Learning SQL

Final Project for Data Management and Database Design

Preface

The aim of this document is to create a wonderful learning experience for the subject Data Management and Database Design (DMDD INFO 6210) in SQL. This can be refered to as a guide for the beginers and/or refresher course for veterans in this field.

Introduction

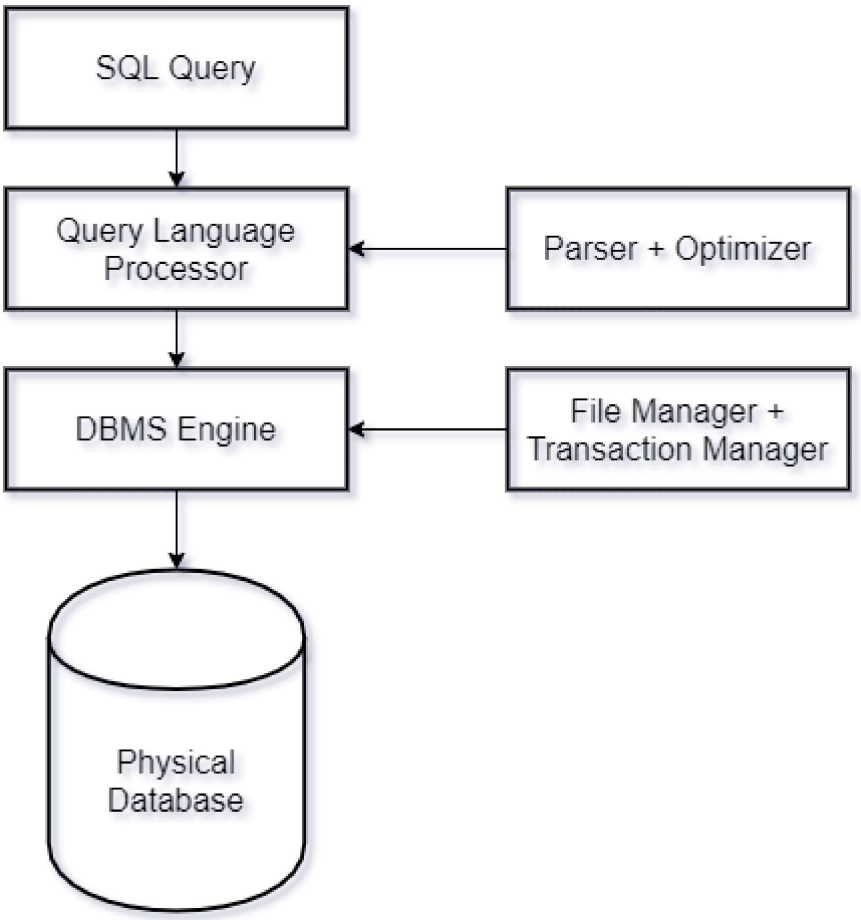
Structured Query Language, also known as SQL, is used to communicate between database. This standardization for communicating with relational database management systems and/or it's data retrieval from a database has been set by American National Standards Institute (ANSI). Some of the common Relational Database Management Systems (RDBMS) are Oracle, Sybase, Microsoft SQL Server, Access, Ingres, and more. Having the common feature of supporting SQL, the database systems also have their own added proprietary extensions which are indigenous to their systems. However, the standard SQL commands like "SELECT", "INSERT", "UPDATE", "DELETE", "CREATE", and "DROP" can be used to achieve almost every state that one needs from the database.

SQL has various advantages which has help gain its property.

- `Lets user to access data in the RDBMS (relational database management systems)`
- `Lets user add description to the data`
- `Lets user define the structure of the database and manipulates its data`
- `Lets user integrate additional modules, libraries and pre-compilers`
- `Lets user control the creation and deletion(dropping) of database and/or tables`
- `Lets user design views, stored procedures and functions in a database`
- `Lets user set permissions on the created tables, procedures and views`

Processing SQL

While executing an SQL command in for any RDBMS, the system determines the most efficient way to carry out the request given, and SQL engine understands on how to interpret the task assigned. There are various gears included in this process. They are: • Query Dispatcher • Optimization Engines • Classic Query Engine • SQL Query Engine and more A classic query engine processes all the non-SQL queries, whereas an SQL query engine would not handle logical files. A simple diagram depicting the SQL Architecture.



Understanding Tables

A Relational Database System (RDBS) consists of one or more objects known as tables. The data or information relevant to the database, is stored on these tables. The tables are uniquely identified with the names assigned to them and are contained of rows and columns. The columns consist of a name of the column, the data type, and any other attribute relevant to that column. The rows consist of the records or relevant data with respect to the columns specified.

Imports

In [1]:
```
import pandas as pd
import json
import csv
from io import StringIO
import mysql.connector
```

In [2]:
```
# Displaying a sample table "poe_stats" which displays details of players playing the
game "Path of Exile"

# Please change password here in the field 'passwd'

connection = mysql.connector.connect(
    host="localhost",
    user="root",
    passwd="password",
    database="poe_dmdd"
)

# Setting up the cursor
crsr = connection.cursor()

# Accessing the table from the database
sql = '''
SELECT *
FROM poe_stats;
'''

# Executing the SQL command
crsr.execute(sql)

# Fetching results
result = crsr.fetchall()

final_result = pd.DataFrame(result)
headers = [i[0] for i in crsr.description]
final_result.columns = headers
final_result
```

Out[2]:

	rank	name	level	class	
0	1	Tzn_NecroIsFineNow	100	Necromancer	3dcddd59f5088893f734f39686350990dae168cc4f4
1	1	RaizNeverFirstQT	100	Necromancer	8f3216db5ac9106c287a834731aafc83c387138f28f
2	1	GucciStreamerAdvantage	100	Necromancer	c6ec2dae3855c551e0597c06ef2da06fbb5512487de
3	1	ChiroxPrime	100	Slayer	c861372da792be0b22c45bf437ccd58437c52e9455e
4	2	Cool_NecroIsFineNow	100	Deadeye	24ae924ceed7989ef3d3d6772612832bb467a609435
...	...	...	...	...	
59771	14999	ПроклятьеРекласта	89	Necromancer	d33b4f6e08c10e365765f9a36a8f36d561fd1d86f10e
59772	15000	IshibashiSummoner	94	Necromancer	5764cfa387e0a87a4bebc1a3c5017e92de8bbb06445
59773	15000	BLively	73	Slayer	9ac75ab75a47cee8a9dfb0a31912df8909720a8b20f
59774	15000	vawddvaw	89	Gladiator	cf02dfc0c90b2df9c7ac76bbedd91e93c2a8a2ca629
59775	15000	Reselin	53	Necromancer	f7ffda5ca2490546344d32930693f8299930ce9451d

59776 rows × 10 columns

SQL Commands

The standard SQL commands to interact with a relational database are CRUD operations. They are written in the command form of CREATE, SELECT, INSERT, UPDATE, DELETE and DROP. The sa id commands can be classified into the given groups based on their nature.

Data Definition Language – DDL

CREATE
    This command creates a new table, a view (of a table), or other similar objects in the database

ALTER
    This command modifies an existing database object, for instance a table.

DROP
    This command deletes an entire table, a view (of a table), or other similar objects in the database

Data Manipulation Language – DML

SELECT
    This command retrieves certain record(s) from one or more tables in accordance with th e set parameters

INSERT
    This command creates record(s) for the table selected

UPDATE
    This command modifies existing record(s) for the table selected

DELETE
    This command deletes existing record(s) for the table selected

Data Control Language – DCL

GRANT
    This command gives privilege(s) to selected user

REVOKE
    This command takes back the privilege(s) granted to the selected user

SQL Constraints

The rules enforced on the data columns in a table are known as constraints. These rules li
mit the input data type into the table. These constraints ensure the accuracy and reliabil
ity of the data from the tables in the database.

