TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Gregor Tammert 179562IADB

Data Validation Optimization in Teradata-based Data Warehouse

Bachelor's thesis

Supervisor: Nadežda Furs
MBA

Tallinn 2021

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Gregor Tammert 179562IADB

Andmete valideerimise optimeerimine Teradata-põhises andmelaos

Bakalaureusetöö

Juhendaja: Nadežda Furs
MBA

Tallinn 2021

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Gregor Tammert

13.05.2021

Abstract

The overall goal of this thesis is to investigate different ways of how to optimize data validation process in a Teradata-based data warehouse.

Optimization topics dicussed during this thesis can be divided into two categories: database-side (administrative) and client-side optimization. This paper focuses mainly on the client-side optimization. After doing the research the author developed a prototype which was used to measure and prove the effectiveness of the presented optimization aspects.

This thesis was done in collaboration with one financial institution which provided the required environment for development and experimentation. One goal of this thesis was to come up with a solution that could outperform the system currently used in the mentioned institution.

This thesis is written in English and is 28 pages long, including 8 chapters, 6 figures and 6 tables.

Annotatsioon

Andmete valideerimise optimeerimine Teradata-põhises andmelaos

Käesoleva lõputöö eesmärgiks oli esmalt uurida ning seejärel välja pakkuda erinevaid viise, kuidas oleks võimalik optimeerida andmete valideerimise protsessi Teradata-põhises andmelaos.

Töös esitatud optimeerimise aspektid saab jagada kaheks: andmebaasi-poolsed (administratiivsed) ja klientrakenduse poolsed. Töö põhirõhk on asetatud klientrakenduse poolsele optimeerimisele, mille raames arendatakse töö käigus välja prototüüp, mille abil saab katsetada ning mõõta erinevate väljapakutud aspektide efektiivsust.

Töö valmis koostöös finantsettevõttega, mis pakkus autorile vajaliku keskkonna nii arendustööks kui ka eksperimenteerimiseks. Lõputöö üheks eesmärgiks oli luua süsteem, mis oleks oma jõudluselt etem, kui ettevõttes hetkel töös olev lahendus. Uurimistulemused on ettevõttele sisendiks uue süsteemi ehitamiseks, mille üheks osaks on andmete valideerimine.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 28 leheküljel, 8 peatükki, 6 joonist, 6 tabelit.

List of abbreviations and terms

.NET	is an open source developer platform, created by Microsoft, for building many different types of applications
AMP	Access module processor
CLI	Command-line interface
DAL	Data access layer
DI	Dependency Injection
EF	Entity Framework
ERD	Entity relationship diagram
GUID	Globally Unique Identifier
MPP	Massively parallel processing
MS	Microsoft
SMP	Symmetric multiprocessing
SQL	Structured Query Language
TD	Teradata
TPL	Task Parallel Library

Table of contents

List of figures

List of tables

1 Introduction

Throughout the day we make many decisions relying on previous experience. Our brains store trillions of bits of data about past events and leverage those memories each time we face the need to make a decision. Like people, companies generate and collect tons of data about the past and this data can be used to make better decisions. [1]

Business decisions are desired to be based on the most trustworthy data available. One factor to contribute to that matter is data validation which tries to eliminate incorrect records from the data set using predefined constraints. That process is quite essential and should be done on regular basis, luckily enough this can be automated. The author of this thesis is currently employed at an enterprise where a solution for the described process is already implemented but the author sees that it is acting rather poorly and could be improved or replaced by another system.

Purpose of this thesis is to do research and come up with a solution that first would outperform the enterprise's current solution and secondly would give an overall analysis which aspects of developed solution prove to be beneficial. This paper will be one of the inputs for developing a testing framework inside the named enterprise and as it is an international institution the research is documented in English.

The solution is directed towards Teradata Database and .NET Provider for Teradata. Practical part of thesis is to develop a prototype that later could be used for collecting performance data of the developed solution so that afterwards both qualitative and quantitative analysis could be done. Goal of this thesis is not to test Teradata Database's limits but rather to attain an optimized solution between the duration of validation and use of database resources.

2 Teradata Database

Following chapter is dedicated for giving a selective overview of Teradata Database's architecture and terminology so that optimization topics discussed later in this thesis would be more comprehensible to the reader.

2.1 Architecture

In this chapter is described a small but essential portion of Teradata Database's architecture specifically there will be description of two virtual processors Access Module Processor and Parsing Engine.

2.1.1 Hardware

The essence of Teradata Database is parallel processing, for achieving that, it demands a particularly dedicated processing hardware which consists of two main components: processor node(s) and BYNET.

Processor nodes are based on Symmetric Multiprocessing (SMP) technology. The hardware (nodes) can be combined with a communications network (BYNET) that connects the SMP systems to form Massively Parallel Processing (MPP) systems. [2]

2.1.2 Virtual Processors

The versatility of Teradata Database is based on virtual processors (vprocs) that eliminate dependency on specialized physical processors. Vprocs are a set of software processes that run on a node under Teradata Parallel Database Extensions (PDE) within the multitasking environment of the operating system. [2] There are several types of vprocs but regarding this thesis one would be particularly interested in AMPs and PEs.

As stated in the documentation [2], **access module processors** (AMP) perform different database functions e. g. executing database queries. Each AMP owns a portion of the overall database storage meaning that rows of one table can and should be distributed between different AMPs to increase the benefit of parallel processing.

The **Parsing Engine** (PE) is the vproc that communicates with the client system on one side and with the AMPs (via the BYNET) on the other side. Each PE executes the database software that manages sessions, decomposes SQL statements into steps, possibly in parallel, and returns the answer rows to the requesting client. [2] One element of PE is Optimizer which role and functions are discussed later in upcoming chapters of this thesis.

Considering the information provided previously the following is a simple high-level illustration of the communication between different logical units in Teradata Database when processing a query.

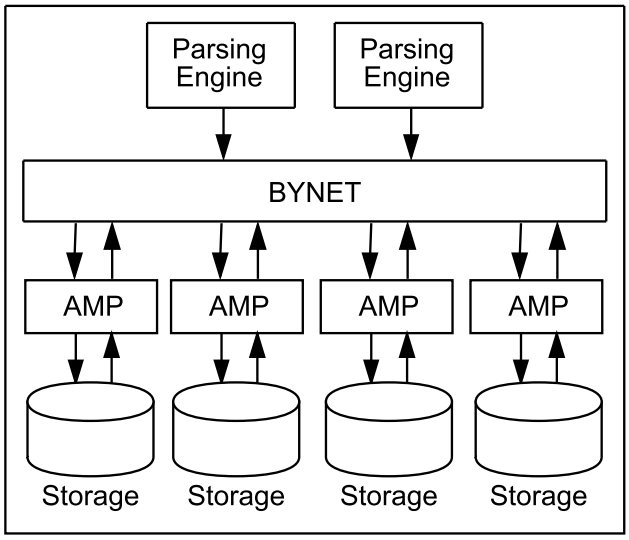


Figure 1. Simplified Teradata query processing [2]

2.2 Terminology

This chapter provides explanations for several Teradata specific terms that are related to the scope of this thesis.

