EXPLORE AI ACADEMY

Maji Ndogo: From analysis to action

Beginning Our Data-Driven Journey in Maji Ndogo

Aziza Naledi
Online

Dear Team,

Congratulations on the successful completion of our extensive survey. Your dedication and hard work have resulted in an invaluable asset - a database of 60,000 records, meticulously collected by our devoted team of engineers, field workers, scientists, and analysts.

Now, the next crucial phase of our mission begins. We need to make sense of this immense data trove and extract meaningful insights. We must breathe life into these records and listen to the story they are telling us.

I urge you to load this database and thoroughly acquaint yourselves with it. Dive deep, explore its structure, understand the variables and the connections between them. Each record is a chapter of our story; each query you run is a thread weaving that story together. This is a process of discovery - to uncover the patterns and nuances in our data. It's a chance to ask the right questions, to identify the pressing problems, and to set the course for our data-driven solutions.

As you proceed, always remember that every bit of information is a piece of the bigger puzzle. Every insight, no matter how small, brings us one step closer to solving our water crisis. Together, we have embarked on this journey to bring about change. Let's continue to march ahead with the same determination and resolve.

I believe in our collective potential. Together, we will unravel the secrets held within these records, and use this knowledge to shape a brighter future for Maji Ndogo.

Best regards,
Aziza Naledi

06:26

Chidi Kunto
Online

Hey Team,

Hope this message finds you well. I'm Chidi Kunto, and I'm really excited to be working with all of you on this upcoming project.

Our president, Aziza Naledi, has given us a monumental task. We've got this mountain of data, and it's our job to sift through it, finding those nuggets of insight that will help us solve our water crisis here in Maji Ndogo.

But you won't be doing this alone. As your mentor, I'll be here not just to guide you but also to show you firsthand how a real data project kicks off. I'll be taking Naledi's instructions and breaking them down into clear, manageable tasks. And of course, I'll be sharing a few valuable tricks of the trade along the way.

I'll help you navigate through this data landscape, turning those raw numbers into meaningful insights. Just remember, every data point can be part of our solution, and every step you take in this project helps us get closer to our goal.

I'm looking forward to working closely with each of you, learning about your unique perspectives, and figuring out how we can best collaborate as a team. With your skills and dedication, I'm confident we can unlock the potential of this data and make a real difference to the people of Maji Ndogo.

Please know that my virtual door is always open. If you need some guidance, want to clarify something, or just have something cool to share, feel free to reach out.

Here's to an exciting journey ahead and the valuable lessons we'll learn together!

Cheers,
Chidi

07:54

Chidi Kunto
Online

Hey there,

You've probably seen President Naledi's message by now. She has emphasised the importance of our newly collected survey data and how vital it is for us to dive in and start making sense of it. As the senior data analyst, I've taken a close look at her message and have broken it down into a series of tasks that we need to tackle. So, let's roll up our sleeves and get started!

08:33

1. Get to know our data: Before we do anything else, let's take a good look at our data. We'll load up the database and pull up the first few records from each table. It's like getting to know a new city - we need to explore the lay of the land before we can start our journey.

08:37

2. Dive into the water sources: We've got a whole table dedicated to the types of water sources in our database. Let's dig into it and figure out all the unique types of water sources we're dealing with.

08:42

3. Unpack the visits to water sources: The 'visits' table in our database is like a logbook of all the trips made to different water sources. We need to unravel this logbook to understand the frequency and distribution of these visits. Let's identify which locations have been visited more than a certain number of times.

08:43

Chidi Kunto
Online

4. Assess the quality of water sources: The quality of water sources is a pretty big deal. We'll turn to the `water_quality` table to find records where the `subjective_quality_score` is within a certain range and the `visit_count` is above a certain threshold. This should help us spot the water sources that are frequently visited and have a decent quality score.

08:50

5. Investigate any pollution issues: We can't overlook the pollution status of our water sources. Let's find those water sources where the pollution_tests result came back as 'dirty' or 'biologically contaminated'. This will help us flag the areas that need immediate attention.

08:57

By working through these tasks, we'll not only be answering President Naledi's call to explore the database and extract meaningful insights, but we'll also be honing our SQL skills. It's a win-win situation! So, are you ready to dive in and start exploring with me?

09:04

Let's do this!

09:05

1. Get to know our data:
Start by retrieving the first few records from each table. How many tables are there in our database? What are the names of these tables? Once you've identified the tables, write a SELECT statement to retrieve the first five records from each table. As you look at the data, take note of the columns and their respective data types in each table. What information does each table contain?

09:06

I don't think they showed you this at the Academy, but when you access a new database in MySQL, a handy initial query to run is `SHOW TABLES`. This will give you a list of all the tables in the database.

09:12

You should see something like this:

Tables_in_md_water_services_student
data_dictionary
employee
global_water_access
location
water_quality
visits
water_source
well_pollution

09:16

It looks like someone took the time to name all of these tables pretty well because we can kind of figure out what each table is about without really having to think too hard. `water_source` probably logs information about each source like where it is, what type of source it is and so on.

09:19

So let's have a look at one of these tables, Let's use `location` so we can use that killer query, `SELECT *` but remember to limit it and tell it which table we are looking at.

09:21

You should get something like this:

location_id	address	province_name	town_name	location_type
AkHa00000	2 Addis Ababa Road	Akatsi	Harare	Urban
AkHa00001	10 Addis Ababa Road	Akatsi	Harare	Urban
AkHa00002	9 Addis Ababa Road	Akatsi	Harare	Urban
AkHa00003	139 Addis Ababa Road	Akatsi	Harare	Urban
AkHa00004	17 Addis Ababa Road	Akatsi	Harare	Urban

09:26

So we can see that this table has information on a specific location, with an address, the province and town the location is in, and if it's in a city (`Urban`) or not. We can't really see what location this is but we can see some sort of identifying number of that location.

09:28

Ok, so let's look at the `visits` table.

09:31

Here's what I get:

record_id	location_id	source_id	time_of_record	visit_count	time_in_queue	assigned_employee_id
0	SoIl32582	SoIl32582224	2021-01-01 09:10:00	1	15	12
1	KiRu28935	KiRu28935224	2021-01-01 09:17:00	1	0	46
2	HaRu19752	HaRu19752224	2021-01-01 09:36:00	1	62	40
3	AkLu01628	AkLu01628224	2021-01-01 09:53:00	1	0	1
4	AkRu03357	AkRu03357224	2021-01-01 10:11:00	1	28	14

09:37

Yeah, so this is a list of `location_id`, `source_id`, `record_id`, and a date and time, so it makes sense that someone (`assigned_employee_id`) visited some location (`location_id`) at some time (`time_of_record`) and found a 'source' there (`source_id`). Often the "_id" columns are related to another table. In this case, the `source_id` in the `visits` table refers to `source_id` in the `water_source` table. This is what we call a foreign key, but we'll get more into this next time.

09:40

Ok, so let's look at the `water_source` table to see what a 'source' is. Normally "_id" columns are related to another table.

08:47

I get:

source_id	type_of_water_source	number_of_people_served
AkHa00000224	tap_in_home	956
AkHa00001224	tap_in_home_broken	930
AkHa00002224	tap_in_home_broken	486
AkHa00003224	clean_well	364
AkHa00004224	tap_in_home_broken	94

08:48

Nice! Ok, we're getting somewhere now... Water sources are where people get their water from! Ok, this database is actually complex, so maybe a good idea for you is to look at the rest of the tables quickly. You can just select them, but remember in good SQL there would be a data dictionary somewhere that documents all of this information, so you should read that as well, and even keep a copy of that close if we need to find information quickly.

09:54

A data dictionary has been embedded into the database. If you query the `data_dictionary table`, an explanation of each column is given there.

09:54

2. Dive into the water sources:
Now that you're familiar with the structure of the tables, let's dive deeper. We need to understand the types of water sources we're dealing with. Can you figure out which table contains this information?

10:00

Once you've identified the right table, write a SQL query to find all the **unique** types of water sources.

10:00

So I get this when I run it:

type_of_water_source
tap_in_home
tap_in_home_broken
well
shared_tap
river

10:06

Let me quickly bring you up to speed on these water source types:

10:09

1. River - People collect drinking water along a river. This is an open water source that millions of people use in Maji Ndogo. Water from a river has a high risk of being contaminated with biological and other pollutants, so it is the worst source of water possible.

10:15

This is a river in the province of Sokoto:

10:16

2. Well - These sources draw water from underground sources, and are commonly shared by communities. Since these are closed water sources, contamination is much less likely compared to a river. Unfortunately, due to the aging infrastructure and the corruption of officials in the past, many of our wells are not clean.

10:21

This well is at 146 Okapi Road, in my home town of Yaounde:

10:22

3. Shared tap - This is a tap in a public area shared by communities.

10:27

This is a shared tap from 18 Twiga Lane, Hawassa, that serves about 2700 people:

10:28

4. Tap in home - These are taps that are inside the homes of our citizens. On average about 6 people live together in Maji Ndogo, so each of these taps serves about 6 people.

10:28

This is a tap in my uncle's home in the capital city, Dahabu:

10:29

5. Broken tap in home - These are taps that have been installed in a citizen's home, but the infrastructure connected to that tap is not functional. This can be due to burst pipes, broken pumps or water treatment plants that are not working.

10:35

This is a water treatment plant in the town of Kintampo that serves about 1000 people:

10:36

An important note on the home taps: About 6-10 million people have running water installed in their homes in Maji Ndogo, including broken taps. If we were to document this, we would have a row of data for each home, so that one record is one tap. That means our database would contain about 1 million rows of data, which may slow our systems down. For now, the surveyors combined the data of many households together into a single record.

10:37

For example, the first record, `AkHa00000224` is for a `tap_in_home` that serves 956 people. What this means is that the records of about 160 homes nearby were combined into one record, with an average of 6 people living in each house `160 x 6 ≈ 956`. So 1 `tap_in_home` or `tap_in_home_broken` record actually refers to multiple households, with the sum of the people living in these homes equal to `number_of_people_served`.

10:39

3. Unpack the visits to water sources:
We have a table in our database that logs the visits made to different water sources. Can you identify this table?

10:44

Write an SQL query that retrieves all records from this table where the `time_in_queue` is more than some crazy time, say 500 min. How would it feel to queue 8 hours for water?

10:47

This is the table I get:

record_id	location_id	source_id	time_of_record	visit_count	time_in_queue	assigned_employee_id
899	SoRu35083	SoRu35083224	2021-01-16 10:14:00	6	515	28
2304	SoKo33124	SoKo33124224	2021-02-06 07:53:00	5	512	16
2315	KiRu26095	KiRu26095224	2021-02-06 14:32:00	3	529	8
3206	SoRu38776	SoRu38776224	2021-02-20 15:03:00	5	509	46
3701	HaRu19601	HaRu19601224	2021-02-27 12:53:00	3	504	0
4154	SoRu38869	SoRu38869224	2021-03-06 10:44:00	2	533	24
5483	AmRu14089	AmRu14089224	2021-03-27 18:15:00	4	509	12
9177	SoRu37635	SoRu37635224	2021-05-22 18:48:00	2	515	1
9648	SoRu36096	SoRu36096224	2021-05-29 11:24:00	2	533	3
11631	AkKi00881	AkKi00881224	2021-06-26 06:15:00	6	502	32

10:54

How is this possible? Can you imagine queueing 8 hours for water?

10:57

I am wondering what type of water sources take this long to queue for. We will have to find that information in another table that lists the types of water sources. If I remember correctly, the table has `type_of_water_source`, and a `source_id` column. So let's write down a couple of these `source_id` values from our results, and search for them in the other table.
`AkKi00881224`
`SoRu37635224`
`SoRu36096224`
If we just select the first couple of records of the `visits` table without a `WHERE` filter, we can see that some of these rows also have 0 mins queue time. So let's write down one or two of these too.

10:58

I chose these two:
`AkRu05234224`
`HaZa21742224`
Ok, so now back to the `water_source` table. Let's check the records for those `source_ids`. You can probably remember there is a cool and a "not so cool" way to do it.

11:05

This is what I get.

source_id	type_of_water_source	number_of_people_served
AkKi00881224	shared_tap	3398
AkLu01628224	bio_dirty_well	210
AkRu05234224	tap_in_home_broken	496
HaRu19601224	shared_tap	3322
HaZa21742224	pol_dirty_well	308
SoRu36096224	shared_tap	3786
SoRu37635224	shared_tap	3920
SoRu38776224	shared_tap	3180

11:09

I added a couple of others... Sorry! Well, if you check them you will see which sources have people queueing. The field surveyors also let us know that they measured sources that had queues a few times to see if the queue time changed.

11:11

Chidi Kunto
Online

4. Assess the quality of water sources:
The quality of our water sources is the whole point of this survey. We have a table that contains a quality score for each visit made about a water source that was assigned by a Field surveyor. They assigned a score to each source from 1, being terrible, to 10 for a good, clean water source in a home. Shared taps are not rated as high, and the score also depends on how long the queue times are.

11:14

Look through the table record to find the table.

