EXPLORE AI ACADEMY

Maji Ndogo: From analysis to action

Charting the course for Maji Ndogo's water future

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

Introduction
Starting the final journey

1 **Joining pieces together**
Finding the data we need across tables

2 **The last analysis**
Finding the final insights from our data

3 **Summary report**
Sharing our knowledge with decision makers

4 **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

Chidi Kunto
Online

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

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:

```
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

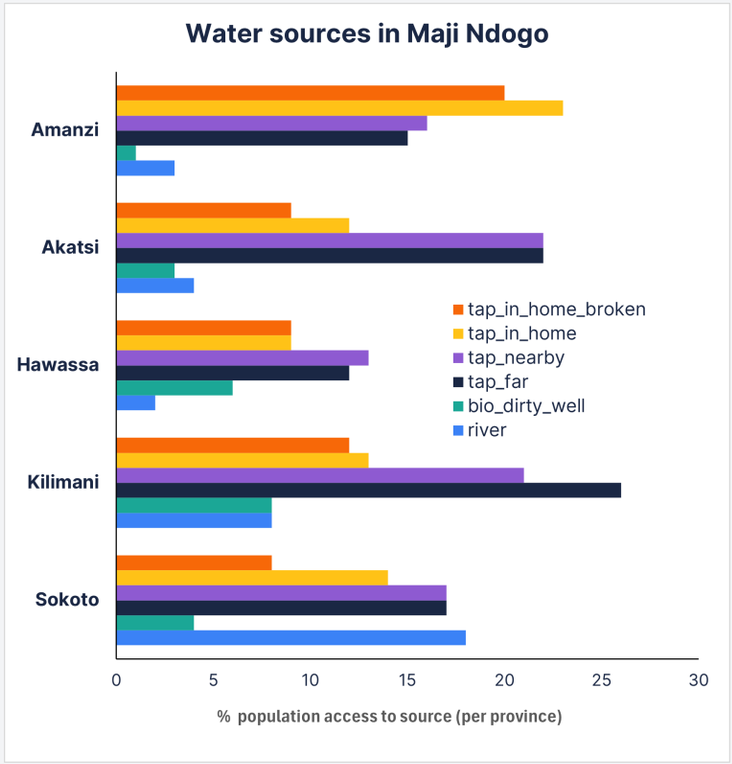
Run the query and see if you can spot any of the following patterns:

- 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

Chidi Kunto
Online

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

Chidi Kunto
Online

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

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

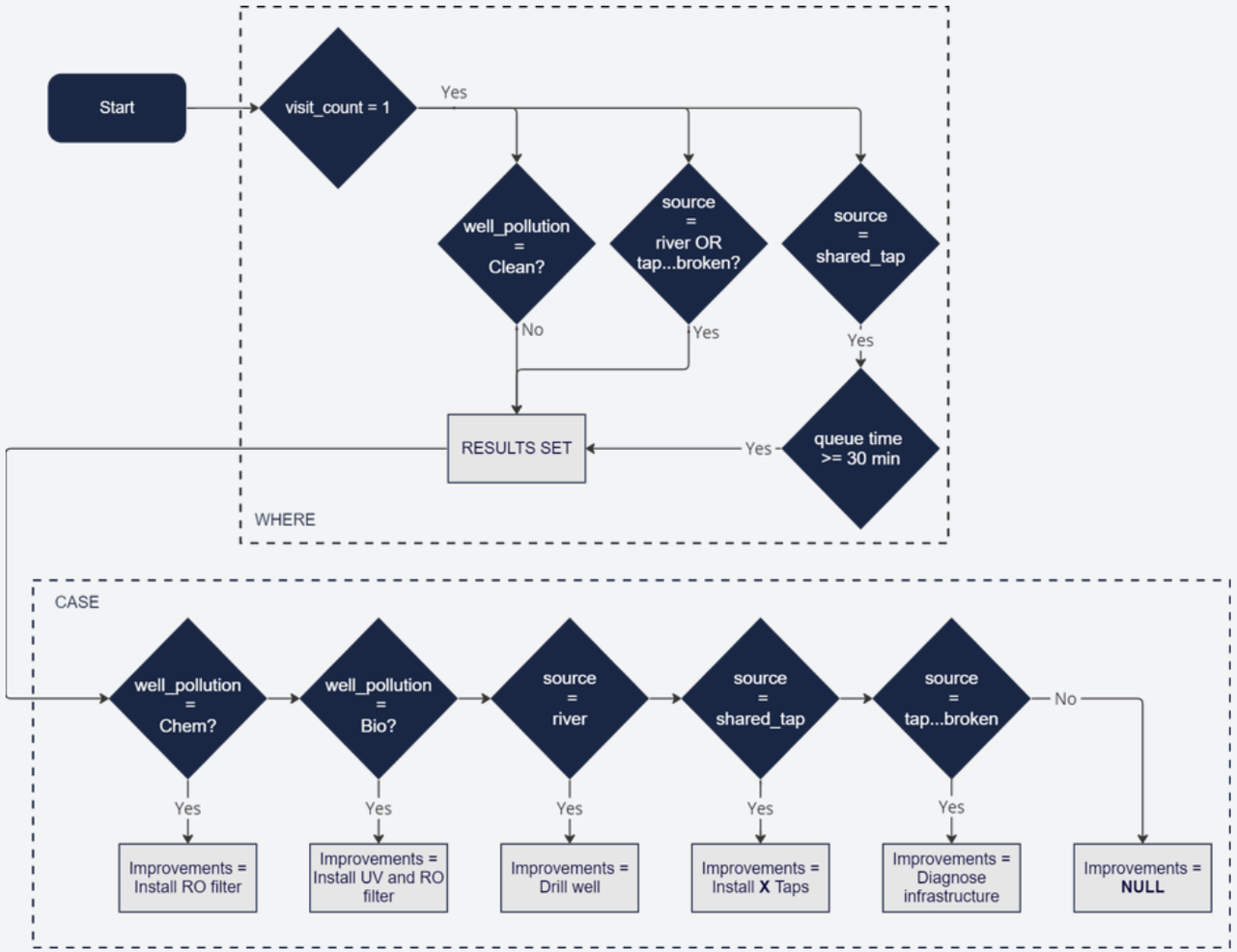
Chidi Kunto
Online

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

Chidi Kunto
Online

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

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