DATABASE APPLICATIONS

hUSEYIN BATOR | S3660418

assignment 1 - 25%

Huseyin Bator

2024

## Task 1

The three tables that I expected to become extremely large and grow over time are:

* The VoterRegistry Table.
* The Ballot Table.
* The ballotPreference Table.

### Justification of choice

VoterRegistry Table.

The VoterRegistry table consists of data that is captured from a voter that is registered to vote for the AEC. The data will be expected to grow as the general population for Australia raises. As of 2023 there are around “17 million” Australians enrolled to vote for the 2023 election which will marginally grow over time. The population Growth over 2023 was about “1.2%” according to population.gov.

The Ballot Table

The Ballot Table will grow in size over time due to the general population voting for the House of representatives. We can assume for every registered voter that will participate in in the voting will be allocated a ballot. The growth in enrolment from 2022 to 2023 is approximately **4.05%,** As from the statistics gathered from the AEC website where in **2022 we had 17,032,907 enrolled and in 2023 17,721,975 enrolled to vote in 2023** which is a 4.05% growth rate. We then can use this to calculate the compounding ballots over 10 years casted. This is assuming that all enrolled voters had voted.

BallotPreference Table

The BallotPreference table will grow in size with along with the Ballot Table. It will almost be a 1 – 1 replica size of the ballot Table (Same amount of records as ballot table). Just different sizing in data per record due to the information being stored.

## Calculations

### Initial Table Size - VoterRegistry

* VoterID (INT): 4 bytes
* FirstName (VARCHAR(100)): 100 bytes
* MiddleNames (VARCHAR(100)): 100 bytes
* LastName (VARCHAR(100)): 100 bytes
* Address (VARCHAR(255)): 255 bytes
* DoB (DATE): 3 bytes
* Gender (VARCHAR (1)): 1 byte
* ResidentialAddress (VARCHAR(255)): 255 bytes
* PostalAddress (VARCHAR(255)): 255 bytes
* ContactPhone (VARCHAR(20)): 20 bytes
* ContactMobile (VARCHAR(20)): 20 bytes
* ContactEmail (VARCHAR(100)): 100 bytes
* DivisionName (VARCHAR(150)): 150 bytes RECORD SIZE = 1363 bytes

1363 bytes (Initial record size) x 17,259,000 (Australians are currently enrolled to vote) = 23,524,017,000 bytes = 23524 MB = Table size in 2022

1.2% growth rate per year x 10 (years)

New Size = Previous Year Size+(Previous Year Size×Growth Rate)

Starting with the initial size in 2022: 23,524,017,000 Bytes

2025: New Size = 23,524,017,000 + (23,524,017,000 × 0.012) = 23,524,017,000 + 282,288,204 = 23,806,305,204 bytes (ELECTION)

2026: New Size = 23,806,305,204 + (23,806,305,204 × 0.012) = 23,806,305,204 + 285,231,662 = 24,091,536,866 bytes

2027: New Size = 24,091,536,866 + (24,091,536,866 × 0.012) = 24,091,536,866 + 288,403,132 = 24,379,939,998 bytes

2028: New Size = 24,379,939,998 + (24,379,939,998 × 0.012) = 24,379,939,998 + 291,659,280 = 24,671,599,278 bytes (ELECTION)

2029: New Size = 24,671,599,278 + (24,671,599,278 × 0.012) = 24,671,599,278 + 295,375,191 = 24,966,974,469 bytes

2030: New Size = 24,966,974,469 + (24,966,974,469 × 0.012) = 24,966,974,469 + 299,603,694 = 25,266,578,163 bytes

2031: New Size = 25,266,578,163 + (25,266,578,163 × 0.012) = 25,266,578,163 + 303,198,938 = 25,569,777,101 bytes (ELECTION)

2032: New Size = 25,569,777,101 + (25,569,777,101 × 0.012) = 25,569,777,101 + 307,086,325 = 25,876,863,426 bytes

2033: New Size = 25,876,863,426 + (25,876,863,426 × 0.012) = 25,876,863,426 + 311,225,861 = 26,188,089,287 bytes

2034: New Size = 26,188,089,287 + (26,188,089,287 × 0.012) = 26,188,089,287 + 315,424,471 = 26,503,513,758 bytes (ELECTION)

After 10 years, the table size will be 26,503,513,758 bytes which will be 26503 MB in 2034

### Initial Table Size – Ballot

* BallotID (INT): 4 bytes
* ElectionEventID(INT): 4 bytes record size = 8 bytes

8 bytes (Initial record size) x 17,806,361 (Australians are currently enrolled to vote assuming they all vote) = 142,450,888 bytes = 135.85 MB = Table size in 2022

4.05% growth rate per year x 10 (years)

New election ballot table size = Previous ballot size (1:1 VoterRegistry)X 1.2%^3(3 = yearbetween election)

Starting with the initial size in 2022: 142,450,888 bytes

142,450,888

New Size = PREVIOUS Electoral pop size X 1.2%^3

1.2 142,450,888 X (1.2 %)^3 = 147,640,904 BYTES

147,640,904 x (1.2%)^3

2025: New Size = 142,450,888 + (142,450,888 × 0.0405) = 142,450,888 + 5,769,313 = 148,220,201 bytes(ELECTION) 1.2% + 1.2% !.2%

2026: New Size = 148,220,201 + (148,220,201 × 0.0405) = 148,220,201 + 6,000,881 = 154,221,082 bytes