11:17

Let's check if this is true. The surveyors only made multiple visits to shared taps and did not revisit other types of water sources. So there should be no records of second visits to locations where there are good water sources, like taps in homes.

11:20

So please write a query to find records where the `subject_quality_score` is 10 -- only looking for home taps -- and where the source was visited a second time. What will this tell us?

11:22

I get 218 rows of data. But this should not be happening! I think some of our employees may have made mistakes. To be honest, I'll be surprised if there are no errors in our data at this scale! I'm going to send Pres. Naledi a message that we have to recheck some of these sources. We can appoint an Auditor to check some of the data independently, and make sure we have the right information!

11:23

5. Investigate pollution issues:
Did you notice that we recorded contamination/pollution data for all of the well sources? Find the right table and print the first few rows.

11:24

Find the right table and print the first few rows.

11:27

I get this:

source_id	date	description	pollutant_ppm	biological	results
KiRu28935224	2021-01-04 09:17:00	Bacteria: Giardia Lamblia	0.0	495.898	Contaminated: Biological
AkLu01628224	2021-01-04 09:53:00	Bacteria: Salmonella Typhi	0.0	376.572	Contaminated: Biological
HaZa21742224	2021-01-04 10:37:00	Inorganic contaminants: Zinc...	2.715	0.0	Contaminated: Chemical
HaRu19725224	2021-01-04 11:04:00	Clean	0.0288593	0.0	Clean
SoRu35703224	2021-01-04 11:29:00	Bacteria: E. coli	0.0	296.437	Contaminated: Biological

11:30

It looks like our scientists diligently recorded the water quality of all the wells. Some are contaminated with biological contaminants, while others are polluted with an excess of heavy metals and other pollutants. Based on the results, each well was classified as `Clean`, `Contaminated: Biological` or `Contaminated: Chemical`. It is important to know this because wells that are polluted with bio- or other contaminants are not safe to drink. It looks like they recorded the `source_id` of each test, so we can link it to a source, at some place in Maji Ndogo.

11:36

In the well pollution table, the descriptions are notes taken by our scientists as text, so it will be challenging to process it. The `biological` column is in units of `CFU/mL`, so it measures how much contamination is in the water. `0` is clean, and anything more than `0.01` is contaminated.
Let's check the integrity of the data. The worst case is if we have contamination, but we think we don't. People can get sick, so we need to make sure there are no errors here.

11:37

So, write a query that checks if the `results` is `Clean` but the `biological` column is `> 0.01`.

11:43

I got this:

source_id	date	description	pollutant_ppm	biological	results
AkRu08936224	2021-01-08 09:22:00	Bacteria: E. coli	0.0406458	35.0068	Clean
AkRu06489224	2021-01-10 09:44:00	Clean Bacteria: Giardia Lamblia	0.0897904	38.467	Clean
SoRu38011224	2021-01-14 15:35:00	Bacteria: E. coli	0.0425095	19.2897	Clean
AkKi00955224	2021-01-22 12:47:00	Bacteria: E. coli	0.0812092	40.2273	Clean
KiHa22929224	2021-02-06 13:54:00	Bacteria: E. coli	0.0722537	18.4482	Clean
KiRu25473224	2021-02-07 15:51:00	Clean Bacteria: Giardia Lamblia	0.0630094	24.4536	Clean
HaRu17401224	2021-03-01 13:44:00	Clean Bacteria: Giardia Lamblia	0.0649209	25.8129	Clean

11:45

If we compare the results of this query to the entire table it seems like we have some inconsistencies in how the well statuses are recorded. Specifically, it seems that some data input personnel might have mistaken the `description` field for determining the cleanliness of the water.

11:46

Chidi Kunto
Online

It seems like, in some cases, if the description field begins with the word “Clean”, the results have been classified as “Clean” in the `results` column, even though the `biological` column is > 0.01.

11:50

When we work with real-world data we may find inconsistencies due to data being misinterpreted based on a description rather than its actual values. Let’s dive deeper into the cause of the issue with the `biological` contamination data.

11:52

Vuyisile has told me that the descriptions should only have the word “Clean” if there is no biological contamination (and no chemical pollutants). Some data personnel must have copied the data from the scientist's notes into our database incorrectly. We need to find and remove the “Clean” part from all the descriptions that do have a biological contamination so this mistake is not made again.

11:59

The second issue has arisen from this error, but it is much more problematic. Some of the field surveyors have marked wells as `Clean` in the `results` column because the description had the word “Clean” in it, even though they have a biological contamination. So we need to find all the results that have a value greater than 0.01 in the `biological` column and have been set to `Clean` in the `results` column.

12:06

First, let's look at the descriptions. We need to identify the records that mistakenly have the word `Clean` in the description. However, it is important to remember that not all of our field surveyors used the description to set the results – some checked the actual data.

12:09

Chidi Kunto
Online

Hint: To find these descriptions, search for the word `Clean` with additional characters after it. As this is what separates incorrect descriptions from the records that should have "Clean".

12:14

The query should return 38 wrong descriptions.

12:20

Now we need to fix these descriptions so that we don't encounter this issue again in the future.

12:21

Looking at the results we can see two different descriptions that we need to fix:
1. All records that mistakenly have `Clean Bacteria: E. coli` should updated to `Bacteria: E. coli`
2. All records that mistakenly have `Clean Bacteria: Giardia Lamblia` should updated to `Bacteria: Giardia Lamblia`

12:26

The second issue we need to fix is in our `results` column. We need to update the `results` column from `Clean` to `Contaminated: Biological` where the `biological` column has a value greater than 0.01.

12:31

Chidi Kunto
Online

Ok, so here is how I did it:

```
-- Case 1a: Update descriptions that mistakenly mention
`Clean Bacteria: E. coli` to `Bacteria: E. coli`
-- Case 1b: Update the descriptions that mistakenly mention
`Clean Bacteria: Giardia Lamblia` to `Bacteria: Giardia Lamblia
-- Case 2: Update the `result` to `Contaminated: Biological` where
`biological`  is greater than 0.01 plus current results is `Clean`
```

12:43

Before we make these changes, here is another nugget of advice: Begin complex queries by commenting on what you will do. This helps us to think through the problem before we write code to solve it, and once we're done, we have a well-documented code.

12:38

Then add how we would do it:

```
-- Case 1a
UPDATE
   -- Update well_pollution table
SET
   -- Change description to'Bacteria: E. coli'
WHERE
   -- Where the description is `Clean Bacteria: E. coli`

-- Case 1b
   -- Try to fill this in
-- Case 2
   -- Try to fill this in
```

12:46

Chidi Kunto
Online

And then fill in some details:

```
-- Case 1a
UPDATE
   well_pollution
SET
   description = 'Bacteria: E. coli'
WHERE
   description = 'Clean Bacteria: E. coli';

-- Case 1b
   -- Try to fill this in
-- Case 2`
   -- Try to fill this in
```

12:49

Ok, go ahead and fill in the rest.

12:53

Now, when we change any data on the database, we need to be SURE there are no errors, as this could fill the database with incorrect values. A safer way to do the UPDATE is by testing the changes on a copy of the table first.

12:54

The `CREATE TABLE new_table AS (query)` approach is a neat trick that allows you to create a new table from the results set of a query. This method is especially useful for creating backup tables or subsets without the need for a separate `CREATE TABLE` and `INSERT INTO` statement.

12:56

Chidi Kunto
Online

So if we run:

```
CREATE TABLE
    md_water_services.well_pollution_copy
AS (
     SELECT
         *
     FROM
         md_water_services.well_pollution
     );
```

13:02

We will get a copy of `well_pollution` called `well_pollution_copy`. Now we can make the changes, and if we discover there is a mistake in our code, we can just delete this table, and run it again.

13:06

Chidi Kunto
Online

So if we now run our query:

```
UPDATE
   well_pollution_copy
SET
   description = 'Bacteria: E. coli'
WHERE
   description = 'Clean Bacteria: E. coli';

UPDATE
   well_pollution_copy
SET
   description = 'Bacteria: Giardia Lamblia'
WHERE
   description = 'Clean Bacteria: Giardia Lamblia';

UPDATE
   well_pollution_copy
SET
   results = 'Contaminated: Biological'
WHERE
   biological > 0.01 AND results = 'Clean';

-- Put a test query here to make sure we fixed the errors.
-- Use the query we used to show all of the erroneous rows
```

13 : 11

We can then check if our errors are fixed using a SELECT query on the `well_pollution_copy` table:

```
SELECT
    *
FROM
    well_pollution_copy
WHERE
    description LIKE "Clean_%"
   OR (results = "Clean" AND biological > 0.01);
```

13 : 13

Then if we're sure it works as intended, we can change the table back to the `well_pollution` and delete the `well_pollution_copy` table.

```
UPDATE
    well_pollution_copy
SET
    description = 'Bacteria: E. coli'
WHERE
    description = 'Clean Bacteria: E. coli';

UPDATE
    well_pollution_copy
SET
    description = 'Bacteria: Giardia Lamblia'
WHERE
    description = 'Clean Bacteria: Giardia Lamblia';

UPDATE
    well_pollution_copy
SET
    results = 'Contaminated: Biological'
WHERE
    biological > 0.01 AND results = 'Clean';

DROP TABLE
    md_water_services.well_pollution_copy;
```

13 : 19

So I hope that today helped you to see what the survey data is all about, I hope that you are as excited as I am to get stuck in! Until then, keep well!

13 : 33

EXPLORE AI ACADEMY

Maji Ndogo: From analysis to action

Clustering data to unveil Maji Ndogo's water crisis

Aziza Naledi
Online

Dear Team,

Our mission, as arduous as it is essential, requires us to delve deeper into our reservoir of data. To truly illuminate the road ahead, we must magnify our analysis, moving beyond isolated data points to discern larger patterns and trends.
In this next step, we will cluster our data, stepping back from the individual figures to gain a panoramic understanding. This bird's eye view will allow us to unearth broader narratives and hidden correlations concealed within our rich dataset.
Next, we must pay heed to the different forms of data in our possession. They are not mere numbers or dates; they are stories waiting to be deciphered. Their unique structure, though challenging, brims with valuable insights. As we process these, we unlock deeper layers of understanding.
Bear in mind that every piece of information you decipher, every category you determine, brings us one stride closer to our noble goal. It's through the intricate details and broader brushstrokes of data that we will uncover the solutions to Maji Ndogo's water crisis.
Your unwavering commitment to this mission emboldens me. Together, we continue marching forward, using data and dedication as our compass, towards a brighter, more secure future for Maji Ndogo.

Thank you for all your tireless efforts.

Warm regards,
Aziza Naledi

06:11

Chidi Kunto
Online

Hi Pres. Naledi,

I hope you're doing well. While diving into our recent survey data for the Maji Ndogo water project, our team stumbled upon some inconsistencies that caught our eye. It's nothing alarming, but we think it's worth a closer look.
Would you consider bringing in an independent auditor to double-check some of the records? I think it's a smart move to ensure everything is on the up-and-up. After all, we're all about accuracy and trust.

Feel free to reach out if you want to chat more about this or need more details.

Take care,
Chidi Kunto

06:45

Aziza Naledi
Online

Hi Chidi,

Thanks for catching that, and for being so attentive to detail. I'm right there with you on this - we want to be sure we're working with the best information possible.
I'll get an independent auditor on this ASAP. They'll touch base with you and the rest of the team to get things rolling. I've cc'ed everyone so that we're all on the same page.

Appreciate your diligence, Chidi. Let's keep up the great work. Maji Ndogo is counting on us.

All the best,
Aziza Naledi

07:04

Chidi Kunto
Online

Before we start, scan through the data dictionary, and perhaps query a couple of tables to get a feel for the database again.

08:31

Cleaning our data

Ok, bring up the `employee` table. It has info on all of our workers, but note that the email addresses have not been added. We will have to send them reports and figures, so let's update it. Luckily the emails for our department are easy: `first_name.last_name@ndogowater.gov`.

08:31

I am going to guide you through this one, so code along.

08:31

We can determine the email address for each employee by:
- selecting the `employee_name` column
- replacing the space with a full stop
- make it lowercase
- and stitch it all together

08:38

We have to update the database again with these email addresses, so before we do, let's use a SELECT query to get the format right, then use UPDATE and SET to make the changes.

08:41

First up, let's remove the space between the first and last names using `REPLACE()`. You can try this:

```
SELECT
    REPLACE(employee_name, ' ','.') -- Replace the space with a full stop
FROM
    employee
```

08:45

Then we can use `LOWER()` with the result we just got. Now the name part is correct.

```
SELECT
    LOWER(REPLACE(employee_name, ' ','.')) -- Make it all lower case
FROM
    employee
```

08:47

We then use `CONCAT()` to add the rest of the email address:

```
SELECT
    CONCAT(
    LOWER(REPLACE(employee_name, ' ', '.')), '@ndogowater.gov') AS new_email -- add it all together
FROM
    employee
```

08:49

Quick win! Since you have done this before, you can go ahead and UPDATE the `email` column this time with the email addresses. Just make sure to check if it worked!

```
UPDATE employee
SET email = CONCAT(LOWER(REPLACE(employee_name, ' ', '.')),
            '@ndogowater.gov')
```

08:50

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.

