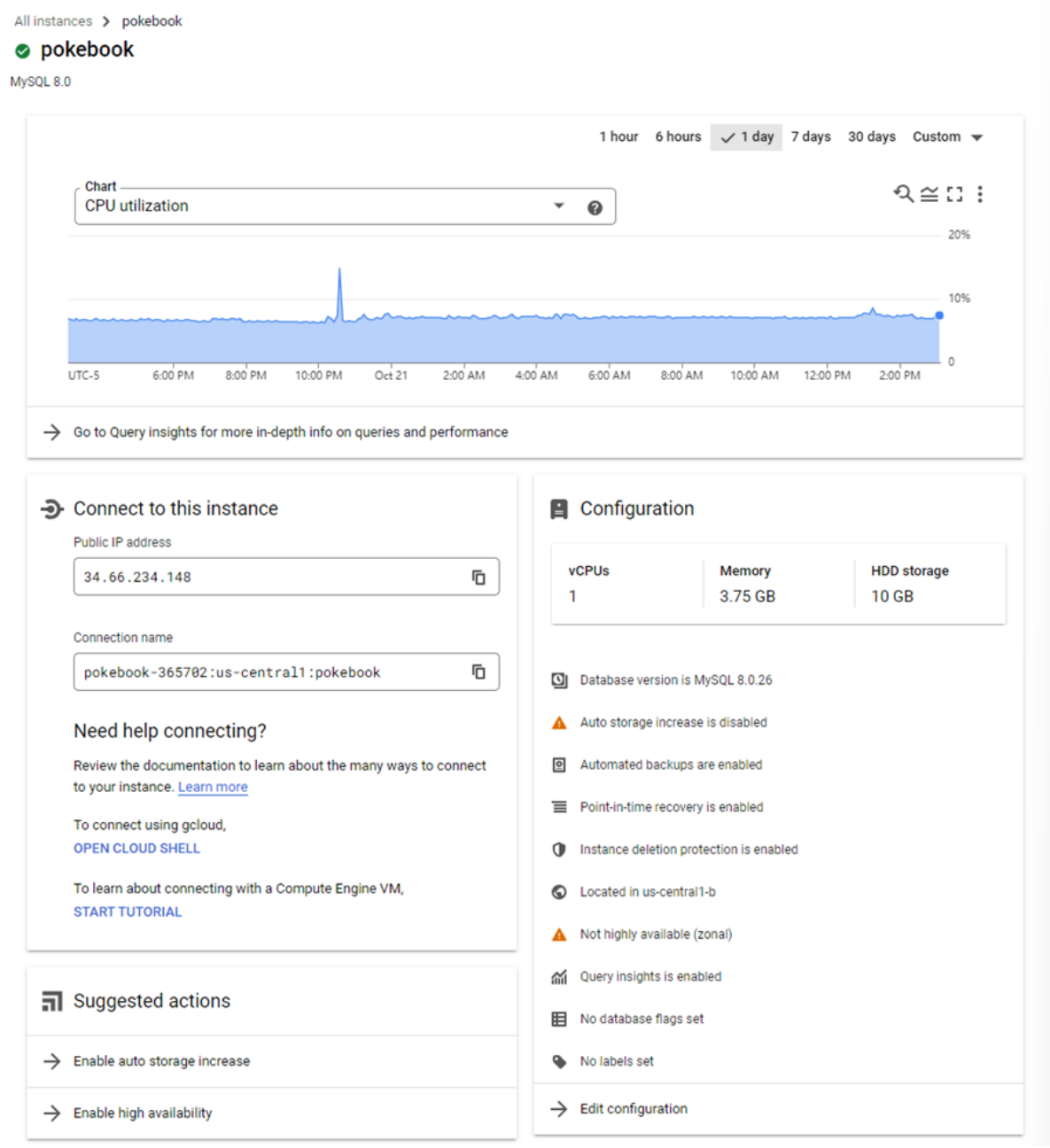
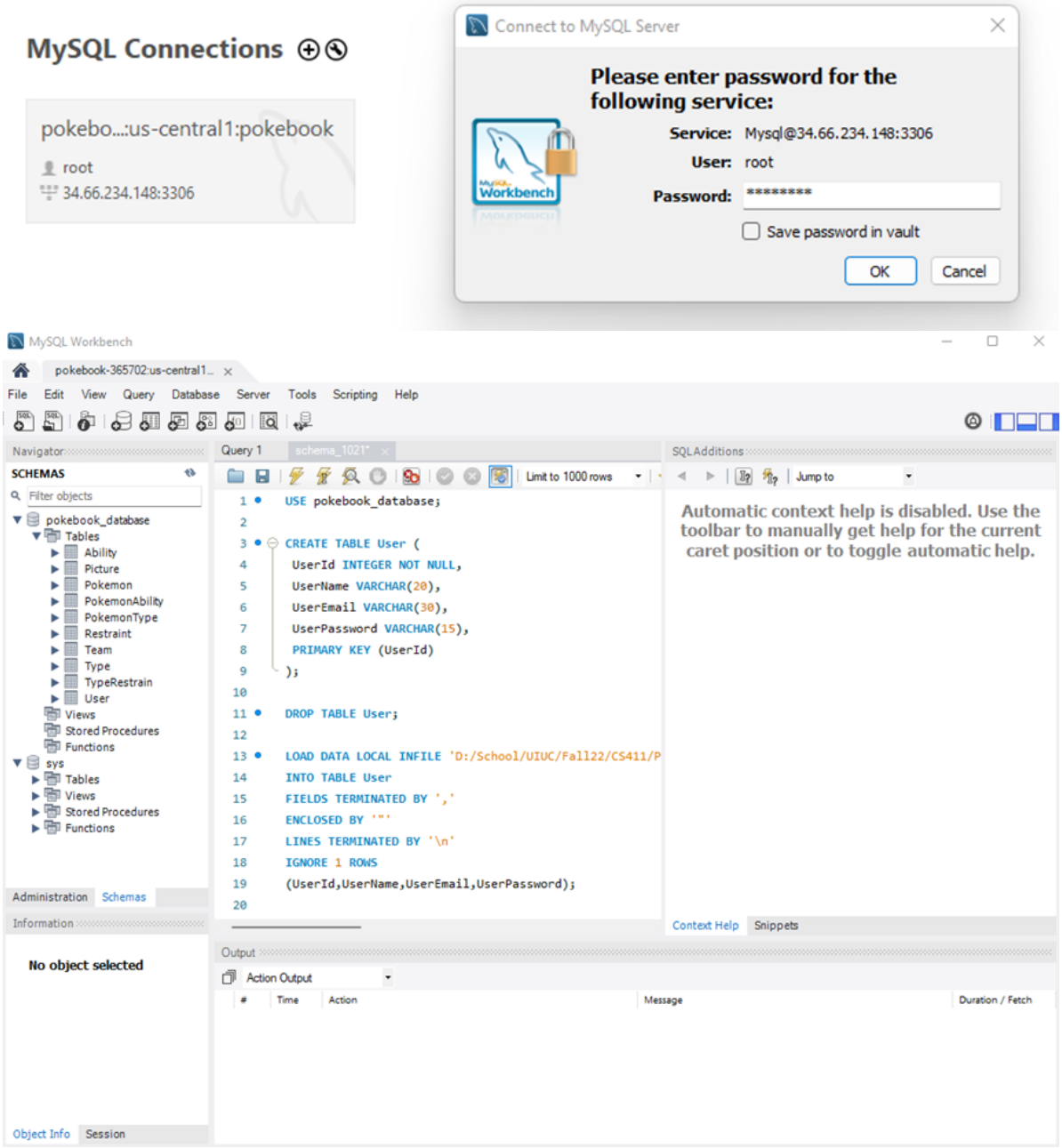
# Database implementation:

