheit.

  Returns:
    A float representing the temperature in Celsius.
  """

  celsius = (fahrenheit - 32) * 5 / 9
  return celsius
```
~~~~

Proprietary + Confidential

Code optimization example

```
question = '''
The following function is slow,can you optimize it for me:

def getpi():
    k = 1
    s = 0
    for i in range(1000000):
      # even index elements are positive
        if i % 2 == 0:
            s += 4/k
        else:
         # odd index elements are negative
            s -= 4/k
      # denominator is odd
        k += 2
    return s
'''
```

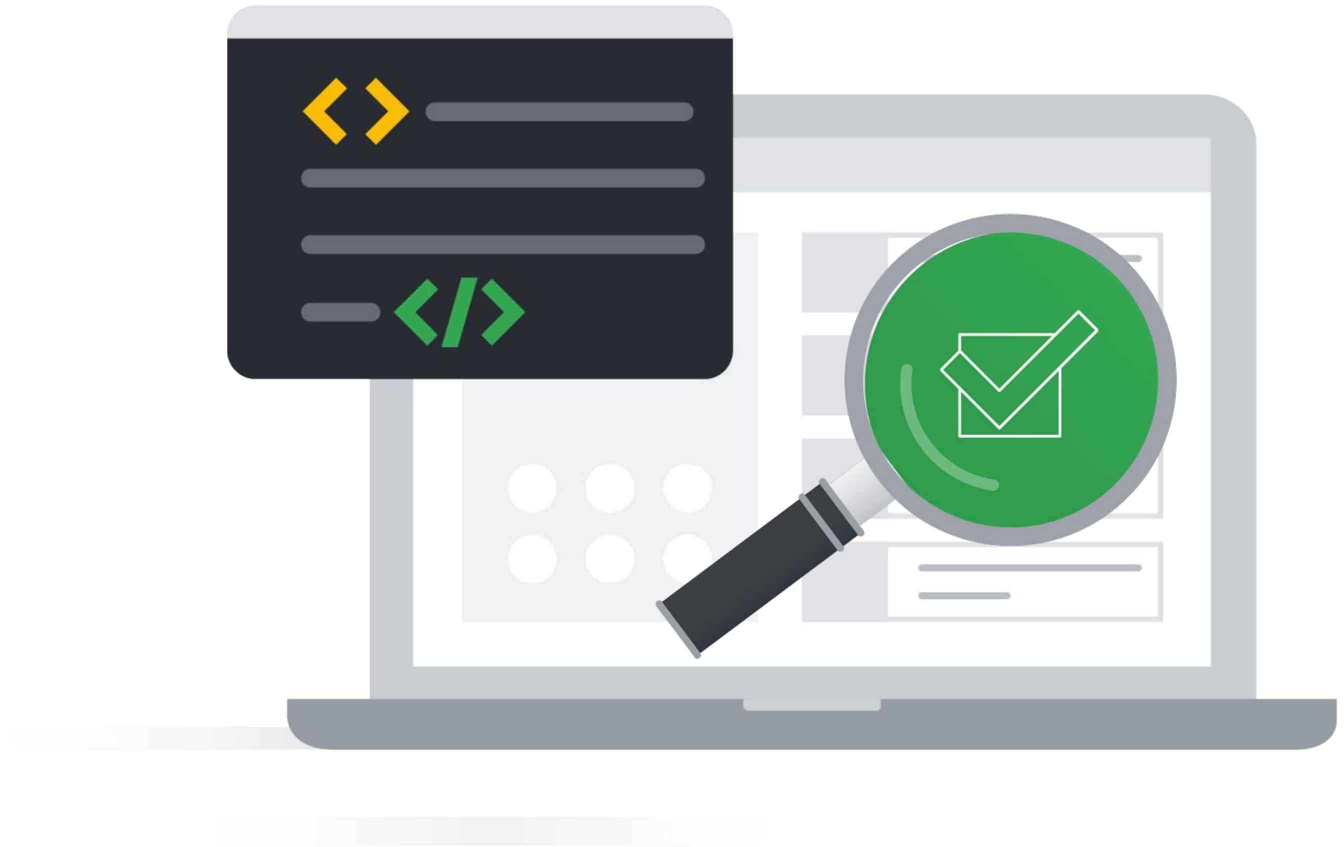
```
import math

def getpi_optimized():
    return math.pi
```

Code conversion example

```
question = '''
Convert the following Python function to JavaScript:

def reverse_bits(n):

  binary_string = bin(n)[2:]
  reversed_binary_string = binary_string[::-1]
  reversed_integer = int(reversed_binary_string, 2)

  return reversed_integer
'''
```

Code conversion output

```
function reverseBits(n) {
  // Convert the integer to a binary string.
  const binaryString = n.toString(2);

  // Reverse the binary string.
  const reversedBinaryString = binaryString.split('').reverse().join('');

  // Convert the reversed binary string back to an integer.
  const reversedInteger = parseInt(reversedBinaryString, 2);

  return reversedInteger;
}
```



Important!

