Bachelor's project

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

Marte Jølsett, Christian Stene and Petter Östergren

Monitoring and reporting of Key Performance Indicators in an Emergency telephone system

Bachelor's project in IT-Operations and Information Security
Supervisor: Christian Johansen

May 2021

NTNU
Kunnskap for en bedre verden

Marte Jølsett, Christian Stene and Petter Östergren

Monitoring and reporting of Key Performance Indicators in an Emergency telephone system

Bachelor's project in IT-Operations and Information Security
Supervisor: Christian Johansen
May 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

NTNU
Norwegian University of
Science and Technology

Acronyms

1NF First normal form. 18, 29, 51

2NF Second normal form. 18

3GPP 3rd Generation Partnership Project. 16

3NF Third normal form. 18

B2BUA Back-to-Back User Agents. 12–14

CAM Call Accounting Manager. 16

CDR Call Detail Record. 2–4, 6, 8, 9, 16, 17, 28, 29, 35–37, 39–42, 48, 51, 53, 54, 99–103

CET Central European Time. 42

CSV Comma-separated values. 16, 26, 27

DST Daylight Saving Time. 42

EMS Emergency Medical Services. 16

ETSI European Telecommunications Standards Institute. 16

FTP File Transfer Protocol. 23

GDPR General Data Protection Regulation. 26, 30

GSMA GSM Association. 16

GUI Graphical user interface. 5, 19, 20, 22, 45

HDO Helsetjenestens Driftsorganisasjon for Nødnett HF. 1–9, 11, 14, 16, 17, 19, 27–31, 35, 40, 41, 45, 46, 48–53

ICCS Integrated Command Control System. 1

IETF Internet Engineering Task Force. 12, 13, 16, 23, 46

ISUP ISDN User Part. 13, 46

IT Information technology. 10, 12

ITU International Telecommunication Union. 16, 46

KAK Kommunikasjon i akuttmedisinsk tjeneste. 1, 2

KPI Key Performance Indicator. 2–5, 16, 26, 28, 45–49, 52, 54

MS-SQL Microsoft SQL Server. 2–4, 6, 9, 17, 19, 21, 22, 24, 27–29, 34, 35, 41, 42, 47, 48, 51, 52

NENA National Emergency Number Association. 16

NER Network Efficiency Ratio. 46–49, 52

NNI Network-Network-Interface. 15

NTNU Norwegian University of Science and Technology. 4, 5, 7, 19

PHP Hypertext Preprocessor. 26

RDBMS Relational Database Management System. 17

RTP Real-Time Transport Protocol. 12, 13

SBC Session Border Controller. 2–4, 8–17, 28, 29, 35, 46, 48, 51, 52, 54

SDP Session Description Protocol. 12, 13

SDR Session Defects Ratio. 46–49, 52

SEER Session Establishment Effectiveness Ratio. 46–48, 52

SER Session Establishment Ratio. 46–48, 52

SFTP SSH File Transfer Protocol. 9, 23, 24, 27–29, 35

SIP Session Initiation Protocol. 2–4, 9, 12–15, 46, 100, 101, 103

SQL Structured Query Language. 4, 6, 18, 19, 22, 24, 33, 35, 40, 48, 52, 54

SSAS Microsoft SQL Server Analysis Services. 19

SSDT Microsoft SQL Server Data Tools. 19

SSH Secure shell. 23

SSIS Microsoft SQL Server Integration Services. 3, 4, 6, 19, 20, 26–29, 35, 36, 40, 41, 51, 52, 54

SSMS Microsoft SQL Server Management Studio. 19, 51

SSRS Microsoft SQL Server Reporting Services. 2–4, 6, 19, 21–23, 27, 28, 30, 41, 52, 54

T-SQL Transact Structured Query language. 9, 18, 19, 22, 42, 43, 54

UNI User-Network Interface. 15

UTC Universal Time Coordinated. 42, 48

Visual Studio Microsoft Visual Studio. 19–21

VoIP Voice over Internet Protocol. 9, 13

VPN Virtual private network. 7

Glossary

Flat-file a Flat-file is an unstructured file with one record per line, that is written in plaintext. 16, 19–21, 26, 27, 39

Grafana Grafana is an Open-Source tool for data analytic and monitoring, utilizing metric visualization using a variation of charts. 2–6, 9, 24, 25, 28, 30, 46–48, 52, 54

Open-Source Software Software that has a public source code that is free and open for people to use, study and in most cases also develop the source code. 27

Parsing Dissecting a string or text by dividing the data by a given delimiter in the objects to make it fit in a data structure. 4, 6, 8–10, 17, 19, 26, 27, 51, 53

Zero-trust Architecture A network architecture that assumes every user is a threat. So users only gets the privileges or access they need to have. 3

Chapter 1

Introduction

1.1 Background

Helsetjenestens Driftsorganisasjon for Nødnett HF (HDO) is an operations center for the health service's use of emergency call services in Norway. HDO owns, operates, and manages today's primary communication solutions for telephone and radio communication with associated equipment for all the country's emergency medical communication centers, emergency centers, and emergency rooms [1]. HDO contributes to safe and secure communication to the emergency medical chain. Their main services include maintenance of the emergency numbers 113 and 116117 and the operation and management of control rooms and radio terminals. HDO's services are focused on being efficient and user-friendly for all users of the emergency network in the emergency medicine chain in all the regional health authorities, in all the country's municipalities, and for other relevant partners. HDO is jointly owned by the four regional health establishments which are under the Ministry of Health and Care Services.

As of May 2021, HDO have started a project named Kommunikasjon i akuttmedisinsk tjeneste (KAK). This project involves the acquisition of a new communication solution that is going to be used between the emergency network and emergency medical communication centers, emergency centers, and emergency rooms. KAK will result in a service platform where the new communication solution is ran . The KAK project is underway due to many important reasons. Firstly, today's communication solution which is being used for telephone- and radio communication is based on hardware technology from the '90s. This technology is called Integrated Command Control System (ICCS), and HDO states that it is considered outdated, the operation of it is costly, there is limited competence on it and it does not scale very well for future technologies and user needs. The ICCS technology is also not adapted to the recommended architectural principles of the Directorate for e-health [2]. In addition to the current communication solution being outdated, The Directorate for Civil Protection and Emergency Planning is preparing the next generation of emergency networks in Norway [3]. The new communication solution that KAK provides will be able to take advantage of the

opportunities from the next generation of emergency networks [Appendix A]. This will result in the services HDO deliver to be more robust, user-friendly, flexible, and 5G compatible [4]. When it comes to the new communication solution, HDO and Telenor are working together to develop and configure a new Session Initiation Protocol (SIP) based telephone system within a Session Border Controller (SBC) setting. The infrastructure itself will act as the link between Telenor and the emergency medical chain. The purpose of this solution is to receive all telephone calls from the public telecommunication network (113, 116117) or internally from the health regions to ensure that telephone calls end up in the correct emergency medical communication center or emergency room [Appendix A].

1.2 Research questions

1.2.1 Problem area

Based on the new communication solution KAK provides, all telephone calls regarding 113, 116117, and the internal health region will end up in the correct emergency medical communication center or emergency room. For each incoming call into the network, Call Detail Record (CDR) data is generated. This data contains all information about the call, for instance, who called, the location of the caller and the quality of the call to name a few. The CDR data is generated by the SBC when a session is established, and the CDR data itself is sent to HDO and stored in their infrastructure within the SBC. In this case, the CDR data is a valuable asset that lays the foundation for our project. HDO's intention is to use the CDR data for monitoring and troubleshooting their telephone network, which leads to them being able to recognize deviations in the network earlier on. For that reason, relevant CDR data will be stored in a Microsoft SQL Server (MS-SQL) database where both Microsoft SQL Server Reporting Services (SSRS) and Grafana are going to fetch data from it. SSRS will be used to generate reports and Key Performance Indicator (KPI)'s will be displayed in Grafana. With the help of concrete reports and a dashboard visualising KPI's, staff at HDO will be able to monitor and troubleshoot their network more efficiently.

In consequence, the project group has been given the following tasks:

- A solution must be developed to automatically load relevant CDR data, from the files received from the SBC's, and into an MS-SQL database (process to occur in given time intervals).
- A sensible database structure must be built to handle relevant CDR data.
- Create report templates for the following scenarios. All report must be able to perform filtering of relevant information such as period, numbers, disconnection code, trunk, and route:
    - Incoming calls with all of its associated information.
    - Calls stopped with disconnection reasons.
    - Attempted calls with associated error codes.

    - The total of incoming calls for a given period on given trunks.
- Grafana should show measurements based on various KPI's and possibly other relevant CDR data

1.2.2 Delimitation

The actuality of a resilient and working management solution for Norway's emergency services is cardinal for a system handling life-threatening situations. Within such a system security needs to be dispensed between hardening each subsystem or shape a strengthened barrier safeguarding the entire infrastructure. HDO works within a Zero-trust Architecture, ensuring authentication and authorization several layers before the zone our implementation will serve. Furthermore, HDO has several different operations utilizing the MS-SQL database entries reliant on plaintext data. Therefore database entries are restricted to be processed as plaintext without any form of hashing procedures.

1.2.3 Purpose

As mentioned earlier, HDO is currently developing a new communication solution based on SIP and SBC in collaboration with Telenor. The purpose of that solution is to receive all telephone calls from the public telecommunication network (113, 116117) or internally from the health regions to ensure that telephone calls end up in the correct emergency medical communication center or emergency room. A part of the new communication solution will address troubleshooting and monitoring. This is the solution that this project group is going to develop, which will help HDO employees recognize deviations in the network earlier on. HDO employees will use our end product to troubleshoot- and monitor the network, which implies analysis of network trends and network efficiency.

1.2.4 Motivation

This project can have great impact in the society since its objective is to develop an end product that is going to be used in the emergency telephony network of the health sector. Developing software products for such critical environments is challenging due to requirements of availability and responsiveness, but also motivating, rewarding, and instructive due to the applicability to handling human health emergency situations. The core of this project is based on several technologies, e.g. Microsoft SQL Server for storing and aggregating data, Microsoft SQL Server Integration Services (SSIS) for data migration, SSRS for reporting, and Grafana for visualizing data and KPI's. These technologies are heavily used in enterprise and commercial systems, which to some extent proves their power and reliability, e.g., Grafana has become one of the most used open-source software for visualizing data trends. Besides the project being useful to HDO, our results can be used in other sectors as long as businesses within that sector are using or are planning to establish a telephony network based on SIP and SBC.

1.3 Target audience

The primary target audience for this report is HDO being the system owners, and the end product that the project group develops will be used by HDO. However, this report could also be relevant for other businesses that want to gather knowledge about CDR parsing in SIP-/SBC networks using tools by Microsoft and Grafana for gathering information used for calculating KPI's and call statistics.

1.4 Academic background

All members of the project group follow the same course of study, which is IT operations and information security at Norwegian University of Science and Technology (NTNU) Gjøvik. All members have basic knowledge about software development, networking, infrastructure, virtual machines, Structured Query Language (SQL), security, and scripting. Although this is important underlying knowledge for the task, it still requires the project group to acquire knowledge about more specific technologies to solve the task. These technologies include SIP-/SBC networks, Grafana and Microsoft technologies such as MS-SQL, SSRS and SSIS.

1.5 Roles

Marte Jølsett

Marte is the group leader, she is responsible for handling meeting rooms, that the project group follows the schedule, and making sure that deadlines are met.

Petter Östergren

Petter is the meeting leader, he is the single point of contact to HDO. He is responsible for e-mail communication towards the client and the supervisor, he is also responsible for writing meeting reports.

Christian Stene

Christian is the document responsible. He is responsible for taking backups, managing files, and make sure that documents are stored securely.

Tasks within the group have been distributed equally, everyone having varying work tasks that include development, testing, researching documentation, troubleshooting, and report writing.

Client: HDO w/Stig Atle Haugen

Stig Atle Haugen [5] is a operations engineer at HDO, he is the contact person for this project.

Supervisor: Christian Johansen

Christian Johansen [6] is an associate professor at NTNU in the Department of Information Security and Communication Technology.

1.6 Requirements

1.6.1 Functional requirements

The end product which the project group develops has the following functional requirements represented in use cases:

Case	View Grafana dashboard to monitor KPI trends in the HDO network.
Participant	HDO operations center
Objective	Recognize deviations in the network earlier on.
Description	Staff at HDO's operation center will have a Grafana dashboard available to monitor several KPI's. HDO will be able to analyze KPI's for a specific time period which enables them to analyze trends within their network. This allows HDO to obtain information early about how their network is performing and potentially find deviations.

Table 1.1: Use case 1

Case	Show a report for troubleshooting attempted calls in the HDO network.
Participant	HDO helpdesk
Objective	Recognize deviations in the network earlier on.
Description	The participant opens the report GUI and deploys the report. After the report has been opened, the participant can troubleshoot by changing the parameters to find when the problem started and where it began.
Alternative	Show a report for troubleshooting stopped calls, show a report for the number of calls for a given time period, or show a report for all incoming calls are the other report alternatives. All four reports can be filtered on different parameters, which means that HDO can display specific data if desired (see section 4.4 for more report details).

Table 1.2: Use case 2

Case	Change parameters in SSRS to show a summary of trunks for a date.
Participant	HDO helpdesk
Objective	Filter out the unnecessary dates in the report.
Description	After a user has deployed a report, the report shows the last 7 days for each used trunk in that period. The user then changes the *from* and *to* date to the dates the user wants to view. This results in the report only showing the rows for the chosen dates.
Alternative	The user can also filter data based on trunk, route, number, time, disconnect reason, and period by altering the parameters, depending on which report is shown.

Table 1.3: Use case 3

1.6.2 Portability and compatibility

As the product is developed in a test environment, it is a requirement that the solution is as portable as possible. This implies that the software used to develop the product is compatible with HDO's infrastructure, where also the version of the software is similar. For instance, using Microsoft SQL Server 2019 instead of other versions and using Microsoft SQL Server Integration Services for CDR parsing instead of other scripting languages. In addition, it will be necessary to document how the components within the product are configured. For instance, document SQL commands used for the database, SSRS reports, and the Grafana configuration to name a few. This means that when the product is fully developed, HDO can move it into their production environment with minimal changes to the software configuration.

1.6.3 Constraints

Technical constraints

- For development and testing, the work-/development environment that HDO has established for this project shall be used.
- Technologies such as MS-SQL, SSRS and Grafana must be used instead of similar software.
- The project group was not in charge of configuring any hardware infrastructure or advanced network configurations, resulting in a strict environment that at times required consultation with HDO before processing with a given technology.

Time constraints

- The group members undertakes to comply with NTNU's deadlines regarding the preliminary project, the final report and the presentation of the final report.

Legal constraints

- The group members undertake to comply with the declaration of confidentiality, VPN (Virtual private network) Access User Agreement (External users) and terms of use issued by HDO.
- The group must follow the laws and regulations that comply with the data used in the report. More about this can be read about in section 3.11.

1.7 Outline

This report begins with a description of the methods used for acquiring data for the result and analysis, followed by a background segment explaining how different protocols are exercised within the system and software correlations. Chapter 4 describes our technical solution to the problem, detailing out database schematics, implementation snippets, and report samples, among others. Finally, we discuss the results and conclusions we gathered that endure from the analysis, with suggestions for further studies. At the end of the report, the references used for the study are listed, followed by a series of appendixes; contracts, configuration manual, interview notes, time logs, and code. The appendix shows detailed documentation on how to set up the developed pipeline, allowing the project owner or other interested parties to recreate the system within a production environment.

Chapter 2

Research Methodology

For the objective of this research, two methods have been used. First, to follow rapidly changing implementation designs and the most contemporary best practices, articles, vendor documentation, and videos have been studied online. Furthermore, to ensure that this study was carried out according to HDO's workflows and wishes, informal consultation with the HDO representative was persistent during the study duration.

2.1 Research Approach

The research approach of this study has been concentrated on qualitative research [7]. Primarily secondary data sources [7] have been utilized to examine standard implementation solutions and development schematics. Due to the continuous update routines technologies rely on today [8], it was necessary to use the Internet as a source for vendor documentation, articles, and video.

Additionally, to collect supplementary information and personal insights on how HDO desired the system to interlace with the current solutions, an unstructured interview was ongoing with Stig Atle, IT Operations Engineer at HDO. The unstructured interview gave the possibility to have ongoing conversations either in person or via online communication services with a low threshold. Stig Atle also provided comprehensive guidance with reading materials and well-known implementation solutions.

2.2 Information Acquisition

2.2.1 Vendor documentation

Vendor documentation has been used when installing and configuring merchant-specific technologies. For example, Ribbon documentation (vendor of the SBC hardware) has been extensively used when parsing CDR files to ensure correct mapping of columns and mapping of error codes. When utilizing this documentation, it is cardinal to use the documentation matching the software versions we

have been using, Microsoft SQL Server 2019, Grafana v7.5.x, and Ribbon version 9.X; this was particularly important for the Ribbon documentation due to severe alterations between versions.

2.2.2 Collecting information through online articles and videos

A support list of websites known for quality and genuine information was the starting point when collecting online articles and videos. These were as following:

SQLAuthority with Pinal Dave (`https://blog.sqlauthority.com/`)

LearnSQL.com (`https://learnsql.com/`)

Helsetjenestens driftsorganisasjon for nødnett HF (`https://www.hdo.no/`)

Direktoratet for e-helse (`https://ehelse.no/`)

Direktoratet for samfunnssikkerhet og beredskap (`https://www.dsb.no/`)

This list was to ensure the sources we handled were reliable. From these domains, we further studied underlying references if mentioned within an article. Additionally, there exists a wide variety of video courses covering database design, VoIP networks, and Microsoft technologies. Such courses were utilized to refresh our understanding of a technology or overview new technologies that could be useful on *Pluralsight (`https://www.pluralsight.com/`)* and *Youtube (`https://www.youtube.com/`)*.

To research common technics in database design and SBC networks, it was necessary to expand data collection to utilize online search engines. The following search terms were used, but not limited to:

Voice over Internet Protocol, Big data storage, Session Border Controller, Session Initiation Protocol, Call Detail Record, SSH File Transfer Protocol, parsing, Transact Structured Query language

If a search regarded implementation solutions *Stackoverflow (`https://stackoverflow.com/`)* and *DBA Stack Exchange (`https://dba.stackexchange.com/`)* were natural locations to look for an explanation. Developers commonly use these sites to post and answer questions regarding development; with the site's built-in voting system, the highest proposed solutions are easy to locate and analyze.

Websites addresses used for this research were systematically written down with date of access.

2.2.3 Interview with IT Operations Engineer at HDO

The interview in this study was carried out in an unstructured way via in-person meet-ups, messages, and online video meetings: the interviewee, Stig Atle, IT Operations engineer at HDO. Stig Atle began the project with a set of presentations explaining the requirements concerning HDO's needs. These talks were then

followed by a series of presentations regarding the functions of an SBC based emergency network.

At the beginning of the project, an online discussion room was created. We could reach out via messages or video calls to have Stig Atle elaborate on questions we felt needed deeper insight. The questions were never sent in advance, and he did not have to answer anything on the spot. Instead, he could discuss the questions with colleges or validate the questions involving policies and documentation before forming an answer.

On feature completion, a meeting was requested to validate that our solution fulfilled the terms discussed at the beginning of the project.

2.3 Data Analysis

The data collected from both of the two methods are equally crucial for satisfying all research questions. The parsing techniques, database architectures, and querying patterns discovered through online articles, video courses, and the interviews, are collected in a structured way. The structural approach is essential in order to be able to conclude which solutions occur regularly within the industry, relying on well-tested solutions to minimize the error rate due to the criticality of the system

2.4 Validity and Reliability

2.4.1 Vendor documentation

Vendor documentation is at large an excellent resource for new and updated documentation for a given product.

Serious vendors typically want to document what makes them unique and show how their products could be utilized in a well-structured manner. However, vendors might also recommend additional products to solve a given problem instead of a feasible programmatic approach.

2.4.2 Online articles and videos

There are numerous concerns and positive sides about the validity and authenticity concerning data obtained from the Internet. As a positive aspect, the information obtained online is often new and updated. In an industry that is continuously changing and developing, this is an essential aspect for researching a topic within the area of IT.

The dilemma with validity and reliability when it comes to online sources is that anyone can undoubtedly post information online, and there are no guarantees that the information is correct. Because of this, it is important to use websites that are known to be reliable, such as websites hosted by companies and representable people within the field.

Another problem is that there is no guarantee that this study has managed to find all information existing on the topic due to the massive amount of information available on the Internet.

2.4.3 Interviews

The purpose of interviewing a working professional in the industry is twofold for this research. First of it is to gain a deeper insight into what a professional operation engineer regards as critical indicators that will require additional monitoring within an SBC network. Secondly, HDO is the employer of this report; hence complying with their current practices is essential for us to ensure that the results found during this study ensures genuine value.

When conducting unstructured interviews, the data collected might be subject to personal opinions and objectives (understandable during this circumstance, since the interviewee is the product owner).

Chapter 3

Technical Background

This chapter explains the technologies, laws, and regulations, and existing solutions utilized to work the problem. The technologies written about are fundamental for understanding how we have applied the technologies in various aspects. Aforementioned this research is targeted towards an audience expected to have basic IT knowledge; hence the texts here will not describe every detail on how a technology works, instead point on significant aspects from a given technology that is relevant for our purposes. Specific technologies will be described in more detail: the SBC and SIP network infrastructure being one.

Lastly, the laws and regulations written about are significant because they have impacted our work regarding data and privacy. Therefore, the existing solutions we have found and used to solve the tasks or problems encountered are written in this chapter.

3.1 Session Initiation Protocol

SIP is a signaling protocol for establishing, modifying, and terminating sessions in an IP network; the protocol was specified by Internet Engineering Task Force (IETF) in the standard RFC 3261 [9]. The session in SIP is a call between two endpoints; SIP can send and receive multimedia content over the Internet; an example of an endpoint is a smartphone [10]. The sessions are described in Session Description Protocol (SDP), for delivering and voice and video over an IP network, an Real-Time Transport Protocol (RTP) is utilized [10].

The architecture of SIP is client/server-based, which means that the tasks are distributed between a server host that provides resources or services to one or more clients that ask for the resources or services, where the client normally does not share anything back [11]. The caller's phone acts as the client, while the callee's phone acts as a server. After the session is established, the caller will be redirected through a server before reaching the callee's phone. SIP uses five different servers: Proxy Server, Registrar Server, Redirect Server, Location Server, and Back-to-Back User Agents (B2BUA).

The Proxy Server works as both client and server, and it takes a request from a user agent and forwards the message from the request, like a router. The Server can be either stateless or stateful: a stateless proxy only forwards the message without any storing of the information of the call or transaction, while a stateful proxy forwards the message and can retransmit the request if there is no response, and it is holding a record of the request and response received if needed in the future [12].
The Registrar Server helps with authentications of users within the network, and receives registration requests from user agents, then accepts them [12].
The Redirect Server receives the requests to look up and redirect to the wanted address based on the address's registries in the Registrar [12].
The Location Server sends information about the caller's possible locations to the redirect server and proxy server [12].

3.1.1 Back-to-back user agent

A B2BUA is a logical network element in SIP that receives a SIP request and reformulates it, then sends the request as a new request [13]. According to Tutorialspoint "A B2BUA agent operates between two endpoints of a phone call and divides the communication channel into two call legs." [13]. A B2BUA participates in all the SIP signaling between both endpoints it has established; the B2BUA operates between both endpoints of a session or call and divides the communication channel into two legs [13]. The B2BUA has different functions, like; hiding the network topology, private addresses, call transfer, and automatic call disconnect.

3.1.2 Standards

SIP is defined in the standard RFC 3261 [9] from IETF. However, SIP contains numerous other protocols like SDP and RTP, where each of them has their own standards. As a result, it is usual for companies who use SIP to modify standards for their use [Appendix A]. The SIPs response codes that can be generated by a client request, codes can be translated to ISDN User Part (ISUP) codes by using a standard code mapping: [14].

3.2 Session Border Controller

SBC is a device that regulates and protects the IP communication flow; the IP communication flow can come from, for example, VoIP and IP video [15]. SBC contains the three terms; session, borders, and controller.

A session is a connection and communication between two devices or parties [16]. When a call is connected, each call has at least one signaling message exchange and at least one media stream. The signaling message controls the call; meanwhile, the media streams carry the audio, video, and other data, including call statistics and quality. These streams make the session [16].

The borders refers to the network borders. An example is that the SBC is managing the flow of data across the borders [16].
The controller refers to the control the SBC has on the sessions'; the SBC can control and modify the sessions that try to cross the network border, both in and out; an example is access control [16].

3.2.1 Use of an SBC

The SBC can do many different tasks. Nonetheless, the two most important uses for HDOs use of SBCs are to hide the network topology and secure the network from incoming traffic [Appendix A].

SBC hides the network topology by using a method called B2BUA which is a SIP protocol that can be read about in subsection 3.1.1 [16]. If the network does not use SIP, the SBC will nevertheless use the same method. Using B2BUA has the outcome that the SBC will be the one in control of the signaling and media traffic of that call. The SBC can choose to provide media services or redirect the media traffic to another place to provide media services. The SBC can have different reasons to redirect the media traffic; for recording, generating music-on-hold or because the SBC cannot provide media services directly [16] [Appendix A].

The SBC also operates as a "traffic control" against intruders outside the IP network's borders, including security, call control, overload protection, interaction, and media management. This is done by inspecting, monitoring, and acting upon packages' signals and basing media packages on real-time policies [Appendix A]. The SBC can also make decisions based on declared policies; the SBC utilizes knowledge of the incoming and outgoing traffic of a call to change a call flow if needed. The SBC also interacts with other parts of the IP-based communication networks like the service providers and businesses [Appendix A].

3.2.2 Network placements

The standard placements for SBCs are on the network border. However, the specific placements for SBCs depend on the network topology and what it intends to do. Examples of different placements of an SBC in a network can be observed in the Figure 3.1.

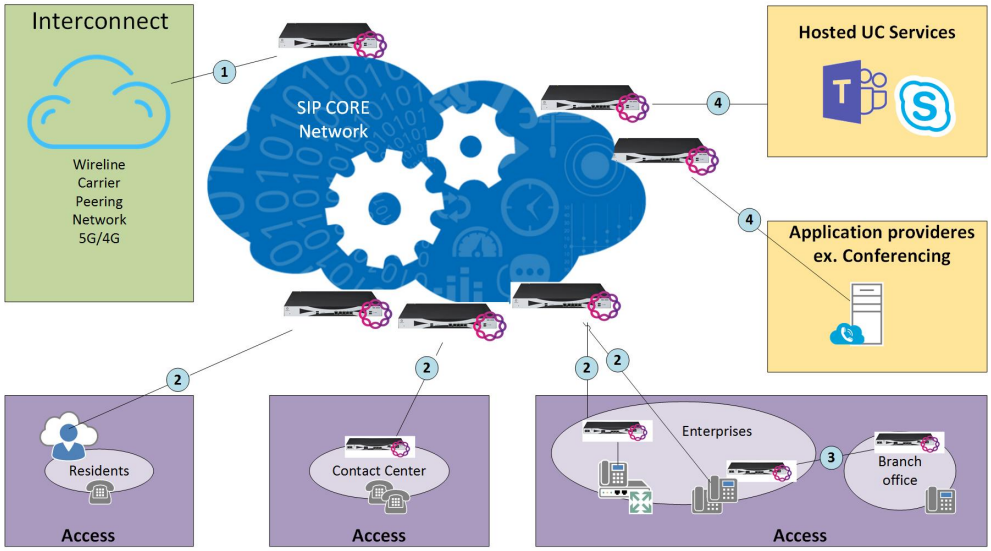


Figure 3.1: The placements of SBCs in a network. The lines between the endpoints are SIP trunks, which means that the communication over the line is grouped coherent to the type of communication and every group shares the same technology on the line.

