DRAFT DESIGN DOCUMENT

Table of contents

will be implemented. Show at least one example using sample data from the data sets provided in class.

5.1.1. Someone registers to vote.

5.1.2. Someone signs up to run for office.

5.1.3. Someone votes.

5.1.4. Data is accumulated across city, district, county, state

5.1.5. Winner of a town-based election is calculated (most votes, top ***n*** votes)

5.1.6. Winner of a district-based election is calculated (most votes)

5.1.7. Winner of a state-based election is calculated (most votes)

5.1.8. Winner of a presidential election is calculated (most votes per state, how many electoral college votes obtained)

5.1.9. Identify anyone with a fraudulent ballot (define what these might be and how you will detect them).

5.2. Backup, logging and redundancy procedures

5.3. Security procedures

5.3.1. Technological security

5.3.2. Physical security

5.3.3. Authentication

5.3.4. Authorization

5.3.5. Confidentiality

5.3.6. Data integrity

5.3.7. Accountability

5.4. Testing procedures

5.4.1. Benchmark testing

5.4.2. Stress testing

6. Results of project (added at the end of the semester)

1. Introduction

1.1 The problem

For this project, we were tasked with modeling a database with the functionality of simulating the United States National Election system. The process of creating this database led us to think about things that must be considered in order to assure that we can accurately simulate the actual National Election system. Our system must be secure; the National Election system stores sensitive information on this country's voters and candidates and to reveal this information will lead to disaster. Our system must be robust; we must be able to handle both valid and invalid inputs into our system while also assuring that the data that does get introduced into our system is formatted and modelled in such a way that we will be able to query and retrieve it so that we may give the public accurate and correct results on election night. Finally, our system must be efficient; on election day, our system would theoretically be receiving hundreds of millions of votes as the ballots are cast by voters whose information we need to store along with the info of the candidates they are voting for and for what position the candidates are running for multiplied by every voting location across the United States. We must be able to store and retrieve a data set of this magnitude all in one day and this is where we set off in our design.

1.2 Constraints and Assumptions

We are assuming that an area's zip code is the smallest region that will be considered when modeling our system. Within a given zip code, there could be one or many smaller local entities which could lead to overcomplication of the design if it were to be carried out at full scale and, this simplification suffices for a fairly accurate representation of the problem. We are also narrowing our scope to ensure that our system works with only a few zip codes from a few different states and abstracting this to apply for the whole country with only a few foreseen changes, like how a given state handles choosing a winner for a given election for example.

1.3 Acceptable Response Times

Inserting a person's data into the system should take a second at most. Inserting a voter's ballot, parsing, and creating the corresponding votes should be no longer than a second or two. Depending on the query, the acceptable response times could range from a few seconds to several minutes. Querying about an individual's information should not take longer than a second or two while in longest-case, determining the winner of the election for every position being ran for could take several minutes to produce a result from the sheer amount of data and the relationships between data sets.

