INTRODUCTION

A Relational Database is a form of database that stores and organizes data elements with established relationships in order to provide quick access. Data is structured into tables in a relational database, which retain information about each object and represent pre-defined categories using rows and columns.

Any object that exists physically or logically can be referred to as an Entity., and a database structure can easily be represented using an Entity Relationship Model.

An Entity Relationship Model (ERM) is a conceptual model that expresses Entities in an application domain (particular field of study), and uses Entities and Relationships to represent the database structure of the domain. An entity relationship diagram (ERD) depicts the desired outcome of modeling with an ERM. As a result, in all Relational Databases, the Database Entity is the most important component.

1.0 ENTITY

In a database, an entity is essentially a container for storing and separating information relevant to the project's objectives.

Individual things, such as people, concepts, or objects, with data stored in a database management system (DBMS) with qualities and relationships to other objects are referred to as entities. An entity in database management might be a single thing, person, place, or object. Such entities can have data saved about them. The Entity Relationship Diagram is a design tool that helps database managers to see the relationships between several entities.

1.2 PROPERTY/ATTRIBUTE

The characterizing features or properties that identify all objects belonging to a specific category and are applied to all cells in a column are known as attributes.

Attributes are classified according to their domain. The permissible values for an attribute are defined by a domain. This comprises information such as the data type, length, values, and other specifics.

An entity characteristic is described by an attribute.

1.2.1 DOMAIN INTEGRITY

Domain integrity is established by the permissible values of an attribute, which ensures that all data in a field is valid.

Domain integrity is defined by:

a. The data type (integer, character, or decimal)

b. Maximum data length.

c. The range, which establishes upper and lower limits.

d. Any constraints, or limitations on allowable values.

e. The type of NULL support (whether an attribute can have an unknown or NULL value).

f. The default value, if any.

g. The date format, if applicable.

RELATIONAL DATABASE

1.1 OVERALL DOMAIN:

A vote and registration system containing together collective individuals categorized as both candidates and electorates.

1.2 LIST OF ENTITIES WITH ATTRIBUTES:

Below is a list of all database entities with their attributes and relationship and part of the domain modelled.

RELATIONSHIP HINT:

1:1 = One to One Cardinality

1:M = One to Many Cardinality

M:N = Many to Many Cardinality

TP = Total Participation

PP = Partial Participation

A. CANDIDATES

This table lists all candidates who have declared that they want to run for a specific position with their party.

- **`candidate_id`**
- **`candidate_name`**
- **`gender`**
- **`age`**
- **`position_id`:** Id for position which candidate is contesting for.
- **`party_id`**: Id of political party which candidate belongs to.
- **`category_id`**: Id showing whether it is a candidate or an electorate.

RELATIONSHIPS/CONSTRAINTS

Parties to Candidates = 1:M, PP

Positions to Candidates = 1:M, PP

Categories to Candidates = 1:M, PP

B. CATEGORIES

This table aids in determining whether or not a voter is a candidate or an electorate.

- `category_id`
- `category_name`

C. ELECTORATES

Table containing all Identified Electorates both registered (identity verified) and non registered.

- `electorate_id`
- `electorate_name`
- `gender`
- `age`
- `polling_unit`: An electorate maintains only one registered Polling Unit.
- `voting_location`: Location where electorate is expected or wishes to vote. This is the location of the Polling Unit.
- `category_id`: Id of category, whether candidate or an electorate.

RELATIONSHIPS/CONSTRAINTS

Unites to Electorates = 1:M, PP

Locations to Electorates = 1:M, PP

Categories to Electorates = 1:M, PP

D. LOCATIONS

Different Locations in the USA where the elections can be conducted.

- **`location_id`**
- **`location_name`** : name of identified location in the United States.

E. PARTIES

A table listing all of the parties to which a candidate can belong in order to vote in the election.

- **`party_id`**
- **`party_name`**
- **`party_color`**: party color which represents the party logo.
- **`office_location`**: location of the party office/headquarter.

RELATIONSHIPS/CONSTRAINTS

Locations to Parties = 1:M, PP

F. POLLING UNITS

This table lists all polling units in each of the polling locations where the election will be held.

- **\`unit_id\`**
- **\`unit_label\`**
- **\`unit_location\`**
- **\`date_created\`**: creation date and time of unit.

RELATIONSHIPS/CONSTRAINTS

Locations to Unites = 1:M, PP

G. POSITIONS

Available Positions for Candidates to contest.

- **`position_id`**
- **`position_name`**: name of contestable position.

H. REGISTRATIONS

This is a table that keeps track of all the candidates and electorates who have registered or been recognized. Only registered voters' votes will be counted in the final results.

- **`registration_id`**
- **`category_id`**: category Id identifying the person making the registration.
- **`candidate_id`**: personal unique Id of candidate registering.
- **`electorate_id`**: personal unique Id of electorate registering.
- **`registration_date`**: date and time of registration.

RELATIONSHIPS/CONSTRAINTS

Categories to Registration s = 1:M, PP

Candidates to Registration = 1:1, PP

Electorates to Registration = 1:1, PP

I. VOTES

A table that records each voter's single vote.

- **`vote_id`**
- **`position_id`**
- **`party_id`**: id of party voted for.
- **`voter_category`**: category which the voter belongs to.
- **`vote_validation_id`**: voter unique registration id.
- **`voting_date`**: date and time of vote.