08:53

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.

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

08:54

That's because there is a space at the end of the number! If you try to send an automated SMS to that number it will fail. This happens so often that they create a function, especially for trimming off the space, called `TRIM(column)`.

It removes any leading or trailing spaces from a string.

08:56

Chidi Kunto
Online

Use `TRIM()` to write a `SELECT` query again, make sure we get the string without the space, and then `UPDATE` the record like you just did for the emails. If you need more information about `TRIM()`, Google "TRIM documentation MySQL".

08:57

Honouring the workers

Before we dive into the analysis, let's get you warmed up a bit!
Let's have a look at where our employees live.

09:06

Use the `employee` table to count how many of our employees live in each town. Think carefully about what function we should use and how we should aggregate the data.

09:07

Chidi Kunto
Online

I get this:

town_name	num_employees
Ilanga	3
Rural	29
Lusaka	4
Zanzibar	4
Dahabu	6
Kintampo	1
Harare	5
Yaounde	1
...	...

09:08

Note how many of our workers are living in smaller communities in the rural parts of Maji Ndogo.

09:18

Pres. Naledi congratulated the team for completing the survey, but we would not have this data were it not for our field workers. So let's gather some data on their performance in this process, so we can thank those who really put all their effort in.

09:19

Pres. Naledi has asked we send out an email or message congratulating the top 3 field surveyors. So let's use the database to get the `employee_ids` and use those to get the names, email and phone numbers of the three field surveyors with the most location visits.

09:20

Chidi Kunto
Online

Let's first look at the number of records each employee collected. So find the correct table, figure out what function to use and how to group, order and limit the results to only see the top 3 `employee_ids` with the highest number of locations visited.

09:22

You should get a table like this but in a **different** order:

assigned_employee_id	number_of_visits
0	1099
1	3708
2	2033

09:29

Make a note of the top 3 `assigned_employee_id` and use them to create a query that looks up the employee's info. Since you're a pro at finding stuff in a database now, you can figure this one out. You should have a column of names, email addresses and phone numbers for our top dogs.

09:41

I'll send that off to Pres. Naledi. But this survey is not primarily about our employees, so let's get working on the main task! We'll start looking at some of the tables in the dataset at a larger scale, identify some trends, summarise important data, and draw insights.

09:49

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 Maji Ndogo.

09:52

Create a query that counts the number of records per town

09:56

For example, if we count the number of records for each **town**, I get:

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

10:02

Now count the records per province.

10:06

My results:

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

10:09

From this table, it's pretty clear that most of the water sources in the survey are situated in small rural communities, scattered across Maji Ndogo.

If we count the records for each province, most of them have a similar number of sources, so every province is well-represented in the survey.

10:12

Can you find a way to do the following:

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 your data is grouped by both **`province_name`** and **`town_name`**.
2. Order your results primarily by **`province_name`**. Within each province, further sort the towns by their record counts in descending order.

10:23

Chidi Kunto
Online

Your table should look something like this:

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

10:27

These results show us that our field surveyors did an excellent job of documenting the status of our country's water crisis. Every province and town has many documented sources.

This makes me confident that the data we have is reliable enough to base our decisions on. This is an insight we can use to communicate data integrity, so let's make a note of that.

10:33

Finally, look at the number of records for each location type

10:35

Chidi Kunto
Online

Like this:

num_sources	location_type
15910	Urban
23740	Rural

10:37

We can see that there are more rural sources than urban, but it's really hard to understand those numbers. Percentages are more relatable. If we use SQL as a very overpowered calculator:

```
SELECT 23740 / (15910 + 23740) * 100
```

We can see that 60% of all water sources in the data set are in rural communities.

10:44

So again, what are some of the insights we gained from the `location` table?

1. Our entire country was properly canvassed, and our dataset represents the situation on the ground.
2. 60% of our water sources are in rural communities across Maji Ndogo. We need to keep this in mind when we make decisions.

10:55

Chidi Kunto
Online

Diving into the sources

Ok, `water_source` is a big table, with lots of stories to tell, so strap in!

11:02

Before I go and spoil it all, 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 you see something I don't.

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?

11:10

I'll leave the first one to you. Try answering the rest on your own too.

11:15

For the second question, we want to count how many of each of the different water source types there are, and remember to sort them.

11:17

I get something like this:

type_of_water_source	number_of_sources
tap_in_home	7265
tap_in_home_broken	5856
...	...

11:18

Which of those sources stands out? It is pretty clear that although there was a drought, water is still abundant in Maji Ndogo. This isn't just an informative result, we will need these numbers to understand how much all of these repairs will cost. If we know how many taps we need to install, and we know how much it will cost to install them, we can calculate how much it will cost to solve the water crisis.

11:25

Ok next up, question 3: What is the average number of people that are served by each water source? Remember to make the numbers easy to read.

11:30

Chidi Kunto
Online

I got:

type_of_water_source	ave_people_per_source
tap_in_home	644
tap_in_home_broken	649
well	279
shared_tap	2071
river	699
...	...

11:32

These results are telling us that 644 people share a `tap_in_home` on average. Does that make sense?

11:40

No it doesn't, right?
Remember I told you a few important things that apply to `tap_in_home` and `broken_tap_in_home`? The surveyors combined the data of many households together and added this as a single tap record, but each household actually has its own tap. In addition to this, there is an average of 6 people living in a home. So 6 people actually share 1 tap (not 644).

11:42

Chidi Kunto
Online

It is always important to think about data. We tend to just analyse, and calculate at the start of our careers, but the value we bring as data practitioners is in understanding the meaning of results or numbers, and interpreting their meaning.

Imagine we were presenting this to the President and all of the Ministers, and one of them asks us: "Why does it say that 644 share a home tap?" and we had no answer.

It happened to me once... I don't miss those days.

11:45

This means that 1 `tap_in_home` actually represents 644 ÷ 6 = ± 100 taps.

11:47

Calculating the average number of people served by a single instance of each water source type helps us understand the typical capacity or load on a single water source. This can help us decide which sources should be repaired or upgraded, based on the average impact of each upgrade. For example, wells don't seem to be a problem, as fewer people are sharing them.

11:48

On the other hand, 2000 share a single public tap on average! We saw some of the queue times last time, and now we can see why. So looking at these results, we probably should focus on improving shared taps first.

11:49

Now let's calculate the total number of people served by each type of water source in total, to make it easier to interpret, order them so the most people served by a source is at the top.

11:50

Chidi Kunto
Online

This is what I got:

type_of_water_source	population_served
shared_tap	11945272
well	4841724
tap_in_home	4678880
tap_in_home_broken	3799720
river	2362544
...	...

11:53

It's a little hard to comprehend these numbers, but you can see that one of these is dominating. To make it a bit simpler to interpret, let's use percentages. First, we need the total number of citizens then use the result of that and divide each of the `SUM(number_of_people_served)` by that number, times 100, to get percentages.

11:56

Make a note of the number of people surveyed in the first question we answered. I get a total of about 27 million citizens!

11:57

Next, calculate the percentages using the total we just got.

12:05

Mine is:

type_of_water_source	percentage_people_per_source
shared_tap	43.2359
well	17.5246
tap_in_home	16.9352
tap_in_home_broken	13.7531
river	8.5512
...	...

12:07

Having percentages with a bunch of decimals really doesn't help get the point across, does it?

12:09

Let's round that off to 0 decimals, and order the results.

12:11

Chidi Kunto
Online

Yeah, this looks better, right?

type_of_water_source	percentage_people_per_source
shared_tap	43
well	18
tap_in_home	17
tap_in_home_broken	14
river	9
...	...

12:13

43% of our people are using shared taps in their communities, and on average, we saw earlier, that 2000 people share one `shared_tap`.

12:18

By adding `tap_in_home` and `tap_in_home_broken` together, we see that 31% of people have water infrastructure installed in their homes, but 45% (14/31) of these taps are not working! This isn't the tap itself that is broken, but rather the infrastructure like treatment plants, reservoirs, pipes, and pumps that serve these homes that are broken.
18% of people are using wells. But only 4916 out of 17383 are clean = 28% (from last week).

12:23

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.

12:26

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.

12:30

Let's look at the results from the last query again:

type_of_water_source	population_served
shared_tap	11945272
well	4841724
tap_in_home	4678880
tap_in_home_broken	3799720
river	2362544
...	...

12:36

It should be clear for you to see what the rank is, but we want to let SQL do it.

12:37

But think about this: If someone has a tap in their home, they already have the best source available. Since we can't do anything more to improve this, we should remove `tap_in_home` from the ranking before we continue.

12:38

So use a window function on the total people served column, converting it into a rank.

12:41

I get something like this:

type_of_water_source	people_served	rank_by_population
shared_tap	111945272	1
well	4841724	2
...	...	...

12:44

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

12:53

So create a query to do this, and 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.

12:57

This is what I got using `RANK`:

source_id	type_of_water_source	number_of_people_served	priority_rank
...	...	...	...
AmRu14978224	river	400	3364
HaDj16848224	river	400	3364
HaRu19509224	shared_tap	3998	1
AkRu05603224	shared_tap	3998	1
...	...	...	...

12:58

By using `RANK()` teams doing the repairs can use the value of rank to measure how many they have fixed, but what would be the benefits of using `DENSE_RANK()`?

Maybe it is easier to explain to the engineers this way, or the priority feels a bit more natural?

13:02

Chidi Kunto
Online

What about `ROW_NUMBER()`? Since each source now has a unique rank, teams don't have to think whether they should repair AkRu05603224, or AkRu04862224 first (both serve 3998 people), because `ROW_NUMBER()` doesn't consider records that are equal.

13:03

Try the different ranking functions in queries. Imagine yourself in an engineer's boots, and try to interpret the priority list. Thinking about the user of a table helps us to design the table better.
In that line of thought, would it make sense to give them a list of source_ids? How would they know where to go?

13:05

Chidi Kunto
Online

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!

13:11

A recap from last time:
The `visits` table documented all of the visits our field surveyors made to each location. For most sources, one visit was enough, but if there were queues, they visited the location a couple of times to get a good idea of the time it took for people to queue for water. So we have the time that they collected the data, how many times the site was visited, and how long people had to queue for water.

13:15

So, look at the information we have available, and think of what we could learn from it. Remember we can use some DateTime functions here to get some deeper insight into the water queueing situation in Maji Ndogo, like which day of the week it was, and what time.

13:17

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?

13:18

Try to answer some of these questions. Think of the data you will need to answer each question, and how to transform that data into the right form to answer each question. Then make some queries to try and answer them. You learned all of the skills you need, so give it a try.

13:22

HINT: I had to read up a bit on control flow, DateTime and window functions to do these, so you probably will have to as well.

13:27

Look at `visits`, especially the `time_of_record` column. It is an SQL DateTime datatype, so we can use all of the DateTime functions to aggregate data for each day and even per hour.

13:29

Question 1:

To calculate how long the survey took, we need to 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. We have to use a function to get the difference in days.

13:36

When I do it, I get 924 days which is about 2 and a half years!

13:42

Chidi Kunto
Online

Just imagine all the visits, meeting all those people on the ground for two years! It is sometimes easy to see data as meaningless numbers and text, but remember that each person in that queue that day could have been someone who walked 10 kilometres, queued for 4-5 hours and then walked all the way back home! Often these are children who need to do this, so they have less time to attend school. It makes me sad and angry that we got to this point!

13:53

Anyway, Mambo yatakuwa sawa as my mother used to say, 'Things will be okay'. The important thing is we're doing our part to stop that kind of thing from happening.

13:57

Question 2:

Let's see how long people have to queue on average in Maji Ndogo. Keep in mind that many sources like `taps_in_home` have no queues. These are just recorded as 0 in the `time_in_queue` column, so when we calculate averages, we need to exclude those rows. Try using `NULLIF()` do to this.

13:59

You should get a queue time of about 123 min. So on average, people take two hours to fetch water if they don't have a tap in their homes.

14:02

That may sound reasonable, but some days might have more people who need water, and only have time to go and collect some on certain days.

14:15

Question 3:

So let's look at the queue times aggregated across the different days of the week.

14:17

`DAY()` gives you the day of the month. It we want to aggregate data for each day of the week, we need to use another DateTime function, `DAYNAME(column)`. As the name suggests, it returns the day from a timestamp as a string. Using that on the `time_of_record` column will result in a column with day names, Monday, Tuesday, etc., from the timestamp.

14:17

To do this, we need to calculate the average queue time, grouped by day of the week. Remember to revise DateTime functions, and also think about how to present the results clearly.

14:19

I got this:

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

14:21

Wow, ok Saturdays have much longer queue times compared to the other days!

14:31

Question 4:

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

14:36

I get something like this:

hour_of_day	avg_queue_time
6	149
7	149
8	149
...	...

14:43

I don't know about you, but the hour number is difficult to interpret. A format like 06:00 will be easier to read, so let's use that.

