Chapter 1

SQL for Data Preparation

1.1 组合数据

1.1.1 连接的类型

You will learn about three fundamental joins, which are illustrated in Figure 1.1—inner joins, outer joins, and cross joins:

外连接

完全外连接将返回左表和右表中的所有行，无论连接谓词是否匹配。对于满足连接谓词的行，两行将被组合起来，就像内部连接一样。对于不满足的行，两个表中的每一行都将被选择为单独的行，并为另一个表中的列填充 **NULL**。完全外连接是通过使用 **FULL OUTER JOIN** 子句并后跟连接谓词来调用的。

1.1.2 Subqueries

If a query only has one column, you can use a subquery with the **IN** keyword in a **WHERE** clause.

1.1.3 Unions

Please note that there are certain conditions that need to be kept in mind when using **UNION**. Firstly, **UNION** requires the subqueries to have the same number of columns and the same data types for the columns. If they do not, the query will fail to run. Secondly, **UNION** technically may not return all the rows from its subqueries. **UNION**, by default, removes all duplicate rows in the output. If you want to retain the duplicate rows, it is preferable to use the **UNION ALL** keyword.

1.1.4 Common Table Expressions(CTEs)

CTEs are simply a different version of subqueries. CTEs establish temporary tables by using the **WITH** clause. The one advantage of CTEs is that they can be designed to be recursive. **Recursive**