2027: New Size = 154,221,082 + (154,221,082 × 0.0405) = 154,221,082 + 6,186,151 = 160,407,233 bytes

2028: New Size = 160,407,233 + (160,407,233 × 0.0405) = 160,407,233 + 6,472,576 = 166,879,809 bytes (ELECTION) 3.6%

2029: New Size = 166,879,809 + (166,879,809 × 0.0405) = 166,879,809 + 6,790,264 = 173,670,073 bytes

2030: New Size = 173,670,073 + (173,670,073 × 0.0405) = 173,670,073 + 7,035,618 = 180,705,691 bytes

2031: New Size = 180,705,691 + (180,705,691 × 0.0405) = 180,705,691 + 7,318,579 = 188,024,270 bytes (ELECTION)

2032: New Size = 188,024,270 + (188,024,270 × 0.0405) = 188,024,270 + 7,615,984 = 195,640,254 bytes

2033: New Size = 195,640,254 + (195,640,254 × 0.0405) = 195,640,254 + 7,927,432 = 203,567,686 bytes

2034: New Size = 203,567,686 + (203,567,686 × 0.0405) = 203,567,686 + 8,253,530 = 211,821,216 byte(ELECTION)

In 10 years, the table size will be 211,821,216 bytes Converted to 202.01 MB

### Initial Table Size – BallotPreferences

* BallotID (INT): 4 bytes
* ElectionEventID(INT): 4 bytes
* Preference varchar(50): record size = 58 bytes

58 bytes (Initial record size) x 17,806,361 (Assuming all registered voters casted a ballot in 2024) = 1,032,768,938 BYTES = 984.93MB = Table size in 2024

Because it should be 1-1 from the ballots table we assume the growth will be the same of 4.05%

4.05% growth rate per year x 10 (years)

New Size = Previous Year Size+(Previous Year Size×Growth Rate)

Starting with the initial size in 2024: 1,032,768,938 bytes

2025: New Size = 1,032,768,938 + (1,032,768,938 × 0.0405) = 1,032,768,938 + 41,324,118 = 1,074,093,056 bytes(ELECTION)

2026: New Size = 1,074,093,056 + (1,074,093,056 × 0.0405) = 1,074,093,056 + 43,000,771 = 1,117,093,827 bytes

2027: New Size = 1,117,093,827 + (1,117,093,827 × 0.0405) = 1,117,093,827 + 45,243,299 = 1,162,337,126 bytes

2028: New Size = 1,162,337,126 + (1,162,337,126 × 0.0405) = 1,162,337,126 + 47,067,648 = 1,209,404,774 bytes(ELECTION)

2029: New Size = 1,209,404,774 + (1,209,404,774 × 0.0405) = 1,209,404,774 + 49,481,885 = 1,258,886,659 bytes

2030: New Size = 1,258,886,659 + (1,258,886,659 × 0.0405) = 1,258,886,659 + 51,495,892 = 1,310,382,551 bytes

2031: New Size = 1,310,382,551 + (1,310,382,551 × 0.0405) = 1,310,382,551 + 53,121,506 = 1,363,504,057 bytes(ELECTION)

2032: New Size = 1,363,504,057 + (1,363,504,057 × 0.0405) = 1,363,504,057 + 55,371,980 = 1,418,876,037 bytes

2033: New Size = 1,418,876,037 + (1,418,876,037 × 0.0405) = 1,418,876,037 + 57,262,461 = 1,476,138,498 bytes

2034: New Size = 1,476,138,498 + (1,476,138,498 × 0.0405) = 1,476,138,498 + 59,810,608 = 1,535,949,106 bytes(ELECTION)

Total of 1,535,949,106 table size in 2034 converted to megabytes = 1,464.80 MB

## Task 2

### 1.

#### Query 1

1. SELECT DivisionName AS DivisionName, COUNT(\*) AS TotalRegVoters
2. FROM VoterRegistry
3. GROUP BY DivisionName
4. ORDER BY DivisionName DESC

#### Index

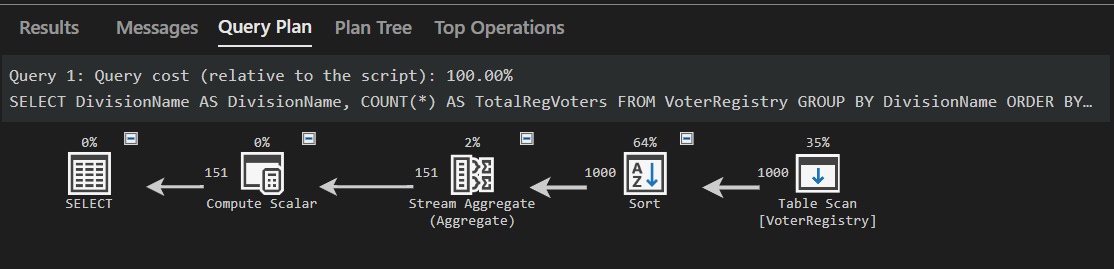
CREATE CLUSTERED INDEX VoterRegistry\_DivN

                ON VoterRegistry(DivisionName);

#### Justification

The Index that I identified that would assist in the query was a clustered index on the DivisonName Column. As a GROUP BY and ORDER BY is utilized, it will efficiently handle these operations by making accessing and data sorting faster as DivisonName is the primary column used in the query.