2. Our Data Model

2.1 Entity/Relationship Model

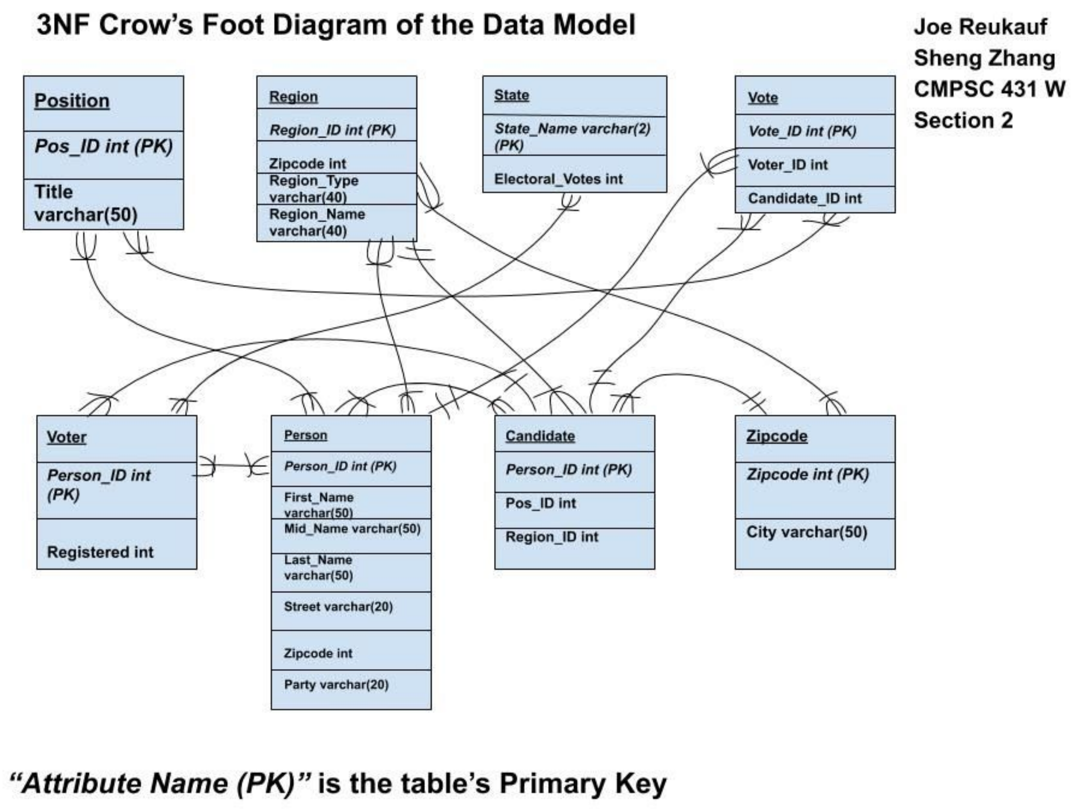

Figure 2.1 Crow's Foot Diagram

2.2 Table Descriptions

TABLE: Position

ATTRIBUTES:

Pos_ID int NOT NULL AUTO_INCREMENT

Title varchar(50)

PRIMARY KEY (Pos_ID)

TABLE: Region

ATTRIBUTES:

Region_ID int NOT NULL AUTO_INCREMENT

Zipcode int

Region_Type varchar(40)

Region_Name varchar(40)

PRIMARY KEY (Region_ID)

TABLE: State

ATTRIBUTES:

State_Name varchar(2)

Electoral_Votes int

PRIMARY KEY (State_Name)

TABLE: Vote

ATTRIBUTES:

Vote_ID int NOT NULL AUTO_INCREMENT

Voter_ID int

Candidate_ID int

PRIMARY KEY (Vote_ID)

TABLE: Voter

ATTRIBUTE:

Person_ID int

Registered int

PRIMARY KEY (Person_ID)

TABLE: Person

ATTRIBUTES:

Person_ID int NOT NULL AUTO_INCREMENT

First_Name varchar(50)

Mid_Name varchar(50)

Last_Name varchar(50)

Street varchar(20)

Zipcode int

Party varchar(20)
PRIMARY KEY (Person_ID)
TABLE: Candidate
ATTRIBUTES:
Person_ID int
Pos_ID int
Region_ID int
PRIMARY KEY (Person_ID)

TABLE: Zip Code
ATTRIBUTES:
Zipcode int
City varchar(50)
PRIMARY KEY (Zip Code)

3. Handling Election Day

3.1 Relevant Technologies

In this section, we will discuss potentially relevant technologies on database management systems that could prove useful in our execution of our system.

3.1.1. Partitioning

Partitioning is a process that when performed on a database means we split large tables into smaller tables that reference the larger data set. By Partitioning our database, we are creating new tables that reference a subset of the total data set which, in some cases, queries we run on the data will not have to scan over the whole data set and could be sped up by only having to reference the partitioned data tables. Overall, this method could speed the querying process which is essential to ensuring that we can produce timely results on election night. So, yes, Partitioning is an appropriate technology to implement into our system.

3.1.2. Striping

Data Striping is a method of separating data in such a way that related pieces of information are stored on different storage devices. This enables a data access to utilize the power of multiple storage disks and decrease the load carried out on any single storage device, increasing performance and speed. A potential risk of Data Striping is since subsequent pieces of data could be stored on separate physical storage devices, the failure of one storage device could mean the corruption of the full data set. Thankfully, this can be solved with the storage of redundant information to combat errors. With all that said, Data Striping is an appropriate technology to implement into our system. The increased performance and speed at the cost of requiring extra storage is well worth it.