The SBCs placement marked with 1, is the SBC set for the interconnect network borders, in this scenario it is called a Network-Network-Interface (NNI). The SBC controls the traffic between the core SIP network and the service provider. In this case, the SBC is called a Trunking SBC. A Trunking SBC can connect to numerous routing possibilities and provide encryption services [17].

The ones marked 2, are Access SBCs and a User-Network Interface (UNI) scenario. This means that the service control interface is between user devices like in 2, the backbone (transport network), and connects subscribers to their service provider (contact center). In this placement, SIP is used to protect the backbone of the network, while the SBCs protects eventual SIP trunks and the network against (D)DOS attacks. Here the SBC also works as access control, standardization of protocols, and regulates the bandwidth through Call Admission Control [18].

The SBCs marked 3 shows how the SBC can be located when a branch is de-centralized.

The SBCs marked at 4 are placed between the core network and the external services and applications, using the SBC to simplify a secure connection to the external services and applications.

3.2.3 SBC and CDR

In Ribbons SBCs, the information to generate CDRs comes from SBCs Call Accounting Manager (CAM). CAM is responsible for receiving the internal messages that include Call Accounting information and passing this information to the last destination [19]. CAM do the tasks of generating CDRs, and to support accounting event logging on SBCs [20]. When the CDRs are generated, they are stored as flat-files with a file-extension of *.ACT*.

3.2.4 Standards

There are different standards for SBC; however, Ribbon supports different standards from GSMA, IETF, International Telecommunication Union (ITU), National Emergency Number Association (NENA), SIP Forum, 3rd Generation Partnership Project (3GPP), European Telecommunications Standards Institute (ETSI), and PacketCable. Since HDO uses SBCs from the vendor Ribbon, they use them as a standard. The supported standards in Ribbon can be viewed here [21].

3.3 Call Detail Record

CDR is a record that contains data collected from telecommunications transcripts like a text message or a call that passes through a chosen device or facility [22]. CDR can collect many different data points; common data points relate to an incoming call's time and data, while others relate to the chain routing path through the infrastructure. A complete list of these data points can be seen in Ribbons' list of field descriptions [23] for CDR.

All these data objects result in an excellent dataset to monitor the business telecommunication system. Monitoring data could, over time, generate data to recognize patterns, both technical such as *what capacity is our main route running at* and personal needed at a given time based on calling patterns. These patterns are referred to as Key Performance Indicators. Additionally, in the context of handling emergency communication, data could, in theory, be selected from the database as imports for secondary software. For example, allowing emergency personnel to receive critical incident data such as the caller's phone number, how long ago the Emergency Medical Services (EMS) received the call.

EMS might also receive inquiries from Law enforcement to gather data from an incident. Considering CDR data can be stored in a database filtering out the data required can be prepared in a standardized procedure.

3.3.1 The CDR Records

Records are stored within the SBC in a flat-file format that looks similar to a CSV file. However, some groups (data finished by a comma) are sub-categorized further than simply commas, indicated by a double quotation mark (") at the beginning of a cell in Code listing 3.1.

```
1 STOP,ARIELATOM,0x00013C3E00000004,6049570,GMT+05:30-Calcutta,05/06/2020,05:26:42.1,
     11,35,[..],SBX_32316_TG_SIP_ING,,fd00:10:6b50:4040::106:6000/fd00:10:6b50
     :4510::70:6000,,10.54.7.6:6000/10.54.81.111:6012,,,196000,1225,0,,,0x00800000,,
     ,,2,2,"SIP,1-17242@fd00:10:6b50:4510::70,[..],0",[..]
```

Code listing 3.1: Partial CDR STOP Record

HDO is using SBC from a vendor called Ribbon. Consequently, our parsing schema will be based on documentation from Ribbon [24]. CDR records from Ribbon is categorized into four different kinds of records; START, STOP, ATTEMPT, and INTERMEDIATE [23]. The description of the START, STOP, ATTEMPT, and INTERMEDIATE records from *Ribbon's CDR field description* are:

START

"This record indicates that the call has been successfully established/connected and session has successfully started." [25].

STOP

"This record indicates that a previously established successful session has now terminated." [26].

ATTEMPT

"This record indicates that a call/session failure scenario where a call/session could not be successfully established." [27].

INTERMEDIATE

"This record captures any changes to signaling/call details during the session. For scenarios where the call duration is long, this record is also generated periodically to indicate session progress." [28].

Due to the records' different content, the records that are interesting for HDO and our use are ATTEMPT and STOP. There are different fields within these records, and the fields that we use are standard in Ribbon [23][Appendix A].

3.4 Microsoft SQL Server

MS-SQL is a Relational Database Management System (RDBMS) developed by Microsoft. An RDBMS is a management software for the database and can manage, administer, update, delete, and create databases. In MS-SQL, the database created is a relational database. However, an RDBMS can also be set up as a hierarchical database, network database, relational database, or object-oriented database.

3.4.1 Relational database

A relational database is a database based on the relational model. The data is organized in tables/relations containing different tables where each table has sets of rows and columns. However, the primary database does not need to contain more than one table. Each of the rows has a primary key, which is their unique identifier. Each row can also have foreign keys, where one foreign key can link such row to another row, ordinarily in another table. The foreign key is often the linked table rows' primary key.

Relational databases can have anomalies. Actions are taken to minimize these anomalies. Tables are constantly reduced to decrease potential inconsistencies within the set. This reduction is known as normalization. Normal forms are accomplished by following standard database normalization practices. There is a various degree of normalization. However, a database is generally considered to be normalized if the original three forms are met. Normal forms function in a hierarchical order, meaning that it needs to meet the first, second, and thirds qualifications to be normalized to the third-degree [29].

First normal form (1NF): eliminates repeated values, names, and groups to identify the primary key and make separate tables for each data set.
Second normal form (2NF): must first reach the First normal form's qualifications, furthermore to be considered 2NF eliminate partial dependencies.
Third normal form (3NF): must be in the Second normal form, indirectly complying with 1NF. Moreover, separate any transitive dependencies into separate tables.

3.4.2 T-SQL

SQL is the standard programming language when interacting with a relational database. However, SQL has limitations. Transact Structured Query language (T-SQL) is Microsoft and Sybase's proprietary enhancement of the SQL language [30] to streamline the execution of frequently used patterns. T-SQL could be seen as a super-set of standard SQL, including expanded capabilities for querying, aggregating, updating, and deleting [31]. Code listing 3.1 highlights some fascinating features on how implementation T-SQL could function.

```
1  -- T-SQL
2  -- Allowing the count of only DISTINCT values
3  SELECT COUNT(DISTINCT col) FROM tab;
4
5  -- Shows the differences within the table before and after the change
6  OUTPUT Deleted.col, Inserted.col
7
8  INSERT INTO tab SELECT col1,col2,... FROM tab_source
```

Code listing 3.1: Shows a subset of T-SQL's capabilities

3.4.3 Standards and services

When writing code, it is essential to follow best practices so the code is less vulnerable to known vulnerabilities and utilizes the most efficient implementations. MS-SQL using T-SQL, which do not currently have any standards. When writing the source code in MS-SQL, we used [32, 33] as our standard, written by Pluralsight author and the owner of SQLAuthority Pinal Dave [34]. The standards or guidelines are a list that is compliant with the best practice for writing secure and efficient SQL code. Although the sites were written in 2007, they still seemed promising considering they comply with the best practices we have previously learned during courses where SQL has been in the syllabus at NTNU. Moreover, the use of a standard ensures that we are delivering a uniform product to HDO.

MS-SQL's many different services and tools that can expand MS-SQL use, e.g., SSIS for parsing data. This thesis used four of these, SSMS, SSDT, SSIS, and SSRS. Implementation for each of the services can be read about in section 3.5, 3.6, 3.7, and 3.8.

3.5 SQL Server Management Studio

SSMS is a Microsoft environment that is integrated with all SQL infrastructure from Microsoft. This means that SSMS is used to configure, administrate and manage a SQL infrastructure, like to manage scripts, tasks and SQL databases [35]. SSMS is also the Graphical user interface (GUI) for MS-SQL.

3.6 SQL Server Data Tools

SSDT is a development tool that uses Visual Studio to design and deploy databases. SSDT can also build SQL server relational databases like the ones written in MS-SQL, and deploy and design reports, visualization and other tasks in SSAS, SSRS, and SSIS [36].

3.7 SQL Server Integration Services

> 'SSIS is a component of the Microsoft SQL Server database software that can be used to perform a broad range of data migration tasks. SSIS is a platform for data integration and workflow applications. It features a data warehousing tool used for data extraction, transformation, and loading (ETL)' [37].

The integration service is capable of extracting and transforming data from many different sources such as flat-file and relational data sources, then load the data into one or more destinations such as a relational database [38]. SSIS packages are

a collection of connections, control flow elements, data flow elements, event handlers, variables, parameters, and configurations that are assembled using Microsoft Visual Studio (Visual Studio) GUI environment or through plain code [39]. The package itself is the main component of SSIS. For instance, it is the package that is being executed on the system to perform a specific task. As mentioned earlier, those tasks are related to data extraction, transformation, and loading. An example would be that an SSIS package is used to import a flat-file, parse and transform specific data from the flat-file and then export the finalized data into a database.

For the SSIS package to be efficient and consistent, it is often utilized with the SQL Server Agent for automating and scheduling the execution of a package on a system, which is accomplished through an SQL Server Agent Job. Since the SQL Server Agent runs as a windows service, the execution of a package can occur periodically as long as windows are running [40].

For the SSIS package, essential elements such as the control flow and the data flow are the foundation of how a package operates and interacts. The package itself can be built through Visual Studio GUI or by plain code. The following figure illustrates the core components that a package consists of:

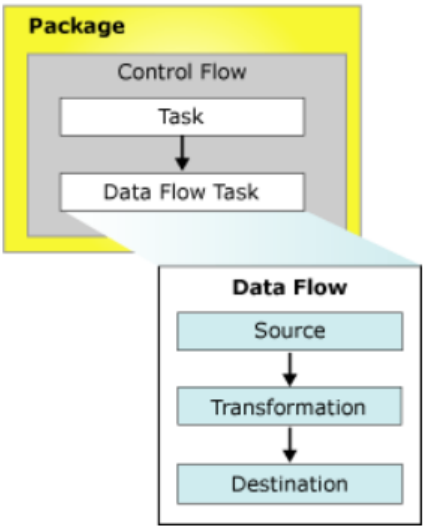


Figure 3.2: SSIS package [39]

3.7.1 Control flow

The control flow defines what to do with the contents from the data source. The control flow itself consists of one or more tasks and containers that execute when the package runs. Precedence constraints are used to connect tasks and containers in a package and to control and define the conditions for running the next task or container [39]. An example of a container is a for each loop or a for loop to

go through a sequence of files, while an example of a task could be a file system task for handling files. A container and a task is often used together, an example is that the container goes through several flat-file in a directory and the task itself is deleting each file from the directory.

3.7.2 Data flow

The control flow uses a data flow task to perform the data flow. The data flow works with the data and consists of three components; Source, Transformation, and Destination, including the path that links the components together [39].

- **Source**

  The source component is responsible for extracting data from various data sources such as a flat-file. The extracted data could for instance be ten columns with various row values. It is the source component that provides data to the other components for further use. The data flow itself can use multiple data sources within a package if necessary [41].

- **Transformation**

  The transformation component has many capabilities. The component can perform various tasks related to the transformation and manipulation of data. More specifically, updating, summarizing, cleaning, merging, and distributing data. It is also capable of modifying column values [41].

- **Destination**

  The destination component is capable of writing non transformed or transformed data into one or several data stores, such as a flat-file or a database [41].

3.8 SQL Server Reporting Services

SSRS is a reporting service component in MS-SQL, where the reports can be made in Visual Studio. SSRS is used to create, deploy, and manage reports, and the reports can be viewed, for example, in a web portal or a browser, on a computer or mobile. The reports can be directly deployed or turned into PDFs using report builder, Visual Studio, or other SSRS programs. The reports are a visual way of showing data. The data visualization occurs by using tables, diagrams or graphs. For business use, SSRS is a tool that can visualize data like a budget or charts to compare how well something is selling. SSRS is used to visualize statistics, like downtime and the number of connections to a network at a given time. Individuals can also use SSRS for their use, like showing their budget or watching exercise progression. There are different types of reports, where some may need multiple data sources and datasets; therefore, SSRS can connect to one or more data sources and datasets, depending on the report's demands. There are different types of

reports and actions that SSRS can make; however, only some are relevant for those made here. The relevant report types are drilldown, and parameterized reports.

Drilldown: is a report type and an action that can hide rows in a table. The user viewing the report can, by clicking on the "+/-" symbol on the side, choose to show or hide the rows [42].
Parameterized reports: are reports where the users use parameters to filter and customize the reports. In this type of report, it is used input value and filters as parameters [43].

The SSRS reports are managed from a report manager, a web-based GUI. The administrator of the report manager can, for instance, choose who has access to the different reports and, if a report is going to be automatically deployed and who will receive the reports.

3.8.1 Data source

A data source is needed to extract data to use in SSRS. The data source can be sources like a database, which the computer or server the SSRS is on must be connected to the source. The data sources are connected using a connection string. The connection string is different depending on what the data source is; a list containing most of the different types with their connection string can be found in this link [44].

The sources can be connected using an embedded or shared data connection. The embedded connection can only connect to one report. Where as, the share data connection can be connected to more than one report.

3.8.2 Dataset

The dataset contains the data from a data source. The datasets can be either shared or embedded; the shared data can be used in other reports, while the embedded dataset can only utilize in the report its creation happened in [45]. The data is extracted from the data source using a query. If the database is from MS-SQL, the user have to use T-SQL instead of SQL, like in Code listing 3.2. The code also shows how dynamic queries are set up in T-SQL in the `WHERE` clause. In this case, the parameter is optional; by the last part of the `WHERE` clause, where the parameter can be `NULL` or not chosen, if a value for the parameter is selected, it will be equal `StartedCalls` from the data source.

```
1  SELECT  TestTimes, StartedCalls, SimCalls
2  FROM    Test_View
3  WHERE   (StartedCalls = @StartedCallsParameter) OR (@StartedCallsParameter IS NULL)
```

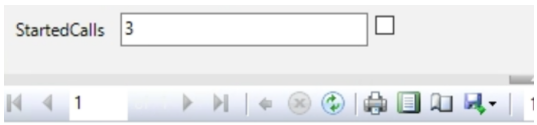
Code listing 3.2: Shows a dynamic query

In the dataset queries, the data can be modified and dropped if needed for the

report. The connection to the parameter is also defined here by using dynamic queries.

3.8.3 Parameter

Parameters in SSRS are utilized for filtering the reports and can take one of the types text, boolean, date/time, integer, or float. Depending on the type, the parameter can be allowed to be: blank value (""), null value, and/or multiple values. The blank value is often for texts where the text can contain a blank value, the null value applies when the parameter is optional, the multiple values are when various values can be applied. The parameters can also have default values and available values [46]. The values listed in default values and available values often contain a drop-down list, which is either specified in a query or each value is specifically chosen. The drop-down list contains names or values that are predetermined in a list for a parameter.



Test

Period Start	Started Calls	Sim Calls
13:00:00	3	1
13:00:00	3	1

Figure 3.3: SSRS test report that contains a parameter that can have a null value or a number value.

3.9 SSH File Transfer Protocol

'SSH File Transfer Protocol (SFTP) is a network protocol that provides file access, file transfer, and file management over any reliable data stream' [47].
The protocol is designed by the IETF and it provides capabilities for secure file transferring as opposed to File Transfer Protocol (FTP). SFTP does not provide any security functionalities by itself, however, it expects the underlying protocol to provide security. This is why SFTP is seen as an extension for Secure shell (SSH) and not FTP since the underlying protocol that provides security for SFTP is SSH. SFTP usually runs on port 22, the same as SSH.

For SFTP to transfer a file from one machine to another over a secure connection, an SFTP client and an SFTP server are required. The server acts as a data store where files are stored, and where files are retrieved. The client is a type of software

that can connect to an SFTP server, further the client can upload files to the server, and it can also download files from the server [48].

3.10 Grafana

'Grafana is a multi-platform open source analytics and interactive visualization web application. It provides charts, graphs, and alerts for the web when connected to supported data sources' [49].
Grafana has many uses for both companies and private individuals, an example is that a private individual can monitor a digital temperature gauge over time to analyze how it has been operating. For Grafana to visualize data, essential building blocks such as the data sources and the dashboards need to be configured appropriately. A data source acts as a storage backend for Grafana, and Grafana utilizes different queries to retrieve specified data from the data source. These queries are different depending on what type of data source is used and what data the user wants to retrieve [50, Data sources]. Different data sources that are compatible with Grafana include:

- Time series databases such as Prometheus and InfluxDB.
- Logging and document databases such as Elasticsearch.
- Enterprise plugins such as Splunk, Oracle, and MongoDB.
- Cloud such as Azure monitor and Google cloud monitoring.
- SQL such as MS-SQL and PostgreSQL.
- Distributed tracing such as Zipkin and Tempo.

Dashboards act as a visualization tool for Grafana, administrators that use Grafana typically operate on several dashboards to monitor large data sets. A dashboard typically contains several panels that represent the visualization of the data, Grafana ships with a variety of panels, and more panels can be installed through their plugin system [50, Dashboards]. A panel has a query editor that is dependent on the data source used for that panel, the query for retrieving data is different depending on the data source. For instance, the query for a time series database is different compared to a SQL database. However, the different types of queries are flexible, for instance, it is possible to execute time series queries towards an SQL database. With the query editor, the panel can visualize the desired data in different styles and formats, such as graphs and gauges [50, Panels]. In addition, Grafana utilizes macros for simplifying syntax, which leads to the queries being more dynamic. For instance, in a query where the goal is to visualize data over time in a graph. One could use built-in macros to simplify the timestamp represented in the x-axis of the graph. An example is the following macro: *$__timeEpoch(dateColumn)* [51]. The dateColumn is represented in the datetime format, while the parameter itself is the time column from the database. This leads to dynamic queries, which makes it easier to correlate data values at specific timestamps in a graph.

```
SELECT
  $__timeEpoch(tTime),
  SER as A,
  SEER as B,
  SDR as C
FROM dbo.test
WHERE $__timeFilter(tTime)
ORDER BY tTime ASC
```

Code listing 3.3: Query on an MS-SQL database. Built in macros are used to simply the syntax. The query results in a panel that displays a graph in Figure 3.4

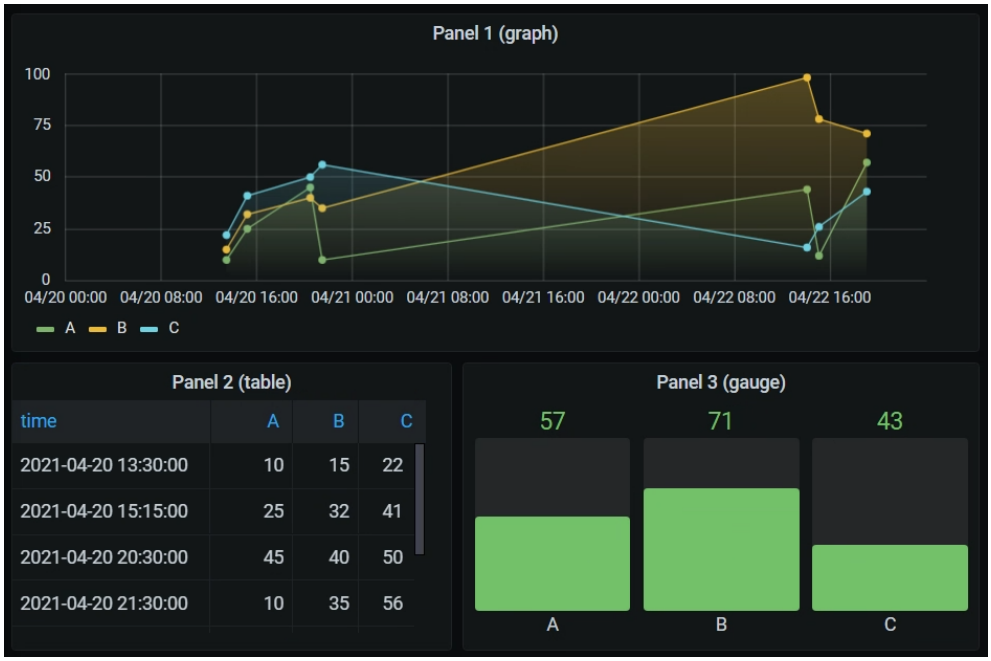


Figure 3.4: Test data displayed in a Grafana dashboard through three different panels based on a MS-SQL data source

By default, Grafana is shipped with basic plugins to meet the visualization needs. However, there are additional plugins available for users to install to enhance and tailor their visualization tasks. These plugins are either created by the Grafana community or by the individual itself if the need is there. Panels, data sources, and apps are the three types of plugins that are supported in Grafana [50, Plugins].

3.11 Laws and Regulations

Managing personal data within an emergency system holds strict outlining concerning how data can be handled in a constitutional matter.

Norwegian law requires all companies that handle customer data to operate within the scope of General Data Protection Regulation (GDPR) and the Personal Data Act (NO: personopplysningsloven). Particular fields can have additional governing documents and regulations on how data shall be processed. The health and care services operate under Normen [52, In Norwegian] and the emergency medicine regulations (NO: akuttmedisinforskriften).

In section 1.1, we explained how the current solution for Norwegians health telephone network is not adequate by today's standards. Paraphrasing The emergency medicine regulations § 12 d "to facilitate the safe operation of the national emergency room number, including providing alternative answering places." Secure that new service centers can be interconnected with a lower threshold and systemic monitoring of weak points within the network could plausibly result in safer operation.

Generating sampling data for monitoring comes with the drawback that data considered personal could be collected for further processing. All of the Laws mentioned above, regulations, and governing documents address the necessity for having procedures in place for handling such information. Limitations to minimum data sufficiency for fulfilling the needs in terms of KPI requirements relevant for our work are discussed further in subsection 4.2.1 and in section 4.5

3.12 Existing Technologies of Special Interest

3.12.1 Parsing of a flat-file

As stated earlier, this thesis uses flat-files as the data source. For collecting and utilizing data from flat-files, the flat-files need to be parsed, and then the extracted data is sent to the database. For this thesis, the two solutions we investigated were using a script or SSIS to parse and send the data values to the database.

Scripting

The first solution that we investigated was parsing with scripts. For using and modifying a script, it is crucial to understand the programming language. Therefore, the parser relevant for this thesis was written in PHP (Hypertext Preprocessor), Rust, Python, and C++. Another criterion was that the parser worked on a flat-file which is a comma and separated using the signs colon and double quotation mark. Since the flat-file we work with is comparable to a CSV file, it was relevant to look at CSV solutions as a starting point, then modify it to our need.

The group had little experience with PHP and Rust. Hence they were later removed as options; meanwhile, everyone in the group has experience with the programming languages Python and C++. The option for a C++ parser was [53], which is written and maintained by Vincent La [54]. The author of the C++ parser stated in the motivation text that it was inspired by the CSV module [55] in Python. Python is a fast programming language that is easy to learn and use. Python also

has a library for writing and reading CSV files, [55] it was easy to find and read the documentation [56] made by Python. With this in mind, Python became the preferred programming language for scripting.

SSIS

However, we learned from an interview with Stig Atle that HDO already uses SSIS for importing flat-files. SSIS looked easier to work with alongside SSRS and MS-SQL rather than using a script. One consequence of having the parser on software instead of a script is that it can be critical for HDO and this thesis if the software becomes unavailable. If the parsing were a script, the script could easily be changed or modified to another programming language without regard to software.

3.12.2 Deciding on an SFTP Server

SFTP is a commonly used technology among numerous divisions within IT. Because of this, numerous different SFTP server software are available on the Internet. HDO informed us that we should opt for a free SFTP server capable of running as a Windows service.

As a first step, we investigated Open-Source solutions. When searching through Open-Source solutions, it was additionally essential to find an active project. This is to ensure that standards were maintained and arising bugs were fixed promptly. As an indicator of the maturity of these types of projects, we looked at the number of commits. When the last commit was pushed, and when new documents were last altered as indicators. After we analyzed different projects, we decided to look further into SFTP using OpenSSH and SFTPgo [57]. However, after seeing some implementation concerns with OpenSSH on Windows and lacking support for MS-SQL within SFTPgo [57], we moved over to look at proprietary solutions that provided free academic licenses. Because of this, Rebex: Buru SFTP Server Free Edition [58] was chosen with features like virtual mapping, allowing us to simulate the root folder (default sign in a directory) and an easy-to-navigate Web Administration dashboard.

Chapter 4

Design and Implementation of the Monitoring System

After reading chapter 3, one would realize that there are many prerequisites which is recommended to be in place for developing the end product. To know these technologies in detail and at the same time know how to use them in different cases are important for developing the end product in a time efficient manner.

This chapter explains how each component is configured and used in an automated workflow that results in an end product. It addresses how the database and its tables are designed to handle data from the CDR files in the most efficient way. It explains how SSIS is used to deploy a package that performs a variety of data migration tasks on every CDR file. The configurations of the package lead to each arriving file from the SBC being handled correctly before the transformed data is written to the database. Finally, chapter 4 deals with how SSRS and Grafana are configured to visualize data in the database. SSRS is configured to visualize reports, while Grafana is configured to visualize KPI's. Using these visualization tools, HDO would be able to monitor and troubleshoot their network for deviations.

4.1 System architecture

The environment for developing the end solution was set up by HDO. The environment is not part of HDO's production environment and it will only be used for developing and testing the end product. The domain consists of two Windows servers, one Linux server, and three VMware horizon remote clients, which are used for accessing the development environment. One windows server is running MS-SQL, SSIS and SFTP, the other windows server is running SSRS, while the Linux server is running Grafana. The software on the servers was not pre-installed, which meant we had to install and configure all software with the necessary packages and tools to have a functioning development environment. The development environment is connected to a SBC. The SBC is not part of HDO's production environment, but it will act as a node for testing the data flow towards the development environment

regarding transmitting CDR data over SFTP.

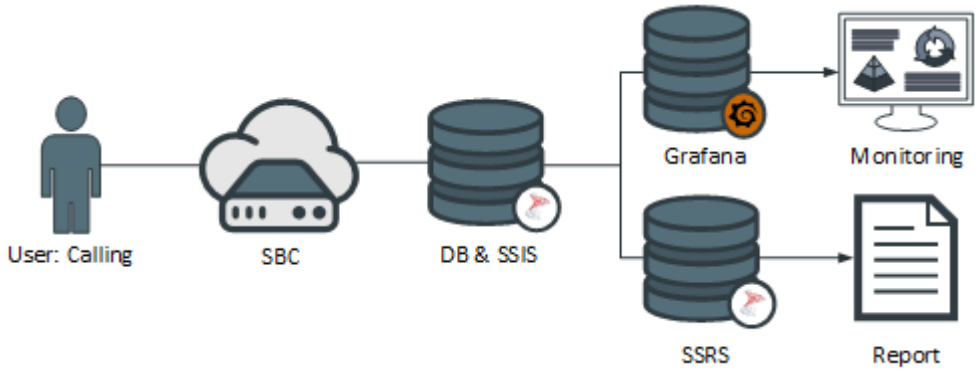


Figure 4.1: Shows a slice of an SBC/SIP network infrastructure, focusing down on the components this report operates in.

4.2 Database design

Database design can be considered the established structure needed to be implemented for the logical and physical components required by the application(s) and user(s) to work efficiently within the information system [59].

In section 3.4, it was mentioned that MS-SQL is constituted as a relational database. Moreover, this section emphasized on normalization practice. In contrast, after examining the CDR data received from the SBC and interacting with HDO [Appendix A], a single raw data table reaching 1NF was singled out as the central target table seen in Figure 4.2.