#### Execution plan (Before)





#### Execution plan (After) 00:00:00.629 00:00:00.017



### 2.

SELECT ElectionEvent.DivisionName AS Division\_Name,  Candidate.CandidateName AS Candidate\_Name, PoliticalParty.PartyName AS Political\_Party, ElectionEvent.ElectionSerialNo AS ElectionSerialNumber

FROM ElectionEvent

JOIN Contests ON ElectionEvent.ElectionEventID = Contests.ElectionEventID

JOIN Candidate ON Contests.CandidateID = Candidate.CandidateID

LEFT JOIN PoliticalParty ON Candidate.PartyCode = PoliticalParty.PartyCode

WHERE ElectionEvent.ElectionSerialNo = '20220521'

ORDER BY ElectionEvent.DivisionName ASC,NEWID();

#### INDEX

CREATE UNIQUE index electioeventIDINDEX

ON ElectionEvent (ElectionEventID);

CREATE CLUSTERED  INDEX EventDivisionNameINDEX

ON ElectionEvent (DivisionName);

CREATE NONCLUSTERED INDEX PolPartyINDEX

ON Candidate (PartyCode);

CREATE UNIQUE CLUSTERED INDEX CandidateIDIndex

ON Candidate (CandidateID)

#### Justification

For this query I decided to create a few indexes to support two main tables used in this query.

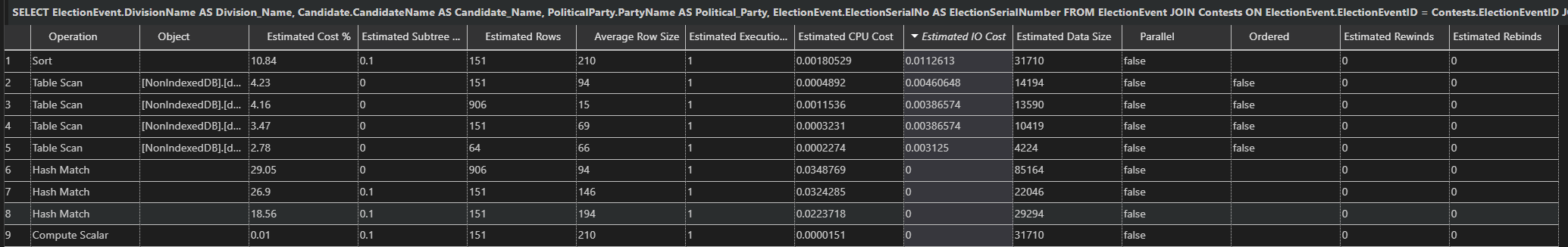
I utilized a unique non-clustered index for the ElectionEventID. As ElectionEventID values are unique and the coloumn is used to identify a specific record for that divison. It will also aid with the query optimizer speeds in retrieving data.

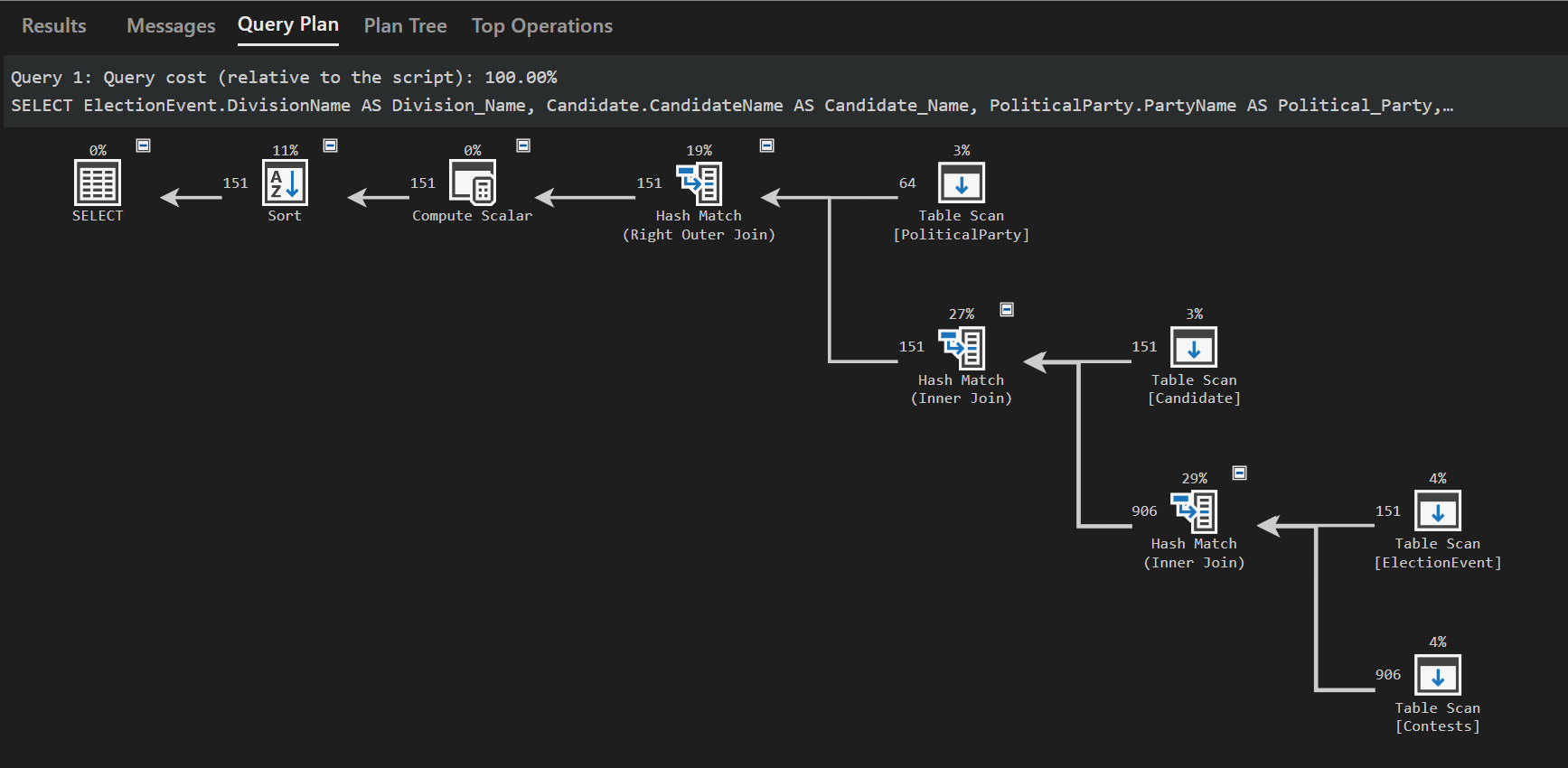
As the DivisionName column is used with an ORDER BY clause of the query I utilized a clustered index for DivisionName. This will help with optimizing the sorting and retrieval of data in the specified order. Since the DivisionName column is not unique across all records but is frequently used in sorting a clustered index improves performance for sorting.