14:50

I'll help you with this one. To format time into a specific display format, we can use `TIME_FORMAT(time, format)`. It takes a `time` data field and converts it into a format like `%H:00` which is easy to read. `HOUR(time_of_record)` gives us an integer value of the hour of the day, that won't work with `TIME_FORMAT()`, so we need to use `TIME(time_of_record)` instead.

14:54

Chidi Kunto
Online

So this line:

```
...
HOUR(time_of_record) AS hour_of_day
...
```

in the previous query should become:

```
...
TIME_FORMAT(TIME(time_of_record), '%H:00') AS hour_of_day
...
```

15:00

Try it, and now look at the format! This is much easier for someone to interpret and helps us to tell the story better.

15:05

I get something like this:

hour_of_day	avg_queue_time
06:00	149
07:00	149
08:00	149
...	...

15:08

Chidi Kunto
Online

Can you see that mornings and evenings are the busiest? It looks like people collect water before and after work. Wouldn't it be nice to break down the queue times for each hour of each day? In a spreadsheet, we can just create a pivot table.

Pivot tables are not widely used in SQL, despite being useful for interpreting results. So there are no built-in functions to do this for us. Sometimes the dataset is just so massive that it is the only option.

15:10

For rows, we will use the hour of the day in that nice format, and then make each column a different day!

15:25

To filter a row we use `WHERE`, but using `CASE()` in `SELECT` can filter columns. We can use a `CASE()` function for each day to separate the queue time column into a column for each day. Let's begin by only focusing on Sunday. So, when a row's `DAYNAME(time_of_record)` is `Sunday`, we make that value equal to `time_in_queue`, and `NULL` for any days.

15:37

If you run this query you will see what I mean.

```
SELECT
    TIME_FORMAT(TIME(time_of_record), '%H:00') AS hour_of_day,
    DAYNAME(time_of_record),
    CASE
        WHEN DAYNAME(time_of_record) = 'Sunday' THEN time_in_queue
        ELSE NULL
    END AS Sunday
FROM
    visits
WHERE
    time_in_queue != 0; -- this exludes other sources with 0 queue times.
```

15:44

Where the day name is Sunday, there are queue time values, and all other rows are null. So now if we aggregate that column, we will use all of the values that are not null.

15:44

By adding `AVG()` around the `CASE()` function, we calculate the average, but since all of the other days' values are 0, we get an average for Sunday only, rounded to 0 decimals. To aggregate by the hour, we can group the data by `hour_of_day`, and to make the table chronological, we also order by `hour_of_day`.

15:51

This is the form of the query we will use:

```sql
SELECT
    TIME_FORMAT(TIME(time_of_record), '%H:00') AS hour_of_day,

    -- Sunday
    ROUND(AVG(
        CASE
        WHEN DAYNAME(time_of_record) = 'Sunday' THEN time_in_queue
        ELSE NULL
    END
        ),0) AS Sunday,

    -- Monday
    ROUND(AVG(
        CASE
        WHEN DAYNAME(time_of_record) = 'Monday' THEN time_in_queue
        ELSE NULL
    END
        ),0) AS Monday

    -- Tuesday
    -- Wednesday

FROM
    visits
WHERE
    time_in_queue != 0 -- this excludes other sources with 0 queue times
GROUP BY
    hour_of_day
ORDER BY
    hour_of_day;
```

16:26

We create separate columns for each day with a CASE() function.

16:24

Ok, so here's your challenge: Fill out the query for the rest of the days, and run it. Make sure to specify the day in the CASE() function, and the alias.

16:28

You should have 8 columns with average queue times for each hour in each day!

hour_of_day	Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
06:00	79	190	134	112	134	153	247
07:00	82	186	128	111	139	156	247
08:00	86	183	130	119	129	153	247
09:00	84	127	105	94	99	107	252
10:00	83	119	99	89	95	112	259
...	...	...	...	...	...	...	...

16:31

Now we can compare the queue times for each day, hour by hour!

16:33

Chidi Kunto
Online

See if you can spot these patterns:

1. Queues are very long on a Monday morning and Monday evening as people rush to get water.
2. Wednesday has the lowest queue times, but long queues on Wednesday evening.
3. People have to queue pretty much twice as long on Saturdays compared to the weekdays. It looks like people spend their Saturdays queueing for water, perhaps for the week's supply?
4. The shortest queues are on Sundays, and this is a cultural thing. The people of Maji Ndogo prioritise family and religion, so Sundays are spent with family and friends.

16:36

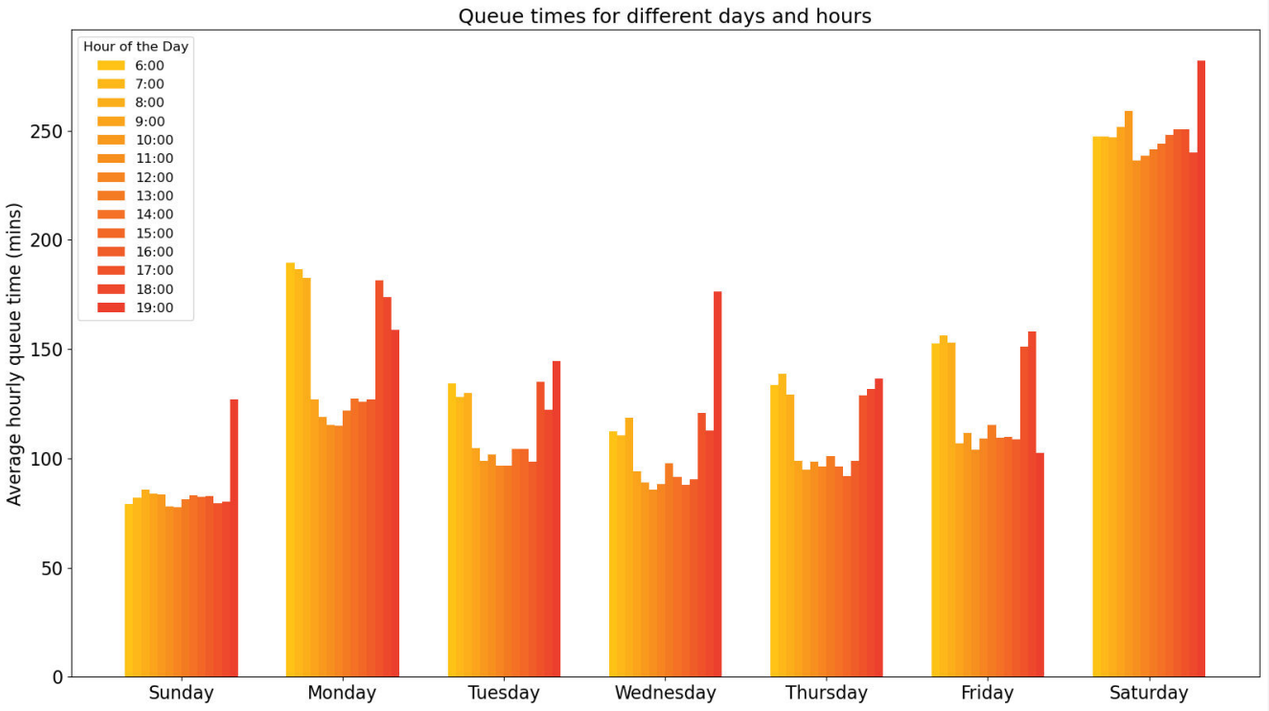
We built a pivot table in SQL! The thing I want you to remember today is: SQL is a set of tools we can apply. By understanding CASE, we could build a complex query that aggregates our data in a format that is very easy to understand.

16:37

To take it one step further, I made a graph! If you copy the pivot table into a spreadsheet, you can too.

16:37

The colors represent the hours of the day, and each bar is the average queue time, for that specific hour and day.



10:39

What do you think about this chart? As we consider presenting this to President Naledi, think about how we can use our findings to tell the story and bring focus to the patterns in the queue times.

16:41

Water Accessibility and infrastructure summary report

This survey aimed to identify the water sources people use and determine both the total and average number of users for each source. Additionally, it examined the duration citizens typically spend in queues to access water.
So let's create a short summary report we can send off to Pres. Naledi:

16:51

Insights

1. Most water sources are rural.
2. 43% of our people are using shared taps. 2000 people often share one tap.
3. 31% of our population has water infrastructure in their homes, but within that group, 45% face non-functional systems due to issues with pipes, pumps, and reservoirs.
4. 18% of our people are using wells of which, but within that, only 28% are clean..
5. Our citizens often face long wait times for water, averaging more than 120 minutes.
6. In terms of queues:

- Queues are very long on Saturdays.
- Queues are longer in the mornings and evenings.
- Wednesdays and Sundays have the shortest queues.

16:58

Start of our plan

We have started thinking about a plan:

1. We want to focus our efforts on improving the water sources that affect the most people.
- Most people will benefit if we improve the shared taps first.
- Wells are a good source of water, but many are contaminated. Fixing this will benefit a lot of people.
- Fixing existing infrastructure will help many people. If they have running water again, they won't have to queue, thereby shorting queue times for others. So we can solve two problems at once.
- Installing taps in homes will stretch our resources too thin, so for now, if the queue times are low, we won't improve that source.
2. Most water sources are in rural areas. We need to ensure our teams know this as this means they will have to make these repairs/upgrades in rural areas where road conditions, supplies, and labour are harder challenges to overcome.

17:04

Practical solutions

1. If communities are using **rivers**, we can dispatch trucks to those regions to provide water temporarily in the short term, while we send out crews to drill for wells, providing a more permanent solution.
2. If communities are using **wells**, we can install filters to purify the water. For wells with **biological** contamination, we can **install UV filters** that kill microorganisms, and for *polluted wells*, we can install **reverse osmosis** filters. In the long term, we need to figure out why these sources are polluted.
3. For **shared taps**, in the short term, we can send additional water tankers to the busiest taps, on the busiest days. We can use the queue time pivot table we made to send tankers at the busiest times. Meanwhile, we can start the work on **installing extra taps** where they are needed. According to UN standards, the maximum acceptable wait time for water is 30 minutes. With this in mind, our aim is to **install taps** to get **queue times below 30 min**.
4. **Shared taps** with **short queue times** (< 30 min) represent a logistical challenge to further reduce waiting times. The most effective solution, installing taps in homes, is resource-intensive and better suited as a long-term goal.
5. **Addressing broken infrastructure** offers a significant impact even with just a single intervention. It is expensive to fix, but so **many people** can **benefit** from repairing one facility. For example, fixing a reservoir or pipe that multiple taps are connected to. We will have to find the commonly affected areas though to see where the problem actually is.

17:10

Think these through a bit and in the meantime I'll send out some emails to get estimates of the cost to repair or improve each of these sources.

17:14

EXPLORE AI ACADEMY

Maji Ndogo: From analysis to action

Weaving the data threads of Maji Ndogo's narrative

Tendai Mubarak
Online

Subject: Results of audit on Maji Ndogo water project

Dear President Naledi,

I hope this email finds you in the best of health and spirits. As you know, my team and I were tasked with conducting an independent audit of the Maji Ndogo water project, specifically the database recording water sources in our country, following the inconsistencies identified by Chidi Kunto and his team.

For clarity, in this specific audit, our objective was to assess the integrity, and accuracy of the data stored in the database. Auditing a database involves verifying that the data it contains is both accurate and has not been tampered with, thereby ensuring that the information can be relied upon for decision-making and governance.

I am pleased to report that the audit is now complete. After a rigorous examination of the database's records, as well as the procedures in place for data entry and modification, we can confirm that the vast majority of the data aligns with the principles of good governance and data-driven decision-making that you have so vigorously championed.

However, we did find some data that was tampered with, which requires your immediate attention. I have attached the records I have re-examined for your review.

Thank you for entrusting us with this crucial task. Your commitment to accountability and transparency is truly commendable.

Sincerely,
Tendai Mubarak
Chief Auditor

Auditor_report.csv

06:11

Introduction
Setting the stage for our data exploration journey.

1 **Generating an ERD**
Understanding the database structure.

2 **Integrating the report**
Adding the auditor report to our database.

3 **Linking records**
Joining employee data to the report.

4 **Gathering evidence**
Building a complex query seeking truth.

Re: Results of audit on Maji Ndogo water project

Dear Tendai,

Thank you for the meticulous work you and your team have put into auditing the Maji Ndogo water project. The depth of your analysis reflects your commitment to excellence and accountability.

I am heartened to learn that our operations are largely in line with our principles of governance. At the same time, your findings are valuable as they highlight areas we can further improve.

Rest assured, I will be convening our data team to address any issues and take any steps necessary to ensure the integrity of our data.

Once again, I commend you and your team for your hard work and dedication. Maji Ndogo is indeed counting on all of us to deliver on our promises. Thank you for playing your part.

All the best,

Aziza Naledi

07:04

Chidi Kunto
Online

ERD

10:30

Hey! It's been a while! I hope you are ready to get stuck in. We got back the auditor's report, so we need to compare their results to ours.

