Contents

Learning SQL 1

July 21, 2020

Part I

Learning SQL

1 Introduction to SQL

SQL is a standard language query for storing, manipulating and retrieving data in databases.

Our SQL tutorial will teach you how to use SQL in: MySQL, SQL Server, MS Access, Oracle, Sybase, Informix, Postgres, and other database systems.

SQL is a standard language for accessing and manipulating databases.

1.1 What is SQL?

- SQL stands for Structured Query Language
- SQL lets you access and manipulate databases
- SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987

1.2 What Can SQL do?

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database
- SQL can create new databases
- SQL can create new tables in a database
- SQL can create stored procedures in a database
- SQL can create views in a database
- SQL can set permissions on tables, procedures, and views

1.3 RDBMS

- RDBMS stands for Relational Database Management System.

- RDBMS is the basis for SQL, and for all modern database systems such as MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.
- The data in RDBMS is stored in database objects called tables. A table is a collection of related data entries and it consists of columns and rows.

Look at the "Customers" table:

[105]:
```
# Create connection:

import mysql.connector
import pandas as pd

def query(query):
    mySql_connector = mysql.connector.connect(
    host="localhost",
    user="root",
    passwd="",
    database="northwind"
    )
    data= pd.read_sql_query(query , mySql_connector)
    return data
```

In case of Error connection in MySQL:

```
SELECT * FROM northwind.customers; Drop user root\@localhost;
CREATE USER root@localhost IDENTIFIED BY 'passpass';
grant all privileges on *.* to root@localhost with grant option;
flush privileges;
```

2 Database Tables

A database most often contains one or more tables. Each table is identified by a name (e.g. "Customers" or "Orders"). Tables contain records (rows) with data.

In this tutorial we will use the well-known Northwind sample database (included in MS Access and MS SQL Server).

SQL keywords are NOT case sensitive: SELECT is the same as SELECT

Some of The Most Important SQL Commands

* SELECT - extracts data from a database

* UPDATE - updates data in a database

* DELETE - deletes data from a database

* INSERT INTO - inserts new data into a database

* CREATE DATABASE - creates a new database

* ALTER DATABASE - modifies a database

* CREATE TABLE - creates a new table

* ALTER TABLE - modifies a table

* DROP TABLE - deletes a table

* CREATE INDEX - creates an index (search key)

* DROP INDEX - deletes an index

The SQL SELECT Statement

SELECT column1, column2, ... FROM table_name;

[106]:
```
data=query("SELECT * FROM northwind.customers;").head()
data
```

[106]:
```
   id    company         last_name first_name email_address  \
0   1  Company A            Bedecs       Anna          None
1   2  Company B  Gratacos Solsona    Antonio          None
2   3  Company C              Axen     Thomas          None
3   4  Company D            Nemati      Parsa          None
4   5  Company E         O'Donnell     Martin          None

                   job_title business_phone home_phone mobile_phone  \
0                      Owner  (123)555-0100       None         None
1                      Owner  (123)555-0100       None         None
2  Purchasing Representative  (123)555-0100       None         None
3         Purchasing Manager  (123)555-0100       None         None
4                      Owner  (123)555-0100       None         None

      fax_number         address         city state_province␣
↪zip_postal_code  \
0  (123)555-0101  123 1st Street      Seattle             WA         ␣
↪ 99999
1  (123)555-0101  123 2nd Street       Boston             MA         ␣
↪ 99999
2  (123)555-0101  123 3rd Street  Los Angelas             CA         ␣
↪ 99999
3  (123)555-0101  123 4th Street     New York             NY         ␣
↪ 99999
4  (123)555-0101  123 5th Street  Minneapolis             MN         ␣
↪ 99999

  country_region web_page notes attachments
0            USA     None  None
1            USA     None  None
2            USA     None  None
3            USA     None  None
4            USA     None  None
```

[107]:
```
list_data=list(data)
list_data
```

[107]:
```
['id',
 'company',
 'last_name',
 'first_name',
 'email_address',
 'job_title',
 'business_phone',
 'home_phone',
 'mobile_phone',
 'fax_number',
 'address',
 'city',
 'state_province',
 'zip_postal_code',
 'country_region',
 'web_page',
 'notes',
 'attachments']
```

[108]:
```
run_query="SELECT last_name as last_Name, first_name, city FROM␣
 ↪northwind.customers;"
query(run_query).head()
```

[108]:

	last_Name	first_name	city
0	Bedecs	Anna	Seattle
1	Gratacos Solsona	Antonio	Boston
2	Axen	Thomas	Los Angelas
3	Nemati	Parsa	New York
4	O'Donnell	Martin	Minneapolis

3 The SQL SELECT DISTINCT Statement

The SELECT DISTINCT ...

statement is used to return only distinct (different) values.

Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values.

[109]:
```
data=query("SELECT distinct state_province FROM northwind.customers;")
data
```

[109]:

	state_province
0	CA
1	CO

```
2                FL
3                HI
4                ID
5                IL
6                MA
7                MN
8                NV
9                NY
10               OR
11               TN
12               UT
13               WA
14               WI
```

[110]:
```
# SELECT COUNT(DISTINCT Country) FROM Customers;
query("SELECT COUNT( state_province) FROM northwind.customers;")
```

[110]:
```
   COUNT( state_province)
0                      29
```

[111]:
```
query('''
SELECT COUNT(DISTINCT state_province) FROM northwind.customers;
''')
```

[111]:
```
   COUNT(DISTINCT state_province)
0                              15
```

4 The SQL WHERE Clause

The WHERE clause is used to filter records.

The WHERE clause is used to extract only those records that fulfill a specified condition.

WHERE Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

[112]:
```
query('''
SELECT  * FROM northwind.customers WHERE city ='seattle';
''')
```

[112]:
```
   id    company last_name     first_name email_address job_title  \
0   1  Company A    Bedecs           Anna          None     Owner
1  17  Company Q     Bagel  Jean Philippe          None     Owner
```

```
  business_phone home_phone mobile_phone     fax_number          ␣
 ↪address  \
0  (123)555-0100       None         None  (123)555-0101   123 1st␣
 ↪Street
1  (123)555-0100       None         None  (123)555-0101  456 17th␣
 ↪Street

      city state_province zip_postal_code country_region web_page␣
 ↪notes  \
0  Seattle             WA           99999            USA     None ␣
 ↪None
1  Seattle             WA           99999            USA     None ␣
 ↪None

  attachments
0
1
```

5 Text Fields vs. Numeric Fields

SQL requires single quotes around text values (most database systems will also allow double quotes).

However, numeric fields should not be enclosed in quotes:

[113]:
```
query('''
    SELECT * FROM northwind.customers
    WHERE id=1;
    ''')
```

[113]:
```
   id    company last_name first_name email_address job_title␣
 ↪business_phone  \
0   1  Company A    Bedecs       Anna           None     Owner ␣
 ↪(123)555-0100

  home_phone mobile_phone     fax_number         address     city  \
0       None         None  (123)555-0101  123 1st Street  Seattle

  state_province zip_postal_code country_region web_page notes␣
 ↪attachments
0             WA           99999            USA     None  None
```

[114]:
```
pwd
```

Operator	Description
=	Equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
<>	Not equal. **Note:** In some versions of SQL this operator may be written as !=
BETWEEN	Between a certain range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

6 SQL AND, OR and NOT Operators

6.1 AND Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 AND condition2 AND condition3 ...;
```

6.2 OR Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 OR condition2 OR condition3 ...;
```

6.3 NOT Syntax

```
SELECT column1, column2, ...
FROM table_name
WHERE NOT condition;
```

[115]:
```
query('''
    SELECT * FROM northwind.customers
    WHERE country_region='USA' AND city='Seattle';
    ''')
```

[115]:
```
   id    company last_name     first_name email_address job_title  \
0   1  Company A    Bedecs           Anna          None     Owner
1  17  Company Q     Bagel  Jean Philippe          None     Owner
```

```
  business_phone home_phone mobile_phone     fax_number          ␣
 ↪address  \
0  (123)555-0100       None         None  (123)555-0101   123 1st␣
 ↪Street
1  (123)555-0100       None         None  (123)555-0101  456 17th␣
 ↪Street

      city state_province zip_postal_code country_region web_page␣
 ↪notes  \
0  Seattle             WA           99999            USA     None ␣
 ↪None
1  Seattle             WA           99999            USA     None ␣
 ↪None

  attachments
0
1
```

[116]:
```
query('''
    SELECT * FROM northwind.customers
    WHERE country_region='USA'  or city = 'Boston';
    ''').head()
```

[116]:
```
   id    company         last_name first_name email_address  \
0   1  Company A            Bedecs       Anna          None
1   2  Company B  Gratacos Solsona    Antonio          None
2   3  Company C              Axen     Thomas          None
3   4  Company D            Nemati      Parsa          None
4   5  Company E         O'Donnell     Martin          None

                   job_title business_phone home_phone mobile_phone  \
0                      Owner  (123)555-0100       None         None
1                      Owner  (123)555-0100       None         None
2  Purchasing Representative  (123)555-0100       None         None
3        Purchasing Manager  (123)555-0100       None         None
4                      Owner  (123)555-0100       None         None

      fax_number         address         city state_province␣
 ↪zip_postal_code  \
0  (123)555-0101  123 1st Street      Seattle             WA         ␣
 ↪ 99999
1  (123)555-0101  123 2nd Street       Boston             MA         ␣
 ↪ 99999
2  (123)555-0101  123 3rd Street  Los Angelas             CA         ␣
 ↪ 99999
```

```
3  (123)555-0101  123 4th Street     New York             NY
 ↪ 99999
4  (123)555-0101  123 5th Street  Minneapolis             MN
 ↪ 99999

  country_region web_page notes attachments
0            USA     None  None
1            USA     None  None
2            USA     None  None
3            USA     None  None
4            USA     None  None
```

[117]:
```
query('''
    SELECT * FROM northwind.customers
    WHERE country_region='USA' AND city='Seattle' and not city =
 ↪'Boston';
    ''')
```

[117]:
```
   id    company last_name     first_name email_address job_title  \
0   1  Company A    Bedecs           Anna          None     Owner
1  17  Company Q     Bagel  Jean Philippe          None     Owner

  business_phone home_phone mobile_phone     fax_number
 ↪address  \
0  (123)555-0100       None         None  (123)555-0101   123 1st
 ↪Street
1  (123)555-0100       None         None  (123)555-0101  456 17th
 ↪Street

      city state_province zip_postal_code country_region web_page
 ↪notes  \
0  Seattle             WA           99999            USA     None
 ↪None
1  Seattle             WA           99999            USA     None
 ↪None

  attachments
0
1
```