RELATIONSHIPS/CONSTRAINTS

Parties to Votes = 1:M, PP

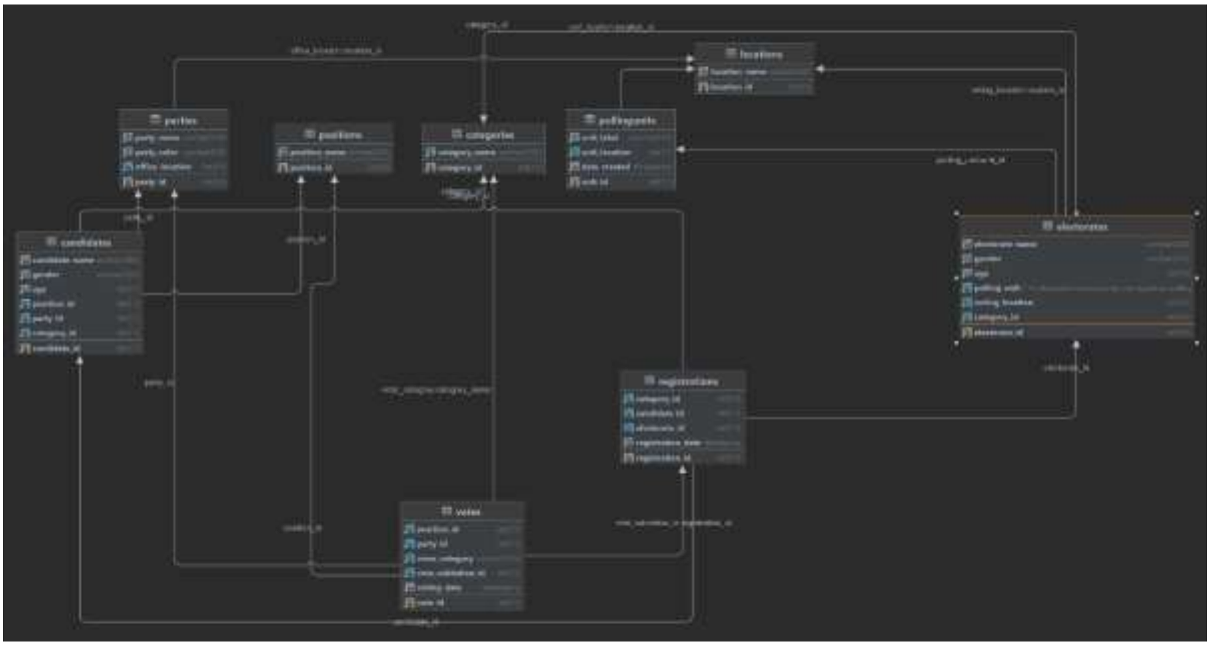
Categories to Votes = 1:M, PP

Registrations to Votes = 1:M, PP

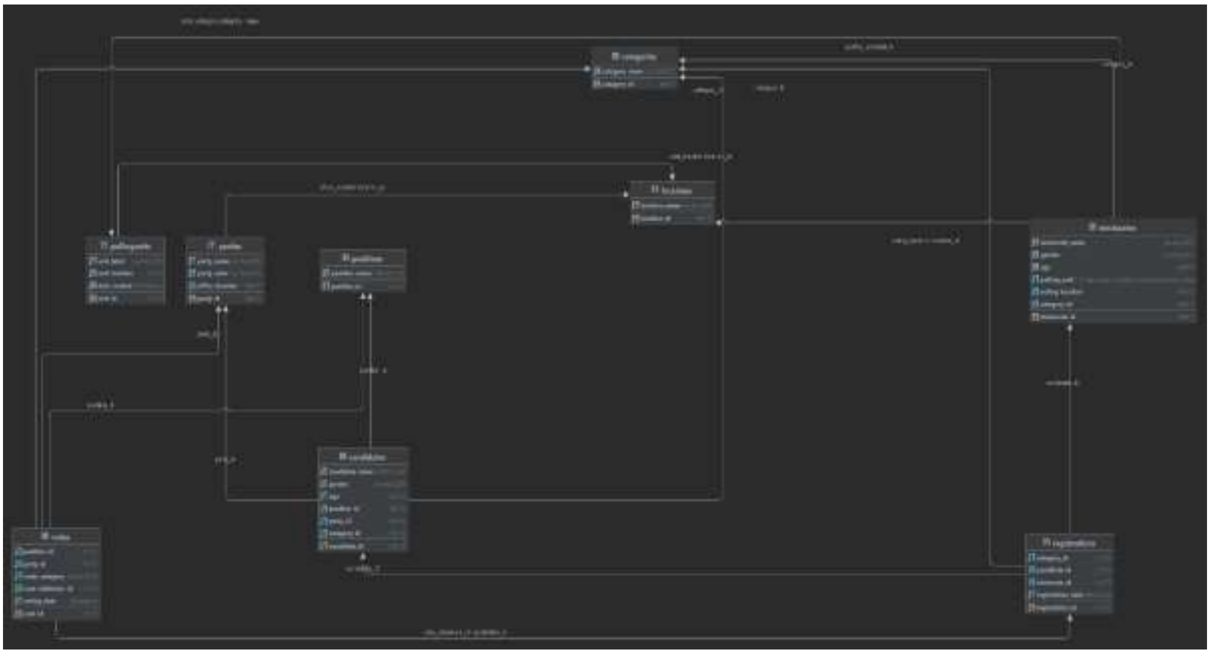
Positions to Votes = 1:M, PP

1.2 ENTITY RELATIONSHIP DIAGRAM

View A



View B



Entity Relationship Diagram of our Relational Database

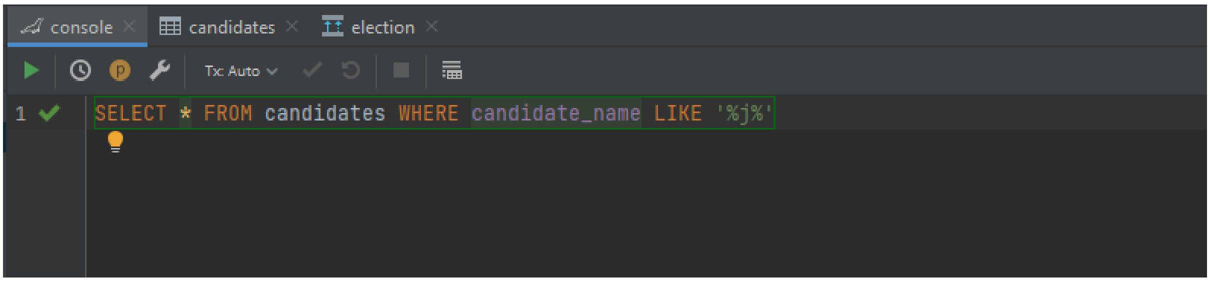
1.3 SQL QUERIES AND RESULTS

Using the Jetbrains Datagrip Software (IDE), The SQL server was connected to and the database was accessed. The following SQL queries were executed and the their results where returned successfully.

1. SELECT, LIKE

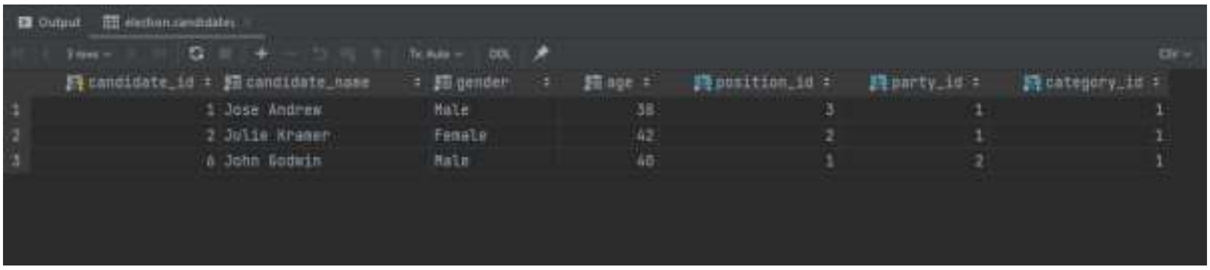
When we wish to obtain data from a table using a certain word or character, we utilize the like operator.

The percent symbol denotes an unknown word or character that can be on the front, rear, or both sides of the letter.



```sql
SELECT * FROM candidates WHERE candidate_name LIKE '%j%'
```

RESULT:



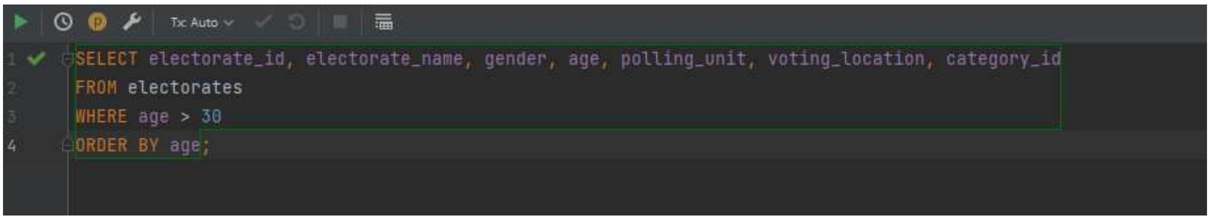
	candidate_id	candidate_name	gender	age	position_id	party_id	category_id
1	1	Jose Andrew	Male	38	3	1	1
2	2	Julie Kramer	Female	42	2	1	1
3	6	John Godwin	Male	46	1	2	1

2. ORDER BY

Using the "ASC" phrase, we were able to extract table values in ascending (smaller to greater) order.

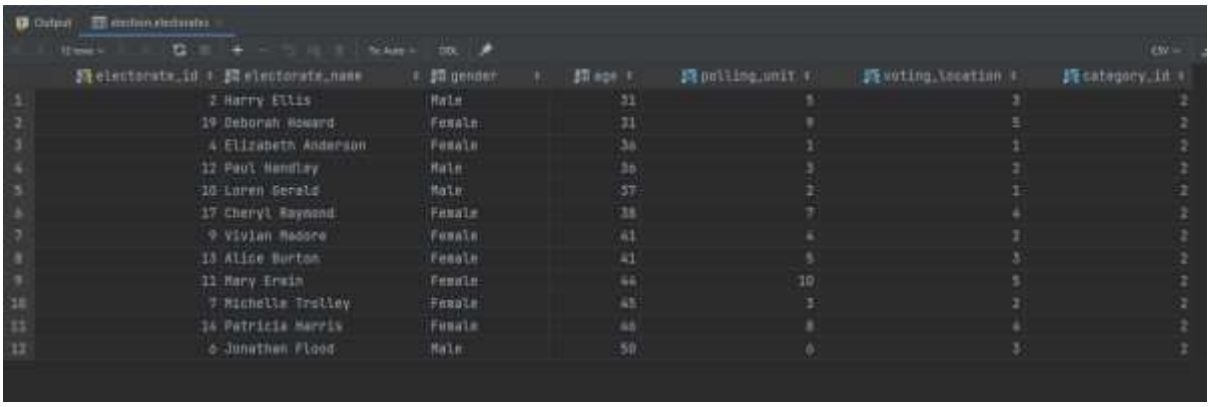
We can sort the values in the number or text columns by row.

The "age" column is reorganized in ascending order, and the rows are rearranged as well.



```sql
SELECT electorate_id, electorate_name, gender, age, polling_unit, voting_location, category_id
FROM electorates
WHERE age > 30
ORDER BY age;
```

RESULT:

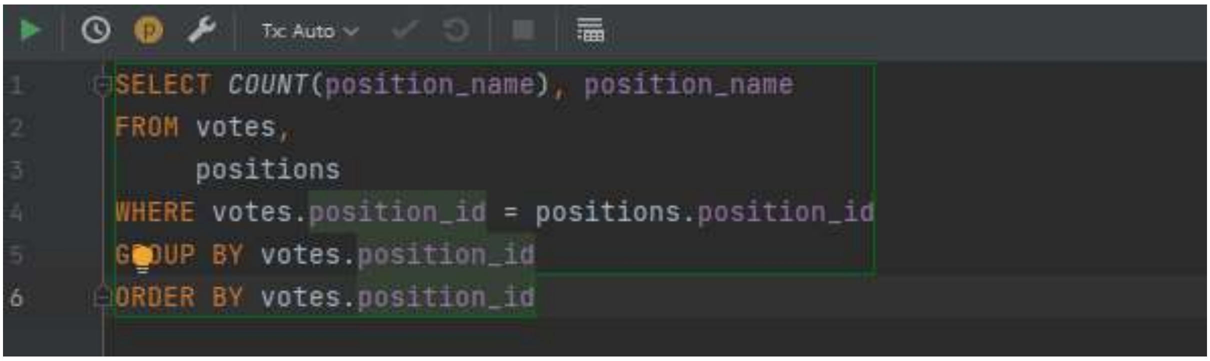


	electorate_id	electorate_name	gender	age	polling_unit	voting_location	category_id
1	2	Harry Ellis	Male	31	5	3	2
2	19	Deborah Howard	Female	31	9	5	2
3	4	Elizabeth Anderson	Female	36	1	1	2
4	12	Paul Handley	Male	36	3	2	2
5	10	Loren Gerald	Male	37	2	1	2
6	17	Cheryl Raymond	Female	38	7	4	2
7	9	Vivian Madore	Female	41	6	2	2
8	13	Alice Burton	Female	41	5	3	2
9	11	Mary Erwin	Female	44	10	5	2
10	7	Michelle Trolley	Female	45	3	2	2
11	14	Patricia Harris	Female	46	8	4	2
12	6	Jonathan Flood	Male	50	6	3	2

3. GROUP BY

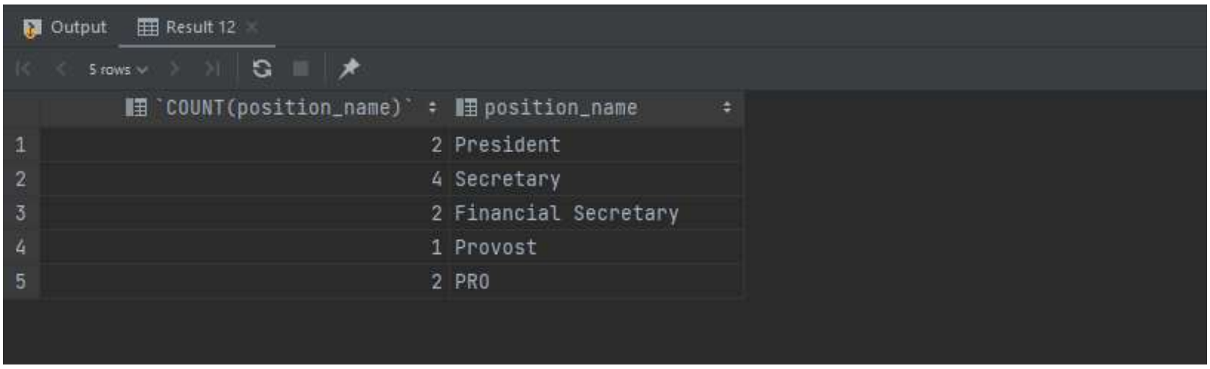
The 'Group By' property divides the output data into groups based on the supplied attribute.