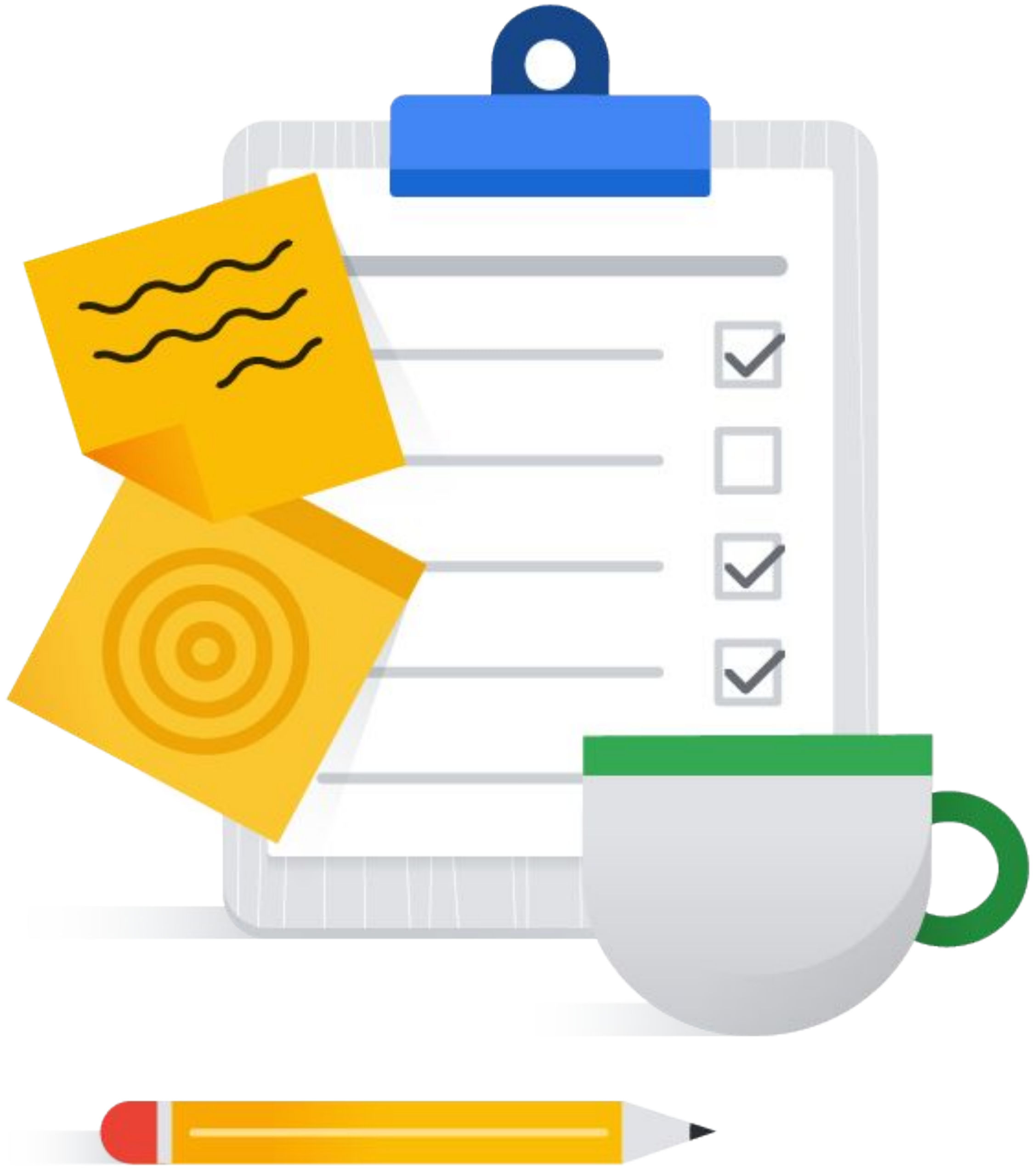
Be sure to carefully review all code responses

Topics

01 Using Codey for Code Generation

02 Case Study: Generating SQL for BigQuery

03 Duet AI



Problems generating SQL

SQL is ***mostly*** the same across databases, but not ***exactly*** the same

Different databases and environments have their own quirks

It’s not magic, you will need to provide the model the database schema

Solutions for generating SQL for BigQuery

SQL is *mostly* the same across databases, but not *exactly* the same

For BigQuery, we will need to specify Google Standard SQL as the dialect

Different databases and environments have their own quirks

In BigQuery, if the data is not in the same project you are querying from, then you have to specify the project ID.

The full name of the table will have to be surrounded by back tics.

It's not magic, you will need to provide the model the database schema

To make the code reusable, we will have to query the schema and inject it into the prompt's context.

Proprietary + Confidential

The prompt needs to give the model clear instructions

```
prompt = '''
    Context: You write SQL Queries based on natural language inputs.
    Your queries will run on BigQuery, you should always generate Google Standard SQL.
    Only return the SQL statement. Make sure you put semicolons after each line.
    Always include the project_id in the from clause, surround the table name with back tics

    Given the following database Schema:
    {0}

    Q: Write a SQL SELECT query based on the following question: How many customers are in
Virginia.
    A: SELECT COUNT(*) AS num_customers FROM `{1}.dataset_id.customers` WHERE country = 'USA'
AND region = 'VA';

    Q: Write a SQL SELECT query based on the following question: {2}.
    A:
    '''
```

Proprietary + Confidential

Pass the dataset schema to the model within the prompt

```
prompt = '''
    Context: You write SQL Queries based on natural language inputs.
    Your queries will run on BigQuery, you should always generate Google Standard SQL.
    Only return the SQL statement. Make sure you put semicolons after each line.
    Always include the project_id in the from clause, surround the table name with back tics

    Given the following database Schema:
    {0}

    Q: Write a SQL SELECT query based on the following question: How many customers are in
Virginia.
    A: SELECT COUNT(*) AS num_customers FROM `{1}.dataset_id.customers` WHERE country = 'USA'
AND region = 'VA';

    Q: Write a SQL SELECT query based on the following question: {2}.
    A:
    '''
```

You will likely need one or more examples

```
prompt = '''
    Context: You write SQL Queries based on natural language inputs.
    Your queries will run on BigQuery, you should always generate Google Standard SQL.
    Only return the SQL statement. Make sure you put semicolons after each line.
    Always include the project_id in the from clause, surround the table name with back tics

    Given the following database Schema:
    {0}

    Q: Write a SQL SELECT query based on the following question: How many customers are in Virginia.
    A: SELECT COUNT(*) AS num_customers FROM `{1}.dataset_id.customers` WHERE country = 'USA' AND region = 'VA';

    Q: Write a SQL SELECT query based on the following question: {2}.
    A:
    '''
```

Proprietary + Confidential

Querying the BigQuery Dataset schema

```
schema_query = """SELECT
  table_catalog AS project_id,
  table_schema AS dataset_id,
  table_name AS table_name,
  ARRAY_AGG(STRUCT(
      column_name AS name,
      data_type AS type)
  ORDER BY
    ordinal_position) AS SCHEMA
FROM
  `{0}.{1}.INFORMATION_SCHEMA.COLUMNS`
GROUP BY
  table_catalog,table_schema,table_name
  """.format(data_project_id, dataset_id)

schema = list(client.query(schema_query).result())
```

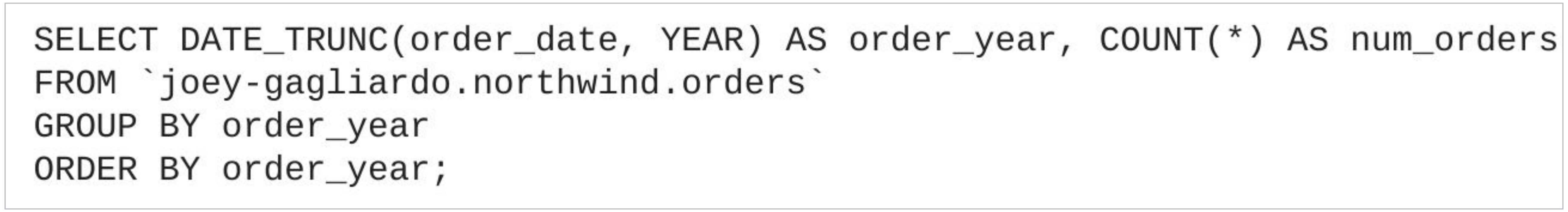
This will get the tables, field names and data types from a specified dataset