3.1.3. RAID

Disk Arrays that implement a combination of data striping and redundancy are called redundant arrays of independent disks, or RAID for short. There are several implementations of RAID, referred to as RAID levels, which vary in their ability to ensure either reliability or performance. In comparison to the other 7 RAID levels, RAID level 6: P+Q Redundancy would be the best choice to implement for our projects. It is described as being appropriate if a higher level of reliability is required. In the context of a national election system, it needs to be ensured that data loss, data corruption, or the misorganization of data does not happen. Our projects are simulating a system that holds mass quantities of sensitive information and important decisions are based off of this data. Reliability may be the most important factor in this consideration.

3.1.4. Sharding

Sharding is similar to Partitioning with one key difference. Partitioning is a general term that in our current context is more suited to mean vertical partitioning which splits large tables into smaller tables by column. Sharding is also “partitioning” these larger tables into smaller tables but by row instead of column, which is referred to as horizontal partitioning. This also has the potential to speed up querying time but to me, it seems that vertical partitioning is more appropriate for this project than Sharding is. In a worst case, a query would have to make a relationship between all of the separate shards to get an accurate view of all of the data which would massively slow response times down. At least with vertical partitioning we still have whole sets of data only separated by what attribute they hold, not whole bits of data strewn about the system in shards.

3.1.5. Data warehousing

Data warehousing is a relational database that is designed for query and analysis rather than for transaction processing. It usually contains historical data derived from transaction data, but it can include data from other sources. This is very appropriate for our project because for one of our tables that we have not included within the 3NF diagram is the History table. This table includes the data for all the previous elections, candidates, votes, years, etc. It is better for the data model to store the information in a data warehouse rather than slowing down the server with all the information from the past elections. This will benefit us because we are still able to reach the information needed to analyze it but not limit the performance of the SQL server. Data warehouse is also nonvolatile and so when the past election's result and information are entering the warehouse, data would not be able to change, so that others would be able to analyze what has occurred.

3.1.6. OLTP

OLTP, or Online Transaction Processing, is a technology mainly utilized in transaction heavy systems in industries such as banking, airlines, and retailers for order entries, retail sales, and financial transactions. From the description of OLTP alone, this technology would not be suited for our system. It is built around the assumption that many users would be able to insert into and pull information out of an OLTP based system on demand where in an election system, many users will indeed be inserting data into our system but they will not be able to retrieve information due to the sensitive nature of the data that will be present that we do not want exposed to the public.

3.1.7. OLAP

OLAP, or Online Analytical Processing is a technology mainly utilized in the reporting of information based on data rather than the retrieval of data like OLTP which was previously discussed. Databases that utilize OLAP use a multi-dimensional model which enables complex analytical queries with a rapid execution time. From the description and contrast from OLTP, OLAP seems to be an appropriate technology to implement into our system. Fast processing time that satisfies the needs of our large data set while fitting a style suited to what we are trying to do, have many users be able to input data on a grand scale but not allow them to access it once it is submitted so that we may report on it.

3.2 Large Spike

Spike or connection spike is very similar to network latency. In essence its the time it takes for a request to travel from the sender to the receiver and for the receiver to process that request it. A spike occurs when a huge amount of data is inserted all at once. In our case, a spike occurred in database insert on election day. By our code, there was no reason to believe that our I/O is the issue that would cause a spike. Hypothetically, all signs indicate of a massive surge in database read/write. In order to find the solution to the sudden spike, we need to eliminate the possibility that it was our queries themselves that were causing the spike. This includes determining whether any strange queries were being run. In our case for the election day spike, we would expect the causes were everyday insert and update queries. There are a few things that could cause a buildup of writes, including poor checkpointing, transaction id wraparounds, and auto vacuums. Bad checkpointing can cause problems since at each checkpoint, all data files will get flushed to the disk. If you let checkpoints build up, that disk flush can get very pricey. This could be solved by simply running the CHECKPOINT command, which forces a checkpoint immediately. This would either lessen the spike or resolve it altogether, if not then transaction ID wraparound is another solution. In essence, for every transaction (generally insert and updates with some occasionally selected) gets assigned a 4 byte integer. The 4-byte integer, after a certain amount of transaction, will need to wrap around, or run out of space to record. The wrapping means that stamping transaction is frozen, which allows all users to see it. Luckily, regular vacuuming takes care of this scenario, since vacuuming will freeze appropriate transaction IDs. The last issues could be the inadvertent heavy writes, which is also called autovacumming. Autovacuum is effectively a daemon process that can't be turned off, and if a big table hits an autovacuum at a peak usage time. It would cause a huge spike or lead to worst detrimental effect. In order to solve this issue, you can tell it to ignore tables and let vaccum know that this could be done later by one's self, in a less peak time. Lastly, a conjecture of indexes could cause spike time on database insert. The sheer number of indexes just overwhelm the system. That is simple in design error so in this case we would have to narrowed it down to a smaller amount of index. Other possible practices to prevent the change of inconsistent state on the disk. This would include locking the whole dataset and possibly doubling down on the RAM. The locking of the data set would increase the uptime and decrease the downtime per day.