2.2.1 Spool space

While processing user requests Teradata Database uses spool space as temporary storage for intermediate and returned rows. It is worth mentioning that spool space is allocated from user's permanent space and is distributed evenly between all AMPs. When use of spool space exceeds in one of the AMPs then query execution is aborted and error message is returned instead. In addition to that all other active queries for that user get aborted as well. [3]

As an example, one can imagine that there is a system which has 400 AMPs. User is given a total of 100GB of spool space which means that every AMP has access to 250MB of that memory. It can be deduced that it is very important that data is distributed evenly between the AMPs otherwise queries can easily run into spool space issues.

2.2.2 Skew

Skewness is the third moment of a distribution. It is a measure of the asymmetry of the distribution about its mean compared with the normal, Gaussian, distribution. The normal distribution has a skewness of 0. [4]

The computing of SKEW is defined as follows:

$$\text{SKEW} = \frac{\text{COUNT(x)}}{(\text{COUNT(x)}-1)(\text{COUNT(x)}-2)} \bullet \text{SUM}\left(\frac{\text{x}-\text{AVG(x)}}{(\text{STDDEV\_SAMP(x)}**3)}\right)$$

Figure 2. Skew calculation using SQL aggregates [4]

Where *x* is a value expression and STDDEV_SAMP is function for calculating the sample standard deviation for the non-null data points in *x*. [4] This is useful for finding skew based on column value distribution.

In appendix 2. author has also provided a query for detecting skewed tables on a database level which is based on the current permanent space usage per AMP and where ideal and minimum value of column "Custom Skew Factor" is 1.

2.2.3 Statistics

The COLLECT STATISTICS statement collects demographic data for one or more columns of a base table, hash index, or join index, computes a statistical profile of the collected data, and stores the synopsis in the Data Dictionary. The Optimizer uses the synopsis data when it generates its table access and join plans. [5] So it can be said that there is a direct link between statistics collection and database performance.

2.2.4 Workload

A workload is a class of database requests with common traits whose access to the database can be managed with a set of rules. Workloads are useful for: [15]

- Setting different access priorities for different types of requests
- Monitoring resource usage patterns, performance tuning, and capacity planning
- Limiting the number of requests or sessions that can run at the same time

In the context of this thesis, the database user running the validations is classified under a workload that allows maximum number of five parallel requests.

3 Data validation

As data validation is quite broad topic and this thesis focuses mainly on optimization rather than on the concept itself then this chapter gives only a brief overview and main takeaways for performing data validation on a relational database level. If the reader finds itself in a position where implementation of data validation is needed at any scale, then the author recommends delving into official paper [8] provided by ESSnet (European Statistical System).

3.1 Definition and foundation

Data validation is an activity verifying whether or not a combination of values is a member of a set of acceptable combinations. Data validation assesses the plausibility of data: a positive outcome will not guarantee that the data is correct, but a negative outcome will guarantee that the data is incorrect. Data validation is a decisional procedure ending with an acceptance or refusal of data as acceptable. The decisional procedure is generally based on rules expressing the acceptable combinations of values. Rules are applied to data. If data satisfy the rules, which means that the combination expressed by the rules is not violated, data are considered valid for the final use they are intended to. [8]

3.2 Validation from perspective of relational databases

Because validation rules can vary between different requisitions and data sets then regarding the scope of this thesis it is needless to discuss them in detail but there are still two statements that can be postulated about them. Firstly, regarding relational databases every validation rule must be able to express itself in a form of SQL. Secondly instead of returning rule violating records the SQL should return a scalar value that indicates the result of validation. This significantly reduces the amount of data transmitted to client which in turn improves the overall validation process. In addition to that a single scalar value can be more informative regarding data validation and it is more humanly readable.

4 Optimization

This thesis assumes that there are two contributors to data validation process, one being the Teradata Database itself and the second one being the client-side application which is dedicated for orchestrating the overall process and persisting the outcome, therefore the optimization topics discussed next can be split accordingly. As the subject of this thesis is database optimization then its inevitable to discuss optimization topics from a perspective of database administrator, as a healthy database is a preliminary for client-side optimization.

4.1 Teradata Database optimization

Following chapter gives overview and explanation of some techniques for optimizing Teradata Database to improve data querying performance. This applies to all queries not just only for data validation ones. Techniques are Teradata Database specific and essential SQL optimization techniques are not relevant at this time.

4.1.1 Even data distribution

As it was previously stated Teradata's performance comes from parallelism. This chapter first tries to illustrate why it is important that data are evenly distributed in AMPs and then later to give guidelines for achieving that.

For explaining the importance of data distribution between AMPs the author has decided to take into play a quite primitive but easily understandable real-life parallel. In that example there is a restaurant which represents a database which has an object stored within it where each row represents a guest for the current night. Every guest had to pick their choice of course from tonight's selection. There's database user named 'KITCHEN' that needs that information to start preparing the dishes, for obtaining that info it must do a query. For processing that query the restaurant has waiters which in this case represent AMPs. On average it takes one waiter 12 seconds to ask guest for its choice of course.

During the current night there are total of 48 guests and 6 waiters at duty. Author has described four different scenarios:

1. For unknown reason only one waiter can serve the guests
2. Waiter no. 6 falls ill and waiter no. 5 will take care of its duties
3. Waiter no. 6 falls ill and its duties are evenly distributed between all other waiters
4. Guests are evenly distributed between all six waiters

In preceding scenarios, the skew factor reduces per scenario number increment, therefore the time consumption to serve all the guests should also appear in descending fashion. After doing the calculations, the processing of KITCHEN's request will take approximately:

1. 576 sec.
2. 192 sec.
3. 115.2 sec.
4. 96 sec.

In addition to time consumption one must consider that putting too heavy load on single waiter can cause burnout. A query that references a skewed table may try to process more rows on some AMPs than others and may run out of spool space. [9]

The cause of skewed tables is inappropriate indexing. Defining the primary index (PI) for a table is the most critical aspect of table design. The system uses the PI to assign each data row to an AMP. A well-chosen PI balances the distribution across AMPs and helps optimize the performance of queries that access the table. PIs also enable efficient joins and aggregations. For guidelines on choosing an effective PI the author suggests referring to chapter "Choosing a Primary Index" in the Teradata Vantage™ Database Administration documentation. [9]

4.1.2 Statistics collection

Collecting statistics provides the Optimizer with the data demographics it needs to generate good query plans. The more accurate and up to date the statistics, the better the Optimizer can decide on plans and choose the fastest way to answer a query. The computed results are stored in the Data Dictionary DBC.StatsTbl for use during the optimizing phase of statement parsing. [9]

It is worth mentioning that optimization that can be done on Teradata Database side is an ongoing process and as the data are constantly moving then from time to time there is a need to come back to previously optimized objects and redesign them.