## 1. Screenshot of the connection

GCP




MySQL connection:




## 2. DDL Command:

```
USE pokebook_database;

CREATE TABLE User (
 UserId INTEGER NOT NULL,
 UserName VARCHAR(20),
 UserEmail VARCHAR(30),
 UserPassword VARCHAR(15),
 PRIMARY KEY (UserId)
);

CREATE TABLE Pokemon (
 PokemonId INTEGER NOT NULL,
 PokemonName VARCHAR(20),
 Generation INTEGER,
```

```
 Height FLOAT,
 Weight FLOAT,
 Total INTEGER,
 Hp INTEGER,
 Attack INTEGER,
 Defense INTEGER,
 SpecialAttack INTEGER,
 SpecialDefense INTEGER,
 Speed INTEGER,
 PRIMARY KEY (PokemonId)
);


CREATE TABLE Team (
 UserId INTEGER,
 PokemonId INTEGER,
 PRIMARY KEY (UserId,PokemonId)
);

CREATE TABLE Picture (
 PictureId INTEGER NOT NULL,
 PictureType VARCHAR(10),
 PictureLocation VARCHAR(200),
 PokemonId INTEGER,
 PRIMARY KEY (PictureId)
);

CREATE TABLE Ability (
 AbilityId INTEGER NOT NULL,
 AbilityName VARCHAR(20),
 Description VARCHAR(50),
 PRIMARY KEY (AbilityId)
);

CREATE TABLE PokemonAbility (
 PokemonId INTEGER,
 AbilityId INTEGER,
 PRIMARY KEY (PokemonId,AbilityId)
);

CREATE TABLE Type (
 TypeId INTEGER NOT NULL,
 TypeName VARCHAR(15),
 PRIMARY KEY (TypeId)
);

CREATE TABLE PokemonType (
 PokemonId INTEGER NOT NULL,
 FirstTypeId INTEGER,
 SecondTypeId INTEGER,

 PRIMARY KEY (PokemonId)
);

CREATE TABLE Restraint (
```

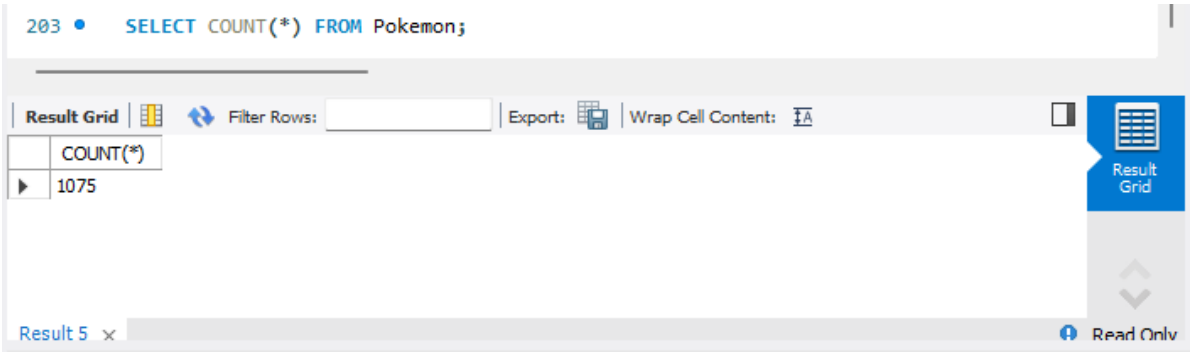
```
 RestraintId INTEGER NOT NULL,
 TheType INTEGER,
 RestrainingType INTEGER,
 PRIMARY KEY (RestraintId)
);

CREATE TABLE TypeRestrain (
 TypeId INTEGER,
 RestraintId INTEGER,
 PRIMARY KEY (TypeId,RestraintId)
);

ALTER TABLE Team ADD CONSTRAINT Team_PokemonId FOREIGN KEY(PokemonId)
REFERENCES Pokemon(PokemonId);

ALTER TABLE Team ADD CONSTRAINT Team_UserId FOREIGN KEY (UserId)
REFERENCES User(UserId);

ALTER TABLE Picture ADD CONSTRAINT Picture_PokemonId FOREIGN KEY(PokemonId)
REFERENCES Pokemon(PokemonId);

ALTER TABLE PokemonAbility ADD CONSTRAINT PokemonAbility_PokemonId FOREIGN
KEY(PokemonId)
REFERENCES Pokemon(PokemonId);

ALTER TABLE PokemonAbility ADD CONSTRAINT PokemonAbility_AbilityId FOREIGN
KEY(AbilityId)
REFERENCES Ability(AbilityId);

ALTER TABLE PokemonType ADD CONSTRAINT PokemonType_PokemonId FOREIGN
KEY(PokemonId)
REFERENCES Pokemon(PokemonId);

ALTER TABLE PokemonType ADD CONSTRAINT PokemonType_FirstTypeId FOREIGN
KEY(FirstTypeId)
REFERENCES Type(TypeId);

ALTER TABLE PokemonType ADD CONSTRAINT PokemonType_SecondTypeId FOREIGN
KEY(SecondTypeId)
REFERENCES Type(TypeId);

ALTER TABLE Restraint ADD CONSTRAINT Restraint_TheType FOREIGN KEY(TheType)
REFERENCES Type(TypeId);

ALTER TABLE Restraint ADD CONSTRAINT Restraint_RestrainingType FOREIGN
KEY(RestrainingType)
REFERENCES Type(TypeId);

ALTER TABLE TypeRestrain ADD CONSTRAINT TypeRestrain_TypeId FOREIGN KEY(TypeId)
REFERENCES Type(TypeId);

ALTER TABLE TypeRestrain ADD CONSTRAINT TypeRestrain_RestraintId FOREIGN
KEY(RestraintId)
REFERENCES Restraint(RestraintId);
```

```
LOAD DATA LOCAL INFILE
'D:/School/UIUC/Fall22/CS411/Project/Stage3/User_data_1.1.csv'
INTO TABLE User
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(UserId,UserName,UserEmail,UserPassword);

LOAD DATA LOCAL INFILE
'D:/School/UIUC/Fall22/CS411/Project/Stage3/Pokemon_data_1.1.csv'
INTO TABLE Pokemon
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(PokemonId,PokemonName,Generation,Height,Weight,Total,Hp,Attack,Defense,SpecialA
ttack,SpecialDefense,Speed);

LOAD DATA LOCAL INFILE
'D:/School/UIUC/Fall22/CS411/Project/Stage3/Abilities_data_1.1.csv'
INTO TABLE Ability
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(AbilityId,AbilityName,Description);

LOAD DATA LOCAL INFILE
'D:/School/UIUC/Fall22/CS411/Project/Stage3/Type_data_1.1.csv'
INTO TABLE Type
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(TypeId,TypeName);

LOAD DATA LOCAL INFILE
'D:/School/UIUC/Fall22/CS411/Project/Stage3/Pokemon_Abilities_data_1.1.csv'
INTO TABLE PokemonAbility
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(PokemonId,AbilityId);

LOAD DATA LOCAL INFILE
'D:/School/UIUC/Fall22/CS411/Project/Stage3/PokemonType_data_1.1.csv'
INTO TABLE PokemonType
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(PokemonId,FirstTypeId,SecondTypeId);
```