The PartyCode column in the Candidate table is used in joins with the PoliticalParty table to identify the candidates party. I used a non-clustered index on PartyCode because it speeds up these join procedures by allowing the database engine to quickly find then match the records based on PartyCode.

The CandidateID column is the unique identifier per candidate, which identifies each record in the Candidate table. By creating a unique clustered index on CandidateID it optimizes queries that use CandidateID for joins and other related queries that untilise the CandidateID. The unique constraint also enforces data integrity by ensuring no two candidates have the same ID.

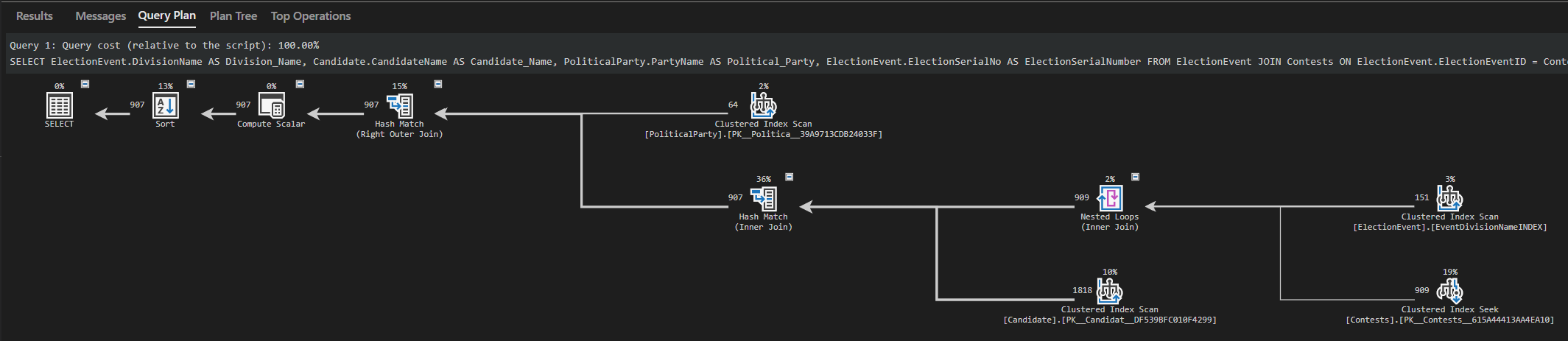
#### Execution plan( Before)





#### Execution plan( After)





## 3.

SELECT FirstName, Address FROM VoterRegistry

WHERE VoterID NOT IN (

SELECT VoterID from IssuanceRecord

WHERE ElectionSerialNo IN ('20220521', '20190518'))

#### INDEX

CREATE UNIQUE CLUSTERED INDEX [Issuance\_VoterIDIndex]

    ON [dbo].[IssuanceRecord]([VoterID] ASC);

CREATE UNIQUE NONCLUSTERED INDEX [IssuanceRecordserialINDEX]

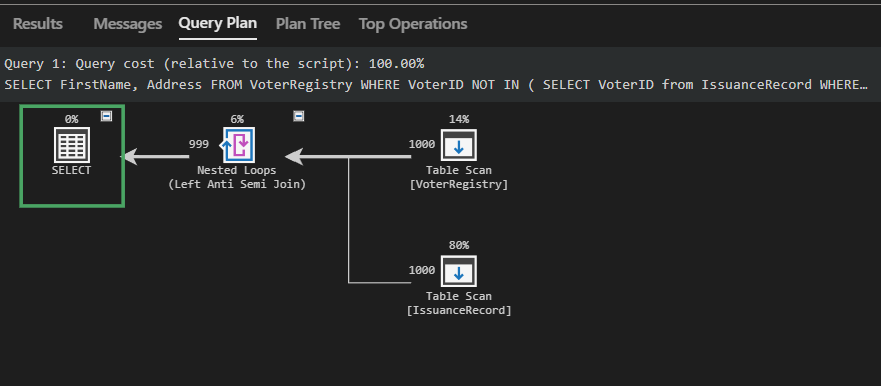
     ON [dbo].[IssuanceRecord]([ElectionSerialNo] ASC);