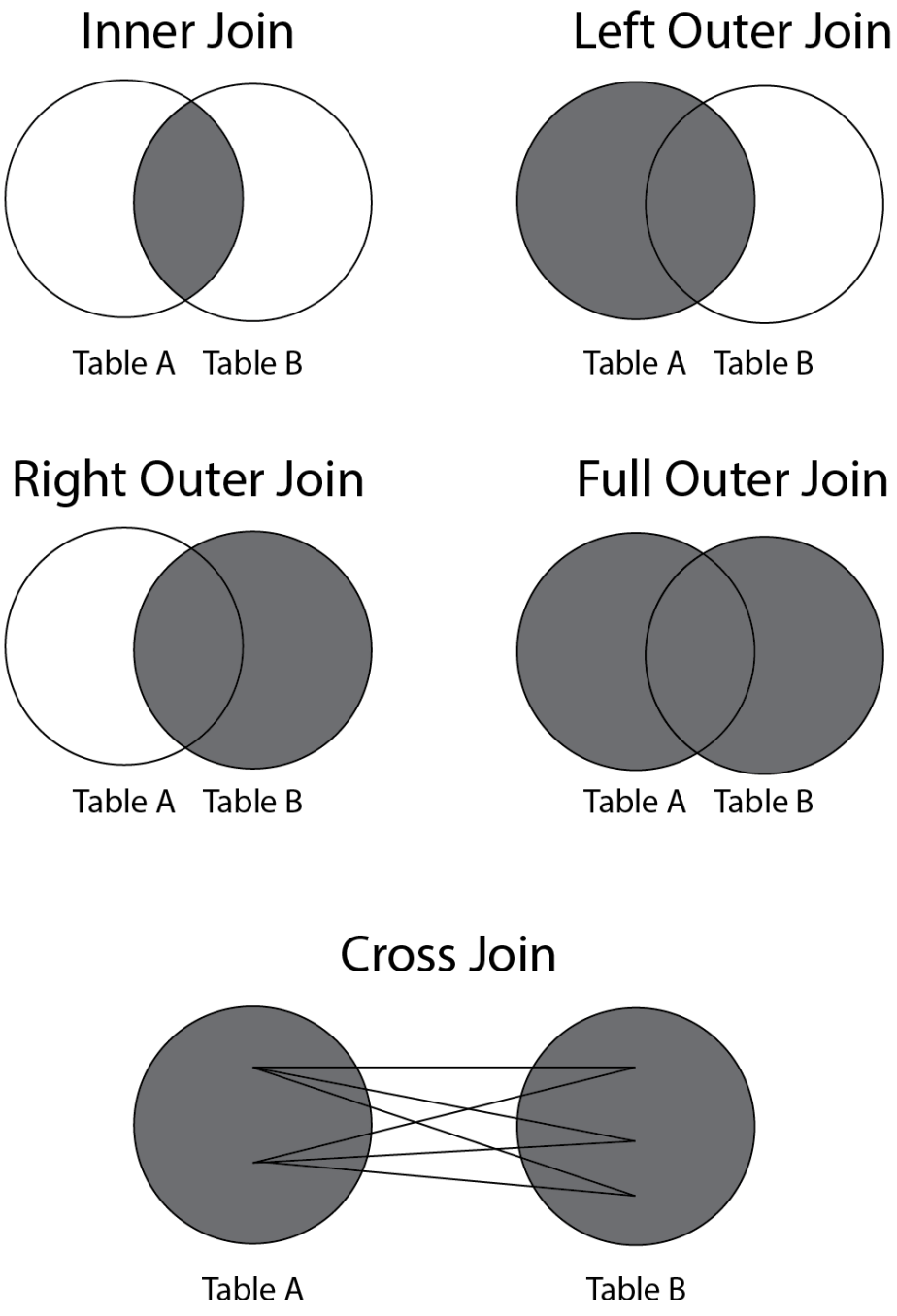

图 1.1: Major types of joins

CTEs can reference themselves. Because of this feature, you can use them to solve problems that other queries cannot.

1.2 Cleaning Data

1.2.1 The CASE WHEN Function

CASE WHEN is a function that allows a query to map various values in a column to other values.

1.2.2 The COALESCE Function

Another common requirement is to replace the NULL values with a standard value. This can be accomplished easily by means of the COALESCE function. COALESCE allows you to list any number of columns and scalar values, and, if the first value in the list is NULL, it will try to fill it in with the second value. The COALESCE function will keep continuing down the list of values until it hits a non-NULL value. If all values in the COALESCE function are NULL, then the function returns NULL.

1.2.3 The NULLIF Function

NULLIF is used as the opposite of COALESCE. While COALESCE is used to convert NULL into a standard value, NULLIF is a two-value function and will return NULL if the first value equals the second value.

1.2.4 The LEAST/GREATEST Functions

Two functions that come in handy for data preparation are the LEAST and GREATEST functions. Each function takes any number of values and returns the least or the greatest of the values, respectively. The simple use of this variable would be to replace the value if it is too high or low.

1.2.5 The Casting Function

To change the data type of a column, you simply need to use the **column::datatype** format, where column is the column name and datatype is the data type you want to change the column to.

Please note that not every data type can be cast to a specific data type.

1.3 Transforming Data

1.3.1 The DISTINCT and DISTINCT ON Functions

When looking through a dataset, you may be interested in determining the unique values in a column or group of columns. This is the primary use case of the DISTINCT keyword.

Another keyword related to DISTINCT is DISTINCT ON. Now, DISTINCT ON allows you to ensure that only one row is returned, and one or more columns are always unique in the set.

Chapter 2

Aggregate Functions for Data Analysis

2.1 Aggregate Functions

2.2 Aggregate Functions with the GROUP BY Clause

2.2.1 Ordered Set Aggregates

Function	Explanation
count(columnX)	Counts the number of rows in columnX that have a non-NULL value
count(*)	Counts the number of rows in the output table
min(columnX)	Returns the minimum value in columnX. For text columns, it returns the value that would appear first alphabetically
max(columnX)	Returns the maximum value in columnX
sum(columnX)	Returns the sum of all values in columnX
avg(columnX)	Returns the average of all values in columnX
stddev(columnX)	Returns the samples standard deviation of all values in columnX
var(columnX)	Returns the samples variance of all values in columnX
regr_slope(columnX, columnY)	Returns the slope of linear regression for columnX as the response variable and columnY as the predictor variable
regr_intercept(columnX, columnY)	Returns the intercept of linear regression for columnX as the response variable and columnY as the predictor variable
corr(columnX, columnY)	Calculates the Pearson correlation between columnX and columnY in the data

Function	Explanation
mode()	Returns the value that appears most often. In the case of a tie, it returns the first value in order
percentile_cont()	Returns a value corresponding to the specified fraction in the ordering, interpolating between adjacent input items if needed
percentile_disc()	Returns the first input value whose position in the ordering equals or exceeds the specified fraction

Chapter 3

3.1 Window Functions

3.1.1 The Basics of Window Functions

Figure 3.1 the dataset is ordered using customer_id, which happens to be the primary key. As such each row has a unique value and forms a value group. The first value group, without any row before it, forms its own window, which contains only the first row. The second value group's window will contain both itself and the row before it, which means the first and second row. Then the third value group's window will contain itself and the two rows before it, and so on and so forth. Every value group has its window. Once the windows are established, for every value group, the window function is calculated based on the window. In this example, this means COUNT is applied to every window. Thus, value group 1 (the first row) gets 1 as the result since its Window 1 contains one row, value group 2 (the second row) gets 2 since its Window 2 contains two rows, and so on and so forth. The results are applied to every row in this value group if the group contains multiple rows. Note that the window is used for calculation only. The results are assigned to rows in the value group, not assigned to the rows in the window.