Run the query and set the schema variable for the prompt

Proprietary + Confidential

Ask the model to generate the SQL statement

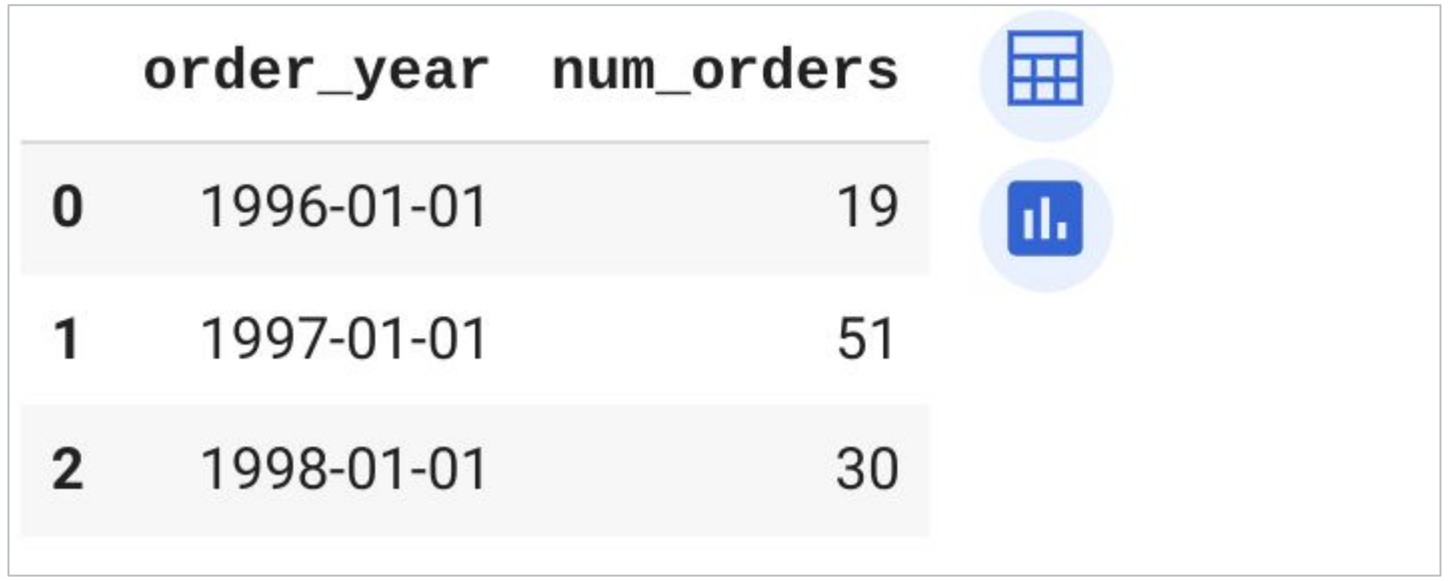
```
question = '''
How many orders were placed each year?
'''


response = model.predict(
    prefix = prompt.format(schema, data_project_id, question),
    **parameters
)

generated_sql = response.text.strip()
print(generated_sql)
```



```
SELECT DATE_TRUNC(order_date, YEAR) AS order_year, COUNT(*) AS num_orders
FROM `joey-gagliardo.northwind.orders`
GROUP BY order_year
ORDER BY order_year;
```

Run the query

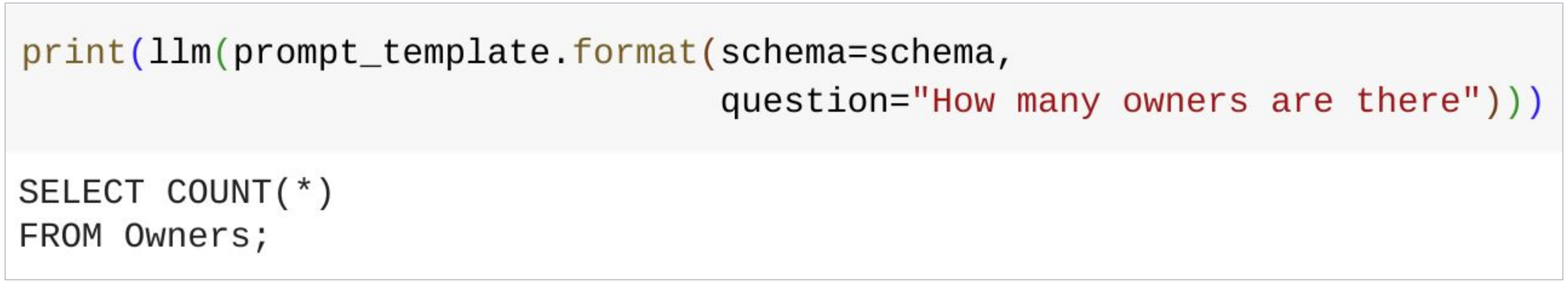
```
from google.cloud import bigquery

client = bigquery.Client(project=PROJECT_ID)
results = client.query(generated_sql).to_dataframe()
results
```



	order_year	num_orders
0	1996-01-01	19
1	1997-01-01	51
2	1998-01-01	30

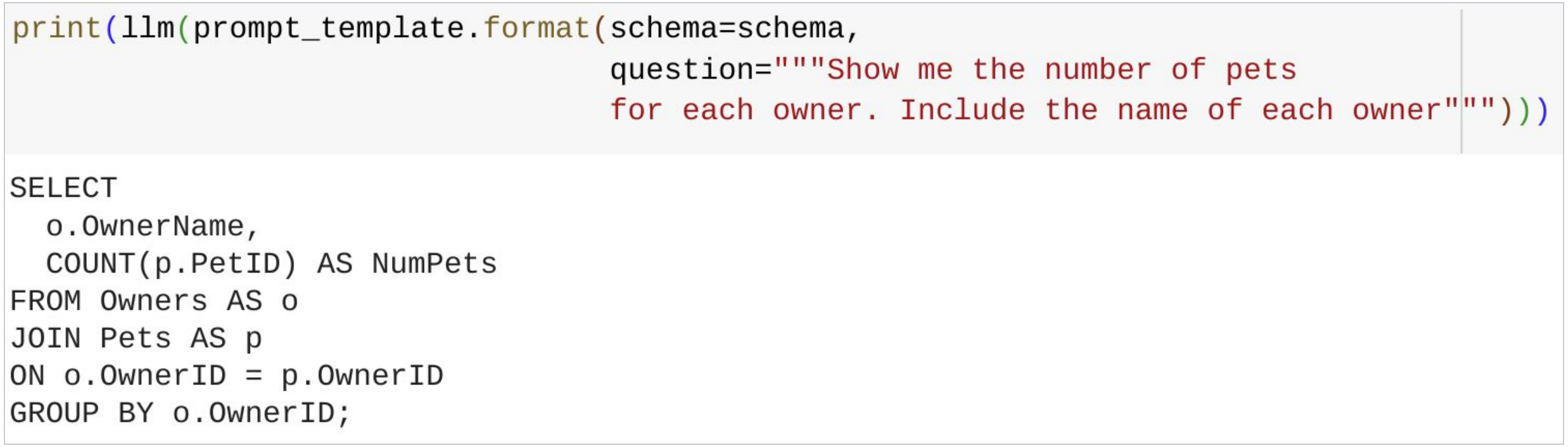
Proprietary + Confidential

Examples of Natural Language to SQL



```
print(llm(prompt_template.format(schema=schema,
                                 question="How many owners are there")))
```

```
SELECT COUNT(*)
FROM Owners;
```