#### Justification

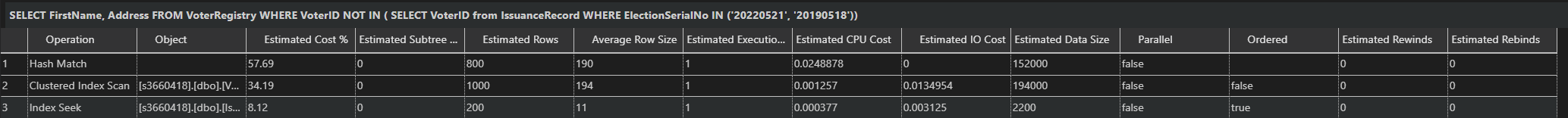
I used a unique clustered index for the VoterID because it will ensure uniqueness of VoterID column as well as optimizing the lookup for the ID of a voter by minimizing the need for additional sorting and searches .

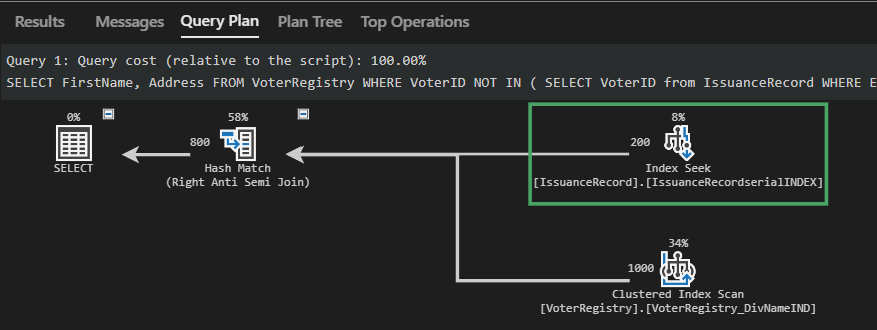
#### Execution plan(Before)





#### Execution plan(After)





### Task 3

The partitioning Method that will be suitable for the “Three extremely large tables” in Task 1 is a Horizontal Partition. Further more we will be going using the Hash partitioning strategy. Due to the contents of the tables not being too vital regarding where they are stored, the hash partitioning is ideal. It will allow the tables to be evenly distributed across the partitions all partitions created improving query performance with parallel processing instead of pulling from all the data from one partition.

This partition method comprises of dividing tables based on rows by using a range or condition. As mentioned from GeeksforGeeks “ It is useful when dealing with tables containing a large number of rows”, It can also be utilized to group certain rows into logical groups if needed. The VoterRegistry table in 2022 is expected to have 17 million rows of registered voters data. Hypothetically if everyone in VoterRegistry vote, there will then be 17 million rows in ballots and BallotPreference.

The Partition keys for the Ballot, BallotPreferences and VoterRegistry table will be the ElectionEventID, Preference and VoterID. These were selected as they were the Primary Keys for the table, as well as the unique identifier for the record.

### DDL

#### Partitioned VoterRegistry

-- Define partition function

CREATE PARTITION FUNCTION VoterPartitionFunction (INT)

AS RANGE LEFT FOR VALUES (1); -- Range boundary value

-- Define partition scheme

CREATE PARTITION SCHEME VoterPartitionScheme

AS PARTITION VoterPartitionFunction

ALL TO ([PRIMARY]); -- Map all partitions

-- Create partitioned table

CREATE TABLE VoterRegistry (

    VoterID INT IDENTITY(1,1),

    FirstName VARCHAR(100),

    MiddleNames VARCHAR(100),

    LastName VARCHAR(100),

    Address VARCHAR(255),

    DoB DATE,

    Gender CHAR(1),

    ResidentialAddress VARCHAR(255),

    PostalAddress VARCHAR(255),

    ContactPhone VARCHAR(20),

    ContactMobile VARCHAR(20),

    ContactEmail VARCHAR(100),

    DivisionName VARCHAR(150),

    HashValue AS (ABS(BINARY\_CHECKSUM(VoterID) % 1)) PERSISTED NOT NULL -- Computed hash value

)

ON VoterPartitionScheme (VoterID); -- Partition scheme used

GO

#### BallotPreferences and Ballot DDL

-- Define partition function for Ballot and BallotPreferences

CREATE PARTITION FUNCTION BallotPartitionFunction (INT)

AS RANGE LEFT FOR VALUES (1)-- Define ranges for hash values

-- Define partition scheme for Ballot and BallotPreferences

CREATE PARTITION SCHEME BallotPartitionScheme

AS PARTITION BallotPartitionFunction

ALL TO ([PRIMARY]); -- Map all partitions to primary filegroup

GO

-- Create Ballot table with hash partitioning

CREATE TABLE [dbo].[Ballot] (

    [BallotID] INT NOT NULL, -- Unique identifier for the ballot

    [ElectionEventID] INT NULL, -- Identifier for the associated election event

    PRIMARY KEY CLUSTERED ([BallotID] ASC), -- Primary key constraint

    HashValue AS (ABS(BINARY\_CHECKSUM(ElectionEventID) % 4)) PERSISTED NOT NULL -- Computed hash value

)

ON BallotPartitionScheme (HashValue); -- Partition scheme based on hash value

GO

CREATE TABLE [dbo].[BallotPreferences] (

    [BallotID] INT NOT NULL, -- Identifier for the ballot

    [CandidateID] INT NOT NULL, -- Identifier for the candidate

    [Preference] INT NULL, -- Preference order

    PRIMARY KEY CLUSTERED ([BallotID] ASC, [CandidateID] ASC), -- Composite primary key

    HashValue AS (ABS(BINARY\_CHECKSUM(Preference) % 4)) PERSISTED NOT NULL -- Computed hash value

)

ON BallotPartitionScheme (HashValue); -- Partition scheme based on hash value

GO

#### Justification

The reason for the design lays within the data that is kept within the tables. All of the data within the tables can be scattered and don’t necessarily need to be ordered, Especially where records in the tables Ballot and BallotPreference need to be anonymous and confidential. By using hash-based partitioning, we ensure that data is evenly spread throughout the available partitions without relying on any specific order.