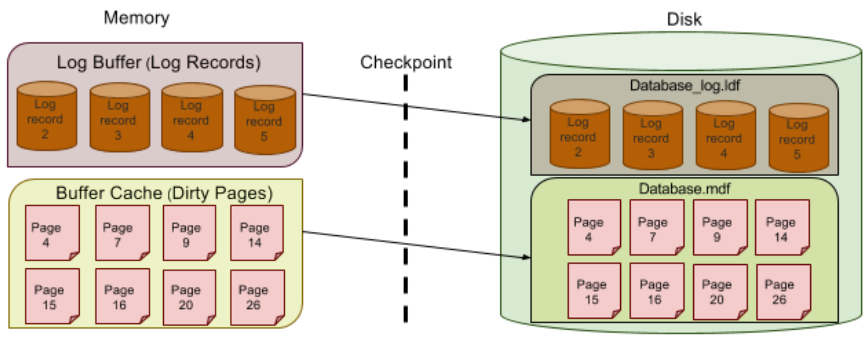

Figure 3.2 Checkpoint

3.3 Necessary Procedures

The amount of data stored in a database has a great impact on its performance. Generally a query becomes slower with additional data in the database. As an example, we analyze the response time of the following query when using two different indexes. In hypothesis the bigger the amount of voters number becomes, the more rows the query selects. In our case we could check the scalability of our database. Scalability shows the dependency of performance on factors like the data volume. A performance value is just a single data point on a scalability chart. The response time of an SQL query depends on many factors. The data volume is one of them. If a query is fast enough under certain testing conditions, it does not mean it will be fast enough in production.

```
+------+------------+--------+-------+------+---------------------
   --+
| type | key        | key_len | ref   | rows | Extra
   |
+------+------------+--------+-------+------+---------------------
   --+
| ref  | scale_slow | 6       | const |    1 | Using index
   condition |
+------+------------+--------+-------+------+---------------------
   --+

+------+------------+--------+------------+-----+-------+
| type | key        | key_len | ref         | rows | Extra |
+------+------------+--------+------------+-----+-------+
| ref  | scale_fast | 12      | const,const |    1 |       |
+------+------------+--------+------------+-----+-------
```

For example, the execution plan are almost identical in the above table. The two components that make an index look up slow would be the table access and scanning a wide index range. Reconstructing the index definitions from the execution plan would help definitely solve our issues with the huge amount of votes from each different polling station.

4. Optimizations

4.1 Modification of 3NF

A database is in third normal form if it is in second normal form and there is no transitive functional dependency. In order to satisfy the condition for the second normal form the database must be in first normal form and all non-key attributes are fully functional dependent on the primary key. While first normal form is formed if it contains only atomic values and there are no repeating groups within the database. Ultimately the root structure must be stable in order for the database to reach third normal form. In our case in the beginning our database was only in first normal form with atomic values and there were no repeating groups among each other. Just listing the attributes that has a relationship with the group from many to many, many to one, and one to one relationship. Eventually we reached second normal form due to modifying all of our non-key attributes to be dependent on the primary key. In the database before the modification of the second normal form, the attributes within voter was only calculating the amount of total votes establish but not relating to each individual that voted. There was no way to differentiate the voters with in the vote. Then eventually we established the third normal form. Within our third normal form database there is no transitive functional dependency. This was done through the elimination of three or more attributes(or database columns) that have a functional dependency between them, meaning Column A in a tables relies on Column B through an intermediate Column C.