This SQL query will select the number of times a position name appeared in the votes table, as well as the "poclause_name" columns from the "positions" table, then filter them by the position id value to include records with the same position id in both tables, then group records with similar position id in the votes table, and finally output them sorted by the same position id. The primary guideline is that in a Select statement, the group by clause must always come after the where clause and must come before the Order by clause.



```sql
SELECT COUNT(position_name), position_name
FROM votes,
     positions
WHERE votes.position_id = positions.position_id
GROUP BY votes.position_id
ORDER BY votes.position_id
```

RESULT:

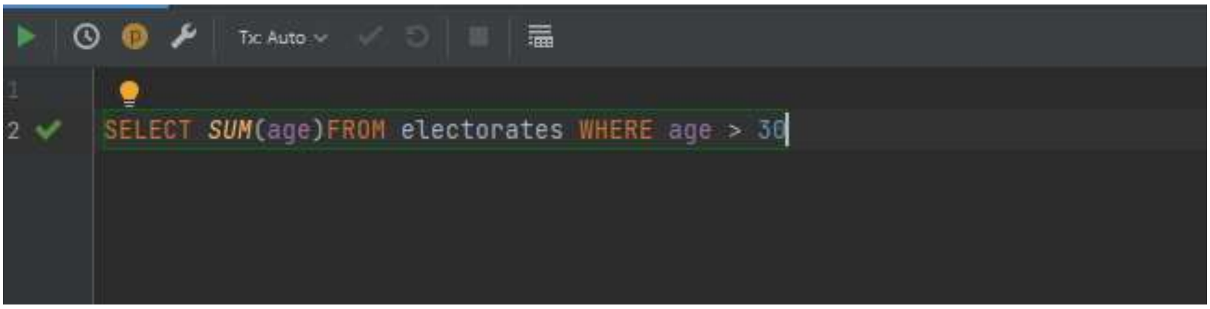


```
   `COUNT(position_name)`  position_name
1                       2  President
2                       4  Secretary
3                       2  Financial Secretary
4                       1  Provost
5                       2  PRO
```

4. SUM, GREATER THAN

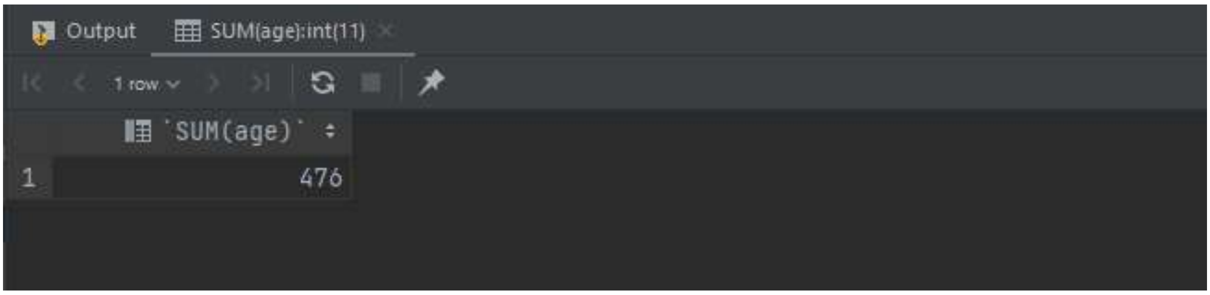
To find the total of the integer values in a column, use the sum function.

The total value of the column age (if larger than 30) is added and obtained in this query.



```
SELECT SUM(age)FROM electorates WHERE age > 30
```

RESULT:

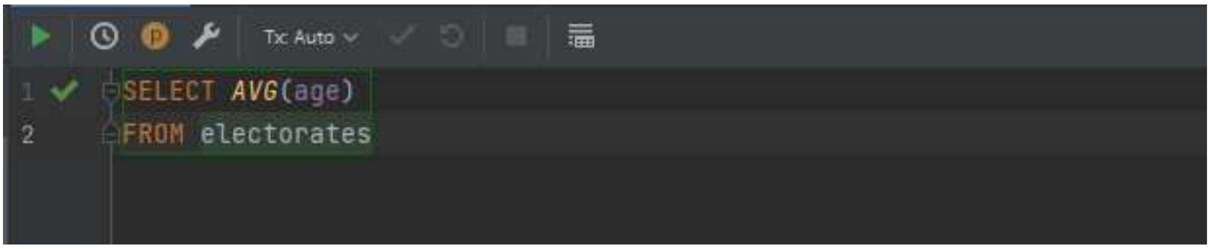


```
`SUM(age)`
1        476
```

5. AVERAGE

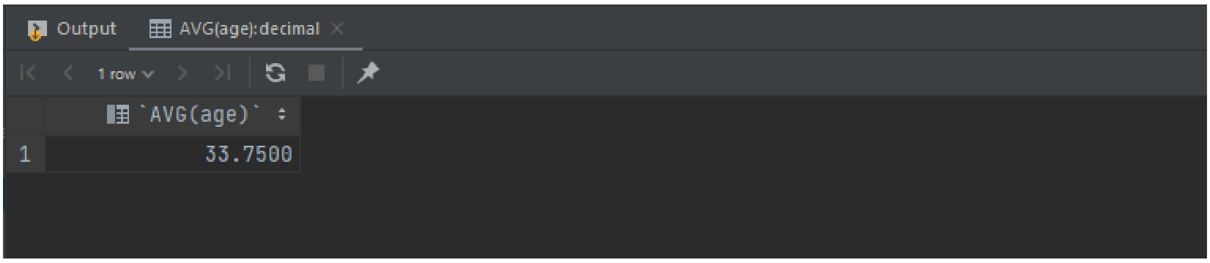
The AVG function is used to calculate the average of all integer values in a column.

The average of the column "age" is returned in this query.



```
SELECT AVG(age)
FROM electorates
```

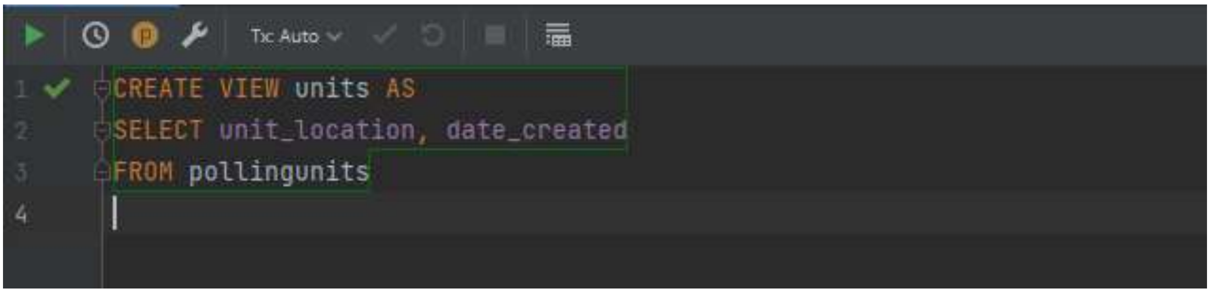
RESULT:



```
  `AVG(age)`
1    33.7500
```

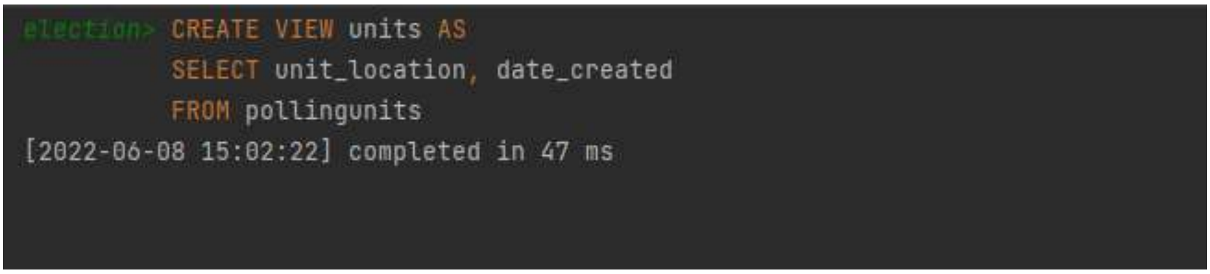
6. VIEW

A view is a customized table that is created in response to a query. It has the same tables and rows as any other table. Running queries in SQL as individual views is usually a good idea since it allows them to be returned later to inspect the query results rather than having to compute the same command every time for a specific set of results.



```sql
CREATE VIEW units AS
SELECT unit_location, date_created
FROM pollingunits
```

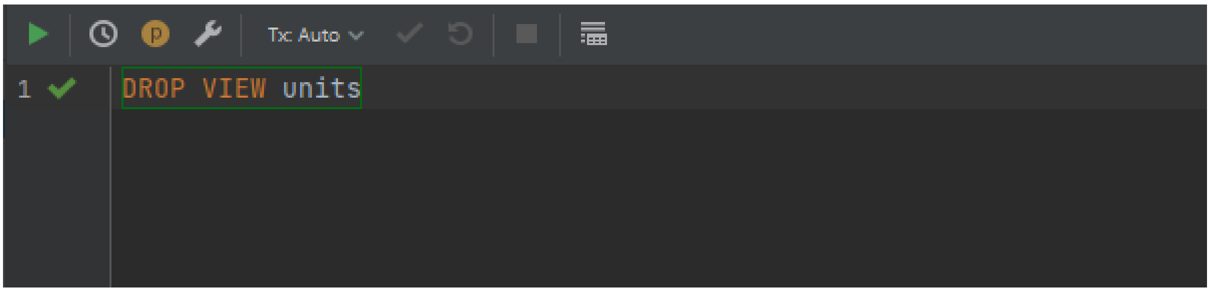
RESULTS:



```
election> CREATE VIEW units AS
          SELECT unit_location, date_created
          FROM pollingunits
[2022-06-08 15:02:22] completed in 47 ms
```

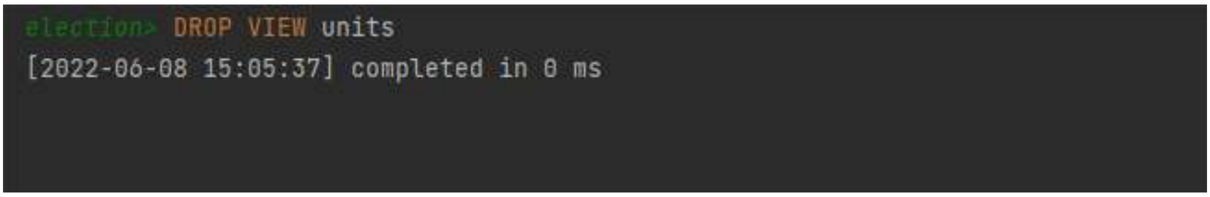
7. DROP

A view named 'units' will be dropped or deleted by this query. The DROP VIEW command is disabled if any views are dependent on the view about to be dropped.



```
DROP VIEW units
```