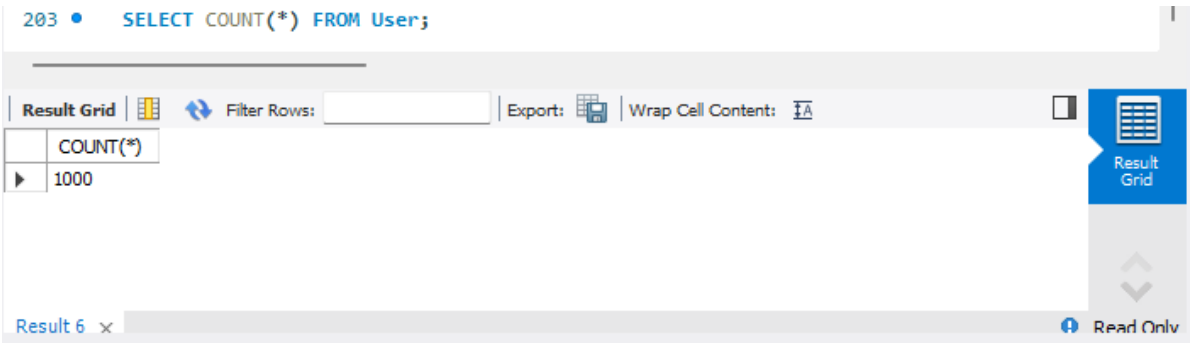
```
LOAD DATA LOCAL INFILE
'D:/School/UIUC/Fall22/CS411/Project/Stage3/Picture_data_1.2.csv'
INTO TABLE Picture
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS
(PictureId,PictureType,PictureLocation,PokemonId);
```

### 3. Count Query :

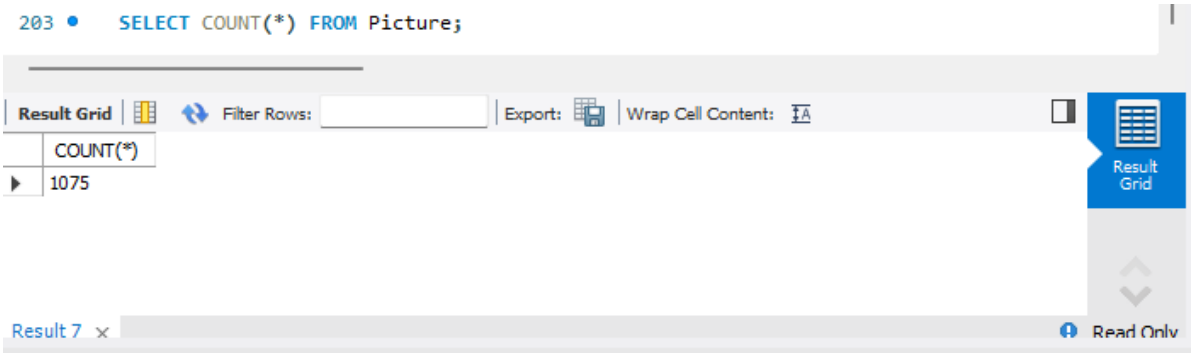
Pokemon Table




User Table




Picture Table




# Advanced Queries:

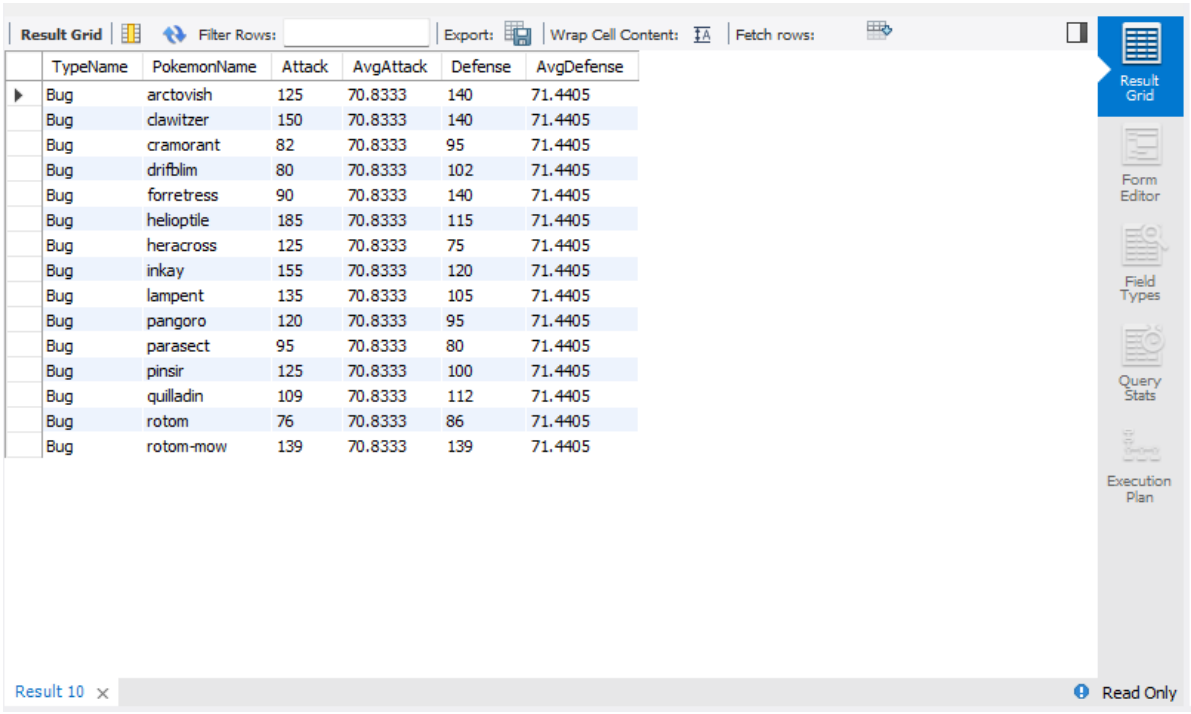
## First Query

Description:

The first query wants Pokémons whose attack and defense value are greater than average attack and average defense for each type. The query returns *TypeName, PokemonName, Attack, AvgAttack, Defense, AvgDefense.*

```
SELECT t1.TypeName, p1.PokemonName, p1.Attack,
      (SELECT AVG(p2.Attack)
                  FROM (Pokemon p2 LEFT JOIN PokemonType pt2 on p2.PokemonId =
pt2.PokemonId) JOIN Type t2 ON (pt2.FirstTypeId = t2.TypeId)
      Where t1.TypeId = t2.TypeId
      GROUP BY t2.TypeId) as AvgAttack,
      p1.Defense,
      (SELECT AVG(p2.Defense)
                  FROM (Pokemon p2 LEFT JOIN PokemonType pt2 on p2.PokemonId =
pt2.PokemonId) JOIN Type t2 ON (pt2.FirstTypeId = t2.TypeId)
      Where t1.TypeId = t2.TypeId
      GROUP BY t2.TypeId) as AvgDefense
FROM (Pokemon p1 LEFT JOIN PokemonType pt1 on p1.PokemonId = pt1.PokemonId) JOIN
Type t1 ON (pt1.FirstTypeId = t1.TypeId)
WHERE p1.Attack > (SELECT AVG(p2.Attack)
                  FROM (Pokemon p2 LEFT JOIN PokemonType pt2 on p2.PokemonId =
pt2.PokemonId) JOIN Type t2 ON (pt2.FirstTypeId = t2.TypeId)
      Where t1.TypeId = t2.TypeId
      GROUP BY t2.TypeId)
      AND
   p1.Defense > (SELECT AVG(p2.Defense)
                  FROM (Pokemon p2 LEFT JOIN PokemonType pt2 on p2.PokemonId =
pt2.PokemonId) JOIN Type t2 ON (pt2.FirstTypeId = t2.TypeId)
      Where t1.TypeId = t2.TypeId
      GROUP BY t2.TypeId)
GROUP BY t1.TypeId, t1.TypeName, p1.PokemonName, p1.Attack, p1.Defense
ORDER BY t1.TypeName, p1.PokemonName
Limit 15;
```

Screenshot:



Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

| TypeName | PokemonName | Attack | AvgAttack | Defense | AvgDefense |
|---|---|---|---|---|---|
| Bug | arctovish | 125 | 70.8333 | 140 | 71.4405 |
| Bug | clawitzer | 150 | 70.8333 | 140 | 71.4405 |
| Bug | cramorant | 82 | 70.8333 | 95 | 71.4405 |
| Bug | drifblim | 80 | 70.8333 | 102 | 71.4405 |
| Bug | forretress | 90 | 70.8333 | 140 | 71.4405 |
| Bug | helioptile | 185 | 70.8333 | 115 | 71.4405 |
| Bug | heracross | 125 | 70.8333 | 75 | 71.4405 |
| Bug | inkay | 155 | 70.8333 | 120 | 71.4405 |
| Bug | lampent | 135 | 70.8333 | 105 | 71.4405 |
| Bug | pangoro | 120 | 70.8333 | 95 | 71.4405 |
| Bug | parasect | 95 | 70.8333 | 80 | 71.4405 |
| Bug | pinsir | 125 | 70.8333 | 100 | 71.4405 |
| Bug | quilladin | 109 | 70.8333 | 112 | 71.4405 |
| Bug | rotom | 76 | 70.8333 | 86 | 71.4405 |
| Bug | rotom-mow | 139 | 70.8333 | 139 | 71.4405 |

Result Grid | Form Editor | Field Types | Query Stats | Execution Plan

Result 10 | Read Only

## Second Query

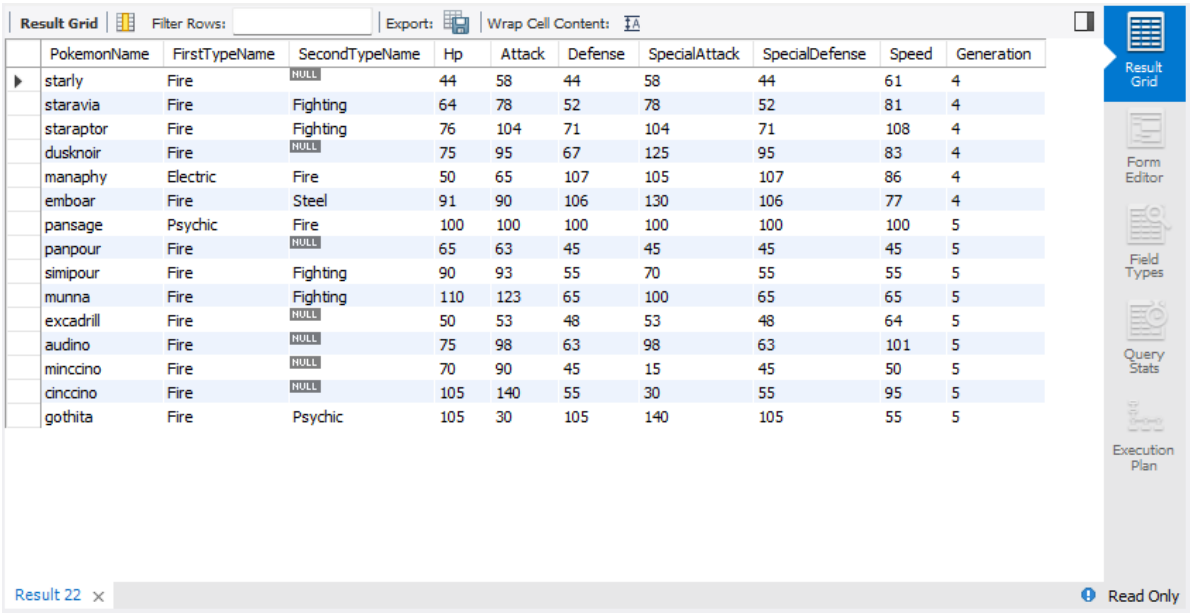
Description:

The second query wants information of Pokémons whose type is fire, water, or grass, and generation is greater or equal to 4. The query returns *PokemonName, FirstTypeName, SecondTypeName, Hp, Attack, Defense, SpecialAttack, SpecialDefense, Speed, Generation.*

```
(SELECT p2.PokemonName, t1.TypeName AS FirstTypeName, t2.TypeName AS
SecondTypeName, p2.Hp, p2.Attack, p2.Defense, p2.SpecialAttack,
p2.SpecialDefense, p2.Speed, p2.Generation
FROM (Pokemon p2 LEFT JOIN PokemonType pt2 on p2.PokemonId = pt2.PokemonId) JOIN
Type t1 ON (pt2.FirstTypeId = t1.TypeId) JOIN Type t2 ON (pt2.SecondTypeId =
t2.TypeId)
WHERE (t1.TypeName like '%Fire%' OR t2.TypeName like '%Fire%') AND p2.Generation
>= 4
ORDER BY p2.PokemonId ASC)

UNION

(SELECT p2.PokemonName, t1.TypeName AS FirstTypeName, t2.TypeName AS
SecondTypeName, p2.Hp, p2.Attack, p2.Defense, p2.SpecialAttack,
p2.SpecialDefense, p2.Speed, p2.Generation
FROM (Pokemon p2 LEFT JOIN PokemonType pt2 on p2.PokemonId = pt2.PokemonId) JOIN
Type t1 ON (pt2.FirstTypeId = t1.TypeId) JOIN Type t2 ON (pt2.SecondTypeId =
t2.TypeId)
WHERE (t1.TypeName like '%Water%' OR t2.TypeName like '%Water%') AND
p2.Generation >= 4
ORDER BY p2.PokemonId ASC)

UNION
```

```
(SELECT p2.PokemonName, t1.TypeName AS FirstTypeName, t2.TypeName AS
SecondTypeName, p2.Hp, p2.Attack, p2.Defense, p2.SpecialAttack,
p2.SpecialDefense, p2.Speed, p2.Generation
FROM (Pokemon p2 LEFT JOIN PokemonType pt2 on p2.PokemonId = pt2.PokemonId) JOIN
Type t1 ON (pt2.FirstTypeId = t1.TypeId) JOIN Type t2 ON (pt2.SecondTypeId =
t2.TypeId)
WHERE (t1.TypeName like '%Grass%' OR t2.TypeName like '%Grass%') AND
p2.Generation >= 4
ORDER BY p2.PokemonId ASC)
LIMIT 15;
```

Screenshot:



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| PokemonName | FirstTypeName | SecondTypeName | Hp | Attack | Defense | SpecialAttack | SpecialDefense | Speed | Generation |
|---|---|---|---|---|---|---|---|---|---|
| starly | Fire | NULL | 44 | 58 | 44 | 58 | 44 | 61 | 4 |
| staravia | Fire | Fighting | 64 | 78 | 52 | 78 | 52 | 81 | 4 |
| staraptor | Fire | Fighting | 76 | 104 | 71 | 104 | 71 | 108 | 4 |
| dusknoir | Fire | NULL | 75 | 95 | 67 | 125 | 95 | 83 | 4 |
| manaphy | Electric | Fire | 50 | 65 | 107 | 105 | 107 | 86 | 4 |
| emboar | Fire | Steel | 91 | 90 | 106 | 130 | 106 | 77 | 4 |
| pansage | Psychic | Fire | 100 | 100 | 100 | 100 | 100 | 100 | 5 |
| panpour | Fire | NULL | 65 | 63 | 45 | 45 | 45 | 45 | 5 |
| simipour | Fire | Fighting | 90 | 93 | 55 | 70 | 55 | 55 | 5 |
| munna | Fire | Fighting | 110 | 123 | 65 | 100 | 65 | 65 | 5 |
| excadrill | Fire | NULL | 50 | 53 | 48 | 53 | 48 | 64 | 5 |
| audino | Fire | NULL | 75 | 98 | 63 | 98 | 63 | 101 | 5 |
| minccino | Fire | NULL | 70 | 90 | 45 | 15 | 45 | 50 | 5 |
| cinccino | Fire | NULL | 105 | 140 | 55 | 30 | 55 | 95 | 5 |
| gothita | Fire | Psychic | 105 | 30 | 105 | 140 | 105 | 55 | 5 |



(*: SecondTypeId could be null because not every Pokemon has two types.)

