M.A.N.U

3/9/2023

Mathematics Association of Nairobi University
isaak@students.uonbi.ac.ke

The `SQL` `AND`, `OR` and `NOT` Operators

- The `WHERE` clause can be combined with `AND`, `OR`, and `NOT` operators.
- The `AND` and `OR` operators are used to filter records based on more than one condition
- The `AND` operator displays a record if all the conditions separated by `AND` are `TRUE`.
- The `OR` operator displays a record if any of the conditions separated by `OR` is `TRUE`.
- The `NOT` operator displays a record if the condition(s) is `NOT TRUE`.

`AND`

Example: Select sub counties in nairobi county which have less than 200000 total population

In [ ]:
```sql
SELECT county_name, subcounty_name, total
FROM subcounty_population_density
WHERE county_name = 'Nairobi' AND total < 200000;
```

`OR`

Example: The following SQL statement selects all fields from "subcounty_population_density" where county_name is "Nairobi" or "Mombasa"

In [ ]:
```sql
SELECT county_name, subcounty_name
FROM subcounty_population_density
WHERE county_name = 'Nairobi' OR county_name = 'Mombasa';
```

`NOT`

Example: The following SQL statement selects all fields from "subcounty_population_density" where the county_name is not Nairobi

In [ ]:
```sql
SELECT county_name, subcounty_name, total
FROM subcounty_population_density
WHERE NOT county_name='Nairobi';
```

Combining **`AND`**, **`OR`** and **`NOT`**

You can also combine the `AND`, `OR` and `NOT` operators.

The following statement selects all fields from `subcounty_population_density` that are in `Nairobi` or `Mombasa` county which have a population density of more than 10000

In [ ]:
```sql
SELECT county_name, subcounty_name, total, pop_density
FROM subcounty_population_density
WHERE (county_name = 'Nairobi' OR county_name = 'Mombasa') AND pop_density > 10000;
```

`BETWEEN`

MySQL "BETWEEN" operator to determine whether a value is in a range of values.

In [ ]:
```sql
SELECT county_name, subcounty_name, square_kms
FROM subcounty_population_density
WHERE square_kms BETWEEN 1000 AND 10000;
```

`IS NULL`

Show the number rows that are missing/NULL values for square_kms column

In [ ]:
```
SELECT county_name, subcounty_name, square_kms
FROM subcounty_population_density
WHERE square_kms IS NULL;
```

Note: IS NULL is different from = `0`

In [ ]:
```
SELECT county_name, subcounty_name, square_kms
FROM subcounty_population_density
WHERE square_kms = 0;
```

LIKE and Wildcards

The `LIKE` operator is a logical operator that tests whether a string contains a specified pattern or not.

`MySQL` provides two wildcard characters for constructing patterns:

- The percentage `%` wildcard matches any string of zero or more characters.
- The underscore `_` wildcard matches any string of one character lengths

For example, `s%` matches any string starts with the character s such as `sun` and `six`. The `se_` matches any string starts with se and is followed by any character such as see and sea

In [ ]:
```
SELECT county_name, subcounty_name
FROM subcounty_population_density
WHERE county_name LIKE "N%";
```

In this example, `MySQL` scans the whole `subcounty_population_table` to find subcounties whose `county_name` start with the letter `N` and are followed by any number of characters.

`%y` matches any `county_name` that ends with letter Y.

> Note : The wildcard is case insensitive.

In [ ]:
```
SELECT county_name, subcounty_name
FROM subcounty_population_density
WHERE county_name LIKE "%y";
```

In [ ]:
```
SELECT county_name, subcounty_name
FROM subcounty_population_density
WHERE subcounty_name LIKE "%east%"
OR subcounty_name LIKE "%west%";
```

Example using the underscore `_` wildcard

In [ ]:
```
SELECT county_name, subcounty_name
FROM subcounty_population_density
WHERE county_name LIKE "k_____";
```

This query is used to select rows where the county_name starts with letter k and followed by five letters

Typically, you'll use the `LIKE` operator in the `WHERE` clause of the `SELECT` , `DELETE` , and `UPDATE` statement.

MySQL NOT LIKE

The `MySQL` allows you to combine the NOT operator with the `LIKE` operator to find a string that does not match a specific pattern.

Suppose you want to search for all subcounties in `Kilifi` that do not have the word `Kilifi` in the subcounty name:

In [ ]:
```
SELECT county_name, subcounty_name
FROM subcounty_population_density
WHERE county_name = "kilifi"
AND subcounty_name NOT LIKE "%kilifi%";
```

MySQL `REGEXP`

MySQL `REGEXP` performs a pattern match of a string expression against a pattern. The pattern is supplied as an argument. Regular

Expressions provide a powerful and flexible pattern match that can help us implement power search utilities for our database systems.