RESULTS:

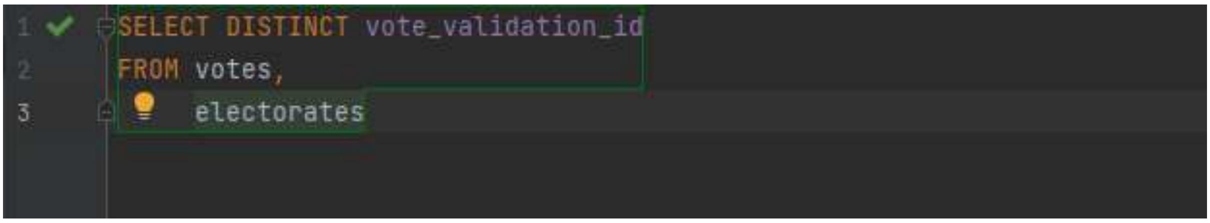


```
election> DROP VIEW units
[2022-06-08 15:05:37] completed in 8 ms
```

8. DISTINCT

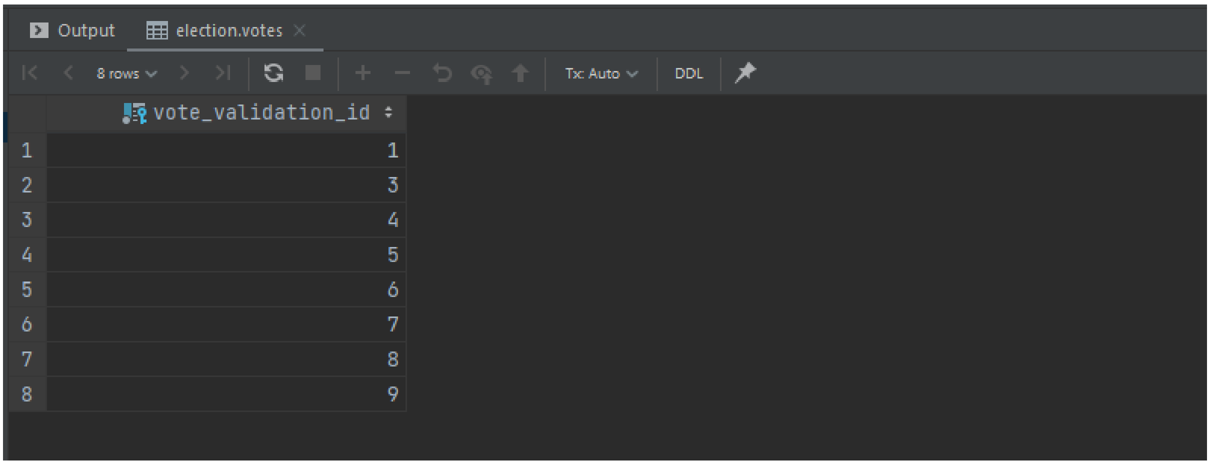
When there are repeated rows in a table, the distinct statement is used to get the distinct values.

The separate values of column “voter_validation_id” are retrieved in this tutorial, which implies that the Validation IDs will not be repeated.



```
SELECT DISTINCT vote_validation_id
FROM votes,
     electorates
```

RESULT:



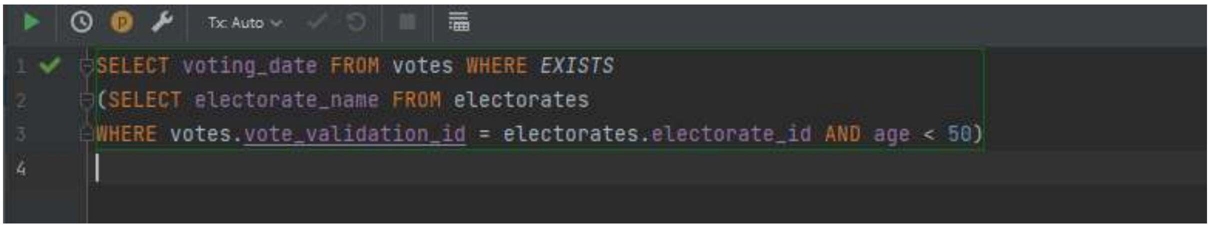
```
   vote_validation_id
1                    1
2                    3
3                    4
4                    5
5                    6
6                    7
7                    8
8                    9
```

9. EXISTS, LESS THAN

Exists is used to check if a sub query returns any values, it returns true if any rows are retrieved else it returns false.

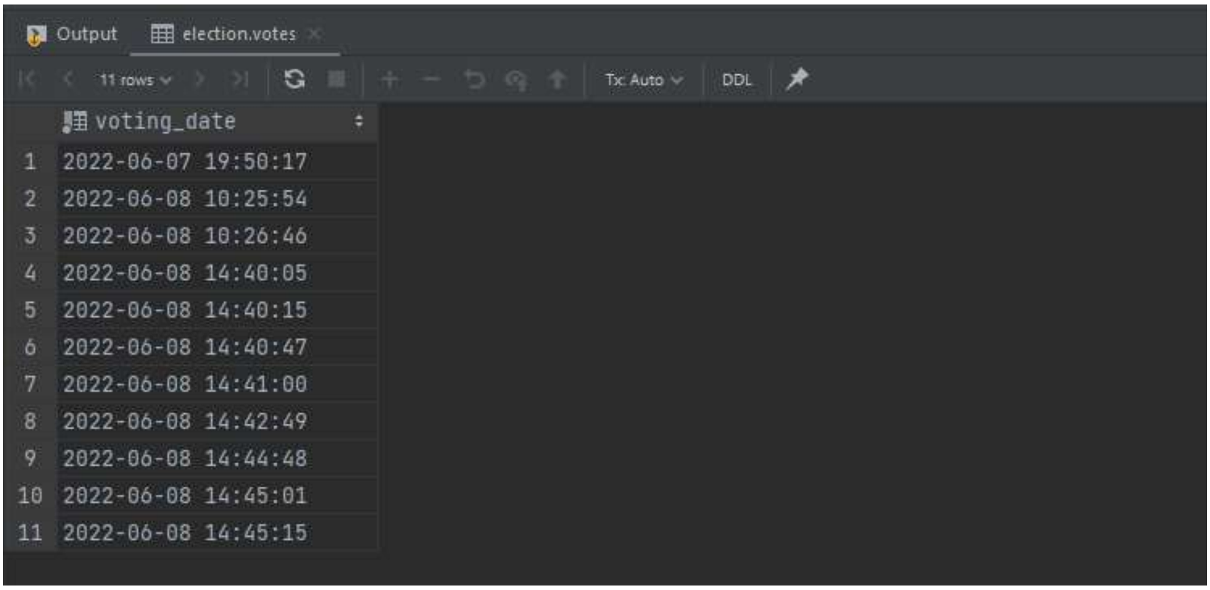
In this tutorial the sub query checks, if the “age” in “electorates” table is less than 50, if there are any rows then it returns true and the query retrieves the “voting_date” from the “votes” table.

Else if there are no rows then it returns false and no rows will be retrieved from the “votes” table.



```
SELECT voting_date FROM votes WHERE EXISTS
(SELECT electorate_name FROM electorates
WHERE votes.vote_validation_id = electorates.electorate_id AND age < 50)
```

RESULT:

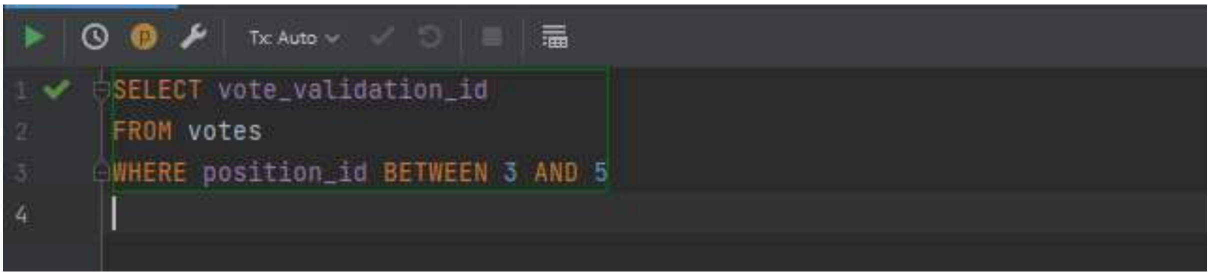


```
   voting_date
1  2022-06-07 19:50:17
2  2022-06-08 10:25:54
3  2022-06-08 10:26:46
4  2022-06-08 14:40:05
5  2022-06-08 14:40:15
6  2022-06-08 14:40:47
7  2022-06-08 14:41:00
8  2022-06-08 14:42:49
9  2022-06-08 14:44:48
10 2022-06-08 14:45:01
11 2022-06-08 14:45:15
```

10. BETWEEN, AND

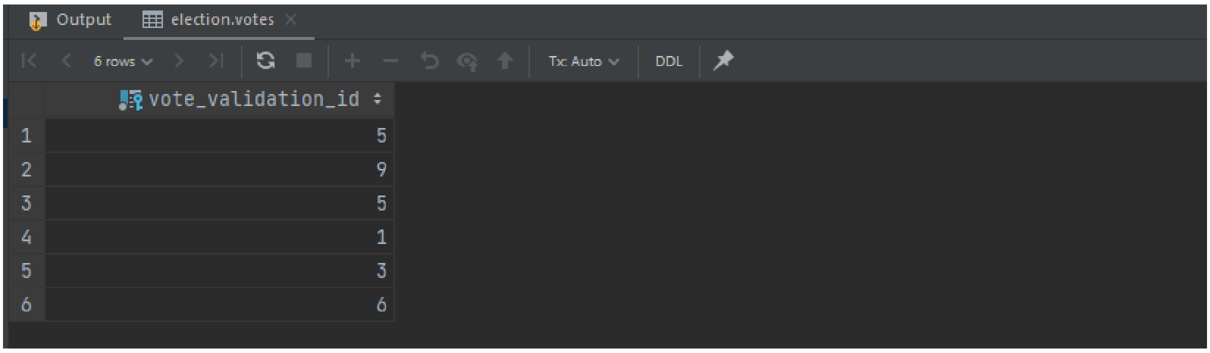
Between operator is used to retrieve rows if the column value is within the given range.