4.2 Primary Index

Indexing is the principal technique used to efficiently answering a given query, and an index is a data structure that facilitates the query answering process by minimizing the number of disk accesses. Index on Sequential File, also called Primary Index, is when the Index associated to a Data File which is turn sorted with respect to the search key. A primary index forces a sequential file organization on the data file. Since a data file can have just one order; there can be just one primary index for data file. The primary index "controls" the storage of records in the data file. One great example of primary indexes are hash tables. It implements an associative array abstract data type where it helps map keys to values.

4.3 Secondary Index

Secondary Indexes facilitate query-answering on attributes other than primary keys or, more generally, on non-ordering attributes. A file can have several secondary indexes and it does not determine the placement of records in the data file. The secondary indexes are always dense and it is sorted with respect to the search key i.e binary search. One important thing is though the data file is not sorted with respect to the secondary index search key. In essence the secondary indexes are less efficient than primary indexes.

5. Procedures

5.1 Pseudocode and SQL

5.1.1. Someone Registers to Vote

We take the new voter's identifying information, as defined by the Person Table of our model: first name, middle name, last name, street address, zip code, and party affiliation. We assign that voter a unique person identification number which is simply an auto-incremented integer value. We then take this data and insert it into the Person table as follows:

Person_ID = unique person ID number

First_Name = voter's first name

Mid_Name = voter's middle name

Last_Name = voter's last name

Street = voter's street address

Zipcode = voter's current zip code

Party = voter's affiliated party

Now that their information is in the Person Table, we take their unique Person_ID and create an insert into the Voter Table with an additional value signifying that they are indeed registered to vote.

Person_ID = unique person ID number

Registered = '1' (they are registered to vote)

Example:

Kendahl Ashad Hendrix wants to register to vote. He lives at 138 Gateview 3 Ct in zip code 08721. He is a registered Democrat.

We insert his data into the Person Table:

INSERT INTO Person (First_Name, Mid_Name, Last_Name, Street, Zipcode, Party) VALUES ('Kendahl', 'Ashad', 'Hendrix', '138 GATEVIEW 3 CT', '08721', 'DEM');

On insert he is assigned Person_ID 6922

Then we insert again into the Voter Table:

INSERT INTO Voter (Person_ID, Registered)

VALUES ('6922', '1');

5.1.2. Someone Signs Up to Run for Office

We take the new candidate's identifying information, as defined by the Person Table of our model: first name, middle name, last name, street address, zip code, and party affiliation. We assign that candidate a unique person identification number which is simply an auto-incremented integer value. We then take this data and insert it into the Person table as follows:

Person_ID = unique person ID number

First_Name = candidate's first name

Mid_Name = candidate's middle name

Last_Name = candidate's last name

Street = candidate's street address

Zipcode = candidate's current zip code

Party = voter's affiliated party

Now that their information is in the Person Table, we take their unique Person_ID and create an insert into the Candidate Table.

Person_ID = unique person ID number

Pos_ID = ID of the position they are running for

Region_ID = ID of the region they are located as defined by the Region Table

Example:

Kristoffer Yosef Wise wants to run for Governor of PA. He lives at 127 Acadia Ct in zip code 16593. He is a registered Democrat.

We insert his data into the Person Table:

INSERT INTO Person (First_Name, Mid_Name, Last_Name, Street, Zipcode, Party) VALUES ('Kristoffer', 'Yosef', 'WISE', '127 Acadia Ct', '16593', 'DEM');

On insert he is assigned Person_ID 26853

Then we insert again into the Candidate Table:

INSERT INTO Candidate (Person_ID, Pos_ID, Region_ID)

VALUES ('26853', '15', '0');

5.1.3. Someone Votes

When a voter goes to vote, they're identified by their assigned Person_ID and when they cast their vote for a candidate who is also identified by their Person_ID. With those two pieces, we put them together with the auto-incremented Vote_ID to create a Vote and put it into the table where:

Vote_ID = auto-incremented value

Voter_ID = voter's Person_ID

Candidate_ID = candidate's Person_ID

We need to locate the voter's Person_ID:

```
For Person in PersonTable:
  //Match voter's identifying information with info already in the PersonTable
using string comparisons
  If "voter's info" == "PersonTable info":
          Voter_ID = PersonTable Person_ID
```