[118]:
```
query('''
    SELECT * FROM northwind.customers
    WHERE country_region='USA' AND (City='Portland' OR City='San
 ↪Francisco');
    ''')
```

[118]:
```
   id    company    last_name first_name email_address  \
0   8  Company H     Andersen  Elizabeth          None
1  23  Company W        Entin    Michael          None
2  16  Company P  Goldschmidt     Daniel          None

                   job_title business_phone home_phone mobile_phone  \
0  Purchasing Representative  (123)555-0100       None         None
1         Purchasing Manager  (123)555-0100       None         None
2  Purchasing Representative  (123)555-0100       None         None

      fax_number          address           city state_province  \
0  (123)555-0101   123 8th Street       Portland             OR
1  (123)555-0101  789 23th Street       Portland             OR
2  (123)555-0101  456 16th Street  San Francisco             CA

  zip_postal_code country_region web_page notes attachments
0           99999            USA     None  None
1           99999            USA     None  None
2           99999            USA     None  None
```

[119]:
```
query('''
    SELECT last_name , zip_postal_code, first_name FROM northwind.
↪customers
    WHERE country_region='USA' AND (City='Portland' OR City='San␣
↪Francisco');
    ''')
```

[119]:
```
     last_name zip_postal_code first_name
0     Andersen           99999  Elizabeth
1        Entin           99999    Michael
2  Goldschmidt           99999     Daniel
```

7 SQL ORDER BY Keyword

- The ORDER BY keyword is used to sort the result-set in ascending or descending order.
- The ORDER BY keyword sorts the records in ascending order by default.
- To sort the records in descending order, use the DESC keyword.

7.1 ORDER BY Syntax

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC | DESC;
```

[120]:
```
# The SQL ORDER BY Keyword
query('''
    SELECT distinct city FROM northwind.customers
    order by city DESC
    ''').head()
```

[120]:
```
             city
0         Seattle
1   San Francisco
2  Salt Lake City
3        Portland
4        New York
```

[121]:
```
list_data
```

[121]:
```
['id',
 'company',
 'last_name',
 'first_name',
 'email_address',
 'job_title',
 'business_phone',
 'home_phone',
 'mobile_phone',
 'fax_number',
 'address',
 'city',
 'state_province',
 'zip_postal_code',
 'country_region',
 'web_page',
 'notes',
 'attachments']
```

The following SQL statement SELECTs all customers from the "Customers" table, sorted by the "city" and the "first_name" column. This means that it orders by city, but if some rows have the same city, it orders them by first_name:

[122]:
```
query('''
    SELECT city,first_name, company , last_name   , job_title ,␣
 ↪country_region , zip_postal_code
    FROM northwind.customers
    order by city , first_name
    ''').head()
```

[122]:
```
      city first_name    company         last_name                ␣
 ↪job_title  \
```

```
0    Boise  Ming-Yang  Company G               Xie                    ␣
↪  Owner
1   Boston    Antonio  Company B  Gratacos Solsona                    ␣
↪  Owner
2   Boston  Catherine  Company R     Autier Miconi  Purchasing␣
↪Representative
3  Chicago       John  Company Y            Rodman         Purchasing␣
↪Manager
4  Chicago     Roland  Company J            Wacker         Purchasing␣
↪Manager

  country_region zip_postal_code
0            USA           99999
1            USA           99999
2            USA           99999
3            USA           99999
4            USA           99999
```

[123]:
```
query('''
    SELECT city,first_name, company , last_name   , job_title ,␣
↪country_region , zip_postal_code
    FROM northwind.customers
    order by city  ASC, first_name DESC
    ''').head()
```

[123]:
```
      city first_name    company         last_name                ␣
↪job_title  \
0    Boise  Ming-Yang  Company G               Xie                    ␣
↪  Owner
1   Boston  Catherine  Company R     Autier Miconi  Purchasing␣
↪Representative
2   Boston    Antonio  Company B  Gratacos Solsona                    ␣
↪  Owner
3  Chicago     Roland  Company J            Wacker         Purchasing␣
↪Manager
4  Chicago       John  Company Y            Rodman         Purchasing␣
↪Manager

  country_region zip_postal_code
0            USA           99999
1            USA           99999
2            USA           99999
3            USA           99999
4            USA           99999
```

8 SQL INSERT INTO Statement

The INSERT INTO statement is used to insert new records in a table.

INSERT INTO Syntax

It is possible to write the INSERT INTO statement in two ways.

The first way specifies both the column names and the values to be inserted:

```
INSERT INTO table_name
(column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. The INSERT INTO syntax would be as follows:

```
INSERT INTO table_name
VALUES (value1, value2, value3, ...);
```

INSERT INTO Example

```
INSERT INTO Customers (CustomerName, ContactName,

Address, City, PostalCode, Country)

VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21',

'Stavanger', '4006', 'Norway');
```

9 SQL NULL Values

A field with a NULL value is a field with no value.

If a field in a table is optional, it is possible to insert a new record or update a record without adding a value to this field. Then, the field will be saved with a NULL value.

Note: A NULL value is different from a zero value or a field that contains spaces. A field with a NULL value is one that has been left blank during record creation!

How to Test for NULL Values? It is not possible to test for NULL values with comparison operators, such as =, <, or <>.

We will have to use the IS NULL and IS NOT NULL operators instead.

9.1 IS NULL Syntax:

```
SELECT column_names
FROM table_name
WHERE column_name IS NULL;
```

9.2 IS NOT NULL Syntax:

```
SELECT column_names
FROM table_name
WHERE column_name IS NOT NULL;
```

9.3 The IS NULL Operator:

The IS NULL operator is used to test for empty values (NULL values).

The following SQL lists all customers with a NULL value in the "Address" field:

```
SELECT CustomerName, ContactName, Address
FROM Customers
WHERE Address IS NULL;
```

9.4 The IS NOT NULL Operator:

The IS NOT NULL operator is used to test for non-empty values (NOT NULL values).

The following SQL lists all customers with a value in the "Address" field:

```
SELECT CustomerName, ContactName, Address
FROM Customers
WHERE Address IS NOT NULL;
```

[125]:
```
print(len(query('''
    SELECT city,first_name, company , last_name   , job_title ,␣
 ↪country_region , zip_postal_code
    FROM northwind.customers
    WHERE company IS NULL;
    ''')))

# IS NOT NULL;

print(len(query('''
    SELECT city,first_name, company , last_name   , job_title ,␣
 ↪country_region , zip_postal_code
    FROM northwind.customers
    WHERE company IS  NOT NULL;
    ''')))
```

```
0
29
```

10 SQL UPDATE Statement

The UPDATE statement is used to modify the existing records in a table.

10.1 UPDATE Syntax

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

10.2 UPDATE Table:

The following SQL statement updates the first customer (CustomerID = 1) with a new contact person and a new city.

```
UPDATE Customers
SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'
WHERE CustomerID = 1;
```

10.3 UPDATE Multiple Records

It is the WHERE clause that determines how many records will be updated.

The following SQL statement will update the contactname to "Juan" for all records where country is "Mexico":

```
UPDATE Customers
SET ContactName='Juan'
WHERE Country='Mexico';
```

[126]:
```
query('''
    SELECT *
    FROM northwind.customers

    ''').head()
```

[126]:
```
   id    company         last_name first_name email_address  \
0   1  Company A            Bedecs       Anna          None
1   2  Company B  Gratacos Solsona    Antonio          None
2   3  Company C              Axen     Thomas          None
3   4  Company D            Nemati      Parsa          None
4   5  Company E         O'Donnell     Martin          None

                    job_title business_phone home_phone mobile_phone  \
0                       Owner  (123)555-0100       None         None
```

```
1                      Owner  (123)555-0100       None         None
2  Purchasing Representative  (123)555-0100       None         None
3         Purchasing Manager  (123)555-0100       None         None
4                      Owner  (123)555-0100       None         None

      fax_number         address         city state_province␣
↪zip_postal_code  \
0  (123)555-0101  123 1st Street      Seattle             WA         ␣
↪ 99999
1  (123)555-0101  123 2nd Street       Boston             MA         ␣
↪ 99999
2  (123)555-0101  123 3rd Street  Los Angelas             CA         ␣
↪ 99999
3  (123)555-0101  123 4th Street     New York             NY         ␣
↪ 99999
4  (123)555-0101  123 5th Street  Minneapolis             MN         ␣
↪ 99999

  country_region web_page notes attachments
0            USA     None  None
1            USA     None  None
2            USA     None  None
3            USA     None  None
4            USA     None  None
```

query("' UPDATE northwind.customers SET firstname = 'Soodabeh' , lastname='Afshar' WHERE id = 1;

"').head()

11 SQL DELETE Statement

DELETE FROM table_name WHERE condition;

12 SQL TOP, LIMIT or ROWNUM Clause

The SELECT TOP clause is used to specify the number of records to return.

The SELECT TOP clause is useful on large tables with thousands of records. Returning a large number of records can impact performance.

```
SELECT column_name(s)
FROM table_name
WHERE condition
LIMIT number;
```

12.1 SQL LIMIT Example

```
SELECT * FROM Customers
LIMIT 3;
```

```
SELECT TOP 50 PERCENT * FROM Customers;
```

12.2 ADD a WHERE CLAUSE

```
SELECT TOP 3 * FROM Customers
WHERE Country='Germany';
```

```
SELECT * FROM Customers
WHERE Country='Germany'
LIMIT 3;
```

[127]:
```
query('''
    SELECT *
    FROM northwind.customers
    LIMIT 3;

    ''')
```

[127]:
```
   id    company         last_name first_name email_address  \
0   1  Company A            Bedecs       Anna          None
1   2  Company B  Gratacos Solsona    Antonio          None
2   3  Company C              Axen     Thomas          None

                   job_title business_phone home_phone mobile_phone  \
0                      Owner  (123)555-0100       None         None
1                      Owner  (123)555-0100       None         None
2  Purchasing Representative  (123)555-0100       None         None

      fax_number         address         city state_province␣
↪zip_postal_code  \
0  (123)555-0101  123 1st Street      Seattle             WA         ␣
↪ 99999
1  (123)555-0101  123 2nd Street       Boston             MA         ␣
↪ 99999
2  (123)555-0101  123 3rd Street  Los Angelas             CA         ␣
↪ 99999
```

```
  country_region web_page notes attachments
0            USA     None  None
1            USA     None  None
2            USA     None  None
```

13 SQL MIN() and MAX() Functions

The MIN() function returns the smallest value of the SELECTed column.