Grounding this solution was that every row within the table should be seen as a confined event. This confinement might sound contradictory to generate historical patterns. However, this RAW table (Figure 4.2) can be regarded moderately as a short-lived arrangement for SSIS interactions and origin for data aggregation procedures. SSIS will systematically restate data inside this table, both inserting rows and deleting rows that surpass a 30-day time-frame, discussed in further detail in section 4.3.

4.2.1 Governing data

Operating as a data controller requires the system to uphold the laws and regulations introduced in section 3.11. Therefore, essential factors when storing CDR data are determining what information is considered personal data. Considering HDO operates in Norway: The Norwegian Data Protection Authority's definition of personal data is used for direction and can be found in detail here [60, In Norwegian].
Only an assortment of data in the CDR file is picked out for further processing

Figure 4.2: CDR_RAW table, data types and cell sizes

when inserting data into the database. Among this selected set, `CallingNumber` and `CalledNumber` are the only fields that fall under the classification of personal data.

Considering HDOs boundary, data should only be stored in a raw format for 30 days and instead aggregate down data important for long-term storage before this mark [Appendix A]. This limitation ensures that the data collection falls within an acceptable period for data storage and is compliant with GDPRs Article 5(e); *data shall not be stored longer than required* [61]. Based on the current data aggregation schema discussed in subsection 4.2.2, no personal data is further processed for long-term storage. It is crucial to note that if this 30-day limitation is increased, an archive table excluding the two columns should be considered. Unless they fulfill a meaningful purpose, other anonymization techniques could be used, such as salted hashes for storing personally identifiable information for extended periods.

4.2.2 Data Aggregation

Clustering data from the primary table CDR_RAW allows subroutines within SSRS and Grafana to utilized a finite set of the data. This aggregation strengthens the performance for common queries. Additionally, aggregation enables the adaption of the table structure; one such adaption can be seen in Figure 4.3.

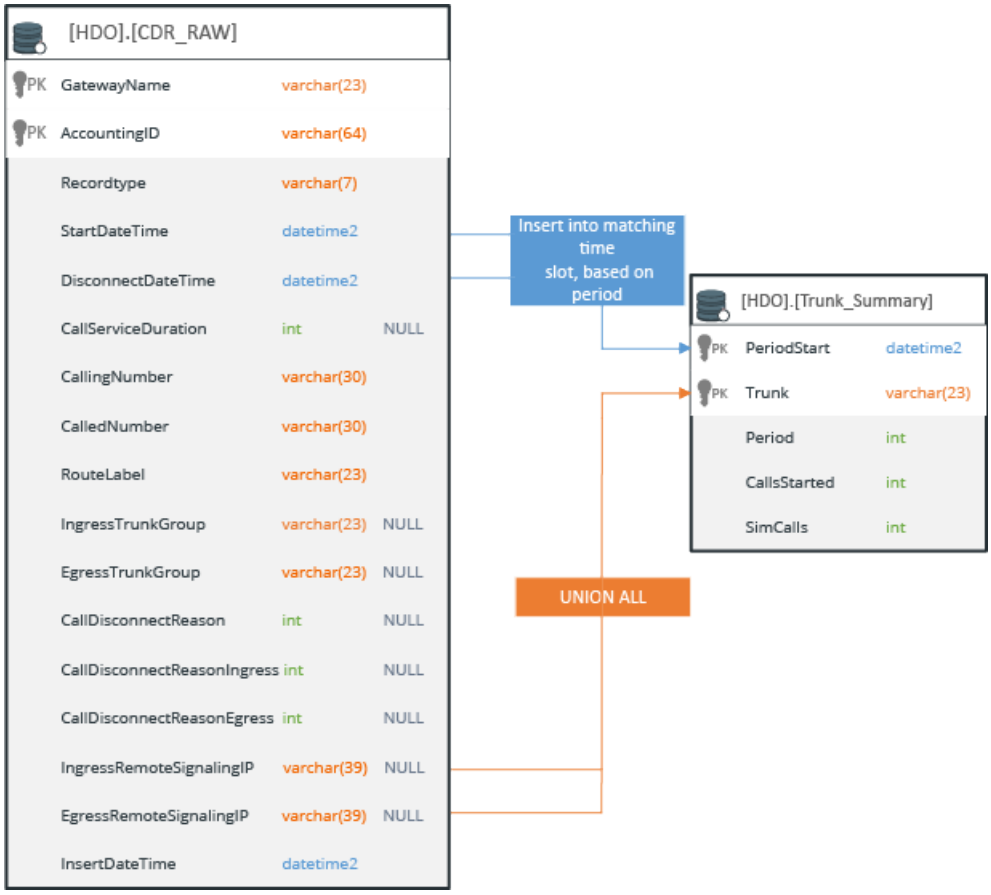


Figure 4.3: Aggregation flow for Trunk Summary

Together with HDO, three aggregation tables (Figure 4.4) were picked out as required. These tables were chosen to enhance underlying data from supplemental monitoring sources with potent bandwidth activity and user trends, including calculating call occurrences, mapping disconnection reason(s), and route activity within a given period [Appendix A].

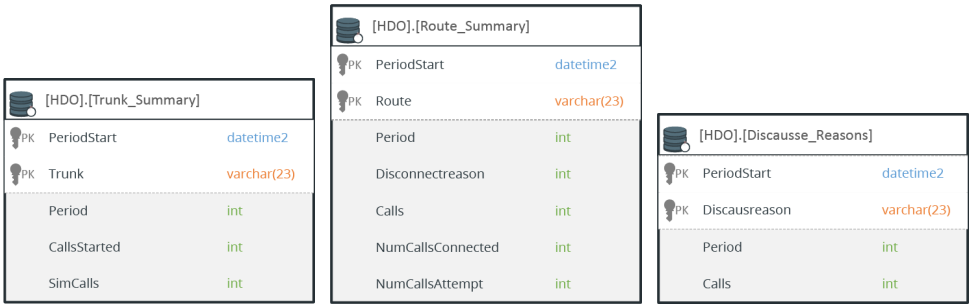


Figure 4.4: Displayes aggregation tables cell names and types

The schematic of the three tables can be seen in Figure 4.4 stored as Trunk_Summary, Route_Summary, and Discause_reasons. Moreover, they are utilizing Stored Procedures for querying, and data insert. The complete procedures can be found in Appendix D.1, D.2, D.3. A notable feature for all of them is the possibility to arrange a dataset matching given criteria into a 30-minute time slot (Shown in Table 4.1, 4.2).

Recordtype	StartDateTime	DisconnectDateTime	In_Trunk
*STOP	2021-02-24 20:24:34.6	2021-02-24 20:33:05.1	I1
*STOP	2021-02-24 20:23:43.1	2021-02-24 20:32:32.9	I1
STOP	2021-02-24 20:31:09.5	2021-02-24 20:32:51.4	I1
ATTEMPT	2021-02-24 20:31:51.1	2021-02-24 20:33:23.2	NULL

Table 4.1: Displays a portion of CDR_RAW with tagged rows (*) symbolizing that the two rows are members of the same 30-min period (20-20:30)

PeriodStart	Period	Trunk	StartedCalls	SimCalls
* 2021-02-24 20:00:00.0	1800	I1	2	2
2021-02-24 20:30:00.0	1800	I1	1	3

Table 4.2: Shows a range of Trunk summery with the marked(*) row representing the grouping from the table above

A Tally Table is created as a supporting table, and this table helps set a boundary when iterating over periods. In this instance, a Tally table with 48 rows is created $\left(\frac{24*60}{30} = 48\right)$. The Tally table results are then used within a derived table (subquery nested inside a `FROM` clause). Finally, it joins the result with the RAW data to map the original row into the newly aggregated periods. Code listing 4.1 shows a base implementation of this concept for all three tables.

```
1  DECLARE @date -- datetime2
2
3  WITH
4      numbers(val)
5      AS
6      (
7              SELECT 1
8          UNION all
9              SELECT val + 1
10             FROM numbers
```

```
11              WHERE val < 48
12      ),
13      periods
14      AS
15      (
16          SELECT @date AS [date], nbr.val,
17              dateadd(minute, (nbr.val - 1) * 30, @date) AS PeriodStart,
18              dateadd(minute, (nbr.val    ) * 30, @date) AS PeriodEnd
19          FROM numbers AS nbr
20      )
21  SELECT
22      pers.PeriodStart
23          , @period AS Period
24          , src.CallDisconnectReASon AS DiscausseReASon
25          , COUNT(*) AS 'Calls'
26  FROM periods AS pers INNER JOIN HDO.CDR_RAW AS src
27      ON src.StartDateTime >= pers.PeriodStart
28      AND src.StartDateTime < pers.PeriodEnd
```

Code listing 4.1: Tally table joined over RAW data with the goal to form time slots

The implementation for the three tables is similar. Nevertheless, Trunk_Summary also has a column that aggregates all simultaneous calls counted over several periods. Simultaneous calls can be defined as the maximum number of operators communicating with a client at once during a period. At the same time, the function needs to take into account calls started in an earlier period that is ongoing into the next period if a call within the next slot is started before the earliest call is terminated, considering two or more operators are working in parallel.

Modifying the base implementation with an additional derived table joins the periods over the raw data and limiting the query result based upon its `StartDateTime` and `DisconnectionDateTime` seen within the `CASE` clauses in Code listing 4.2. This behavior allows us to generate a count based on whether the current row is greater than the second row. The function for this is shown on lines 81-118 within Appendix D.1. It can then be iterated over with the help of two SQL functions `MAX(COALESCE([COLUMN], delimiter (0)))`. `COALESCE` inserts all rows into an array until the delimiter zero (0) is met. Applying `MAX` on the output array allows us to only return the highest value within the array, which equals the maximum simultaneous calls for a given period.

```
1  WITH Numbers(val) AS
2  (
3    [..] -- Implemented as previously shown in Codelisting 4.1
4  )
5  ,ConnectionPeriod AS
6  (
7    SELECT
8      period.period_start
9     ,period.period_end
```

```
10      ,@period as Period
11      ,src.Trunk
12      ,CASE
13         WHEN src.StartDateTime < period.period_start THEN period.period_start
14         ELSE src.StartDateTime
15       END AS StartDateTime
16      ,CASE
17         WHEN src.DisconnectDateTime > period.period_end THEN period.period_end
18         ELSE src.DisconnectDateTime
19       END AS DisconnectDateTime
20    FROM
21      (
22        SELECT
23          dateadd(minute, (val - 1) * 30, @date) as period_start
24         ,dateadd(minute, (val    ) * 30, @date) as period_end
25        FROM
26          numbers
27      ) period
28    INNER JOIN
29      (
30        SELECT
31          IngressTrunkGroup as Trunk
32         ,StartDateTime
33         ,DisconnectDateTime
34        FROM
35          [HDO].[CDR_RAW]
36                  WHERE StartDateTime BETWEEN @ST AND @DT -- Limits to Timeinterval
37          UNION ALL
38        SELECT
39          EgressTrunkGroup
40         ,StartDateTime
41         ,DisconnectDateTime
42        FROM
43          [HDO].[CDR_RAW]
44                  WHERE StartDateTime BETWEEN @ST AND @DT -- Limits to Timeinterval
45      ) src
46        ON src.StartDateTime >= period.period_start
47            AND src.StartDateTime < period.period_end
```

Code listing 4.2: Trunk_Summary additional joining over RAW data to allow calculating cell result based on forthcoming rows. Before time slot grouping.

4.2.3 SQL job automation

MS-SQL is equipped with a manager to handle job automation tasks. This scheduler is SQL Server Agent and allows a database administrator to list jobs for execution. A benefit of utilizing SQL Server Agent to execute stored procedures as the three managing data aggregation within this system is to calculate parameters that are part of the execution call as a precursor to the call. For example, the stored procedures discussed in subsection 4.2.2 utilize this precursor to parameterize a date value (Code listing 4.3) essential for the functionality within the procedure

to grouping records to a time slot. Moreover, the request of the current date needs to be dynamic as this job is scheduled to occur automatically once every day.

```
DECLARE @RUN_DATE datetime2(1) = GETDATE();
DECLARE @START date = DATEADD(day,-1, @RUN_DATE);
DECLARE @END date= DATEADD(day, 1, @START);


EXECUTE [HDO].[GenerateRouteSum] @date = @START, @period = 1800,
    @ST = @START, @DT = @END;
```

Code listing 4.3: SQL Server Agent scheduling the execution of Route_Summary with parameters

4.3 SSIS

The SSIS package that is developed under this project is responsible for importing CDR files, extract and transform relevant data from those files, and export the resulting data into a SQL database for further use. The package is set to run every hour on the system to ensure that all the CDR files that are received by the SBC are handled as efficiently as possible. To accomplish this, the SQL Server Agent has a job that is related to executing the package every hour. This will ensure consistency and reliability. (Appendix H illustrates how the SQL Agent Job is configured to automatically execute the SSIS package.) The package itself consists of two important elements that act as the foundation for the package in terms of what the package can do. These elements are the control flow and the data flow.

The control flow of this package is built up of several tasks and one container to handle and manage CDR files. The data flow is connected to the control flow through a data flow task. The data flow for this package is built up of source-, transformation-, and destination components to handle the data migration process. The SSIS package that is configured for this project is inspired by the previous package that HDO used. Much is relatively similar in both the control flow and in the data flow. Nevertheless, there are some differences. In our package, for instance, date and time are handled differently, fewer control flow tasks are used, and fewer data flow tasks are used.

However, for the package to run as desired, SSIS needs to have a folder environment the package can work towards. That environment is related to where the CDR files are on the system, and where to move them after processing. The following bullet points explain how the folder environment is connected to the SSIS packages:

- CDR files that arrive at the SBC are sent to the Windows server that runs MS-SQL and SSIS (with SFTP). CDR files are automatically put in a folder named *fromSBC*.

- The SSIS package imports all the CDR files available in the *fromSBC* folder for further transformation.
- If the SSIS package manages to execute without any errors, the CDR file is moved from the *fromSBC* folder into the folder named *archive*. Successful execution will also result in the transformed data being inserted into the database. If any errors occur during the transformation process of a CDR file, the file is moved from the *fromSBC* folder into the folder named *error*.

4.3.1 The Package

Figure 4.5 illustrates the contents and the flow of the control flow within the package, while Table 4.3 illustrates a more detailed overview. The detailed overview provides an insight into which components are used within SSIS for configuring the control flow. For instance, the data flow task, file system task and the for each loop container. The figure also contains information on how the components are used, the field named *description* addresses that. Figure 4.6 and Table 4.4 does also show the same information as the control flow, but for the data flow instead.

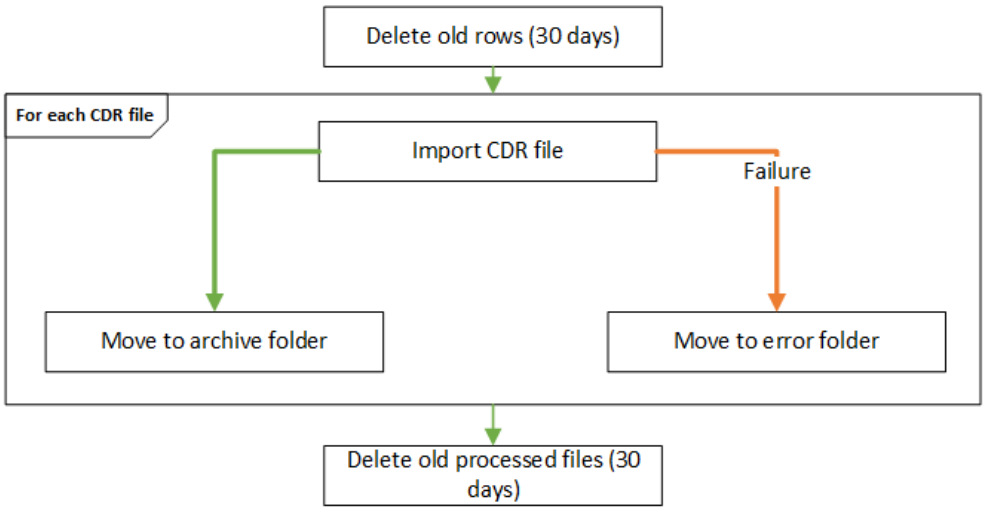


Figure 4.5: Control flow, explained in Table 4.3

Name	**Control flow component**	**Description**
Delete old rows (30 days)	Execute SQL task	Delete all rows in the CDR_RAW table that have been stored in the database for over 30 days.
For each CDR file	Foreach loop container	Go through all the CDR files in a folder (.ACT extension). This ensures that every CDR file is going to be processed in the data flow.
Import CDR file	Data flow task	The following data flow task is responsible for importing, transforming, and exporting the CDR file. See Figure 4.6 and Table 4.4.
Move to archive folder	File system task	During a data flow task for a CDR file, if the data flow task executes without any errors, then the CDR file is moved to the archive folder.
Move to error folder	File system task	During a data flow task for a CDR file, if the data flow task executes with errors, then the CDR file is moved into the error folder for manual analysis.
Delete old processed files (30 days)	Script task	Delete all CDR files in the archive folder that have been stored for over 30 days.

Table 4.3: Overview of the control flow

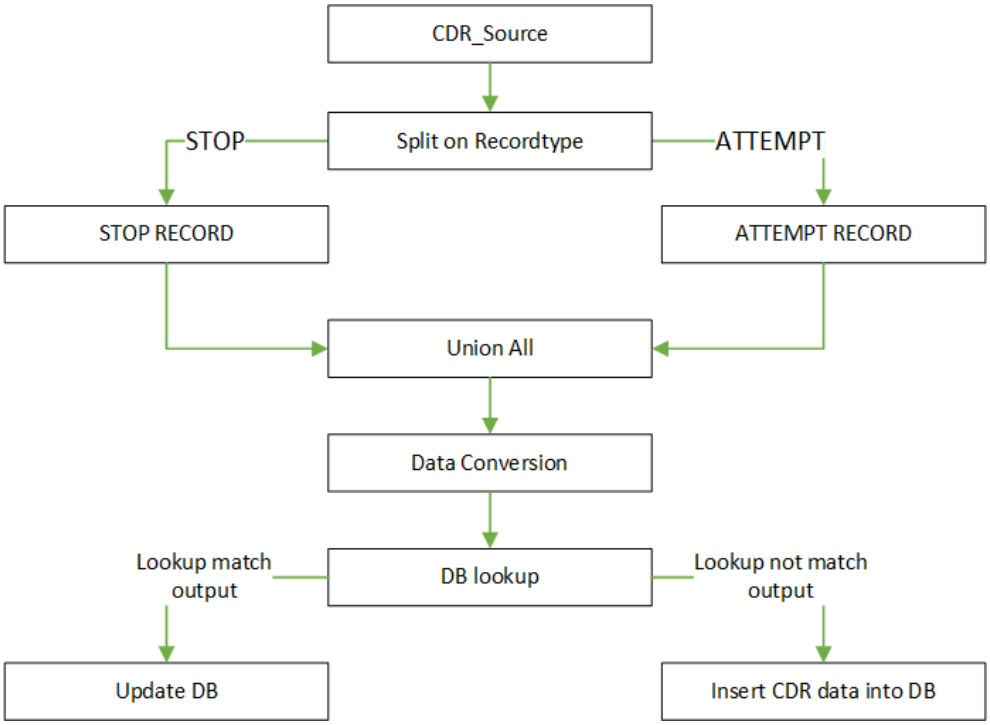


Figure 4.6: Data flow, explained in Table 4.4

Name	**Data flow component**	**Description**
CDR_Source	Flat file source	Imports a flat-file (CDR file) where the desired columns from the file are extracted for further processing.
Split on Recordtype	Conditional split	Each CDR file is split based on what the Recordtype column contains, files containing STOP or ATTEMPT are used for further processing.
STOP- and ATTEMPT record	Derived Column	In these two derived column tasks, functions are used to build expressions to handle different columns. Columns are transformed to contain the desired data. For instance, merge a date column and a time column into one column. In addition, columns with potential NULL values are handled here.
Union All	Union All	Union all is responsible for combining the two input records into one output for further processing.
Data Conversion	Data Conversion	The data types for some columns from the CDR file is not compatible with the database. For those columns, their data type is changed. E.g. Unicode to Non-Unicode string.
DB lookup	Lookup	A lookup is performed into the database to compare source- and destination data on the primary key. Depending on the comparison, two conditions can occur.
Insert CDR data into DB	OLE DB Destination	First condition: If the primary key for a CDR record is not present in the database, the following record is inserted into the database.
Update DB	OLE DB Command	Second condition: If the CDR record already exist in the database, but some of the row values are not correct in that record. An update statement is ran to update the rows in the database with the correct value from the data source.

Table 4.4: Overview of the data flow

Figure 4.7 illustrates how the STOP RECORD task is configured within the data

flow. *Derived column name* is responsible for creating and naming the columns, what the columns contain is decided by how the expression is configured. The expressions are built up by square brackets with a number that relates to the column number from the CDR file, which means that the correct column is retrieved from the source file. Functions are also used to handle potential `NULL` values and specific data types. For instance, there is an expression that retrieves column number two from the CDR file and all of its row values. In this case, that is the column named *GatewayName*. That column is not manipulated at all, as there is no need for it. However, the column named *CallServiceDuration* has been manipulated with a type cast function to make sure it contains the correct value in the database. What indicates that the data type of a column needs to be manipulated and transformed will vary according to each individual column. It all depends on what type of data SSIS and SQL are compatible with managing, or what data HDO is interested in. For instance, the values from the *CallServiceDuration* column are represented in milliseconds from the source file. In this derived column, that is altered since HDO wants the value to be in seconds instead.

The headers named *data type* and *length* are responsible to show what type of data the different columns are, and how many characters are allowed to be stored in that column. For the majority of the columns, if necessary, their data type is converted in the task named data conversion that occurs later in the data flow and not in the derived column task.

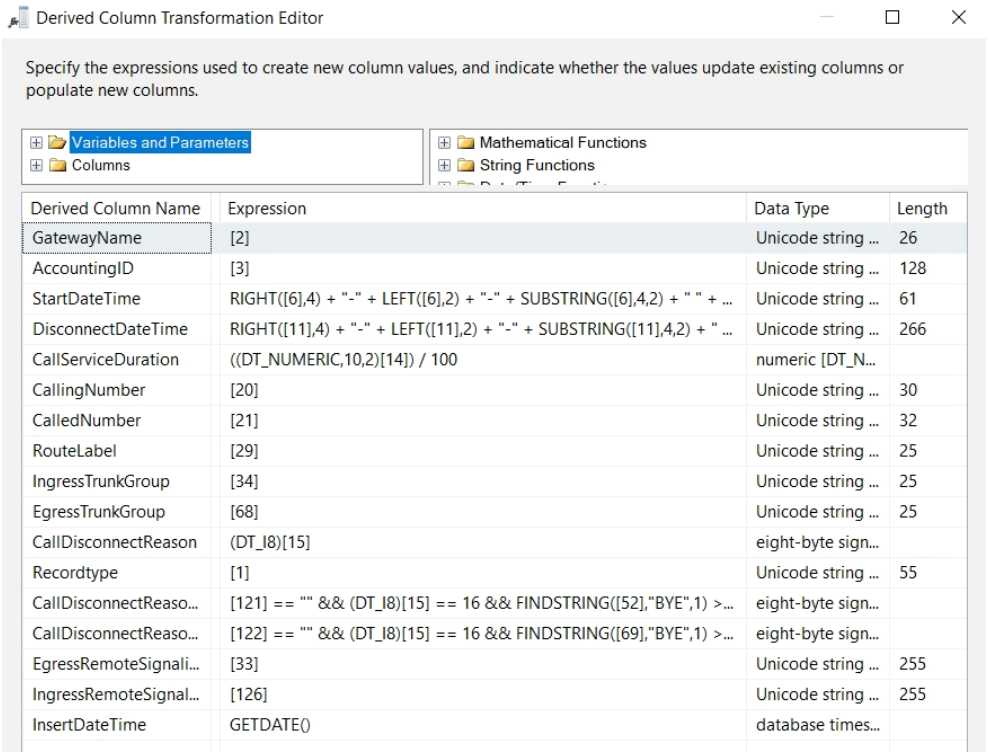


Figure 4.7: Configuration of the STOP RECORD task in SSIS

The configurations for the SSIS package is available in [Appendix G], it contains, among other things, this:

- Configuration of tasks and containers in the control flow
- Configuration of tasks in the data flow
- Configuration of properties for tasks
- Connection managers
- Expressions for the derived columns tasks
- Variables
- Event handlers

4.4 SSRS

SSRS is utilized to visualize data from the CDR files. SSRS are connected to MS-SQL as the data source. We made four reports [Appendix I.1] that can be filtered by different parameters, depending on the content of the reports, these are detailed below. The information and filters set in the reports were discussed together with HDO [Appendix A].

- **Call search** shows all the incoming calls and the associated information to the calls in a chosen time period, and can be filtered by Date/Time, Route,

Trunk, calling number and called number.

- **Disconnect reason stopped calls** shows the calls that stopped, with their disconnection codes and route. This report is sorted by routes and is filtered by Date/Time, Route and Disconnect reason.
- **Disconnect reason attempted calls** shows the attempted calls with their associated error codes. This report is sorted by routes and is filtered by Date/Time, Route and Disconnect reason.
- **Trunk Summary** shows the number of calls for a given time period. The report is sorted by trunks and is filtered by Date/Time and Trunk.

4.4.1 Data Source

The reports use views that are made in MS-SQL as a shared data source. The views select all the columns from the original table since all or most of the columns are used in the reports, further selections are done in the data set queries in each report. The first report uses a view made from the CDR_RAW table as seen in Figure 4.2. The three other reports uses views based on the aggregated tables from subsection 4.2.2. However, the view used for the reports *Disconnect reason for stopped calls* and *Disconnect reason for attempted calls* have used a left outer join to connect the rows from the tables *Route_summery* and *ISUP_SIP_MAP* by using the disconnected number. In all the reports, it is essential to display the correct time for the viewers, this is because the reports can be filtered on time, where the input time is the same time zone as in Norway.

4.4.2 Time Zone

The time registered in the CDR files are in the time zone UTC+00.00; however, Norway is in the time zone Central European Time (CET), which means that Norway is in UTC+01.00 during the wintertime and UTC+02.00 during the summertime or Daylight Saving Time (DST). The DST date changes every year in Norway, since the start is on the date of the last Sunday in March, and DST ends on the date of the last Sunday in October. Because of this, the time needed to be changed according to the UTC, coherent with the date from the records. This was solved by using the feature `AT TIME ZONE` in T-SQL, `AT TIME ZONE` uses the time zone from the Windows Registry to connect the name of the chosen time zone to what UTC it is [62]. It also checks if the time zone has a DST or not and, if so when it is changed to DST. The `AT TIME ZONE` can also change the time to the correct time in another time zone by using time and date, the name of the current time zone, and the name of the time zone that the time is going to be changed into as seen in Code listing 4.4. The code in Code listing 4.4 was used to create a view based on the table *trunk_Summery* and makes a new field containing the right time for the time zone in Norway by converting the time using `AT TIME ZONE`. The rest of the code used to create all the views are illustrated in [Appendix F].

```
CREATE VIEW dbo.Trunk_Summary_View
AS
SELECT *, CONVERT(DateTime2(1), (PeriodStart AT TIME ZONE 'UTC' AT TIME ZONE
↪  'Central European Standard Time')) AS RightPeriodStartTime --Selecting every
↪  field and adding a field with time/date in our time zone
FROM [HDO_CDR].[HDO].[Trunk_Summery];
GO
```

Code listing 4.4: How a SQL view is made and AT TIME ZONE used to change the time to right time zone

4.4.3 The Reports

The data sets are used to extract the data from the data source. Each report has an embedded dataset, which can be seen in [Appendix J.3]. This is because the reports use different data, and most of the data are in different views. However, the two reports for Disconnect reasons use the same view, but extracts some different data as seen in section 4.4, therefore they do not use a shared data set. The data sets are written in T-SQL and use a simple `SELECT` to extract the needed data, it also uses the `WHERE` clause to filter data. Dynamic queries are also used so that the parameters and wildcards work as filters as seen in Code listing 4.5.

```
SELECT TOP(1000) GatewayName, AccountingID, Recordtype, RightStartTime,
↪  RightDisconnectDateTime, CallServiceDuration, CallingNumber, CalledNumber,
↪  RouteLabel, IngressTrunkGroup, EgressTrunkGroup,
↪  CallDisconnectReason,CallDisconnectReasonIngress, CallDisconnectReasonEgress,
↪  IngressRemoteSignalingIP, EgressRemoteSignalingIP, RightInsertDateTime
FROM CDR_RAW_View

WHERE ((EgressTrunkGroup IN (@TrunkGroup)) OR (IngressTrunkGroup IN (@TrunkGroup)))
↪  AND  (RouteLabel IN (@RouteParameter)) AND (CallingNumber LIKE
↪  '%'+@CallingNumberParameter+'%') AND  (CalledNumber LIKE
↪  '%'+@CalledNumberParameter+'%') AND (RightStartTime BETWEEN @FromDate AND
↪  @ToDate)
```

Code listing 4.5: The query used as dataset in the Call search report

As seen at the start of section 4.4 all the reports use parameters. The parameters are used as filters; however, the *Call search* report also uses wildcards for two of the parameters. Wildcards are used for the calling and called number, the code for wildcards can be seen in Code listing 4.5 in the `WHERE` clause in the code containing `LIKE`. If a number or a sequence of numbers are written, the query will check if the number contains the number(s). Due to the size of the table CDR_RAW, it also uses `TOP(1000)` in the *Call search* report to narrow down the results, and since it is often unnecessary to show more rows [Appendix A].