图 3.1: Windows for customers using COUNT(*) ordered by the customer_id window query

表 3.1: Statistical window functions

Name	Description
row_number	Number the current row within its partition starting from 1
dense_rank	Rank the current row within its partition without gaps
rank	Rank the current row within its partition with gaps
lag	Return a vale evaluated at the row that is at a specified physical offset row before the current row within the partition
lead	Return a value evaluated at the row that is offset rows after the current row within the partition
ntile	Divide rows in partition as equally as possible and assign each row an integer starting from 1 to the argument value

3.2 Statistics with Window Functions

Note

One question regarding RANK() is the handling of tied values. RANK() is defined as the rank of rows, not the rank of values. For example, if the first two rows have a tie, the third row will get 3 from the RANK() function. DENSE_RANK() could also be used just as easily as RANK(), but it is defined as the rank of values, not the rank of rows. In the example above, the value of DENSE_RANK() for the third row will be 2 instead of 3, as the third row contains the 2nd value in the list of values.

3.3 Summary

In this chapter, you learned about the window functions, which generate output for a row based on its position inside the dataset or subgroups within the dataset. This is different from the simple functions you learned in SQL for Data Preparation, that generates an output for a row regardless of the characteristics of the dataset, and different from the aggregate functions you learned in Aggregate Functions for Data Analysis, that generates an output for all rows in a dataset or subgroups in the dataset.

You learned some of the most common window functions including COUNT, SUM, and RANK. You also learned how to construct a basic window using OVER. The output of window function depends on the current row's position in the dataset or subgroups within the dataset, which is called partition, as well as the collection of rows required by the calculation, which is called window. As such there are several keywords that may impact how the calculation is done, such as PARTITION BY, ORDER BY, and window frame keywords. The PARTITION BY clause determines the partition, the ORDER BY clause determines the position of the row within the partition, and the window frame keywords determine the range and size of the window. You then learned how to use window functions to get analytical insights. For example, by defining window frame over a daily summary such as daily sales, you can create rolling statistics, and gain useful insights into the time trend of the sales.

Chapter 4

导入与导出数据

4.1 The COPY Command

4.1.1 Running the psql Command

The syntax of the psql command is as follows:

```
psql -h <host> -p <port> -d <database> -U <username>
```

In this command, you pass in flags that provide the information needed to make the database connection. In this case, you have the following:

- -h is the flag for the hostname. The string that comes after it (separated by a space) should be the hostname for your database, which can be an IP address, a domain name, or localhost if it is run on the local machine.
- -p is the flag for the database port. Usually, this is 5432 for PostgreSQL databases.
- -d is the flag for the database name. The string that comes after it should be the database name.
- -U is the flag for the username. It is succeeded by the username.

4.1.2 Creating Temporary Views

4.1.3 Configuring COPY and \COPY

There are several options that you can use to configure the COPY and \COPY commands:

- FORMAT: format_name can be used to specify the format. The options for format_name are csv, text, or binary. Alternatively, you can simply specify CSV or BINARY without the FORMAT keyword, or not specify the format at all and let the output default to a text file format.
- DELIMITER: delimiter_character can be used to specify the delimiter character for CSV or text files (for example, for CSV files, or | for pipe-separated files).

- NULL: null_string can be used to specify how NULL values should be represented (for example, whether blanks represent NULL values or NULL if that is how missing values should be represented in the data).
- HEADER: This specifies that the header should be output.
- QUOTE: quote_character can be used to specify how fields with special characters (for example, a comma in a text value within a CSV file) can be wrapped in quotes so that they are ignored by COPY.
- ESCAPE: escape_character specifies the character that can be used to escape the following character.
- ENCODING: encoding_name allows the specification of the encoding, which is particularly useful when you are dealing with foreign languages that contain special characters or user input.

4.1.4 Using COPY and \COPY to Bulk Upload Data to Your Database

The COPY and \COPY commands are far more efficient at uploading data than an INSERT statement. There are a few reasons for this:

- When using COPY, there is only one push of a data block, which occurs after all the rows have been properly allocated.
- There is less communication between the database and the client, so there is less network latency.
- PostgreSQL includes optimizations for COPY that would not be available through INSERT.