The MAX() function returns the largest value of the SELECTed column.

```
SELECT MIN(column_name) | MAX(column_name)
FROM table_name
WHERE condition;
```

14 The SQL COUNT(), AVG() and SUM() Functions

The COUNT() function returns the number of rows that matches a specified criteria.

The AVG() function returns the average value of a numeric column.

The SUM() function returns the total sum of a numeric column.

```
SELECT COUNT(column_name) | AVG(column_name) | SUM(column_name)
FROM table_name
WHERE condition;
```

Note: NULL values are ignored.

[128]:
```
query('''
    SELECT MIN(list_price) AS Smallest_Price ,
    MAX(list_price) as Bigest_Price ,
    AVG(list_price) as Average_Price ,
    SUM(list_price) as SUM_Price ,
    COUNT(list_price) as COUNT_Price
    FROM products;

    ''')
```

[128]:
```
   Smallest_Price  Bigest_Price  Average_Price  SUM_Price  COUNT_Price
0             1.2          81.0      15.845778     713.06           45
```

15 SQL LIKE Operator

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

There are two wildcards often used in conjunction with the LIKE operator:

% - The percent sign represents zero, one, or multiple characters

_ - The underscore represents a single character

Note: MS Access uses an asterisk (*) instead of the percent sign (%), and a question mark (?) instead of the underscore (_).

```
SELECT column1, column2, ...
FROM table_name
WHERE columnN LIKE pattern;
```

Here are some examples showing different LIKE operators with '%' and '_' wildcards:

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length
WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

[129]:
```
query('''
    SELECT *
    FROM northwind.customers
    WHERE first_name
    LIKE 'a%';
    ''')
```

[129]:
```
   id     company         last_name first_name email_address  \
0  19   Company S           Eggerer  Alexander          None
1  28  Company BB            Raghav  Amritansh          None
2  13   Company M            Ludick      Andre          None
3   1   Company A            Bedecs       Anna          None
4   2   Company B  Gratacos Solsona    Antonio          None

                    job_title business_phone home_phone mobile_phone  \
0       Accounting Assistant  (123)555-0100       None         None
1         Purchasing Manager  (123)555-0100       None         None
2  Purchasing Representative  (123)555-0100       None         None
3                      Owner  (123)555-0100       None         None
4                      Owner  (123)555-0100       None         None
```

```
      fax_number          address         city state_province␣
↪zip_postal_code  \
0  (123)555-0101  789 19th Street  Los Angelas             CA        ␣
↪  99999
1  (123)555-0101  789 28th Street      Memphis             TN        ␣
↪  99999
2  (123)555-0101  456 13th Street      Memphis             TN        ␣
↪  99999
3  (123)555-0101   123 1st Street      Seattle             WA        ␣
↪  99999
4  (123)555-0101   123 2nd Street       Boston             MA        ␣
↪  99999

  country_region web_page notes attachments
0            USA     None  None
1            USA     None  None
2            USA     None  None
3            USA     None  None
4            USA     None  None
```

[130]:
```
query('''
    SELECT *
    FROM northwind.customers
    WHERE first_name
    LIKE '%a';
    ''')
```

[130]:
```
   id    company last_name first_name email_address            ␣
↪job_title  \
0   1  Company A    Bedecs       Anna          None                 ␣
↪Owner
1   4  Company D    Nemati      Parsa          None    Purchasing␣
↪Manager
2  15  Company O   Kupkova     Helena          None    Purchasing␣
↪Manager
3  22  Company V     Ramos    Luciana          None  Purchasing␣
↪Assistant

  business_phone home_phone mobile_phone     fax_number          ␣
↪address  \
0  (123)555-0100       None         None  (123)555-0101   123 1st␣
↪Street
1  (123)555-0100       None         None  (123)555-0101   123 4th␣
↪Street
2  (123)555-0100       None         None  (123)555-0101  456 15th␣
↪Street
```

```
3  (123)555-0100       None         None  (123)555-0101  789 22th␣
↪Street

        city state_province zip_postal_code country_region web_page␣
↪notes  \
0    Seattle             WA           99999            USA     None ␣
↪None
1   New York             NY           99999            USA     None ␣
↪None
2   Honolulu             HI           99999            USA     None ␣
↪None
3  Milwaukee             WI           99999            USA     None ␣
↪None

  attachments
0
1
2
3
```

[131]:
```
query('''
    SELECT *
    FROM northwind.customers
    WHERE first_name
    LIKE '%rs%';
    ''')
```

[131]:
```
   id    company last_name first_name email_address          ␣
↪job_title  \
0   4  Company D    Nemati      Parsa          None  Purchasing␣
↪Manager

  business_phone home_phone mobile_phone     fax_number        ␣
↪address  \
0  (123)555-0100       None         None  (123)555-0101  123 4th␣
↪Street

       city state_province zip_postal_code country_region web_page␣
↪notes  \
0  New York             NY           99999            USA     None ␣
↪None

  attachments
0
```

[132]:
```
query('''
    SELECT *
    FROM northwind.customers
    WHERE first_name
    LIKE '__rs%';
    ''')
```

[132]:
```
   id    company last_name first_name email_address          ␣
↪job_title  \
0   4  Company D    Nemati      Parsa          None  Purchasing␣
↪Manager

  business_phone home_phone mobile_phone     fax_number        ␣
↪address  \
0  (123)555-0100       None         None  (123)555-0101  123 4th␣
↪Street

       city state_province zip_postal_code country_region web_page␣
↪notes  \
0  New York             NY           99999            USA     None ␣
↪None

  attachments
0
```

[133]:
```
query('''
    SELECT *
    FROM northwind.customers
    WHERE first_name
    NOT LIKE '_a%';
    ''').head()
```

[133]:
```
   id    company         last_name first_name email_address  \
0   1  Company A            Bedecs       Anna          None
1   2  Company B  Gratacos Solsona    Antonio          None
2   3  Company C              Axen     Thomas          None
3   6  Company F      Pérez-Olaeta  Francisco          None
4   7  Company G               Xie  Ming-Yang          None

                    job_title business_phone home_phone mobile_phone  \
0                       Owner  (123)555-0100       None         None
1                       Owner  (123)555-0100       None         None
2  Purchasing Representative  (123)555-0100       None         None
3         Purchasing Manager  (123)555-0100       None         None
4                       Owner  (123)555-0100       None         None
```

```
      fax_number         address         city state_province␣
↪zip_postal_code  \
0  (123)555-0101  123 1st Street      Seattle             WA         ␣
↪ 99999
1  (123)555-0101  123 2nd Street       Boston             MA         ␣
↪ 99999
2  (123)555-0101  123 3rd Street  Los Angelas             CA         ␣
↪ 99999
3  (123)555-0101  123 6th Street    Milwaukee             WI         ␣
↪ 99999
4  (123)555-0101  123 7th Street        Boise             ID         ␣
↪ 99999

  country_region web_page notes attachments
0            USA     None  None
1            USA     None  None
2            USA     None  None
3            USA     None  None
4            USA     None  None
```

16 SQL IN Operator

The IN operator allows you to specify multiple values in a WHERE clause.

The IN operator is a shorthand for multiple OR conditions.

IN Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, ...);
```

or:

SELECT column_name(s)

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (SELECT STATEMENT);
```

[134]:
```
query('''
    SELECT *
    FROM northwind.customers
    WHERE job_title IN ('Owner');
    ''')
```

[134]:
```
   id    company         last_name     first_name email_address␣
↪job_title  \
```

```
0   1  Company A            Bedecs           Anna          None    ␣
↪Owner
1   2  Company B  Gratacos Solsona        Antonio          None    ␣
↪Owner
2   5  Company E         O'Donnell         Martin          None    ␣
↪Owner
3   7  Company G               Xie      Ming-Yang          None    ␣
↪Owner
4  17  Company Q             Bagel  Jean Philippe          None    ␣
↪Owner
5  24  Company X        Hasselberg          Jonas          None    ␣
↪Owner

  business_phone home_phone mobile_phone     fax_number          ␣
↪address  \
0  (123)555-0100       None         None  (123)555-0101   123 1st␣
↪Street
1  (123)555-0100       None         None  (123)555-0101   123 2nd␣
↪Street
2  (123)555-0100       None         None  (123)555-0101   123 5th␣
↪Street
3  (123)555-0100       None         None  (123)555-0101   123 7th␣
↪Street
4  (123)555-0100       None         None  (123)555-0101  456 17th␣
↪Street
5  (123)555-0100       None         None  (123)555-0101  789 24th␣
↪Street

             city state_province zip_postal_code country_region␣
↪web_page  \
0         Seattle             WA           99999            USA    ␣
↪None
1          Boston             MA           99999            USA    ␣
↪None
2     Minneapolis             MN           99999            USA    ␣
↪None
3           Boise             ID           99999            USA    ␣
↪None
4         Seattle             WA           99999            USA    ␣
↪None
5  Salt Lake City             UT           99999            USA    ␣
↪None

  notes attachments
0  None
```

```
1  None
2  None
3  None
4  None
5  None
```

[135]:
```
query('''
    SELECT *
    FROM northwind.customers
    WHERE job_title NOT IN ('Owner' , 'Purchasing Manager');
    ''')
```

[135]:
```
   id    company      last_name first_name email_address  \
0   3  Company C           Axen     Thomas          None
1   8  Company H       Andersen  Elizabeth          None
2  13  Company M         Ludick      Andre          None
3  14  Company N          Grilo     Carlos          None
4  16  Company P    Goldschmidt     Daniel          None
5  18  Company R  Autier Miconi  Catherine          None
6  19  Company S        Eggerer  Alexander          None
7  21  Company U           Tham    Bernard          None
8  22  Company V          Ramos    Luciana          None
9  26  Company Z            Liu        Run          None

                   job_title business_phone home_phone mobile_phone  \
0  Purchasing Representative  (123)555-0100       None         None
1  Purchasing Representative  (123)555-0100       None         None
2  Purchasing Representative  (123)555-0100       None         None
3  Purchasing Representative  (123)555-0100       None         None
4  Purchasing Representative  (123)555-0100       None         None
5  Purchasing Representative  (123)555-0100       None         None
6       Accounting Assistant  (123)555-0100       None         None
7         Accounting Manager  (123)555-0100       None         None
8       Purchasing Assistant  (123)555-0100       None         None
9       Accounting Assistant  (123)555-0100       None         None

      fax_number          address           city state_province  \
0  (123)555-0101   123 3rd Street    Los Angelas             CA
1  (123)555-0101   123 8th Street       Portland             OR
2  (123)555-0101  456 13th Street        Memphis             TN
3  (123)555-0101  456 14th Street         Denver             CO
4  (123)555-0101  456 16th Street  San Francisco             CA
5  (123)555-0101  456 18th Street         Boston             MA
6  (123)555-0101  789 19th Street    Los Angelas             CA
7  (123)555-0101  789 21th Street    Minneapolis             MN
8  (123)555-0101  789 22th Street      Milwaukee             WI
9  (123)555-0101  789 26th Street          Miami             FL
```

```
  zip_postal_code country_region web_page notes attachments
0           99999            USA     None  None
1           99999            USA     None  None
2           99999            USA     None  None
3           99999            USA     None  None
4           99999            USA     None  None
5           99999            USA     None  None
6           99999            USA     None  None
7           99999            USA     None  None
8           99999            USA     None  None
9           99999            USA     None  None
```