10:33

To integrate the auditor's report, we will need to access many of the tables in the database, so it is important to understand the database structure. To do this we really need to understand the relationships first, so we know where to pull information from. Can you please get the ERD for the `md_water_services` database? Spend a few minutes looking at the diagram.

10:55

Note that the visits table is the central table. `location_id`, `source_id` and `assigned_employee_id` are primary keys in their respective tables, but are all foreign keys in `visits`. These are mostly dsone-to-many relationships. Let's think some of these through so we are sure this is correct.

10:59

The `visits` table logs all the times we've been to different places, and we can see that some locations have been visited multiple times. On the other hand, the `location` table has all the specifics about each place we've been but it only includes each location once. So, it's a one-to-many relationship: for each unique location in the `location` table, there might be many corresponding records in the `visits` table detailing all the different times we went there.

11:01

Chidi Kunto
Online

Let's look at the relationship between the `visits` and `water_quality` tables. For every entry in the `visits` table, there should be one unique corresponding record in the `water_quality` table. This means each visit recorded is associated with a specific water quality score, ensuring a one-to-one relationship between the `visits` and `water_quality` tables.

11:03

But if we look at the ERD, it shows a many-to-one relationship. This does not agree with our thinking. Errors like these can cause problems, so let's fix that.

11:04

We should normally be careful making a change like this. First you should check that `record_id` is unique for both tables, and are indeed one-to-one.

11:04

I did that, so all we have to do is right-click on the relationship line and select `Edit relationship`, then at the bottom, select the `Foreign key` tab, and change the `Cardinality` to `one-to-one`. Now the relationship line should indicate a one-to-one relation.

11:04

Integrating the Auditor's report

11:10

Ok, I sent you a .csv file of the auditor's results. You should get it loaded into SQL. You will have to think back to the start of our journey on how to do that.

11:13

To make sure we have the same names, use this query to create the table first, then Import the CSV file:

```
DROP TABLE IF EXISTS `auditor_report`;

CREATE TABLE `auditor_report` (
  `location_id` VARCHAR(32),
  `type_of_water_source` VARCHAR(64),
  `true_water_source_score` int DEFAULT NULL,
  `statements` VARCHAR(255)
);
```

11:14

Got it? Good. Looking at the table and the data dictionary I sent you, you can see the type of info we have.

11:16

Chidi Kunto
Online

Wow! First off, it looks like we have 1620 records, or sites that they re-visited. I see a location_id, type of water source at that location, and the quality score of the water source, that is now independently measured. Our auditor also investigated each site a bit by speaking to a few locals. Their statements are also captured in his results.

11:22

We need to tackle a couple of questions here.
1. Is there a difference in the scores?
2. If so, are there patterns?

11:28

For the first question, we will have to compare the quality scores in the `water_quality` table to the auditor's scores. The `auditor_report` table used `location_id`, but the quality scores table only has a `record_id` we can use. The `visits` table links `location_id` and `record_id`, so we can link the `auditor_report` table and `water_quality` using the `visits` table.

11:37

So first, grab the `location_id` and `true_water_source_score` columns from `auditor_report`.

11:44

You should see this kind of thing:

location_id	true_water_source_score
AkHa00008	3
AkHa00053	9
...	...

11:46

Now, we join the visits table to the auditor_report table. Make sure to grab `subjective_quality_score`, `record_id` and `location_id`.

11:53

I get:

audit_location	true_water_source_score	visit_location	record_id
SoRu34980	1	SoRu34980	5185
AkRu08112	3	AkRu08112	59367
AkLu02044	0	AkLu02044	37379
...	...	...	...

12:00

Note that I specified from which table each selected column is from in this query:

```
SELECT
    auditor_report.location_id AS audit_location,
    auditor_report.true_water_source_score,
    visits.location_id AS visit_location,
    visits.record_id
FROM
    auditor_report
JOIN
    visits
    ON auditor_report.location_id = visits.location_id;
```

12:02

Now that we have the `record_id` for each location, our next step is to retrieve the corresponding scores from the `water_quality` table. We are particularly interested in the `subjective_quality_score`. To do this, we'll `JOIN` the `visits` table and the `water_quality` table, using the `record_id` as the connecting key.

12:04

Whoo, first try:

audit_location	true_water_source_score	visit_location	record_id	subjective_quality_score
SoRu34980	1	SoRu34980	5185	1
AkRu08112	3	AkRu08112	59367	3
AkLu02044	0	AkLu02044	37379	0

12:11

It doesn't matter if your columns are in a different format, because we are about to clean this up a bit. Since it is a duplicate, we can drop one of the `location_id` columns. Let's leave `record_id` and rename the scores to `surveyor_score` and `auditor_score` to make it clear which scores we're looking at in the results set.

12:17

I got this:

location_id	record_id	auditor_score	employee_score
SoRu34980	5185	1	1
AkRu08112	59367	3	3
AkLu02044	37379	0	0
AkHa00421	51627	3	3
...	...	...	...

12:18

Since were joining 1620 rows of data, we want to keep track of the number of rows we get each time we run a query. We can either set the maximum number of rows we want from "Limit to 1000 rows" to a larger number like 10000, or we can force SQL to give us all of the results, using LIMIT 10000.

12:19

Ok, let's analyse! A good starting point is to check if the auditor's and exployees' scores agree. There are many ways to do it. We can have a `WHERE` clause and check if `surveyor_score = auditor_score`, or we can subtract the two scores and check if the result is 0.

12:25

Chidi Kunto
Online

You try and figure this one out. If you run into an error, remember that you can't use aliases in `WHERE`, so you have to use the same name as in the `SELECT` part, like this: `auditor_report.true_water_source_score`

12:28

You got 2505 rows right? Some of the locations were visited multiple times, so these records are duplicated here. To fix it, we set visits.visit_count = 1 in the `WHERE` clause. Make sure you reference the alias you used for `visits` in the join.

12:30

With the duplicates removed I now get 1518. What does this mean considering the auditor visited 1620 sites?

12:31

I think that is an excellent result. 1518/1620 = 94% of the records the auditor checked were correct!!

12:43

But that means that 102 records are incorrect. So let's look at those. You can do it by adding one character in the last query!

12:47

I get this:

location_id	record_id	auditor_score	employee_score
AkRu05215	21160	3	10
KiRu29290	7938	3	10
KiHa22748	43140	9	10
SoRu37841	18495	6	10
KiRu27884	33931	1	10
KiZu31170	17950	9	10
...	...	...	...

12:52

Since we used some of this data in our previous analyses, we need to make sure those results are still valid, now we know some of them are incorrect. We didn't use the scores that much, but we relied a lot on the `type_of_water_source`, so let's check if there are any errors there.

12:59

So, to do this, we need to grab the `type_of_water_source` column from the `water_source` table and call it `survey_source`, using the `source_id` column to `JOIN`. Also select the `type_of_water_source` from the `auditor_report` table, and call it `auditor_source`.

13:02

I got this:

location_id	auditor_source	survey_source	record_id	auditor_score	employee_score
AkRu05215	well	well	21160	3	10
KiRu29290	shared_tap	shared_tap	7938	3	10
KiHa22748	tap_in_home_broken	tap_in_home_broken	43140	9	10
SoRu37841	shared_tap	shared_tap	18495	6	10
KiRu27884	well	well	33931	1	10

13:03

So what I can see is that the types of sources look the same! So even though the scores are wrong, the integrity of the `type_of_water_source` data we analysed last time is not affected.

13:14

Once you're done, remove the columns and JOIN statement for water_sources again.

13:15

Chidi Kunto
Online

Linking records to employees
Next up, let's look at where these errors may have come from. At some of the locations, employees assigned scores incorrectly, and those records ended up in this results set.

13:21

I think there are two reasons this can happen.
1. These workers are all humans and make mistakes so this is expected.
2. Unfortunately, the alternative is that someone assigned scores incorrectly on purpose!

13:25

In either case, the employees are the source of the errors, so let's `JOIN` the `assigned_employee_id` for all the people on our list from the `visits` table to our query. Remember, our query shows the shows the 102 incorrect records, so when we join the employee data, we can see which employees made these incorrect records.

13:26

Chidi Kunto
Online

I get this:

location_id	record_id	assigned_employee_id	auditor_score	employee_score
AkRu05215	21160	34	3	10
KiRu29290	7938	1	3	10
KiHa22748	43140	1	9	10
SoRu37841	18495	34	6	10
KiRu27884	33931	1	1	10
KiZu31170	17950	5	9	10
...	...	...	...	...

13:31

So now we can link the incorrect records to the employees who recorded them. The ID's don't help us to identify them. We have employees' names stored along with their IDs, so let's fetch their names from the `employees` table instead of the ID's.

13:41

I get this table:

location_id	record_id	employee_name	auditor_score	employee_score
AkRu05215	21160	Rudo Imani	3	10
KiRu29290	7938	Bello Azibo	3	10
KiHa22748	43140	Bello Azibo	9	10
SoRu37841	18495	Rudo Imani	6	10
...	...	...	...	...

13:52

Well this query is massive and complex, so maybe it is a good idea to save this as a CTE, so when we do more analysis, we can just call that CTE like it was a table. Call it something like `Incorrect_records`. Once you are done, check if this query `SELECT * FROM Incorrect_records`, gets the same table back.

13:57

Now that we defined `Incorrect_records`, we can query it like any other table.

14:00

Let's first get a unique list of employees from this table. Think back to the start of your SQL journey to answer this one. I got 17 employees.

14:02

Next, let's try to calculate how many mistakes each employee made. So basically we want to count how many times their name is in `Incorrect_records` list, and then group them by name, right?

14:07

Something like this:

employee_name	number_of_mistakes
Rudo Imani	5
Bello Azibo	26
Zuriel Matembo	17
Yewande Ebele	3
...	...

14:10

Introduction
Setting the stage for our data exploration journey.

1 **Generating an ERD**
Understanding the database structure.

2 **Integrating the report**
Adding the auditor report to our database.

3 **Linking records**
Joining employee data to the report.

4 **Gathering evidence**
Building a complex query seeking truth.

Chidi Kunto
Online

Do you see a pattern there? Order the list if you can't see it yet.

14:15

It looks like some of our surveyors are making a lot of "mistakes" while many of the other surveyors are only making a few. I don't like where this is going!

14:17

Chidi Kunto
Online

Gathering some evidence
Ok, so thinking about this a bit. How would we go about finding out if any of our employees are corrupt?

14:21

Let's say all employees make mistakes, if someone is corrupt, they will be making a lot of "mistakes", more than average, for example. But someone could just be clumsy, so we should try to get more evidence...

14:23

Our auditor did say some of the things he heard on the streets were quite shady, and he recorded this in the `statements` column. Considering both of these sources should give us a pretty reliable answer.

14:25

So let's try to find all of the employees who have an above-average number of mistakes. Let's break it down into steps first:

1. We have to first calculate the number of times someone's name comes up. (we just did that in the previous query). Let's call it `error_count`.
2. Then, we need to calculate the average number of mistakes employees made. We can do that by taking the average of the previous query's results. Something like this:

```
SELECT
    AVG(number_of_mistakes)
FROM
    error_count;
```

   Let's call that result `avg_error_count_per_empl`, which would be a scalar value.

3. Finaly we have to compare each employee's `error_count` with `avg_error_count_per_empl`. We will call this results set our `suspect_list`. Remember that we can't use an aggregate result in `WHERE`, so we have to use `avg_error_count_per_empl` as a subquery.

```
SELECT
    employee_name,
    number_of_mistakes
FROM
    error_count
WHERE
    number_of_mistakes > (avg_error_count_per_empl);
```

14:26

1. Let's start by cleaning up our code a bit. First, `Incorrect_records` is a result we'll be using for the rest of the analysis, but it makes the query a bit less readable. So, let's convert it to a `VIEW`. We can then use it as if it was a table. It will make our code much simpler to read, but, it comes at a cost. We can add comments to CTEs in our code, so if we return to that query a year later, we can read those comments and quickly understand what `Incorrect_records` represents. If we save it as a `VIEW`, it is not as obvious. So we should add comments in places where we use `Incorrect_records`.

14:27

Chidi Kunto
Online

So, replace `WITH` with `CREATE VIEW` like this, and note that I added the `statements` column to this table in line 8 too:

14:28

```sql
CREATE VIEW Incorrect_records AS (
SELECT
    auditor_report.location_id,
    visits.record_id,
    employee.employee_name,
    auditor_report.true_water_source_score AS auditor_score,
    wq.subjective_quality_score AS employee_score,
    auditor_report.statements AS statements
FROM
    auditor_report
JOIN
    visits
    ON auditor_report.location_id = visits.location_id
JOIN
    water_quality AS wq
    ON visits.record_id = wq.record_id
JOIN
    employee
    ON employee.assigned_employee_id = visits.assigned_employee_id
WHERE
    visits.visit_count =1
    AND auditor_report.true_water_source_score != wq.subjective_quality_score);
```

14:29

Now, calling `SELECT * FROM Incorrect_records` gives us the same result as the CTE did.