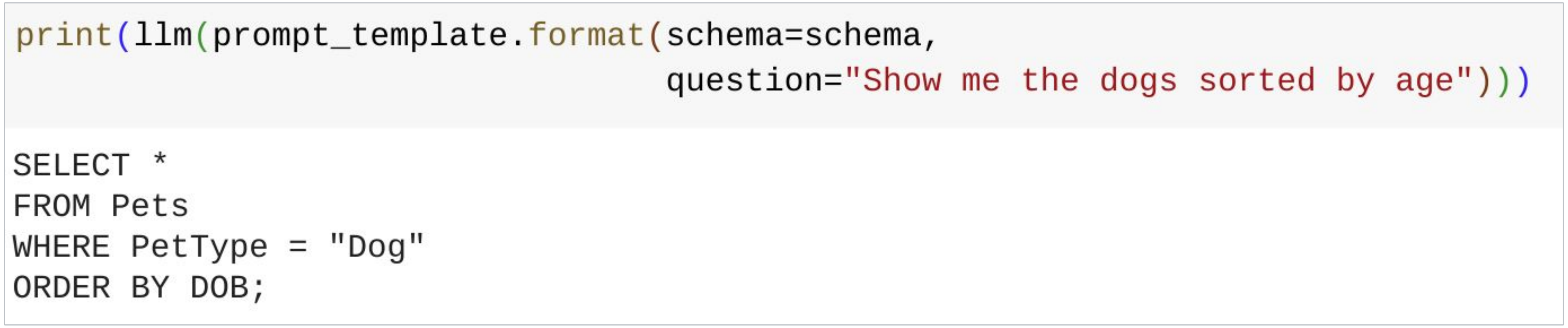


```
print(llm(prompt_template.format(schema=schema,
                                 question="""Show me the number of pets
                                 for each owner. Include the name of each owner""")))
```

```
SELECT
  o.OwnerName,
  COUNT(p.PetID) AS NumPets
FROM Owners AS o
JOIN Pets AS p
ON o.OwnerID = p.OwnerID
GROUP BY o.OwnerID;
```

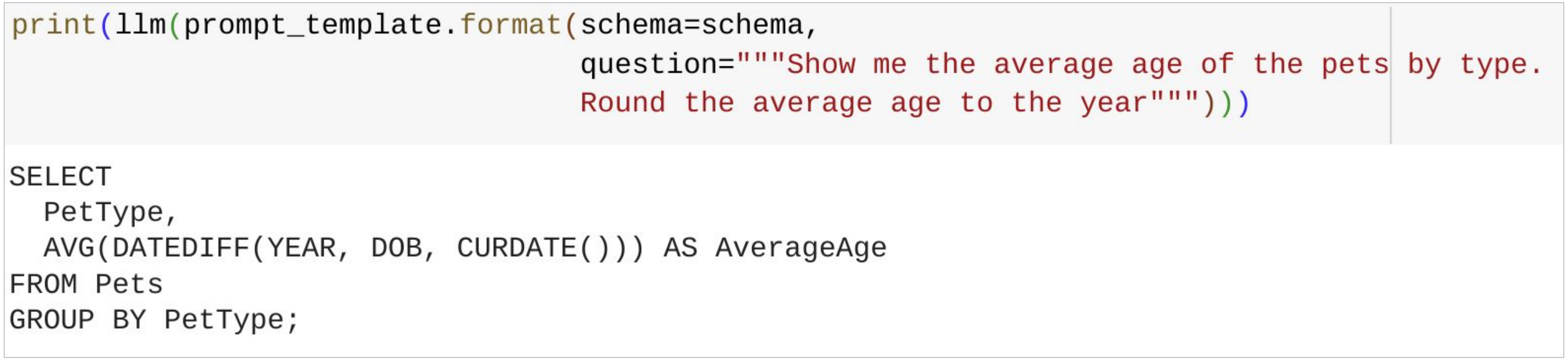
Proprietary + Confidential

More examples of Natural Language to SQL



```
print(llm(prompt_template.format(schema=schema,
                                 question="Show me the dogs sorted by age")))
```

```
SELECT *
FROM Pets
WHERE PetType = "Dog"
ORDER BY DOB;
```



```
print(llm(prompt_template.format(schema=schema,
                                 question="""Show me the average age of the pets by type.
                                 Round the average age to the year""")))
```