Suppose you want to show the sub counties that have the word `east`

In [ ]:
```
SELECT county_name, subcounty_name
FROM subcounty_population_density
WHERE subcounty_name REGEXP "east"
```

In [ ]:
```
SELECT county_name, subcounty_name
FROM subcounty_population_density
WHERE subcounty_name REGEXP "east|west"
```

In [ ]:
```
SELECT county_name, subcounty_name
FROM subcounty_population_density
WHERE subcounty_name REGEXP "east|west|north|south"
```

Updating Values

Updating data is one of the most important tasks when you work with the database.

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

In [ ]:
```
UPDATE subcounty_population_density
SET subcounty_name = "RACHUONYO NORTH"
WHERE subcounty_name = "RACHUONYONORTH"
```

In [ ]:
```
-- Update records for Rachuonyo East
UPDATE subcounty_population_density
SET subcounty_name = "RACHUONYO EAST"
WHERE subcounty_name = "RACHUONYOEAST";

-- Update records for Rachuonyo South
UPDATE subcounty_population_density
SET subcounty_name = "RACHUONYO SOUTH"
WHERE subcounty_name = "RACHUONYOSOUTH";

-- Update records for NYANDARUASOUTH
UPDATE subcounty_population_density
SET subcounty_name = "NYANDARUA SOUTH"
WHERE subcounty_name = "NYANDARUASOUTH";


-- Update records for NYANDARUA central
UPDATE subcounty_population_density
SET subcounty_name = "NYANDARUA CENTRAL"
WHERE subcounty_name = "NYANDARUACENTRAL";

-- Update records for NYANDARUA WEST
UPDATE subcounty_population_density
SET subcounty_name = "NYANDARUA WEST"
WHERE subcounty_name = "NYANDARUAWEST";

-- Update records for NYANDARUA WEST
UPDATE subcounty_population_density
SET subcounty_name = "NYANDARUA WEST"
WHERE subcounty_name = "NYANDARUAWEST";

-- Update records for NYANDARUA WEST
UPDATE subcounty_population_density
SET subcounty_name = "NYANDARUA NORTH"
WHERE subcounty_name = "NYANDARUANORTH";
```

> **Warning: Be careful when updating records. If you omit the WHERE clause, ALL records will be updated!**

Deleting Records

To delete data from a table, you use the MySQL `DELETE` statement.

> **Warning: Notice that the WHERE clause is optional. If you omit the WHERE clause, the DELETE statement will delete all rows in the table.**

Example

Since the sub county `LAKE BARINGO` has no population, we can `DELETE` the row

In [ ]:
```
DELETE FROM subcounty_population_density
WHERE subcounty_name = "LAKE BARINGO";
```

Deleting multiple rows is equally easy. Let's delete all rows where the population is `NULL`

In [ ]:
```
DELETE FROM subcounty_population_density
WHERE total IS NULL;
```

Note that once you delete data, it is gone.

To delete all rows from a table, you use the `DELETE` statement without the `WHERE` clause

In [ ]:
```
DELETE table_name;
```

SQL ALTER TABLE Statement

The `ALTER TABLE` statement is used to `add` , `delete` , or `modify` columns in an existing table.

The `ALTER TABLE` statement is also used to add and drop various constraints on an existing table.

`ALTER TABLE` - `ADD Column`

To add a new column on a table in SQL

In [ ]:
```
ALTER TABLE table_name
ADD column_name datatype;
```

`ALTER TABLE` - `DROP COLUMN`

In [ ]:
```
ALTER TABLE table_name
DROP COLUMN column_name;
```

`ALTER TABLE MODIFY COLUMN`

In [ ]:
```
ALTER TABLE table_name
MODIFY COLUMN column_name datatype;
```

`SQL MIN() and MAX() Functions`

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

In [ ]:
```
SELECT MIN(total) AS least_populated_subcounty
FROM subcounty_population_density;
```

The `MAX()` function returns the biggest value of the selected column

In [ ]:
```
SELECT MAX(pop_density) AS highest_pop_density
FROM subcounty_population_density;
```

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

Use a `GROUP BY` and a `MAX`

In [ ]:
```
SELECT county_name, AVG(pop_density) AS avg_pop_density
FROM subcounty_population_density
GROUP BY county_name
ORDER BY avg_pop_density DESC;
```

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js