[136]:
```
# The following SQL statement SELECTs all customers that are from the␣
↪same countries as the suppliers:

query('''
    SELECT *
    FROM northwind.customers
    WHERE first_name not in
    (  SELECT first_name FROM northwind.suppliers);

    ''').head()
```

[136]:
```
   id    company         last_name first_name email_address  \
0   1  Company A            Bedecs       Anna          None
1   2  Company B  Gratacos Solsona    Antonio          None
2   3  Company C              Axen     Thomas          None
3   4  Company D            Nemati      Parsa          None
4   5  Company E         O'Donnell     Martin          None

                    job_title business_phone home_phone mobile_phone  \
0                       Owner  (123)555-0100       None         None
1                       Owner  (123)555-0100       None         None
2  Purchasing Representative  (123)555-0100       None         None
3         Purchasing Manager  (123)555-0100       None         None
4                       Owner  (123)555-0100       None         None

      fax_number         address         city state_province␣
↪zip_postal_code  \
0  (123)555-0101  123 1st Street      Seattle             WA         ␣
↪ 99999
1  (123)555-0101  123 2nd Street       Boston             MA         ␣
↪ 99999
2  (123)555-0101  123 3rd Street  Los Angelas             CA         ␣
↪ 99999
```

```
3  (123)555-0101  123 4th Street     New York             NY
↪ 99999
4  (123)555-0101  123 5th Street  Minneapolis             MN
↪ 99999

  country_region web_page notes attachments
0            USA     None  None
1            USA     None  None
2            USA     None  None
3            USA     None  None
4            USA     None  None
```

17 SQL BETWEEN Operator

The BETWEEN operator SELECTs values within a given range. The values can be numbers, text, or dates.

The BETWEEN operator is inclusive: begin and end values are included.

BETWEEN Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

17.1 For example:

```
SELECT * FROM Orders
WHERE OrderDate BETWEEN #01/07/1996# AND #31/07/1996#;
```

```
SELECT * FROM Products
WHERE Price BETWEEN 10 AND 20;
```

```
SELECT * FROM Products
WHERE Price NOT BETWEEN 10 AND 20;
```

```
SELECT * FROM Products
WHERE Price BETWEEN 10 AND 20
AND CategoryID NOT IN (1,2,3);
```

```
SELECT * FROM Products
WHERE ProductName BETWEEN 'Carnarvon Tigers' AND 'Mozzarella di Giovanni'
ORDER BY ProductName;

SELECT * FROM Products
WHERE ProductName BETWEEN "Carnarvon Tigers" AND "Chef Anton's Cajun Seasoning"
ORDER BY ProductName;


SELECT * FROM Products
WHERE ProductName NOT BETWEEN 'Carnarvon Tigers' AND 'Mozzarella di Giovanni'
ORDER BY ProductName;


SELECT * FROM Orders
WHERE OrderDate BETWEEN '1996-07-01' AND '1996-07-31';
```

[137]:
```
query('''
    SELECT list_price AS Smallest_Price
    FROM products
    WHERE list_price between 14 and 20;
    ''')
```

[137]:
```
   Smallest_Price
0           18.00
1           14.00
2           18.40
3           19.50
4           17.00
5           15.99
```

[138]:
```
query('''
    SELECT distinct list_price AS Smallest_Price
    FROM products
    WHERE list_price not between 2 and 81
    ORDER BY Smallest_Price ;
    ''')
```

[138]:
```
   Smallest_Price
0            1.20
1            1.30
2            1.50
3            1.80
4            1.89
5            1.95
```

18 SQL Aliases

SQL aliases are used to give a table, or a column in a table, a temporary name.

Aliases are often used to make column names more readable.

An alias only exists for the duration of the query. Aliases can be useful when:

- There are more than one table involved in a query

- Functions are used in the query

- Column names are big or not very readable

- Two or more columns are combined together

Alias Column Syntax

```
SELECT column_name AS alias_name
FROM table_name;
```

```
Alias Table Syntax

SELECT column_name(s)
FROM table_name AS alias_name;
```

18.1 For Example

```
SELECT CustomerID AS ID, CustomerName AS Customer
FROM Customers;
```

```
SELECT CustomerName AS Customer, ContactName AS [Contact Person]
FROM Customers;

SELECT CustomerName, Address + ', ' + PostalCode + ' ' + City + ', ' + Country AS A
FROM Customers;

SELECT CustomerName, CONCAT(Address,', ',PostalCode,', ',City,', ',Country) AS Addr
FROM Customers;

SELECT o.OrderID, o.OrderDate, c.CustomerName
FROM Customers AS c, Orders AS o
WHERE c.CustomerName='Around the Horn' AND c.CustomerID=o.CustomerID;
```

```
SELECT Orders.OrderID, Orders.OrderDate, Customers.CustomerName
FROM Customers, Orders
```

```
WHERE Customers.CustomerName='Around the Horn' AND Customers.CustomerID=Orders.Cust
```

[139]:
```
query('''
    SELECT distinct list_price AS Smallest_Price
    FROM products as pro
    WHERE pro.list_price=14;
    ''')
```

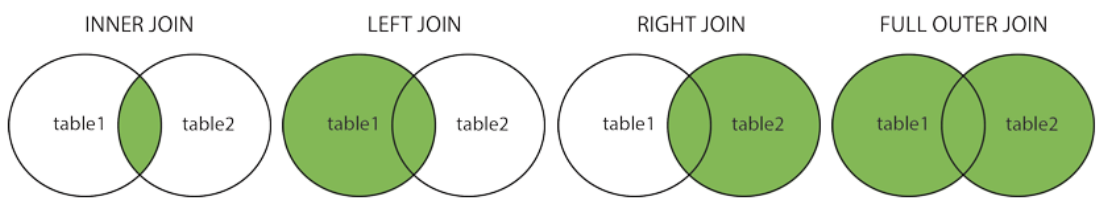
[139]:
```
   Smallest_Price
0            14.0
```

19 SQL JOIN

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

19.1 Different Types of SQL JOINs

Here are the different types of the JOINs in SQL:

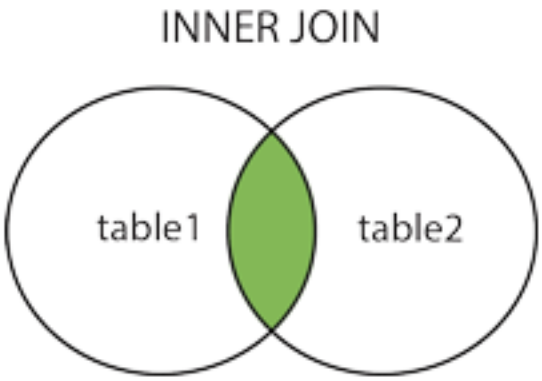



```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

19.2 SQL INNER JOIN Example

```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
FROM Orders
INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;
```

19.2.1 (INNER) JOIN: Returns records that have matching values in both tables



The INNER JOIN keyword SELECTs records that have matching values in both tables.

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

SQL INNER JOIN Example

```
SELECT Orders.OrderID, Customers.CustomerName
FROM Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
```

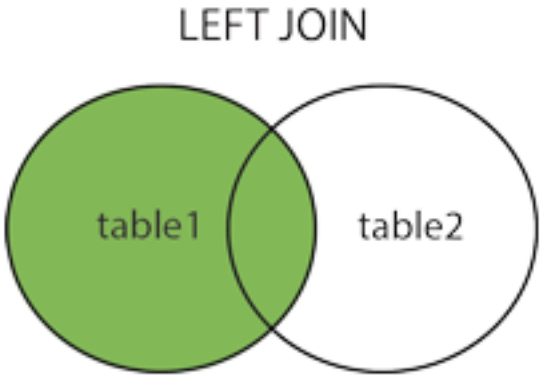
JOIN Three Tables

The following SQL statement SELECTs all orders with customer and shipper information:

Example

```
SELECT Orders.OrderID, Customers.CustomerName, Shippers.ShipperName
FROM ((Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID)
INNER JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID);
```

19.2.2 LEFT (OUTER) JOIN: Returns all records from the left table, and the matched records from the right table



The LEFT JOIN keyword returns all records from the left table (table1), and the matched records from the right table (table2). The result is NULL from the right side, if there is no match.

priorety is Table 1

LEFT JOIN Syntax:

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```

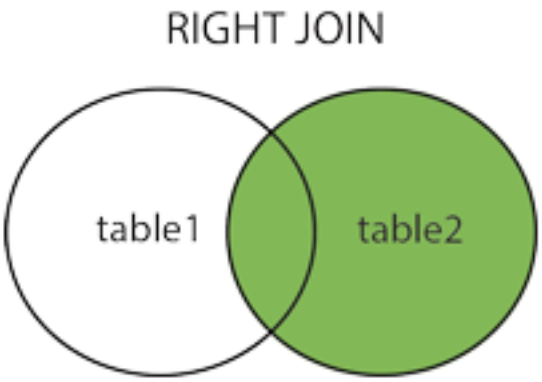
Example

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID
ORDER BY Customers.CustomerName;
```

Note: The LEFT JOIN keyword returns all records from the left table (Customers), even if there are no matches in the right table (Orders).

19.2.3 RIGHT (OUTER) JOIN: Returns all records from the right table, and the matched records from the left table

The RIGHT JOIN keyword returns all records from the right table (table2), and the matched records from the left table (table1). The result is NULL from the left side, when there is no match.