The parameters for `Trunk`, `Route`, and `Disconnect reason` utilizes a drop-down list. The drop-down list is set as the available values, and all the values of the drop down list are selected by default. This is to make it easier for the viewer of the

reports to select the different options. The values are collected from a data set that is made for each drop-down list and uses the same view that the main data set uses. However, the data sets that the parameters utilize contain a simple query as in Code listing 4.6 that only extracts distinct values for a parameter. The query ends with a `WHERE` clause that ensures that no empty values were selected.

```
1 SELECT DISTINCT RouteLabel
2 FROM   CDR_RAW_View
3 WHERE  RouteLabel IS NOT NULL
```

Code listing 4.6: Only distinct routenames will be shown in the drop-down list, and if the route is missing, it will not show NULL

The Date/Time and the wildcards do not utilize drop-down lists, but they also contain default values. The default setting for the date interval was set from 7 days ago to now. The default value for the wildcard was set to %, which means show all the numbers.

4.4.4 The Drill Down Reports

The three reports, *Trunk Summary*, *Disconnect reason for stopped calls*, and *Disconnect reason for attempted calls*, uses drill-down action connected to the groups to have a clean and clear view of the channels. If the + button is pressed, it will show more information. Each report is grouped by the channels; *Trunk Summary* that uses the `trunk` as the channel, while the *Disconnect reason for stopped calls* and *Disconnect reason for attempted calls* use the `route` as the channel. The Disconnect reason reports as seen in the Figure 4.8, are further grouped by the column `Disconnect reason` within the route groups. The Disconnect reason groups also have the columns `Enumeration`, `SIP`, and `ISUP` as its child groups. The child groups are directly connected to the channel group, which is also the parent group in the reports.

The groups are sorted by the channels alphabetically, and within routes, it is further sorted by the Disconnect reason alphabetically. The first row of the groups uses the function `SUM` to summarise numbers, like time slot of calls and number of calls; however, the *Trunk Summary* report also uses `MAX` to show the maximum number of simulation calls that have occurred.

Route	Disconnect Reason	Enumeration	SIP	ISUP	Right Period Start Time	Timeslot of calls	Num Calls Connected
[redacted]	⊞ 16	CPC_DISC_NORMAL_CALL_CLEARING	200	16		120	116
	⊟ 31	CPC_DISC_NORMAL_UNSPECIFIED	480	31		90	13
					2021-04-30 12:00:00	30	3
					2021-04-30 12:30:00	30	4
					2021-04-30 13:00:00	30	6
[redacted]	⊞ 16	CPC_DISC_NORMAL_CALL_CLEARING	200	16		120	536

Figure 4.8: A part of the report named *Disconnect reason for stopped calls*

4.4.5 Reporting Services GUI

When a report is deployed, the report can be found in the Reporting Services' GUI under the project folder. The GUI can be found at the address: *http://localhost/reports/*.

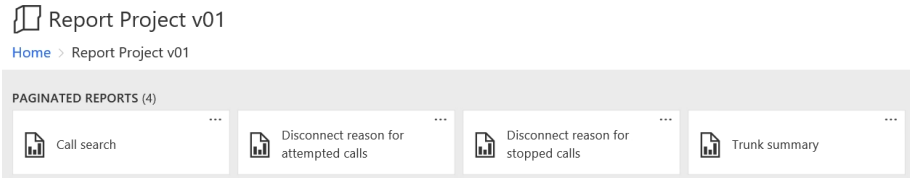


Figure 4.9: Reporting Services' GUI

In the GUI, the reports can be viewed, deleted, and managed. If choosing to manage, a person can change the data source and make changes to the parameters. For the parameters, this means that the parameter can be hidden, get the name changed, and stop using default values or change the default values.

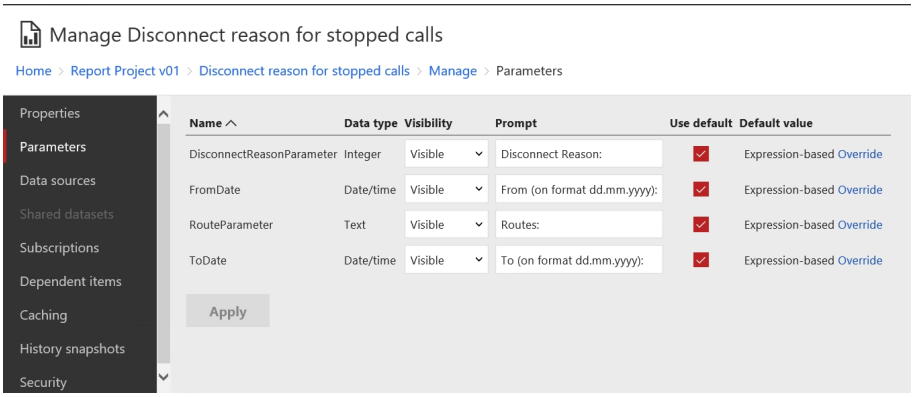


Figure 4.10: GUI for managing the reports

4.5 Key Performance Indicators (KPI)

> 'A performance indicator or key performance indicator (KPI) is a type of performance measurement. KPIs evaluate the success of an organization or of a particular activity (such as projects, programs, products and other initiatives) in which it engages' [63].

Within the telephone- and radio communication platform, there are many interesting KPI's that are valuable to monitor. Each KPI shows valuable information when monitored over certain time periods, in that case, it is important to choose the KPI's that provide the desired information for HDO. Together with HDO, it

was discussed which KPI's should be monitored [Appendix A]. These interactions with the client resulted in four KPI's: Session Establishment Ratio (SER), Session Establishment Effectiveness Ratio (SEER), Session Defects Ratio (SDR), and Network Efficiency Ratio (NER). These four constitute only the first KPI's. On a later occasion, HDO mentioned that they would want to monitor additional KPI's, but it would not happen until in a couple of months or years. That these KPI's were prioritized in the beginning is motivated by the fact that they are standard in the telecommunications industry as they offer important information. For HDO's part, those KPI's are important because they offer information related to network quality, network performance, server failure, call success rate, and information related to route congestion, to name a few. The information these KPI's offer will give HDO the opportunity to see possible trends in their network and which will give them a basis for determining whether their network is behaving in an undesirable state or not.

The KPI's that have been selected to be monitored in Grafana are calculated from specific formulas. The formulas used for SER, SEER, and SDR are defined by the IETF, while the formula for NER is defined by the ITU. Each formula utilizes different variables to be able to calculate a specific KPI. These variables are retrieved from the database and manipulated in Grafana to show the correct KPI metric for a given time period. *CallDisconnectReasonIngress* and *CallDisconnectReasonEgress* are two columns in the database where their value is representing a SIP response code. The SIP response code contains a three-digit integer that explains the status of the request for a call. For instance, the SIP response code 5xx addresses server failure responses while the more specific code 502 addresses bad gateway [64]. SER, SEER, and SDR are the KPI's that utilize SIP response codes as one of the variables for calculating the KPI metric. These three KPI's are going to be monitored on the ingress and the egress side of the SBC trunks, meaning that HDO is able to monitor disconnect reasons for both inbound and outbound calls.

There is also a column named *CallDisconnectReason* in the database where their value is representing an ISUP signaling release cause code. For every call, an ISUP code is generated. That code is similar to a SIP response code in the way that it tells some information on how the call generally went. For instance, ISUP code 1 addresses an unallocated number while code 16 addresses normal call clearing [65]. NER utilizes the ISUP codes as a variable for calculating the KPI metric.

4.5.1 KPI details

SER

SER is a metric that is used to detect the ability of a terminating user agent or downstream proxy to successfully establish sessions per new session INVITE requests [66]. The following formula is used to calculate SER:

$$\text{SER} = \frac{\text{\# of INVITE Request w/ associated 200 OK}}{\text{(Total \# of INVITE Request) - (\# of INVITE Requests w/ 3XX Response)}} * 100$$

Figure 4.11: Formula for calculating SER

SEER

SEER is a metric that is similar to SER, however, SEER excludes user behavior from the metric and therefore focuses on the performance of the network [66]. The following formula is used to calculate SEER:

$$\text{SEER} = \frac{\text{\# of INVITE Request w/ associated 200 OK, 480, 486, 600, or 603}}{\text{(Total \# of INVITE Request) - (\# of INVITE Requests w/ 3XX Response)}} * 100$$

Figure 4.12: Formula for calculating SEER

SDR

SDR is a metric that is used to detect server failure responses within a network [67]. The following formula is used to calculate SDR:

$$\text{SDR} = \frac{\text{\# of INVITE Request w/ associated 500, 503, or 504}}{\text{Total \# of INVITE Request}} * 100$$

Figure 4.13: Formula for calculating SDR

NER

NER is a metric that is designed to measure the ability of a network to deliver a call to the called terminal. NER excludes user behaviour, which means it represents network performance [68]. The following formula is used to calculate NER:

$$\text{NER} = \frac{\text{Answered calls (16) + User busy (17) + Ring no answer (18, 19) + Terminal rejects (21)}}{\text{Total \# of Call Attempts}} * 100$$

Figure 4.14: Formula for calculating NER

4.6 Grafana

As mentioned in section 4.5, the KPI's are going to be visualized in Grafana. Grafana is connected to an MS-SQL data source where it queries data from the CDR_RAW

table. Grafana is configured to visualize one dashboard containing eight panels, these panels visualize the required KPI's. Each panel contains a specific query that retrieves the necessary data in order to display the given KPI. Grafana is executing the queries automatically every 24 hours, which means that new data in the database are ready to be visualized without any manual interactions. The panels are:

- Network Efficiency Ratio (NER) - Overall
- Recordtype
- Session Establishment Ratio (SER) - Ingress
- Session Establishment Ratio (SER) - Egress
- Session Establishment Effectiveness Ratio (SEER) - Ingress
- Session Establishment Effectiveness Ratio (SEER) - Egress
- Session Defects Ratio (SDR) - Ingress
- Session Defects Ratio (SDR) - Egress

The whole dashboard is set to display data in the last 24 hours, this allows the operation team at HDO to monitor KPI's every 5 minutes for deviations within that 24-hour window. Since there can be a lot of calls within the HDO network, the KPI's would show unclear trends if it was addressing larger time intervals such as an hour. Therefore, the interval is set to 5 minutes, which enables one to look at more detailed data and monitor KPI trends within a specific hour. Since the data that is stored in CDR_RAW are stored for 30 days, the dashboard is quite flexible. If necessary, the operations team has the opportunity to change the time range to a week or a month if they need to view older data.

4.6.1 Time Zone

When the CDR files are generated by the SBC, the time zone for each record is set to UTC+00.00. For Norway, that time zone is incorrect. When it is wintertime, the time zone must be UTC+01.00. When it is summertime, the time zone must be UTC+02.00. These changes has to be done manually in the Grafana dashboard, the process for it is very simple. See [Appendix K.3].

4.6.2 Configuration

The Code listing 4.7 visualizes the query used to achieve the panel displayed in Figure 4.15. The query results in a graph visualizing NER and the total amount of calls within a 5 minute interval. As mentioned earlier, the data source from which Grafana retrieves data is an MS-SQL, which means that all the queries are written in SQL. The query uses basic SQL statements such as `SELECT` and `WHERE`. It contains macros such as *$__timeGroup* and *$__timeFilter*, these macros lead to a graph where the column *StartDateTime* is grouped by every 5-minutes to visualize data within that time interval. The main piece of the query consists of different variables which are retrieved from the database, those variables are used in a formula to visualize the KPI. For instance, Code listing 4.7 uses the formula in Figure 4.14

to calculate NER. In addition, advanced functions such as `CAST`, numeric functions such as `SUM`, and conditional statements such as `CASE` are used within the formula to achieve the correct KPI metric.

```
SELECT
  $__timeGroup(StartDateTime, '5m', 0) AS time,
100 * ((sum(case when CallDisconnectReason = 16 then 1 else 0 end) +
        sum(case when CallDisconnectReason = 17 then 1 else 0 end) +
        sum(case when CallDisconnectReason = 18 then 1 else 0 end) +
        sum(case when CallDisconnectReason = 19 then 1 else 0 end) +
        sum(case when CallDisconnectReason = 21 then 1 else 0 end))
        / ((CAST((count(Recordtype)) as FLOAT)))) as NER,
        (count(Recordtype)) as Calls
FROM HDO.CDR_RAW
WHERE $__timeFilter(StartDateTime)
GROUP BY $__timeGroup(StartDateTime, '5m')
ORDER BY 1
```

Code listing 4.7: Query for NER, results in the panel in Figure 4.15

When HDO's operation team is monitoring the dashboard, specific data in the panels determine if HDO's network is in the desired state or not. For instance, in Figure 4.15 that visualizes NER, it is desirable that NER is one hundred percent. This is desired since that means all the calls within that 5 minute time interval ended with a preferred disconnect reason. For HDO, it is desirable and more realistic that NER is as close to one hundred percent as possible. An undesirable situation will be that NER is close to zero percent. The closer NER is zero percent, the more calls in that 5-minute time interval ended with an unpreferred disconnect reason. It is also common that NER is exactly zero percent, which means that within a 5 minute time interval, there were no calls. What has been discussed does not only apply to NER but also to all the other KPI's except SDR. For SDR, it is desirable that the metric is as close to zero percent as possible instead of one hundred percent. In [Appendix K], the configuration and the resulting panel for the other KPI's are illustrated.

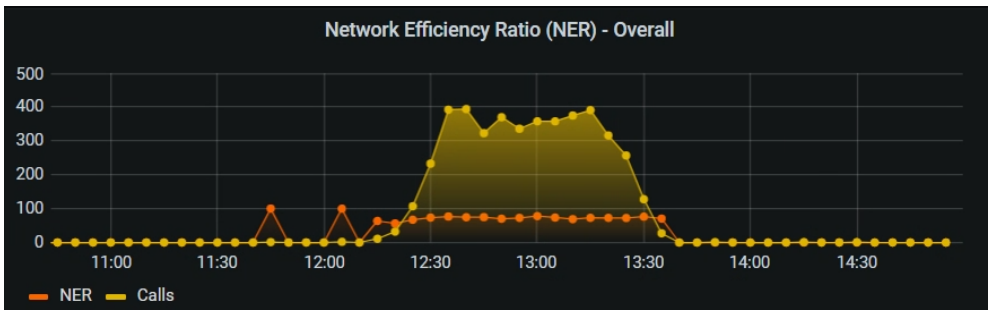


Figure 4.15: Panel displaying a graph that visualizes NER

The Code listing 4.8 visualizes the query used to achieve the panel in Figure 4.16.

The query results in a panel that visualizes total calls (total records), number of STOP records, and number of ATTEMPT records. The query uses a variable named *$number_days*. This makes the panel more dynamic, one can then visualize total calls (total records), the number of STOP records, and the number of ATTEMPT records within the last 1, 2, 3, 7, 14, 21, or 30 days. For HDO, it is interesting to monitor the number of calls that occurred in different time intervals, and by how many of those calls were either STOP- or ATTEMPT records. This gives an indication of how many calls were established successfully and how many calls were not successfully established. See subsection 3.3.1 for more information regarding the STOP- and ATTEMPT records.

```
SELECT count(Recordtype) as Total_Calls,
sum(case when Recordtype = 'STOP' then 1 else 0 end) AS STOP,
sum(case when Recordtype = 'ATTEMPT' then 1 else 0 end) AS ATTEMPT
FROM HDO.CDR_RAW
WHERE StartDateTime >= DATEADD(DAY, $number_days, GETDATE())
AND StartDateTime <= GETDATE()
```

Code listing 4.8: Query for the Recordtype panel, results in a panel with the gauges displayed in Figure 4.16

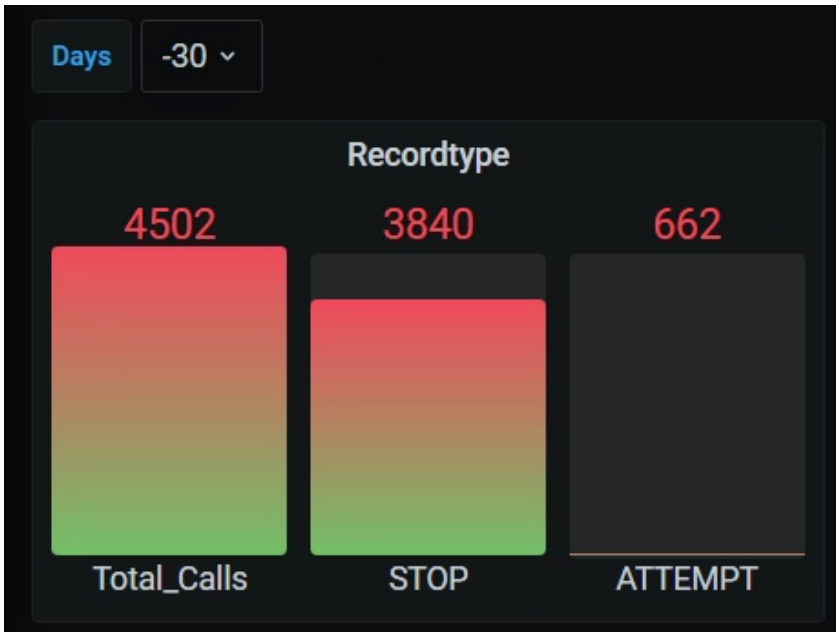


Figure 4.16: Panel with gauges that visualize different Recordtypes the last 30 days

Chapter 5

Discussion and Conclusion

5.1 Discussion

This research and development project aimed to investigate and produce an automatic monitoring workflow for an emergency telephone network capable of CDR parsing, database management, reporting and visualization of crucial information.

5.1.1 Discussion regarding the Final Result

Initially, while looking through existing material, it became clear that numerous different approaches could be used to automate the injection and parsing of data from a CDR file. With the awareness that our database was required to be built as a MS-SQL supporting our agreement with HDO, we narrowed down our selection by considering centralized administration for large portions of the solution. Therefore, we decided to use SSIS to build our parsing package. SSIS packages can be easily connected to an Microsoft SQL Server instance, and it allows for seamless lookup and alteration to existing database records explained in section 4.3. Additionally, SSIS packages can be scheduled to be executed on a given interval directly in Microsoft SQL Server Management Studio via SQL Server Agents.

The fundamental database table is constructed as a single table in 1NF. This table is used as an ingestion point for SSIS packages and as a source for underlying aggregation procedures, discussed in subsection 4.2.2. The decision to adopt a single 1NF table instead of additional segmentation was weighed against the fact that stored procedures aggregate essential information daily, normalizing the data from personally identifiable data. The ingest table is then cleansed from the database once a record strikes 30 days. Following HDO's wishes, the deletion is automatically handled by the SSIS package. A benefit of storing the records for a short period after the aggregation process is that it gives the engineer the possibility to lookup an entire record obtained from the SBC in full when troubleshooting, while still respecting the concept of not collecting private data longer than required for the system.

With the ingest and storing procedure resolved, the subsequent research prob-

lems were addressed: Report generation with SSRS and KPI visualized using Grafana. HDO requested four reports within SSRS discussed at length in section 4.4; three of them focused on summarizing data from trunks, routes, and disconnect reasons, while the last report should be able to extract data directly from the raw table. All of the reports were built with embedded data sets capable of requesting filtered data from their data sources. For the first three reports, this source is their corresponding tables generated during the data aggregation process, while the last report queries data directly from the main table. The filtering operation is managed through an arrangement of menu selections. Based on the selected filters, a unique query is constructed.

The different filtering options are specifically optimized for each of the reports. Filtering could be cumbersome to construct so that they are compliant with the underlying data structure. However, when designing our database, we decided to utilize stored procedures for daily data aggregation. This process results in three aggregated tables that closely resemble the final reports, meaning that SSRS does not need to handle any of the calculations, such as finding the maximum amount of simultaneous calls between 8:00 and 8:30. Instead, SSIS can filter out pre-calculated rows and columns as a simplistic SQL `select` statement with groups and `where` clauses. It would have been possible to do all of these computations from SSRS when requesting the data. However, this would have increased the complexity of this module as well as loss in performance. When choosing the different filtering vectors, we spoke with an HDO representative to understand typical queries the engineer usually probes while troubleshooting. Additionally, we discussed database response constraints to mitigate bandwidth deficiencies and decided to set a hard limit of 1000 rows for the report that operates directly on the raw data.

For continuous monitoring of the telephone network's health status, Grafana has been configured to detail four KPIs for both ingress- and egress-traffic within a single dashboard. This dashboard will primarily be displayed within the HDOs monitoring center. Each of the KPIs: SER, SEER, SDR, and NER was selected based on information communicated during meetings with HDO. The formulas for these KPIs can be seen in subsection 4.5.1. Following the KPI selection, we validated that parsed data ingested to the database from SSIS that the currently parsed columns were sufficient for calculating the KPIs.

5.1.2 Discussion of our Methodology

In order to collect information about an SBC network workflow for monitoring, both of our methodologies became useful. As far as our research went, we did not come across a single source that replicated our project's distinct requirements. Instead, we needed to stitch together information from several sources.

Researching information on the Internet became our primary source for finding commonly used best practices within all of our subtasks, such as Microsoft SQL Server configuration, database architectures, and visualization vectors, used today.

This decision was made because the development of technological products happens rapidly, and the Internet generally provides the most updated information. At the same time, it can be hard to judge what information is actionable or not. That is why we tried to make sure we took information from trusted websites within the DevOps field and searched for information from several sources such as vendor documentation to strengthen the results. However, for establishing client-specific details, the interviewee helped to extend our perception of the overall product.

The ongoing interview with HDO gave us in-depth insights into how professional operation engineers handle troubleshooting and how historical data could be utilized to prioritize error handling and hardware upgrades. However, it could have been interesting to conduct more interviews, particularly interviewing people with different working titles such as database architects, operation engineers, and data engineers. Conducting these types of interviewers could have brought our understanding of the overall system and helped optimize subsystems with their knowledge within a given field. Regrettably, we did not manage to conduct these interviews within our time limit. Moreover, we were not sure how to approach outside interviewees fittingly regarding our confidentiality agreements with HDO.

Finally, it is important to state once again that technology is a constantly evolving field. During our research period, were there minor updates made to the tools and technologies researched and developed. By the time this paper is published, there might be new and refined best practices or tools that could have been utilized. Nevertheless, this paper stands as documentation of a viable development solution as of May 2021.

5.1.3 Complications

During the development period, some issues arose regarding the development environment provided by HDO. The issue concerned that we were not able to connect to the remote development environment from our local machines. These events occurred a couple of times due to server downtime. This was not a significant issue because the development process was shifted by only a few hours to a maximum of one day, which was manageable but undesirable. In addition, some of us established a local development environment to not delay the development process.

In addition, during the beginning of the development process, an issue arose around parsing CDR files in a Microsoft environment. The issue became time-consuming since we used much time looking for scripts for parsing the CDR files when we should have investigated whether Microsoft had a solution for parsing files. The issue arose due to not possessing enough knowledge regarding parsing CDR files at this point in the process and by not considering investigating solutions especially from Microsoft, although most of the other technologies utilized in this project were Microsoft technologies.

5.2 Conclusion

As new technologies are systematically introduced around us, it is essential to adapt currently implemented systems to provide more fulfilling products. Within an emergency phone SBC network, a tremendous amount of data flows through it daily. It is crucial to understand how this data could be captured and used to help establish an idea about how the network is operating.

Capturing and processing data to represent Key Performance Indicator and generating reports of call activities requires a complete pipeline consisting of data deconstruction, storing mechanic, and a visualization aspect. By looking into CDR, a package generation within an SBC, it becomes clear that lots of information are collected during a telephone call.

This research and development project utilizes several different Microsoft enterprise tools and Grafana to construct an automated workflow for handling and monitoring results from a CDR record. These CDR records can be dissected and sent to a SQL server for storage with the help of SSIS. T-SQL and stored procedures facilitate further processing of stored records by aggregating data into pre-calculated tables for key factors within the network. Visualizing the data from within the SQL database allows an engineer to promptly regulate a shifting environment by monitoring KPIs within Grafana calculated upon the stored data. SSRS reports can be used as support to realize funding for system upgrades based on historical call patterns data. The force that shimmers through the entire project comes back to collecting and sorting out the correct information into a normalized format available for further processing.

5.2.1 Further work

Further research into monitoring an SBC emergency network and other highly critical telephone networks should focus on collecting data directly from the SBC source, reducing the latency between requested and visualized results. This could be the next project where they analyze data from their REST-API. This gives them the opportunity to visualize real-time data, which means that they can discover and handle deviations much faster on their telephone network compared to the solution we developed where the data being visualized is 5 minutes delayed from the SBC. By analyzing data from the REST-API, opportunities such as visualizing additional KPI's that are better suited for real-time becomes an option. In addition, visualizing KPI's with SSRS reports is an interesting option that offers another surface for analytical monitoring for historical data in comparison with Grafana. Furthermore, while this implementation utilizes advanced techniques with temporary and derived tables when aggregating data, further studies are needed to gain more insight into how well the system performs under heavy load (100,000+ records a day).

Bibliography

0

[1] HDO, *Om oss*, no, Jan. 2021. [Online]. Available: https://www.hdo.no/om-oss (visited on 27/01/2021).

[2] HDO, *Kommunikasjon i Akuttmedisinsk Kjede (KAK)*, no, Nov. 2020. [Online]. Available: https://www.hdo.no/kak#veien-til-kak (visited on 19/05/2021).

[3] DSB, *DSB og Nkom publiserer felles notat om neste generasjon nødnett*, no, Oct. 2017. [Online]. Available: https://www.dsb.no/nyhetsarkiv/2017/dsb-og-nkom-publiserer-felles-notat-om-neste-generasjon-nodnett/ (visited on 03/02/2021).

[4] HDO, *Ny kommunikasjonsløsning til akuttkjeden*, no, Dec. 2020. [Online]. Available: https://www.hdo.no/nyheter/ny-kommunikasjonslosning-til-akuttkjeden (visited on 27/01/2021).

[5] *Stig Atle Hauge*. [Online]. Available: https://no.linkedin.com/in/stigatle (visited on 23/01/2021).

[6] *Christian Johansen*. [Online]. Available: https://www.ntnu.no/ansatte/christian.johansen (visited on 20/01/2021).

[7] A. Bryman, 'Social research methods,' in, 4th ed. Oxford University Press, 2012, pp. 36, 310–316, 470–480.