4.2 Client-side optimization

In contrast to Teradata Database side optimization, techniques and aspects discussed in this chapter are persistent, meaning that once they are set there is no need to alter, modify or reapply them in the future. It should be noted that this chapter's content is oriented towards .NET Data Provider for Teradata and C# but most of the concepts described can be translated to other technologies or programming languages as well.

4.2.1 Limiting Factors & Requirements

Before elaborating on client-side optimization topics it should be mentioned that there are some limiting factors and requirements to bear in mind while planning the solution:

- Max spool space allocated to database user
- Max number of allowed Teradata connections/sessions
- Validation results must be saved/reported

4.2.2 Connection pooling

A connection in the .NET Data Provider for Teradata is managed by the TdConnection class. When TdConnection.Open is called, a connection is established to the Teradata Database specified in the connection string using the Data Source attribute. When a connection is established, a Teradata Session is opened, and internal objects are initialized. Each connection manages one Teradata Session. Each time an application calls TdConnection.Open to open a connection to a Teradata Database, the provider goes

through the overhead of opening a Teradata Session and initializing internal objects associated with the connection. For applications that continually open and close connections, this overhead will significantly degrade performance. To reduce this overhead, the provider supports Connection Pooling. [6] It is possible to configure max pool size parameter which as the name suggests limits the number of connections in a pool. This parameter is a good way to control application's load on database. Max pool size should always be less than or equal to max allowed sessions for current database user.

4.2.3 Parallelism

Validation of a single rule on a data set (object) is a totally independent process meaning that other validations do not depend on it. This gives an opportunity to execute validations in parallel but one caveat being that there is a high chance of running out of resources, mainly out of spool space, so this issue must be addressed. Author suggests executing validations in a fixed-size heap, that way extensive use of resources can be more or less avoided.

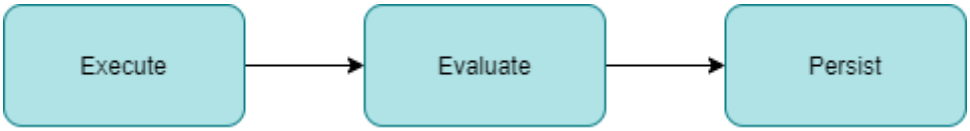


Figure 3. Validation steps (high-level)

Validation process consists of three main steps where step Evaluation is the least troublesome because it is the only one that does not use or depend on any external resource. Execution step of course depends on Teradata Database and step Persist depends on a filesystem or database where results are stored. It seems that most of the heavy lifting is done via communication between client application and data stores so in conclusion the less amount of connections that need to be established and requests that need to be sent the more efficient is the outcome.

The .NET Data Provider for Teradata can support multiple concurrent connections (TdConnection) to Teradata Database. Each connection can be used by one thread at any moment in time and instance members are not guaranteed to be thread safe. For example, two threads should not try to execute two separate commands (TdCommand) simultaneously against the same connection. However, two threads can execute two separate commands (TdCommand) associated with two separate and distinct connections

(TdConnection). [6] This means that to achieve thread safety parallelism every query execution must establish a new connection or take existing one from the pool.

4.2.4 Data persistence

When it comes to data persistence then there is also an optimal balance that should be met. It is rather useless to persist every validation result one-by-one instead it should be done in a batch with configurable size that should not be too large otherwise when a transaction fails then there will be plenty amount of validations that need to be re-executed. Practical example of data persistence solution using Oracle Database and Entity Framework Core is given in chapter ‘Prototype’.

5 Prototype

Following chapter gives an overview of TdValidator, a client-side prototype, that was developed regarding this thesis. From here on term prototype and TdValidator denote to the same subject and their use is alternating. This chapter intertwines aspects discussed earlier in the Optimization chapter with an actual implementation. In addition, there are given annotations about variables that could possibly influence validation performance and therefore should be measured during the experimentation phase which is described in detail in the chapter Experiment.

5.1 Architecture

In this chapter author describes the developed prototype in a conceptual manner leaving out actual technological implementations.

5.1.1 Layers

Application is divided into three layers: Presentation, Service and Data Access Layer. Following is a high-level visualisation of application components based on Traditional "N-Layer" architecture. [10]