RIGHT JOIN Syntax:

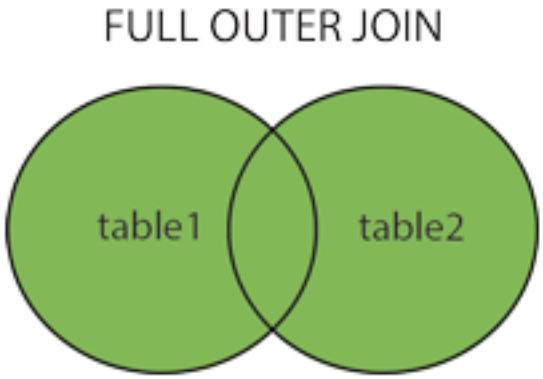
```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```

Note: In some databases RIGHT JOIN is called RIGHT OUTER JOIN.

Example

```
SELECT Orders.OrderID, Employees.LastName, Employees.FirstName
FROM Orders
RIGHT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
ORDER BY Orders.OrderID;
```

19.2.4 FULL (OUTER) JOIN: Returns all records when there is a match in either left or right table



The FULL OUTER JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records.

Note: FULL OUTER JOIN can potentially return very large result-sets!

Tip: FULL OUTER JOIN and FULL JOIN are the same.

FULL OUTER JOIN Syntax

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name
WHERE condition;
```

Example:

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
FULL OUTER JOIN Orders ON Customers.CustomerID=Orders.CustomerID
ORDER BY Customers.CustomerName;
```

Note: The FULL OUTER JOIN keyword returns all matching records from both tables whether the other table matches or not. So, if there are rows in "Customers" that do not have matches in "Orders", or if there are rows in "Orders" that do not have matches in "Customers", those rows will be listed as well.

20 SQL Self JOIN

A self JOIN is a regular join, but the table is joined with itself.

Self JOIN Syntax

```
SELECT column_name(s)
FROM table1 T1, table1 T2
WHERE condition;
```

T1 and T2 are different table aliases for the same table.

You use a self join when a table references data in itself.

[147]:
```
query('''
    SELECT *
    FROM northwind.customers as T1 ,  northwind.customers as T2
    WHERE T1.Last_name = T2.Last_name
    ORDER BY T1.job_title;
    ''').head()
```

[147]:
```
   id    company         last_name first_name email_address  \
0  19  Company S           Eggerer  Alexander          None
1  26  Company Z               Liu        Run          None
2  21  Company U              Tham    Bernard          None
```

```
3   1  Company A            Bedecs       Anna          None
4   2  Company B  Gratacos Solsona    Antonio          None

              job_title business_phone home_phone mobile_phone    ␣
↪fax_number  \
0  Accounting Assistant  (123)555-0100       None         None ␣
↪(123)555-0101
1  Accounting Assistant  (123)555-0100       None         None ␣
↪(123)555-0101
2    Accounting Manager  (123)555-0100       None         None ␣
↪(123)555-0101
3                 Owner  (123)555-0100       None         None ␣
↪(123)555-0101
4                 Owner  (123)555-0100       None         None ␣
↪(123)555-0101

   ... mobile_phone     fax_number          address         city  \
0  ...         None  (123)555-0101  789 19th Street  Los Angelas
1  ...         None  (123)555-0101  789 26th Street        Miami
2  ...         None  (123)555-0101  789 21th Street  Minneapolis
3  ...         None  (123)555-0101   123 1st Street      Seattle
4  ...         None  (123)555-0101   123 2nd Street       Boston

  state_province zip_postal_code country_region web_page  notes␣
↪attachments
0             CA           99999            USA     None   None
1             FL           99999            USA     None   None
2             MN           99999            USA     None   None
3             WA           99999            USA     None   None
4             MA           99999            USA     None   None

[5 rows x 36 columns]
```

21 SQL UNION Operator

The UNION operator is used to combine the result-set of two or more SELECT statements.

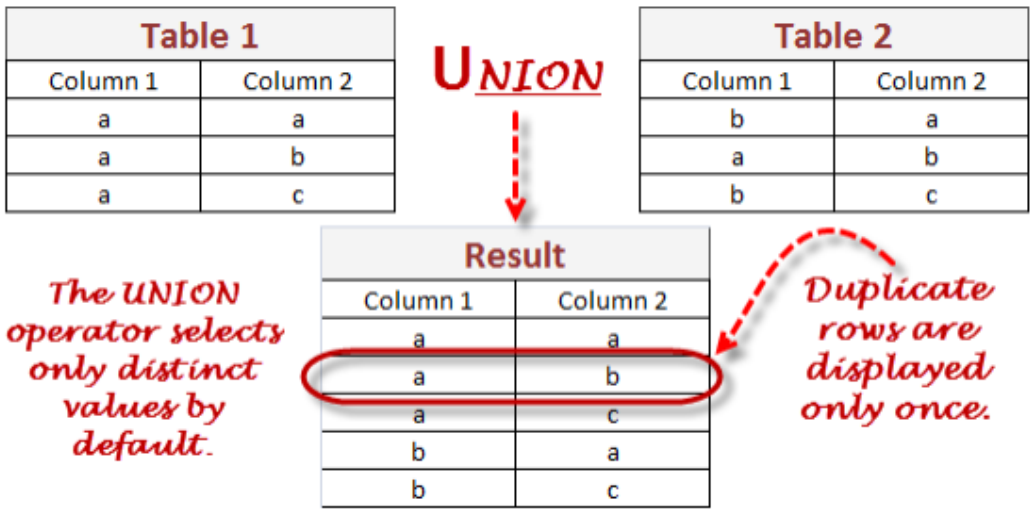
- Each SELECT statement within UNION must have the same number of columns
- The columns must also have similar data types
- The columns in each SELECT statement must also be in the same order

UNION Syntax:

```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;
```

UNION ALL Syntax The UNION operator selects only distinct values by default. To allow duplicate values, use UNION ALL:

```
SELECT column_name(s) FROM table1
UNION ALL
SELECT column_name(s) FROM table2;
```



[154]:
```
query('''
    SELECT first_name FROM northwind.customers
    UNION
    SELECT first_name FROM northwind.suppliers;
    ORDER BY first_name;
    ''').head()
```

[154]:
```
  first_name
0  Alexander
1  Amritansh
2      Andre
3       Anna
4    Antonio
```

[155]:
```
query('''
    SELECT first_name FROM northwind.customers
    UNION ALL
    SELECT first_name FROM northwind.suppliers;
    ORDER BY first_name;
    ''').head()
```

[155]:
```
  first_name
0  Alexander
```

```
1  Amritansh
2      Andre
3       Anna
4    Antonio
```

[164]:
```
# SQL UNION With WHERE
# only distinct values Alexander
query('''
    SELECT first_name FROM northwind.customers
    WHERE first_name='Parsa'
    UNION
    SELECT first_name FROM northwind.suppliers;
    WHERE first_name='Alexander'
    ORDER BY first_name;
    ''')
```

[164]:
```
      first_name
0          Parsa
1          Amaya
2      Bryn Paul
3       Cornelia
4   Elizabeth A.
5           Luis
6      Madeleine
7         Mikael
8          Naoki
9         Satomi
10        Stuart
```

22 The SQL GROUP BY Statement

The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

The GROUP BY statement is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the result-set by one or more columns.

GROUP BY Syntax

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s);
```

[171]:
```
query('''
    SELECT Avg(list_price) as COUNT_Price, product_name
```

```
    FROM products
    GROUP BY product_name
    order by product_name DESC;
    ''').head()
```

[171]:

```
   COUNT_Price                      product_name
0        23.25         Northwind Traders Walnuts
1         1.89  Northwind Traders Vegetable Soup
2         2.00       Northwind Traders Tuna Fish
3        17.00    Northwind Traders Tomato Sauce
4         4.00             Northwind Traders Tea
```

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country;


SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
ORDER BY COUNT(CustomerID) DESC;


SELECT Shippers.ShipperName, COUNT(Orders.OrderID) AS NumberOfOrders FROM Orders
LEFT JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID
GROUP BY ShipperName;
```

23 HAVING Syntax

The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```

[190]:

```
query('''
    SELECT Avg(list_price) as Avg_Price, product_name
    FROM products
    GROUP BY product_name
    having Avg_Price between 5 and 10
    order by product_name DESC;
```

```
    ''')
```

[190]:

	Avg_Price	product_name
0	10.00	Northwind Traders Syrup
1	10.00	Northwind Traders Scones
2	7.00	Northwind Traders Long Grain Rice
3	5.00	Northwind Traders Hot Cereal
4	9.65	Northwind Traders Clam Chowder
5	9.20	Northwind Traders Chocolate Biscuits Mix
6	10.00	Northwind Traders Almonds

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5;


SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5
ORDER BY COUNT(CustomerID) DESC;


SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders
FROM (Orders
INNER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID)
GROUP BY LastName
HAVING COUNT(Orders.OrderID) > 10;


SELECT Employees.LastName, COUNT(Orders.OrderID) AS NumberOfOrders
FROM Orders
INNER JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
WHERE LastName = 'Davolio' OR LastName = 'Fuller'
GROUP BY LastName
HAVING COUNT(Orders.OrderID) > 25;
```

24 The SQL EXISTS Operator

The EXISTS operator is used to test for the existence of any record in a subquery.

The EXISTS operator returns true if the subquery returns one or more records.

EXISTS Syntax

```
SELECT column_name(s)
FROM table_name
WHERE EXISTS
(SELECT column_name FROM table_name WHERE condition);
```

[203]:
```
query('''
    SELECT first_name
    FROM northwind.customers
    WHERE EXISTS
    (  SELECT first_name FROM northwind.suppliers
    WHERE first_name = 'Luis'
    );

    ''').head()
```

[203]:
```
  first_name
0  Alexander
1  Amritansh
2      Andre
3       Anna
4    Antonio
```

```
SELECT SupplierName
FROM Suppliers
WHERE EXISTS (SELECT ProductName FROM Products WHERE Products.SupplierID =
Suppliers.supplierID AND Price < 20);


FROM Suppliers
WHERE EXISTS (SELECT ProductName FROM Products WHERE Products.SupplierID =
Suppliers.supplierID AND Price = 22);
```

25 SQL ANY and ALL Operators

The ANY and ALL operators are used with a WHERE or HAVING clause.

The ANY operator returns true if any of the subquery values meet the condition.

The ALL operator returns true if all of the subquery values meet the condition.

