Name ___Christopher Holmes_____

Background

One of the strengths of SQL is that the details of how indexes are used and joins are done is determined by the DBMS. This allows the database designer to add new indexes for better performance and programs will automatically begin using them without having to modify the program. Indexes can be changed or dropped and application programs do not require modification.

But how does the database designer know which indexes to add or modify, and would a new index actually be used by the DBMS? What indexes are not used and could be dropped? Having extra unneeded indexes hurts performance of insert/delete/update operations because when data changes, the indexes must also be read and updated.

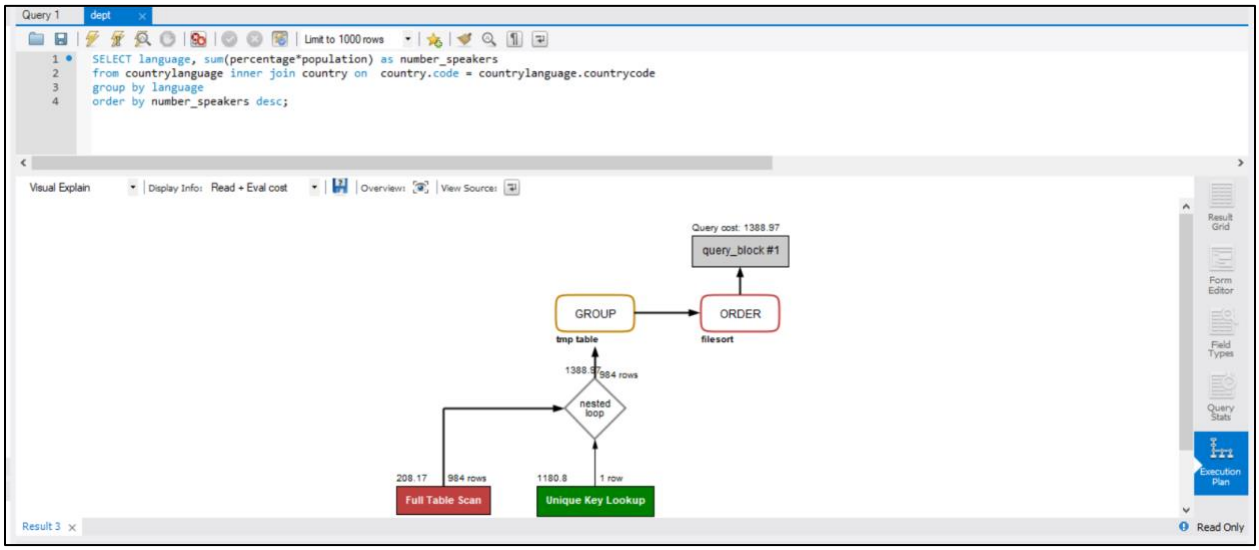
Relational DBMS usually provide some sort of SQL explain capability to shows

- Which indexes are used and how are they accessed
- How are join executed
- Other operations such as sorting that need to be done to get the query result

More information about query plan and explain can be found at

- http://use-the-index-luke.com/
- https://dev.mysql.com/doc/refman/5.7/en/execution-plan-information.html

MySQL Workbench will show the query plan in a graphical format by clicking on the Query Plan button on the right-hand side of the query result window. The plan is read from the bottom to top (the first actions are shown on the bottom and the later actions on top).



The rows of each table involved in the query can be accessed in several ways

- **Table scan** reads the table file sequentially from beginning to end. No index is used.

- **Index key equal** uses the b-tree portion of the index to find the record(s) rba (record byte address) and used the RBA to randomly read the data rows. A table can have multiple indexes and the query plan will indicate which index is being used.
- **Index key range** same as index key equal but then reads additional index records using the linked list structure of the index and randomly retrieves data rows. This type of index access is used if you have the predicate such as " age between 21 and 45" and there is an index on the age column.
- **Index scan** uses the index linked list is read and data rows are retrieved if necessary. This type of access is used if you have a compound index on multiple columns (name, age) and the query has a predicate on age such as "age > 50". The b-tree cannot be used because there is no value given for name. But the index linked list be read to find the RBAs of rows with age > 50. This will generally be faster than doing a table scan because the index linked list is smaller in size than the table file.
- **Index only access** indicates the data required by the query is entirely in the index and no table rows are read. An example would be the query

```
select count(*) from country where code like 'A%'
```

and there is an index on column code. Index only access does not show up in the graphical query plan. But if you click on the SHOW SOURCE button

in the visual explain window, you will see the query plan details in JSON format. Look for the name value pair **"using_index": true**

The DBMS considers how selective an index is (what percentage of data rows are returned) and whether it is faster to retrieve the rows using a table scan or using an index. Using an index often requires many random reads which are slow, while a table scan reads the table in sequence which may be faster if a large portion of the table must be read.

You should understand the difference between a nested loop join and merge join. Note that MySQL does not use merge joins but many other DBMSs do.

See http://use-the-index-luke.com/sql/join about different type of join plans.