Constraints can be set on a column or table wide. Column level constraints would only be a
pplicable to one column whereas, table level constraints would be applicable to the entire
table.

The most commonly used constraints available in SQL

- NOT NULL
  - This ensures that a column cannot contain a NULL value
- DEFAULT
  - A default value is provided when no such input is specified for the column
- UNIQUE
  - This ensures that all values in the column are different from other values
- PRIMARY Key
  - The data in the database is uniquely identified by the row/record
- FOREIGN Key
  - The data from another database is uniquely identified by the row/record
- CHECK
  - This ensures that all the values in a column satisfy certain conditions
- INDEX
  - This is used to create and retrieve data from a database instantaneously

Data Integrity

Entity Integrity

This ensures that there are no duplicate rows present in a table

Domain Integrity

This enforces all entries for a given columns are valid by restricting the type, the format, or the range of values

Referential Integrity

This ensures that the rows of the table cannot be deleted, which are used by other records in the database

User-Defined Integrity

This enforces some specific business rules that do not fall into either of the before mentioned Integrities; Entity, Domain or Referential

Database Normalization

The process of efficiently organizing data in a database is referred to as Database Normalization. This process is performed due to two main reasons

- To eliminate redundancy data in the database, i.e. storing identical data in more than one table
- To ensure the data dependencies are logical

These goals also help ensure that they reduce the amount the space a database would require and ensures that data is stored logically. The guidelines for normalization should be followed to create a good database structure.

The guidelines for the normalization are divided into normal forms. The aim of the normal forms is to help organize the database structure, which helps compile the rules of the First Normal Form (1NF), then with Second Normal Form (2NF), and finally with Third Normal Form (3NF). There are more normal forms after the third, like forth normal form, fifth normal form and so on. But the Third Normal Form is enough to achieve a good database structure.

First Normal Form (1NF)

- (First rule) The data items are to be defined, which helps decide the columns in a table
- Insert the relevant data items into the table
- (Second rule) There should be no repeating groups of data
- (Third rule) There should be a primary key for the table created

Second Normal Form (2NF)

- All the conditions of the First Normal Form must be followed
- There should not be any partial dependencies of any column on the primary key of the table

Third Normal Form (3NF)

- All the conditions of the Second Normal Form must be followed
- All non-primary fields must be dependent on the primary key. No transitive dependencies must be present between the table columns

Example for Database Normalization

The example below effectively displays the normalization of the database with it's explaination

Reading the data from the CSV file obtained

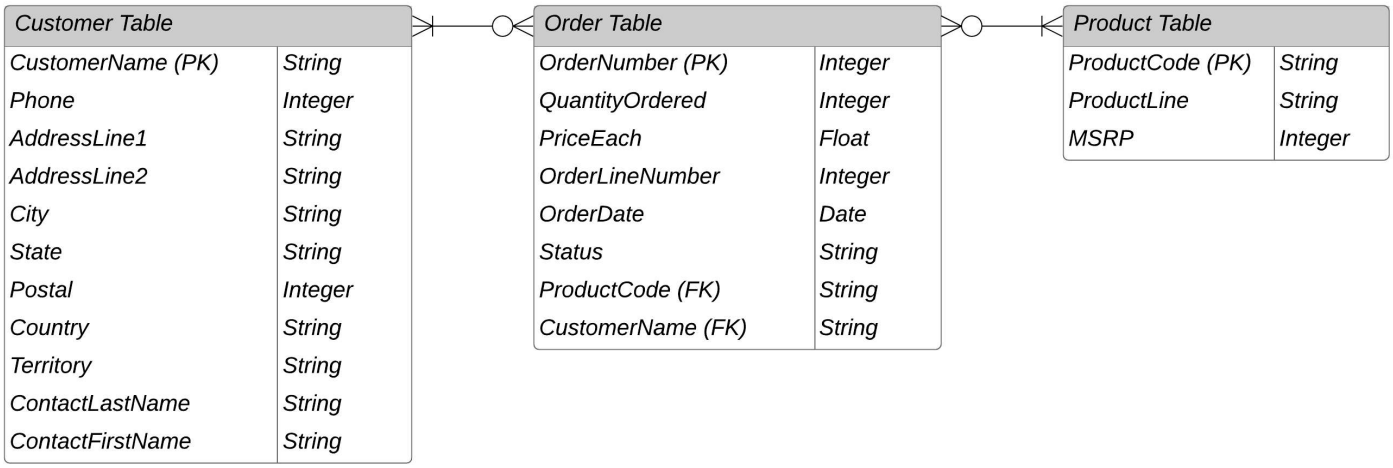
In [3]:
```
# Reading the data from the CSV file obtained

with open('sales_data_sample2.csv') as f:
    data = f.read()

data = StringIO(data)
data = pd.read_csv(data)

# Creating and viewing master table/datasheet

data
```

Out[3]:

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	SALES	ORDERDATE	S
0	10107	30	95.70	2	2871.00	2/24/2003 0:00	
1	10121	34	81.35	5	2765.90	5/7/2003 0:00	
2	10134	41	94.74	2	3884.34	7/1/2003 0:00	
3	10145	45	83.26	6	3746.70	8/25/2003 0:00	
4	10159	49	100.00	14	5205.27	10/10/2003 0:00	
...	...	...	...	...	...	...	
2818	10350	20	100.00	15	2244.40	12/2/2004 0:00	
2819	10373	29	100.00	1	3978.51	1/31/2005 0:00	
2820	10386	43	100.00	4	5417.57	3/1/2005 0:00	R
2821	10397	34	62.24	1	2116.16	3/28/2005 0:00	
2822	10414	47	65.52	9	3079.44	5/6/2005 0:00	

2823 rows × 25 columns

Entity Relationship Diagram (ERD)




```
Legend
        PK - Primary Key
        FK - Foreign Key
```

Creating Order table

Here the table is for order details which include the columns for 'ORDERNUMBER', 'QUANTITYORDERED', 'PRICEEACH', 'ORDERLINENUMBER', 'ORDERDATE', 'STATUS', 'PRODUCTCODE' 'CUSTOMERNAME'.

Here the Primary Key is 'ORDERNUMBER' which is unique to all orders. CUSTOMERNAME and PRODUCTCODE are the Foreign Keys in this table. These are representative for the other order relevant data which is redundant in the table.

In [4]:
```
order_table = data.drop(columns=['PRODUCTLINE', 'MSRP', 'PHONE', 'ADDRESSLINE1', 'ADDRESSLINE2', 'CITY',
                                 'STATE', 'POSTALCODE', 'COUNTRY', 'TERRITORY', 'CONTACTLASTNAME', 'CONTACTFIRSTNAME',
                                 'SALES', 'QTR_ID', 'MONTH_ID', 'YEAR_ID', 'DEALSIZE'
])
# order_table


# Removing duplicity within the table

order_table_final = order_table.drop_duplicates()
order_table_final
```

Out[4]:

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	ORDERDATE	STATUS
0	10107	30	95.70	2	2/24/2003 0:00	Shipped
1	10121	34	81.35	5	5/7/2003 0:00	Shipped
2	10134	41	94.74	2	7/1/2003 0:00	Shipped
3	10145	45	83.26	6	8/25/2003 0:00	Shipped
4	10159	49	100.00	14	10/10/2003 0:00	Shipped
...	...	...	...	...	...	...
2818	10350	20	100.00	15	12/2/2004 0:00	Shipped
2819	10373	29	100.00	1	1/31/2005 0:00	Shipped
2820	10386	43	100.00	4	3/1/2005 0:00	Resolved
2821	10397	34	62.24	1	3/28/2005 0:00	Shipped
2822	10414	47	65.52	9	5/6/2005 0:00	On Hold

2823 rows × 8 columns

In [5]:
```
#Auditing

order_table_final.isnull().sum()
```

Out[5]:
```
ORDERNUMBER         0
QUANTITYORDERED     0
PRICEEACH           0
ORDERLINENUMBER     0
ORDERDATE           0
STATUS              0
PRODUCTCODE         0
CUSTOMERNAME        0
dtype: int64
```

Creating Customer table

Here the table is for customer details which include the columns for 'CUSTOMERNAME', 'PHONE', 'ADDRESSLINE1', 'ADDRESSLINE2', 'CITY', 'STATE', 'POSTALCODE', 'COUNTRY', 'TERRITORY', 'CONTACTLASTNAME', 'CONTACTFIRSTNAME'.