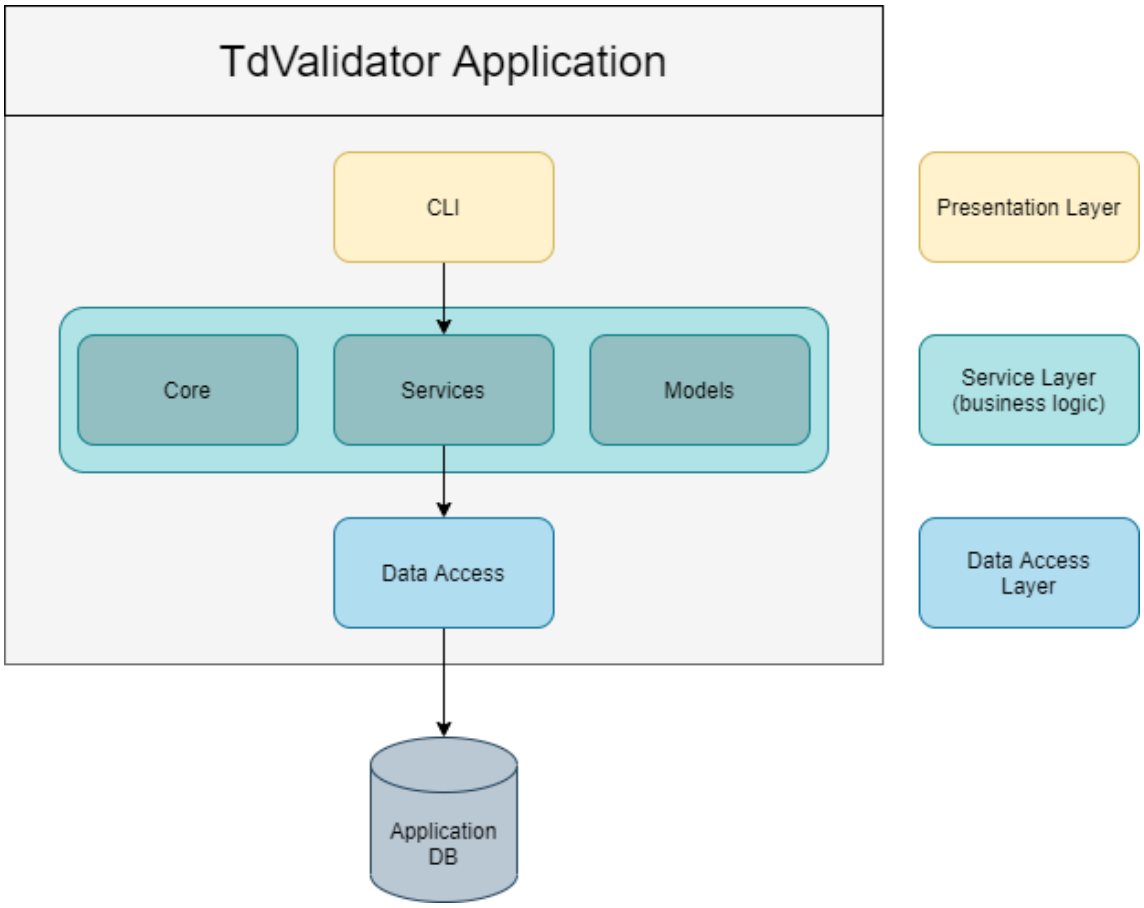


Figure 4. N-layer architecture

Presentation Layer's purpose is to provide an interface to the end-user for interacting with the application. Presentation Layer communicates with Service Layer only via Services module. Regarding TdValidator the CLI must be capable of starting validation process for specific project and logging the process with its results. In addition to that there should be a convenient way for seeding test data.

Service Layer contains the business logic of the solution and is divided into three modules: Core, Services and Models. Core module contains logic that fulfils application's business needs. Models module contains code representation of business domain entities that are persisted to data store. Services module acts as a middleman between user interfaces (Presentation Layer), business logic (Core) and Data Access Layer.

Data Access Layer provides interfaces for Services module to interact with application data store. It also provides a business transaction solution to avoid persisting partial data.

5.1.2 Data model

This chapter gives description of domain entities presented in the data model. ERD can be found in Appendix 3. In following text business domain entities are written in **bold** text.

Component represents a TD Database object which has environment identifier attached to it which is taken into account when generating or executing validation queries.

Project, which is created by end-user, is an entity which has unique name and acts as a logical unit of data validations. There is a many-to-many relationship between **Component** and **Project** which results in **ProjectComponent** entity.

ValidationRule entity stores within itself a query template that is used when generating an actual validation query. It has also Operator, Operand, ValueType columns that work in conjunction and are used for evaluating query execution results.

ComponentValidation has execution ready query attached to it which is generated from **ValidationRule's** query template. Parameters that need to be bound to query are stored as **QueryParameter** entities and relationships between rules and parameters are stored using **RuleParameter** entity.

ValidationResult is artifact created during validating a single **ComponentValidation**. Every result contains evaluation status, query execution result and duration.

5.1.3 Main process and classes

In this chapter author describes the purpose of different business logic classes. It is suggested to refer to Appendix 4. where main process flow of data validation is presented. In addition to that in Appendix 5. main elements of the Core module are visualised on object-oriented level. Class names in following paragraphs are written in **bold** text and references to method names end with open and closing parenthesis.

ValidationQueue is a conventional queue data structure that has possibility to enqueue and dequeue validations. At the start of validation process all the validations of a project are enqueued, and then dequeued whenever there is an empty spot in the execution heap which size is defined in the validation profile.

TdQueryExecutor is for executing queries against TD Database. It is instantiated by passing in a connection string that should not be altered during the lifecycle of the instance. It has a method ExecuteScalar() that is used for executing validation queries.

QueryExecutionManager as the name insists manages the query execution. It controls the execution heap size and provides functionality for scheduling and executing validation queries. By scheduling is meant to start a new parallel query execution. It has property OnTaskComplete which is called after query execution completes.

ResultEvluator is used for evaluating **ValidationResult** based on input scalar value. As evaluation process is executed in parallel this class also keeps track of evaluation workers (parallel running tasks). **ResultEvaluator** also has property OnTaskComplete which is a method declaration that worker calls when it has finished the task. OnTaskComplete takes in a **ValidationResult** as an argument.

BatchManager takes care of organizing batches of validation results. AddToBatch() method adds validation result to current batch. CompleteBatch() method creates a new batch if specified and starts a worker (parallel task) that calls delegate stored in OnTaskComplete property.

Through **ValidationProfile** it is possible to configure the overall validation process. Configurable properties are: ConnectionPooling (boolean), MaxPoolSize, HeapSize, BatchSize. For gaining overview of how those properties impact validation process, author suggests referring to chapters Experiment and Analysis.

TdDataValidator is a class that assembles all the classes previously mentioned into one functional unit. It is instantiated by passing in a connection string and validation profile. Method Validate() takes in a Project entity, validates all of its **ComponentValidations** and creates a **ValidationReport** which is a data structure containing info about single validation run. It contains following:

- Run ID – GUID of validation run
- Project ID
- Profile used

- Validation results
- Validation start and end datetime
- Validation time report (total time and absolute times per validation step)

5.2 Implementation

In this chapter author describes the actual implementation of the planned solution. It should be noted that the architecture presented earlier can be implemented using various technologies, but the ones used regarding this thesis were solely chosen based on the author's previous experience and products available in the enterprise.

5.2.1 CLI

The command-line interface was built as .NET 5 console app that implements a generic host. Generic host enables application to use built-in functionality such as: Dependency Injection, Logging and Configuration. [11] Through dependency injection the CLI can access application services that first need to be registered in the host configuration.

In addition to that the CLI implemented Microsoft.Extensions.CommandLineUtils NuGet package which can be used to create conventional console applications with ease. By conventional it is meant a console app that is capable of parsing out commands, arguments and options from user input.

```
Usage:  validate [arguments] [options]

Arguments:
  PROJECT_ID  Id of project that defines components to be validated

Options:
  -e | --env        Environment where to run validations
  -p | --profile    Validation profile to use. If not provided then 'Default' is used
  -? | -h | --help  Show help information
```

Figure 5. Console output of validate command with help option

Validation can be initiated by passing in identifier of validation project and target environment. There is also an option for selecting the validation profile which is loaded from configuration file called appsettings.json.

5.2.2 Data Access Layer

The DAL was implemented as .NET 5 class library using Entity Framework Core in conjunction with Oracle Data Provider. Implementation was done by following two complementing architectural patterns: Unit of Work and Generic Repository.

At the implementation level, a repository is simply a class with data persistence code coordinated by a unit of work when performing updates. [12] Generic repository pattern is just a further development of it which provides one common repository that through generic input type parameter specifies the underlying domain entity for the repository instance. So, all the repositories created regarding this thesis derived from one common parent class RepositoryBase<TEntity> where TEntity is the generic type described above.