Do the following selects involving the WORLD sample database. Study the query plan for each select and answer the questions.

1. Study the query plan for the following two selects.
   a. What index is used in each case?
   b. Why do you think changing the value 110 to 2000 changes the index being used?

b- The index changes when the value changes because it is faster to use the whole table than to do a range based on the number of rows that are in the range.

```
select id from city
```

```
where id between 100 and 110 and population > 8000000;
```

```
   a- Range scan
```

```
select id from city
where id between 100 and 2000 and population > 8000000;
```

```
   a- Full table scan
```

We are interesting in knowing the largest city for each country. Translating this to SQL presents a problem: finding the largest city population for each country can be done with

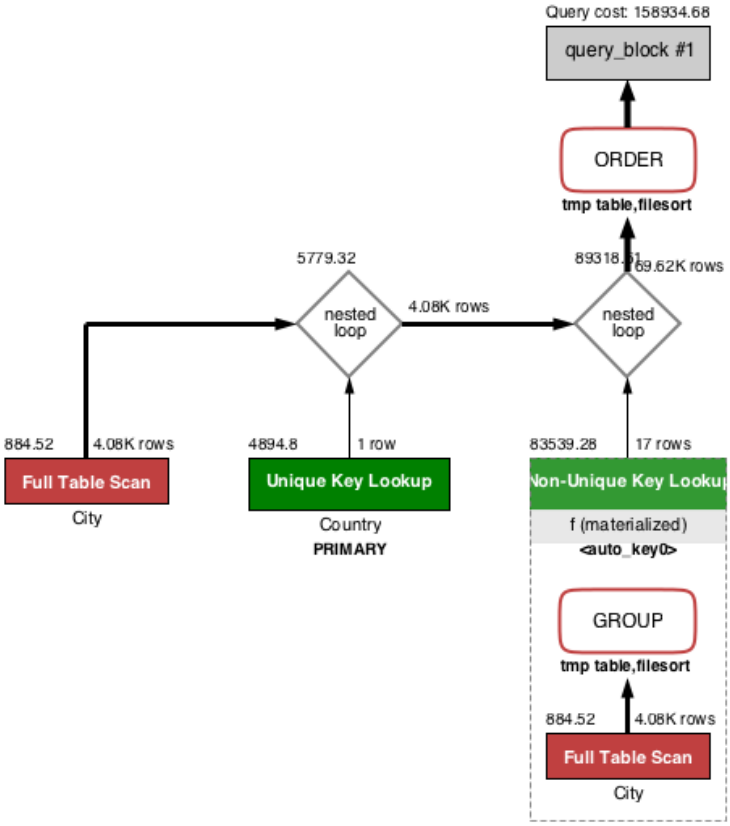
```
select countrycode, max(population)
from city
group by countrycode
```

but we also want to return the city name from the query and sql grouping allows for returning an aggregate value such as max(population), or the grouping column such as countrycode but other column values from the row with maximum value.

There are ways to do this question. One is to the do the grouping as above but then join the result back to the city table using a join predicate on population to get the city name.

```
select country.name, city.name, f.population
from country
     inner join city on country.code=city.countrycode
     inner join ( select countrycode,
                         max(population) population
                  from city
                  group by countrycode) as f
          on city.countrycode=f.countrycode
             and city.population=f.population
order by 1;
```

2. Copy and paste the query plan and explain in your own words what is happening.

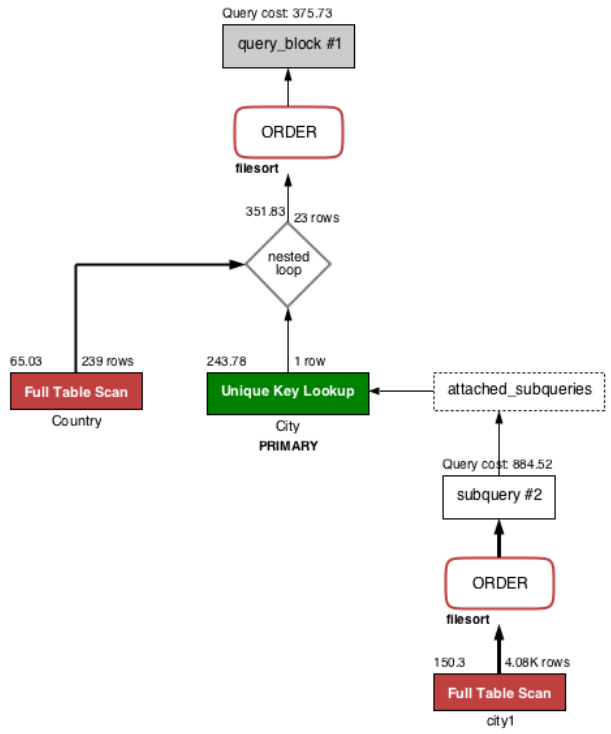


1- First the entire City table is returned with all columns
2- The select statement to return information from the country table with a join to match the information from the city table using the country code
3- There is then the join statement to get the max population of each of the cities in the different countries
4- The final step is ordering it