[8] C. Pang and A. Hindle, 'Continuous maintenance,' in *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2016, pp. 458–462. DOI: 10.1109/ICSME.2016.45.

[9] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, E. Schooler, *SIP: Session Initiation Protocol*, en, Jun. 2002. [Online]. Available: https://tools.ietf.org/html/rfc3261 (visited on 25/04/2021).

[10] Tutorialspoint, *Session Initiation Protocol - Introduction*, en. [Online]. Available: https://www.tutorialspoint.com/session_initiation_protocol/session_initiation_protocol_introduction.htm (visited on 06/05/2021).

[11] Wikipedia contributors, *Client–server model*, en, Apr. 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Client%5C%E2%5C%80%5C%93server_model&oldid=1018410018 (visited on 06/05/2021).

[12] Tutorialspoint, *SIP - Network Elements*, en. [Online]. Available: https://www.tutorialspoint.com/session_initiation_protocol/session_initiation_protocol_network_elements.htm (visited on 06/05/2021).

[13] Tutorialspoint, *SIP - B2BUA*, en. [Online]. Available: https://www.tutorialspoint.com/session_initiation_protocol/session_initiation_protocol_b2bua.htm (visited on 06/05/2021).

[14] Ribbon Communications, *SIP to ISUP Disconnect Code Mapping*, en, Aug. 2020. [Online]. Available: https://support.sonus.net/pages/viewpage.action?pageId=232916126 (visited on 06/05/2021).

[15] Ribbon Communications, *What is a Session Border Controller (SBC)?* en. [Online]. Available: https://ribboncommunications.com/company/get-help/glossary/session-border-controller-sbc (visited on 15/04/2021).

[16] Wikipedia contributors, *Session border controller*, en, 2019. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Session_border_controller&oldid=932244954 (visited on 09/04/2021).

[17] Ribbon Communications, *Peering (NNI) Scenarios*, en, Aug. 2020. [Online]. Available: https://support.sonus.net/pages/viewpage.action?pageId=232920281 (visited on 08/04/2021).

[18] Ribbon Communications, *Access (UNI) Scenarios*, en, Aug. 2020. [Online]. Available: https://support.sonus.net/pages/viewpage.action?pageId=232920362 (visited on 08/04/2021).

[19] Ribbon Communications, *SBC Module Descriptions*, en, Aug. 2020. [Online]. Available: https://support.sonus.net/pages/viewpage.action?pageId=232921077 (visited on 08/04/2021).

[20] Ribbon Communications, *Billing Introduction*, en, Aug. 2020. [Online]. Available: https://support.sonus.net/pages/viewpage.action?pageId=232916120 (visited on 08/04/2021).

[21] Ribbon Communications, *Supported Standards*, en, Aug. 2020. [Online]. Available: https://support.sonus.net/pages/viewpage.action?pageId=232921080 (visited on 08/04/2021).

[22] Wikipedia contributors, *Call detail record*, en, Mar. 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Call_detail_record&oldid=1011853404 (visited on 14/03/2021).

[23] Ribbon Communications, *CDR Field Descriptions*, en, Sep. 2020. [Online]. Available: https://support.sonus.net/pages/viewpage.action?pageId=232916123 (visited on 14/03/2021).

[24] Ribbon, *About Us*, en. [Online]. Available: https://ribboncommunications.com/about-us (visited on 14/03/2021).

[25] Ribbon Communications, *CDR Field Descriptions*, en, Sep. 2020. [Online]. Available: https://support.sonus.net/pages/viewpage.action?pageId=232916123#CDRFieldDescriptions-start (visited on 09/03/2021).

[26] Ribbon Communications, *CDR Field Descriptions*, en, Sep. 2020. [Online]. Available: https://support.sonus.net/pages/viewpage.action?pageId=232916123#CDRFieldDescriptions-stop (visited on 09/04/2021).

[27] Ribbon Communications, *CDR Field Descriptions*, en, Sep. 2020. [Online]. Available: https://support.sonus.net/pages/viewpage.action?pageId=232916123#CDRFieldDescriptions-attempt (visited on 09/04/2021).

[28] Ribbon Communications, *CDR Field Descriptions*, en, Sep. 2020. [Online]. Available: https://support.sonus.net/pages/viewpage.action?pageId=232916123#CDRFieldDescriptions-intermediate (visited on 09/04/2021).

[29] Wikipedia contributors, *Database normalization*, en, Mar. 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Database_normalization&oldid=1014649450 (visited on 08/04/2021).

[30] Wikipedia contributors, *Transact-sql*, en, Aug. 2020. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Transact-SQL&oldid=973213386 (visited on 09/04/2021).

[31] Dorota Wdzięczna, *What's the Difference Between SQL and T-SQL?* en-us, Feb. 2019. [Online]. Available: https://learnsql.com/blog/t-sql-vs-standard-sql-whats-the-difference/ (visited on 09/04/2021).

[32] Pinal Dave, *SQL SERVER – Database Coding Standards and Guidelines – Part 1*, en, Jun. 2007. [Online]. Available: https://blog.sqlauthority.com/2007/06/04/sql-server-database-coding-standards-and-guidelines-part-1/ (visited on 28/01/2021).

[33] Pinal Dave, *SQL SERVER – Database Coding Standards and Guidelines – Part 2*, en, Jun. 2007. [Online]. Available: https://blog.sqlauthority.com/2007/06/05/sql-server-database-coding-standards-and-guidelines-part-2/ (visited on 28/01/2021).

[34] *Pinal Dave*. [Online]. Available: https://www.pluralsight.com/authors/pinal-dave (visited on 28/01/2021).

[35] Microsoft, *What is SQL Server Management Studio (SSMS)?* en, Sep. 2019. [Online]. Available: https://docs.microsoft.com/en-us/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver15 (visited on 14/03/2021).

[36] Microsoft, *Download SQL Server Data Tools (SSDT) for Visual Studio*, en, Feb. 2020. [Online]. Available: https://docs.microsoft.com/en-us/sql/ssdt/download-sql-server-data-tools-ssdt?view=sql-server-ver15 (visited on 26/03/2021).

[37] Wikipedia contributors, *Sql server integration services*, en, Feb. 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=SQL_Server_Integration_Services&oldid=1004134623 (visited on 03/05/2021).

[38] Microsoft, *SQL Server Integration Services*, en, Jul. 2018. [Online]. Available: https://docs.microsoft.com/en-us/sql/integration-services/sql-server-integration-services?view=sql-server-ver15 (visited on 27/03/2021).

[39] Microsoft, *Integration Services (SSIS) Packages*, en, Aug. 2016. [Online]. Available: https://docs.microsoft.com/en-us/sql/integration-services/integration-services-ssis-packages?view=sql-server-ver15 (visited on 27/03/2021).

[40] Microsoft, *SQL Server Agent Jobs for Packages*, en, Jun. 2020. [Online]. Available: https://docs.microsoft.com/en-us/sql/integration-services/packages/sql-server-agent-jobs-for-packages?view=sql-server-ver15 (visited on 02/04/2021).

[41] Microsoft, *Data Flow*, en, Mar. 2017. [Online]. Available: https://docs.microsoft.com/en-us/sql/integration-services/data-flow/data-flow?view=sql-server-ver15 (visited on 27/03/2021).

[42] Tutorial Gateway, *Drill Down Reports in SSRS*, en. [Online]. Available: https://www.tutorialgateway.org/ssrs-drill-down-report/ (visited on 06/05/2021).

[43] Esat Erkec, *How to add parameters to SSRS mobile reports*, en, Oct. 2018. [Online]. Available: https://www.sqlshack.com/how-to-add-parameters-to-ssrs-mobile-reports/ (visited on 01/05/2021).

[44] Microsoft, *Create data connection strings - Report Builder & SSRS*, en, May 2020. [Online]. Available: https://docs.microsoft.com/en-us/sql/reporting-services/report-data/data-connections-data-sources-and-connection-strings-report-builder-and-ssrs?view=sql-server-ver15 (visited on 01/05/2021).

[45] Kathi Kellenberger, *Reporting Services Basics: Understanding Data Sources and Datasets*, en, Jul. 2019. [Online]. Available: https://www.red-gate.com/simple-talk/sql/bi/reporting-services-basics-understanding-data-sources-and-datasets/ (visited on 01/05/2021).

[46] Microsoft, *Tutorial: Add a Parameter to Your Report (Report Builder)*, en, Mar. 2017. [Online]. Available: https://docs.microsoft.com/en-us/sql/reporting-services/tutorial-add-a-parameter-to-your-report-report-builder?view=sql-server-ver15 (visited on 01/05/2021).

[47] Wikipedia contributors, *Ssh file transfer protocol*, en, Feb. 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=SSH_File_Transfer_Protocol&oldid=1007330433 (visited on 03/05/2021).

[48] wpadminsrt, *What is SFTP?* en, Nov. 2018. [Online]. Available: https://titanftp.com/2018/11/09/what-is-sftp/ (visited on 10/04/2021).

[49] Wikipedia contributors, *Grafana*, en, Apr. 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Grafana&oldid=1020321845 (visited on 03/05/2021).

[50] Grafana, *Grafana documentation*, en. [Online]. Available: https://grafana.com/docs/grafana/latest/ (visited on 18/04/2021).

[51] Grafana, *Using MySQL in Grafana*, en. [Online]. Available: https://grafana.com/docs/grafana/latest/datasources/mysql/#macros (visited on 18/04/2021).

[52] *Normen – Norm for informasjonssikkerhet og personvern i helse- og omsorgssektoren*, no. [Online]. Available: https://ehelse.no/normen/normen-for-informasjonssikkerhet-og-personvern-i-helse-og-omsorgssektoren (visited on 05/05/2021).

[53] Vincent La, *csv-parser*, en, Jan. 2021. [Online]. Available: https://github.com/vincentlaucsb/csv-parser (visited on 14/03/2021).

[54] Vincent La, *Vincent La*, en. [Online]. Available: https://www.linkedin.com/in/vincent-la-sb (visited on 14/03/2021).

[55] Python, *cpython*, en. [Online]. Available: https://github.com/python/cpython/blob/3.9/Lib/csv.py (visited on 14/03/2021).

[56] Python, *csv — CSV File Reading and Writing*, en. [Online]. Available: https://docs.python.org/3/library/csv.html (visited on 14/03/2021).

[57] *sftpgo*. [Online]. Available: https://github.com/drakkan/sftpgo (visited on 09/03/2021).

[58] *Rebex Buru SFTP Server*, en. [Online]. Available: https://www.rebex.net/buru-sftp-server/ (visited on 26/04/2021).

[59] J. Paredaens, P. D. Bra, M. Gyssens and D. van Gucht, *The Structure of the Relational Database Model*. Springer Publishing Company, Incorporated, 2013, ISBN: 3642699588.

[60] *Personopplysninger*, no. [Online]. Available: https://www.datatilsynet.no/rettigheter-og-plikter/personopplysninger/ (visited on 03/05/2021).

[61] *Art. 5 GDPR – Principles relating to processing of personal data*, en-US. [Online]. Available: https://gdpr-info.eu/art-5-gdpr/ (visited on 03/05/2021).

[62] Microsoft, *AT TIME ZONE (Transact-SQL)*, en, Jun. 2019. [Online]. Available: https://docs.microsoft.com/en-us/sql/t-sql/queries/at-time-zone-transact-sql?view=sql-server-ver15 (visited on 07/05/2021).

[63] Wikipedia contributors, *Performance indicator*, en, Apr. 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Performance_indicator&oldid=1019952744 (visited on 06/05/2021).

[64] Wikipedia contributors, *List of sip response codes*, en, Apr. 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=List_of_SIP_response_codes&oldid=1019749911 (visited on 03/05/2021).

[65] Wikipedia contributors, *Isdn user part*, en, Jan. 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=ISDN_User_Part&oldid=1003529155 (visited on 03/05/2021).

[66] D. Malas, A. Morton, *Basic Telephony SIP End-to-End Performance Metrics*, en, Jan. 2011. [Online]. Available: https://tools.ietf.org/html/rfc6076#section-4.7 (visited on 25/04/2021).

[67] D. Malas, A. Morton, *SIP End-to-End Performance Metrics draft-ietf-pmol-sip-perf-metrics-04*, en, Sep. 2009. [Online]. Available: https://tools.ietf.org/id/draft-ietf-pmol-sip-perf-metrics-04.html#anchor21 (visited on 25/04/2021).

[68] Wikipedia contributors, *Network effectiveness ratio*, en, Jul. 2019. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Network_Effectiveness_Ratio&oldid=907955056 (visited on 03/05/2021).

[69] Steve Danner, *Delete files older than 3 months old in a directory using .net*, en. [Online]. Available: https://stackoverflow.com/questions/2222348/delete-files-older-than-3-months-old-in-a-directory-using-net (visited on 25/03/2021).

Appendix A

Summary of interviews with HDO, Stig Atle

The following question and answers are based on notes from our interviews with Stig Atle. This summary is based on report-relevant statements from our sessions, both from messaging and meetings. Stig Atle has gone through the composed questions to verify the soundness of the arguments.

Does SIP have a specific standard that we should follow?
No, not precisely; SIP follows RFC3261, but this is a very generic implementation. Usually, companies that work within a SIP environment utilize this standard to simply pick out the relevant parts related to the companies business purpose.

What is the main reasoning behind building the new solution utilizing SBCs?
An SBC allows for further integration with new technology such as 5G and could potentially lead to additional integrations with commonly used applications and communication methods in the future. Moreover, SBC is a way to hide our inner network topology and secures the network against incoming traffic with the help of the B2BUA protocol.

Whom or what is in charge of routing the traffic?
Since this network is built up with an SBC, the SBC controls the traffic. Incoming calls could redirect calls into another route, which could occur based on the error coded or an additional set of criteria such as playing music while on hold.

When you say that the SBC secures the network, do you mean that it acts like a firewall? If this is the case, how is this archived on a technical level?
Yes, the SBC can utilize policies on how to act based on the package. For example, the SBC can inspect, monitor traffic patterns, and redirect the traffic based on the results.

The CDR contains four different record types; Would you like us to store all of them?
No, for our purpose, ATTEMPT and STOP records are the only relevant ones.

We are looking into the database schematic and are thinking about segmenting the data. Do you have any concerns or ideas about this?
It would be best to keep the RAW table in the same format as the CDR file, meaning a single table contains all of the data from one call session. By maintaining all relevant information about a case in one table, troubleshooting a specific call is simplified. Additionally, we will not do many direct queries on the RAW data since we are aggregating key information.

For how long should we store data within the RAW table (data containing personal information)?
Thirty days would be a good mark.

What data should be aggregated within the database?
I think that three tables should be sufficient: Trunk_Summary, Route_Summary, and Discause_Reason. All of the tables should store records grouped over 30 minutes and be ran once a day. Trunk summary should contain, Startdate, period length, the number of calls started within a period, trunk-/ingress/egress unioned, the maximum amount of simultaneous calls (consider calls over periods). Route summary should include, Startdate, period, route, disconnection reason, number of calls, number of calls stop, number of calls attempted. Discause reason will be built similarly, start date, period, disconnection reason, the total number of calls,

Are there any bandwidth limitations that we need to consider?
It would typically be fine. First, however, let us limit the number of returns for the call search report to the first 1000 records.

Do you have any suggestions on how we should approach the parsing of CDR packages?
We have previously used SSIS for this task; you could take a look into that.

Is the Update DB task in the SSIS data flow necessary?
It can be good to have, it is not really necessary as this project does not address start records in the parsing process. Either way, you may want to use that task just in case.

Do you have any recommendations regarding which KPI's we need to monitor?
To begin with, we recommend that you monitor Network efficiency Ratio (NER), Session Establishment Ratio (SER), Session Establishment Effectiveness Ratio (SEER), and Session Defects Ratio (SDR). These KPI' are a good starting point and will offer a lot of important information about our

network when displayed in Grafana. It should be said that we want to monitor several KPI's, however, which it will be will not be decided until in a few months to a couple of years. This is because the new communication solution must have been in use long enough before we know what other data we want to monitor.

What reports from SSRS do you see as beneficial for HDO?

Call search on the main table for troubleshooting. Summary reports for trunk traffic and a summary of route reports for both of the record types; stop and attempt records with associated disconnection codes.

In the call search report, which columns are necessary for the report?

Most of the columns are necessary, however, the columns AccountingID and InsertDateTime is not. In this report, the GatewayName is not needed as a parameter, and there should be only one trunk parameter for both IngressTrunkGroup and EgressTrunkGroup.

For the parameters, should the parameters have enabled NULL value or should the parameters have default values?

All of the parameters should have default values. The Routes, Trunks and Disconnect Reasons should also have a list of available values, where all of the values are chosen as default values. While the number parameters should be wildcards, and I recommend it to have % as a default value.

Appendix B

Environment configuration

B.1 SQL Server

```
;SQL Server 2019 Configuration File
[OPTIONS]

; By specifying this parameter and accepting Microsoft Python Open and
    Microsoft Python Server terms, you acknowledge that you have read
    and understood the terms of use.

IACCEPTPYTHONLICENSETERMS="False"

; Specifies a Setup work flow, like INSTALL, UNINSTALL, or UPGRADE.
    This is a required parameter.

ACTION="Install"

; By specifying this parameter and accepting Microsoft R Open and
    Microsoft R Server terms, you acknowledge that you have read and
    understood the terms of use.

IACCEPTROPENLICENSETERMS="False"

; Specifies that SQL Server Setup should not display the privacy
    statement when ran from the command line.

SUPPRESSPRIVACYSTATEMENTNOTICE="False"

; Use the /ENU parameter to install the English version of SQL Server
    on your localized Windows operating system.

ENU="True"

; Setup will not display any user interface.
```

```
QUIET="False"

; Setup will display progress only, without any user interaction.

QUIETSIMPLE="False"

; Parameter that controls the user interface behavior. Valid values are
     Normal for the full UI,AutoAdvance for a simplied UI, and
    EnableUIOnServerCore for bypassing Server Core setup GUI block.

UIMODE="Normal"

; Specify whether SQL Server Setup should discover and include product
    updates. The valid values are True and False or 1 and 0. By default
     SQL Server Setup will include updates that are found.

UpdateEnabled="True"

; If this parameter is provided, then this computer will use Microsoft
    Update to check for updates.

USEMICROSOFTUPDATE="False"

; Specifies that SQL Server Setup should not display the paid edition
    notice when ran from the command line.

SUPPRESSPAIDEDITIONNOTICE="False"

; Specify the location where SQL Server Setup will obtain product
    updates. The valid values are "MU" to search Microsoft Update, a
    valid folder path, a relative path such as .\MyUpdates or a UNC
    share. By default SQL Server Setup will search Microsoft Update or
    a Windows Update service through the Window Server Update Services.

UpdateSource="MU"

; Specifies features to install, uninstall, or upgrade. The list of top-
    level features include SQL, AS, IS (Integration Services components)
    , MDS, and Tools. The SQL feature will install the Database Engine,
     Replication, Full-Text, and Data Quality Services (DQS) server.
    The Tools feature will install shared components.

FEATURES=SQLENGINE,IS

; Displays the command line parameters usage.

HELP="False"

; Specifies that the detailed Setup log should be piped to the console.
```

```
INDICATEPROGRESS="False"

; Specifies that Setup should install into WOW64. This command line
    argument is not supported on an IA64 or a 32-bit system.

X86="False"

; Specify a default or named instance. MSSQLSERVER is the default
    instance for non-Express editions and SQLExpress for Express
    editions. This parameter is required when installing the SQL Server
     Database Engine (SQL), or Analysis Services (AS).

INSTANCENAME="MSSQLSERVER"

; Specify the root installation directory for shared components. This
    directory remains unchanged after shared components are already
    installed.

INSTALLSHAREDDIR="C:\Program Files\Microsoft SQL Server"

; Specify the root installation directory for the WOW64 shared
    components. This directory remains unchanged after WOW64 shared
    components are already installed.

INSTALLSHAREDWOWDIR="C:\Program Files (x86)\Microsoft SQL Server"

; Specify the Instance ID for the SQL Server features you have
    specified. SQL Server directory structure, registry structure, and
    service names will incorporate the instance ID of the SQL Server
    instance.

INSTANCEID="MSSQLSERVER"

; Startup type for Integration Services.

ISSVCSTARTUPTYPE="Automatic"

; Account for Integration Services: Domain\User or system account.

ISSVCACCOUNT="REDACTED, SQL Agent"

; Account for SQL Server CEIP service: Domain\User or system account.

SQLTELSVCACCT="NT Service\SQLTELEMETRY"

; Startup type for the SQL Server CEIP service.

SQLTELSVCSTARTUPTYPE="Automatic"
```

```
; Specify the installation directory.

INSTANCEDIR="C:\Program Files\Microsoft SQL Server"

; Agent account name

AGTSVCACCOUNT="REDACTED, SQL Agent"

; Auto-start service after installation.

AGTSVCSTARTUPTYPE="Automatic"

; CM brick TCP communication port

COMMFABRICPORT="0"

; How matrix will use private networks

COMMFABRICNETWORKLEVEL="0"

; How inter brick communication will be protected

COMMFABRICENCRYPTION="0"

; TCP port used by the CM brick

MATRIXCMBRICKCOMMPORT="0"

; Startup type for the SQL Server service.

SQLSVCSTARTUPTYPE="Automatic"

; Level to enable FILESTREAM feature at (0, 1, 2 or 3).

FILESTREAMLEVEL="0"

; The max degree of parallelism (MAXDOP) server configuration option.

SQLMAXDOP="2"

; Set to "1" to enable RANU for SQL Server Express.

ENABLERANU="False"

; Specifies a Windows collation or an SQL collation to use for the
    Database Engine.

SQLCOLLATION="Danish_Norwegian_CI_AS"

; Account for SQL Server service: Domain\User or system account.
```

```
SQLSVCACCOUNT="REDACTED: sql_engine"

; Set to "True" to enable instant file initialization for SQL Server
    service. If enabled, Setup will grant Perform Volume Maintenance
    Task privilege to the Database Engine Service SID. This may lead to
     information disclosure as it could allow deleted content to be
    accessed by an unauthorized principal.

SQLSVCINSTANTFILEINIT="True"

; Windows account(s) to provision as SQL Server system administrators.

SQLSYSADMINACCOUNTS="REDACTED"

; The default is Windows Authentication. Use "SQL" for Mixed Mode
    Authentication.

SECURITYMODE="SQL"

; The number of Database Engine TempDB files.

SQLTEMPDBFILECOUNT="4"

; Specifies the initial size of a Database Engine TempDB data file in
    MB.

SQLTEMPDBFILESIZE="8"

; Specifies the automatic growth increment of each Database Engine
    TempDB data file in MB.

SQLTEMPDBFILEGROWTH="64"

; Specifies the initial size of the Database Engine TempDB log file in
    MB.

SQLTEMPDBLOGFILESIZE="8"

; Specifies the automatic growth increment of the Database Engine
    TempDB log file in MB.

SQLTEMPDBLOGFILEGROWTH="64"

; The Database Engine root data directory.

INSTALLSQLDATADIR="C:\Program Files (x86)\Microsoft SQL Server"

; Default directory for the Database Engine backup files.
```

```
SQLBACKUPDIR="C:\SQLBackups"

; Default directory for the Database Engine user databases.

SQLUSERDBDIR="C:\SQLData"

; Default directory for the Database Engine user database logs.

SQLUSERDBLOGDIR="C:\SQLLogs"

; Directories for Database Engine TempDB files.

SQLTEMPDBDIR="C:\TempDB"

; Provision current user as a Database Engine system administrator for
    SQL Server 2019 Express.

ADDCURRENTUSERASSQLADMIN="False"

; Specify 0 to disable or 1 to enable the TCP/IP protocol.

TCPENABLED="0"

; Specify 0 to disable or 1 to enable the Named Pipes protocol.

NPENABLED="0"

; Startup type for Browser Service.

BROWSERSVCSTARTUPTYPE="Disabled"

; Use SQLMAXMEMORY to minimize the risk of the OS experiencing
    detrimental memory pressure.

SQLMAXMEMORY="2147483647"

; Use SQLMINMEMORY to reserve a minimum amount of memory available to
    the SQL Server Memory Manager.

SQLMINMEMORY="0"
```

B.2 Visual Studio 2019 Enterprise, Offline installer

For full reference see: `https://docs.microsoft.com/en-us/visualstudio/install/create-an-offline-installation-of-visual-studio?view=vs-2019`

On a computer with Internet connectivity

1. Download vs2019_enterprise.
    - `https://visualstudio.microsoft.com/downloads/`
2. Open Powershell and locate the download file
3. Run: *`vs2019.exe --layout C:\..OUTPUT..\vs_data_and_procedure --add Microsoft.VisualStudio.Workload.CoreEditor --includeRecommended --includeOptional --add Microsoft.VisualStudio.Workload.Data --includeRecommended --includeOptional --add Microsoft.VisualStudio.Component.Roslyn.LanguageServices --add Microsoft.VisualStudio.Component.CoreEditor --add Microsoft.VisualStudio.Component.SQL.SSDT --add Microsoft.Net.Component.4.TargetingPack --add Microsoft.Net.Component.4.5.TargetingPack --add Microsoft.Net.Component.4.7.TargetingPack`*

Components list and Additional Extensions

```
## Components list
#   "components": [
#     "Microsoft.VisualStudio.Component.CoreEditor",
#     "Microsoft.VisualStudio.Workload.CoreEditor",
#     "Microsoft.NetCore.Component.Runtime.5.0",
#     "Microsoft.NetCore.Component.Runtime.3.1",
#     "Microsoft.NetCore.Component.SDK",
#     "Microsoft.VisualStudio.Component.NuGet",
#     "Microsoft.VisualStudio.Component.Roslyn.Compiler",
#     "Microsoft.VisualStudio.Component.Roslyn.LanguageServices",
#     "Microsoft.VisualStudio.ComponentGroup.WebToolsExtensions",
#     "Microsoft.VisualStudio.Component.DockerTools",
#     "Microsoft.Net.Component.4.8.SDK",
#     "Microsoft.Net.Component.4.7.2.TargetingPack",
#     "Microsoft.Net.ComponentGroup.DevelopmentPrerequisites",
#     "Microsoft.VisualStudio.Component.TypeScript.4.1",
#     "Microsoft.VisualStudio.Component.JavaScript.TypeScript",
#     "Microsoft.VisualStudio.Component.JavaScript.Diagnostics",
#     "Microsoft.Component.MSBuild",
#     "Microsoft.VisualStudio.Component.TextTemplating",
#     "Component.Microsoft.VisualStudio.RazorExtension",
#     "Microsoft.VisualStudio.Component.IISExpress",
#     "Microsoft.VisualStudio.Component.SQL.ADAL",
#     "Microsoft.VisualStudio.Component.SQL.LocalDB.Runtime",
#     "Microsoft.VisualStudio.Component.Common.Azure.Tools",
#     "Microsoft.VisualStudio.Component.SQL.CLR",
#     "Microsoft.VisualStudio.Component.MSODBC.SQL",
#     "Microsoft.VisualStudio.Component.MSSQL.CMDLnUtils",
#     "Microsoft.VisualStudio.Component.ManagedDesktop.Core",
#     "Microsoft.Net.Component.4.5.2.TargetingPack",
#     "Microsoft.Net.Component.4.5.TargetingPack",
```

```
#     "Microsoft.VisualStudio.Component.SQL.SSDT",
#     "Microsoft.VisualStudio.Component.SQL.DataSources",
#     "Component.Microsoft.Web.LibraryManager",
#     "Component.Microsoft.WebTools.BrowserLink.WebLivePreview",
#     "Microsoft.VisualStudio.ComponentGroup.Web",
#     "Microsoft.VisualStudio.Component.Web",
#     "Microsoft.Net.Component.4.TargetingPack",
#     "Microsoft.Net.Component.4.5.1.TargetingPack",
#     "Microsoft.Net.Component.4.6.TargetingPack",
#     "Microsoft.Net.ComponentGroup.TargetingPacks.Common",
#     "Microsoft.VisualStudio.Component.Azure.Compute.Emulator",
#     "Microsoft.VisualStudio.Component.Azure.Storage.Emulator",
#     "Microsoft.VisualStudio.Component.Azure.ClientLibs",
#     "Microsoft.VisualStudio.Component.Azure.AuthoringTools",
#     "Microsoft.VisualStudio.Component.CloudExplorer",
#     "Microsoft.Net.Component.4.7.TargetingPack",
#     "Microsoft.VisualStudio.Component.Azure.Waverton.BuildTools",
#     "Microsoft.VisualStudio.Component.Azure.Waverton",
#     "Microsoft.Component.Azure.DataLake.Tools",
#     "Microsoft.VisualStudio.Workload.Data"
#   ]

## Additional Extensions
#   Microsoft RDLC Report Designer - V15.3.1
#   Microsoft Reporting Services Project - V2.6.7
#   SQL Server Integration Services Projects - V3.12
```