Here the Primary Key for the table is 'CUSTOMERNAME'. No Foreign Key exists here.

In [6]:
```
customer_table = data.drop(columns=['ORDERNUMBER', 'QUANTITYORDERED', 'PRICEEACH', 'ORDERLINENUMBER',
                                    'SALES', 'ORDERDATE', 'STATUS', 'QTR_ID', 'MONTH_ID', 'YEAR_ID', 'PRODUCTCODE',
                                    'PRODUCTLINE', 'MSRP', 'DEALSIZE'])
# customer_table


# Removing duplicity within the table

customer_table_final = customer_table.drop_duplicates()


# Dropping columns as the data is biased and is empty but can lead to missing imformation

customer_table_final_edit = customer_table_final.drop(columns = ['ADDRESSLINE2', 'STATE', 'TERRITORY'])


customer_table_final_edit
```

Out[6]:

	CUSTOMERNAME	PHONE	ADDRESSLINE1	CITY	POSTALCODE	COUNTRY	CONTACTLA
0	Land of Toys Inc.	2125557818	897 Long Airport Avenue	NYC	10022	USA	
1	Reims Collectables	26.47.1555	59 rue de l'Abbaye	Reims	51100	France	
2	Lyon Souveniers	+33 1 46 62 7555	27 rue du Colonel Pierre Avia	Paris	75508	France	
3	Toys4GrownUps.com	6265557265	78934 Hillside Dr.	Pasadena	90003	USA	
4	Corporate Gift Ideas Co.	6505551386	7734 Strong St.	San Francisco	NaN	USA	
...	...	...	...	...	...	...	
483	Australian Collectables, Ltd	61-9-3844-6555	7 Allen Street	Glen Waverly	3150	Australia	
554	Gift Ideas Corp.	2035554407	2440 Pompton St.	Glendale	97561	USA	
567	Bavarian Collectables Imports, Co.	+49 89 61 08 9555	Hansastr. 15	Munich	80686	Germany	Do
571	Royale Belge	(071) 23 67 2555	Boulevard Tirou, 255	Charleroi	B-6000	Belgium	
937	Auto-Moto Classics Inc.	6175558428	16780 Pompton St.	Brickhaven	58339	USA	

92 rows × 8 columns

In [7]:
```
#Auditing

customer_table_final_edit.isnull().sum()
```

Out[7]:
```
CUSTOMERNAME        0
PHONE               0
ADDRESSLINE1        0
CITY                0
POSTALCODE          3
COUNTRY             0
CONTACTLASTNAME     0
CONTACTFIRSTNAME    0
dtype: int64
```

Creating Product table

Here the table is for product details which include the columns for 'PRODUCTCODE', 'PRODUCTLINE', 'MSRP'.

Here, 'PRODUCTCODE' acts as the table's Primary Key. No Foreign Key is present or necessary in the table.

In [8]:
```
product_table = data.drop(columns=['ORDERNUMBER', 'QUANTITYORDERED', 'PRICEEACH', 'ORDERLINENUMBER',
                                   'SALES', 'ORDERDATE', 'STATUS', 'QTR_ID', 'MONTH_ID', 'YEAR_ID', 'CUSTOMERNAME',
                                   'PHONE', 'ADDRESSLINE1', 'ADDRESSLINE2', 'CITY', 'STATE', 'POSTALCODE', 'COUNTRY',
                                   'TERRITORY', 'CONTACTLASTNAME', 'CONTACTFIRSTNAME', 'DEALSIZE'])
# product_table


# Removing duplicity within the table

product_table_final = product_table.drop_duplicates()
product_table_final
```

Out[8]:

	PRODUCTLINE	MSRP	PRODUCTCODE
0	Motorcycles	95	S10_1678
26	Classic Cars	214	S10_1949
54	Motorcycles	118	S10_2016
80	Motorcycles	193	S10_4698
106	Classic Cars	136	S10_4757
...	...	...	...
2691	Ships	100	S700_3505
2717	Ships	99	S700_3962
2743	Planes	74	S700_4002
2770	Planes	49	S72_1253
2797	Ships	54	S72_3212

109 rows × 3 columns

In [9]:
```
#Auditing

product_table_final.isnull().sum()
```

Out[9]:
```
PRODUCTLINE    0
MSRP           0
PRODUCTCODE    0
dtype: int64
```

Hands-on Experience with SQL

Connecting with the Database

In [10]:
```
# Connecting to the database

# Please change password here in the field 'passwd'

connection = mysql.connector.connect(
    host="localhost",
    user="root",
    passwd="password"
)

print(connection)
```

```
<mysql.connector.connection.MySQLConnection object at 0x00000232E01BD508>
```

Setting up the Cursor

In [11]:
```
# cursor
crsr = connection.cursor()
```

SQL commmands

In [12]:
```
# Listing all databases
crsr.execute("SHOW DATABASES")

for x in crsr:
    print(x)
```

```
('information_schema',)
('mysql',)
('performance_schema',)
('poe_dmdd',)
('sakila',)
('sales_dmdd',)
('sys',)
('testdb00',)
('testdb02',)
('world',)
```

CREATE function

Creating a database

In [13]:
```
# Creating the database testDb01
sql = '''
CREATE DATABASE testDb01;
'''

# Executing the SQL command
crsr.execute(sql)
```

Using the database

In [14]:
```
# Using the database testDb01
sql = '''
USE testDb01;
'''

# Executing the SQL command
crsr.execute(sql)
```

Creating a table with SCHEMA as

```
Agent_Code CHAR as PRIMARY KEY,
Agent_Name CHAR,
Working_Area CHAR,
Commission DECIMAL,
Phone_No CHAR,
Country VARCHAR
```

In [15]:
```
# Creating table in the database
sql = '''
CREATE TABLE Agent
(
Agent_Code CHAR(6) NOT NULL PRIMARY KEY,
Agent_Name CHAR(40),
Working_Area CHAR(35),
Commission DECIMAL(10,2),
Phone_No CHAR(15),
Country VARCHAR(25)
);
'''

# Executing the SQL command
crsr.execute(sql)
```

Saving the changes made

In [16]:
```
# To save the changes in the files. Never skip this.
# If we skip this, nothing will be saved in the database.
connection.commit()
```

INSERT function

Inserting data into the table created

In [17]:
```
# Inserting a singular data into the table AGENT
sql = '''
INSERT INTO agent VALUES ('A107', 'Kalvin', 'Boston', '0.17', '076-87346523', 'USA');
'''

# Executing the SQL command
crsr.execute(sql)
```

In [18]:
```
# To save the changes in the files. Never skip this.
# If we skip this, nothing will be saved in the database.
connection.commit()
```