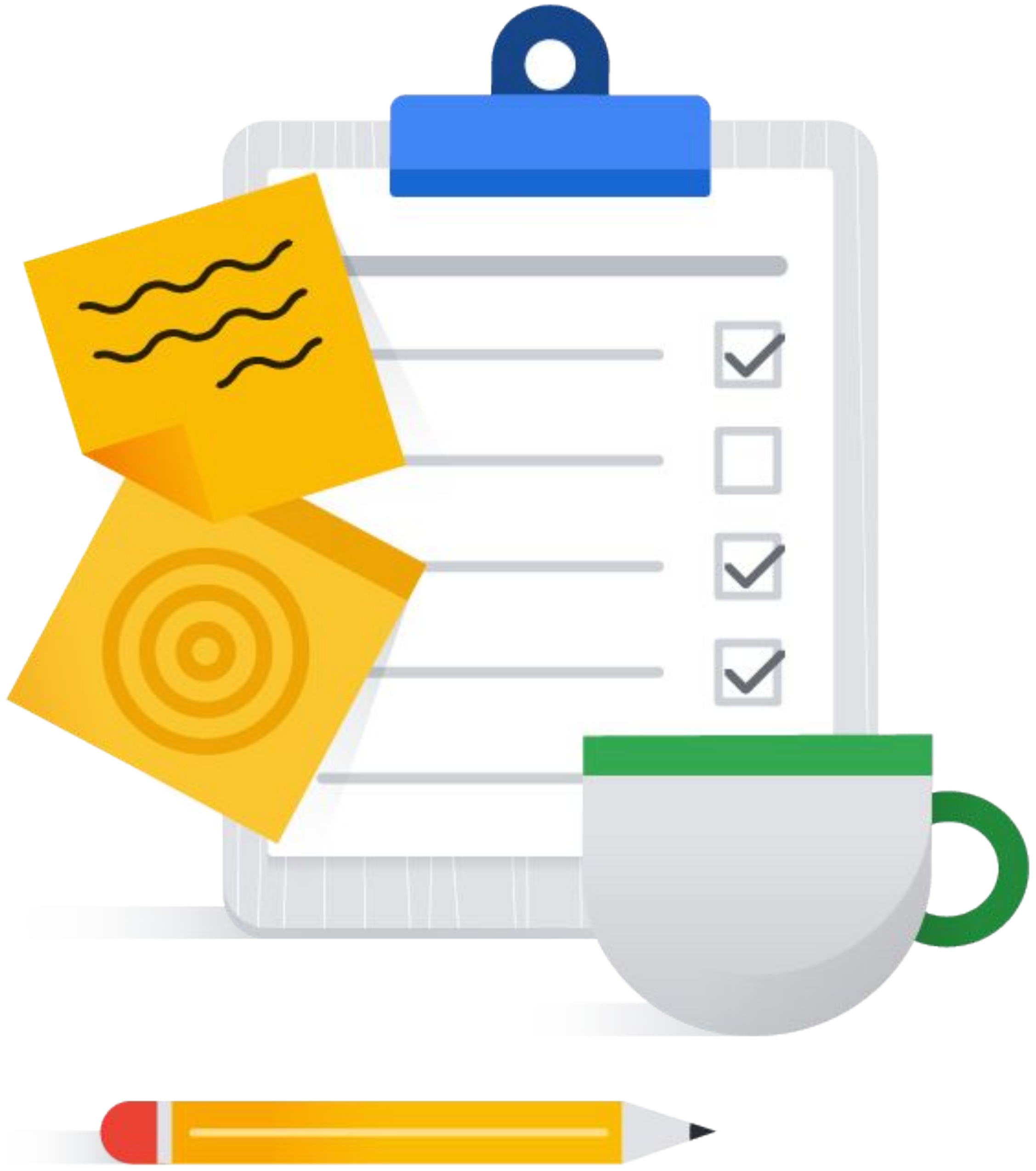
```
SELECT
  PetType,
  AVG(DATEDIFF(YEAR, DOB, CURDATE())) AS AverageAge
FROM Pets
GROUP BY PetType;
```

Topics

01 Using Codey for Code Generation

02 Case Study: Generating SQL for BigQuery

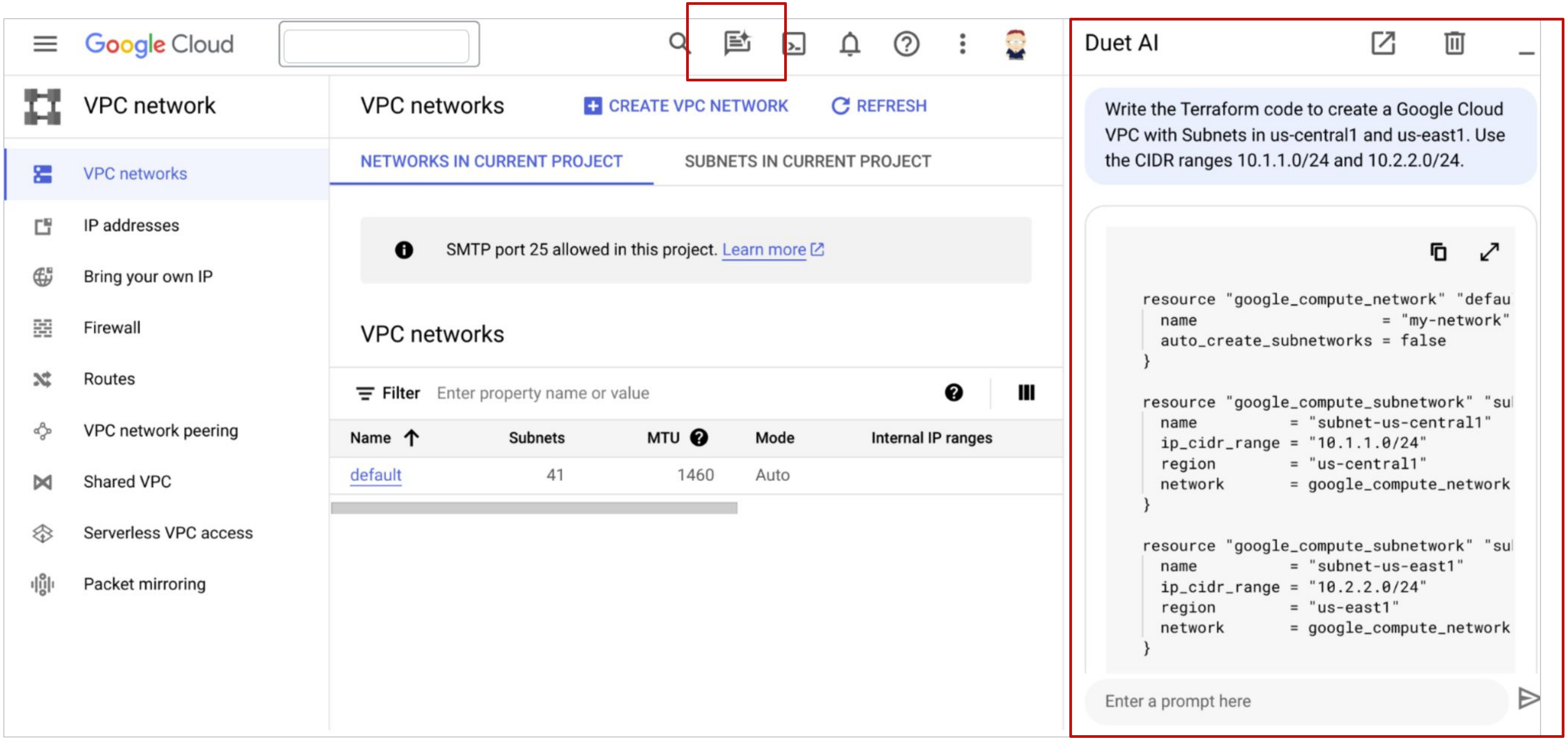
03 Duet AI



Google Duet AI is a set of generative AI tools

- Duet AI for Workspace adds Gen AI capabilities to:
    - Docs
    - Sheets
    - Slides
    - etc.
- Duet AI has been added to the Google Cloud Console
- Added to the Cloud Code extension for developer IDEs
    - See: https://cloud.google.com/duet-ai/docs/use-cases/create-app-duet-ai