Appendix C

SQL Code - The database and tables

C.1 Create DB and main table

Purpose Create the main database, and the main RAW data table

Primary key (PK) Gateway, AccountingID

Params N/A

```
SET NOCOUNT ON
-- Run this script to create our initial db, and known tables
USE [master];
GO

-- Checking to see if our database exists and if it does drop it
IF DATABASEPROPERTYEX ('HDO_CDR', 'Version') IS NOT NULL
BEGIN
        ALTER DATABASE [HDO_CDR] SET SINGLE_USER
        WITH ROLLBACK IMMEDIATE;
        DROP DATABASE [HDO_CDR];
END
GO

CREATE DATABASE [HDO_CDR];
GO

ALTER DATABASE [HDO_CDR] SET RECOVERY SIMPLE;
GO

USE [HDO_CDR];
GO

CREATE SCHEMA [HDO];
GO

CREATE TABLE [HDO].[CDR_RAW] (
```

```
    [GatewayName] VARCHAR(23) NOT NULL,        -- Field AS2, Max 23 char   | PK
    [AccountingID] VARCHAR(64) NOT NULL,-- Field AS3, 64-Bit Hex    | PK
    [Recordtype] VARCHAR(7) NOT NULl, -- Field AS1 : Max 7 char
    [StartDateTime] DATETIME2(1) NOT NULL, -- Field AS6 and AS7   YYYY-MM-DD
    ↪  hh:mm:ss[.nnnnnnn]
    [DisconnectDateTime] DATETIME2(1) NOT NULL,-- Field Date: S11, A105 Time: S12,
    ↪  A10
    [CallServiceDuration] INT NULL,        -- Field S14, Attempt NULL
    [CallingNumber] VARCHAR(30) NOT NULL, -- Field S20, A17
    [CalledNumber] VARCHAR(30) NOT NULL,-- Field S21, A18
    [RouteLabel] VARCHAR(23) NULL,        -- Field S29, A26
    [IngressTrunkGroup] VARCHAR(23) NULL, -- Field S34, A31
    [EgressTrunkGroup] VARCHAR(23) NULL, -- Field S68, A58
    [CallDisconnectReason] INT NOT NULL, -- Field S15, A12
    [CallDisconnectReasonIngress] INT NULL, -- Field S121, A111
    [CallDisconnectReasonEgress] INT NULL, -- Field S122, A112
        [IngressRemoteSignalingIP] VARCHAR(39) NULL,
        [EgressRemoteSignalingIP] VARCHAR(39) NULL,
        [InsertDateTime] datetime NOT NULL
)
GO

ALTER TABLE [HDO].[CDR_RAW]
        ADD CONSTRAINT PK_CDR_RAW PRIMARY KEY ([GatewayName],[AccountingID]);
GO
```

C.2 Create Aggregation table

Purpose Create the tables used by the stored procedures

Trunk: PK PeriodStart, Trunk

Route: PK PeriodStart, Route

Discausse: PK PeriodStart, DiscausseReason

Params N/A

```
SET NOCOUNT ON
-- Run this script to create our initial db
USE [master];
GO

USE [HDO_CDR];
GO
-- Trunk Summery
-- Checking to see if our database exists and if it does drop it
DROP TABLE IF exists HDO.Trunk_Summery
GO

CREATE TABLE [HDO].[Trunk_Summery] (
[PeriodStart] DATETIME2(1) NOT NULL, -- Date and timeslot
[Period] INT NOT NULl, -- Timeslot duration in secounds
```

```
[Trunk] VARCHAR(23) NOT NULL, -- Ingress and egress union
[StartedCalls] INT NOT NULL, -- Active calls during a timeslot
[SimCalls] INT NOT NULL -- MAX ammount of simultanius calles during a timeslot
)
GO

ALTER TABLE [HDO].[Trunk_Summery]
        ADD CONSTRAINT PK_Trunk_Summery PRIMARY KEY ([PeriodStart], [Trunk]);
GO

-- Route Summery
DROP TABLE IF exists HDO.Route_Summery
GO

CREATE TABLE [HDO].[Route_Summery] (
[PeriodStart] DATETIME2(1) NOT NULL, -- PK | Date and timeslot
[Period] INT NOT NULl, -- Timeslot duration in secounds
[Route] VARCHAR(23) NOT NULL, -- PK | Route
[DisconnectReason] INT NOT NULL, -- CalldisconnectReason from CDR_RAW
[Calls] INT NOT NULL, -- Number of calls during a timeslot with given disconnect
↪  code
[NumCallsConnected] INT NOT NULL, -- Number of calls connected during a timeslot
↪  with given disconnect code
[NumCallsAttempt] INT NOT NULL -- Number of calls attempted during a timeslot with
↪  given disconnect code
)
GO


ALTER TABLE [HDO].[Route_Summery]
        ADD CONSTRAINT PK_Route_Summery PRIMARY KEY ([PeriodStart], [Route],
        ↪  [DisconnectReason]);
GO

-- HDO.Discausse_Reasons
DROP TABLE IF exists HDO.Discausse_Reasons
GO

CREATE TABLE [HDO].[Discausse_Reasons] (
[PeriodStart] DATETIME2(1) NOT NULL, -- PK | Date and timeslot
[Period] INT NOT NULl, -- Timeslot duration in secounds
[DiscausseReason] INT NOT NULL, -- CalldisconnectReason from CDR_RAW
[Calls] INT NOT NULL, -- Number of calls during a timeslot with given disconnect
↪  code
)
GO


ALTER TABLE [HDO].[Discausse_Reasons]
        ADD CONSTRAINT PK_Discausse_Reasons PRIMARY KEY ([PeriodStart],
        ↪  [DiscausseReason]);
GO
```

Appendix D

SQL Code - Stored Procedures

D.1 Trunk_Summary

Purpose Aggregate out relevant call information group by trunk(s), with numbers of calls started within a period and simultaneous calls that accounts for calls in earlier periods.

Params:

@date Input: `Date`
Object: Create 30 minute periods for a 24 hour period on the given date. These periods is then used as limitations for the requested data (`src.StartDateTime [>=, <] period.period_start`)

@period: Input: `INT`
Object: Static parameter set to 1800 secounds (30 min) used to visualize period in returened results

@ST: Input: `Datetime2`
Object: Lower limit for query results on the RAW Table

@DT: Input: `Datetime2`
Object: Upper limit for query results on the RAW Table

```
SET NOCOUNT ON
USE [HDO_CDR]
GO
CREATE OR ALTER PROCEDURE [HDO].[GenerateTrunkSum]
@date datetime2(7),
@ST datetime2(1),
@DT datetime2(1),
@period int
AS
BEGIN
WITH Numbers(val) AS
(
```

```
  SELECT
    1 UNION ALL SELECT val + 1 FROM numbers  WHERE val < 48
)
,ConnectionPeriod AS
(
  SELECT
    period.period_start
   ,period.period_end
   ,@period as Period
   ,src.Trunk
   ,CASE
      WHEN src.StartDateTime < period.period_start THEN period.period_start
      ELSE src.StartDateTime
    END AS StartDateTime
   ,CASE
      WHEN src.DisconnectDateTime > period.period_end THEN period.period_end
      ELSE src.DisconnectDateTime
    END AS DisconnectDateTime
  FROM
    (
      SELECT
        dateadd(minute, (val - 1) * 30, @date) as period_start
       ,dateadd(minute, (val    ) * 30, @date) as period_end
      FROM
        numbers
    ) period
  INNER JOIN
    (
      SELECT
        IngressTrunkGroup as Trunk
       ,StartDateTime
       ,DisconnectDateTime
      FROM
        [HDO].[CDR_RAW]
                WHERE StartDateTime BETWEEN @ST AND @DT -- Limits to Timeinterval
        UNION ALL
      SELECT
        EgressTrunkGroup
       ,StartDateTime
       ,DisconnectDateTime
      FROM
        [HDO].[CDR_RAW]
                WHERE StartDateTime BETWEEN @ST AND @DT -- Limits to Timeinterval
    ) src
      ON src.StartDateTime >= period.period_start
          AND src.StartDateTime < period.period_end
) --Select -- Comment out line below this to instead just show without insert
INSERT INTO [HDO].[Trunk_Summery] SELECT
  PeriodSummary.period_start
 ,PeriodSummary.Period
 ,PeriodSummary.Trunk
 ,PeriodSummary.[all]
 ,MAX(COALESCE(ConnectionSummary.ConnectionCount,0)) AS sim
FROM
```

```
  (
    SELECT -- Line 81
      period_start
     ,period_end
     ,@period as Period
     ,Trunk
     ,COUNT(*) as [all]
    FROM
    ConnectionPeriod
    GROUP BY
     Trunk
    ,period_start
    ,period_end
  ) PeriodSummary
INNER JOIN
  (
    SELECT
      cp1.period_start
     ,cp1.Trunk
     ,cp1.StartDateTime
     ,cp1.DisconnectDateTime
     ,COUNT(*) AS ConnectionCount
    FROM
      ConnectionPeriod cp1
    LEFT JOIN
      ConnectionPeriod cp2
        ON cp2.Trunk = cp1.Trunk
            AND cp2.StartDateTime < cp1.DisconnectDateTime
            AND cp2.DisconnectDateTime > cp1.StartDateTime
--        WHERE cp1.Trunk != '' -- Removes '' and NULL Trunks could be adjusted.
    GROUP BY
      cp1.period_start
     ,cp1.Trunk
     ,cp1.StartDateTime
     ,cp1.DisconnectDateTime
  ) ConnectionSummary
    ON ConnectionSummary.Trunk = PeriodSummary.Trunk
        AND ConnectionSummary.period_start =  PeriodSummary.Period_Start
WHERE NOT EXISTS -- Line 118
-- Validetes that PRIMARY KEY dosen't exists within [HDO].[]
        (
        SELECT  @@ROWCOUNT as 'Existed', ts.PeriodStart, ts.Trunk
        FROM    [HDO].[Trunk_Summery] as ts
        WHERE   ts.PeriodStart = PeriodSummary.period_start AND ts.Trunk =
        ↪  PeriodSummary.Trunk
        )
GROUP BY
  PeriodSummary.period_start
 ,PeriodSummary.Period
 ,PeriodSummary.Trunk
 ,PeriodSummary.[all];
END
GO
```

```
-- This illustrates an execution run, typically handle via SQL Agent
DECLARE @RUN_DATE datetime2(1) = GETDATE(); -- '2021-04-29 00:00:00.0'
--DECLARE @Rounded datetime2(1) = DATEADD(mi, DATEDIFF(mi, 0, @RUN_DATE)/30*30, 0)
DECLARE @START date = DATEADD(day,-1 , @RUN_DATE);
DECLARE @END date= DATEADD(day, 1, @START);
SELECT @START, @END;
EXECUTE [HDO].[GenerateTrunkSum] @date = @RUN_DATE, @period = 1800, @ST = @START,
↪  @DT = @END;
SELECT @@ROWCOUNT as 'Inserted';
GO
```

D.2 Route_Summary

Purpose Aggregate out relevant call information group by route(s), with numbers of calls, calls terminated and calls attempted; with its associated disconnection code.

Params:

@date Input: `Date`
Object: Create 30 minute periods for a 24 hour period on the given date. These periods is then used as limitations for the requested data (`src.StartDateTime [>=, <] period.period_start`)

@period: Input: `INT`
Object: Static parameter set to 1800 secounds (30 min) used to visualize period in returened results

@ST: Input: `Datetime2`
Object: Lower limit for query results on the RAW Table

@DT: Input: `Datetime2`
Object: Upper limit for query results on the RAW Table

```
USE [HDO_CDR]
GO

CREATE OR ALTER PROCEDURE [HDO].[GenerateRouteSum]
@date datetime2(1),
@period int,
@ST datetime2(1),
@DT datetime2(1)
AS
BEGIN

with numbers(val) as
    (select 1 union all select val + 1 from numbers where val < 48),
periods as (
    select @date as [date], nbr.val,
   dateadd(minute, (nbr.val - 1) * 30, @date) as PeriodStart,
   dateadd(minute, (nbr.val    ) * 30, @date) as PeriodEnd
```

```
   from numbers as nbr)
--SELECT
INSERT INTO [HDO].[Route_Summery] SELECT
        pers.PeriodStart
        ,30 as Period
        ,src.RouteLabel as [Route]
        ,src.CallDisconnectReason as DisconnectReason
        ,COUNT(*) as 'Calls'
        ,COUNT(case when src.Recordtype = 'STOP' then 1 else null end) as
        ↪  NumCallsConnected
        ,COUNT(case when src.Recordtype = 'ATTEMPT' then 1 else null end) as
        ↪  NumCallsAttempt
from periods as pers inner join HDO.CDR_RAW as src
on src.StartDateTime >= pers.PeriodStart and src.StartDateTime < pers.PeriodEnd
WHERE (src.StartDateTime >= @ST and src.StartDateTime <= @DT) AND
-- Validetes that PRIMARY KEY dosen't exists within [HDO].[]
      src.RouteLabel is not NULL and  NOT EXISTS(
       SELECT  @@ROWCOUNT as 'Existed', rs.PeriodStart, rs.[Route],
       ↪  rs.DisconnectReason
       FROM    [HDO].[Route_Summery] as rs
        WHERE   rs.PeriodStart = pers.PeriodStart AND rs.[Route] = src.RouteLabel
        ↪  AND rs.[DisconnectReason] = src.CallDisconnectReason
        )
group by pers.PeriodStart, src.RouteLabel, src.CallDisconnectReason
order by pers.PeriodStart, src.RouteLabel, src.CallDisconnectReason
END
GO

-- This illustrates an execution run, typically handle via SQL Agent
DECLARE @RUN_DATE datetime2(1) = GETDATE();
--DECLARE @Rounded datetime2(1) = DATEADD(mi, DATEDIFF(mi, 0, @RUN_DATE)/30*30, 0)
DECLARE @START date = DATEADD(day,-1 , @RUN_DATE);
DECLARE @END date= DATEADD(day, 1, @START);
SELECT @START, @END;
EXECUTE [HDO].[GenerateRouteSum] @date = @START, @period = 1800, @ST = @START, @DT
↪  = @END;
SELECT @@ROWCOUNT as 'Inserted'
GO
```

D.3 Discausse_Reasons

Purpose Aggregate out terminated calls with its associated disconnection code.

Params:

@date Input: `Date`
Object: Create 30 minute periods for a 24 hour period on the given date. These periods is then used as limitations for the requested data (`src.StartDateTime [>=, <] period.period_start`)

@period: Input: `INT`

Object: Static parameter set to 1800 secounds (30 min) used to visualize period in returened results

@ST: Input: `Datetime2`
Object: Lower limit for query results on the RAW Table

@DT: Input: `Datetime2`
Object: Upper limit for query results on the RAW Table

```
USE [HDO_CDR]
GO

CREATE OR ALTER PROCEDURE [HDO].[GenerateDiscausseReasons]
@date datetime2(1),
@period int,
@ST datetime2(1),
@DT datetime2(1)
AS
BEGIN

with numbers(val) as
    (select 1 union all select val + 1 from numbers where val < 48),
periods as (
    select @date as [date], nbr.val,
   dateadd(minute, (nbr.val - 1) * 30, @date) as PeriodStart,
   dateadd(minute, (nbr.val    ) * 30, @date) as PeriodEnd
   from numbers as nbr)
INSERT INTO [HDO].[Discausse_Reasons] SELECT
        pers.PeriodStart
        ,@period as Period
        ,src.CallDisconnectReason as DiscausseReason
        ,COUNT(*) as 'Calls'
from periods as pers inner join HDO.CDR_RAW as src
on src.StartDateTime >= pers.PeriodStart and src.StartDateTime < pers.PeriodEnd
WHERE (src.StartDateTime >= @ST and src.StartDateTime <= @DT)
        -- Validetes that PRIMARY KEY dosen't exists within [HDO].[]
        AND NOT EXISTS
        (
        SELECT  @@ROWCOUNT as 'Existed', rs.PeriodStart, rs.[DiscausseReason]
        FROM    [HDO].[Discausse_Reasons] as rs
        WHERE   rs.PeriodStart = pers.PeriodStart AND rs.[DiscausseReason] =
        ↪  src.CallDisconnectReason
        )
group by pers.PeriodStart, src.CallDisconnectReason
order by pers.PeriodStart, src.CallDisconnectReason
END
GO

-- This illustrates an execution run, typically handle via SQL Agent
DECLARE @RUN_DATE datetime2(1) = GETDATE();
--DECLARE @Rounded datetime2(1) = DATEADD(mi, DATEDIFF(mi, 0, @RUN_DATE)/30*30, 0)
DECLARE @START date = DATEADD(day,-1 , @RUN_DATE);
DECLARE @END date= DATEADD(day, 1, @START);
SELECT @START, @END;
```

```
EXECUTE [HDO].[GenerateDiscausseReasons] @date = @START, @period = 1800, @ST =
↪  @START, @DT = @END;
SELECT @@ROWCOUNT as 'Inserted'
GO
```

Appendix E

SQL Agent Job - Stored Procedures

Note: This following types are common across all of the subsequent job implementations

Owner: sa

Type: T-SQL

Database: HDO_CDR

E.1 Daily jobs

Note: This is two different SQL Server Agent jobs

Name: Daily_Discausse_Reasons_Summary_Aggregation
Daily_Route_Summary_Aggregation

Step: Generate Daily Discausse Reasons from the last day
Generate Route Sumamry from the last day

Schedule: Quarter_Past_Midnight_Daily_Run;
Recurring Daily at 00:15:00

```
DECLARE @RUN_DATE datetime2(1) = GETDATE();
DECLARE @START date = DATEADD(day,-1, @RUN_DATE);
DECLARE @END date= DATEADD(day, 1, @START);

-- Discausse
EXECUTE [HDO].[GenerateDiscausseReasons] @date = @START, @period = 1800,
@ST = @START, @DT = @END
-- Route
EXECUTE [HDO].[GenerateRouteSum] @date = @START, @period = 1800,
@ST = @START, @DT = @END;
```

Name: Daily_Trunk_Summary_Aggregation

Step: Generate Trunk Summary from the last day

Schedule: Quarter_Past_Midnight_Daily_Run;
Recurring Daily at 00:15:00

```
DECLARE @RUN_DATE date = GETDATE();
DECLARE @LOW date;
DECLARE @HIGH date;

SET @LOW = DATEADD(day, -1, @RUN_DATE)
SET @HIGH = DATEADD(day, 0, @RUN_DATE)

EXECUTE [HDO].[GenerateTrunkSum] @date = @LOW, @period = 1800,
@ST = @LOW, @DT = @HIGH;
```

E.2 Manual Jobs

These job were created to so alternative implementation option were shorter iteration options ar used.

Name: Manual_Discausse_Reasons_Summary_current_day_from_midnight_to_last_full_period
Manual_Route_Summary_current_day_from_midnight_to_last_full_period

Step: Generate Discausse Reasons from midnight to last finished period
Generate Route Sumamry from midnight to last full period

```
DECLARE @RUN_DATE datetime2(1) = GETDATE();
DECLARE @Rounded datetime2(1) = DATEADD(mi, DATEDIFF(mi, 0, @RUN_DATE)/30*30, 0)
DECLARE @START date = CAST(@RUN_DATE AS date)
DECLARE @END datetime2(1);

SET @START = DATEADD(day, 0, @RUN_DATE)
SET @END = DATEADD(SECOND, 1, @Rounded)
--DiscausseReason
EXECUTE [HDO].[GenerateDiscausseReasons] @date = @START, @period = 1800, @ST =
↪  @START, @DT = @END;
-- Route
EXECUTE [HDO].[GenerateRouteSum] @date = @START, @period = 1800,
@ST = @START, @DT = @END;
```

Note: This is two different SQL Server Agent jobs

Name: Manual_Discausse_Reasons_summary_Last_2h_from_last_full_period
Manual_Route_summary_Last_2h_from_last_full_period

Step: Generating Discausse Reasons summery for the last 2h since last full period
Generating Route summery for the last 2h since last full period

```
DECLARE @RUN_DATE datetime2(1) = GETDATE();
DECLARE @Rounded datetime2(1) = DATEADD(mi, DATEDIFF(mi, 0, @RUN_DATE)/30*30, 0)
DECLARE @START datetime2(1);
DECLARE @END datetime2(1);

SET @START = DATEADD(hour, -2, @Rounded);
SET @END = DATEADD(mi, 1, @Rounded);
-- Discausse
EXECUTE [HDO].[GenerateDiscausseReasons] @date = @START, @period = 1800, @ST =
↪  @START, @DT = @END;
-- Route
EXECUTE [HDO].[GenerateRouteSum] @date = @START, @period = 1800,
@ST = @START, @DT = @END;
```

Note: This is two different SQL Server Agent jobs

Name: Manual_Trunk_Summary_current_day_from_midnight_to_last_full_period

Step: Generate Trunk Summary from midnight to last finished period

```
DECLARE @RUN_DATE datetime2(1) = GETDATE();
DECLARE @Rounded datetime2(1) = DATEADD(mi, DATEDIFF(mi, 0, @RUN_DATE)/30*30, 0)
DECLARE @LOW date = CAST(@RUN_DATE AS date)
DECLARE @HIGH datetime2(1);

SET @LOW = DATEADD(day, 0, @RUN_DATE)
SET @HIGH = DATEADD(SECOND, 1, @Rounded)

EXECUTE [HDO].[GenerateTrunkSum] @date = @LOW, @period = 1800,
@ST = @LOW, @DT = @HIGH;
```

Note: This is two different SQL Server Agent jobs

Name: Manual_Trunk_Summary_Last_2h_from_last_full_period

Step: Generating Trunk summery for the last 2h since last full period

```
DECLARE @RUN_DATE datetime2(1) = GETDATE();
DECLARE @Rounded datetime2(1) = DATEADD(mi, DATEDIFF(mi, 0, @RUN_DATE)/30*30, 0)
DECLARE @LOW datetime2(1);
DECLARE @HIGH datetime2(1);

SET @LOW = DATEADD(hour, -2, @Rounded)
SET @HIGH = DATEADD(mi, 1, @Rounded)

EXECUTE [HDO].[GenerateTrunkSum] @date = @LOW, @period = 1800,
@ST = @LOW, @DT = @HIGH;
```

Appendix F

SQL Code - View

F.1 CDR_RAW_View

```
USE ReportServer
GO

CREATE VIEW dbo.CDR_RAW_View
AS
SELECT *, CONVERT(DateTime2(1), StartDateTime AT TIME ZONE 'UTC' AT TIME ZONE
↪  'Central European Standard Time') AS RightStartTime, CONVERT(DateTime2(1),
↪  DisconnectDateTime AT TIME ZONE 'UTC' AT TIME ZONE 'Central European Standard
↪  Time') AS RightDisconnectDateTime, CONVERT(DateTime, InsertDateTime AT TIME
↪  ZONE 'UTC' AT TIME ZONE 'Central European Standard Time') AS
↪  RightInsertDateTime --Selecting every field and adding fields with time/date in
↪  our time zone

FROM [HDO_CDR].[HDO].[CDR_RAW];
GO

-- Showing the view
SELECT *
FROM dbo.CDR_RAW_View;
```

F.2 Trunk_Summary_View

```
USE ReportServer
GO

CREATE VIEW dbo.Trunk_Summary_View
AS
SELECT *, CONVERT(DateTime2(1), (PeriodStart AT TIME ZONE 'UTC' AT TIME ZONE
↪  'Central European Standard Time')) AS RightPeriodStartTime --Selecting every
↪  field and adding a field with time/date in our time zone
FROM [HDO_CDR].[HDO].[Trunk_Summery];
GO

-- Showing the view
```

```sql
SELECT *
FROM dbo.Trunk_Summary_View;
```

F.3 Route_Summary_View

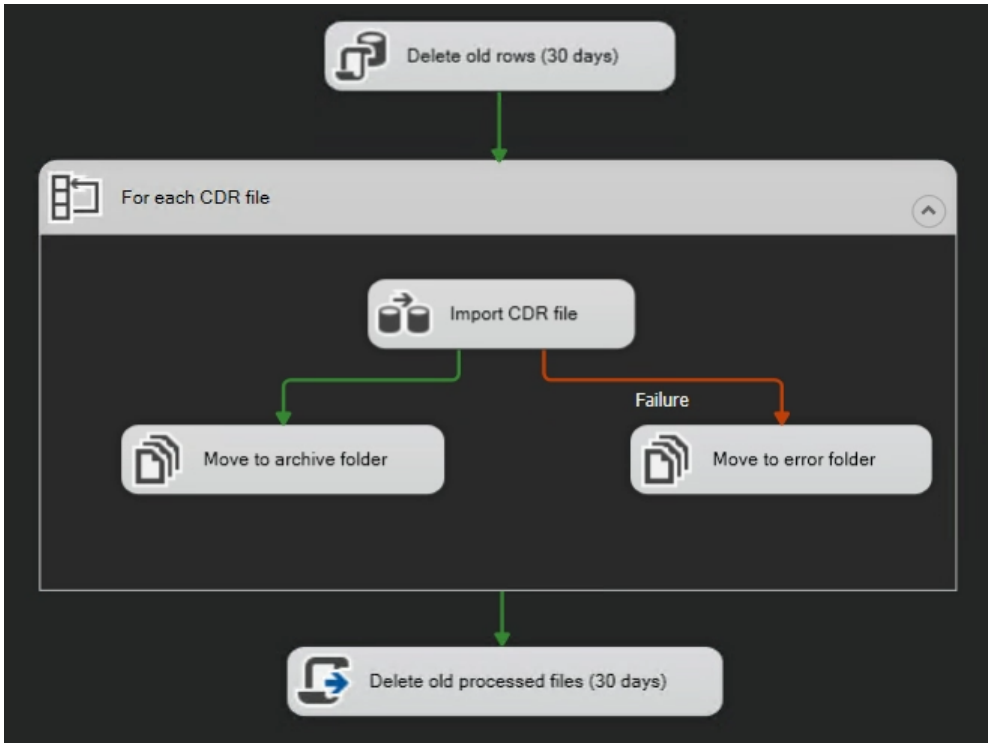
```sql
USE ReportServer
GO

CREATE VIEW dbo.Route_Summary_With_Discause_View
AS
SELECT *, CONVERT(DateTime2(1), (PeriodStart AT TIME ZONE 'UTC' AT TIME ZONE
↪  'Central European Standard Time')) AS RightPeriodStartTime
FROM [HDO_CDR].[HDO].[Route_Summery] LEFT OUTER JOIN
                        [HDO_CDR].[HDO].[ISUP_SIP_MAP] ON
                        ↪  [HDO_CDR].[HDO].[Route_Summery].DisconnectReason =
                        ↪  [HDO_CDR].[HDO].[ISUP_SIP_MAP].Reason_CodeReceived
GO

-- The view
SELECT *
FROM dbo.Route_Summary_With_Discause_View;
```

Appendix G

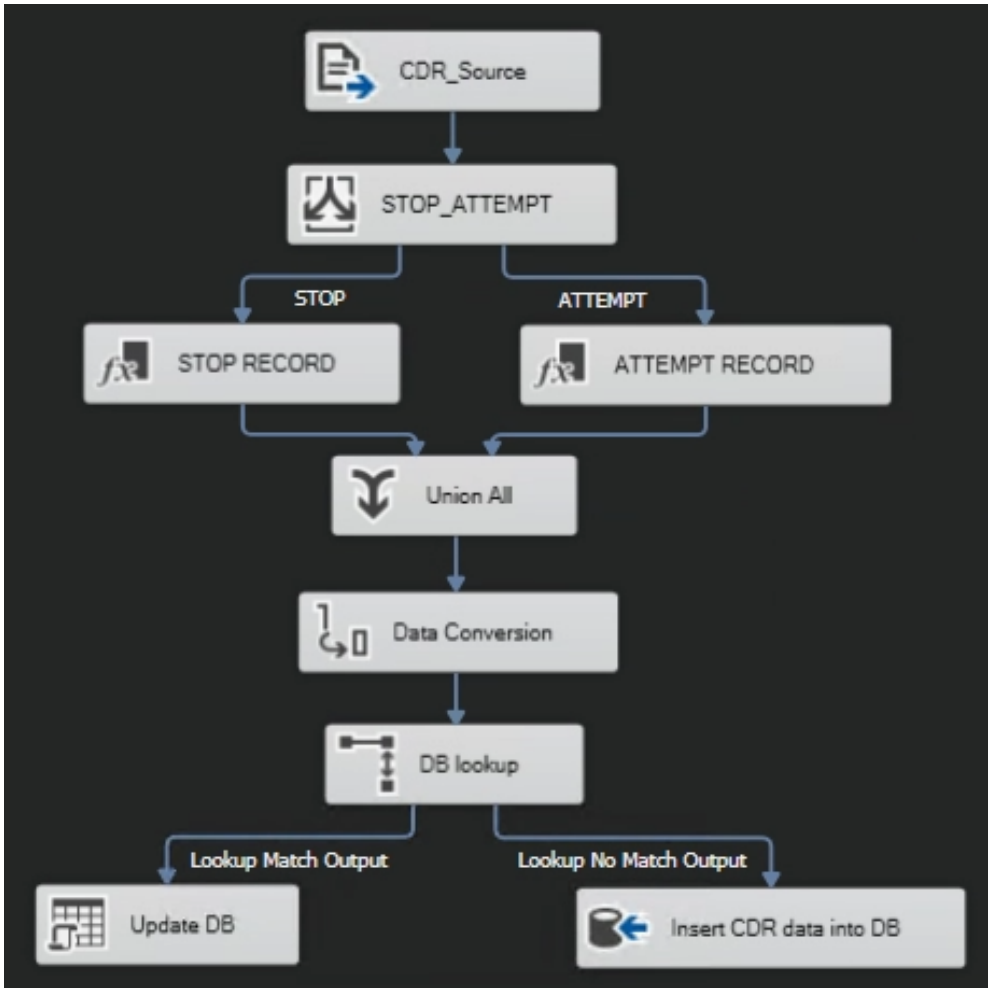
SSIS Configuration

G.1 Connections

Control flow:
• To change the connection between two tasks to an error output. Simply right-click the current connection which is green and select "Failure".