In [19]:
```
# Inserting a multiple data into the table AGENT
sql = '''
INSERT INTO agent VALUES
('A103', 'Liza ', 'London', '0.12', '075-78439832', 'UK'),
('A108', 'Alfred', 'New York', '0.14', '044-67547834', 'USA'),
('A111', 'Kate', 'Hydrabad', '0.17', '077-34556434', 'IND'),
('A110', 'Kim', 'Delhi', '0.15', '007-98453765', 'IND'),
('A112', 'Kyle', 'San Diego', '0.11', '044-21905478', 'USA'),
('A105', 'Andrew', 'Brisban', '0.12', '045-12093487', 'CA'),
('A101', 'Kristine', 'Tokyo', '0.13', '077-34874567', 'JP'),
('A102', 'Suresh', 'Hydrabad', '0.19', '029-98765432', 'IND'),
('A106', 'McMiller', 'London', '0.14', '078-09543873', 'UK'),
('A104', 'Ivanka', 'London', '0.16', '008-22598437', 'CA'),
('A109', 'Ramesh', 'Hampshair', '0.15', '008-22576436', 'UK');
'''

# Executing the SQL command
crsr.execute(sql)
```

In [20]:
```
# To save the changes in the files. Never skip this.
# If we skip this, nothing will be saved in the database.
connection.commit()
```

Displaying the changes (table)

In [21]:
```
# Accessing the table from the database
sql = '''
SELECT *
FROM agent;
'''

# Executing the SQL command
crsr.execute(sql)

# Fetching results
result = crsr.fetchall()

final_result = pd.DataFrame(result)
headers = [i[0] for i in crsr.description]
final_result.columns = headers
final_result
```

Out[21]:

	Agent_Code	Agent_Name	Working_Area	Commission	Phone_No	Country
0	A101	Kristine	Tokyo	0.13	077-34874567	JP
1	A102	Suresh	Hydrabad	0.19	029-98765432	IND
2	A103	Liza	London	0.12	075-78439832	UK
3	A104	Ivanka	London	0.16	008-22598437	CA
4	A105	Andrew	Brisban	0.12	045-12093487	CA
5	A106	McMiller	London	0.14	078-09543873	UK
6	A107	Kalvin	Boston	0.17	076-87346523	USA
7	A108	Alfred	New York	0.14	044-67547834	USA
8	A109	Ramesh	Hampshair	0.15	008-22576436	UK
9	A110	Kim	Delhi	0.15	007-98453765	IND
10	A111	Kate	Hydrabad	0.17	077-34556434	IND
11	A112	Kyle	San Diego	0.11	044-21905478	USA

UPDATE function

Updating a record present in the table

In [22]:
```
# Updating a record present in the table AGENTS
sql = '''
UPDATE agent
SET commission = 0.17
WHERE agent_code = 'A101'
'''

# Executing the SQL command
crsr.execute(sql)
```

Displaying the changes (table)

In [23]:
```
# Accessing the table from the database
sql = '''
SELECT *
FROM agent;
'''

# Executing the SQL command
crsr.execute(sql)

# Fetching results
result = crsr.fetchall()

final_result = pd.DataFrame(result)
headers = [i[0] for i in crsr.description]
final_result.columns = headers
final_result
```

Out[23]:

	Agent_Code	Agent_Name	Working_Area	Commission	Phone_No	Country
0	A101	Kristine	Tokyo	0.17	077-34874567	JP
1	A102	Suresh	Hydrabad	0.19	029-98765432	IND
2	A103	Liza	London	0.12	075-78439832	UK
3	A104	Ivanka	London	0.16	008-22598437	CA
4	A105	Andrew	Brisban	0.12	045-12093487	CA
5	A106	McMiller	London	0.14	078-09543873	UK
6	A107	Kalvin	Boston	0.17	076-87346523	USA
7	A108	Alfred	New York	0.14	044-67547834	USA
8	A109	Ramesh	Hampshair	0.15	008-22576436	UK
9	A110	Kim	Delhi	0.15	007-98453765	IND
10	A111	Kate	Hydrabad	0.17	077-34556434	IND
11	A112	Kyle	San Diego	0.11	044-21905478	USA

In [24]:
```
# To save the changes in the files. Never skip this.
# If we skip this, nothing will be saved in the database.
connection.commit()
```

ALTER function

Changing table structure

In [25]:
```
# Altering the table structure of the table AGENTS
sql = '''
ALTER TABLE agent
ADD Salary INT
'''

# Executing the SQL command
crsr.execute(sql)
```

Displaying the changes (table)

In [26]:
```
# Accessing the table from the database
sql = '''
SELECT *
FROM agent;
'''

# Executing the SQL command
crsr.execute(sql)

# Fetching results
result = crsr.fetchall()

final_result = pd.DataFrame(result)
headers = [i[0] for i in crsr.description]
final_result.columns = headers
final_result
```

Out[26]:

	Agent_Code	Agent_Name	Working_Area	Commission	Phone_No	Country	Salary
0	A101	Kristine	Tokyo	0.17	077-34874567	JP	None
1	A102	Suresh	Hydrabad	0.19	029-98765432	IND	None
2	A103	Liza	London	0.12	075-78439832	UK	None
3	A104	Ivanka	London	0.16	008-22598437	CA	None
4	A105	Andrew	Brisban	0.12	045-12093487	CA	None
5	A106	McMiller	London	0.14	078-09543873	UK	None
6	A107	Kalvin	Boston	0.17	076-87346523	USA	None
7	A108	Alfred	New York	0.14	044-67547834	USA	None
8	A109	Ramesh	Hampshair	0.15	008-22576436	UK	None
9	A110	Kim	Delhi	0.15	007-98453765	IND	None
10	A111	Kate	Hydrabad	0.17	077-34556434	IND	None
11	A112	Kyle	San Diego	0.11	044-21905478	USA	None

In [27]:
```
# Updating salary records present in the table AGENTS
sql = '''
UPDATE agent
SET salary = 70000
WHERE country = 'JP'
'''

# Executing the SQL command
crsr.execute(sql)
```

Displaying the changes (table)

In [28]:
```
# Accessing the table from the database
sql = '''
SELECT *
FROM agent;
'''

# Executing the SQL command
crsr.execute(sql)

# Fetching results
result = crsr.fetchall()

final_result = pd.DataFrame(result)
headers = [i[0] for i in crsr.description]
final_result.columns = headers
final_result
```

Out[28]:

	Agent_Code	Agent_Name	Working_Area	Commission	Phone_No	Country	Salary
0	A101	Kristine	Tokyo	0.17	077-34874567	JP	70000.0
1	A102	Suresh	Hydrabad	0.19	029-98765432	IND	NaN
2	A103	Liza	London	0.12	075-78439832	UK	NaN
3	A104	Ivanka	London	0.16	008-22598437	CA	NaN
4	A105	Andrew	Brisban	0.12	045-12093487	CA	NaN
5	A106	McMiller	London	0.14	078-09543873	UK	NaN
6	A107	Kalvin	Boston	0.17	076-87346523	USA	NaN
7	A108	Alfred	New York	0.14	044-67547834	USA	NaN
8	A109	Ramesh	Hampshair	0.15	008-22576436	UK	NaN
9	A110	Kim	Delhi	0.15	007-98453765	IND	NaN
10	A111	Kate	Hydrabad	0.17	077-34556434	IND	NaN
11	A112	Kyle	San Diego	0.11	044-21905478	USA	NaN

In [29]:
```
# Updating salary records present in the table AGENTS
sql = '''
UPDATE agent
SET salary = 60000
WHERE country = 'IND'
'''

# Executing the SQL command
crsr.execute(sql)
```

In [30]:
```
# Updating salary records present in the table AGENTS
sql = '''
UPDATE agent
SET salary = 65000
WHERE country = 'USA'
'''

# Executing the SQL command
crsr.execute(sql)
```

In [31]:
```
# Updating salary records present in the table AGENTS
sql = '''
UPDATE agent
SET salary = 75000
WHERE country = 'UK'
'''

# Executing the SQL command
crsr.execute(sql)
```

In [32]:
```
# Updating salary records present in the table AGENTS
sql = '''
UPDATE agent
SET salary = 69000
WHERE country = 'CA'
'''

# Executing the SQL command
crsr.execute(sql)
```