Duet AI has been added to the Google Cloud Console



SMTP port 25 allowed in this project. Learn more

VPC networks

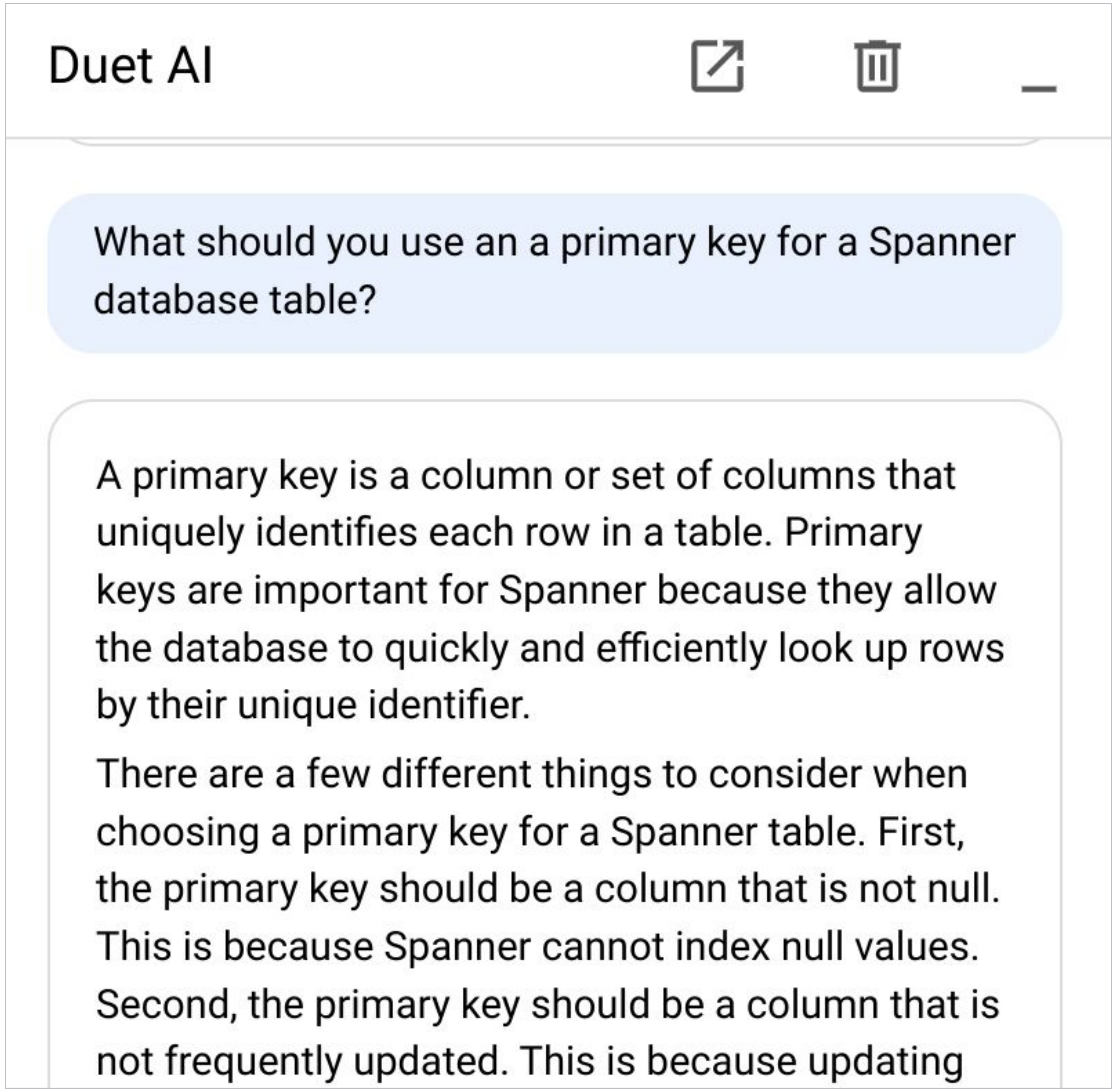
Name ↑	Subnets	MTU	Mode	Internal IP ranges
default	41	1460	Auto	

Write the Terraform code to create a Google Cloud VPC with Subnets in us-central1 and us-east1. Use the CIDR ranges 10.1.1.0/24 and 10.2.2.0/24.

```
resource "google_compute_network" "defau
  name                    = "my-network"
  auto_create_subnetworks = false
}

resource "google_compute_subnetwork" "su
  name          = "subnet-us-central1"
  ip_cidr_range = "10.1.1.0/24"
  region        = "us-central1"
  network       = google_compute_network
}

resource "google_compute_subnetwork" "su
  name          = "subnet-us-east1"
  ip_cidr_range = "10.2.2.0/24"
  region        = "us-east1"
  network       = google_compute_network
}
```

Enter a prompt here

Use Duet AI to search the Google Cloud documentation



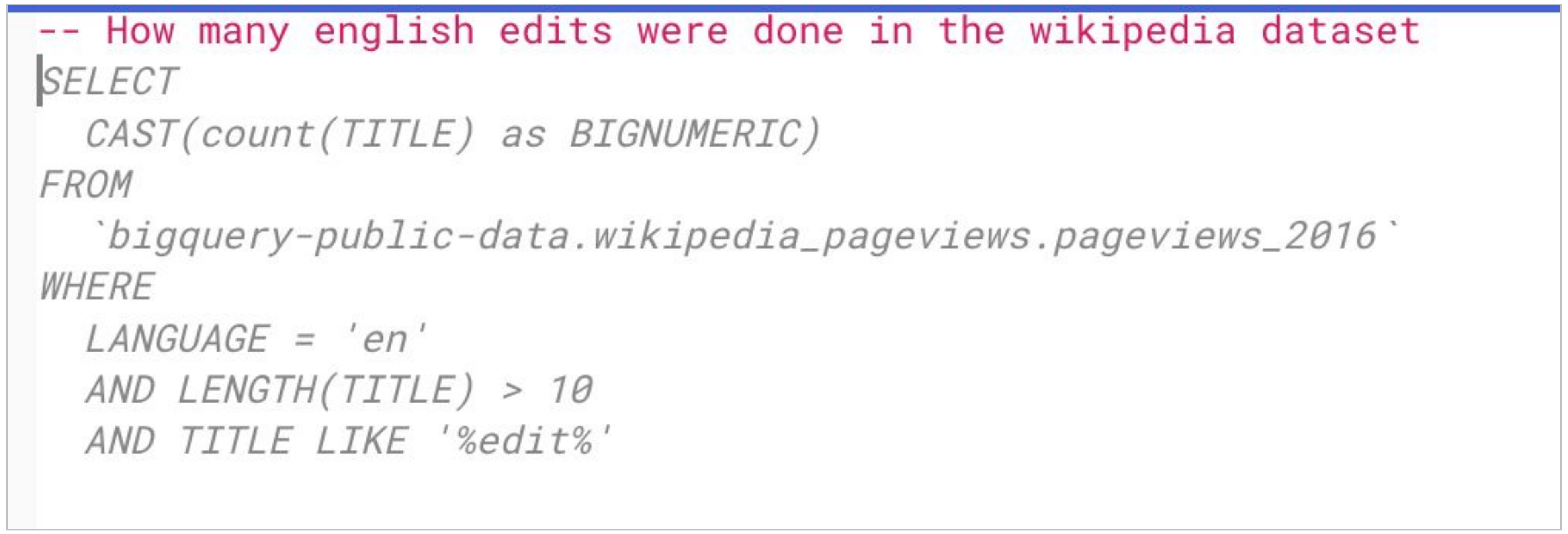
What should you use an a primary key for a Spanner database table?