Also, to check and retrieve rows which satisfy more than one conditions you can use AND operator.



```
SELECT vote_validation_id
FROM votes
WHERE position_id BETWEEN 3 AND 5
```

RESULT:



```
   vote_validation_id
1                   5
2                   9
3                   5
4                   1
5                   3
6                   6
```

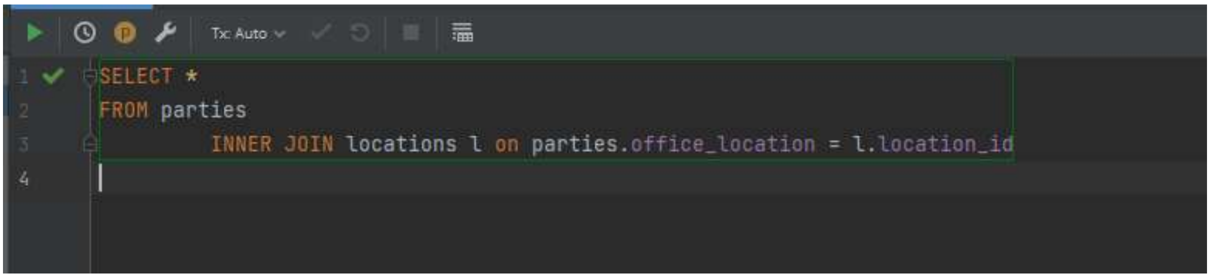
11. INNER JOIN, JOIN, ON

We may extract the values from two tables by using a single query.

To connect two tables, the join clause is utilized.

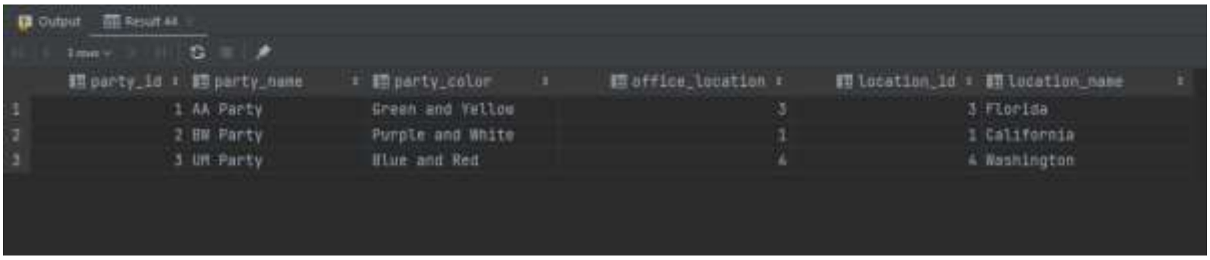
We're connecting two tables in this query: "parties," which has information about the parties, and "location," which provides information about all possible location.

The "location_id" field, which is present in both tables, is used to link them.



```
SELECT *
FROM parties
        INNER JOIN locations l on parties.office_location = l.location_id
```

RESULT:



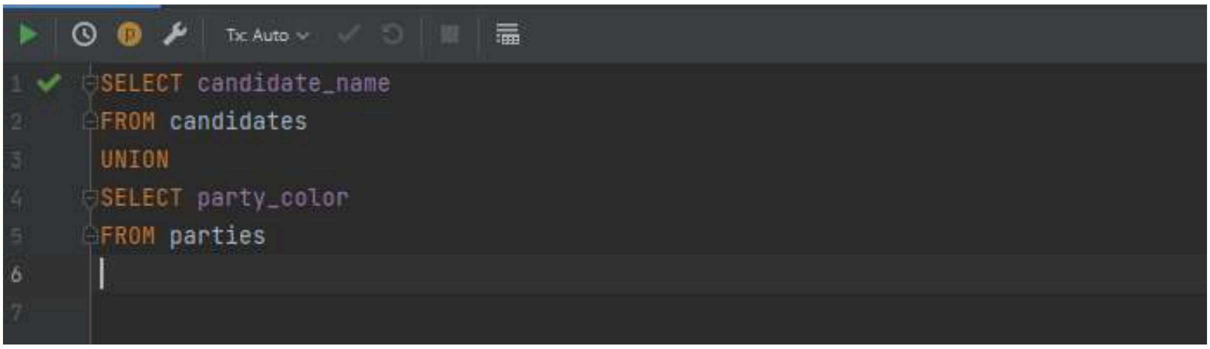
	party_id	party_name	party_color	office_location	location_id	location_name
1	1	AA Party	Green and Yellow	3	3	Florida
2	2	BW Party	Purple and White	1	1	California
3	3	UM Party	Blue and Red	4	4	Washington

12. UNION

Union operator is used to combine one or more select statements into one.

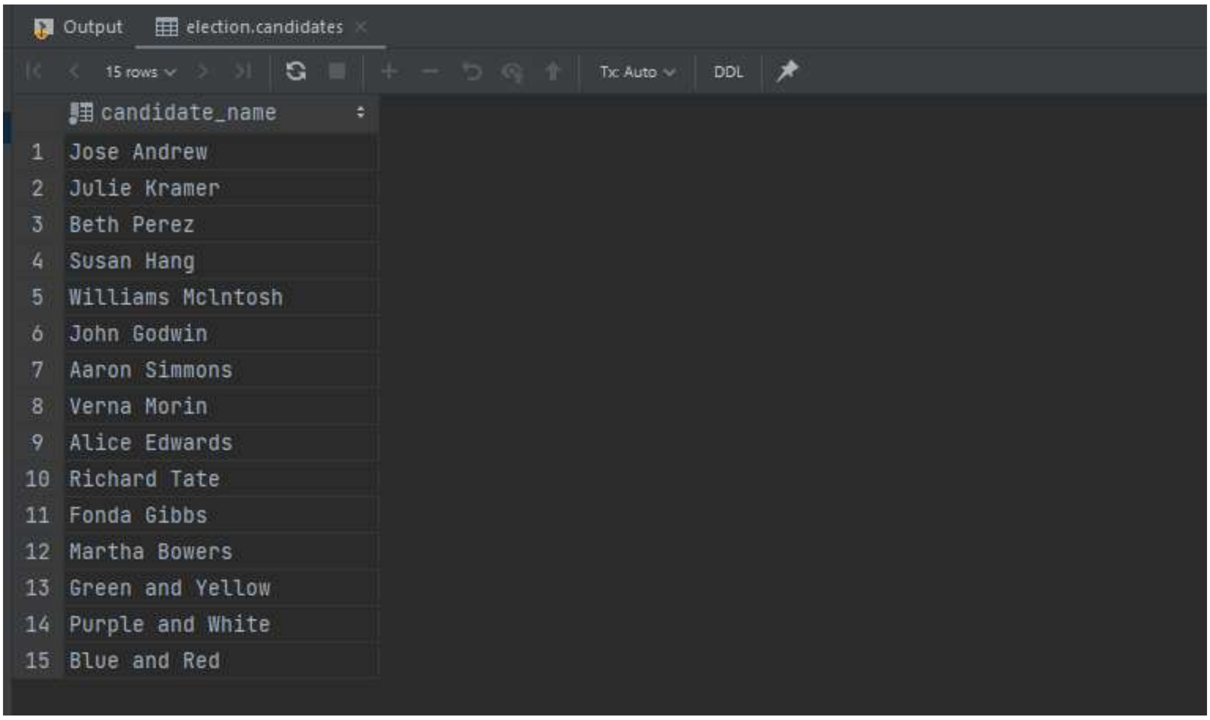
Rules for using union operator are, the select statements must have the same number of columns and same data types in same order.

Here, we combine the "candidates" table and "parties" table together.



```
SELECT candidate_name
FROM candidates
UNION
SELECT party_color
FROM parties
```

RESULT:



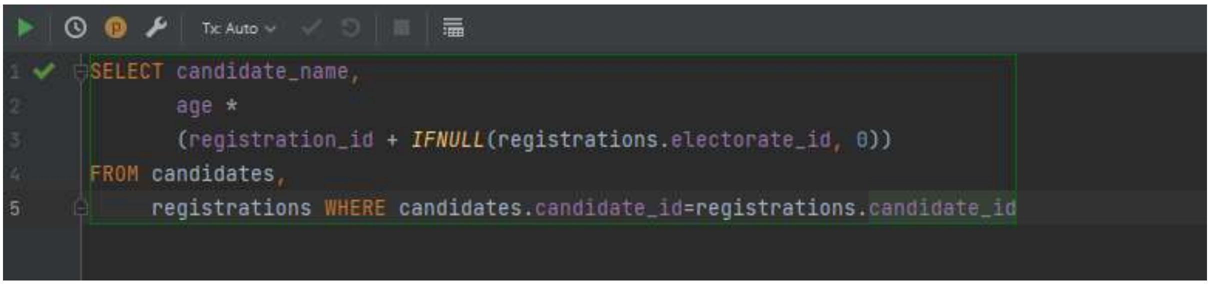
```
   candidate_name
1  Jose Andrew
2  Julie Kramer
3  Beth Perez
4  Susan Hang
5  Williams Mclntosh
6  John Godwin
7  Aaron Simmons
8  Verna Morin
9  Alice Edwards
10 Richard Tate
11 Fonda Gibbs
12 Martha Bowers
13 Green and Yellow
14 Purple and White
15 Blue and Red
```

13. IFNULL

The term NULL is used to describe a value that is not present. The term NULL does not imply "zero." A condition in a domain that appears to be empty is a

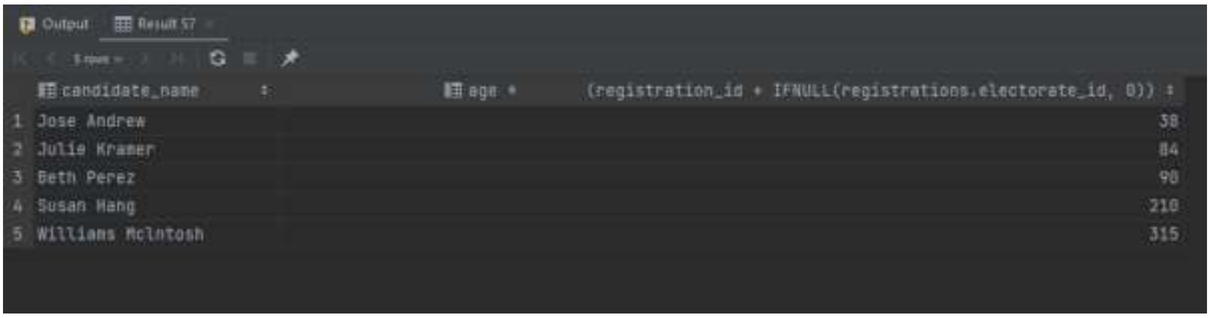
NULL value in a table field. A domain with an absent value is a column with a NULL value.

