Connecting to MySQL database

In [1]:
```
# Load and activate the SQL extension
%load_ext sql
```

In [2]:
```
# Establish a connection to the local database using the '%sql' magic command.
# Replace 'password' with our connection password and `db_name` with our database r
# If you get an error here, please make sure the database name or password is corre

%sql mysql+pymysql://username:password@localhost:3306/md_water_services
```

Connecting to 'mysql+pymysql://root:***@localhost:3306/md_water_services'

Cleaning our data

In [4]:
```
%%sql
SELECT *
FROM employee
LIMIT 10;
```

Running query in 'mysql+pymysql://root:***@localhost:3306/md_water_services'

10 rows affected.

Out[4]:

	assigned_employee_id	employee_name	phone_number	email	address	province_name	town_n
	0	Amara Jengo	+99637993287	None	36 Pwani Mchangani Road	Sokoto	Ila
	1	Bello Azibo	+99643864786	None	129 Ziwa La Kioo Road	Kilimani	F
	2	Bakari Iniko	+99222599041	None	18 Mlima Tazama Avenue	Hawassa	F
	3	Malachi Mavuso	+99945849900	None	100 Mogadishu Road	Akatsi	Lu
	4	Cheche Buhle	+99381679640	None	1 Savanna Street	Akatsi	F
	5	Zuriel Matembo	+99034075111	None	26 Bahari Ya Faraja Road	Kilimani	F
	6	Deka Osumare	+99379364631	None	104 Kenyatta Street	Akatsi	F
	7	Lalitha Kaburi	+99681623240	None	145 Sungura Amanpour Road	Kilimani	F
	8	Enitan Zuri	+99248509202	None	117 Kampala Road	Hawassa	Zanz
	10	Farai Nia	+99570082739	None	33 Angélique Kidjo Avenue	Amanzi	Dah

In [10]:

```
%%sql
/* putting email addresses for all workers.
Here, I assume that the email format is "firstname.lastname@ndogowater.gov"
*/
SELECT assigned_employee_id, employee_name, CONCAT(LOWER(REPLACE(employee_name,' '
FROM employee
LIMIT 10;
```

Running query in 'mysql+pymysql://root:***@localhost:3306/md_water_services'

10 rows affected.

Out[10]:

assigned_employee_id	employee_name	new_email
0	Amara Jengo	amara.jengo@ndogowater.gov
1	Bello Azibo	bello.azibo@ndogowater.gov
2	Bakari Iniko	bakari.iniko@ndogowater.gov
3	Malachi Mavuso	malachi.mavuso@ndogowater.gov
4	Cheche Buhle	cheche.buhle@ndogowater.gov
5	Zuriel Matembo	zuriel.matembo@ndogowater.gov
6	Deka Osumare	deka.osumare@ndogowater.gov
7	Lalitha Kaburi	lalitha.kaburi@ndogowater.gov
8	Enitan Zuri	enitan.zuri@ndogowater.gov
10	Farai Nia	farai.nia@ndogowater.gov

Truncated to displaylimit of 10.

In [16]:
```
%%sql
-- update the database with these email addresses
UPDATE employee
SET email=CONCAT(LOWER(REPLACE(employee_name,' ','.')),'@ndogowater.gov');
```

Running query in 'mysql+pymysql://root:***@localhost:3306/md_water_services'

56 rows affected.

Out[16]:

In [111…
```
%%sql
SELECT *
FROM employee
WHERE assigned_employee_id = 20 OR assigned_employee_id = 22
LIMIT 10;
```

Running query in 'mysql+pymysql://root:***@localhost:3306/md_water_services'

2 rows affected.

Out[111]:

assigned_employee_id	employee_name	phone_number	email	address	provi
20	Kunto Asha	+99176320477	kunto.asha@ndogowater.gov	30 Nyoka Achebe Street	
22	Lesedi Kofi	+99611183730	lesedi.kofi@ndogowater.gov	52 Moroni Avenue	

I picked up another bit we have to clean up. Often when databases are created and updated, or information is collected from different sources, errors creep in. For example, if you look at the phone numbers in the phone_number column, the values are stored as strings.

The phone numbers should be 12 characters long, consisting of the plus sign, area code (99), and the phone number digits. However, when we use the LENGTH(column) function, it returns 13 characters, indicating there's an extra character.

In [18]:
```
%%sql
SELECT
LENGTH(phone_number)
FROM
employee;
```

Running query in 'mysql+pymysql://root:***@localhost:3306/md_water_services'

56 rows affected.

Out[18]:

LENGTH(phone_number)
13
13
13
13
13
13
13
13
13
13

Truncated to displaylimit of 10.