14:30

Chidi Kunto
Online

Next, we convert the query `error_count`, we made earlier, into a CTE. Test it to make sure it gives the same result again, using `SELECT * FROM Incorrect_records`. On large queries like this, it is better to build the query, and test each step, because fixing errors becomes harder as the query grows.

14:31

Like this:

```
WITH error_count AS ( -- This CTE calculates the number of mistakes each employee made
    SELECT
        employee_name,
        COUNT(employee_name) AS number_of_mistakes
    FROM
        Incorrect_records
            /* Incorrect_records is a view that joins the audit report to the database
            for records where the auditor and
            employees scores are different*/
    GROUP BY
        employee_name)
-- Query
SELECT * FROM error_count;
```

14:33

2. Now calculate the average of the `number_of_mistakes` in `error_count`. You should get a single value.

14:35

3. To find the employees who made more mistakes than the average person, we need the employee's names, the number of mistakes each one made, and filter the employees with an above-average number of mistakes.

HINT: Use `SELECT AVG(mistake_count) FROM error_count` as a custom filter in the WHERE part of our query.

14:37

There they are:

employee_name	number_of_mistakes
Zuriel Matembo	17
Malachi Mavuso	21
Lalitha Kaburi	7
Bello Azibo	26

These are the employees who made more mistakes, on average, than their peers, so let's have a closer look at them.

14:39

We should look at the `Incorrect_records` table again, and isolate all of the records these four employees gathered. We should also look at the statements for these records to look for patterns.

14:40

First, convert the `suspect_list` to a CTE, so we can use it to filter the records from these four employees. Make sure you get the names of the four "suspects", and their mistake count as a result, using `SELECT employee_name FROM suspect_list`.

14:41

You should get a column of names back. So let's just recap here...

1. We use `Incorrect_records` to find all of the records where the auditor and employee scores don't match.
2. We then used `error_count` to aggregate the data, and got the number of mistakes each employee made.
3. Finally, `suspect_list` retrieves the data of employees who make an above-average number of mistakes.

Now we can filter that `Incorrect_records` CTE to identify all of the records associated with the four employees we identified.

14:43

Firstly, let's add the `statements` column to the `Incorrect_records` CTE. Then pull up all of the records where the employee_name is in the suspect list. **HINT:** Use `SELECT employee_name FROM suspect_list` as a subquery in WHERE.

14:44

I got something like this:

employee_name	location_id	statements
Bello Azibo	Kils23853	Villagers' wary accounts of an official's arrogance...
Bello Azibo	KiHa22748	A young girl's hopeful eyes...
Bello Azibo	KiRu27884	A traditional healer's empathy t...
Zuriel Matembo	KiZu31170	A community leader stood with...
Bello Azibo	AkRu06495	A healthcare worker in...

14:45

Using this kind of query:

```
...
-- This query filters all of the records where the "corrupt" employees gathered data.
SELECT
...
FROM
    Incorrect_records
WHERE
    employee_name IN (SELECT employee_name FROM suspect_list);
```

14:46

This query is complex, right! But, if we document it well, it is simpler to understand. Oh, and you don't want to see what this query looks like using only subqueries!

```
WITH error_count AS ( -- This CTE calculates the number of mistakes each employee made
    SELECT
        employee_name,
        COUNT(employee_name) AS number_of_mistakes
    FROM
        Incorrect_records
            /* Incorrect_records is a view that joins the audit report to the database
            for records where the auditor and
            employees scores are different*/
    GROUP BY
        employee_name),
suspect_list AS (-- This CTE SELECTS the employees with above-average mistakes
    SELECT
        employee_name,
        number_of_mistakes
    FROM
        error_count
    WHERE
        number_of_mistakes > (SELECT AVG(number_of_mistakes) FROM error_count))

-- This query filters all of the records where the "corrupt" employees gathered data.
SELECT
    employee_name,
    location_id,
    statements
FROM
    Incorrect_records
WHERE
    employee_name in (SELECT employee_name FROM suspect_list);
```

14:48

Chidi Kunto
Online

If you have a look, you will notice some alarming statements about these four officials (look at these records: `AkRu04508`, `AkRu07310`, `KiRu29639`, `AmAm09607`, for example. See how the word "cash" is used a lot in these statements.

14:49

Filter the records that refer to "cash".

14:50

Let's just do one more check to make sure...

14:57

Check if there are any employees in the `Incorrect_records` table with statements mentioning "cash" that are not in our suspect list. This should be as simple as adding one word.

14:58

I get an empty result, so no one, except the four suspects, has these allegations of bribery.

15:04

Chidi Kunto
Online

So we can sum up the evidence we have for Zuriel Matembo, Malachi Mavuso, Bello Azibo and Lalitha Kaburi:

1. They all made more mistakes than their peers on average.
2. They all have incriminating statements made against them, and only them.

Keep in mind, that this is not decisive proof, but it is concerning enough that we should flag it. Pres. Naledi has worked hard to stamp out corruption, so she would urge us to report this.

15:06

I am a bit shocked to be honest! After all our teams set out to do, it is hard for me to uncover this. I'll let Pres. Naledi know what we found out.

15:07

EXPLORE AI ACADEMY

Maji Ndogo: From analysis to action

Charting the course for Maji Ndogo's water future

Aziza Naledi
Online

Dear Team,

I would like to thank the team for uncovering the corruption of our field workers and letting me know. As you all know, I have no tolerance for people who look after themselves first, at the cost of everyone else, so I have taken the necessary steps!

Our journey continues, as we aim to convert our data into actionable knowledge. Understanding the situation is one thing, but it's the translation of that understanding into informed decisions that will truly make a difference.

As we step into this next phase, you will be shaping our raw data into meaningful views - providing essential information to decision-makers. This will enable us to discern the materials we need, plan our budgets, and identify the areas requiring immediate attention. We're not just analysing data; we're making it speak in a language that everyone involved in this mission can understand and act upon.

Lastly, we'll be creating job lists for our engineers. Their expertise will be invaluable in tackling the challenges we face, but they can only do their job effectively when they have clear, data-driven directions.

Remember, each step you take in this process contributes to a larger goal - the transformation of Maji Ndogo. Your diligence and dedication is instrumental in shaping a brighter future for our community. Thank you for being part of this journey.

All the best,
Aziza

07:13

1. **Introduction**
   Starting the final journey
2. **Joining pieces together**
   Finding the data we need across tables
3. **The last analysis**
   Finding the final insights from our data
4. **Summary report**
   Sharing our knowledge with decision makers
5. **A practical plan**
   From analysis to action

Hey! How have you been? I am moving to another project soon, so this might be the last time we work together for a while. I thought you would appreciate this:

07:20

It's really good to see justice taking the front seat. Maji Ndogo is changing, and I think it's because President Naledi gets it—our country's at a tipping point. She's serious about using data and having no room for corruption, and that gives me hope.

07:22

We still have a bit of analysis to wrap up, and then we need to create a table to track our progress. Let's start with the last bit of analysis.

07:24

So I used to be tempted to put all of the columns from all of the tables in one place/table, and then analyse the data, but on a dataset of this size, we're going to run into performance issues.

07:27

So, we should rather spend a minute thinking about the questions we still have, and create queries to answer them, specifically. Doing this means that we will only use the data we need to answer our question.

07:33

Let's summarise the data we need, and where to find it:

- All of the information about the location of a water source is in the `location` table, specifically the town and province of that water source.
- `water_source` has the type of source and the number of people served by each source.
- `visits` has queue information, and connects `source_id` to `location_id`. There were multiple visits to sites, so we need to be careful to include duplicate data (`visit_count` > 1).
- `well_pollution` has information about the quality of water from only wells, so we need to keep that in mind when we join this table.

07:25

Previously, we couldn't link provinces and towns to the type of water sources, the number of people served by those sources, queue times, or pollution data, but we can now. So, what type of relationships can we look at?

07:39

Chidi Kunto
Online

Things that spring to mind for me:

1. Are there any specific provinces, or towns where some sources are more abundant?
2. We identified that `tap_in_home_broken` taps are easy wins. Are there any towns where this is a particular problem?

07:42

To answer question 1, we will need `province_name` and `town_name` from the `location` table. We also need to know `type_of_water_source` and `number_of_people_served` from the `water_source` table.

07:49

The problem is that the `location` table uses `location_id` while `water_source` only has `source_id`. So we won't be able to join these tables directly. But the `visits` table maps `location_id` and `source_id`. So if we use `visits` as the table we query from, we can join `location` where the `location_id` matches, and `water_source` where the `source_id` matches.

07:54

Before we can analyse, we need to assemble data into a table first. It is quite complex, but once we're done, the analysis is much simpler!

07:55

Start by joining `location` to `visits`.

07:58

You should have the following columns:

province_name	town_name	visit_count	location_id
Sokoto	Ilanga	1	Soll32582
...	...	...	...

08:00

Now, we can join the `water_source` table on the key shared between `water_source` and `visits`.

08:04

This is what you should have:

province_name	town_name	visit_count	location_id	type_of_water_source	number_of_people_served
Akatsi	Harare	1	AkHa00000	tap_in_home	956
Akatsi	Harare	1	AkHa00001	tap_in_home_broken	930
Akatsi	Harare	1	AkHa00002	tap_in_home_broken	486
Akatsi	Harare	1	AkHa00003	well	364
...	...	...	...	...	...

08:09

Note that there are rows where `visit_count > 1`. These were the sites our surveyors collected additional information for, but they happened at the same source/location. For example, add this to your query: `WHERE visits.location_id = 'AkHa00103'`

08:21

This is what I got:

province_name	town_name	visit_count	location_id	type_of_water_source	number_of_people_served
Akatsi	Harare	1	AkHa00103	shared_tap	3340
Akatsi	Harare	2	AkHa00103	shared_tap	3340
Akatsi	Harare	3	AkHa00103	shared_tap	3340
Akatsi	Harare	4	AkHa00103	shared_tap	3340
Akatsi	Harare	5	AkHa00103	shared_tap	3340
Akatsi	Harare	6	AkHa00103	shared_tap	3340
Akatsi	Harare	7	AkHa00103	shared_tap	3340
Akatsi	Harare	8	AkHa00103	shared_tap	3340

08:24

There you can see what I mean. For one location, there are multiple `AkHa00103` records for the same location. If we aggregate, we will include these rows, so our results will be incorrect. To fix this, we can just select rows where `visits.visit_count = 1`.

08:27

Remove `WHERE visits.location_id = 'AkHa00103'` and add the `visits.visit_count = 1` as a filter.

08:31

Ok, now that we verified that the table is joined correctly, we can remove the `location_id` and `visit_count` columns.

08:37

Add the `location_type` column from `location` and `time_in_queue` from `visits` to our results set.

08:38

We should have a table like this:

province_name	town_name	type_of_water_source	location_type	number_of_people_served	time_in_queue
Sokoto	Ilanga	river	Urban	402	15
Kilimani	Rural	well	Rural	252	0
Hawassa	Rural	shared_tap	Rural	542	62
Akatsi	Lusaka	well	Urban	210	0
Akatsi	Rural	shared_tap	Rural	2598	28
Kilimani	Rural	river	Rural	862	9
Akatsi	Rural	tap_in_home_broken	Rural	496	0
Kilimani	Rural	tap_in_home	Rural	562	0

08:40

Last one! Now we need to grab the `results` from the `well_pollution` table.
This one is a bit trickier. The `well_pollution` table contained only data for `well`. If we just use JOIN, we will do an inner join, so that only records that are in `well_pollution` AND `visits` will be joined. We have to use a LEFT JOIN to join the`results` from the `well_pollution` table for well sources, and will be NULL for all of the rest. Play around with the different JOIN operations to make sure you understand why we used LEFT JOIN.

08:50

Here is the code:

```
-- This table assembles data from different tables into one to simplify analysis
SELECT
    water_source.type_of_water_source,
    location.town_name,
    location.province_name,
    location.location_type,
    water_source.number_of_people_served,
    visits.time_in_queue,
    well_pollution.results
FROM
    visits
LEFT JOIN
    well_pollution
    ON well_pollution.source_id = visits.source_id
INNER JOIN
    location
    ON location.location_id = visits.location_id
INNER JOIN
    water_source
    ON water_source.source_id = visits.source_id
WHERE
    visits.visit_count = 1;
```

08:56

So this table contains the data we need for this analysis. Now we want to analyse the data in the results set. We can either create a CTE, and then query it, or in my case, I'll make it a `VIEW` so it is easier to share with you. I'll call it the `combined_analysis_table`.

09:01

Chidi Kunto
Online

Here it is:

```
CREATE VIEW combined_analysis_table AS
-- This view assembles data from different tables into one to simplify analysis
SELECT
    water_source.type_of_water_source AS source_type,
    location.town_name,
    location.province_name,
    location.location_type,
    water_source.number_of_people_served AS people_served,
    visits.time_in_queue,
    well_pollution.results
FROM
    visits
LEFT JOIN
    well_pollution
    ON well_pollution.source_id = visits.source_id
INNER JOIN
    location
    ON location.location_id = visits.location_id
INNER JOIN
    water_source
    ON water_source.source_id = visits.source_id
WHERE
    visits.visit_count = 1;
```

09:05

This view creates a "table" that pulls all of the important information from different tables into one. You may notice our query is starting to slow down because it involves a lot of steps, and runs on 60000 rows of data.