4.1.5 Using COPY and \COPY to Bulk Upload Data to Your Database

Note

For these maintenance tasks, you can use `pg_dump` for a specific table and `pg_dumpall` for an entire database or schema. These commands even let you save data in a compressed (tar) format, which saves space. Unfortunately, the output format from these commands is typically SQL, and it cannot be readily consumed outside of PostgreSQL. Therefore, it does not help you with importing or exporting data to and from other analytics tools, such as Python.

4.1.6 What is SQLAlchemy?

SQLAlchemy is a Python SQL toolkit and Object-Relational Mapper (ORM) that maps representations of objects to database tables. An ORM builds up mappings between SQL tables and programming language objects; in this case, Python objects. For example, in the following figure, there is a customer table in the database. The Python ORM will thus create a class called customer and keep the content in the object synchronized with the data in the table. For each row in the customer table, a customer object will be created inside the Python runtime. When there are changes (inserts, updates, and/or deletes), the ORM can initialize a sync and make the two sides consistent.

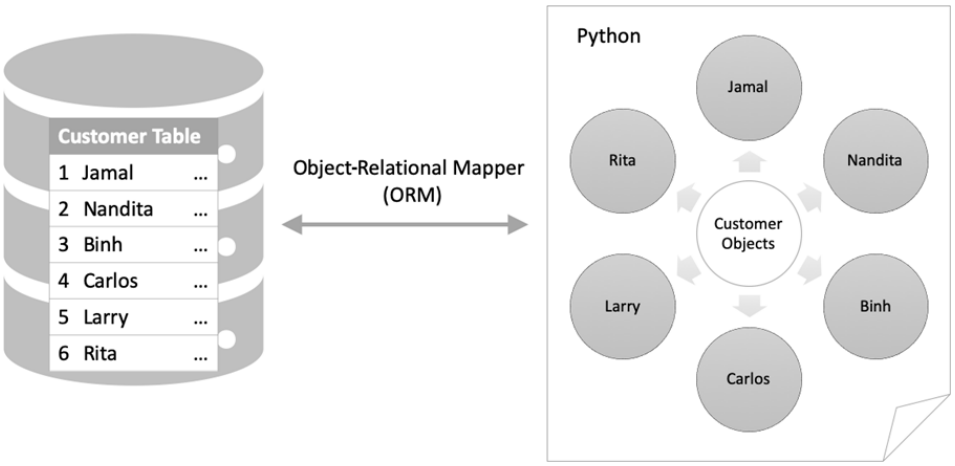

图 4.1: An ORM maps rows in a database to objects in memory

4.1.7 Writing Data to the Database Using Python

If you have your data in a pandas DataFrame, you can write data back to the database using the pandas `to_sql(⋯)` function, which requires two parameters: the name of the table to write to and the connection. Best of all, the `to_sql(⋯)` function can also create the target table for you by inferring column types using a DataFrame's data types.

4.1.8 Best Practices for Importing and Exporting Data

At this point, you have seen several different methods for reading and writing data between your computer and your database. Each method has its own use case and purpose. Generally, there are going to be two key factors that should guide your decision-making process:

- You should try to access the database with the same tool that you will use to analyze the data. As you add more steps to get your data from the database to your analytics tool, you increase the ways in which new errors can arise. When you cannot access the database using the same tool that you will use to process the data, you should use psql to read and write CSV files to your database.
- When writing data, you can save time by using the COPY or \COPY commands.

4.2 Going Passwordless

A .pgpass file specifies the parameters that you use to connect to your database, including your password. All of the programmatic methods of accessing the database discussed in this chapter (using either psql or Python) will allow you to skip the password parameter if your .pgpass file contains the password for the matching hostname, database, and username. This not only saves you time but also increases the security of your database because you can freely share your code without having to worry about passwords embedded in the code.

On Unix-based systems and macOS, you can create the .pgpass file in your home directory. On Windows, you can create the file in `%APPDATA%\postgresql\ pgpass.conf`.(如果路径不存，手动新建

文件夹和文件) `%APPDATA%` is a Windows system value that points to the current application data folder. You can get the actual value of it by opening Windows Explorer, typing the exact word `%APPDATA%`, into the address bar and hitting Enter. The folder you are in is the folder this `%APPDATA%` value points to. The .pgpass file should contain one line for every database connection that you want to store, and it should follow this format (customized for your database parameters):

```
hostname:port:database:username:password
# localhost:5432:sql4da:postgres:my_password
```

创建完之后，`psql -h localhost -p 5432 -d sql4da -U postgres` 就不再需要输入密码了。