A Unit of Work keeps track of everything done during a business transaction that can affect the database. When the transaction is finished, it figures out everything that needs to be done to alter the database accordingly. [13] That way it is ensured that data integrity is remained between transactions and data corruption is avoided. Following is a illustration of the structure of DAL where the amount of repositories is truncated for simplicity's sake.

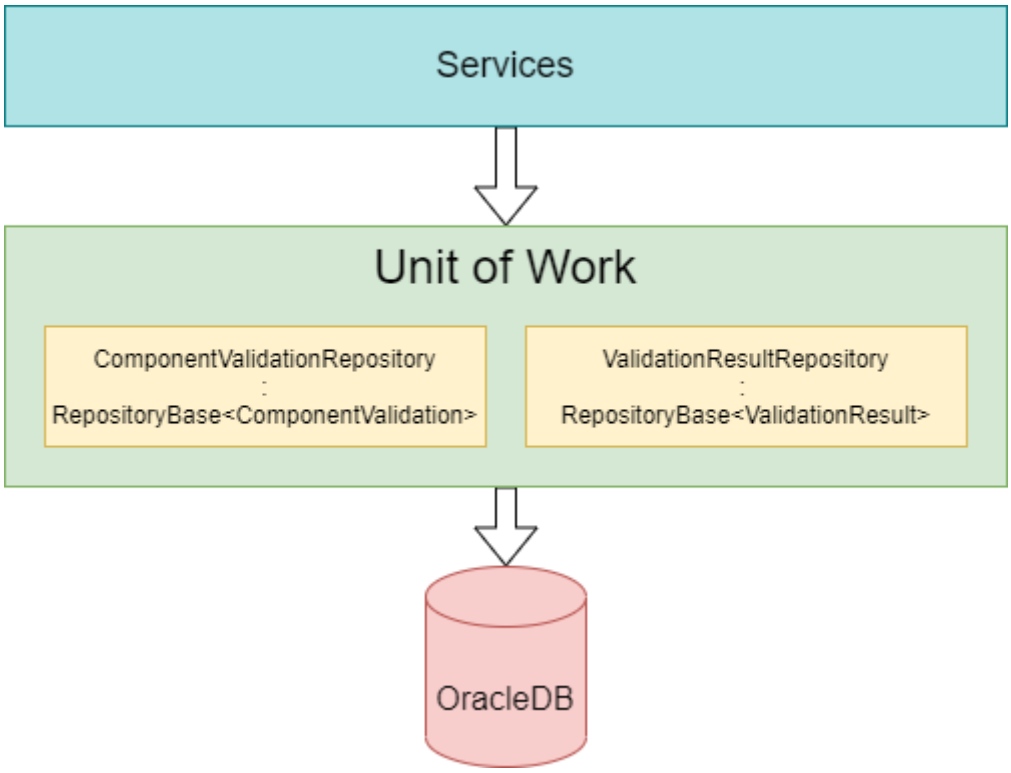


Figure 6. Implementation of Unit of Work and generic repositories

Entity Framework Core was used to implement DAL which meant that there was a need to create a database context class which is used to access different tables. Unit of Work is basically a wrapper class that passes the same instance of database context to different repositories therefore enabling business transactions.

5.2.3 Validation Service

Main component of the Service Layer is definitely validation service which is responsible for initiating the validation process through TdDataValidator class and persisting the validation results using UnitOfWork that is injected to the service via Dependency Injection. It is also capable of binding validation profile from application configuration which again is acquired with the help of DI.

5.2.4 Parallelism

The parallelism was achieved by using Task Parallel Library which is a set of public types and APIs in the System.Threading and System.Threading.Tasks namespaces. The purpose of the TPL is to make developers more productive by simplifying the process of adding parallelism and concurrency to applications. [14]

BatchManager, QueryExecutionManager and ResultEvaluator all derive from an abstract class ParallelWorker.which is capable of keeping track of all parallel tasks in the current validation run. It also measures duration of every task completion separately which is then accumulated and regarding this thesis that accumulated time is referred to as absolute time.

ParallelWorker has method AddWorker() which can be used to add new task to the worker pool. It also has method WaitAll() which can be used to wait all current tasks in the pool to finish. This comes handy when the validation queue becomes empty and the program wants to start reporting the results. There is a special case for BatchManager where all pre-exsiting tasks are waited before adding a new worker because EF Core is using a classed called DbContext, for communicating with the application’s database, which is not thread safe.

ParallelWorker class has property named OnTaskComplete which type is generic and needs to be passed in while a new instance is created. It is noted that underlying generic type must be a subtype of Delegate meaning that OnTaskCompleted is basically a

callback that is invoked when a parallel task completes e. g. ResultEvaluator is derived from ParallelWorker<Action<ValidationResult>>. In C# Action is a delegate that has a void return type meaning that in a previous example when evaluation task completes then it calls a method passing in a first parameter of type ValidationResult.

6 Experiment

Following are described several experiments that were conducted during the creation of this thesis. This chapter is the main input for quantitative analysis that is performed in the next chapter Analysis. Before delving into the experiments, author sets out some aspects about experimentation environment and results.

6.1 Common methodology

All the measurements, except for old solution, are based on validation profiles and measured in a controlled environment meaning that only one variable was changed at a time. In ideal scenario, for the most precise results, the number of iterations per experiment should be as high as possible but regarding this thesis there were usually five iterations, so to that extent there is definitely some amount of ambiguity in the results.

6.2 Obscure factors

As previously mentioned, for almost every experiment, there were thousand fixed validations so that query execution times per validation would be somewhat static, of course there are other factors contributing to that matter e. g. network speed and current workload on Teradata Database itself. Regarding network it is stated that the system under test (Teradata), client-side application and its database all reside in the same internal network. When it comes to Teradata's environment then the experiments were done during time periods where the overall load on database was approximately 3% compared to usual 20 - 100%.

6.3 Experiment A – Connection Pooling efficiency

While this experiment focuses on illustrating the importance of connection pooling the number of queries was reduced from 1000 to 10. Total time of query executions was measured (in milliseconds) and then average for every profile was calculated accordingly.

Profile	Connection Pooling	Run 1	Run 2	Run 3	Run 4	Run 5	Average
A1	No	5 062	4 157	4 455	4 470	4 647	4 558
A2	Yes	2 138	2 358	2 048	2 307	2 286	2 227

Table 1. Results of Experiment A

It can be seen that there is clearly a difference between executing queries with or without connection pooling.