09:10

Chidi Kunto
Online

The last analysis
We're building another pivot table! This time, we want to break down our data into provinces or towns and source types. If we understand where the problems are, and what we need to improve at those locations, we can make an informed decision on where to send our repair teams.

09:14

We did most of this before, so I'll give you the queries I used, explain them a bit, and then we'll look at the results.

09:16

The queries I am sharing with you today are not formatted well because I am trying to fit them into my chat messages, but make sure you add comments, and document your code well so you can use it again.

09:17

This is the query I used:

```sql
WITH province_totals AS (-- This CTE calculates the population of each province
    SELECT
        province_name,
        SUM(people_served) AS total_ppl_serv
    FROM
        combined_analysis_table
    GROUP BY
        province_name
)
 SELECT
    ct.province_name,
    -- These case statements create columns for each type of source.
    -- The results are aggregated and percentages are calculated
    ROUND((SUM(CASE WHEN source_type = 'river'
        THEN people_served ELSE 0 END) * 100.0 / pt.total_ppl_serv), 0) AS river,
    ROUND((SUM(CASE WHEN source_type = 'shared_tap'
        THEN people_served ELSE 0 END) * 100.0 / pt.total_ppl_serv), 0) AS shared_tap,
    ROUND((SUM(CASE WHEN source_type = 'tap_in_home'
        THEN people_served ELSE 0 END) * 100.0 / pt.total_ppl_serv), 0) AS tap_in_home,
    ROUND((SUM(CASE WHEN source_type = 'tap_in_home_broken'
        THEN people_served ELSE 0 END) * 100.0 / pt.total_ppl_serv), 0) AS tap_in_home_broken,
    ROUND((SUM(CASE WHEN source_type = 'well'
        THEN people_served ELSE 0 END) * 100.0 / pt.total_ppl_serv), 0) AS well
FROM
    combined_analysis_table ct
JOIN
    province_totals pt ON ct.province_name = pt.province_name
GROUP BY
    ct.province_name
ORDER BY
    ct.province_name;
```

09:21

`province_totals` is a CTE that calculates the sum of all the people surveyed grouped by province. If you replace the query above with this one:

```
SELECT
    *
FROM
    province_totals;
```

09:21

You should get a table of province names and summed up populations for each province.

09:23

The main query selects the province names, and then like we did last time, we create a bunch of columns for each type of water source with CASE statements, sum each of them together, and calculate percentages.

09:25

We join the `province_totals` table to our `combined_analysis_table` so that the correct value for each province's pt.total_ppl_serv value is used.

09:31

Finally we group by `province_name` to get the provincial percentages.

09:36

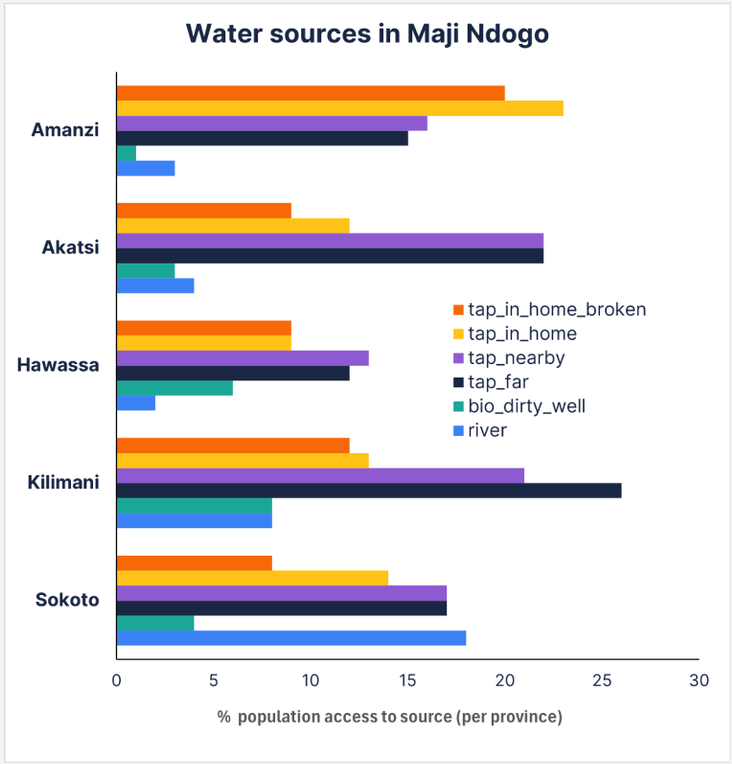
Run the query and see if you can spot any of the following patterns:

- Look at the river column, Sokoto has the largest population of people drinking river water. We should send our drilling equipment to Sokoto first, so people can drink safe filtered water from a well.
- The majority of water from Amanzi comes from taps, but half of these home taps don't work because the infrastructure is broken. We need to send out engineering teams to look at the infrastructure in Amanzi first. Fixing a large pump, treatment plant or reservoir means that thousands of people will have running water. This means they will also not have to queue for water, so we improve two things at once.

Spot any other interesting patterns?

09:42

I made another graph to help us see the patterns:



07:20

Chidi Kunto
Online

Let's aggregate the data per town now. You might think this is simple, but one little town makes this hard. Recall that there are two towns in Maji Ndogo called Harare. One is in Akatsi, and one is in Kilimani. Amina is another example. So when we just aggregate by town, SQL doesn't distinguish between the different Harare's, so it combines their results.

09:54

To get around that, we have to group by province first, then by town, so that the duplicate towns are distinct because they are in different towns.

09:55

Here is the query:

```sql
WITH town_totals AS (-- This CTE calculates the population of each town
-- Since there are two Harare towns, we have to group by province_name and town_name
    SELECT province_name, town_name, SUM(people_served) AS total_ppl_serv
    FROM combined_analysis_table
    GROUP BY province_name,town_name
)
SELECT
    ct.province_name,
    ct.town_name,
    ROUND((SUM(CASE WHEN source_type = 'river'
        THEN people_served ELSE 0 END) * 100.0 / tt.total_ppl_serv), 0) AS river,
    ROUND((SUM(CASE WHEN source_type = 'shared_tap'
        THEN people_served ELSE 0 END) * 100.0 / tt.total_ppl_serv), 0) AS shared_tap,
    ROUND((SUM(CASE WHEN source_type = 'tap_in_home'
        THEN people_served ELSE 0 END) * 100.0 / tt.total_ppl_serv), 0) AS tap_in_home,
    ROUND((SUM(CASE WHEN source_type = 'tap_in_home_broken'
        THEN people_served ELSE 0 END) * 100.0 / tt.total_ppl_serv), 0) AS tap_in_home_broken,
    ROUND((SUM(CASE WHEN source_type = 'well'
        THEN people_served ELSE 0 END) * 100.0 / tt.total_ppl_serv), 0) AS well
FROM
    combined_analysis_table ct
JOIN -- Since the town names are not unique, we have to join on a composite key
    town_totals tt ON ct.province_name = tt.province_name AND ct.town_name = tt.town_name
GROUP BY -- We group by province first, then by town.
    ct.province_name,
    ct.town_name
ORDER BY
    ct.town_name;
```

10:01

Here the CTE calculates `town_totals` which returns three columns:
`province_name`,
`town_name`,
`total_ppl_serv`.

10:04

In the main query we select the `province_name` and the `town_name` and then calculate the percentage of people using each source type, using the CASE statements.
Then we join `town_totals` to `combined_analysis_table`, but this time the `town_names` are not unique, so we have to join `town_totals`, but we check that both the `province_name` and `town_name` matches the values in `combined_analysis_table`.

10:11

Then we group it by `province_name`, then `town_name`. This query can take a while to calculate, so hopefully, you start to see how a query can quickly become slow as it becomes more complex.

10:12

This query can take a while to calculate, so hopefully you start to see how a query can quickly become slow as it becomes more complex.

10:14

Before we jump into the data, let's store it as a temporary table first, so it is quicker to access.

10:15

Chidi Kunto
Online

Temporary tables in SQL are a nice way to store the results of a complex query. We run the query once, and the results are stored as a table. The catch? If you close the database connection, it deletes the table, so you have to run it again each time you start working in MySQL. The benefit is that we can use the table to do more calculations, without running the whole query each time.

10:16

To do it, add this to the start of your query:

```
CREATE TEMPORARY TABLE town_aggregated_water_access
WITH town_totals AS                                  .
...
```

10:17

So this result:

province_name	town_name	tap_in_home	tap_in_home_broken	shared_tap	well	river
Akatsi	Harare	28	27	17	27	2
Akatsi	Kintampo	31	26	15	26	2
Akatsi	Lusaka	28	28	17	26	2
Akatsi	Rural	9	5	59	22	6
Amanzi	Abidjan	22	19	53	4	2
Amanzi	Amina	3	56	24	9	8
Amanzi	Asmara	24	20	49	4	3
Amanzi	Bello	20	22	53	3	3
Amanzi	Dahabu	55	1	37	4	3
Amanzi	Pwani	20	21	53	4	3
Amanzi	Rural	30	30	27	10	3
Hawassa	Amina	19	24	14	42	2
Hawassa	Deka	23	21	16	38	3
...	...	...	...	...	...	...

10:19

So, let's order the results set by each column. If we order `river` DESC it confirms what we saw on a provincial level; People are drinking river water in Sokoto.

10:20

But look at the `tap_in_home` percentages in Sokoto too. Some of our citizens are forced to drink unsafe water from a river, while a lot of people have running water in their homes in Sokoto. Large disparities in water access like this often show that the wealth distribution in Sokoto is very unequal. We should mention this in our report. We should also send our drilling teams to Sokoto first to drill some wells for the people who are drinking river water, specifically the rural parts and the city of Bahari.

10:21

Next, sort the data by `province_name` next and look at the data for Amina in Amanzi. Here only 3% of Amina's citizens have access to running tap water in their homes. More than half of the people in Amina have taps installed in their homes, but they are not working. We should send out teams to go and fix the infrastructure in Amina first. Fixing taps in people's homes, means those people don't have to queue for water anymore, so the queues in Amina will also get shorter!

10:24

There are still many gems hidden in this table. For example, which town has the highest ratio of people who have taps, but have no running water? Running this:

```
SELECT
    province_name,
    town_name,
    ROUND(tap_in_home_broken / (tap_in_home_broken + tap_in_home) * 100,0) AS Pct_broken_taps
FROM
    town_aggregated_water_access
```

We can see that Amina has infrastructure installed, but almost none of it is working, and only the capital city, Dahabu's water infrastructure works. Strangely enough, all of the politicians of the past government lived in Dahabu, so they made sure they had water. The point is, look how simple our query is now! It's like we're back at the beginning of our journey!

10:25

It would be so nice to see this data, right? But because there are so many sources, and so many towns, it is hard to explain this visually without some better tools. Imagine we could have a graph where we do this kind of filtering and sorting of data in the graph!! Well, you will meet Dalila soon, and she will help us to build something like that!

10:26

Summary report

10:28

Insights
Ok, so let's sum up the data we have.
A couple of weeks ago we found some interesting insights:

1. Most water sources are rural in Maji Ndogo.
2. 43% of our people are using shared taps. 2000 people often share one tap.
3. 31% of our population has water infrastructure in their homes, but within that group,
4. 45% face non-functional systems due to issues with pipes, pumps, and reservoirs. Towns like Amina, the rural parts of Amanzi, and a couple of towns across Akatsi and Hawassa have broken infrastructure.
5. 18% of our people are using wells of which, but within that, only 28% are clean. These are mostly in Hawassa, Kilimani and Akatsi.
6. Our citizens often face long wait times for water, averaging more than 120 minutes:
    - Queues are very long on Saturdays.
    - Queues are longer in the mornings and evenings.
    - Wednesdays and Sundays have the shortest queues.

10:30

Plan of action

1. We want to focus our efforts on improving the water sources that affect the most people.
    - Most people will benefit if we improve the shared taps first.
2. Wells are a good source of water, but many are contaminated. Fixing this will benefit a lot of people.
3. Fixing existing infrastructure will help many people. If they have running water again, they won't have to queue, thereby shorting queue times for others. So we can solve two problems at once.
4. Installing taps in homes will stretch our resources too thin, so for now if the queue times are low, we won't improve that source.
5. Most water sources are in rural areas. We need to ensure our teams know this as this means they will have to make these repairs/upgrades in rural areas where road conditions, supplies, and labour are harder challenges to overcome.