Displaying the changes (table)

In [33]:
```
# Accessing the table from the database
sql = '''
SELECT *
FROM agent;
'''

# Executing the SQL command
crsr.execute(sql)

# Fetching results
result = crsr.fetchall()

final_result = pd.DataFrame(result)
headers = [i[0] for i in crsr.description]
final_result.columns = headers
final_result
```

Out[33]:

	Agent_Code	Agent_Name	Working_Area	Commission	Phone_No	Country	Salary
0	A101	Kristine	Tokyo	0.17	077-34874567	JP	70000
1	A102	Suresh	Hydrabad	0.19	029-98765432	IND	60000
2	A103	Liza	London	0.12	075-78439832	UK	75000
3	A104	Ivanka	London	0.16	008-22598437	CA	69000
4	A105	Andrew	Brisban	0.12	045-12093487	CA	69000
5	A106	McMiller	London	0.14	078-09543873	UK	75000
6	A107	Kalvin	Boston	0.17	076-87346523	USA	65000
7	A108	Alfred	New York	0.14	044-67547834	USA	65000
8	A109	Ramesh	Hampshair	0.15	008-22576436	UK	75000
9	A110	Kim	Delhi	0.15	007-98453765	IND	60000
10	A111	Kate	Hydrabad	0.17	077-34556434	IND	60000
11	A112	Kyle	San Diego	0.11	044-21905478	USA	65000

Renaming the table name

In [34]:
```
# Altering (Renaming) the table structure of the table AGENTS
sql = '''
ALTER TABLE Agent RENAME TO Agents
'''

# Executing the SQL command
crsr.execute(sql)
```

Displaying the changes (table)

In [35]:
```
# Accessing the table from the database
sql = '''
SELECT *
FROM agents;
'''

# Executing the SQL command
crsr.execute(sql)

# Fetching results
result = crsr.fetchall()

final_result = pd.DataFrame(result)
headers = [i[0] for i in crsr.description]
final_result.columns = headers
final_result
```

Out[35]:

	Agent_Code	Agent_Name	Working_Area	Commission	Phone_No	Country	Salary
0	A101	Kristine	Tokyo	0.17	077-34874567	JP	70000
1	A102	Suresh	Hydrabad	0.19	029-98765432	IND	60000
2	A103	Liza	London	0.12	075-78439832	UK	75000
3	A104	Ivanka	London	0.16	008-22598437	CA	69000
4	A105	Andrew	Brisban	0.12	045-12093487	CA	69000
5	A106	McMiller	London	0.14	078-09543873	UK	75000
6	A107	Kalvin	Boston	0.17	076-87346523	USA	65000
7	A108	Alfred	New York	0.14	044-67547834	USA	65000
8	A109	Ramesh	Hampshair	0.15	008-22576436	UK	75000
9	A110	Kim	Delhi	0.15	007-98453765	IND	60000
10	A111	Kate	Hydrabad	0.17	077-34556434	IND	60000
11	A112	Kyle	San Diego	0.11	044-21905478	USA	65000

In [36]:
```
# To save the changes in the files. Never skip this.
# If we skip this, nothing will be saved in the database.
connection.commit()
```

INDEX function

Indexing the tabluar data for faster processing and search results

In [37]:
```
# Creating an UNIQUE INDEX column for the table structure of the table AGENTS_DETAILS
sql = '''
CREATE UNIQUE INDEX Agent_Initial
ON agents(agent_name);
'''

# Executing the SQL command
crsr.execute(sql)
```

Dropping the created Index if unnecessary

In [38]:
```
# Dropping the created UNIQUE INDEX column for the table structure of the table AGENT
S_DETAILS
sql = '''
ALTER TABLE agents
DROP INDEX Agent_Initial;
'''

# Executing the SQL command
crsr.execute(sql)
```

In [39]:
```
# To save the changes in the files. Never skip this.
# If we skip this, nothing will be saved in the database.
connection.commit()
```

DELETE function

Creating a Save point to come back or revert back.

In [40]:
```
# Starting a Transaction to revert back when ROLLBACK needed
sql = '''
START TRANSACTION;
'''

# Executing the SQL command
crsr.execute(sql)
```

Deleting a data entry from the table

In [41]:
```
# Deleting agent with agent_code = A102
sql = '''
DELETE
FROM agents
WHERE agent_code = 'A102';
'''

# Executing the SQL command
crsr.execute(sql)
```

Displaying the changes (table)

In [42]:
```
# Accessing the table from the database
sql = '''
SELECT *
FROM agents;
'''

# Executing the SQL command
crsr.execute(sql)

# Fetching results
result = crsr.fetchall()

final_result = pd.DataFrame(result)
headers = [i[0] for i in crsr.description]
final_result.columns = headers
final_result
```

Out[42]:

	Agent_Code	Agent_Name	Working_Area	Commission	Phone_No	Country	Salary
0	A101	Kristine	Tokyo	0.17	077-34874567	JP	70000
1	A103	Liza	London	0.12	075-78439832	UK	75000
2	A104	Ivanka	London	0.16	008-22598437	CA	69000
3	A105	Andrew	Brisban	0.12	045-12093487	CA	69000
4	A106	McMiller	London	0.14	078-09543873	UK	75000
5	A107	Kalvin	Boston	0.17	076-87346523	USA	65000
6	A108	Alfred	New York	0.14	044-67547834	USA	65000
7	A109	Ramesh	Hampshair	0.15	008-22576436	UK	75000
8	A110	Kim	Delhi	0.15	007-98453765	IND	60000
9	A111	Kate	Hydrabad	0.17	077-34556434	IND	60000
10	A112	Kyle	San Diego	0.11	044-21905478	USA	65000

ROLLBACK function

This function works only when a transaction has been started or a save point has been created.

This will not undo any TRUNCATE function as no backup remains after TRUNCATE function is executed.

The Syntax of starting a transaction is mentioned in the comments below.

In [43]:
```
"""
# First set a statement of BEGIN TRANSACTION

sql = '''
BEGIN TRANSACTION
-- Any work or clauses here
'''

# Executing the SQL command
crsr.execute(sql)

"""



# THEN ROLLBACK to the state before BEGIN TRANSACTION

sql = '''
ROLLBACK;
'''

# Executing the SQL command
crsr.execute(sql)
```

Displaying the rollback changes (table)

In [44]:
```
# Accessing the table from the database
sql = '''
SELECT *
FROM agents;
'''

# Executing the SQL command
crsr.execute(sql)

# Fetching results
result = crsr.fetchall()

final_result = pd.DataFrame(result)
headers = [i[0] for i in crsr.description]
final_result.columns = headers
final_result
```

Out[44]:

	Agent_Code	Agent_Name	Working_Area	Commission	Phone_No	Country	Salary
0	A101	Kristine	Tokyo	0.17	077-34874567	JP	70000
1	A102	Suresh	Hydrabad	0.19	029-98765432	IND	60000
2	A103	Liza	London	0.12	075-78439832	UK	75000
3	A104	Ivanka	London	0.16	008-22598437	CA	69000
4	A105	Andrew	Brisban	0.12	045-12093487	CA	69000
5	A106	McMiller	London	0.14	078-09543873	UK	75000
6	A107	Kalvin	Boston	0.17	076-87346523	USA	65000
7	A108	Alfred	New York	0.14	044-67547834	USA	65000
8	A109	Ramesh	Hampshair	0.15	008-22576436	UK	75000
9	A110	Kim	Delhi	0.15	007-98453765	IND	60000
10	A111	Kate	Hydrabad	0.17	077-34556434	IND	60000
11	A112	Kyle	San Diego	0.11	044-21905478	USA	65000

In [45]:
```
# To save the changes in the files. Never skip this.
# If we skip this, nothing will be saved in the database.
connection.commit()
```

TRUNCATE function

The lost data cannot be recovered back after this funciton is performed

In [46]:
```
# Erasing the entire table

sql = '''
TRUNCATE TABLE Agents;
'''

# Executing the SQL command
crsr.execute(sql)
```

In [47]:
```
# Listing all tables
crsr.execute("SHOW TABLES")

for x in crsr:
    print(x)
```

```
('agents',)
```

Displaying the changes (table)

In [48]:
```
# Accessing the table from the database
sql = '''
SELECT *
FROM agents;
'''

# Executing the SQL command
crsr.execute(sql)

# Fetching results
result = crsr.fetchall()

final_result = pd.DataFrame(result)

final_result
```

Out[48]:

DROP function

Dropping the table Agents

In [49]:
```
# Dropping the table AGENTS_DETAILS
sql = '''
DROP TABLE agents;
'''

# Executing the SQL command
crsr.execute(sql)
```

Listing the tables present in the database. This would give no results as there are no tables present in the database.