So, I trim the column

In [20]:
```
%%sql
SELECT
LENGTH(TRIM(phone_number)) AS LENGTH
FROM
employee;

UPDATE employee
SET phone_number = TRIM(phone_number);

SELECT
LENGTH(TRIM(phone_number)) AS LENGTH
FROM
employee;
```

Running query in 'mysql+pymysql://root:***@localhost:3306/md_water_services'

56 rows affected.

56 rows affected.

56 rows affected.

Out[20]:

LENGTH
12
12
12
12
12
12
12
12
12
12

Truncated to displaylimit of 10.

In [3]:
```
%%sql


SHOW TABLES;
```

Running query in 'mysql+pymysql://root:***@localhost:3306/md_water_services'

8 rows affected.

Out[3]:

Tables_in_md_water_services
data_dictionary
employee
global_water_access
location
visits
water_quality
water_source
well_pollution

NOTES: This query will show all of the tables in a database. When you start a data project, this is a good first command to use to get to know the database.

Now, using the employees table to count how many employees live in each town.

In [112…
```
%%sql
SELECT town_name, COUNT(assigned_employee_id) AS number_of_employees
FROM employee
WHERE province_name = 'Kilimani'
GROUP BY town_name;
```

Running query in 'mysql+pymysql://root:***@localhost:3306/md_water_services'

3 rows affected.

Out[112]:

town_name	number_of_employees
Rural	9
Ilanga	1
Harare	2

Honouring the workers

Say the president of this hometown asks to send out an email or message congratulating the top 3 field surveryors. Get the employee_ids and use that to get the contact details of field surveryors with the most locations visited.

In [30]:
```
%%sql
SELECT assigned_employee_id, count(location_id) AS number_of_visits
FROM visits
GROUP BY assigned_employee_id
ORDER BY number_of_visits DESC
LIMIT 3;
```

Running query in 'mysql+pymysql://root:***@localhost:3306/md_water_services'

3 rows affected.

Out[30]:

assigned_employee_id	number_of_visits
1	3708
30	3676
34	3539

In [109…
```
%%sql
SELECT assigned_employee_id, count(location_id) AS number_of_visits
FROM visits
GROUP BY assigned_employee_id
ORDER BY number_of_visits
LIMIT 2;
```

Running query in 'mysql+pymysql://root:***@localhost:3306/md_water_services'

2 rows affected.

Out[109]:

assigned_employee_id	number_of_visits
20	15
22	143

Analysing locations

Looking at the location table, let's focus on the province_name, town_name and location_type to understand where the water sources are

In [34]:
```
%%sql
-- counting the number of records per town
SELECT COUNT(*) AS records_per_town, town_name
FROM location
```

```
GROUP BY town_name
ORDER BY records_per_town DESC
LIMIT 6;
```

Running query in 'mysql+pymysql://root:***@localhost:3306/md_water_services'

6 rows affected.

Out[34]:

records_per_town	town_name
23740	Rural
1650	Harare
1090	Amina
1070	Lusaka
990	Mrembo
930	Asmara

In [35]:

```
%%sql
-- counting the number of records per province
SELECT COUNT(*) AS records_per_province, province_name
FROM location
GROUP BY province_name
ORDER BY records_per_province DESC
LIMIT 6;
```

Running query in 'mysql+pymysql://root:***@localhost:3306/md_water_services'

5 rows affected.

Out[35]:

records_per_province	province_name
9510	Kilimani
8940	Akatsi
8220	Sokoto
6950	Amanzi
6030	Hawassa

1. Create a result set showing: • province_name • town_name • An aggregated count of records for each town (consider naming this records_per_town). • Ensure data is grouped by both province_name and town_name.
2. Order results primarily by province_name. Within each province, further sort the towns by their record counts in descending order.

In [45]:

```
%%sql
SELECT province_name, town_name, COUNT(*) AS records_per_town
FROM location
GROUP BY province_name, town_name
ORDER BY province_name, records_per_town desc
```

Running query in 'mysql+pymysql://root:***@localhost:3306/md_water_services'

31 rows affected.

Out[45]:

province_name	town_name	records_per_town
Akatsi	Rural	6290
Akatsi	Lusaka	1070
Akatsi	Harare	800
Akatsi	Kintampo	780
Amanzi	Rural	3100
Amanzi	Asmara	930
Amanzi	Dahabu	930
Amanzi	Amina	670
Amanzi	Pwani	520
Amanzi	Abidjan	400

Truncated to displaylimit of 10.

Look at number of records for each location type.

In [47]:
```
%%sql
SELECT COUNT(*) AS num_sources, location_type
FROM location
GROUP BY location_type
;
```

Running query in 'mysql+pymysql://root:***@localhost:3306/md_water_services'

2 rows affected.

Out[47]:

num_sources	location_type
15910	Urban
23740	Rural

In [48]:
```
%%sql
-- checking what percentage of water sources are in rural communities
SELECT 23740/(15910+23740)*100
```

Running query in 'mysql+pymysql://root:***@localhost:3306/md_water_services'

1 rows affected.