10:33

Practical solutions:

1. If communities are using **rivers**, we will dispatch trucks to those regions to provide water temporarily in the short term, while we send out crews to drill for wells, providing a more permanent solution. Sokoto is the first province we will target.
2. If communities are using **wells**, we will install filters to purify the water. For **chemically polluted wells**, we can install **reverse osmosis** (RO) filters, and for wells with **biological** contamination, we can **install UV filters** that kill microorganisms - but we should install RO filters too. In the long term, we must figure out why these sources are polluted.
3. For **shared taps**, in the short term, we can send additional water tankers to the busiest taps, on the busiest days. We can use the queue time pivot table we made to send tankers at the busiest times. Meanwhile, we can start the work on **installing extra taps** where they are needed. According to UN standards, the maximum acceptable wait time for water is 30 minutes. With this in mind, our aim is to **install taps** to get **queue times below 30 min**. Towns like Bello, Abidjan and Zuri have a lot of people using shared taps, so we will send out teams to those towns first.
4. **Shared taps** with **short queue** times (< 30 min) represent a logistical challenge to further reduce waiting times. The most effective solution, installing taps in homes, is resource-intensive and better suited as a long-term goal.
5. Addressing **broken infrastructure** offers a significant impact even with just a single intervention. It is expensive to fix, but so many people can benefit from repairing one facility. For example, fixing a reservoir or pipe that multiple taps are connected to. We identified towns like Amina, Lusaka, Zuri, Djenne and rural parts of Amanzi seem to be good places to start.

10:37

A practical plan

10:40

Our final goal is to implement our plan in the database.
We have a plan to improve the water access in Maji Ndogo, so we need to think it through, and as our final task, create a table where our teams have the information they need to fix, upgrade and repair water sources. They will need the addresses of the places they should visit (street address, town, province), the type of water source they should improve, and what should be done to improve it.
We should also make space for them in the database to update us on their progress. We need to know if the repair is complete, and the date it was completed, and give them space to upgrade the sources. Let's call this table `Project_progress`.

10:42

This query creates the Project_progress table:

```
CREATE TABLE Project_progress (

    Project_id SERIAL PRIMARY KEY,
    /* Project_id -- Unique key for sources in case we visit the same
                     source more than once in the future.
    */
    source_id VARCHAR(20) NOT NULL REFERENCES water_source(source_id) ON DELETE CASCADE ON UPDATE CASCADE,
    /* source_id -- Each of the sources we want to improve should exist,
                    and should refer to the source table. This ensures data integrity.
    */
    Address VARCHAR(50), -- Street address
    Town VARCHAR(30),
    Province VARCHAR(30),
    Source_type VARCHAR(50),
    Improvement VARCHAR(50), -- What the engineers should do at that place
    Source_status VARCHAR(50) DEFAULT 'Backlog' CHECK (Source_status IN ('Backlog', 'In progress', 'Complete')),
        /* Source_status -- We want to limit the type of information engineers can give us, so we
        limit Source_status.
            - By DEFAULT all projects are in the "Backlog" which is like a TODO list.
            - CHECK() ensures only those three options will be accepted. This helps to maintain clean data.
    */
    Date_of_completion DATE, -- Engineers will add this the day the source has been upgraded.
    Comments TEXT -- Engineers can leave comments. We use a TEXT type that has no limit on char length
);
```

10:48

Here is a less commented one so it is easier to see how we design the `Project_progress` table:

```
CREATE TABLE Project_progress (

    Project_id SERIAL PRIMARY KEY,
    source_id VARCHAR(20) NOT NULL REFERENCES water_source(source_id) ON DELETE CASCADE ON UPDATE CASCADE,
    Address VARCHAR(50),
    Town VARCHAR(30),
    Province VARCHAR(30),
    Source_type VARCHAR(50),
    Improvement VARCHAR(50),
    Source_status VARCHAR(50) DEFAULT 'Backlog' CHECK (Source_status IN ('Backlog', 'In progress', 'Complete')),
    Date_of_completion DATE,
    Comments TEXT
);
```

10:49

Run this query, and then we are going to build the query we need to add the data in there.

10:51

Chidi Kunto
Online

At a high level, the `Improvements` are as follows:

1. `Rivers` → `Drill wells`
2. `wells`: if the well is contaminated with chemicals → `Install RO filter`
3. `wells`: if the well is contaminated with biological contaminants → `Install UV and RO filter`
4. `shared_taps`: if the queue is longer than 30 min (30 min and above) → `Install X taps` nearby where X number of taps is calculated using `X = FLOOR(time_in_queue / 30)`.
5. `tap_in_home_broken` → `Diagnose local infrastructure`

10:56

Can you see that for wells and shared taps we have some `IF` logic, so we should be thinking `CASE` functions! Let's take the various `Improvements` one by one, then combine them into one query at the end.

11:03

To make this simpler, we can start with this query:

```
-- Project_progress_query

SELECT
    location.address,
    location.town_name,
    location.province_name,
    water_source.source_id,
    water_source.type_of_water_source,
    well_pollution.results
FROM
    water_source
LEFT JOIN
    well_pollution ON water_source.source_id = well_pollution.source_id
INNER JOIN
    visits ON water_source.source_id = visits.source_id
INNER JOIN
    location ON location.location_id = visits.location_id
```

11:04

It joins the `location`, `visits`, and `well_pollution` tables to the `water_source` table. Since `well_pollution` only has data for wells, we have to join those records to the water_source table with a LEFT JOIN and we used `visits` to link the various id's together.

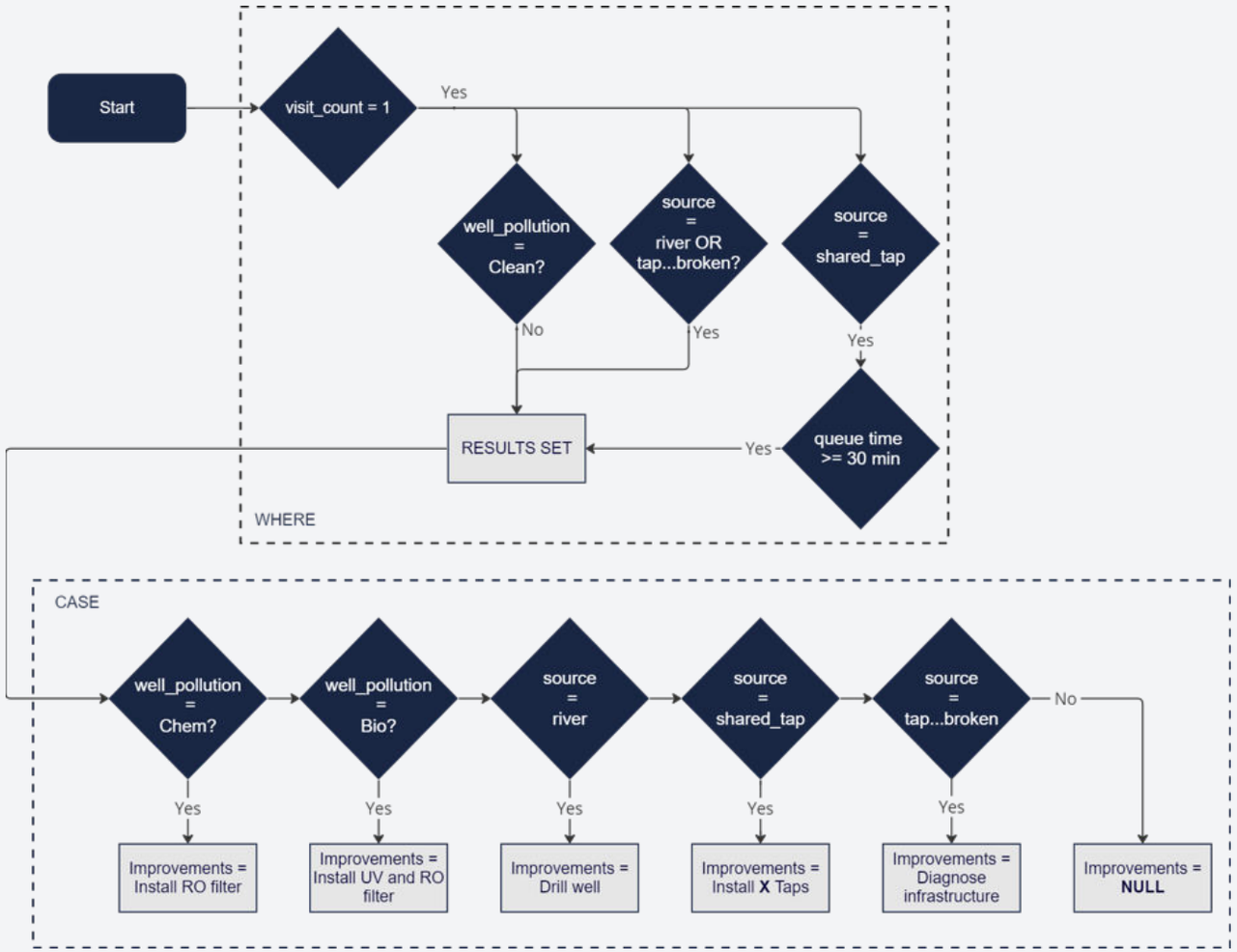
11:05

First things first, let's filter the data to only contain sources we want to improve by thinking through the logic first.

1. Only records with visit_count = 1 are allowed.
2. Any of the following rows can be included:
    a. Where shared taps have queue times over 30 min.
    b. Only wells that are contaminated are allowed -- So we exclude wells that are `Clean`
    c. Include any `river` and `tap_in_home_broken` sources.

11:06

Visually:



11:07 32

Note that I split up the logic into the WHERE and CASE clauses. While this makes the logic more complex to follow, my thinking is to remove all of the data we don't need first, like records with visit_count > 1, and then do calculations.

11:09

So, lets start with the WHERE section:

```
...
WHERE
    visits.visit_count = 1 -- This must always be true
    AND ( -- AND one of the following (OR) options must be true as well.
        ... != 'Clean'
        OR ... IN ('tap_in_home_broken','...')
        OR (... = 'shared_tap' AND ...)
    )
```

11:13

Fill in the blanks in the `WHERE` filter, and add it to `Project_progress_query`.

11:15

You should get 25398 rows of data.

11:18

Step 1: Wells
Let's start with wells. Depending on whether they are chemically contaminated, or biologically contaminated — we'll decide on the interventions.

11:27

Use some control flow logic to create `Install UV filter` or `Install RO filter` values in the `Improvement` column where the results of the pollution tests were `Contaminated: Biological` and `Contaminated: Chemical` respectively. Think about the data you'll need, and which table to find it in. Use `ELSE NULL` for the final alternative.

11:33

If you did it right, there should be `Install RO filter` and `Install UV and RO filter` values in the `Improvements` column now, and lots of `NULL` values.

11:42

Step 2: Rivers
Now for the rivers. We upgrade those by drilling new wells nearby.

11:44

Add `Drill well` to the `Improvements` column for all river sources.

11:47

Check your records to make sure you see `Drill well` for `river` sources.

11:50

Chidi Kunto
Online

Step 3: Shared taps
Next up, shared taps. We need to install one tap near each shared tap for every 30 min of queue time.
This is my logic:

```
CASE
    ...
    WHEN type_of_water_source = ... AND ... THEN CONCAT("Install ", FLOOR(...), " taps nearby")
ELSE NULL
```

I am using `FLOOR()` here because I want to round the calculation down. Say the queue time is 45 min. The result of `45/30 = 1.5`, which could round up to 2. We only want to install a second tap if the queue is > 60 min. Using `FLOOR()` will round down everything below 59 mins to one extra tap, and if the queue is 60 min, we will install two taps, and so on.

11:51

Use this code, and fill in the blanks to update the `Improvement` column for shared_taps with long queue times.

11:57

Check to make sure you're getting `Installed x taps` values in the `Improvement` column.

12:05

Step 4: In-home taps
Lastly, let's look at in-home taps, specifically broken ones. These taps indicate broken infrastructure. So these need to be inspected by our engineers.

12:11

Chidi Kunto
Online

Add a case statement to our query updating broken taps to `Diagnose local infrastructure`.

12:12

So our final query should now return 25398 rows of data, with rivers, various wells, shared taps and broken taps flagged for improvement, and importantly, no `NULL` values!

12:20

Step 6: Add the data to Project_progress

12:24

Now that we have the data we want to provide to engineers, populate the `Project_progress` table with the results of our query.

HINT: Make sure the columns in the query line up with the columns in `Project_progress`. If you make any mistakes, just use `DROP TABLE project_progress`, and run your query again.

12:26

There we go, all done! Now we send off our summary report to Pres. Naledi with our main findings, so they can start organising the teams. We'll also explain the Project_progress table, and how this will help us track our progress.

12:30

Chidi Kunto
Online

Finally, thank you for sticking with me through this project. I know there were some tough times in this project; Window Functions, JOINS, and even corruption! I'm so glad you struggled through it. The Academy does its best to show you how SQL works, but it is only when you start solving problems like this that you truly understand how to use this tool to answer data questions.

12:31

I heard you are meeting up with our visualisation expert soon, Dalila. She mentored me when I joined the team, so I'm sure you will learn a lot from her!

12:37

My friend, Pula! In Maji Ndogo, it means "rain" and signifies blessings and prosperity.

12:43

I hope we talk soon.
Take care! 🚀

12:47