In [50]:
```
# Listing all tables
crsr.execute("SHOW TABLES")

for x in crsr:
    print(x)
```

Dropping the database testDb01

In [51]:
```
# Dropping the database testDb01
sql = '''
DROP DATABASE testDb01;
'''

# Executing the SQL command
crsr.execute(sql)
```

Listing all databases

In [52]:
```
# Listing all databases
crsr.execute("SHOW DATABASES")

for x in crsr:
    print(x)
```

```
('information_schema',)
('mysql',)
('performance_schema',)
('poe_dmdd',)
('sakila',)
('sales_dmdd',)
('sys',)
('testdb00',)
('testdb02',)
('world',)
```

In [53]:
```
# To save the changes in the files. Never skip this.
# If we skip this, nothing will be saved in the database.
connection.commit()
```

Understanding SQL functions and syntaxes hereafter.

In [54]:
```
# using the before mentioned data of Sales database for learning purpose, which is st
ored in the database
# "sales_dmdd" and establishing that connection and finally displaying the entire tab
le for reference

# Please change password here in the field 'passwd'

connection = mysql.connector.connect(
    host="localhost",
    user="root",
    passwd="password",
    database="sales_dmdd"
)

# Setting up the cursor
crsr = connection.cursor()
```

In [55]:
```
# Accessing the table from the database
sql = '''
SELECT *
FROM ordertable;
'''

# Executing the SQL command
crsr.execute(sql)

# Fetching results
result = crsr.fetchall()

final_result = pd.DataFrame(result)
headers = [i[0] for i in crsr.description]
final_result.columns = headers
final_result
```

Out[55]:

	ORDERNUMBER	QUANTITYORDERED	PRICEEACH	ORDERLINENUMBER	ORDERDATE	STATUS
0	10107	30	95.70	2	2003-02-24	Shipped
1	10121	34	81.35	5	2003-05-07	Shipped
2	10134	41	94.74	2	2003-07-01	Shipped
3	10145	45	83.26	6	2003-08-25	Shipped
4	10159	49	100.00	14	2003-10-10	Shipped
...	...	...	...	...	...	...
2818	10350	20	100.00	15	2004-12-02	Shipped
2819	10373	29	100.00	1	2005-01-31	Shipped
2820	10386	43	100.00	4	2005-03-01	Resolved
2821	10397	34	62.24	1	2005-03-28	Shipped
2822	10414	47	65.52	9	2005-05-06	On Hold

2823 rows × 8 columns

Select Functions

Creating a SELECT SQL command

In [56]:
```
# Creating a SELECT SQL command
sql = '''
SELECT ordernumber, quantityordered, priceeach
FROM ordertable;
'''

# Executing the SQL command
crsr.execute(sql)

# Fetching results
result = crsr.fetchall()

final_result = pd.DataFrame(result)
headers = [i[0] for i in crsr.description]
final_result.columns = headers
final_result
```

Out[56]:

	ordernumber	quantityordered	priceeach
0	10107	30	95.70
1	10121	34	81.35
2	10134	41	94.74
3	10145	45	83.26
4	10159	49	100.00
...	...	...	...
2818	10350	20	100.00
2819	10373	29	100.00
2820	10386	43	100.00
2821	10397	34	62.24
2822	10414	47	65.52

2823 rows × 3 columns

Creating a SELECT SQL command with WHERE condition and Comparitive operators

In [57]:
```
# Creating a SELECT SQL command with WHERE condition and Comparitive operators
sql = '''
SELECT ordernumber, quantityordered, priceeach
FROM ordertable
WHERE quantityordered < 35;
'''

# Executing the SQL command
crsr.execute(sql)

# Fetching results
result = crsr.fetchall()

final_result = pd.DataFrame(result)
headers = [i[0] for i in crsr.description]
final_result.columns = headers
final_result
```

Out[57]:

	ordernumber	quantityordered	priceeach
0	10107	30	95.70
1	10121	34	81.35
2	10180	29	86.13
3	10201	22	98.57
4	10237	23	100.00
...	...	...	...
1399	10283	33	51.32
1400	10293	32	60.06
1401	10350	20	100.00
1402	10373	29	100.00
1403	10397	34	62.24

1404 rows × 3 columns

Creating a SELECT SQL command with WHERE and LIKE condition

In [58]:
```
# Creating a SELECT SQL command with WHERE and LIKE condition
sql = '''
SELECT ordernumber, quantityordered, priceeach, customername
FROM ordertable
WHERE customername LIKE 'T%';
'''

# Executing the SQL command
crsr.execute(sql)

# Fetching results
result = crsr.fetchall()

final_result = pd.DataFrame(result)
headers = [i[0] for i in crsr.description]
final_result.columns = headers
final_result
```

Out[58]:

	ordernumber	quantityordered	priceeach	customername
0	10145	45	83.26	Toys4GrownUps.com
1	10168	36	96.66	Technics Stores Inc.
2	10251	28	100.00	Tekni Collectables Inc.
3	10299	23	100.00	Toys of Finland, Co.
4	10140	37	100.00	Technics Stores Inc.
...	...	...	...	...
208	10339	50	57.86	Tokyo Collectables, Ltd
209	10401	28	72.55	Tekni Collectables Inc.
210	10155	34	49.16	Toys of Finland, Co.
211	10339	27	76.31	Tokyo Collectables, Ltd
212	10400	20	56.12	The Sharp Gifts Warehouse

213 rows × 4 columns

Creating a SELECT SQL command with WHERE, GROUP BY condition and Comparitive operators

In [59]:
```
# Creating a SELECT SQL command with WHERE, GROUP BY condition and Comparitive operat
ors
sql = '''
SELECT productcode, priceeach
FROM ordertable
WHERE priceeach > 50
GROUP BY productcode;
'''

# Executing the SQL command
crsr.execute(sql)

# Fetching results
result = crsr.fetchall()

final_result = pd.DataFrame(result)
headers = [i[0] for i in crsr.description]
final_result.columns = headers
final_result
```

Out[59]:

	productcode	priceeach
0	S10_1678	95.70
1	S10_1949	100.00
2	S10_2016	99.91
3	S10_4698	100.00
4	S10_4757	100.00
...	...	...
104	S700_3505	81.14
105	S700_3962	100.00
106	S700_4002	61.44
107	S72_1253	52.64
108	S72_3212	56.78

109 rows × 2 columns

Creating a SELECT SQL command with DISTINCT filter

In [60]:
```
# Creating a SELECT SQL command with DISTINCT filter
sql = '''
SELECT DISTINCT productcode
FROM ordertable
WHERE priceeach = 100;
'''

# Executing the SQL command
crsr.execute(sql)

# Fetching results
result = crsr.fetchall()

final_result = pd.DataFrame(result)
headers = [i[0] for i in crsr.description]
final_result.columns = headers
final_result
```