6.4 Experiment Z – Old solution vs. TdValidator

As one of the goals of this thesis is to improve an already existing solution, therefore measurements between the two systems must be taken. For measuring the prototype, a validation profile was created that would try to mimic the currently used solution in the enterprise. The parameter values were:

- Connection Pooling: true
- Heap size: 1
- Batch size: 1

System	Run 1	Run 2	Run 3	Run 4	Run 5	Average
Existing	3 303 042	3 215 881	3 471 262	3 538 554	3 229 003	3 351 548
Prototype	1 370 000	1 368 538	1 545 145	1 482 557	1 421 741	1 437 596

Table 2. Results of Experiment Z

It is worth mentioning that a profile like this is theoretically not using any of the optimization functionality that was developed for the prototype, except execution and persistence is done in parallel so that could possibly affect results between the systems but overall these should not be very drastic. Measurements were done in milliseconds.

6.5 Experiment B – Heap size

In this experiment the goal was to measure duration of the validation process for different heap sizes. As the maximum number of requests for current workload was five and to avoid queuing requests on database side which in turn can result in connection timeouts then the heap size variable was measured for values 2, 3 and 5, but to investigate the effect of inappropriate heap size also a series of runs were done for a heap size of 16.

During the experiment the number of connection timeouts (60 seconds) and spool errors were measured. In addition to that also the average processing time of validation queries was measured. Note, that by processing time it is meant a time span between sending out the request from client and getting a response from the database, this includes the time a query spent in the waiting list. Results of the experiment are presented on the next page.

Profile	Profile Run No	Heap Size	Num of Timeouts	Spool Errors	Query Processing Avg. (ms)	Total Time (ms)	Formatted Total Time
B2	1	2	0	12	1 971	986 902	00:16:26.933
B2	2	2	0	22	2 030	1 016 202	00:16:56.229
B2	3	2	0	21	2 151	1 076 564	00:17:56.591
B2	4	2	0	14	1 996	999 666	00:16:39.688
B2	5	2	0	19	2 110	1 056 575	00:17:36.597
B3	1	3	0	21	2 787	932 496	00:15:32.513
B3	2	3	0	15	2 689	901 015	00:15:01.030
B3	3	3	0	15	2 689	909 603	00:15:09.615
B3	4	3	0	22	2 877	963 386	00:16:03.398
B3	5	3	0	15	2 805	939 752	00:15:39.764
B5	1	5	1	22	4 407	889 813	00:14:49.823
B5	2	5	2	22	4 326	872 909	00:14:32.920
B5	3	5	0	29	4 255	856 674	00:14:16.686
B5	4	5	5	29	4 666	940 424	00:15:40.434
B5	5	5	2	27	4 159	838 229	00:13:58.239
B16	1	16	7	96	7 381	493 076	00:08:13.083
B16	2	16	4	103	7 492	489 724	00:08:09.731
B16	3	16	11	91	7 255	502 255	00:08:22.262
B16	4	16	7	97	7 344	485 381	00:08:05.389
B16	5	16	8	82	7 783	506 295	00:08:26.328

Table 3. Results of Experiment B

Profile	Heap Size	Num of Timeouts	Spool Errors	Query Processing Avg. (ms)	Total Time (ms)	Formatted Total Time
B2	2	0	18	2 052	1 027 182	00:17:07.182
B3	3	0	18	2 769	929 250	00:15:29.250
B5	5	2	25	4362	879 609	00:14:39.609
B16	16	7	94	7 451	495 346	00:08:15.346

Table 4. Results of Experiment B aggregated per profile

The results clearly indicate benefits of parallel query execution. Results of Experiment B are analysed in more detail, later in this thesis.

6.6 Experiment C – Batch Size

This experiment tried to map the impact of batch size parameter on the validation process by measuring total time of the process and absolute time of data persistence. There were three different scenarios for testing batch size: 1, 10 and 100.

After conducting the experiment, the results were as follows:

Profile	Profile Run No	Batch Size	Absolute time of Data Persistence	Total Time	Data Persistence % of Total Time
C1	1	1	14 390	834 907	1.724
C1	2	1	11 428	812 301	1.407
C1	3	1	13 781	828 020	1,664
C1	4	1	13 583	826 902	1,643
C1	5	1	11 874	818 661	1,450
C2	1	10	4 262	808 596	0,527
C2	2	10	3 504	813 222	0,431
C2	3	10	3 711	885 468	0,419
C2	4	10	4 158	822 713	0,505
C2	5	10	4 337	814 054	0,533
C3	1	100	2 989	857 271	0,348
C3	2	100	3 353	822 335	0,407
C3	3	100	2 886	842 007	0,343
C3	4	100	3 421	824 407	0,415
C3	5	100	2 997	854 127	0,351

Table 5. Results of Experiment C

Profile	Batch Size	Absolute time of Data Persistence	Total Time	Data Persistence % of Total Time
C1	1	13 011	824 158	1,579
C2	10	3 994	828 814	0,482
C3	100	3 129	840 029	0,372

Table 6. Results of Experiment C aggregated per profile

7 Analysis

This chapter is dedicated for synthesizing a qualitative and quantitative analysis of optimization aspects proposed and experiment results gathered during this thesis.

7.1 Connection Pooling

Based on the results of Experiment A it can be stated that Connection Pooling is a heavy contributor to the overall performance. The runs with Connection Pooling were on average two times faster than the ones without the pooling. As the average query execution time decreases then the cruciality of Connection Pooling increases and vice versa.

7.2 Parallelism

The results of Experiment B indeed indicate the benefit of running validations in parallel, yet the outcomes are somewhat controversial. After examining the results more thoroughly there was a need to determine the cause of certain phenomena. Following proportions and resolutions are tightly coupled with the specific validation set used during the experimentation phase hence there can be some fluctuation between different sets.

7.2.1 Spool errors

First there seem to be constant spool errors for certain validations even when executing them sequentially (with heap size 1). After verifying, that it really is like that, it can be said that regarding spool space profiles B2 and B3 in essence are not that problematic but there is still a minor failure rate of 0.9%. When it comes to profile B5 and B16 then it's obvious to see a raise in spool errors which proportions are 1.9% and 8.5% accordingly.

Possible solutions would be to increase database user's spool space, refine queries or fix possible skew problems, but all of these are administrative tasks. From the perspective of software engineering, a solution for limiting spool space errors would be to enqueue all spool space related (failed) validations to a side queue which validations will be re-

executed sequentially when the main queue finishes as at one point those validations need to be re-executed anyways so why not to do it right away. Of course, that sort of functionality can be made configurable.

7.2.2 Occupying queries

When inspecting the results of Experiment B, there appears to be an increase in connection timeouts when the heap size is nearing the maximum (B5) or is totally over it (B16). There is also a positive correlation between average query processing time and heap size. Furthermore, if to look at the relationship between the increase of total time and increment of heap size then it's not linear but rather logarithmic.