We also need to locate the candidate's Person_ID:

```
For Person in PersonTable:
  //Same process as finding Voter_ID except
  Candidate_ID = PersonTable Person_ID
```

We have found what we need from the tables so we insert it into the Vote Table:

```
INSERT INTO Vote (Voter_ID, Candidate_ID)
VALUES ('found Voter_ID', 'found Candidate_ID');
```

Example:

Kellsie Tyka Martinez votes for Ruth Lucille Beil for President

We look up Martinez's Person_ID number to find it is '1'. Then we look up Beil's Person_ID number to find it is '26844'. Then we insert this into the Vote Table.

```
INSERT INTO Vote(Voter_ID, Candidate_ID)
VALUES ('1', '26844');
```

5.1.4. Data is Accumulated Across City, District, County, State

As new data from different locations is accumulated, entries to the Region Table will be made to differentiate between types of elections taking place across America. This differentiation will be made according to an area's zip code and further divided depending on the types of elections taking place in that zip code. This method also helps us organize Candidates when they want to register and run for office.

```
For Region in RegionTable:
          If Zip code is not in RegionTable:
                  If City-Based Election is in Zip code:
                          //Add to Region table (zip, 'City', 'City_Name')
                  If District-Based Election is in Zip code:
                          //Add to Region table (zip, 'Type of District',
'District#')
        If State-Based Election is in Zip code:
            //Add to Region table (zip, 'State', 'State_Name')
```

Example:

There is a race for Mayor of Ocean City, New Jersey. That region hasn't been accounted for in the Region Table so we must add it.

```
INSERT INTO Region (Zipcode, Region_Type, Region_Name)
VALUES ('08223', 'City', 'Ocean City');
```

5.1.5. Winner of Town-Based Election

To determine who wins in any election, we must count the votes cast for each candidate in a given election.

```
For every Candidate in CandidateTable:
    If Candidate is From a Region where Region_Type == "City":
        For every Vote in VoteTable:
            If Vote's Candidate_ID == Candidate's Person_ID:
                Candidate += 1 vote
Return Candidate_ID with greatest number of votes
```

By adding a conditional checking that the candidate is running in a City region, we are only accounting for town-based elections.

```
Select position, first, mid, last, MAX(count)
From (
        Select P.Title as position, N.First_Name as first, N.Mid_Name as mid,
        N.Last_Name as last, COUNT(V.Candidate_ID) as count
        FROM Position P, Person N, Candidate C, Vote V, Region R
        WHERE V.Candidate_ID = C.Person_ID AND C.Pos_ID = P.Pos_ID
        AND V.Candidate_ID = N.Person_ID AND C.Region_ID =
        R.Region_ID AND R.Region_Type = 'City'
        Group By P.Title, C.Person_ID
        ORDER BY count DESC
) as t
Group By position;
```

5.1.6. Winner of District-Based Election

To determine who wins in any election, we must count the votes cast for each candidate in a given election.

```
For every Candidate in CandidateTable:
    If Candidate is From a Region where Region_Type == "StateHouse":
        For every Vote in VoteTable:
            If Vote's Candidate_ID == Candidate's Person_ID:
                Candidate += 1 vote
Return Candidate_ID with greatest number of votes
```

By adding a conditional checking that the candidate is running in a StateHouse region, we are only accounting for district-based elections.

```
Select position, first, mid, last, MAX(count)
From (
    Select P.Title as position, N.First_Name as first, N.Mid_Name as mid,
    N.Last_Name as last, COUNT(V.Candidate_ID) as count
    FROM Position P, Person N, Candidate C, Vote V, Region R
    WHERE V.Candidate_ID = C.Person_ID AND C.Pos_ID = P.Pos_ID
    AND V.Candidate_ID = N.Person_ID AND C.Region_ID =
    R.Region_ID AND R.Region_Type = 'StateHouse'
    Group By P.Title, C.Person_ID
    ORDER BY count DESC
) as t
Group By position;
```

5.1.7. Winner of State-Based Election

To determine who wins in any election, we must count the votes cast for each candidate in a given election.

```
For every Candidate in CandidateTable:
    If Candidate is From a Region where Region_Type == "State":
        For every Vote in VoteTable:
            If Vote's Candidate_ID == Candidate's Person_ID:
                Candidate += 1 vote
Return Candidate_ID with greatest number of votes
```