25.1 Any Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator ANY
(SELECT column_name FROM table_name WHERE condition);
```

25.2 ALL Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator ALL
(SELECT column_name FROM table_name WHERE condition);
```

Note: The operator must be a standard comparison operator (=, <>, !=, >, >=, <, or <=).

```
SELECT ProductName
FROM Products
WHERE ProductID = ANY (SELECT ProductID FROM OrderDetails
WHERE Quantity = 10);

SELECT ProductName
FROM Products
WHERE ProductID = ANY (SELECT ProductID FROM OrderDetails
WHERE Quantity > 99);


SELECT ProductName
FROM Products
WHERE ProductID = ALL
(SELECT ProductID FROM OrderDetails WHERE Quantity = 10);
```

26 The SQL SELECT INTO Statement

The SELECT INTO statement copies data from one table into a new table.

SELECT INTO Syntax

Copy all columns into a new table:

```
SELECT *
INTO newtable [IN externaldb]
FROM oldtable
WHERE condition;

SELECT column1, column2, column3, ...
INTO newtable [IN externaldb]
FROM oldtable
WHERE condition;
```

SQL SELECT INTO Examples

```
SELECT * INTO CustomersBackup2017
FROM Customers;
```

The following SQL statement uses the IN clause to copy the table into a new table in another database:

```
SELECT * INTO CustomersBackup2017 IN 'Backup.mdb'
FROM Customers;
```

The following SQL statement copies only a few columns into a new table:

```
SELECT CustomerName, ContactName INTO CustomersBackup2017
FROM Customers;
```

The following SQL statement copies only the German customers into a new table:

```
SELECT * INTO CustomersGermany
FROM Customers
WHERE Country = 'Germany';
```

The following SQL statement copies data from more than one table into a new table:

```
SELECT Customers.CustomerName, Orders.OrderID
INTO CustomersOrderBackup2017
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID;
```

Tip: SELECT INTO can also be used to create a new, empty table using the schema of another. Just add a WHERE clause that causes the query to return no data:

```
SELECT * INTO newtable
FROM oldtable
WHERE 1 = 0;
```

27 The SQL INSERT INTO SELECT Statement

The INSERT INTO SELECT statement copies data from one table and inserts it into another table.

INSERT INTO SELECT requires that data types in source and target tables match

The existing records in the target table are unaffected

INSERT INTO SELECT Syntax

Copy all columns from one table to another table:

```
INSERT INTO table2
SELECT * FROM table1
WHERE condition;
```

Copy only some columns from one table into another table:

```
INSERT INTO table2 (column1, column2, column3, ...)
SELECT column1, column2, column3, ...
FROM table1
```

```
WHERE condition;
```

27.1 SQL INSERT INTO SELECT Examples

```
INSERT INTO Customers (CustomerName, City, Country)
SELECT SupplierName, City, Country FROM Suppliers;

INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Countr
SELECT SupplierName, ContactName, Address, City, PostalCode, Country FROM Suppliers

INSERT INTO Customers (CustomerName, City, Country)
SELECT SupplierName, City, Country FROM Suppliers
WHERE Country='Germany';
```

28 SQL CASE Statement

The CASE statement goes through conditions and returns a value when the first condition is met (like an IF-THEN-ELSE statement). So, once a condition is true, it will stop reading and return the result. If no conditions are true, it returns the value in the ELSE clause.

If there is no ELSE part and no conditions are true, it returns NULL.

```
CASE
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
    WHEN conditionN THEN resultN
    ELSE result
END;
```

28.1 Example

```
SELECT OrderID, Quantity,
CASE
    WHEN Quantity > 30 THEN 'The quantity is greater than 30'
    WHEN Quantity = 30 THEN 'The quantity is 30'
    ELSE 'The quantity is under 30'
END AS QuantityText
FROM OrderDetails;

-----------------


SELECT CustomerName, City, Country
FROM Customers
ORDER BY
```

```
(CASE
    WHEN City IS NULL THEN Country
    ELSE City
END);
```

[214]:
```
query('''
    SELECT order_id, quantity,
    CASE
        WHEN quantity > 30 THEN 'The quantity is greater than 30'
        WHEN quantity = 30 THEN 'The quantity is 30'
        ELSE 'The quantity is under 30'
    END AS quantityText
    FROM order_details
    WHERE quantity between 25 and 30 ;
    ''')
```

[214]:

	order_id	quantity	quantityText
0	30	30.0	The quantity is 30
1	33	30.0	The quantity is 30
2	44	25.0	The quantity is under 30
3	44	25.0	The quantity is under 30
4	44	25.0	The quantity is under 30
5	48	25.0	The quantity is under 30
6	48	25.0	The quantity is under 30
7	51	25.0	The quantity is under 30
8	51	30.0	The quantity is 30
9	51	30.0	The quantity is 30
10	79	30.0	The quantity is 30
11	79	30.0	The quantity is 30
12	76	30.0	The quantity is 30

[223]:
```
query('''
    SELECT *
    FROM order_details
    WHERE quantity between 35 and 40
    ORDER BY (
    CASE
        WHEN quantity > 30 THEN 'The quantity is greater than 30'
        ELSE 'The quantity is under 30'
    END ) ;
    ''')
```

[223]:

	id	order_id	product_id	quantity	unit_price	discount	status_id
0	70	78	17	40.0	39.00	0.0	2
1	73	75	48	40.0	12.75	0.0	2

```
2  74        74          48      40.0       12.75       0.0          2
3  77        71          40      40.0       18.40       0.0          2
4  81        60          72      40.0       34.80       0.0          2
5  84        58          20      40.0       81.00       0.0          2
6  85        58          52      40.0        7.00       0.0          2

  date_allocated purchase_order_id  inventory_id
0           None              None           120
1           None              None           123
2           None              None           124
3           None              None           127
4           None              None           131
5           None              None           134
6           None              None           135
```

29 SQL Stored Procedures for SQL Server

A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again.

So if you have an SQL query that you write over and over again, save it as a stored procedure, and then just call it to execute it.

You can also pass parameters to a stored procedure, so that the stored procedure can act based on the parameter value(s) that is passed.

Stored Procedure Syntax

```
CREATE PROCEDURE procedure_name
AS
sql_statement
GO;

# Execute a Stored Procedure
EXEC procedure_name;
```