When NULL values are permitted in a field, computations based on such values will also yield NULL results. The IFNULL operator can be used to avoid this. When a calculation encounters a field with a NULL value, a value of zero is returned rather than a value of NULL in the following query:



```sql
SELECT candidate_name,
       age *
       (registration_id + IFNULL(registrations.electorate_id, 0))
FROM candidates,
     registrations WHERE candidates.candidate_id=registrations.candidate_id
```

RESULT:



	candidate_name	age * (registration_id + IFNULL(registrations.electorate_id, 0))
1	Jose Andrew	38
2	Julie Kramer	84
3	Beth Perez	90
4	Susan Hang	210
5	Williams McIntosh	315

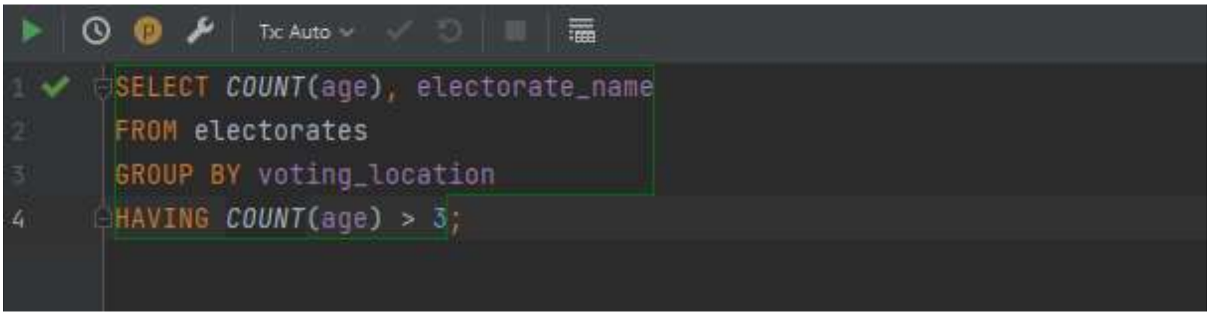
14. HAVING

We can’t use where clause along with aggregate function and group by clause, so we use having clause to check conditions.

Having clause is used to check condition along with the aggregate function (e.g. Count(), sum(), avg(), min(), max()).

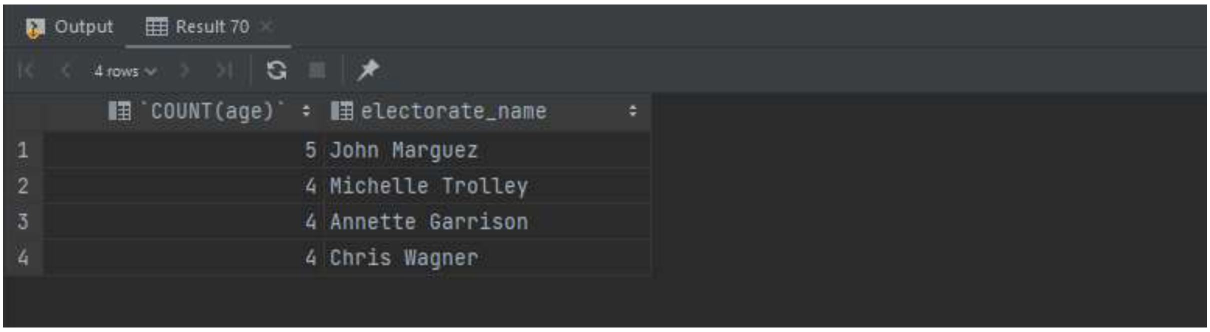
This query counts the total occurrence of each value in age column and then it checks the condition to get the rows where the count value of age greater than 3.

It then displays the rows having count value greater than 3.



```
SELECT COUNT(age), electorate_name
FROM electorates
GROUP BY voting_location
HAVING COUNT(age) > 3;
```

RESULT:

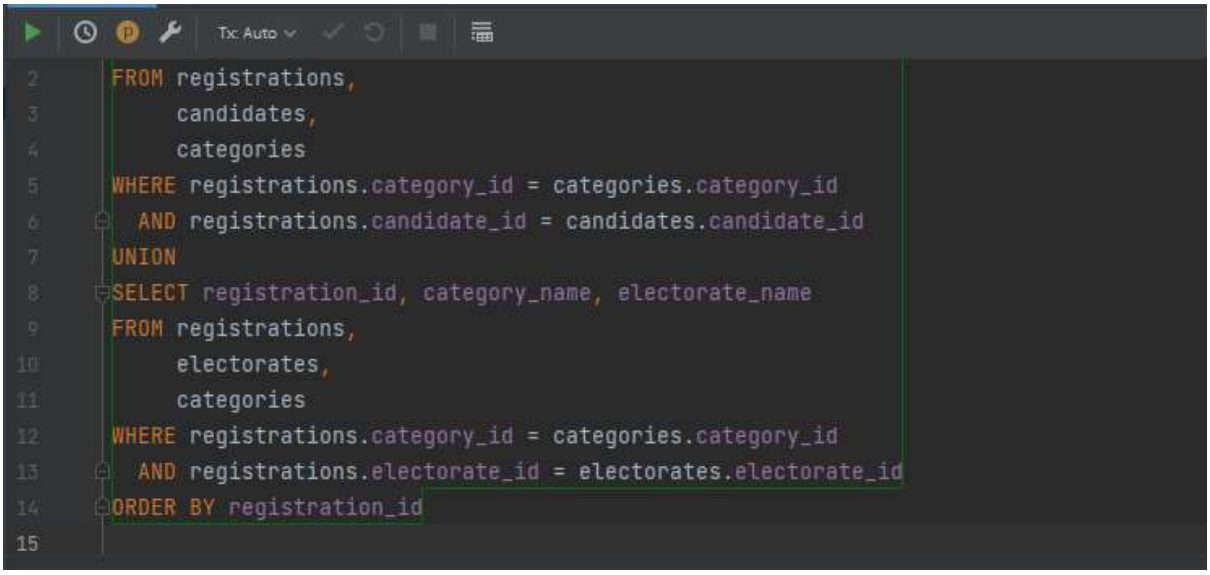


```
     `COUNT(age)`  electorate_name
1               5  John Marguez
2               4  Michelle Trolley
3               4  Annette Garrison
4               4  Chris Wagner
```

15. UNION, ORDER BY

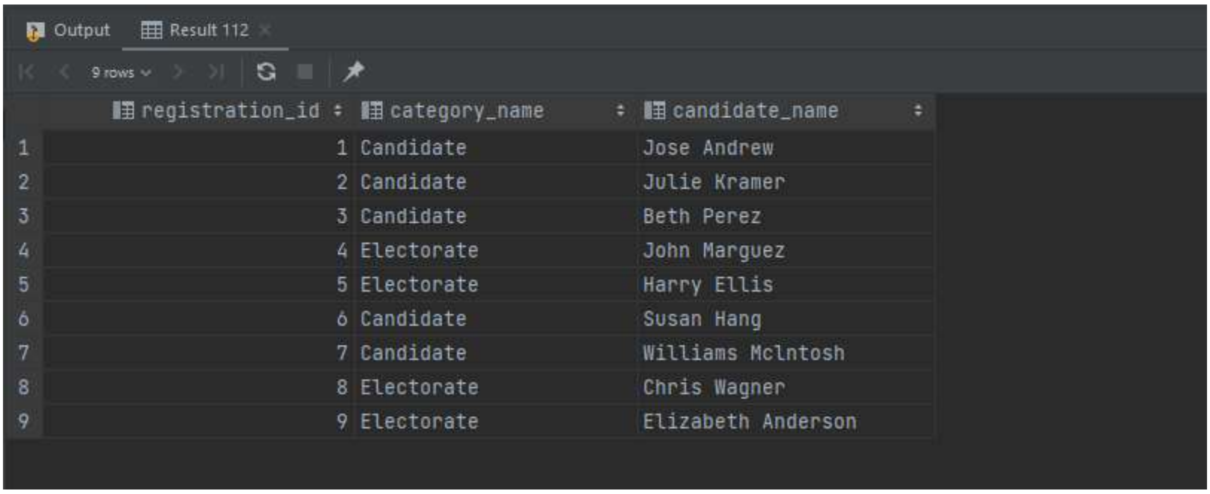
We can combine the results of two SQL query examples into one naturally with the UNION keyword. Here we want to create a new table by combining the candidate's name and the electorate name from registration table. With a list of both combined ordered by their registration ids, one can look for patterns or make comparisons easily.