One factor contributing to that matter is the fact that as the heap size is nearing its maximum allowed value the likelihood of requests being queued on the database side increases. Average query processing time expresses that pretty clearly. Secondly, after analysing the validations it was found that approximately 20 – 25% (depending on heap size) of queries took longer to process than the average query processing time where the vast majority of them took over 15 seconds. This creates scenarios where queries that would run quickly (below average processing time) must wait behind longer running queries which are occupying current heap.

7.3 Data Persistence frequency

When it comes to batch size variable then it can be deduced that there is indeed a legitimacy between the growth of batch size and validation performance. As the results present also a ratio between time spent on data persistence and validation process overall then the merit becomes questionable, namely the average proportion of time spent on persistence compared to the whole process ranges between 0,372 - 1,579% which is rather marginal. So, to sum it up it can be said that from the profiles presented in Experiment C the profile C2 is the victor as its time contribution is acceptable and batch size compact.

7.4 Old solution vs. TdValidator

First when comparing just the results between old system versus TdValidator mimicking configuration of the former one, already then there is a huge performance difference roughly about 2.3 times. Author finds two possible reasons for that being:

- Platform upgrade from .NET Framework 4.5.2 to .NET 5
- In new system there is zero interference between data persistence and validation execution compared to the old system which shared same connection between execution and persistence

Secondly, after enabling the developed parallelism functionality the difference grows even more, to about 3.6 times when using profile B3 which proved to be reliable enough. The parallelism clearly improves the performance, yet the author intended to get even better results, but the outcome is at least satisfactory. In addition to performance increase the TdValidator provides a more flexible system that can be adjusted by the user through validation profiles.

8 Summary

In conclusion it can be said that the expectations raised in the beginning of thesis were reached and the optimization aspects found confirmation. Although the experimentation environment was not ideal then the results still indicate the expedience of the proposed solution.

This research will definitely add value to the underlying enterprise which current system was the main driver to write this thesis. In addition to that this paper is a useful source of information for anyone who is interested in data validation optimization (or query execution in general) in Teradata Database, both from administrative and software engineering perspective but it’s worth mentioning the empirical part of thesis focuses solely on developed client-side application.

References

[1] Altexsoft, “Enterprise Data Warehouse: Concepts, Architecture, and Components”, 24. Oct. 2019, Accessed on: 21.03.2021 [Online]. Available: https://www.altexsoft.com/blog/enterprise-data-warehouse-concepts/

[2] Teradata Vantage™ - Database Introduction (March 2019, version 16.20), Teradata Corporation, San Diego, CA, USA. Accessed on: 27.03.2021 [Online]. Available: https://docs.teradata.com/r/WJbR7YRqjgLIL9fZwxIBOg/root

[3] Teradata Vantage™ - Database Design (March 2019, version 16.20), Teradata Corporation, San Diego, CA, USA. Accessed on: 27.03.2021 [Online]. Available: https://docs.teradata.com/r/2W4uUc~MxQ6lsG5v30LY5w/root

[4] Teradata Vantage™ - SQL Functions, Expressions, and Predicates (March 2019, version 16.20), Teradata Corporation, San Diego, CA, USA. Accessed on: 27.03.2021 [Online]. Availablehttps://docs.teradata.com/r/756LNiPSFdY~4JcCCcR5Cw/root

[5] Teradata Vantage™ SQL Fundamentals (March 2019, version 16.20), Teradata Corporation, San Diego, CA, USA. Accessed on: 28.03.2021 [Online]. Available: https://docs.teradata.com/r/aFcrqJBUrrMhnpBHTtr71g/root

[6] .NET Data Provider for Teradata (2019), Teradata Corporation, San Diego, CA, USA. Accessed on: 04.04.2021 [Online]. Available: https://teradata-docs.s3.amazonaws.com/doc/connectivity/tdnetdp/16.20/help/webframe.html

[7] ADO.NET Documentation, Microsoft Corporation, Redmond, Washington, USA. Accessed on: 04.04.2021 [Online]. Available: https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/

[8] Methodology for data validation 1.0 (June, 2016), Marco Di Zio, Nadežda Fursova, Tjalling Gelsema, Sarah Gießing, Ugo Guarnera, Jūratė Petrauskienė, Lucas Quenselvon Kalben, Mauro Scanu, K.O. ten Bosch, Mark van der Loo, Katrin Walsdorfer ESSnet (European Statistical System), Eurostat Accessed on: 22.04.2021 [Online]. Available: https://ec.europa.eu/eurostat/cros/system/files/methodology_for_data_validation_v1.0_rev-2016-06_final.pdf

[9] Teradata Vantage™ Database Administration (March 2019, version 16.20), Teradata Corporation, San Diego, CA, USA. Accessed on: 22.04.2021 [Online]. Available: https://docs.teradata.com/r/ueCMQAxljdET5klb6rbF1g/root

[10] Common web application architectures (January 2020), Microsoft Corporation, Redmond, Washington, USA. Accessed on: 24.04.2021 [Online]. Available: https://docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures

[11] .NET Generic Host in ASP.NET Core (April 2020), Microsoft Corporation, Redmond, Washington, USA. Accessed on: 08.05.2021 [Online]. Available: https://docs.microsoft.com/en-us/aspnet/core/fundamentals/host/generic-host

[12] Implement the infrastructure persistence layer with Entity Framework Core (January 2021), Microsoft Corporation, Redmond, Washington, USA. Accessed on: 08.05.2021 [Online]. Available: https://docs.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-implementation-entity-framework-core

[13] Unit of Work, Martin Fowler, Accessed on: 09.05 [Online] Available: https://martinfowler.com/eaaCatalog/unitOfWork.html

[14] Task Parallel Library (March 2017), Microsoft Corporation, Redmond, Washington, USA. Accessed on: 11.05.2021 [Online]. Available: https://docs.microsoft.com/en-us/dotnet/standard/parallel-programming/task-parallel-library-tpl

[15] Teradata Vantage™ Workload Management User Guide (March 2019, version 16.20), Teradata Corporation, San Diego, CA, USA. Accessed on: 11.05.2021 [Online]. Available: https://docs.teradata.com/r/MdJpFyYdQYA50wtFdSdNPA/root

Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis[1]

I Gregor Tammert

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis “Data Validation Optimization in Teradata-based Data Warehouse”, supervised by Nadežda Furs
   1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
   1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