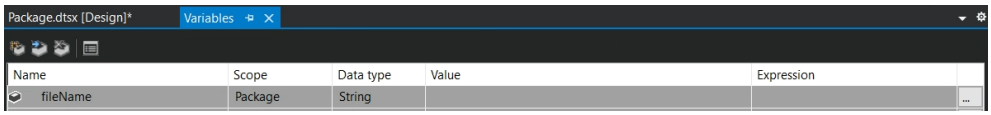
Data flow:



G.2 Variables

Create a variable such as "fileName" below:
- Set "Scope" to package.
- Set "Data type" to string.
- Set "Value" to blank.

This variable is going to be used in the control flow for reading/importing each CDR file in a folder for further processing (dynamic solution).



G.3 Control flow

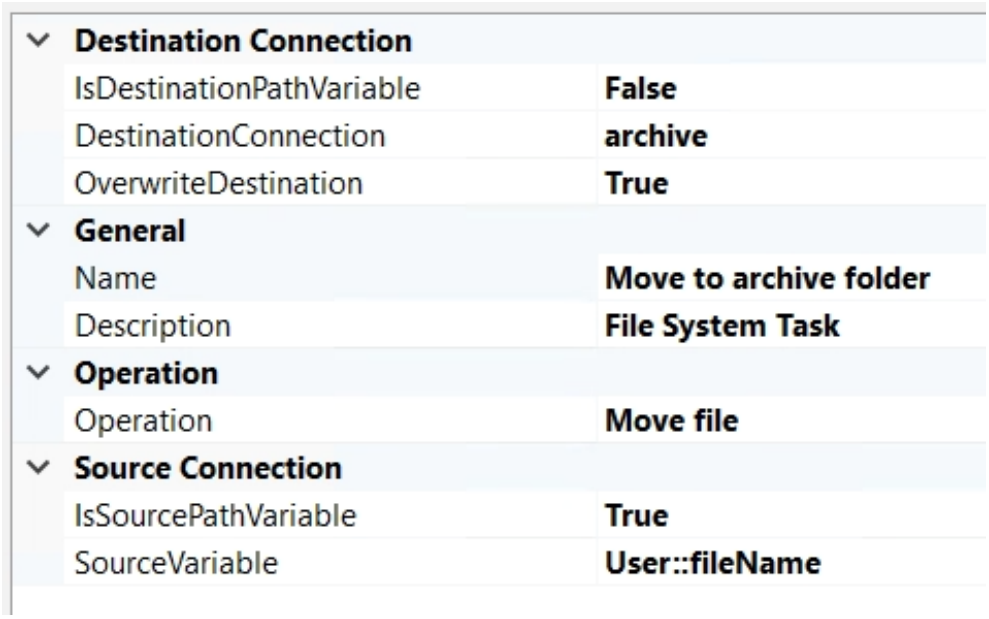
SSIS tasks and containers (the components a package is built of) are available in the "SSIS Toolbox" within the SSIS GUI.

Name: For each CDR file
Component: For each loop container
Configuration:
Within the For each loop container under "Collection":
• Choose the folder that contains the CDR files
• Make sure to only loop over ACT files, this is because the CDR files have a .ACT extension. Like this: *.ACT*
• Set retrieve file name to "Fully qualified"
Under "Variable Mappings", select:
• The created fileName variable, index 0.

Variable	Index
User::fileName	0

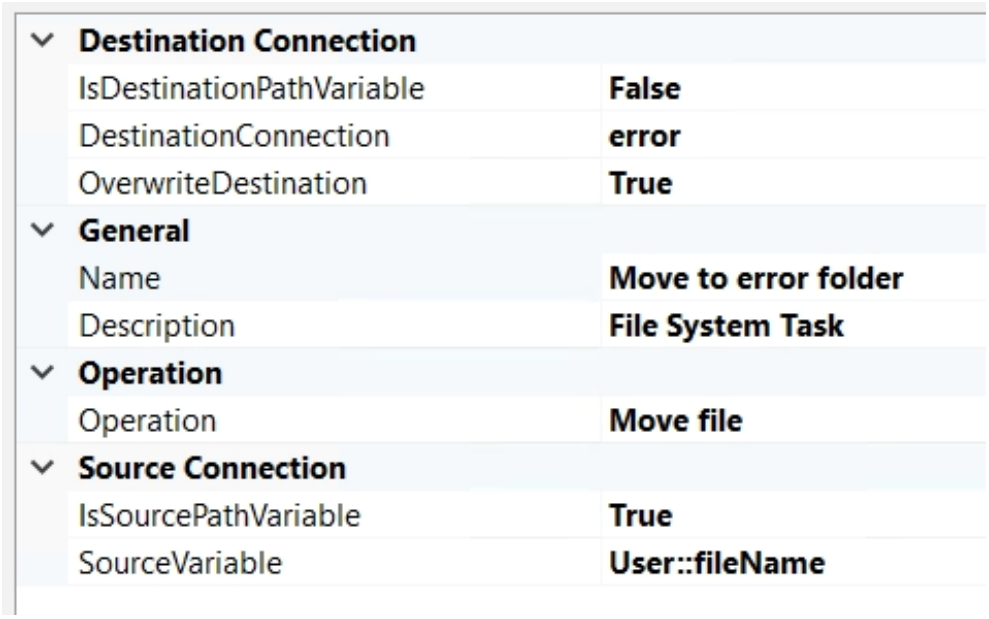
Name: Import CDR file
Component: Data flow task
Configuration: For the configuration of the Data flow task, see [Appendix G.4].

Name: Move to archive folder
Component: File system task
Configuration:
• Set DestinationConnection to a folder in the filesystem where the CDR files are going to be moved into after a successful import of a file.
• Make sure the SourceVariable is set to the same variable used in the For each loop container.
• Otherwise configure as shown in the figure below:



When the file system task is configured. Right-click it and select properties. Set "DelayValidation" as True. This is to deal with the variable that is blank, since this task is using that variable under run-time when moving a given CDR file.

Name: Move to error folder
Component: File system task
Configuration:
• Set DestinationConnection to a folder in the filesystem where the CDR files are going to be moved into after a failed import of a file.
• Make sure the SourceVariable is set to the same variable used in the For each loop container.
• Otherwise configure as shown in the figure below:



When the file system task is configured. Right-click it and select properties. Set "DelayValidation" as True. This is to deal with the variable that is blank, since this task is using that variable under run-time when moving a given CDR file. See [Appendix G.1] to manage the connection between the "Import CDR file" task and the "Move to error folder" task.

Name: Delete old processed files (30 days)
Component: Script task
Configuration:
There may be bugs related to the Script Task, a prerequisite that often solves those errors are having the most recent version of visual studio tools installed.

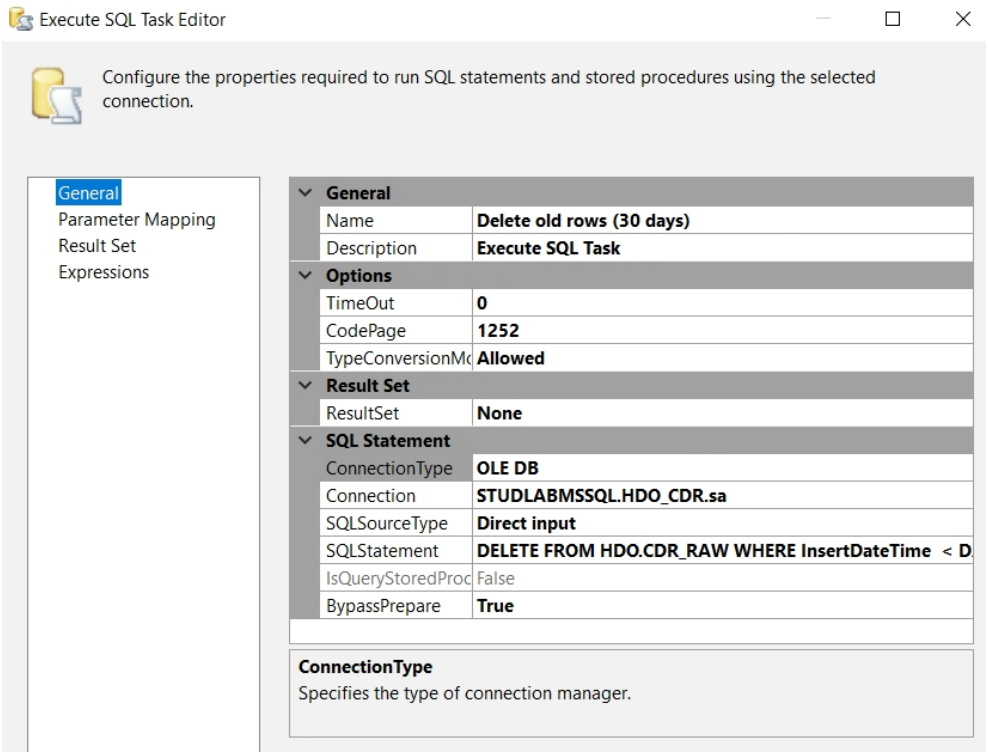
Within the Script Task, click on "Edit Script"
• Within the C# code, under "Namespaces". Add: using System.IO;
Add the following C# code to Main():

```
public void Main() {
        string[] files = Directory.GetFiles("C:\\dataImport\\archive");

        foreach (string file in files) {
            FileInfo fi = new FileInfo(file);
            if (fi.CreationTime < DateTime.Now.AddDays(-30))
                fi.Delete();
        }
Dts.TaskResult = (int)ScriptResults.Success;
```

The C# code is inspired by [69]

Name: Delete old rows (30 days)

Component: Delete old rows (30 days)
Configuration:

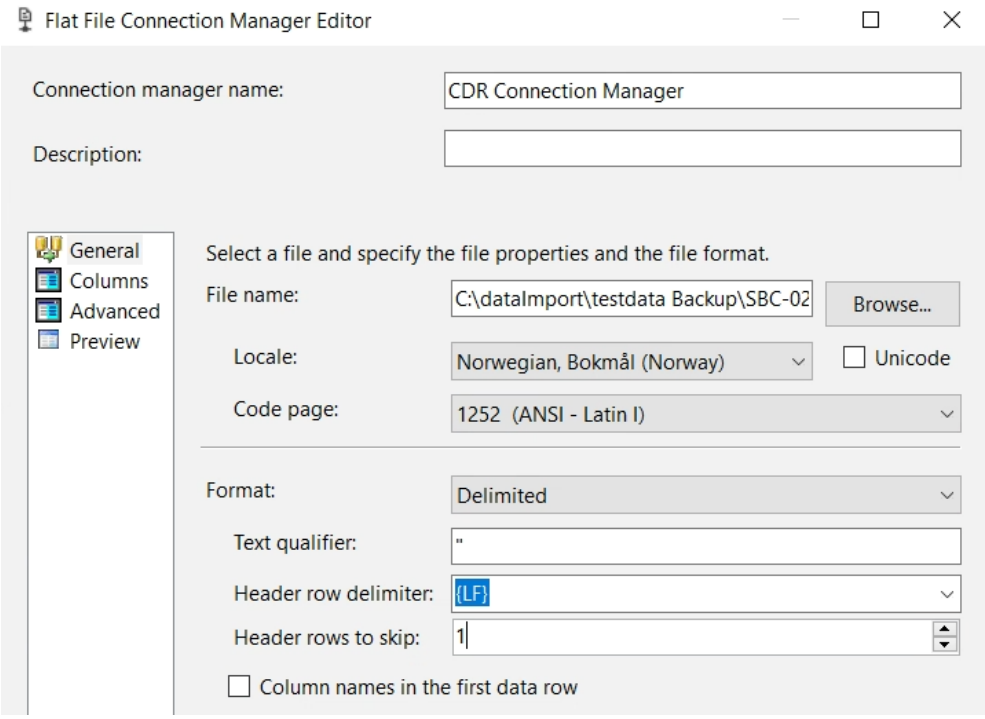


Make sure "Connection" is the OLE DB connection manager that is going to be created in the data flow. This leads to this task being configured when the whole data flow configuration is done.
Insert the following SQL code into "SQLStatement":

```
1  DELETE FROM HDO.CDR_RAW WHERE InsertDateTime  < DATEADD(day, -30, GETDATE())
```

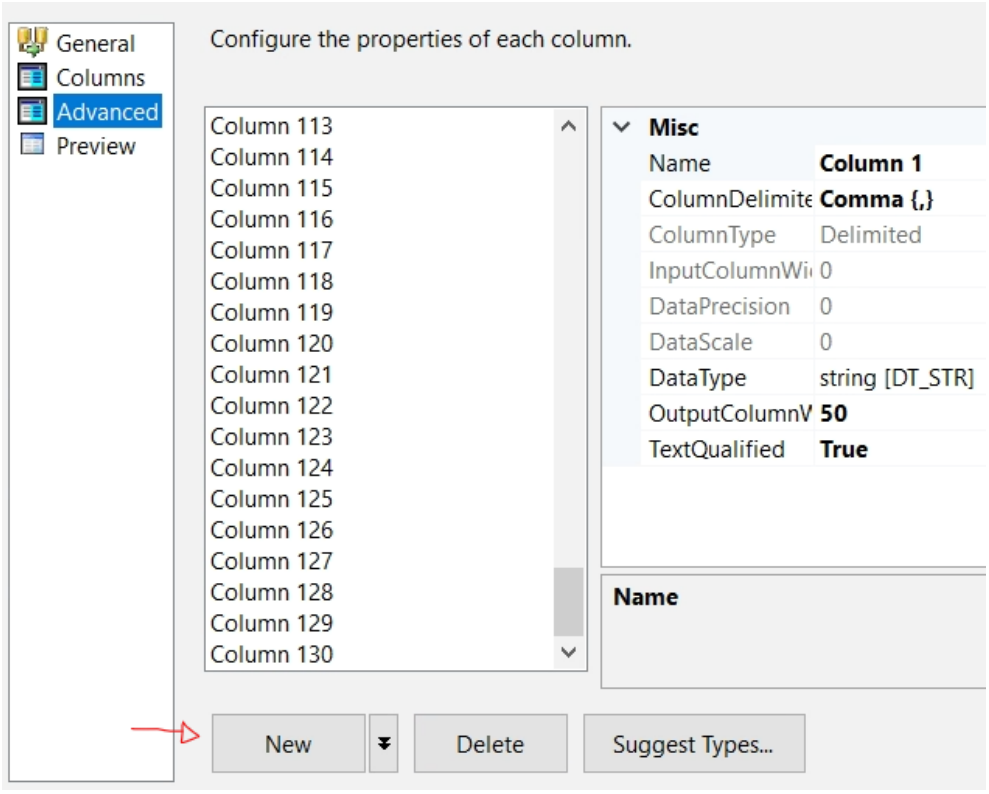
G.4 Data flow

Name: CDR_Source
Component: Flat file source
Configuration:
Within the Flat file source, add a Flat file connection manager. In "General", configure the flat file connection manager as the figure below. Keep in mind that the File name is an existing CDR file that is only used to retrieve the correct columns from the file. Make sure that this CDR file contains both STOP and ATTEMPT records.

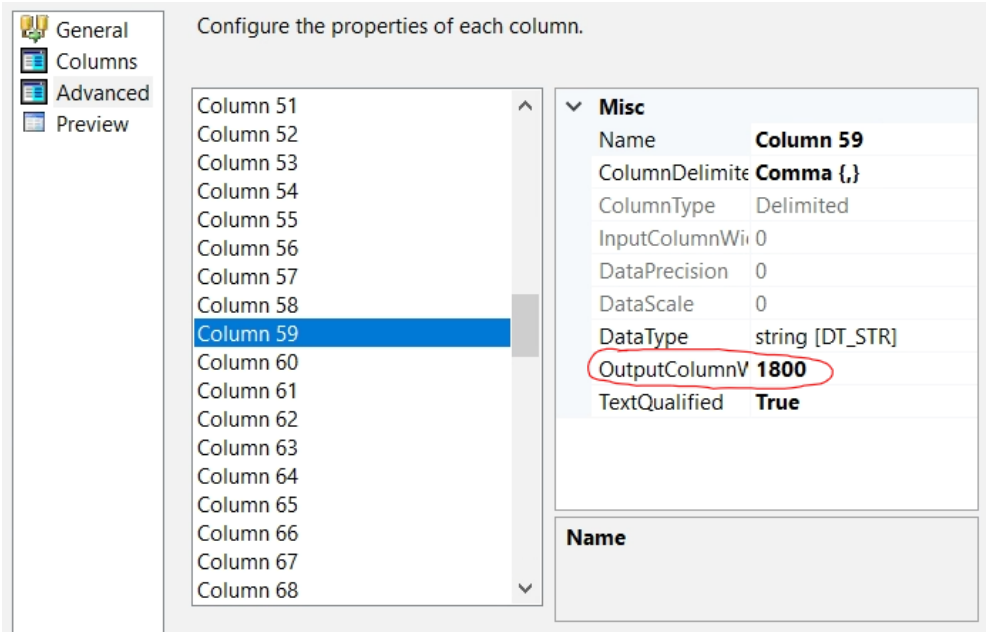


In "Columns", make sure that:
- Row delimiter is set to {LF}
- Column delimiter is set to Comma {,}

When configuring the "Advanced" section, there are only some columns appearing at the start. Since the CDR file contains many more columns, it is necessary to click on "New" to add the required columns. In this project, 130 columns are added/used as shown in the figure below. This is because column 126 is the "highest column value" used by the package in further transformation tasks.



Further, in the data flow, there are expressions that are using findstring functions. This is because it is necessary to search for some characters regarding a conditional statement. The findstring function searches in columns 45, 52, 59, and 69 for a given CDR file. Since the string in those columns have a very large length, which means the length must be handled early to avoid truncate errors. Within that long string, we are searching for "BYE" or "CAN". See [Appendix G.4] and [Appendix G.4]]. Therefore, the length is set to 1800 as shown in the figure below (that the length is set to 1800 is probably an exaggeration, a lower length is possible). Set the length to 1800 for the four columns. It is also worth mentioning that it is not necessary to alter the DataType for each column in this configuration step, that's because it's easier to do it in later tasks in the data flow.

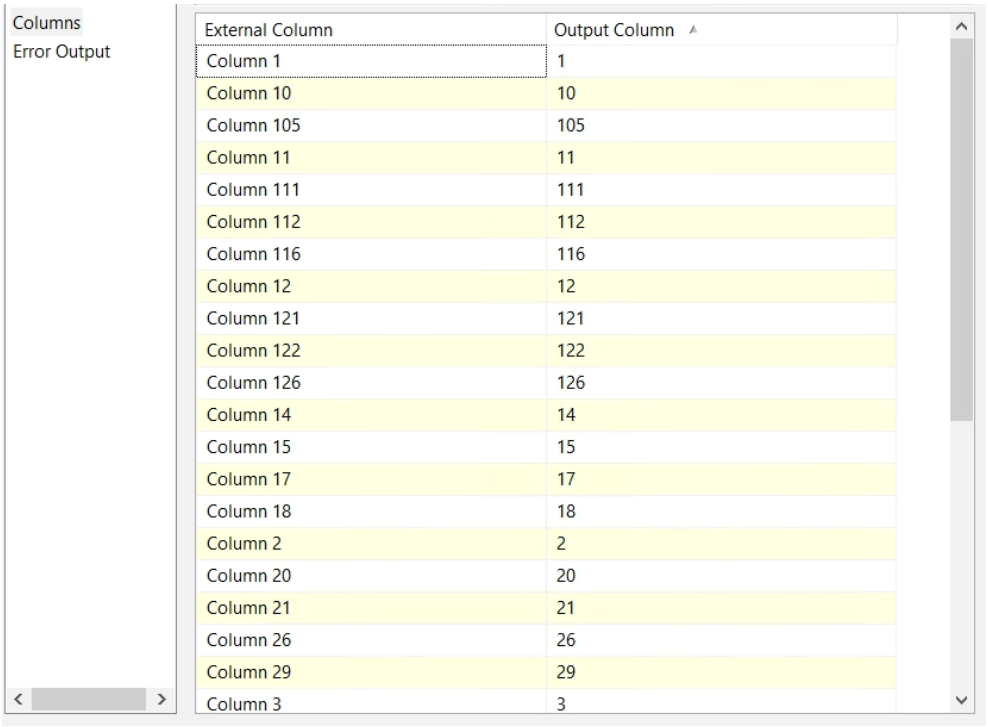


After the flat file connection manager is configured. Click on the file flat source again, the flat file connection manager should be added automatically. Now the focus is to make the desired columns available for further use in the data flow.

In "Columns", make sure that all the desired columns that are going to be used further in the data flow are ticked. This implies for the following columns: 1, 2, 3, 6, 7, 10, 11, 12, 14, 15, 17, 18, 20, 21, 26, 29, 30, 31, 33, 34, 45, 52, 58, 59, 68, 69, 105, 111, 112, 116, 121, 122, 126.

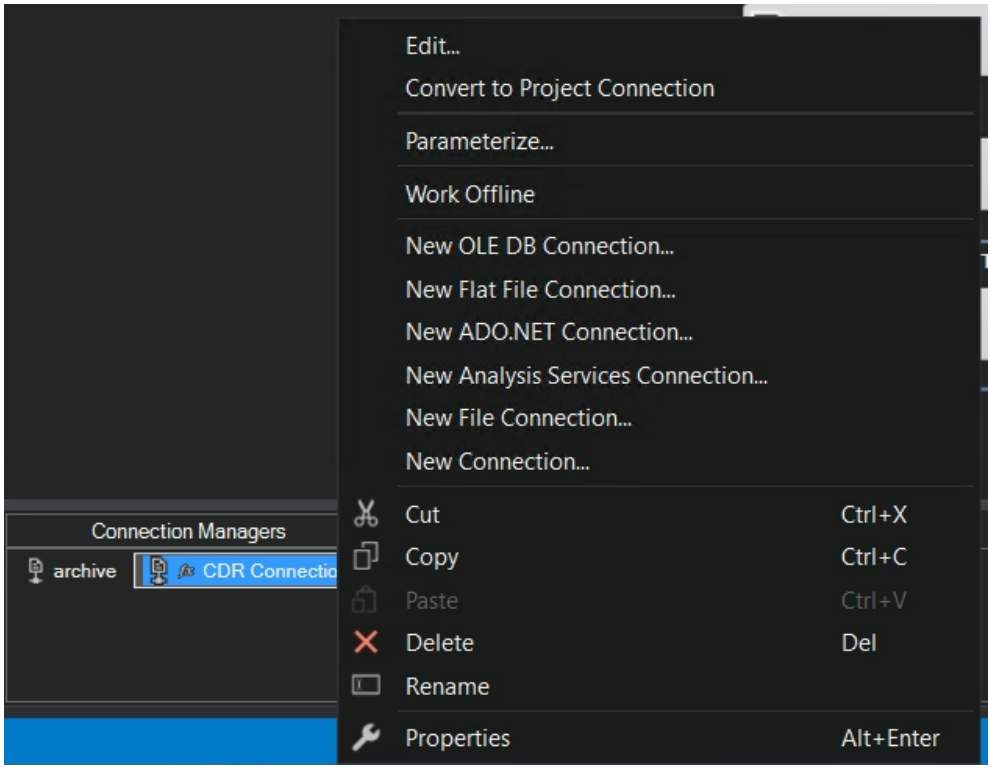
Make sure to change the output column name, this will make it simpler and clearer

when using the columns in other tasks within the data flow. For instance, change Column 1 to 1 etc.