# Indexing Analysis:

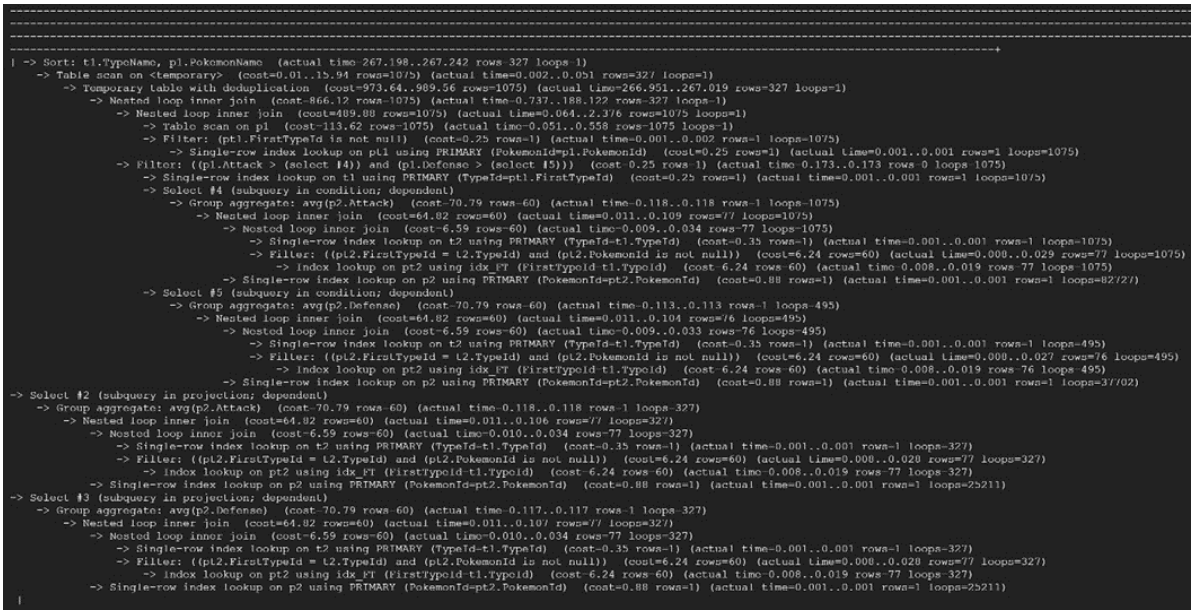
## First Query

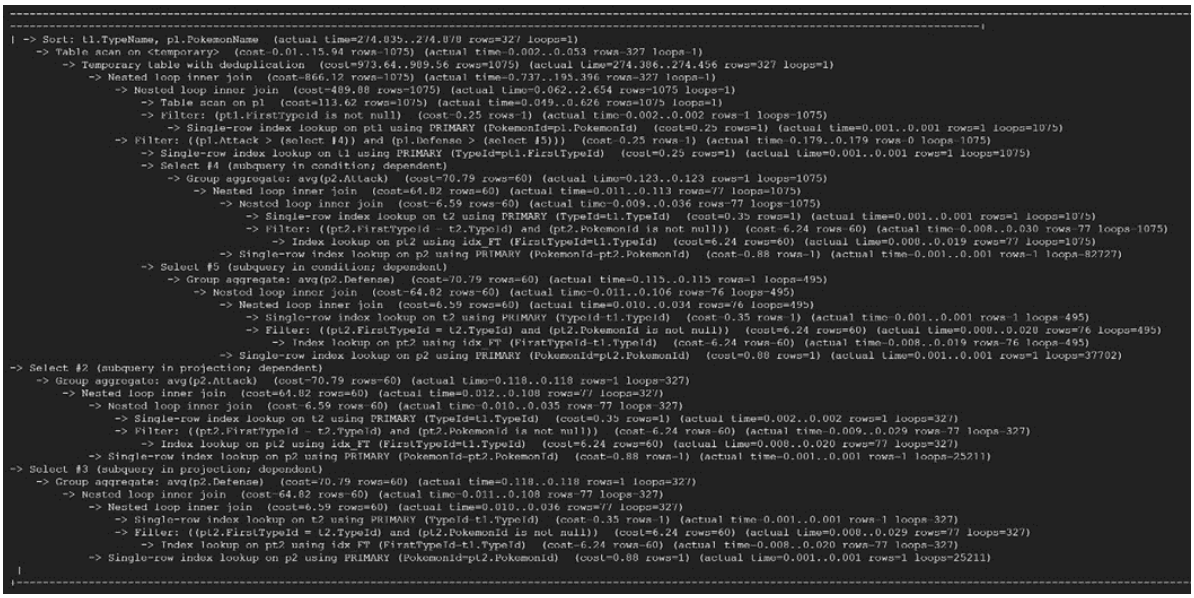
```
SELECT t1.TypeName, p1.PokemonName, p1.Attack,
      (SELECT AVG(p2.Attack)
                  FROM (Pokemon p2 LEFT JOIN PokemonType pt2 on p2.PokemonId =
pt2.PokemonId) JOIN Type t2 ON (pt2.FirstTypeId = t2.TypeId)
      Where t1.TypeId = t2.TypeId
      GROUP BY t2.TypeId) as AvgAttack,
      p1.Defense,
      (SELECT AVG(p2.Defense)
                  FROM (Pokemon p2 LEFT JOIN PokemonType pt2 on p2.PokemonId =
pt2.PokemonId) JOIN Type t2 ON (pt2.FirstTypeId = t2.TypeId)
      Where t1.TypeId = t2.TypeId
      GROUP BY t2.TypeId) as AvgDefense
FROM (Pokemon p1 LEFT JOIN PokemonType pt1 on p1.PokemonId = pt1.PokemonId) JOIN
Type t1 ON (pt1.FirstTypeId = t1.TypeId)
WHERE p1.Attack > (SELECT AVG(p2.Attack)
                  FROM (Pokemon p2 LEFT JOIN PokemonType pt2 on p2.PokemonId =
pt2.PokemonId) JOIN Type t2 ON (pt2.FirstTypeId = t2.TypeId)
```

```
        Where t1.TypeId = t2.TypeId
        GROUP BY t2.TypeId)
        AND
    p1.Defense > (SELECT AVG(p2.Defense)
                  FROM (Pokemon p2 LEFT JOIN PokemonType pt2 on p2.PokemonId =
pt2.PokemonId) JOIN Type t2 ON (pt2.FirstTypeId = t2.TypeId)
        Where t1.TypeId = t2.TypeId
        GROUP BY t2.TypeId)
GROUP BY t1.TypeId, t1.TypeName, p1.PokemonName, p1.Attack, p1.Defense
ORDER BY t1.TypeName, p1.PokemonName
Limit 15;
```