The RANGE LEFT is used in the PARTITION FUNCTION is for range-based partitioning. Its utilized for the hash-based distribution by creating boundaries that the hash function can use.

Due to computer requirements, I have left the partition store to 1 in the schema. However ultimately if there was more filegroups the design would map the partitions in multiple filegroups. If needed, different filegroups can be used to distribute partitions across improving optimization and query performance.

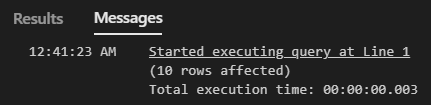
HashValue takes in the computed column, in our case its “ElectionEventID, Preferences and VoteID” which then uses the AS clause to compute the expression that calculates the hash column. Binary\_Checksum (PartitionKey) creates a number that helps identify the Partition Key Value based on its content. Then modular 4 takes Binary\_Checksum which is calculated and gives a output that is used to group data. This value is then used in (hashvalue) parameter on the following line.

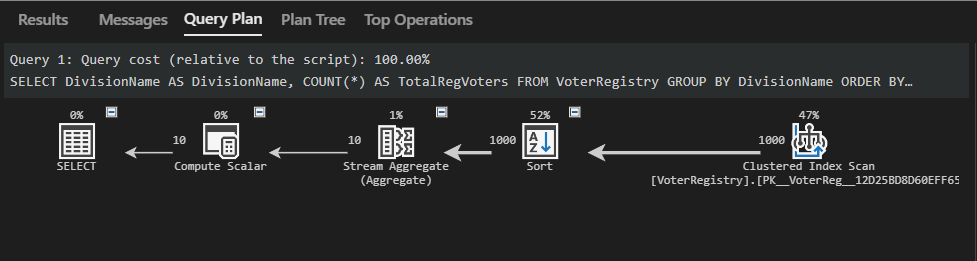
We can utilise Partition pruning to filter through the Partitions based of number preferences for BallotPreferences.Preferences this way we can eliminate the other preferences that are not equal to 1. This will then make 1 partition available to calculate the number of first preferences per candidate. This hypothetically this could be done by using:

SELECT \* FROM BallotPreferences\_Primaryvote PARTITION (1);

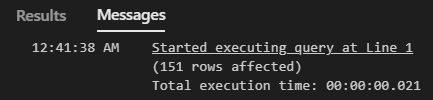
For Queries 1 and 3 for task 2 can benefit from the partition table of VoterRegistry as both queries require a full table scan to produce the result. This can then be utilized for a Parallel SQL execution where threads will work on different parts of the query to produce a faster query time.

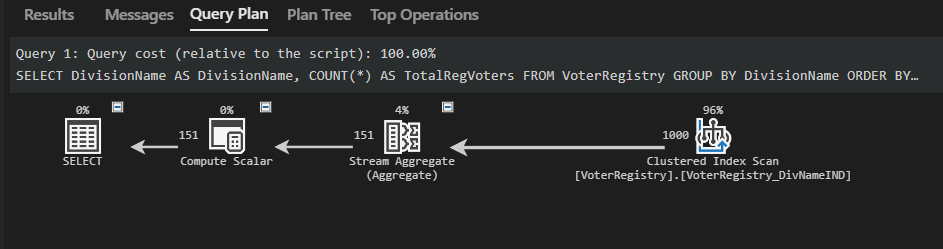
Here is the query time for the partitioned query 1:





Without query 1:





### Task 4

CREATE OR ALTER FUNCTION Previously\_Voted

(

    @VoterIdentification AS INT,

    @DivisionName AS VARCHAR(150),

    @EventSerialNumber AS INT

)

RETURNS VARCHAR(30)

AS

BEGIN

    DECLARE @Previouslyvoted AS VARCHAR(10);

    -- Check if VoterID exists in the IssuanceRecord

    IF NOT EXISTS (SELECT 1 FROM IssuanceRecord WHERE VoterID = @VoterIdentification)

    BEGIN

        RETURN 'VoterID does not exist ';

    END;

    -- Check if DivisionName exists in the VoterRegistry

    IF NOT EXISTS (SELECT 1 FROM VoterRegistry WHERE DivisionName = @DivisionName)

    BEGIN

        RETURN 'DivisionName Does not exist';

    END;

    -- Check if DivisionName matches VoterID

    IF NOT EXISTS (SELECT 1 FROM VoterRegistry WHERE VoterID = @VoterIdentification AND DivisionName = @DivisionName)

    BEGIN

        RETURN 'DivisonName and ID dont match';

    END;

    -- Check if ElectionSerialNo exists

    IF NOT EXISTS (SELECT 1 FROM IssuanceRecord WHERE ElectionSerialNo = @EventSerialNumber)

    BEGIN

        RETURN 'EventSerialNo does not exist';

    END;