Out[60]:

	productcode
0	S10_1678
1	S10_1949
2	S10_2016
3	S10_4698
4	S10_4757
...	...
102	S700_3167
103	S700_3505
104	S700_3962
105	S700_4002
106	S72_3212

107 rows × 1 columns

Creating a SELECT SQL command with COUNT(*) function

In [61]:
```
# Creating a SELECT SQL command with COUNT(*) function
sql = '''
SELECT COUNT(*)
FROM ordertable;
'''

# Executing the SQL command
crsr.execute(sql)

# Fetching results
result = crsr.fetchall()

final_result = pd.DataFrame(result)
headers = [i[0] for i in crsr.description]
final_result.columns = headers
final_result
```

Out[61]:

	COUNT(*)
0	2823

Creating a SELECT SQL command with COUNT and GROUP BY condition

In [62]:
```
# Creating a SELECT SQL command  with COUNT and GROUP BY condition
sql = '''
SELECT productcode, COUNT(*)
FROM ordertable
GROUP BY productcode;
'''

# Executing the SQL command
crsr.execute(sql)

# Fetching results
result = crsr.fetchall()

final_result = pd.DataFrame(result)
headers = [i[0] for i in crsr.description]
final_result.columns = headers
final_result
```

Out[62]:

	productcode	COUNT(*)
0	S10_1678	26
1	S10_1949	28
2	S10_2016	26
3	S10_4698	26
4	S10_4757	27
...	...	...
104	S700_3505	26
105	S700_3962	26
106	S700_4002	27
107	S72_1253	27
108	S72_3212	26

109 rows × 2 columns

Creating a SELECT SQL command with GROUP BY and ORDER BY condition

In [63]:
```
# Creating a SELECT SQL command with GROUP BY and ORDER BY condition
sql = '''
SELECT customername, productcode
FROM ordertable
GROUP BY customername
ORDER BY customername ASC, productcode DESC;
'''

# Executing the SQL command
crsr.execute(sql)

# Fetching results
result = crsr.fetchall()

final_result = pd.DataFrame(result)
headers = [i[0] for i in crsr.description]
final_result.columns = headers
final_result
```

Out[63]:

	customername	productcode
0	Alpha Cognac	S10_4757
1	Amica Models & Co.	S10_1949
2	Anna's Decorations, Ltd	S10_1949
3	Atelier graphique	S10_2016
4	Australian Collectables, Ltd	S18_1342
...	...	...
87	UK Collectables, Ltd.	S10_1678
88	Vida Sport, Ltd	S12_1099
89	Vitachrome Inc.	S10_1678
90	Volvo Model Replicas, Co	S10_1949
91	West Coast Collectables Co.	S10_1949

92 rows × 2 columns

Creating a SELECT SQL command with Aggregate functions and GROUP BY, ORDER BY conditions

In [64]:
```
# Creating a SELECT SQL command with Aggregate functions and GROUP BY, ORDER BY condi
tions
sql = '''
SELECT customername, AVG(quantityordered), MIN(priceeach), SUM(quantityordered)
FROM ordertable
GROUP BY customername
ORDER BY MAX(quantityordered) DESC;
'''

# Executing the SQL command
crsr.execute(sql)

# Fetching results
result = crsr.fetchall()

final_result = pd.DataFrame(result)
headers = [i[0] for i in crsr.description]
final_result.columns = headers
final_result
```

Out[64]:

	customername	AVG(quantityordered)	MIN(priceeach)	SUM(quantityordered)
0	Mini Caravy	41.0000	40.25	779
1	Tekni Collectables Inc.	43.1429	35.35	906
2	The Sharp Gifts Warehouse	41.4000	40.25	1656
3	Euro Shopping Channel	36.0116	32.99	9327
4	Salzburg Collectables	36.0500	42.67	1442
...	...	...	...	...
87	Auto Assoc. & Cie.	35.3889	30.20	637
88	Microscale Inc.	38.1000	36.93	381
89	Cambridge Collectables Co.	32.4545	57.61	357
90	Double Decker Gift Stores, Ltd	29.7500	52.14	357
91	Boards & Toys Co.	34.0000	64.33	102

92 rows × 4 columns

Creating a SELECT SQL command with IN clause

In [65]:
```
# Creating a SELECT SQL command with IN clause
sql = '''
SELECT *
FROM producttable
WHERE msrp IN (35,50)
ORDER BY code;
'''

# Executing the SQL command
crsr.execute(sql)

# Fetching results
result = crsr.fetchall()

final_result = pd.DataFrame(result)
headers = [i[0] for i in crsr.description]
final_result.columns = headers
final_result
```

Out[65]:

	Code	Line	MSRP
0	S18_4668	Vintage Cars	50
1	S24_1628	Classic Cars	50
2	S24_2840	Classic Cars	35

Creating a SELECT SQL command with BETWEEN clause

In [66]:
```
# Creating a SELECT SQL command with BETWEEN clause
sql = '''
SELECT *
FROM producttable
WHERE msrp BETWEEN 35 AND 50
ORDER BY code;
'''

# Executing the SQL command
crsr.execute(sql)

# Fetching results
result = crsr.fetchall()

final_result = pd.DataFrame(result)
headers = [i[0] for i in crsr.description]
final_result.columns = headers
final_result
```

Out[66]:

	Code	Line	MSRP
0	S18_4668	Vintage Cars	50
1	S24_1628	Classic Cars	50
2	S24_2022	Vintage Cars	44
3	S24_2840	Classic Cars	35
4	S24_2972	Classic Cars	37
5	S24_3969	Vintage Cars	41
6	S32_2206	Motorcycles	40
7	S50_1341	Vintage Cars	43
8	S72_1253	Planes	49

Creating a SELECT SQL command with HAVING clause

In [67]:
```
# Creating a SELECT SQL command with HAVING clause
sql = '''
SELECT customername, SUM(quantityordered) AS TotalQuantityOrdered
FROM ordertable
GROUP BY customername
HAVING TotalQuantityOrdered > 700
ORDER BY customername;
'''

# Executing the SQL command
crsr.execute(sql)

# Fetching results
result = crsr.fetchall()

final_result = pd.DataFrame(result)
headers = [i[0] for i in crsr.description]
final_result.columns = headers
final_result
```

Out[67]:

	customername	TotalQuantityOrdered
0	Amica Models & Co.	843
1	Anna's Decorations, Ltd	1469
2	Australian Collectables, Ltd	705
3	Australian Collectors, Co.	1926
4	Auto Canal Petit	1001
...	...	...
59	Toys of Finland, Co.	1051
60	Toys4GrownUps.com	1060
61	UK Collectables, Ltd.	1046
62	Vida Sport, Ltd	1078
63	Vitachrome Inc.	787

64 rows × 2 columns

Creating a SELECT SQL command with LIMIT clause

In [68]:
```
# Creating a SELECT SQL command with LIMIT clause
sql = '''
SELECT customername, SUM(quantityordered) AS TotalQuantityOrdered
FROM ordertable
GROUP BY customername
HAVING TotalQuantityOrdered > 700
ORDER BY TotalQuantityOrdered DESC
LIMIT 5;
'''

# Executing the SQL command
crsr.execute(sql)

# Fetching results
result = crsr.fetchall()

final_result = pd.DataFrame(result)
headers = [i[0] for i in crsr.description]
final_result.columns = headers
final_result
```

Out[68]:

	customername	TotalQuantityOrdered
0	Euro Shopping Channel	9327
1	Mini Gifts Distributors Ltd.	6366
2	Australian Collectors, Co.	1926
3	La Rochelle Gifts	1832
4	AV Stores, Co.	1778

Understanding JOINS

SQL JOIN clause helps combine records from two or multiple tables existing in the databas e. A JOIN would mean a combination of fields from two or more tables having a common value between them. Usually the common value is a Primary/Foreign Key as they are unique and hel ps match data with other table in a better and non conflicting way.