By adding a conditional checking that the candidate is running in a State region, we are only accounting for state-based elections.

```
Select position, first, mid, last, MAX(count)
From (
    Select P.Title as position, N.First_Name as first, N.Mid_Name as mid,
    N.Last_Name as last, COUNT(V.Candidate_ID) as count
    FROM Position P, Person N, Candidate C, Vote V, Region R
```

```
        WHERE V.Candidate_ID = C.Person_ID AND C.Pos_ID = P.Pos_ID
        AND V.Candidate_ID = N.Person_ID AND C.Region_ID =
        R.Region_ID AND R.Region_Type = 'State'
        Group By P.Title, C.Person_ID
        ORDER BY count DESC
) as t
Group By position;
```

5.1.8. Winner of Presidential Election

To determine who wins in any election, we must count the votes cast for each candidate in a given election.

```
For every Candidate in CandidateTable:
    If Candidate is From a Region where Position Title == "President":
        For every Vote in VoteTable:
            If Vote's Candidate_ID == Candidate's Person_ID:
                Candidate += 1 vote
Return Candidate_ID with greatest number of votes
```

By adding a conditional checking that the candidate is running for President, we are only accounting for presidential elections.

```
Select position, first, mid, last, MAX(count)
From (
        Select P.Title as position, N.First_Name as first, N.Mid_Name as mid,
        N.Last_Name as last, COUNT(V.Candidate_ID) as count
        FROM Position P, Person N, Candidate C, Vote V
        WHERE V.Candidate_ID = C.Person_ID AND C.Pos_ID = P.Pos_ID
        AND V.Candidate_ID = N.Person_ID AND P.Title = "President"
        Group By P.Title, C.Person_ID
        ORDER BY count DESC
) as t
Group By position;
```

5.1.9. Fraudulent Ballot

One type of fraudulent ballot we can account for is a Voter who is not registered, casting votes.

```
For Voter in VoterTable:
    Registered = True
    If Person's info is in VoterTable:
        //They are registered, create their votes
```

```
**Else: //Not in VoterTable**
        //Add their info to the person table
        //Add their info to the voter table with Registered value ‘0’
**If Not Registered:**
        //Either throw away or keep the votes
```

Example:

Koray Brian Rowland is not registered to vote but tries to vote anyway

Check if he is in the Person Table:

SELECT *

FROM Person

WHERE First_Name = ‘Koray’ AND Mid_Name = ‘Brian’ AND … (for all of his identifying information); Returns Empty Set

Add his info to the Person Table:

INSERT INTO Person (First_Name, Mid_Name, Last_Name, Street, Zipcode, Party)

VALUES (‘Koray’, ‘Brian’, ‘ROWLAND’, ‘162 COOPER ALY’, ‘08223’, ‘REP’);

He is given Person_ID number 26868.

Finally we add him to the Voter Table as “unregistered”:

INSERT INTO Voter

VALUES (‘26868’, ‘0’);

5.2 Backup, Logging and Redundancy Procedures

Full backup is the most common backup type and it includes everything including objects, system tables data, and transactions that occur during the backup. With a full backup, one can restore their database to the state, when it was originally backed up. It will not truncate your transaction log but if you database is in full recovery. Transaction log backup would be the better option for backing up to a certain point. Transaction log will backup all transactions that have occurred since the last log backup or truncation, then it will truncation the transaction log. It will capture all transaction information, both DML and DDL, that has occured on the database. The benefit of transaction log backup allows the user to restore a database to a particular point in time aka point-in time recovery, like right before a data loss events. Then there is differential backup, it offers a means to maintain a complete history of your database but without storing redundant data. It is only useful if used in tandem with full backup but allows one to delete/remove previous differential backups as they are redundant. Lastly the file-group and file backup is the best for larger databases. This type of backup will store all related data in files or file groups (one or more). To use file backups to successfully restore a database, the transaction log has to be backed up to cover all of the file groups from beginning to the end. When backing up your SQL database the location where backups reside from the server where the database itself, exists is critical, because otherwise, some failures that affect the database might also mitigate your ability to use the backups to recover from it. This is only the first step, once accomplished, you will want to be able to replicate this process over and over again, automatically and on a schedule. Schedule automated backups is critical for ensuring database continuity while reducing the manual effort required to achieve this. Lastly a backup is only good if it can be restored successfully and this must be verified and continuously re-verified to ensure your backup and restore strategy can be executed successfully when needed. This could be done through verifying the backup was created successfully, if it is currently intact, physically and that all the files not only exist but are also readable, the transaction are consistent and backup can be restored, when it is needed.