**First try of indexing design:**

Index on **Attack**



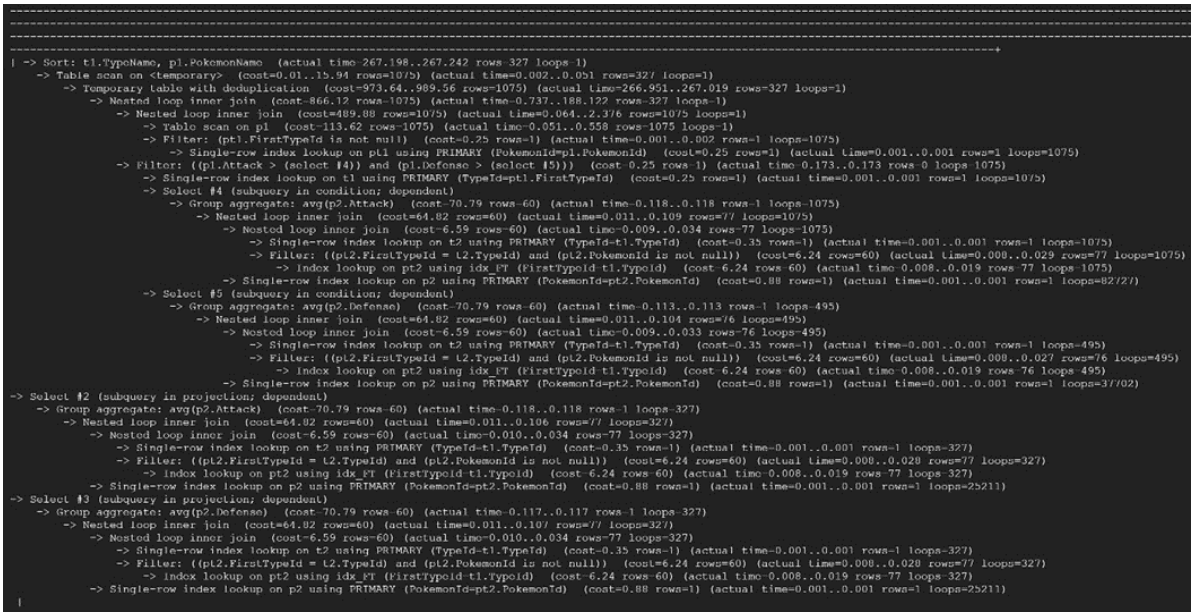

Since we wish to filter Pokemon with higher-than-average attack and defense, we have decided to create an index on Attack. However, the results are quite unsatisfactory, the running time became worse. The main reason is possibly that when we have attack as index, we are distorting the original data structure and make it more redundant.

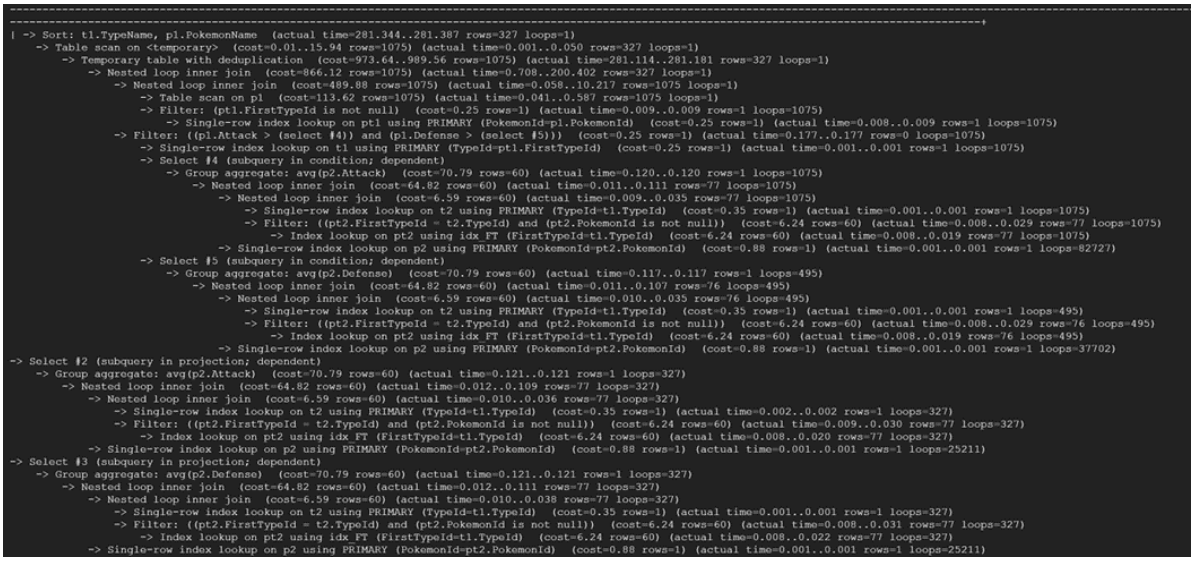
Running time before indexing: 267.198, 0.118, 0.117

Running time after indexing: 274.386, 0.118, 0.117

**Second try of indexing design:**

Index on TypeName



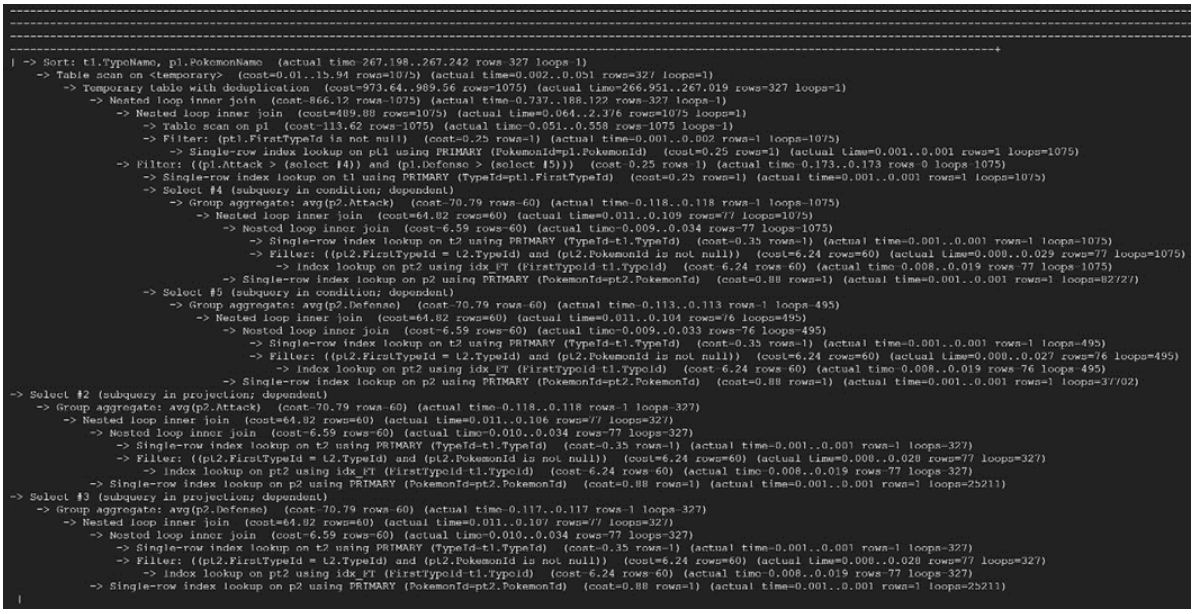

We are trying to optimize performance by putting types of Pokemons into category since it might cost less to traverse between data points. However, the truth turns out to be otherwise and none of the results are better. In addition to distorting original data structure, we consider there is another reason: the index type is B tree and sorting the data and there might be too much rotation and splits on the tree in order to maintain the new index.