There are different type of JOINS present in SQL :

- INNER JOIN : This would return rows when there is a match fpund in both the tables s elected

- LEFT JOIN : This would return all rows from the left table, even if there are no mat ches found in the right table

- RIGHT JOIN : This would return all rows from the right table, even if there are no m atches found in the left table

- FULL JOIN : This would return rows when there is a match found in either of the tabl es selected

- SELF JOIN : This is employed to join a table with itself as if the table were two di fferent tables, temporarily renaming at least either one of the table in the SQL statement

- CARTESIAN JOIN : This would return the Cartesian Product of the sets of records from the tables selected in the join.

Joining the ORDERTABLE and CUSTOMERTABLE

In [69]:
```
# Joining the ORDERTABLE and CUSTOMERTABLE

sql = '''
SELECT ordernumber, contactfirstname, phone, ordertable.customername
FROM ordertable
RIGHT JOIN customertable ON ordertable.customername = customertable.customername
GROUP BY ordernumber
ORDER BY ordernumber ASC;
'''

# Executing the SQL command
crsr.execute(sql)

# Fetching results
result = crsr.fetchall()

final_result = pd.DataFrame(result)
headers = [i[0] for i in crsr.description]
final_result.columns = headers
final_result
```

Out[69]:

	ordernumber	contactfirstname	phone	customername
0	10100	Valarie	6035558647	Online Diecast Creations Co.
1	10101	Roland	+49 69 66 90 2555	Blauer See Auto, Co.
2	10102	Michael	2125551500	Vitachrome Inc.
3	10103	Jonas	07-98 9555	Baane Mini Imports
4	10104	Diego	(91) 555 94 44	Euro Shopping Channel
...	...	...	...	...
302	10421	Valarie	4155551450	Mini Gifts Distributors Ltd.
303	10422	Kyung	2155551555	Diecast Classics Inc.
304	10423	Catherine	(02) 5554 67	Petit Auto
305	10424	Diego	(91) 555 94 44	Euro Shopping Channel
306	10425	Janine	40.67.8555	La Rochelle Gifts

307 rows × 4 columns

Joining the ORDERTABLE, CUSTOMERTABLE and producttable

In [70]:
```
# Joining the ORDERTABLE and CUSTOMERTABLE

sql = '''
SELECT ordernumber, contactfirstname, phone, ordertable.customername, line
FROM ordertable
RIGHT JOIN customertable ON ordertable.customername = customertable.customername
LEFT JOIN producttable ON producttable.code = ordertable.productcode
GROUP BY ordernumber
ORDER BY ordernumber ASC;
'''

# Executing the SQL command
crsr.execute(sql)

# Fetching results
result = crsr.fetchall()

final_result = pd.DataFrame(result)
headers = [i[0] for i in crsr.description]
final_result.columns = headers
final_result
```

Out[70]:

	ordernumber	contactfirstname	phone	customername	line
0	10100	Valarie	6035558647	Online Diecast Creations Co.	Vintage Cars
1	10101	Roland	+49 69 66 90 2555	Blauer See Auto, Co.	Vintage Cars
2	10102	Michael	2125551500	Vitachrome Inc.	Vintage Cars
3	10103	Jonas	07-98 9555	Baane Mini Imports	Classic Cars
4	10104	Diego	(91) 555 94 44	Euro Shopping Channel	Classic Cars
...	...	...	...	...	...
302	10421	Valarie	4155551450	Mini Gifts Distributors Ltd.	Vintage Cars
303	10422	Kyung	2155551555	Diecast Classics Inc.	Vintage Cars
304	10423	Catherine	(02) 5554 67	Petit Auto	Vintage Cars
305	10424	Diego	(91) 555 94 44	Euro Shopping Channel	Classic Cars
306	10425	Janine	40.67.8555	La Rochelle Gifts	Classic Cars

307 rows × 5 columns

Understanding VIEWS

Creating a view for sales of the product

In [71]:
```
# Creating a view for sales data

sql = '''
CREATE VIEW sales_data AS
SELECT ordernumber, contactfirstname, phone, ordertable.customername, line, productta
ble.code,
(producttable.msrp*ordertable.quantityordered) AS saleamount
FROM ordertable
RIGHT JOIN customertable ON ordertable.customername = customertable.customername
LEFT JOIN producttable ON producttable.code = ordertable.productcode
ORDER BY ordernumber ASC;
'''


# Executing the SQL command
crsr.execute(sql)


# Selecting the view created
sql = '''
SELECT *
FROM sales_data;
'''

# Executing the SQL command
crsr.execute(sql)

# Fetching results
result = crsr.fetchall()

final_result = pd.DataFrame(result)
headers = [i[0] for i in crsr.description]
final_result.columns = headers
final_result
```

Out[71]:

	ordernumber	contactfirstname	phone	customername	line	code	saleamount
0	10100	Valarie	6035558647	Online Diecast Creations Co.	Vintage Cars	S18_1749	5100
1	10100	Valarie	6035558647	Online Diecast Creations Co.	Vintage Cars	S18_2248	3000
2	10100	Valarie	6035558647	Online Diecast Creations Co.	Vintage Cars	S18_4409	2024
3	10100	Valarie	6035558647	Online Diecast Creations Co.	Vintage Cars	S24_3969	2009
4	10101	Roland	+49 69 66 90 2555	Blauer See Auto, Co.	Vintage Cars	S18_2325	3175
...	...	...	...	...	...	...	...
2818	10425	Janine	40.67.8555	La Rochelle Gifts	Trucks and Buses	S24_2300	6223
2819	10425	Janine	40.67.8555	La Rochelle Gifts	Classic Cars	S24_2840	1085
2820	10425	Janine	40.67.8555	La Rochelle Gifts	Trucks and Buses	S32_1268	3936
2821	10425	Janine	40.67.8555	La Rochelle Gifts	Trucks and Buses	S32_2509	594
2822	10425	Janine	40.67.8555	La Rochelle Gifts	Trucks and Buses	S50_1392	2070

2823 rows × 7 columns

Dropping the view created

In [72]:
```
# Dropping sales_view
sql = '''
DROP VIEW sales_data
'''

# Executing the SQL command
crsr.execute(sql)
```

Saving the changes made to the database

In [73]:
```
# To save the changes in the files. Never skip this.
# If we skip this, nothing will be saved in the database.
connection.commit()
```

Ending the connection with MySQL database and ending the cursor

In [74]:
```
# close the cursor
crsr.close()
# close the connection
connection.close()
```

Report

The data files used in this document are

```
poe_stats.csv
    contained within the database poe_dmdd

ordertable.csv, producttable.csv, customertable.csv
    contained within sales_dmdd
```

The description of every code can be found in the comments or description heading them. The flow of the document is created in a format where no residual content is left on the operating workstation and can help better understand the concepts regarding SQL.

Conclusion

In this document, we understand the basics as well as few of the advanced concepts of SQL. We have successfully understood the basic structure of SQL databases and tables with it's syntax, datatypes and operators. We have glanced over various types of SQL commands like Data Definition Language (DDL) and Data Manipulation Language (DML). We have understood the usage of clauses and conditions to extract meaningful data. The concept of Data Normalization has been overviewd throughly to create better databases. Advanced concepts of JOINS and VIEWS were also understood. The procedure for employing transactions and rollback features in the database had been throughly displayed and understood in the document.

Contribution

The code is completely original to employ for learning and practicing for understanding SQL.

The data is sourced from Kaggle (links) https://www.kaggle.com/gagazet/path-of-exile-league-statistic (https://www.kaggle.com/gagazet/path-of-exile-league-statistic) https://www.kaggle.com/kyanyoga/sample-sales-data (https://www.kaggle.com/kyanyoga/sample-sales-data)

License