Out[48]:

23740/(15910+23740)*100
59.8739

Diving into the sources

Ok, water_source is a big table, with lots of stories to tell

Before I go and spoil it all, I open up the table, look at the various columns, make some notes on what we can do with them, and go ahead and make some queries and explore the dataset. Perhaps there's more to tell. 11:04 The way I look at this table; we have access to different water source types and the number of people using each source. These are the questions that I am curious about.

1. How many people did we survey in total?
2. How many wells, taps and rivers are there?
3. How many people share particular types of water sources on average?
4. How many people are getting water from each type of source?

In [49]:
```
%%sql
-- exploring the table
SELECT *
FROM water_source
LIMIT 1;
```

Running query in 'mysql+pymysql://root:***@localhost:3306/md_water_services'

1 rows affected.

Out[49]:

source_id	type_of_water_source	number_of_people_served
AkHa00000224	tap_in_home	956

In [56]:
```
%%sql
-- How many people did we survey in total?
SELECT SUM(number_of_people_served) AS number_of_people_surveyed
FROM water_source
```

Running query in 'mysql+pymysql://root:***@localhost:3306/md_water_services'

1 rows affected.

Out[56]:

number_of_people_surveyed
27628140

In [52]:
```
%%sql
-- How many wells, taps and rivers are there?
SELECT type_of_water_source, COUNT(*) AS count
FROM water_source
GROUP BY type_of_water_source
```

Running query in 'mysql+pymysql://root:***@localhost:3306/md_water_services'

5 rows affected.

Out[52]:

type_of_water_source	count
tap_in_home	7265
tap_in_home_broken	5856
well	17383
shared_tap	5767
river	3379

In [55]:
```
%%sql
-- How many people share particular types of water sources on average?
SELECT type_of_water_source, ROUND(AVG(number_of_people_served)) AS avg_people_shar
FROM water_source
GROUP BY type_of_water_source
```

Running query in 'mysql+pymysql://root:***@localhost:3306/md_water_services'

5 rows affected.

Out[55]:

type_of_water_source	avg_people_sharing
tap_in_home	644
tap_in_home_broken	649
well	279
shared_tap	2071
river	699

In [54]:
```
%%sql
--  How many people are getting water from each type of source?
SELECT type_of_water_source, SUM(number_of_people_served) AS num_people_getting
FROM water_source
GROUP BY type_of_water_source
```

Running query in 'mysql+pymysql://root:***@localhost:3306/md_water_services'

5 rows affected.

Out[54]:

type_of_water_source	num_people_getting
tap_in_home	4678880
tap_in_home_broken	3799720
well	4841724
shared_tap	11945272
river	2362544

In [59]:
```
%%sql
--  How many people are getting water from each type of source?
SELECT type_of_water_source, ROUND((SUM(number_of_people_served)/27628140)*100) AS
FROM water_source
GROUP BY type_of_water_source
ORDER BY pcnt_people_getting DESC
```

Running query in 'mysql+pymysql://root:***@localhost:3306/md_water_services'

5 rows affected.

Out[59]:

type_of_water_source	pcnt_people_getting
shared_tap	43
well	18
tap_in_home	17
tap_in_home_broken	14
river	9

NOTES: This query fetches 10 records from the `location` table. This is the results set I got, with only 10 rows.

Start of a solution

At some point, we will have to fix or improve all of the infrastructure, so we should start thinking about how we can make a data-driven decision how to do it. I think a simple

approach is to fix the things that affect most people first. So let's write a query that ranks each type of source based on how many people in total use it. 'rank' should tell you we are going to need a window function to do this, so let's think through the problem.

We will need the following columns:

- Type of sources -- Easy
- Total people served grouped by the types -- We did that earlier, so that's easy too.
- A rank based on the total people served, grouped by the types -- A little harder.

In [75]:
```
%%sql
--  How many people are getting water from each type of source?
SELECT type_of_water_source, SUM(number_of_people_served) AS population_served, RAN
FROM water_source
GROUP BY type_of_water_source
```

Running query in 'mysql+pymysql://root:***@localhost:3306/md_water_services'
5 rows affected.

Out[75]:

type_of_water_source	population_served	rank_of_water_source
shared_tap	11945272	1
well	4841724	2
tap_in_home	4678880	3
tap_in_home_broken	3799720	4
river	2362544	5

Ok, so I should fix shared taps first, then wells, and so on. But the next question is, which shared taps or wells should be fixed first? I can use the same logic; the most used sources should really be fixed first.

I definitely should keep these requirements in mind:

1. The sources within each type should be assigned a rank.
2. Limit the results to only improvable sources.
3. Think about how to partition, filter and order the results set.
4. Order the results to see the top of the list.