An alternative way uses a correlated sub-select and the limit keyword to get the most populous city for each country.

```
select country.name, city.name as cityname, city.population
from country inner join city on country.code=city.countrycode
where city.id = (select id from city city1
                 where city1.countrycode = country.Code
                 order by city1.population desc limit 1)
order by 1;
```

3. Copy and paste the query plan along with a short explanation that describes how the query is being processed.

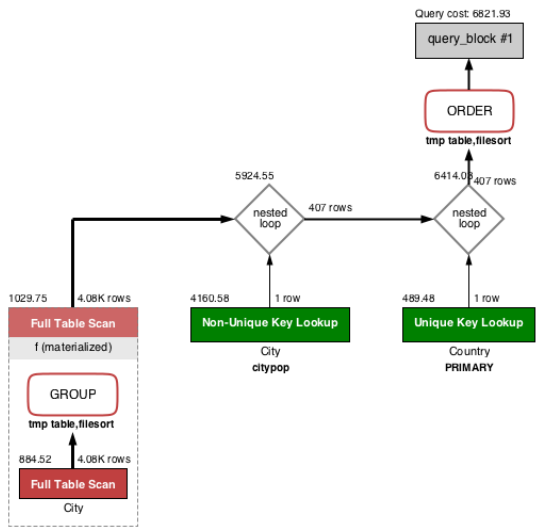


1- The full city name and country name is obtained by joining the city and country table and sorting it by population id.
2- The where clause is used to find the highest populated city by country code and display it in the table of results.

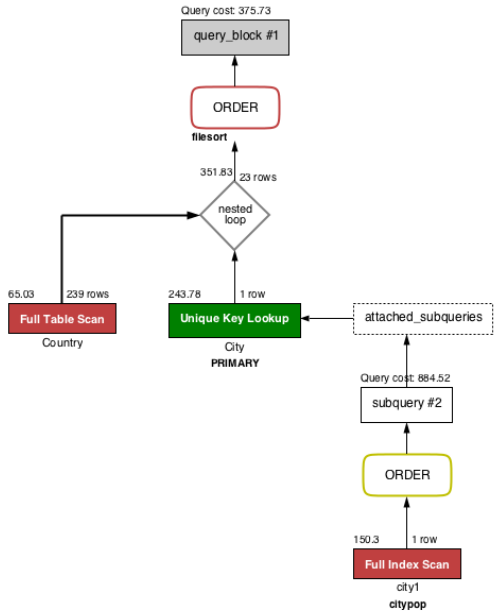
4. Add an index on population to the city table. Refresh the schema and verify the index exists

```
create index  citypop  on city (population);
```

5. Now redo the two queries and look at the new query plans.
   a. Copy and paste the new query plans and write a short explanation.
   b. Is the index on population being used?
   c. The query cost number shown in the plan is an estimated amount of work that must be done to compute the result set. Can you find a better index definition that has a smaller query cost number? What is that definition?



b- The index population is not being used in this query



b- the index population is not used in this query

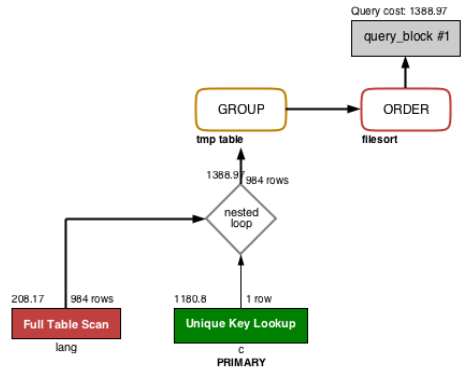
6. For each language, how many people in the world speak that language?

   Do the following query and look at the query plan

```
select language, sum(c.population*percentage) as number_speakers
from country c
```

```
     inner join countrylanguage lang
     on c.code=lang.countrycode
group by language
order by 2 desc;
```

Copy and paste the query plan and in your own words describe how the query is being processed.

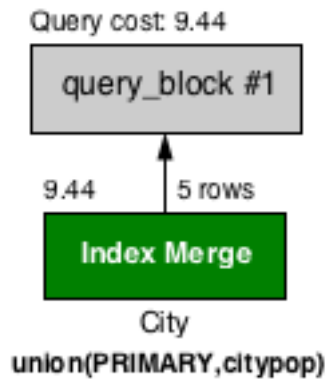


1- This query joins the country table and the country language table.
2- It then loops through adding the number of people for each language so that we can return the number of speakers per language
3- Results are grouped by language, and ordered from highest to lowest

7. Before doing this question, make sure you still have the index on city population.

```
select id from city
where id =100 or population =100000;
```

Look at the query plan for this query. Describe in your words how the query is being executed.



1- The primary index and the citypop index were joined. Because these two indexes had everything that was needed, it was able to find the information that we wanted in only 5 rows.