    -- Check if the Voter has previously voted in the election

    SET @Previouslyvoted = CASE

        WHEN EXISTS (

            SELECT 1

            FROM IssuanceRecord

            WHERE VoterID = @VoterIdentification

            AND ElectionSerialNo = @EventSerialNumber

        )

        THEN 'TRUE'

        ELSE 'FALSE'

    END;

    RETURN @Previouslyvoted;

END;

GO

-- to run

-- working example

SELECT dbo.Previously\_Voted(1 ,'cowper', 20220521) AS Previously\_Voted;

-- fail example

SELECT dbo.Previously\_Voted(2 ,'Hinkler', 20220521) AS Previously\_Voted;

## Task 5

CREATE OR ALTER PROCEDURE PrimaryVoteCount

AS

DECLARE --declare variables

    @CandidateID INT,

    @DivisionName VARCHAR(100),

    @ElectionEventID INT,

    @roundcount INT;

-- FOR TESTING--

SET @DivisionName = 'Bennelong';        -- Replace with a valid DivisionName T

SET @ElectionEventID = 10;

SET @roundcount = 1;

BEGIN -- input validation -- check if DivisionName exists in electionEvent

        IF NOT EXISTS (SELECT 1 FROM ElectionEvent  WHERE DivisionName = @DivisionName)

        BEGIN

            RAISERROR (' DivisonName does not exist, Try again', 10, 0);

            RETURN;

        END;

        -- check if electioneventID exists in electionevent

           IF NOT EXISTS (SELECT 1 FROM ElectionEvent WHERE ElectionEventID = @ElectionEventID)

        BEGIN

            RAISERROR ('electioneventID number does not exist, Try again', 10, 0);

            RETURN;

        END;

        -- check if eventid and div name exist within each other

         IF NOT EXISTS (SELECT 1 FROM ElectionEvent WHERE ElectionEventID = @ElectionEventID AND DivisionName = @DivisionName)

        BEGIN

            RAISERROR ('electioneventID number does not match divisionName , Try again', 10, 0);

            RETURN;

    END;

WITH GetCandidateID AS  -- CTE to get candidate ID from user inputs on division and serialNO

(  -- SAME A QUEUE 2 QUERY

    SELECT ElectionEvent.DivisionName, ElectionEvent.ElectionEventID, Candidate.CandidateID

    FROM ElectionEvent

    JOIN Contests ON ElectionEvent.ElectionEventID = Contests.ElectionEventID

    JOIN Candidate ON Contests.CandidateID = Candidate.CandidateID

    WHERE ElectionEvent.ElectionEventID = @ElectionEventID

    AND ElectionEvent.DivisionName = @DivisionName

),

primaryVoteCount AS

(

    SELECT BallotPreferences.CandidateID, COUNT(\*) AS VoteCount -- counting preferences

    FROM BallotPreferences

    JOIN GetCandidateID ON BallotPreferences.CandidateID = GetCandidateID.CandidateID -- joining GETCAN CTE with Ballot table to get matching CandID

    WHERE BallotPreferences.Preference = '1' -- counting preferences only with 1

    GROUP BY BallotPreferences.CandidateID  --Grouping  candidates in preferences

)

INSERT INTO PreferenceTallyPerRoundPerCandidate (ElectionEventID,RoundNo,CandidateID,PreferenceTally)

SELECT

    @ElectionEventID,          -- Use the variable for electioneventID

    @roundcount,                    -- Use the variable for RoundNo

    primaryVoteCount.CandidateID,    -- CandidateID from the CTE

    primaryVoteCount.VoteCount       -- VoteCount from the CTE

FROM primaryVoteCount;

WITH EliminatedCandidate AS

(

    SELECT CandidateID, PreferenceTally

    FROM PreferenceTallyPerRoundPerCandidate

    WHERE PreferenceTally = (

    SELECT MIN (PreferenceTally)

    FROM PreferenceTallyPerRoundPerCandidate

)

)

INSERT INTO PrefCountRecord (ElectionEventID,RoundNo,EliminatedCandidateID,PreferenceAggregate)

SELECT

    @ElectionEventID,          -- Use the variable for electioneventID

    @roundcount,                    -- Use the variable for RoundNo

    CandidateID,    -- CandidateID from the CTE

    PreferenceTally --  PREF AGGREGATE from the CTE

FROM EliminatedCandidate;

END;

## References

Australian Electoral Commission. (2019). *Enrolment statistics*. [online] Available at: https://www.aec.gov.au/Enrolling\_to\_vote/Enrolment\_stats/index.htm.

“17mill”

Australian Government Centre for Population (2022). *2022-23 Budget: AUSTRALIA’S FUTURE POPULATION*. [online] Available at: https://population.gov.au/sites/population.gov.au/files/2022-04/2022-23\_budget\_overview.pdf.

1.2%

Australian Electoral Commission. (2021). *Gazetted enrolment (31 December 2021)*. [online] Available at: https://www.aec.gov.au/Enrolling\_to\_vote/Enrolment\_stats/gazetted/2021/12.htm [Accessed 14 Sep. 2024].

4.5% growth statistics

GeeksforGeeks. (2023). *Vertical Partitioning vs Horizontal Partitioning*. [online] Available at: https://www.geeksforgeeks.org/vertical-partitioning-vs-horizontal-partitioning/.