In [92]:
```
%%sql
SELECT
    source_id,
    type_of_water_source,
    Number_of_people_served,
    RANK() OVER (PARTITION BY type_of_water_source ORDER BY Number_of_people_served
FROM
    water_source
WHERE
    type_of_water_source IN ('river','tap_in_home_broken', 'well', 'shared_tap')
ORDER BY
    source_rank ASC;
```

Running query in 'mysql+pymysql://root:***@localhost:3306/md_water_services'
32385 rows affected.

Out[92]:

source_id	type_of_water_source	Number_of_people_served	source_rank
SoRu36122224	well	398	1
SoRu36201224	well	398	1
SoRu35723224	well	398	1
SoRu36436224	well	398	1
SoRu36899224	well	398	1
SoRu36142224	well	398	1
SoRu36355224	well	398	1
SoRu34665224	well	398	1
SoRu35277224	well	398	1
SoRu36774224	well	398	1

Truncated to displaylimit of 10.

Analysing Queues

Ok, this is the really big, and last table we'll look at this time. The analysis is going to be a bit tough, but the results will be worth it, so stretch out, grab a drink, and let's go!

Ok, these are some of the things I think are worth looking at:

1. How long did the survey take?
2. What is the average total queue time for water?
3. What is the average queue time on different days?
4. How can we communicate this information efficiently?

Question 1: To calculate how long the survey took, I'll get the first and last dates (which functions can find the largest/smallest value), and subtract them. Remember with DateTime data, we can't just subtract the values. So I'll use a function to get the difference in days.

In [95]:
```
%%sql
SELECT *
FROM visits
LIMIT 1;
```

Running query in 'mysql+pymysql://root:***@localhost:3306/md_water_services'

1 rows affected.

Out[95]:

record_id	location_id	source_id	time_of_record	visit_count	time_in_queue	assigned_employee
0	Soll32582	Soll32582224	2021-01-01 09:10:00	1	15	

In [100…
```
%%sql
SELECT
    MIN(time_of_record) AS start_date,
    MAX(time_of_record) AS end_date,
    DATEDIFF(MAX(time_of_record), MIN(time_of_record)) AS survey_duration_in_days
```

```
FROM
    visits;
```

Running query in 'mysql+pymysql://root:***@localhost:3306/md_water_services'
1 rows affected.

Out[100]:

start_date	end_date	survey_duration_in_days
2021-01-01 09:10:00	2023-07-14 13:53:00	924

Question 2: Let's see how long people have to queue on average in Maji Ndogo.

In [101…

```
%%sql
SELECT AVG(NULLIF(time_in_queue,0)) AS avg_queue_time
FROM visits;
```

Running query in 'mysql+pymysql://root:***@localhost:3306/md_water_services'
1 rows affected.

Out[101]:

avg_queue_time
123.2574

Question 3: Time to look at the queue times aggregated across the different days of the week.

In [104…

```
%%sql
SELECT DAYNAME(time_of_record) AS day_of_week, ROUND(AVG(NULLIF(time_in_queue,0)))
FROM visits
GROUP BY day_of_week
```

Running query in 'mysql+pymysql://root:***@localhost:3306/md_water_services'
7 rows affected.

Out[104]:

day_of_week	avg_queue_time
Friday	120
Saturday	246
Sunday	82
Monday	137
Tuesday	108
Wednesday	97
Thursday	105

In [107…

```
%%sql
SELECT
    HOUR(time_of_record) AS hour_of_day,
    ROUND(AVG(NULLIF(time_in_queue, 0))) AS avg_queue_time
FROM
    visits
GROUP BY
    hour_of_day
ORDER BY avg_queue_time DESC;
```

Running query in 'mysql+pymysql://root:***@localhost:3306/md_water_services'
14 rows affected.

Out[107]:

hour_of_day	avg_queue_time
19	168
7	149
8	149
17	149
6	149
18	147
9	118
13	115
10	114
14	114

Truncated to displaylimit of 10.

The hour number is difficult to interpret. A format like 06:00 will be easier to read, so let's use that

Question 4: We can also look at what time during the day people collect water. Try to order the results in a meaningful way.

In [108…

```
%%sql
SELECT
    TIME_FORMAT(TIME(time_of_record), '%H:00') AS hour_of_day,
    ROUND(AVG(NULLIF(time_in_queue, 0))) AS avg_queue_time
FROM
    visits
GROUP BY
    hour_of_day
ORDER BY avg_queue_time DESC;
```

Running query in 'mysql+pymysql://root:***@localhost:3306/md_water_services'
14 rows affected.

Out[108]:

hour_of_day	avg_queue_time
19:00	168
07:00	149
08:00	149
17:00	149
06:00	149
18:00	147
09:00	118
13:00	115
10:00	114
14:00	114

Truncated to displaylimit of 10.