5.3 Security Procedures

5.3.1 Technological security

When it comes to database security, it's about securing your SQL server instances and also securing the application which connects to the SQL Server instance. The reason is that data flows take place between SQL server instance and application. The entities participating in this communication must be secured. The entities typically includes database server and instance, clients(i.e Application server or direct client connections), and network connection. To secure the application, do not expose user passwords in code or in external files that are used by the application. Used encrypted connection strings instead. Prefer using Windows Authentication for application service accounts that connect to your SQL Server instance instead of mixed mode. This allows client applications to not send any password to SQL server thus making the process more secure. Establishing an encrypted connection to your SQL server blocks attackers that intercepts the network traffic between your client and the SQL server instance. He or she will not be able to read the data because it would be encrypted data packets.

5.3.2 Physical Security

Securing the physical environment of your database is very crucial. Imagine having your SQL server instance hardened to the maximum security level but leaving the physical location of the database server with weak security. This would truly be a contradiction. As such, one would need to limit the physical access to the physical database server. To achieve this, one must establish the proper procedures to be followed with adequate controls in order only authorized personnel to have physical access to the servers.

5.3.3. Authentication

There is authentication mode when installing SQL server. The authentication mode is more secure because with Windows Authentication, SQL server validates the user's credentials using the Windows principal token in the operating system. This makes the whole process more secure because the authentication is made using the security protocol.

5.3.4. Authorization

Authorization allows object owners to use Grant and Revoke SQL statements to set the user privileges for specific database objects or for a specific SQL actions. The can also use their roles to administer privileges. The Grant statement can assign permissions to a group or role that can be inherited by the database users. However the Deny statement takes precedence over all permission statements. Therefore a user who has denied permission cannot inherit it from another role.

5.3.5. Confidentiality

To provide confidentiality a flexible fine-grained encryption framework is proposed which allows data owners to construct, a from a set of access policies, a single encryption database that can be stored and exchanged by all parties. Access is granted by separate disseminating keys.

5.3.6. Data integrity

To provide integrity, an efficient authentication mechanism is described which can be used to detect tampering when data is stored by an untrusted database. One form of example would be ensuring that you are connecting with SQL server using Windows Authentication, not SQL Authentication; which requires a username and password to be provided with the connecting string.

5.3.7 Accountability

Accountability ensures that unauthorized or suspicious application based user activity can be tracked and audited with enough veracity and detail to be used as legal evidence. It must be able to associate with specific database transactions with individual application end users.

5.4 Testing Procedures

5.4.1. Benchmark testing

Benchmark is a test at higher levels of predefined load for comparison against a baseline. For example, a runner in a marathon might see a heart rate of 110 to 130. The marathon is the athlete's performance benchmark, and from that benmark they can extrapolate many other conclusions about their overall physical fitness and maximum potential. There are software applications used as performance testing tools including CPU-z, CrystalDisk Mark, PassMar PerformanceTest etc.

5.4.2 Stress testing

Stress testing is simply running a number of statements/transactions against a SQL server database from various connections or threads all to give the database server a good routine. It can be used for performance tuning, to uncover bottlenecks, and for capacity planning among other things. Stress testing may occur because of poor performance on a production server, or trying to plan for an increase of users, or trying to workout a new application that is no yet “live” to determine if it will be able to handle a production load. In order to perform a stress test, one must needs to know the conditions before running the test. For example, how many virtual users will you want to test with, what are the main SQL statements and procedures that you will execute during the test, how long should each statement or procedure take to execute, what location do you want to run the stress test and many more. While giving our server a workout we must gather performance related statistics. For these statistics there must be something to be measured against and also comparing the results to a preset thresholds listed by Microsoft and industry professionals.

6. Results of the project (added at the end of the semester)