29.1 An Example

```
CREATE PROCEDURE SelectAllCustomers
AS
SELECT * FROM Customers
GO;

EXEC SelectAllCustomers;
```

Part II

SQL Database

[ ]:
```
-- ################################################ DATABASE␣
↪OPERATIONS    ################################################
-- List all the databases present in mysql
-- SHOW DATABASES;

-- Create a database of name 'Health_industry'
CREATE DATABASE Health_industry;
-- SHOW DATABASES;

-- Create a database of name 'SEARCH_ENGINE'
CREATE DATABASE Search_engine;
-- SHOW DATABASES;

-- Drop a database
DROP DATABASE Health_industry;
-- SHOW DATABASES;


-- Use
USE Search_engine;



-- ################################################ TABLE OPERATIONS  ␣
↪################################################

-- 1. DONE: Create
-- 2. Alter (add extra column)
-- 3. Delete a table
-- 4. List all the tables present in a db
-- 5. Describe a table

-- Create table :: page
-- INFO::
-- Title       : String
-- Description : String
-- Price       : Int
-- Avg_stars   : Float
-- No_of_reviews: Int
-- Page_id      : String
CREATE TABLE Page(
    Title VARCHAR(200),
```

```
    Description VARCHAR(200),
    Price INT,
    Avg_stars FLOAT,
    No_of_reviews INT,
    Page_id VARCHAR(200)
);

-- SHOW TABLES;

-- create table:: page_model
-- INFO::
-- Page_id       : String
-- Category      : String
CREATE TABLE page_model(
    Page_id VARCHAR(55),
    Category VARCHAR(10)
);




-- create Table :: Review
-- INFO::
-- Review_id       : String
-- Review      : String
-- Reviewer_name : String
-- Review star   : Int
-- Review time   : DATETIME
-- Page_id       : String
CREATE TABLE Review(
    Review_id VARCHAR(200),
    Review VARCHAR(150),
    Reviewer_name VARCHAR(100),
    Review_star INT,
    Review_time DATETIME,
    Page_id VARCHAR(100)
);


-- Create table :: review_model
-- INFO::
-- Review_id     : String
-- Category     : String
-- Sentiment    : String
-- Attribute    : Array
CREATE TABLE review_model(
    Review_id VARCHAR(100),
```

```
    Category VARCHAR(50),
    Sentiment VARCHAR(10),
    Attribute VARCHAR(100)
);

-- SHOW TABLES;


-- ALTER A TABLE
-- Add new feature called 'language' to table 'Review_model'
ALTER TABLE review_model
ADD language VARCHAR(250);

-- DESCRIBE review_model;


-- delete a table 'Review_model'
DROP TABLE review_model;


-- List all the tables
-- SHOW TABLES;

-- print all the columns and types of a table "Review"
-- DESCRIBE Page;

-- Delete a particular column
-- ALTER TABLE Page
-- DROP Title;

-- DESCRIBE Page;


-- ################################################ DOCUMENT␣
↪OPERATIONS   ################################################

-- 1. Insert documents to table
-- 2. Show all the documents present in the table
-- 3. Deletes all the documents present in the table
-- 4. Update a document

-- insert document into table :: page
-- INFO::
-- Title       : String
-- Description : String
-- Price       : Int
-- Avg_stars   : Float
```

```sql
-- No_of_reviews: Int
-- Page_id      : String
INSERT INTO Page (Title, Description, Price, Avg_stars,␣
 ↪No_of_reviews, Page_id) VALUES
("iphone x", "32GB black", 999, 4.7, 3200, "Id_1"),
("Samsung s6", "64GB white", 777, 4.3, 1200, "Id_2");

-- SELECT
-- SELECT * FROM Page;


-- insert document into 'Page_model' table
-- INFO::
-- Page_id       : String
-- Category      : String

ALTER TABLE page_model
MODIFY Category TEXT;

INSERT INTO page_model (Page_id, Category) VALUES
("id_1", "Electronics"),
("id_2", "Electronics");




-- Show all the documents present in the table 'PAGE'
-- SELECT * FROM Page;

-- show all the documents present in the table 'PAGE' with Price␣
 ↪greater than 800
-- SELECT * FROM Page
-- WHERE Price>800;


-- delete all the documents present in the table 'PAGE'
-- DELETE FROM Page; -- delete all documents present in schema
-- DROP TABLE Page;  -- entirely drop the schema
-- SELECT * FROM Page;


-- delete documents present in the table 'PAGE' with Price greater␣
 ↪than 800
-- DELETE FROM Page WHERE Price>800;
```

```
-- Update the price of the document as 1000  if price is greater than␣
↪800
UPDATE Page SET Price=1000 WHERE Price>800;
-- SELECT * FROM Page;

-- ############################################## Constraints on␣
↪columns   ##############################################

-- 1. Not null constraint
-- 2. Default constraint
-- 3. Unique constraint
-- 4. Primary key constraint
-- 5. Foreign key constraint
-- 6. Check constraint

-- Create a table called 'page' according to following information␣
↪and specify constraint as mentioned
-- Create table :: page
-- INFO::
-- Title       : String   : Not Null
-- Description : String
-- Price       : Int      : Not Null
-- Avg_stars   : Float
-- No_of_reviews: Int
-- Page_id      : String   : Not Null
-- CREATE TABLE page(
--     Title VARCHAR(200) NOT NULL,
--     Description VARCHAR(200),
--     Price INT NOT NULL,
--     Avg_stars FLOAT,
--     No_of_reviews INT,
--     Page_id VARCHAR(200) NOT NULL
-- );
-- INSERT INTO page (Title, Description, Price, Avg_stars,␣
↪No_of_reviews, Page_id) VALUES
-- ("iphone x", "32 GB ram", 999, 4.5, 3200, "_id_1");

-- SELECT * FROM page;


-- Create a table called 'page' according to following information␣
↪and specify constraint as mentioned
-- Create table :: page
-- INFO::
-- Title       : String   : Not Null
-- Description : String
-- Price       : Int      : Not Null
```

```
-- Avg_stars    : Float     : Default value to be 0**
-- No_of_reviews: Int
-- Page_id      : String    : Not Null
-- CREATE TABLE page(
--     Title VARCHAR(200) NOT NULL,
--     Description VARCHAR(200),
--     Price INT NOT NULL,
--     Avg_stars FLOAT DEFAULT 0,
--     No_of_reviews INT,
--     Page_id VARCHAR(200) NOT NULL
-- );
-- INSERT INTO page (Title, Description, Price, Page_id) VALUES
-- ("iphone x", "32 GB ram", 999, "_id_1");

-- SELECT * FROM page;

-- Create a table called 'page' according to following information␣
 ↪and specify constraint as mentioned
-- Create table :: page
-- INFO::
-- Title       : String    : Not Null
-- Description : String
-- Price       : Int       : Not Null
-- Avg_stars    : Float     : Default value to be 0
-- No_of_reviews: Int
-- Page_id      : String    : Not Null & UNIQUE**
-- CREATE TABLE page(
--     Title VARCHAR(200) NOT NULL,
--     Description VARCHAR(200),
--     Price INT NOT NULL,
--     Avg_stars FLOAT DEFAULT 0,
--     No_of_reviews INT,
--     Page_id VARCHAR(200) NOT NULL UNIQUE
-- );
-- INSERT INTO page (Title, Description, Price, Page_id) VALUES
-- ("iphone x", "32 GB ram", 999, "_id_1"),
-- ("Samsung s6", "64 GB ram", 888, "_id_2");

-- SELECT * FROM page;


-- set primary key in the table : Page
-- Create table :: page
-- INFO::
-- Title       : String    : Not Null
-- Description : String
-- Price       : Int       : Not Null
```

```sql
-- Avg_stars   : Float    : Default value to be 0
-- No_of_reviews: Int
-- Page_id      : String   : PRIMARY KEY
CREATE TABLE page(
    Title VARCHAR(200) NOT NULL,
    Description VARCHAR(200),
    Price INT NOT NULL,
    Avg_stars FLOAT DEFAULT 0,
    No_of_reviews INT,
    Page_id VARCHAR(200) PRIMARY KEY
);
INSERT INTO page (Title, Description, Price, Page_id) VALUES
("iphone x", "32 GB ram", 999, "_id_1"),
("Samsung s6", "64 GB ram", 888, "_id_2");


-- set foreign key in the table page_model
CREATE TABLE Page_Model(
    Page_id VARCHAR(200),
    Category VARCHAR(200),
    FOREIGN KEY(Page_id) REFERENCES page(Page_id)
);

SELECT * FROM page;


-- check constraint
CREATE TABLE page(
    Title VARCHAR(200) NOT NULL,
    Description VARCHAR(200),
    Price INT NOT NULL,
    Avg_stars FLOAT DEFAULT 0 CHECK Avg_stars<=5,
    No_of_reviews INT,
    Page_id VARCHAR(200) PRIMARY KEY
);
```

Part III

Nirvana Class

There are 3 types of databases are present:

1. Relational databases: A relational database is a database that organises information into one or more tables consisting rows which represent an item and columns represents the properties of an item.

2. Graph databases

3. Non relational databases

How does mySQL database looks:

How does mySQL database looks:

1 Case study

2 Creating a database and operations on database

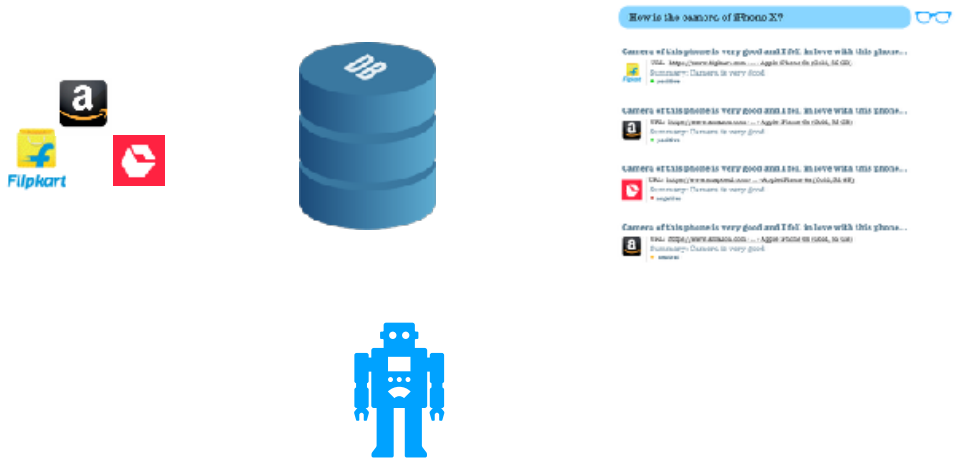
3 Creating tables and operations on tables

4 Create documents in tables and operations on documents

5 Constraints on columns

Case study

1 Case study



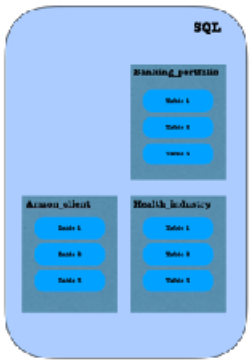
Let's say you want to build a review search engine. In order to build such search engine you need to collect reviews from different websites and store it. You might need to store the data for various reasons. For example-

- In order to collect reviews from different websites.
- In order to provide reviews when a user searches for some query.
- In order to store the results of models which will help you to provide search service.

How would you design a mysql storage architecture for your search engine.

DATABASE

2 Creating a database and operations on database




```
CREATE DATABASE Search_engine;
```

Operations with respect to database

Database commands

Create a database	`mysql> CREATE DATABASE db_name;`
Use a database	`mysql> USE DB_NAME;`
Delete a database	`mysql> SHOW DATABASES;`
List all databases	`mysql> DROP DATABASE db_name;`

TABLE

2 Creating tables and operations on tables

List all the things that we need to build the review search engine

1. We need to collect product data from different websites like amazon, Flipkart etc
2. We need to collect review data for individual pages from different websites.
3. We might need to run different models and store the results of those models.
   - A. We might need to run some models with respect to page data
   - B. We might need to run some models with respect to review data

page_data

Category model

Model output on page data

Review data

Category model
Sentiment model
Attribute model

Model output on review data

Let's gather some information about these tables

How we can describe each of these tables?

What is the type of each descriptor?

Page

title	String
description	String
Price	Int
avg_stars	Float
no_of_reviews	Int
Page_id	String

Page_models

Category model

Page_id	String
category	String

Review

Review_id	String
Review	String
reviewer_name	String
review_star	Int
review_time	Date
Page_id	String

Review_models

Category model
Sentiment model
Attribute model

Review_id	String
category	String
Sentiment	Int
Attributes	String

Let's create a table

Page

title	**String**
description	**String**
Price	**Int**
avg_stars	**Float**
no_of_reviews	**Int**
Page_id	**String**

Name of the table

```
CREATE TABLE Page(
    Title VARCHAR(255),
    Description
    VARCHAR(250),
    Price INT,
    Avg_stars Float,
    No_of_reviews Int,
    Page_id VARCHAR(250)
    );
```

Operations with respect to table

Table commands

- Create a table
- Alter a table

```
mysql> ALTER TABLE Review_model
       ADD language VARCHAR(250);
```

- Delete a table

```
mysql> DROP TABLE Review_model;
```

- List all tables

```
mysql> SHOW TABLES;
```

- List all columns

```
mysql> DESCRIBE TABLE_NAME;
```

3 Let's create some documents in tables

Name of the table

Name of the columns

```
mysql>

INSERT INTO PAGE (Title, Description, Price, Avg_stars, No_of_reviews, Page_id)
VALUES
      ('iphone X','iphone x with 64GB', 999, 4.5, 3000, 'id_1'),
      ('Samsing s6','s6 with 64GB', 777, 4.2, 2400, 'id_2');
```

Records

Operations with respect to documents

Database commands

- Insert a document
- Show documents
- Delete a document
- Update document

- Insert a document
- Show documents

```
mysql> SELECT * FROM PAGE;
```

- Delete a document

```
mysql> DELETE FROM PAGE;
mysql> DELETE FROM PAGE WHERE PRICE<50;
```

- Update a document

```
mysql> UPDATE PAGE SET PRICE=1000
       WHERE PRICE>900
```

SQL Commands to manipulate data

SQL manipulations

1. Data Definition Language(DDL):
   - CREATE
   - ALTER
   - DROP
2. Data Manipulation Langauge(DML)
   - INSERT
   - UPDATE
   - DELETE
3. Data Query Langauge(DQL)
   - SELECT
   - SHOW
   - DESCRIBE

5 Constraints on columns

We can specify certain constraints while creating SQL database

SQL constraints

- Not Null constraint → **The NOT NULL constraint enforces a column to NOT accept NULL values. This means that you cannot insert a new record, or update a record without adding a value to this field.**
- Default constraint → **The DEFAULT constraint is used to insert a default value into a column.The default value will be added to all new records, if no other value is specified.**
- Unique constraint → **The UNIQUE constraint uniquely identifies each record in a database table. The UNIQUE constraints both provide a guarantee for uniqueness for a column or set of columns.**
- Primary Key constraint → **The PRIMARY KEY constraint uniquely identifies each record in a database table. You can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table. A primary key column cannot contain NULL values.**
- Foreign Key constraint → **A FOREIGN KEY in one table points to a PRIMARY KEY in another table.**
- Check constraint → **The CHECK constraint is used to limit the value range that can be placed in a column. If you define a CHECK constraint on a single column it allows only certain values for this column.**

Summary

Database commands

- Create a database
- Use a database
- Delete a database
- List all databases

Document commands

- Insert a document
- Show documents
- Delete a document
- Update document

Table commands

- Create a database
- Use a database
- Delete a database
- List all databases

Constraints

- Not Null constraint
- Default constraint
- Unique constraint
- Primary Key constraint
- Foreign Key constraint
- Check constraint

1 Data Definition Language(DDL):

DDL commands

- Create a table
- Alter a table
- Drop a table

Create a table

- It is used to specify a new relation by giving it a name and specifying its attributes and initial constraints.
- The attributes are specified first, and each attribute is given a name, a data type to specify its domain of values, and any attribute constraints, such as NOT NULL.