The UNION keyword makes it possible to combine JOINS and other criteria to achieve a very powerful new table generation potential.



```sql
FROM registrations,
     candidates,
     categories
WHERE registrations.category_id = categories.category_id
  AND registrations.candidate_id = candidates.candidate_id
UNION
SELECT registration_id, category_name, electorate_name
FROM registrations,
     electorates,
     categories
WHERE registrations.category_id = categories.category_id
  AND registrations.electorate_id = electorates.electorate_id
ORDER BY registration_id
```

RESULT:



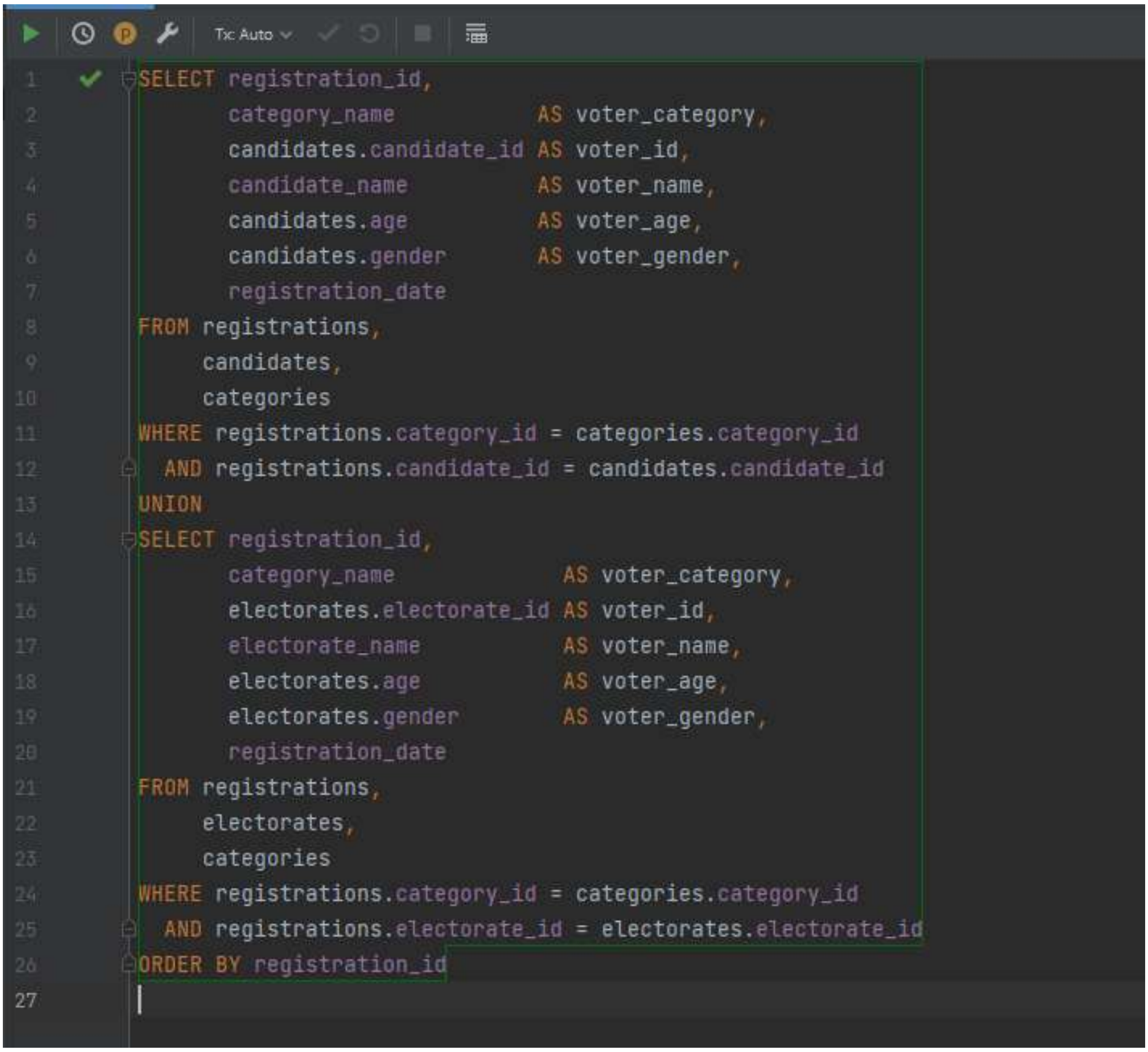
	registration_id	category_name	candidate_name
1	1	Candidate	Jose Andrew
2	2	Candidate	Julie Kramer
3	3	Candidate	Beth Perez
4	4	Electorate	John Marguez
5	5	Electorate	Harry Ellis
6	6	Candidate	Susan Hang
7	7	Candidate	Williams Mclntosh
8	8	Electorate	Chris Wagner
9	9	Electorate	Elizabeth Anderson

16. UNION, AS (Alias Name), …

While retrieving table values, we can denote the column head by our own name.

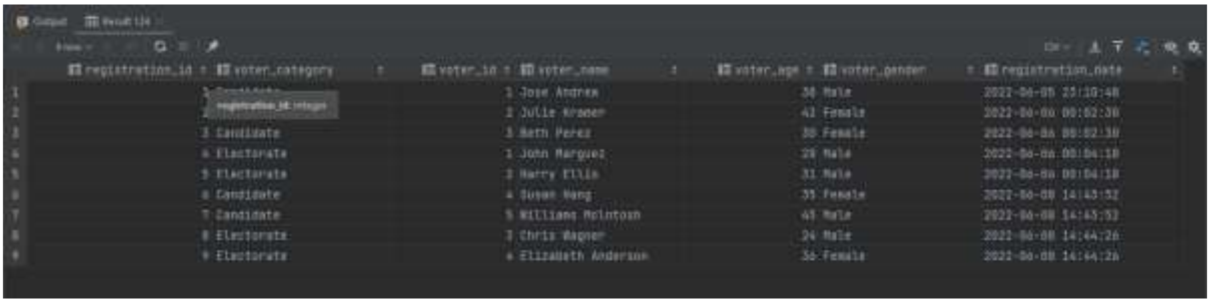
This is mostly used to hide the column name from the user and to display the column after some operation.

It is denoted by original column name followed by as keyword or a space then followed by duplicate column name to be shown.



```sql
SELECT registration_id,
       category_name           AS voter_category,
       candidates.candidate_id AS voter_id,
       candidate_name          AS voter_name,
       candidates.age          AS voter_age,
       candidates.gender       AS voter_gender,
       registration_date
FROM registrations,
     candidates,
     categories
WHERE registrations.category_id = categories.category_id
  AND registrations.candidate_id = candidates.candidate_id
UNION
SELECT registration_id,
       category_name             AS voter_category,
       electorates.electorate_id AS voter_id,
       electorate_name           AS voter_name,
       electorates.age           AS voter_age,
       electorates.gender        AS voter_gender,
       registration_date
FROM registrations,
     electorates,
     categories
WHERE registrations.category_id = categories.category_id
  AND registrations.electorate_id = electorates.electorate_id
ORDER BY registration_id
```

RESULT:



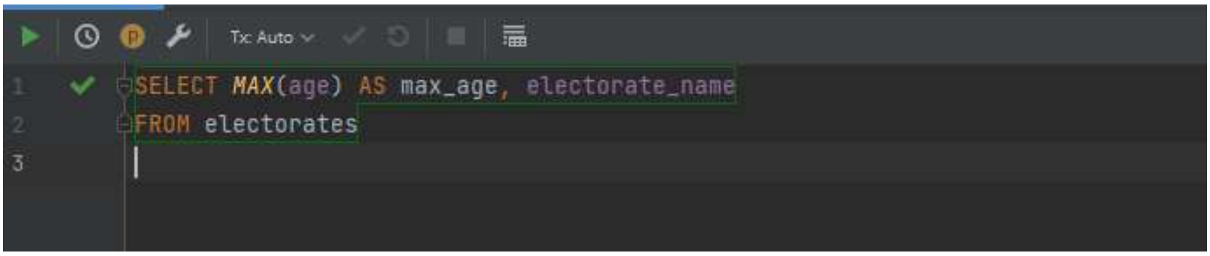
	registration_id	voter_category	voter_id	voter_name	voter_age	voter_gender	registration_date
1	1	Candidate	1	Jose Andrew	38	Male	2022-06-05 23:10:48
2	2	[illegible]	2	Julie Kramer	42	Female	2022-06-06 00:02:38
3	3	Candidate	3	Beth Perez	30	Female	2022-06-06 00:02:38
4	4	Electorate	1	John Marquez	28	Male	2022-06-06 00:04:18
5	5	Electorate	2	Harry Ellis	31	Male	2022-06-06 00:04:18
6	6	Candidate	4	Susan Wang	35	Female	2022-06-08 14:43:52
7	7	Candidate	5	Williams McIntosh	45	Male	2022-06-08 14:43:52
8	8	Electorate	3	Chris Wagner	24	Male	2022-06-08 14:44:26
9	9	Electorate	4	Elizabeth Anderson	36	Female	2022-06-08 14:44:26

17. MAX

Max function returns the maximum integer value from the specified column.

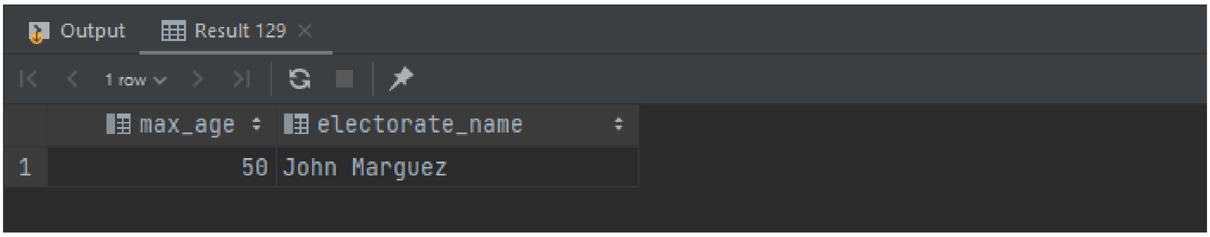
In this query, the maximum value from “age” column is retrieved.

The retrieved value is 50 and displayed under the column name “max_age”.



```sql
SELECT MAX(age) AS max_age, electorate_name
FROM electorates
```

RESULT:



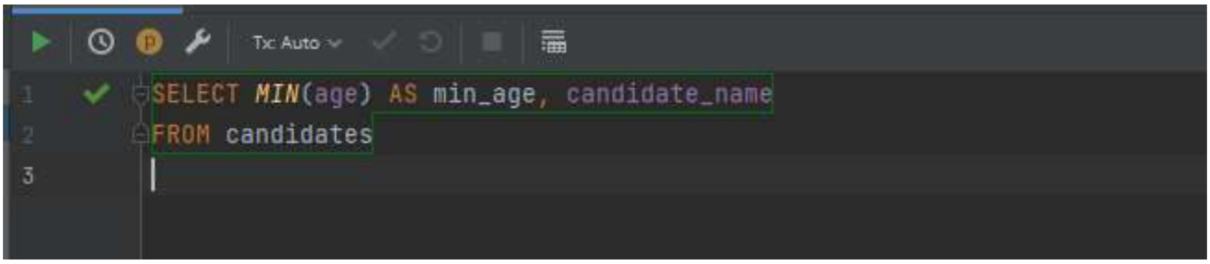
```
    max_age  electorate_name
1        50  John Marguez
```

18. MIN

Min function returns the minimum integer value from the specified column.