13.05.2021

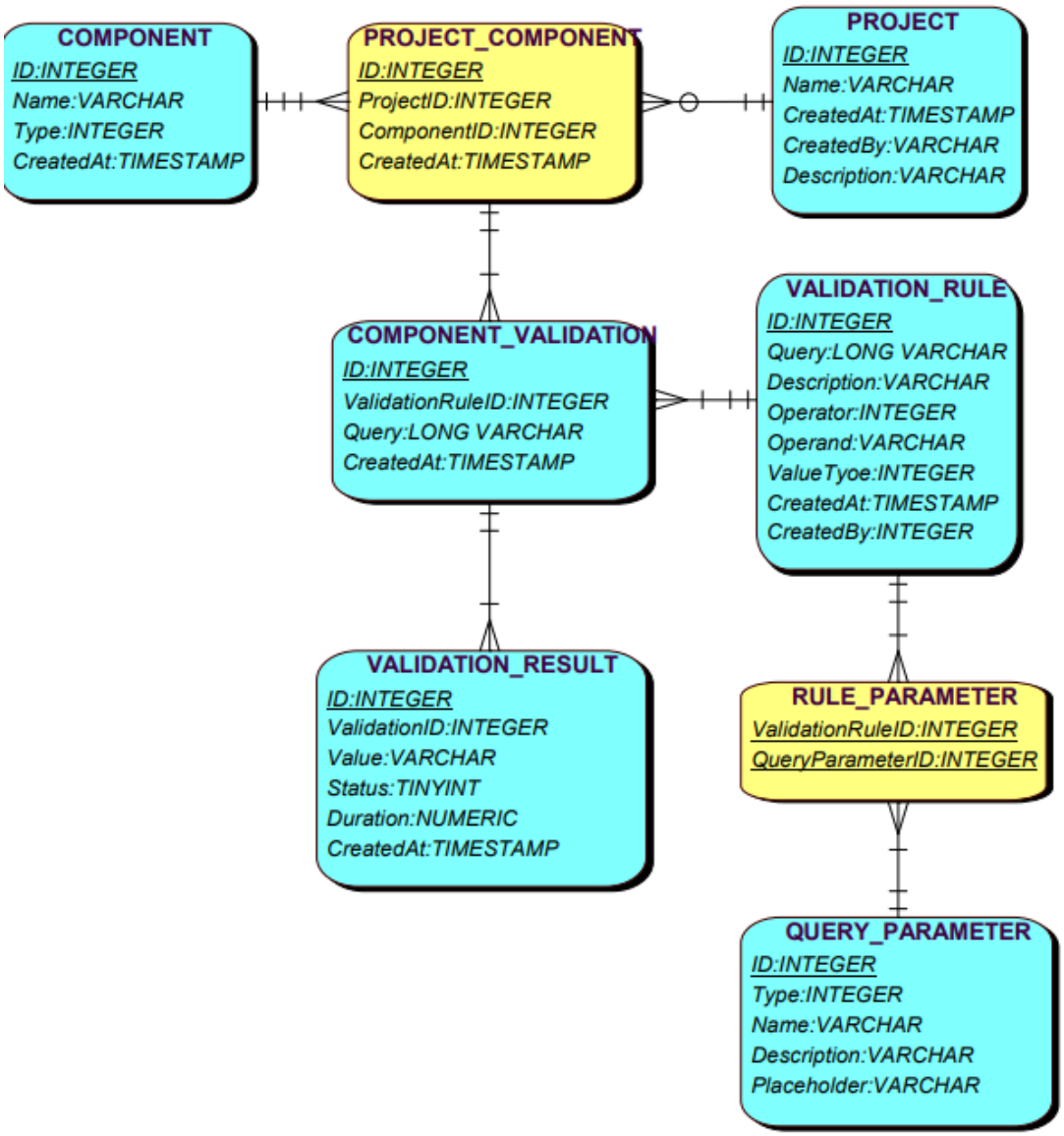
1 The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.

Appendix 2 – Query for finding skewed tables

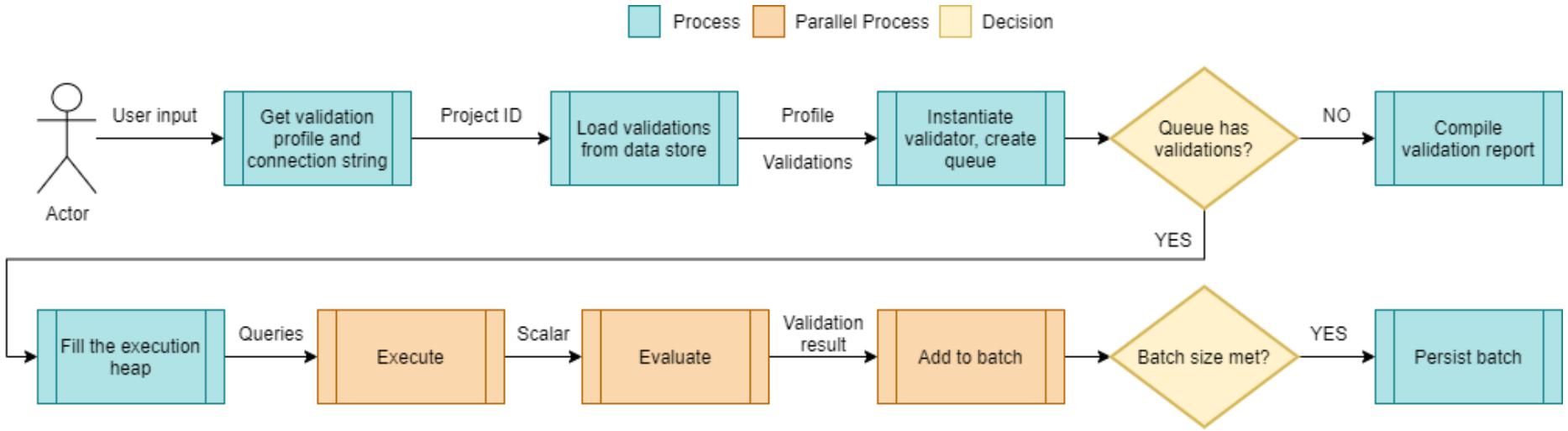
```
-- Based on: https://docs.teradata.com/r/B7Lgdw6r3719WUyiCSJcgw/Y8b2MSxk_FjfSX~O1QCRmQ
SELECT
  TableName (FORMAT 'X(20)'),
  MIN(CurrentPerm) AS "AMP Minimum",
  AVG(CurrentPerm) AS "AMP Average",
  MAX(CurrentPerm) AS "AMP Maximum",
  (("AMP Average" / "AMP Minimum") + ("AMP Maximum" / "AMP Average")) / 2 AS "Custom Skew Factor"
FROM DBC.TableSizeV
WHERE DatabaseName = 'DatabaseName' -- Replace with acutal database name
GROUP BY TableName
ORDER BY "Custom Skew Factor" DESC;
```

Appendix 3 – Entity relationship diagram of prototype

Entity Relationship Diagram



Appendix 4 – Validation process flow



Appendix 5 – Core module classes