Name of the table

```
mysql> CREATE TABLE COMPANY(
       Company varchar(255) NOT NULL,PRIMARY KEY(Company));
```

Name of the Attribute

Data type of Attribute

Constraints

1 Data Definition Language(DDL):

DDL commands

- Create a table
- Alter a table
- Drop a table

Alter a table

- We can add columns or make changes to table later also by altering the table using ALTER TABLE COMMAND

```
mysql> ALTER TABLE COMPANY ADD Est_year int;
```

Drop a table

```
mysql> drop table table_name;
```

2 Data Manipulation Langauge(DML)

DML commands

- ✓ Insert into a table
- Update a table
- Delete all data from table

Insert into a table

```
mysql>
        INSERT INTO COMPANY (Company, Est_year) VALUES
        ('Microsoft', 1975),
        ('Apple',1976),
        ('Amazon',1944),
        ('Alphabet', 2015),
        ('Facebook',2015),
        ('IBM',1911);
```

Name of the table

Name of the columns

Records

2 Data Manipulation Langauge(DML)

DML commands

- ✓ Insert into a table
- ✓ Update a table
- Delete all data from table

Update a table

Name of the table

Update values

```
mysql>
        UPDATE COMPANY
        SET Est_year = 1976, Company = 'Apple'
        WHERE Company = 'Appleee';
```

Condition

2 Data Manipulation Langauge(DML)

DML commands

- Insert into a table
- Update a table
- Delete all data from table

Delete all data from table

- Unlike drop delete will only delete all the entries present in the table but it will retain the table structure

```
mysql>

        Delete from table_name
```

3 Data Query Langauge(DQL)

DQL commands

- Select
- Show
- Describe

Select

- You can use select command to show the rows present in the table

```
mysql> select * from table_name;
```

- You can use also use select command along with 'where' to make conditional query

```
mysql> select * from table_name
       Where column_name = column_value;
```

- You can use also select particular column from a query result.

```
mysql> select column_name from table_name
       Where column_name = column_value;
```

3 Data Query Langauge(DQL)

DQL commands

- Select
- Show
- Describe

Show

- You can use show command to show all the tables present in a particular database

```
mysql> show tables;
```

Describe

- You can use describe command to show all the properties of a table present in the database

```
mysql> describe table_name;
```

constraints

SQL constraints

- Not Null constraint
- Default constraint
- Unique constraint
- Primary Key constraint
- Foreign Key constraint
- Check constraint

Not Null constraint

- If you want to create a table with a column where null value shouldn't be allowed then you can use not null constraint to create such columns

- Let' say we want there shouldn't be any null value present for company_name then we can create table as:

```
mysql> CREATE TABLE COMPANY(
       Company varchar(255) NOT NULL );
```

Default constraint

- If user inserts a document in a table and if he does not specify any value for a particular attribute or column then we can assign a default value for it.

- While creating a table you can specify a default value for a column

```
mysql> CREATE TABLE ORG(
       Name varchar(255),
       City varchar(250) Default 'EARTH' );
```

constraints

SQL constraints

- Not Null constraint
- Default constraint
- Unique constraint
- Primary Key constraint
- Foreign Key constraint
- Check constraint

Unique constraint

- The UNIQUE constraint ensures that all values in a column are different.
- Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns.
- A PRIMARY KEY constraint automatically has a UNIQUE constraint.
- However, you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

```
mysql> CREATE TABLE Persons(
       ID int NOT NULL UNIQUE,
       LastName varchar(255) NOT NULL,
       FirstName varchar(255),Age int);
```

constraints

SQL constraints

- Not Null constraint
- Default constraint
- Unique constraint
- Primary Key constraint
- Foreign Key constraint
- Check constraint

Primary Key constraint

- The PRIMARY KEY constraint uniquely identifies each record in a table.
- Primary keys must contain UNIQUE values, and cannot contain NULL values.
- A PRIMARY KEY constraint automatically has a UNIQUE constraint.
- A table can have only ONE primary key;

```
mysql> CREATE TABLE Persons(
       ID int NOT NULL UNIQUE,
       LastName varchar(255) NOT NULL,
       FirstName varchar(255),Age int,
       PRIMARY KEY (ID));
```

constraints

SQL constraints

- ✓ Not Null constraint
- ✓ Default constraint
- ✓ Unique constraint
- ✓ Primary Key constraint
- ✓ Foreign Key constraint
- Check constraint

Foreign Key constraint

- A FOREIGN KEY is a key used to link two tables together.
- A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table.
- The table containing the foreign key is called the child table, and the table containing the candidate key is called the referenced or parent table..

```
mysql> CREATE TABLE artists (
       id    INTEGER PRIMARY KEY,
       name  TEXT
     );
```

```
mysql> CREATE TABLE tracks (
  traid     INTEGER,
  title   TEXT,
  artist INTEGER,
  FOREIGN KEY(artist) REFERENCES artists(id)
);
```

constraints

SQL constraints

- Not Null constraint
- Default constraint
- Unique constraint
- Primary Key constraint
- Foreign Key constraint
- Check constraint

Check constraint

- The CHECK constraint is used to limit the value range that can be placed in a column.
- If you define a CHECK constraint on a single column it allows only certain values for this column.
- The table containing the foreign key is called the child table, and the table containing the candidate key is called the referenced or parent table..

```
mysql> CREATE TABLE Persons (
       ID int NOT NULL,
       LastName varchar(255) NOT NULL,
       FirstName varchar(255),
       Age int,
       CHECK (Age>=18));
```

SQL cheat sheet

For more awesome cheat sheets visit rebellabs.org!

REBELLABS by ZEROTURNAROUND

Basic Queries

```
-- filter your columns
   SELECT col1, col2, col3, ... FROM table1
-- filter the rows
   WHERE col4 = 1 AND col5 = 2
-- aggregate the data
   GROUP by ...
-- limit aggregated data
   HAVING count(*) > 1
-- order of the results
   ORDER BY col2
```

Useful keywords for **SELECTS**:

DISTINCT - return unique results
BETWEEN a **AND** b - limit the range, the values can be numbers, text, or dates
LIKE - pattern search within the column text
IN (a, b, c) - check if the value is contained among given.

Data Modification

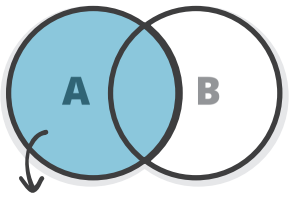
```
-- update specific data with the WHERE clause
   UPDATE table1 SET col1 = 1 WHERE col2 = 2
-- insert values manually
   INSERT INTO table1 (ID, FIRST_NAME, LAST_NAME)
      VALUES (1, 'Rebel', 'Labs');
-- or by using the results of a query
   INSERT INTO table1 (ID, FIRST_NAME, LAST_NAME)
      SELECT id, last_name, first_name FROM table2
```

Views

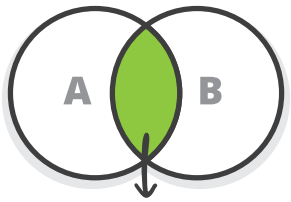
A **VIEW** is a virtual table, which is a result of a query.
They can be used to create virtual tables of complex queries.

```
CREATE VIEW view1 AS
SELECT col1, col2
FROM table1
WHERE ...
```

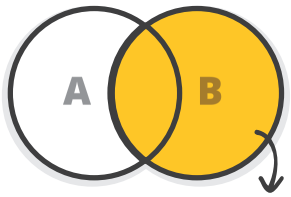
The Joy of JOINs



LEFT OUTER JOIN - *all rows from table A, even if they do not exist in table B*



INNER JOIN - *fetch the results that exist in both tables*



RIGHT OUTER JOIN - *all rows from table B, even if they do not exist in table A*

Updates on JOINed Queries

You can use **JOIN**s in your **UPDATE**s

```
UPDATE t1 SET a = 1
FROM table1 t1 JOIN table2 t2 ON t1.id = t2.t1_id
WHERE t1.col1 = 0 AND t2.col2 IS NULL;
```

NB! Use database specific syntax, it might be faster!

Semi JOINs

You can use subqueries instead of **JOIN**s:

```
SELECT col1, col2 FROM table1 WHERE id IN
   (SELECT t1_id FROM table2 WHERE date >
      CURRENT_TIMESTAMP)
```

Indexes

If you query by a column, index it!
CREATE INDEX index1 **ON** table1 (col1)

Don't forget:
Avoid overlapping indexes
Avoid indexing on too many columns
Indexes can speed up **DELETE** and **UPDATE** operations

Useful Utility Functions

```
-- convert strings to dates:
   TO_DATE (Oracle, PostgreSQL), STR_TO_DATE (MySQL)
-- return the first non-NULL argument:
   COALESCE (col1, col2, "default value")
-- return current time:
   CURRENT_TIMESTAMP
-- compute set operations on two result sets
   SELECT col1, col2 FROM table1
   UNION / EXCEPT / INTERSECT
   SELECT col3, col4 FROM table2;
```

Union - returns data from both queries
Except - rows from the first query that are not present in the second query
Intersect - rows that are returned from both queries

Reporting

Use aggregation functions

COUNT - return the number of rows
SUM - cumulate the values
AVG - return the average for the group
MIN / MAX - smallest / largest value

BROUGHT TO YOU BY XRebel