‌

QUERY 1

SELECT DivisionName AS DivisionName, COUNT(\*) AS TotalRegVoters

FROM VoterRegistry

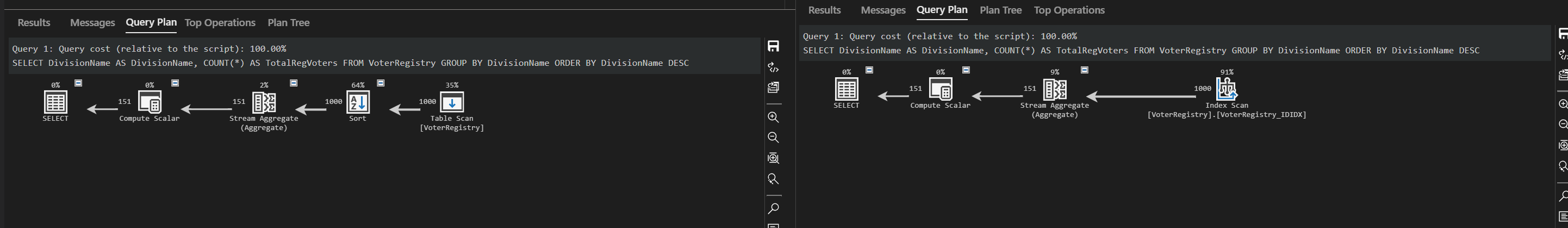
GROUP BY DivisionName

ORDER BY DivisionName DESC;

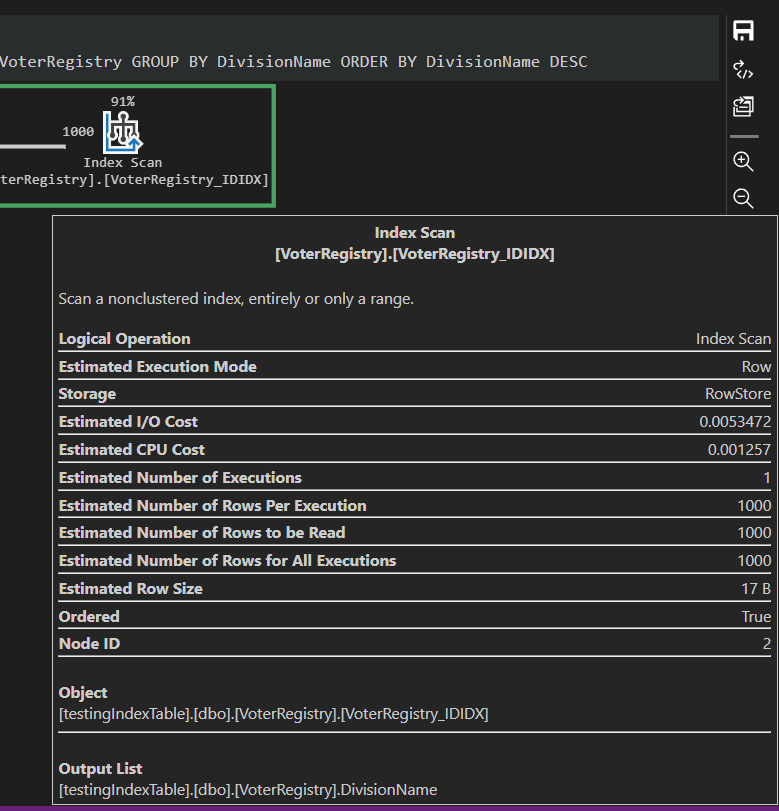
-- can add LIMIT to limit data size

CREATE INDEX VoterRegistry

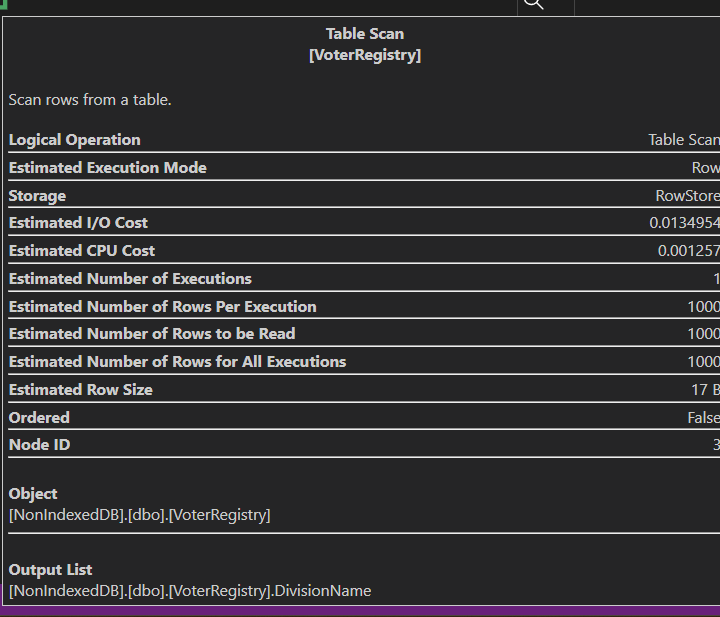
ON VoterRegistry(DivisionName);

Query Plans 

Indexed query



Non indexed query optimization



TASK 4

CREATE PROCEDURE PreviouslyVoted

AS

SELECT

    CASE

        WHEN EXISTS (

            SELECT 1

            FROM IssuanceRecord

            WHERE IssuanceRecord.VoterID = VoterRegistry.VoterID

            AND IssuanceRecord.ElectionSerialNo = '20220521'

        ) THEN 'TRUE'

        ELSE 'FALSE'

    END AS PreviouslyVoted

FROM VoterRegistry;

Task 5