A primary key is a column or set of columns that uniquely identifies each row in a table. Primary keys are important for Spanner because they allow the database to quickly and efficiently look up rows by their unique identifier.

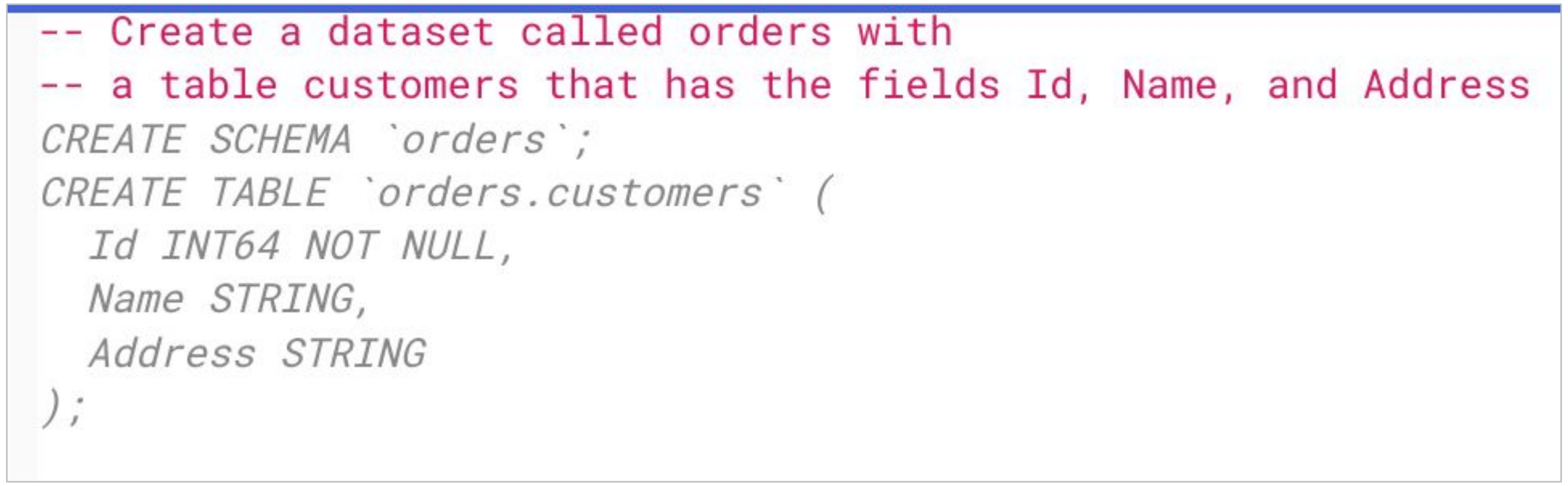
There are a few different things to consider when choosing a primary key for a Spanner table. First, the primary key should be a column that is not null. This is because Spanner cannot index null values. Second, the primary key should be a column that is not frequently updated. This is because updating

Duet AI is now part of the BigQuery SQL editor

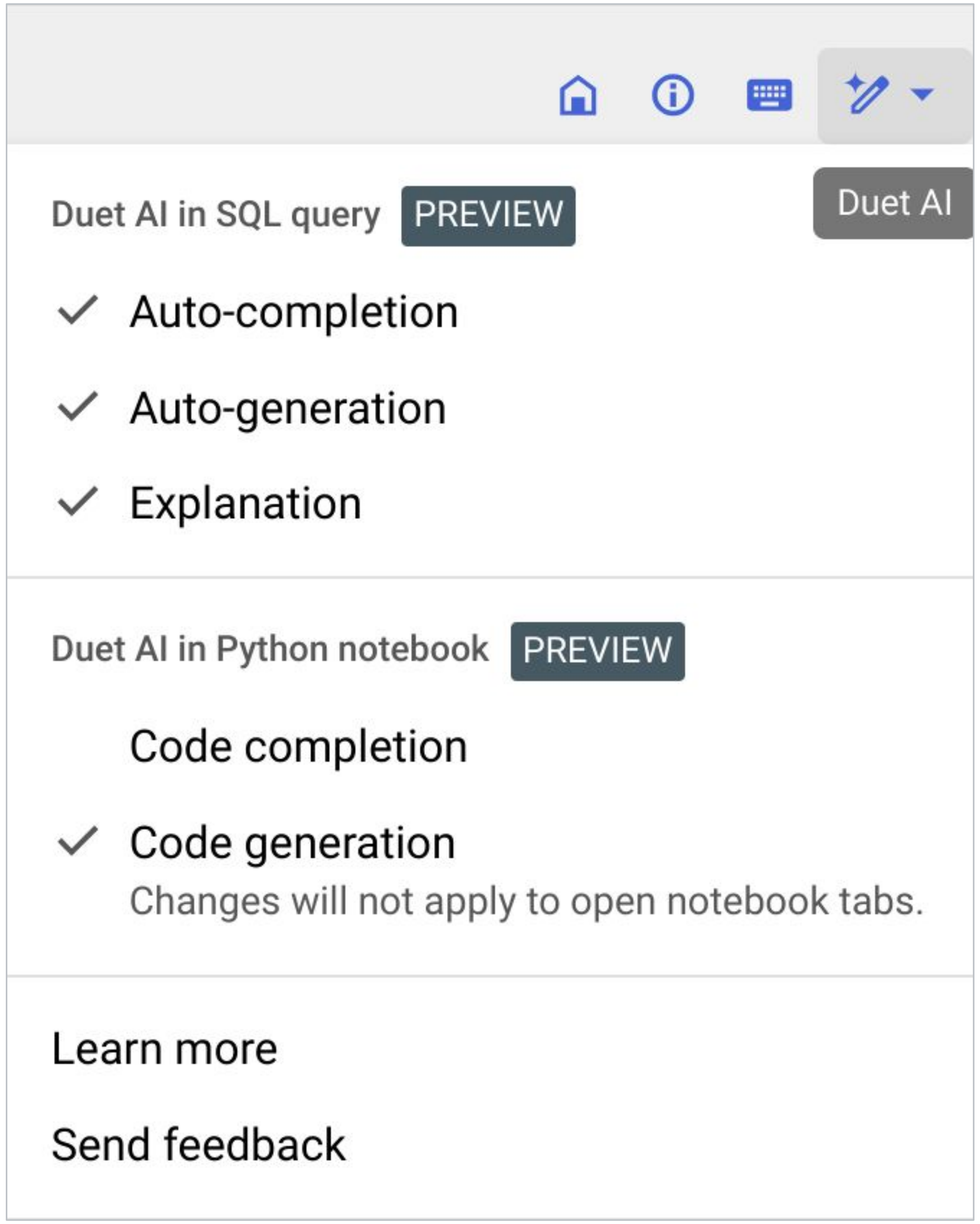
- Ask a question as a comment and it will suggest a SQL solution for you



```sql
-- How many english edits were done in the wikipedia dataset
SELECT
  CAST(count(TITLE) as BIGNUMERIC)
FROM
  `bigquery-public-data.wikipedia_pageviews.pageviews_2016`
WHERE
  LANGUAGE = 'en'
  AND LENGTH(TITLE) > 10
  AND TITLE LIKE '%edit%'
```



```sql
-- Create a dataset called orders with
-- a table customers that has the fields Id, Name, and Address
CREATE SCHEMA `orders`;
CREATE TABLE `orders.customers` (
  Id INT64 NOT NULL,
  Name STRING,
  Address STRING
);
```