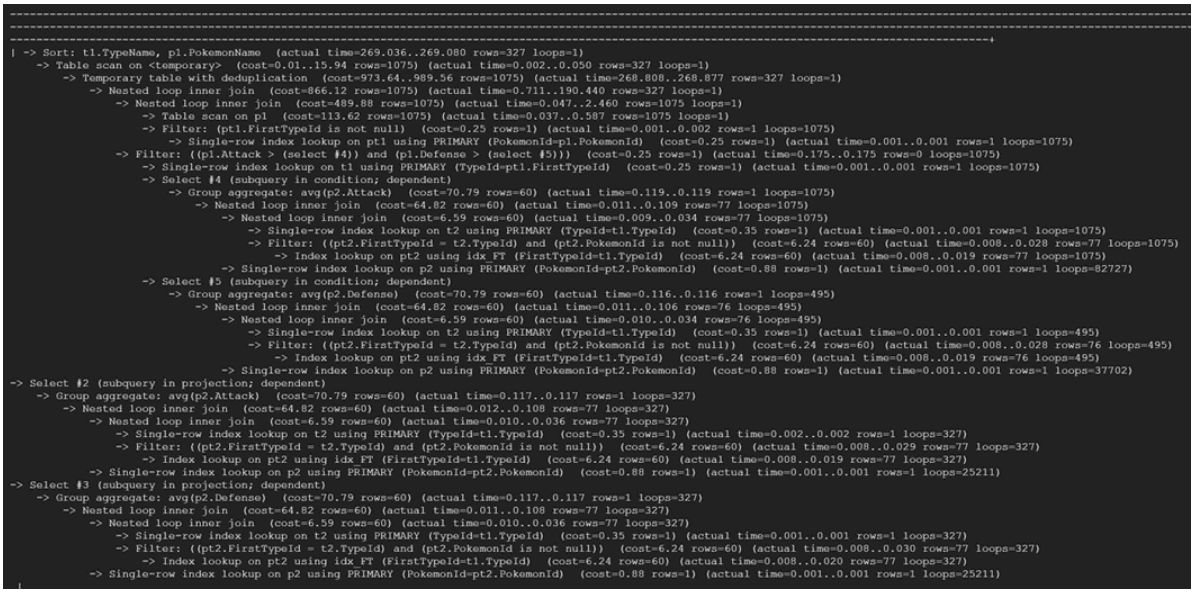
Running time before indexing: 267.198, 0.118, 0.117

Running time after indexing: 281.334, 0.121, 0.121

**Third try of indexing design:**

Index on Generation




We have decided to choose generation as another index because as the Pokemon evolves, their abilities are getting better (and their type remains the same). As the data shows, there are slight improvements in terms of some of the nested loops joins. Overall, the performance is similar, and we think that our premises might not be entirely correct. Different generations of Pokemon does not mean they are getting better, it's more of an adjustment rather than evolution.

Running time before indexing: 267.198, 0.118, 0.117

Running time after indexing: 269.036, 0.117, 0.117

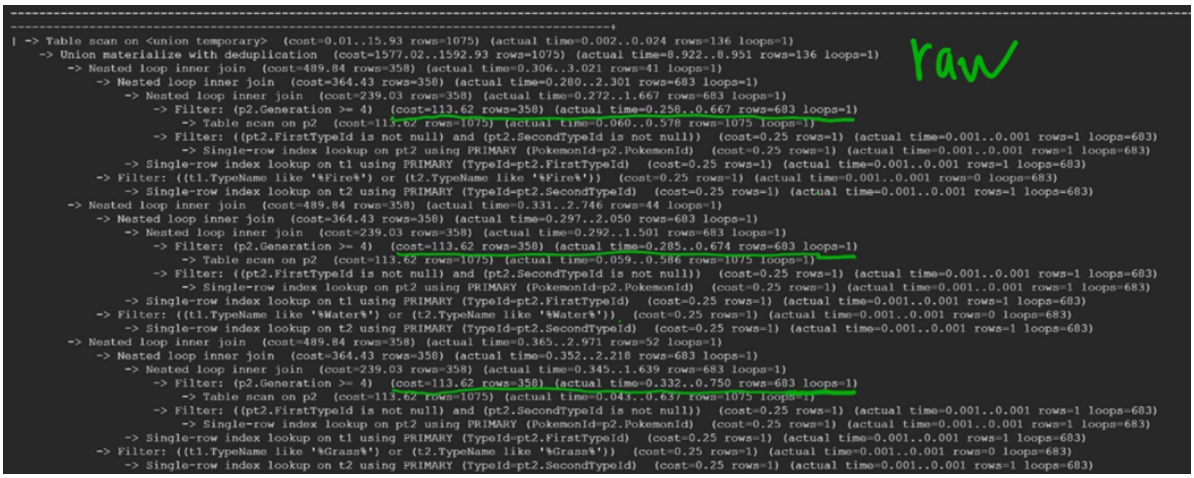
## Second Query

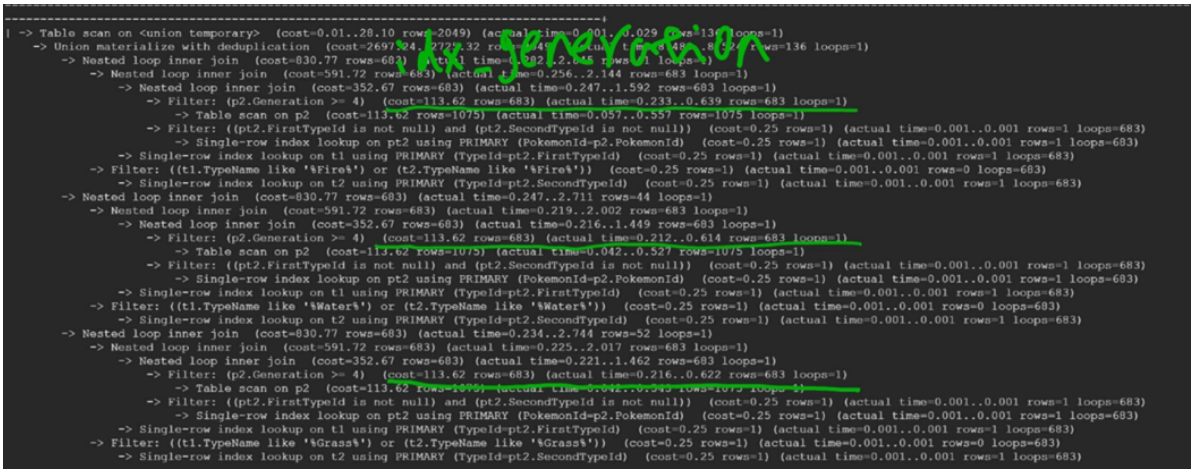
```
(SELECT p2.PokemonName, t1.TypeName AS FirstTypeName, t2.TypeName AS
SecondTypeName, p2.Hp, p2.Attack, p2.Defense, p2.SpecialAttack,
p2.SpecialDefense, p2.Speed, p2.Generation
```

```
FROM (Pokemon p2 LEFT JOIN PokemonType pt2 on p2.PokemonId = pt2.PokemonId) JOIN
Type t1 ON (pt2.FirstTypeId = t1.TypeId) JOIN Type t2 ON (pt2.SecondTypeId =
t2.TypeId)
WHERE (t1.TypeName like '%Fire%' OR t2.TypeName like '%Fire%') AND p2.Generation
>= 4
ORDER BY p2.PokemonId ASC)

UNION

(SELECT p2.PokemonName, t1.TypeName AS FirstTypeName, t2.TypeName AS
SecondTypeName, p2.Hp, p2.Attack, p2.Defense, p2.SpecialAttack,
p2.SpecialDefense, p2.Speed, p2.Generation
FROM (Pokemon p2 LEFT JOIN PokemonType pt2 on p2.PokemonId = pt2.PokemonId) JOIN
Type t1 ON (pt2.FirstTypeId = t1.TypeId) JOIN Type t2 ON (pt2.SecondTypeId =
t2.TypeId)
WHERE (t1.TypeName like '%Water%' OR t2.TypeName like '%Water%') AND
p2.Generation >= 4
ORDER BY p2.PokemonId ASC)

UNION

(SELECT p2.PokemonName, t1.TypeName AS FirstTypeName, t2.TypeName AS
SecondTypeName, p2.Hp, p2.Attack, p2.Defense, p2.SpecialAttack,
p2.SpecialDefense, p2.Speed, p2.Generation
FROM (Pokemon p2 LEFT JOIN PokemonType pt2 on p2.PokemonId = pt2.PokemonId) JOIN
Type t1 ON (pt2.FirstTypeId = t1.TypeId) JOIN Type t2 ON (pt2.SecondTypeId =
t2.TypeId)
WHERE (t1.TypeName like '%Grass%' OR t2.TypeName like '%Grass%') AND
p2.Generation >= 4
ORDER BY p2.PokemonId ASC)
LIMIT 15;
```