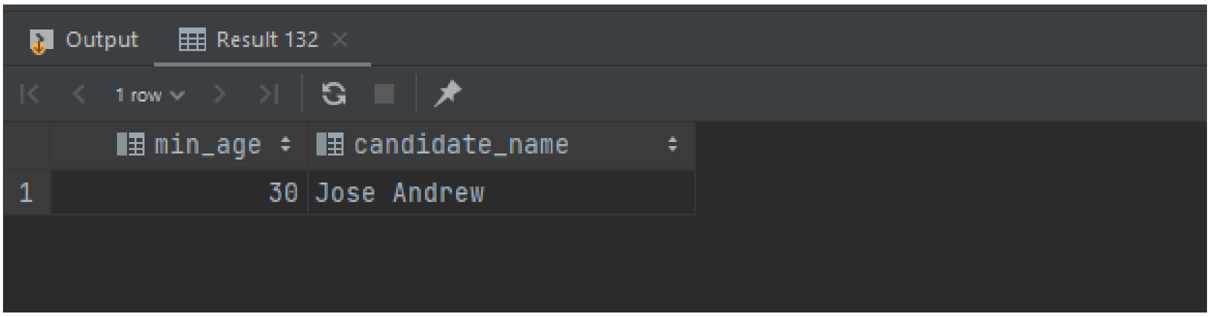
In this tutorial the minimum value from “age” column is retrieved.

The retrieved value is 30 and displayed under the column name “min_age”.



```sql
SELECT MIN(age) AS min_age, candidate_name
FROM candidates
```

RESULT:

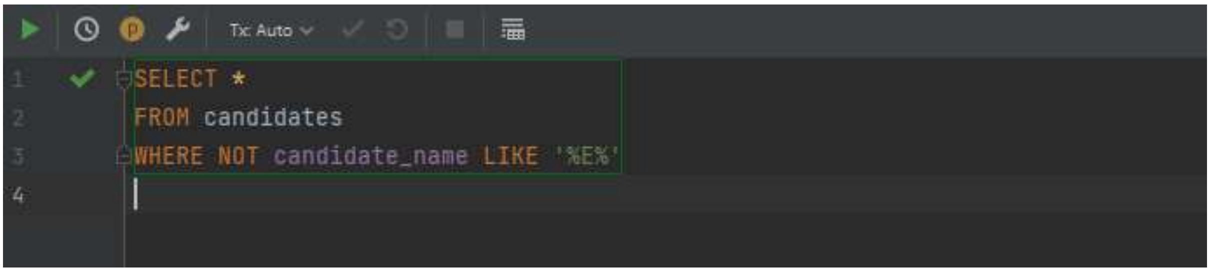


```
    min_age  candidate_name
1        30  Jose Andrew
```

19. NOT, LIKE

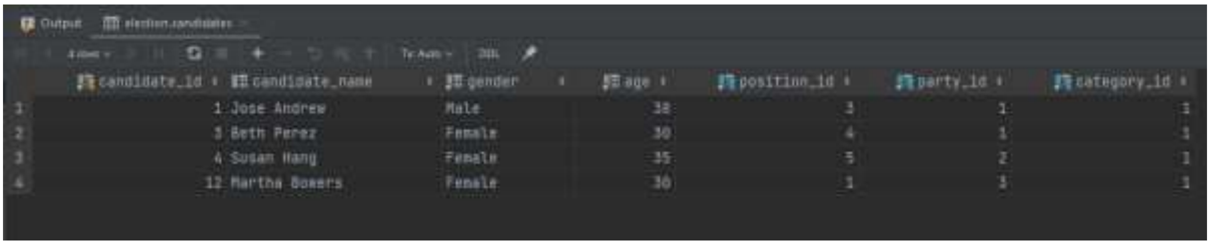
To check and retrieve rows which can’t satisfy the given condition you can use NOT operator.

In this query, the rows which doesn’t have 'E' letter in the column “candidate_name” are retrieved.



```sql
SELECT *
FROM candidates
WHERE NOT candidate_name LIKE '%E%'
```

RESULT:



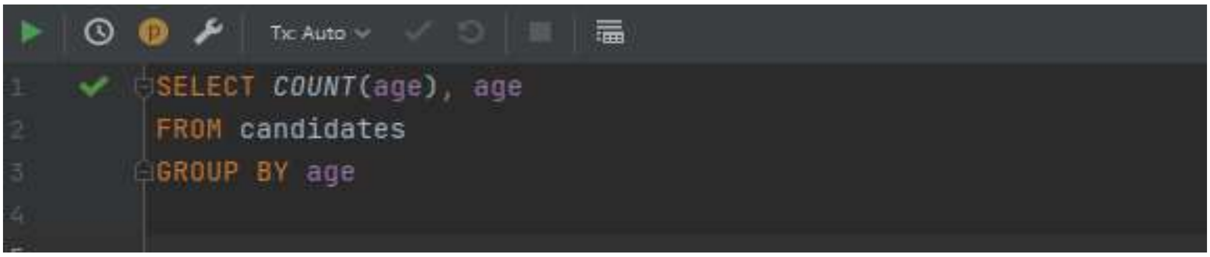
	candidate_id	candidate_name	gender	age	position_id	party_id	category_id
1	1	Jose Andrew	Male	38	3	1	1
2	3	Beth Perez	Female	30	4	1	1
3	4	Susan Hang	Female	35	5	2	1
4	12	Martha Bowers	Female	30	1	3	1

20. COUNT, GROUP BY

Count function can be used along with Group by to retrieve the particular count of each value in a column.

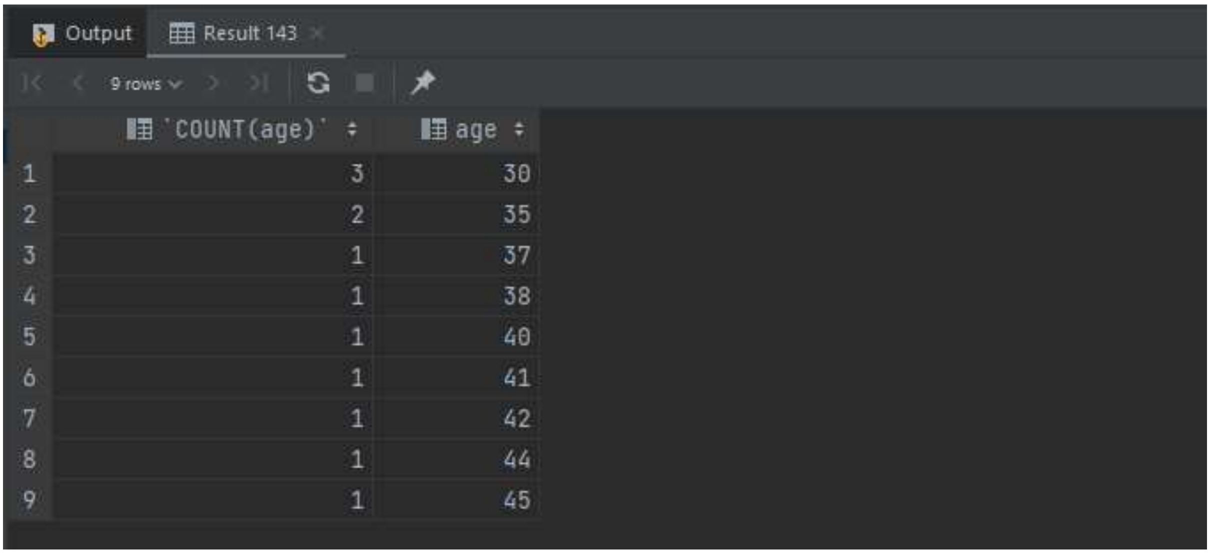
By using Group by and specifying a column name, the total number of occurrences of each value is taken and displayed.

In this query, the column value of “age” in each row is checked and the total occurrence of each column value is displayed.



```
SELECT COUNT(age), age
FROM candidates
GROUP BY age
```

RESULT:

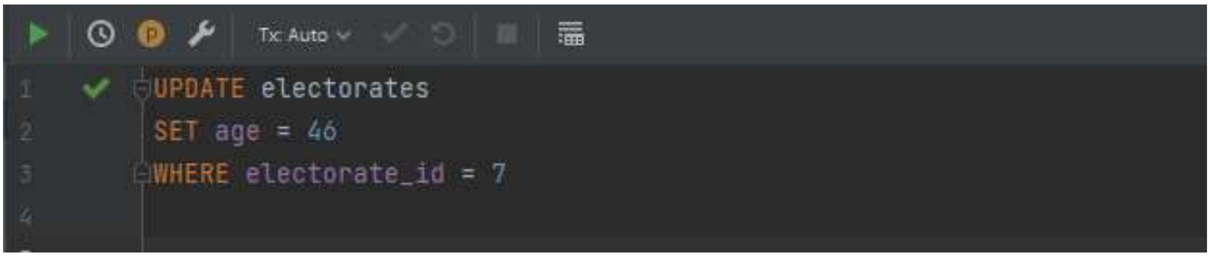


	`COUNT(age)`	age
1	3	30
2	2	35
3	1	37
4	1	38
5	1	40
6	1	41
7	1	42
8	1	44
9	1	45

21. UPDATE

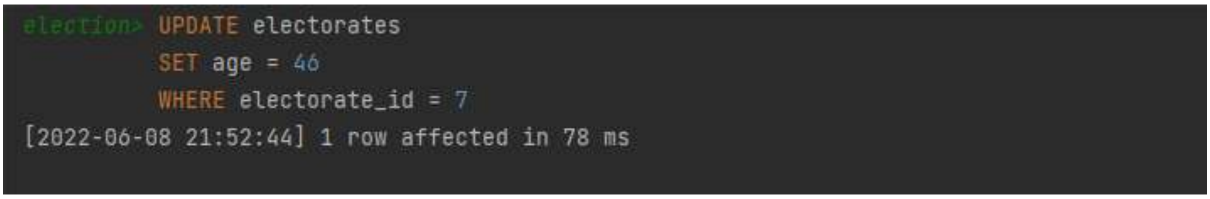
Update statement is used to update the table values.

In this query, “electorates” table is updated, it sets the “age” value as 46 (age=46) for the row which has “electorate_id” as 7(electorate_id=7).



```
UPDATE electorates
SET age = 46
WHERE electorate_id = 7
```

RESULT:



```
election> UPDATE electorates
          SET age = 46
          WHERE electorate_id = 7
[2022-06-08 21:52:44] 1 row affected in 78 ms
```