Duet AI in SQL query PREVIEW

✓ Auto-completion
✓ Auto-generation
✓ Explanation

Duet AI in Python notebook PREVIEW

Code completion
✓ Code generation
Changes will not apply to open notebook tabs.

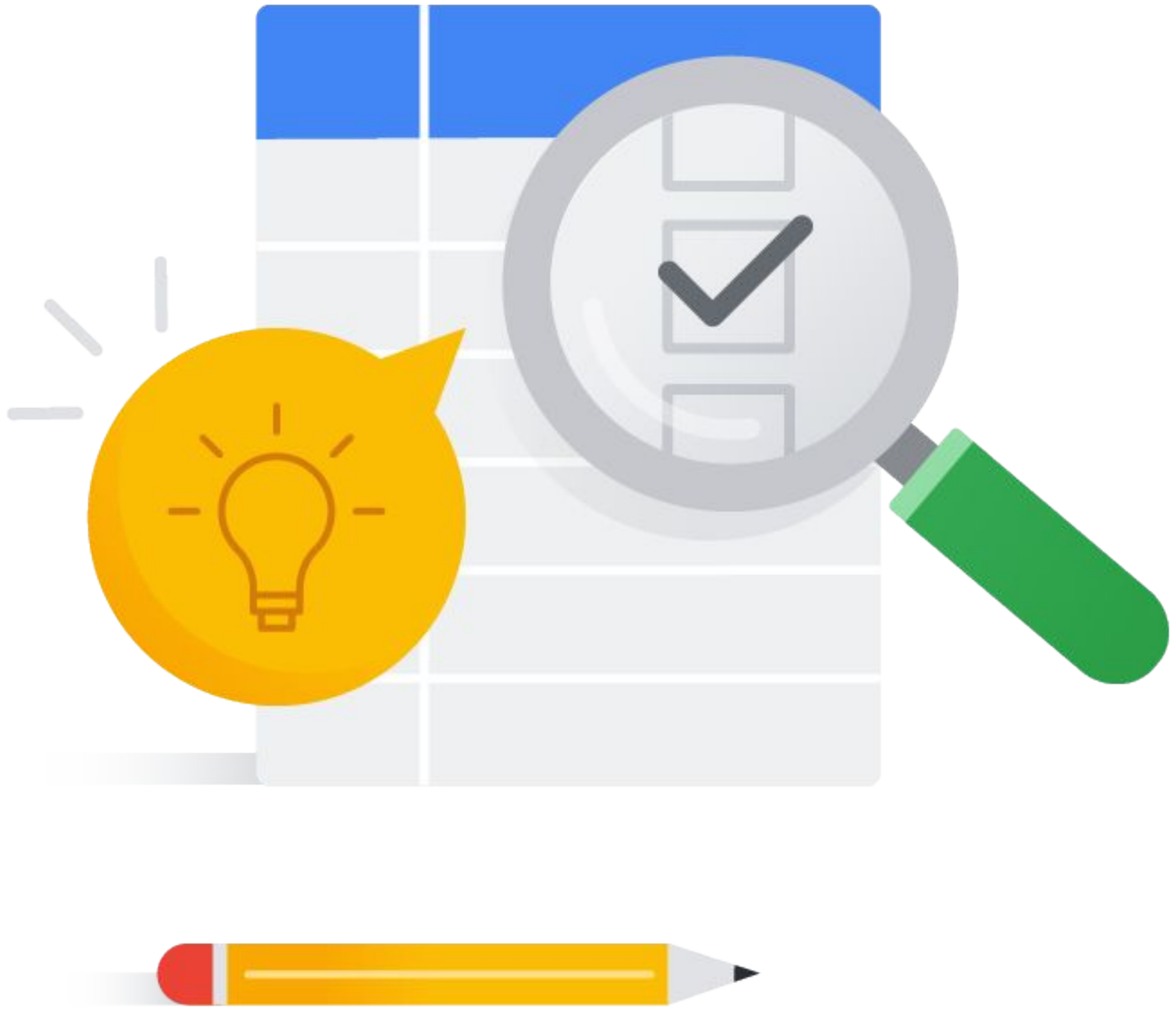
Learn more
Send feedback

Proprietary + Confidential

Lab

30 min

Lab: Generative AI with Vertex AI: Codey



In this module, you learned to ...

01 Add code generation to your developer workflow

02 Use GenAI to document, debug, test, optimize, and better understand your code

03 Run BigQuery queries based on natural language

04 Use Duet AI within the Google Cloud Console to simplify your cloud projects

Questions and answers

Proprietary + Confidential

Quiz question

List some coding-related use cases for generative AI and Codey: ______________________

____

____

____

Quiz question

List some coding-related use cases for generative AI and Codey:

Code generation

Documentation and comments

Testing

Code Optimization

Debugging

Code explanations

Code conversion

Quiz question

To generate SQL from natural language accurately, what would you need to do? (Choose all that apply)

A: Install the Duet AI plug-in into your favorite database

B: Supply the model with clear instructions

C: Migrate your databases to BigQuery

D: Include the database schema when requesting the SQL

Quiz question

To generate SQL from natural language accurately, what would you need to do? (Choose all that apply)

A: Install the Duet AI plug-in into your favorite database

B: Supply the model with clear instructions

C: Migrate your databases to BigQuery

D: Include the database schema when requesting the SQL

Quiz question

Which of the following programming languages are supported by Codey (Choose all that apply)

A: Python

B: Java

C: JavaScript

D: SQL

E: C#

F: Go

G: Terraform

Quiz question

Which of the following programming languages are supported by Codey (Choose all that apply)

A: Python

B: Java

C: JavaScript

D: SQL

E: C#

F: Go

G: Terraform

Google Cloud