**First try of indexing design:**

Index on **Generation**





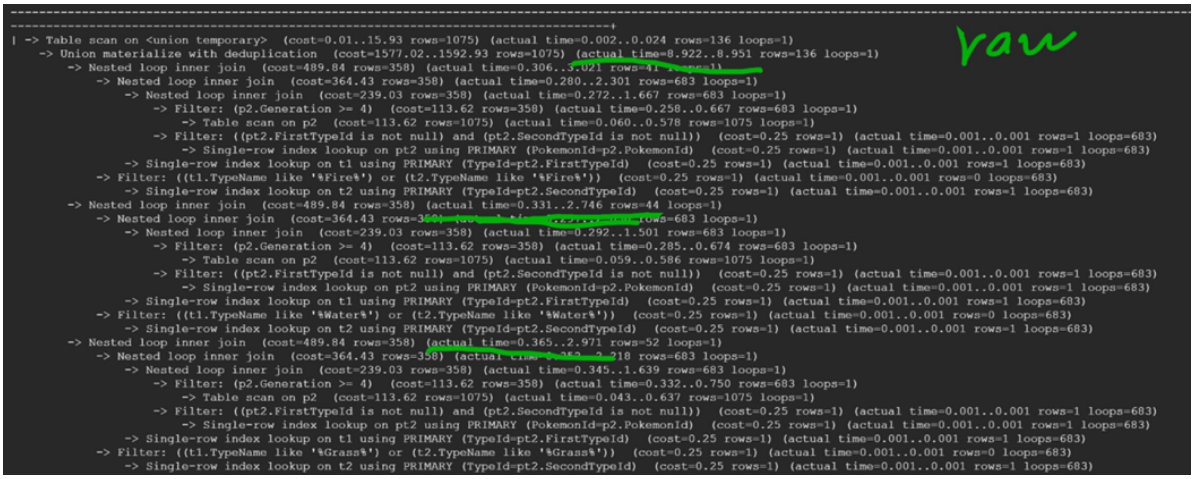
In this experiment we have used generation as an index, since one important filters in the query is whether generation >= 4. As seen from the pictures shown above, in terms of filtering, the efficiency has greatly been improved compared to the original method. This can be explained by the less redundant pattern in searching (since nodes are categorized by generation and generations < 4 are separated)

Running time before indexing: 0.258, 0.285, 0.332

Running time after indexing: 0.233, 0.212, 0.216

**Second try of indexing design:**

Index on **FirstTypeId**



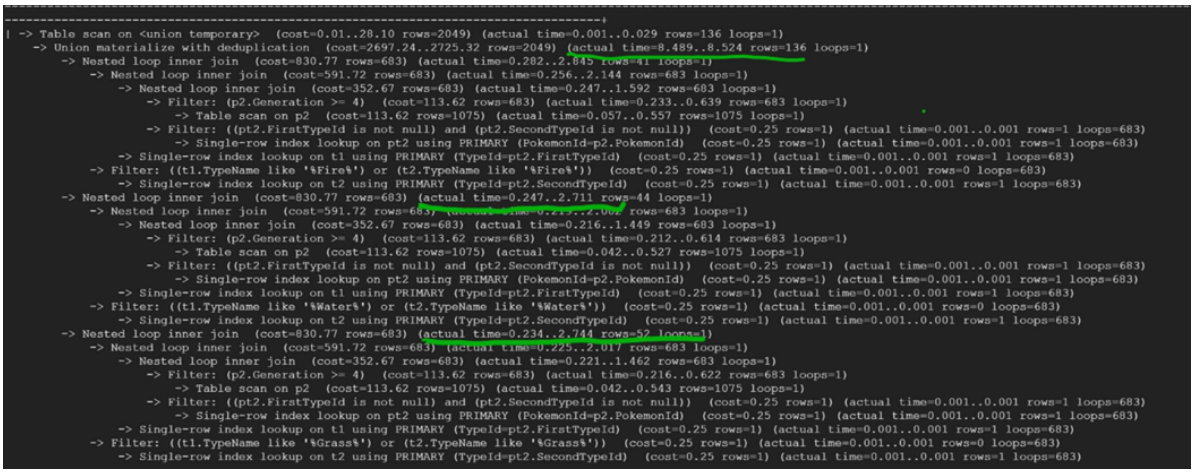
In this experiment we have used FirstTypeId as an index, in order to cluster similar types together. When it comes to the filtering results on our index, there is not too much difference since the actual time are too small to tell apart (both with values of 0.001). However, the nested loop join efficiencies are quite intriguing. Two of them became faster but one became significantly slower. We are not sure about the causation but we are quite certain that this issue is correlated with the distribution of “Fire” Pokemon in our database.

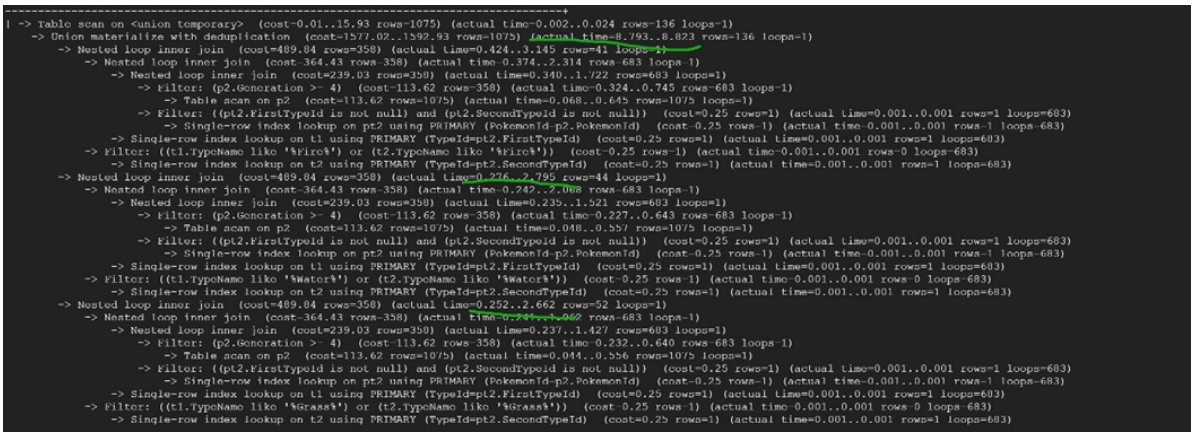
Running time before indexing: 8.92, 0.331, 0.365

Running time after indexing: 13.56, 0.236, 0.237

**Third try of indexing design:**

Index on **PokemonName**




Since Pokemons are like animals, their naming conventions are also categorical. We are trying to explore whether the names of Pokemon can be of value to optimizing our query. It turns out that the results are worse than the original result. So, it can be concluded that naming conventions do not have effects on the searching of certain types or generations of Pokemon. However, we believe that with more advanced “identifying methods” such as prefixes, the estimated result would be better than the present one, in which we are only using alphabetical order as index.

Running time before indexing: 9.489, 0.247, 0.234

Running time after indexing: 8.793, 0.276, 0.252