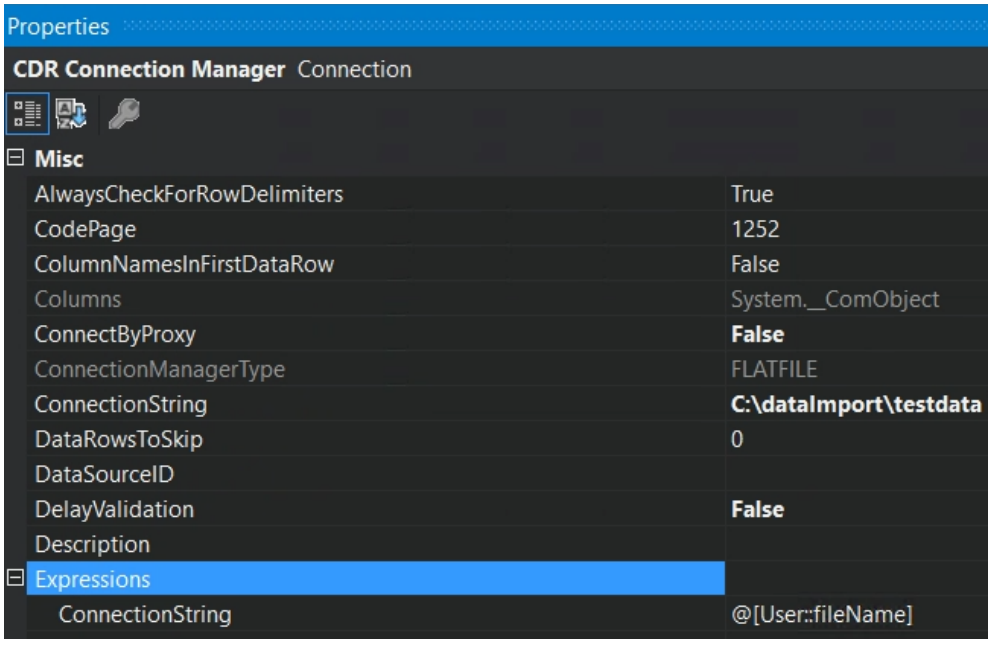


External Column	Output Column
Column 1	1
Column 10	10
Column 105	105
Column 11	11
Column 111	111
Column 112	112
Column 116	116
Column 12	12
Column 121	121
Column 122	122
Column 126	126
Column 14	14
Column 15	15
Column 17	17
Column 18	18
Column 2	2
Column 20	20
Column 21	21
Column 26	26
Column 29	29
Column 3	3

Now we need to make sure that the for loop is going to process each CDR file dynamically. Since the flat file connection manager is already created, right-click on it and click on properties as shown in the figure below.



Make sure "AlwaysCheckForRowDelimiters" are set to True. Under "Expressions", add the fileName variable as a ConnectionString.



Name: Split on Recordtype
Component: Conditional split
Configuration:
Column 1 in the CDR file which is "[1]" now refers to the Recordtype. As a conditional split is necessary to separate STOP and ATTEMPT records for further processing, do the following configuration:

Order	Output Name	Condition
1	STOP	[1] == "STOP"
2	ATTEMPT	[1] == "ATTEMPT"

See [Appendix G.1] to check how the connection from the "Conditional split" task is to the two "Derived column" tasks.

Name: Stop Record
Component: Derived column
Configuration:
Within the Derived column editor for the Stop Record, add "Derived Column Name" and "Expression" as following:

Derived Column Name	Expression	Expression purpose
GatewayName	`[2]`	Obtain the column value directly from the CDR file.
AccountingID	`[3]`	Obtain the column value directly from the CDR file.
Recordtype	`[1]`	Obtain the column value directly from the CDR file.
StartDateTime	`RIGHT([6],4) + "-" + LEFT([6],2) + "-" + SUBSTRING([6],4,2) + " " + [7]`	Uses functions such as LEFT, RIGHT, and SUBSTRING to obtain the correct date from column 6. Column 7 is added as that column represents the time. The date and time is represented in the datetime2 format.
DisconnectDateTime	`RIGHT([11],4) + "-" + LEFT([11],2) + "-" + SUBSTRING([11],4,2) + " " + [12]`	Uses functions such as LEFT, RIGHT, and SUBSTRING to obtain the correct date from column 11. Column 12 is added as that column represents the time. The date and time is represented in the datetime2 format.

CallServiceDuration	`((DT_NUMERIC,10,2) [14]) / 100`	Obtain the column value directly from the CDR file. However, the data type is altered and the value is divided by hundred to get the value in seconds.
CallingNumber	`[20]`	Obtain the column value directly from the CDR file.
CalledNumber	`[21]`	Obtain the column value directly from the CDR file.
RouteLabel	`[29]`	Obtain the column value directly from the CDR file.
IngressTrunkGroup	`[34]`	Obtain the column value directly from the CDR file.
EgressTrunkGroup	`[68]`	Obtain the column value directly from the CDR file.
CallDisconnectReason	`(DT_I8)[15]`	Obtain the column value directly from the CDR file. However, the data type is altered to an eight-byte signed integer for compatibility reasons.
CallDisconnectReasonIngress	`[121] == "" && (DT_I8)[15] == 16 && FINDSTRING([52],"BYE",1) > 0 || FINDSTRING([52],"CAN",1) > 0 ? 200 : [121] == "" ? NULL(DT_I8) : (DT_I8)[121]`	A nested conditional statement that handles several factors. If column 121 is blank, and if CallDisconnectReason contains 16, and if the string "BYE" or "CAN" are found in column 52. Then the column value is set to 200, else, the value is set to `NULL`. If column 121 is not blank, the column keeps the value, for instance, if the value is already 480. That the column value is set to 200 is referring to the SIP response codes, 200 indicates that the call request was successful.

CallDisconnectReasonEgress	`[122] == "" && (DT_I8)[15] == 16 && FINDSTRING([69],"BYE",1) > 0 || FINDSTRING([69],"CAN",1) > 0 ? 200 : [122] == "" ? NULL(DT_I8) : (DT_I8)[122]`	A nested conditional statement which handles several factors. If column 122 is blank, and if CallDisconnectReason contains 16, and if the string "BYE" or "CAN" are found in column 69. Then the column value is set to 200, else, the value is set to `NULL`. If column 122 is not blank, the column keeps the value, for instance, if the value is already 480. That the column value is set to 200 is referring to the SIP response codes, 200 indicates that the call request was successful.
IngressRemoteSignalingIP	`[126]`	Obtain the column value directly from the CDR file.
EgressRemoteSignalingIP	`[33]`	Obtain the column value directly from the CDR file.
InsertDateTime	`GETDATE()`	Return the current database system date and time. Format: YYYY-MM-DD hh:mm:ss.mmm

Name: Attempt Record
Component: Derived column
Configuration:
Within the Derived column editor for the Attempt Record, add "Derived Column Name" and "Expression" as following:

Derived Column Name	Expression	Expression purpose
GatewayName	`[2]`	Obtain the column value directly from the CDR file.
AccountingID	`[3]`	Obtain the column value directly from the CDR file.
Recordtype	`[1]`	Obtain the column value directly from the CDR file.

StartDateTime	`RIGHT([6],4) + "-" + LEFT([6],2) + "-" + SUBSTRING([6],4,2) + " " + [7]`	Uses functions such as LEFT, RIGHT, and SUBSTRING to obtain the correct date from column 6. Column 7 is added as that column represents the time. The date and time is represented in the datetime2 format.
DisconnectDateTime	`RIGHT([6],4) + "-" + LEFT([6],2) + "-" + SUBSTRING([6],4,2) + " " + [10]`	Uses functions such as LEFT, RIGHT, and SUBSTRING to obtain the correct date from column 6. Column 10 is added as that column represents the time. The date and time is represented in the datetime2 format.
CallServiceDuration	`NULL(DT_WSTR,50)`	Since ATTEMPT records does not have any column named CallServiceDuration, the value is set to `NULL`.
CallingNumber	`[17]`	Obtain the column value directly from the CDR file.
CalledNumber	`[18]`	Obtain the column value directly from the CDR file.
RouteLabel	`[26] == "" ? NULL(DT_WSTR,30) : [26]`	Conditional statement. Either obtain the column value directly from the CDR file, if the value is blank, the value is set to `NULL`.
IngressTrunkGroup	`[31] == "" ? NULL(DT_WSTR,23) : [31]`	Conditional statement. Either obtain the column value directly from the CDR file, if the value is blank, the value is set to `NULL`.
EgressTrunkGroup	`[58] == "" ? NULL(DT_WSTR,23) : [58]`	Conditional statement. Either obtain the column value directly from the CDR file, if the value is blank, the value is set to `NULL`.
CallDisconnectReason	`(DT_I8)[12]`	Obtain the column value directly from the CDR file. However, the data type is altered to an eigth-byte signed integer for compatibility reasons.

CallDisconnectReasonIngress	`[111] == "" && (DT_I8)[12] == 16 && FINDSTRING([45],"BYE",1) > 0 || FINDSTRING([45],"CAN",1) > 0 ? 200 : [111] == "" ? NULL(DT_I8) : (DT_I8)[111]`	A nested conditional statement that handles several factors. If column 111 is blank, and if CallDisconnectReason contains 16, and if the string "BYE" or "CAN" are found in column 45. Then the column value is set to 200, else, the value is set to NULL. If column 111 is not blank, the column keeps the value, for instance, if the value is already 480. That the column value is set to 200 is referring to the SIP response codes, 200 indicates that the call request was successful.
CallDisconnectReasonEgress	`[112] == "" && (DT_I8)[12] == 16 && FINDSTRING([59],"BYE",1) > 0 || FINDSTRING([59],"CAN",1) > 0 ? 200 : [112] == "" ? NULL(DT_I8) : (DT_I8)[112]`	A nested conditional statement that handles several factors. If column 112 is blank, and if CallDisconnectReason contains 16, and if the string "BYE" or "CAN" are found in column 59. Then the column value is set to 200, else, the value is set to NULL. If column 112 is not blank, the column keeps the value, for instance, if the value is already 480. That the column value is set to 200 is referring to the SIP response codes, 200 indicates that the call request was successful.
IngressRemoteSignalingIP	`[30] == "" ? NULL(DT_WSTR,39) : [30]`	Conditional statement. Either obtain the column value directly from the CDR file, if the value is blank, the value is set to NULL.
EgressRemoteSignalingIP	`[116]`	Obtain the column value directly from the CDR file.
InsertDateTime	`GETDATE()`	Return the current database system date and time. Format: YYYY-MM-DD hh:mm:ss.mmm

Name: Union All
Component: Union All
Configuration:
Union All Input 1 = STOP Record

Union All Input 2 = ATTEMPT Record

Keep in mind that the ATTEMPT record does not have a CallServiceDuration column and that is why it is set to "<ignore>".

Output Column Name	Union All Input 1	Union All Input 2
StartDateTime	StartDateTime	StartDateTime
RouteLabel	RouteLabel	RouteLabel
Recordtype	Recordtype	Recordtype
InsertDateTime	InsertDateTime	InsertDateTime
IngressTrunkGroup	IngressTrunkGroup	IngressTrunkGroup
IngressRemoteSignalingIP	IngressRemoteSignalingIP	IngressRemoteSignalingIP
GatewayName	GatewayName	GatewayName
EgressTrunkGroup	EgressTrunkGroup	EgressTrunkGroup
EgressRemoteSignalingIP	EgressRemoteSignalingIP	EgressRemoteSignalingIP
DisconnectDateTime	DisconnectDateTime	DisconnectDateTime
CallServiceDuration	CallServiceDuration	<ignore>
CallingNumber	CallingNumber	CallingNumber
CalledNumber	CalledNumber	CalledNumber
CallDisconnectReasonIngress	CallDisconnectReasonIngress	CallDisconnectReasonIngress
CallDisconnectReasonEgress	CallDisconnectReasonEgress	CallDisconnectReasonEgress
CallDisconnectReason	CallDisconnectReason	CallDisconnectReason
AccountingID	AccountingID	AccountingID

Name: Data Conversion
Component: Data Conversion
Configuration:
Within the data conversion task under "Available Input Columns", make sure to include all the relevant columns which are sent from the union all task. This is done by ticking the relevant boxes. Remember to change the data type for the columns and their length as shown in the figure below. This is because of compatibility reasons since SSIS handles data types different than MS-SQL DB.

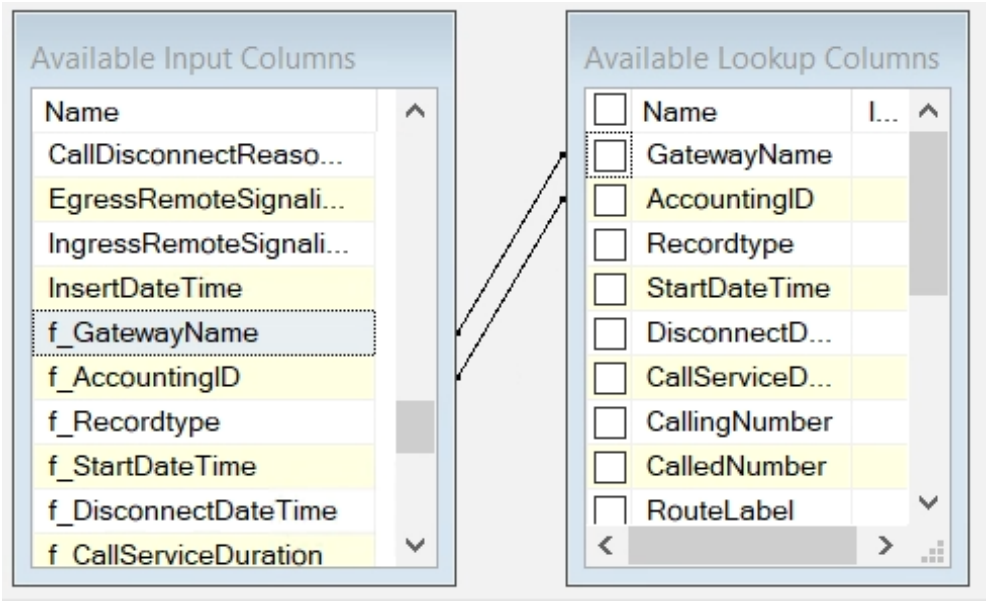
Input Column	Output Alias	Data Type	Length
InsertDateTime	f_InsertDateTime	database timestamp [DT_DBTIMESTAMP]	
StartDateTime	f_StartDateTime	database timestamp with precision [DT_DBTIMESTAMP2]	
DisconnectDateTime	f_DisconnectDateTi...	database timestamp with precision [DT_DBTIMESTAMP2]	
CallDisconnectReas...	f_CallDisconnectRea...	eight-byte signed integer [DT_I8]	
CallDisconnectReas...	f_CallDisconnectRea...	eight-byte signed integer [DT_I8]	
CallDisconnectReas...	f_CallDisconnectRea...	eight-byte signed integer [DT_I8]	
CallServiceDuration	f_CallServiceDuration	eight-byte signed integer [DT_I8]	
Recordtype	f_Recordtype	string [DT_STR]	7
IngressTrunkGroup	f_IngressTrunkGroup	string [DT_STR]	23
IngressRemoteSigna...	f_IngressRemoteSig...	string [DT_STR]	23
EgressRemoteSignal...	f_EgressRemoteSign...	string [DT_STR]	23
EgressTrunkGroup	f_EgressTrunkGroup	string [DT_STR]	23
RouteLabel	f_RouteLabel	string [DT_STR]	23
CallingNumber	f_CallingNumber	string [DT_STR]	30
CalledNumber	f_CalledNumber	string [DT_STR]	30
GatewayName	f_GatewayName	string [DT_STR]	23
AccountingID	f_AccountingID	string [DT_STR]	18

Name: DB lookup
Component: Lookup
Configuration:
Within the lookup component, there are three main configuration options we are going to configure. These are under General, Connection, and Columns.

Under "General", set:
• Cache mode to Full cache.
• Connection type to OLE DB connection manager.
• Handle rows with no matching entries to redirect rows to no match output.

Under "Connection":
• Connect to the desired database by creating a new OLE DB connection manager. The OLE DB connection manager that is created is also going to be used further in the data flow.
• Connect to the desired table within that database.

Under "Columns", set:



The lookup is then done on the primary key which is GatewayName + AccountingID.

Name: Insert CDR data into DB
Component: OLE DB Destination
Configuration:
Within the OLE DB Destination component under "Connection Manager", connect to the desired database by using the OLE DB connection manager that was created in the previous step. Connect to the desired table as well.

Under "Mappings", make sure that the input columns that are used within the package are mapped to the correct destination column in the database as shown in the figure below.

Input Column	Destination Column
f_GatewayName	GatewayName
f_AccountingID	AccountingID
f_Recordtype	Recordtype
f_StartDateTime	StartDateTime
f_DisconnectDateTime	DisconnectDateTime
f_CallServiceDuration	CallServiceDuration
f_CallingNumber	CallingNumber
f_CalledNumber	CalledNumber
f_RouteLabel	RouteLabel
f_IngressTrunkGroup	IngressTrunkGroup
f_EgressTrunkGroup	EgressTrunkGroup
f_CallDisconnectReason	CallDisconnectReason
f_CallDisconnectReasonIngress	CallDisconnectReasonIngress
f_CallDisconnectReasonEgress	CallDisconnectReasonEgress
f_IngressRemoteSignalingIP	IngressRemoteSignalingIP
f_EgressRemoteSignalingIP	EgressRemoteSignalingIP
f_InsertDateTime	InsertDateTime

Name: Update DB
Component: OLE DB Command
Configuration:
Within the OLE DB Command under "Connection Manager", make sure to use the already created OLE DB connection manager to connect to the database. Under "Component Properties", within the SqlCommand option, add the following SQL code:

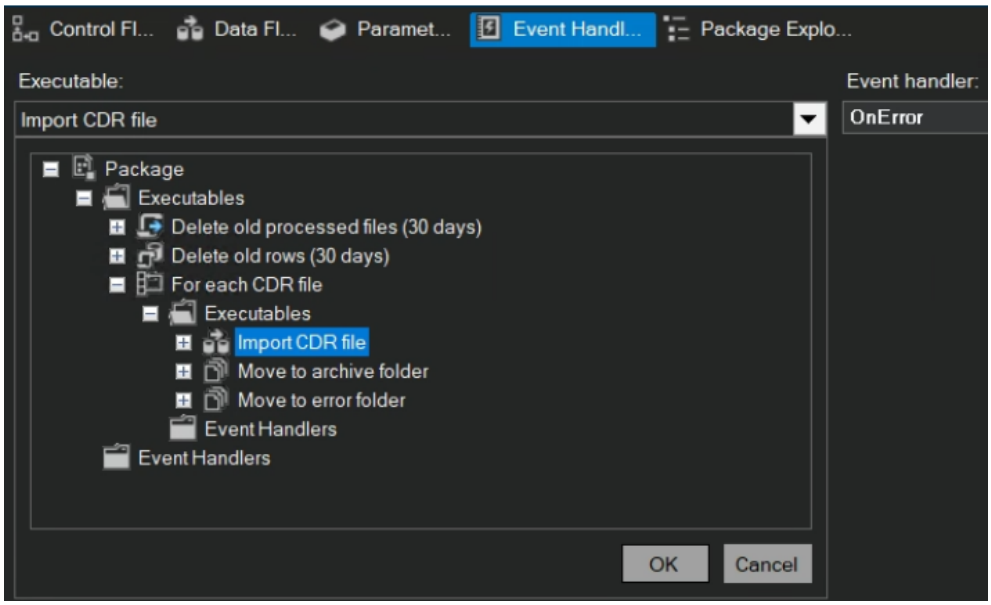
```
update HDO.CDR_RAW
set Recordtype=?, StartDateTime=?, DisconnectDateTime=?, CallServiceDuration=?,
↪  CallingNumber=?, CalledNumber=?, RouteLabel=?, IngressTrunkGroup=?,
↪  EgressTrunkGroup=?, CallDisconnectReason=?, CallDisconnectReasonIngress=?,
↪  CallDisconnectReasonEgress=?, IngressRemoteSignalingIP=?,
↪  EgressRemoteSignalingIP=?, InsertDateTime=? where GatewayName=? and
↪  AccountingID=?
```

Under "Column Mappings", make sure the correct column values are mapped to the correct parameter as shown in the figure below. The parameters are created through the "=?" in the SqlCommand.

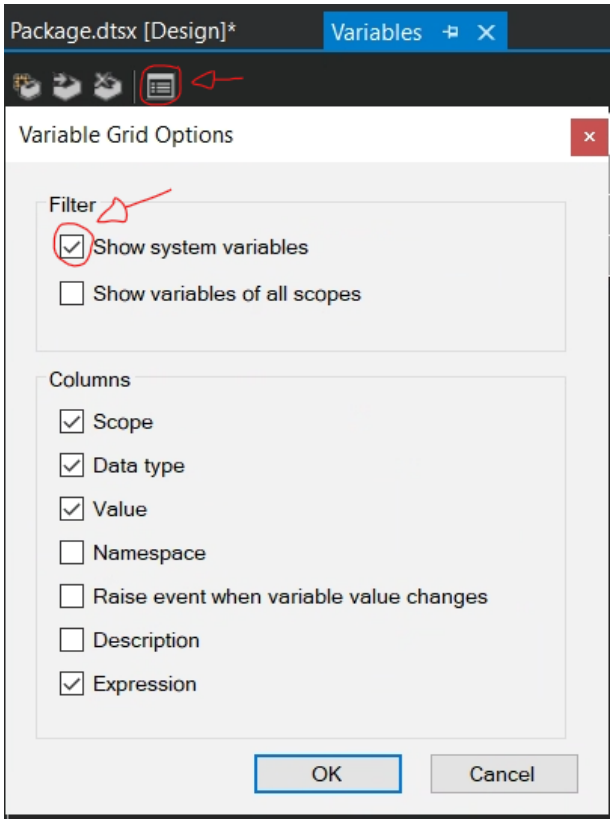
Input Column	Destination Column
f_StartDateTime	Param_1
f_DisconnectDateTime	Param_2
f_CallServiceDuration	Param_3
f_CallingNumber	Param_4
f_CalledNumber	Param_5
f_RouteLabel	Param_6
f_IngressTrunkGroup	Param_7
f_EgressTrunkGroup	Param_8
f_CallDisconnectReason	Param_9
f_CallDisconnectReasonIngress	Param_10
f_CallDisconnectReasonEgress	Param_11
f_IngressRemoteSignalingIP	Param_12
f_EgressRemoteSignalingIP	Param_13
f_InsertDateTime	Param_14
f_GatewayName	Param_15
f_AccountingID	Param_16

G.5 Event handler

Since the SSIS package is processing several CDR files when it is executed, if a minimum of one of those files are processed and it resulted in an error, the package will exit. This means that CDR files that are going to be processed after the error file are not processed at all. The consequence then is that the database does not receive the necessary data. To make sure that the package keep processing all the files even if one file resulted in an error, an event handler needs to be configured. Add an event handler for the data flow task used in the control flow, in this case, it is named "Import CDR file", make sure it is set to "OnError".



Under Variables -> Grid Options -> Show system variables must be ticked:

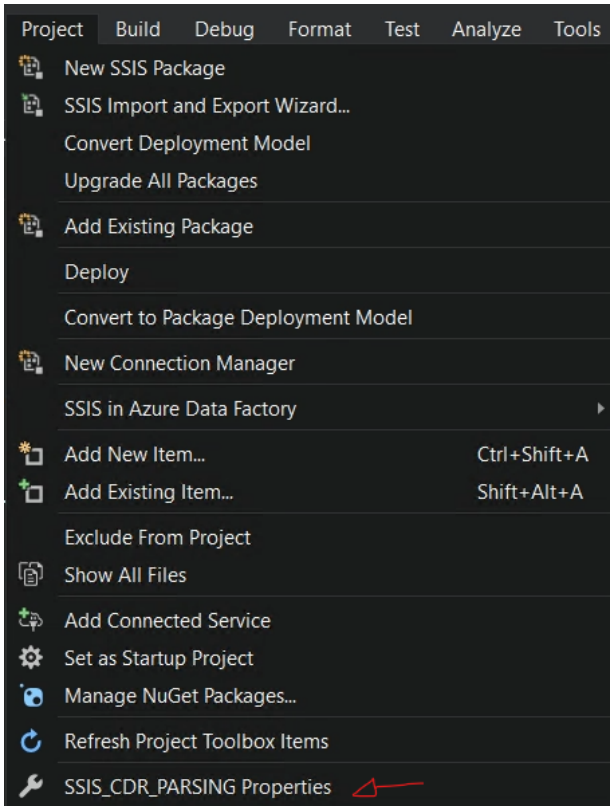


For the system variable Propagate, set the value to False:

Propagate	OnError	Boolean	False

G.6 Security options

At the project level under the properties settings. Set ProtectionLevel to DontSave-Sensitive.

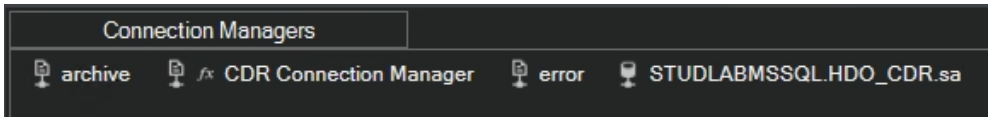


At the package level within the control flow, right-click on the background and select properties. Scroll down and set ProtectionLevel to DontSaveSensitive.

That the ProtectionLevel is set to "DontSaveSensitive" means that sensitive information is not written to the XML file for the package upon saving the package. See subsection 1.2.2 regarding that security is not part of the scope for this project.

G.7 Connection managers

There are in total four connection managers:



These connection managers are used within this project to access external resources such as flat files, folders, and databases for read and write operations. The connection managers are essential for the package to operate as we intend it to do. They are created throughout the configuration of the control flow and the data flow, and they can always be altered afterward if necessary.

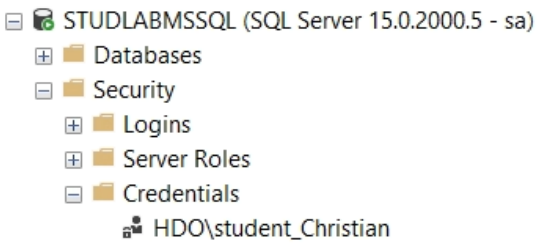
Appendix H

SQL Agent Job - SSIS Package

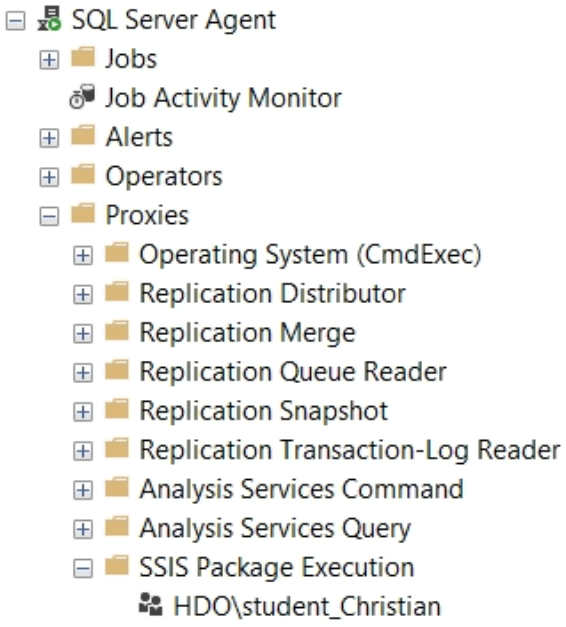
Required running windows services:
- SQL Server
- SQL Server Agent
- SQL Server Integration Services

Make sure that the SSIS package executes without any errors in the visual studio environment before proceeding (manual job). In SSMS, we are logged in as SA. Since the user SA did not create the SSIS package, we need to use a proxy account.

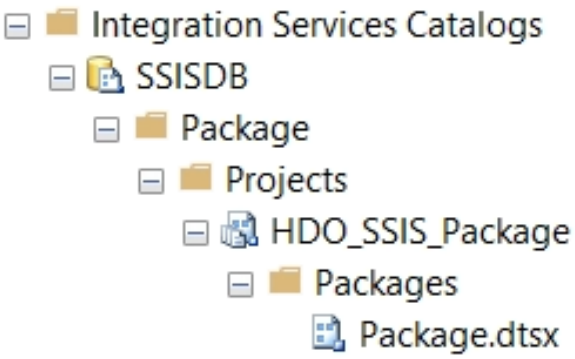
1. Create a new credential. Enter credential name, identity, and password.



2. Create a new proxy. Enter proxy name, for the credential name use the credential created previously. Tick off the box named "SQL Server Integration Services Package".

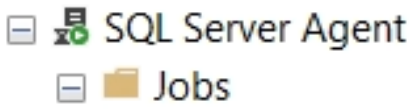


3. Have the following folder hierarchy (enter desired names for the new folders if necessary):



Under "Projects", select Import Packages. Follow the integration services project conversion wizard, the process is straightforward.
• Select the source folder where the package is located, select the package afterward.
• When entering the project name, set the protection level to "DontSaveSensitive".
• No additional configuration is necessary, follow the steps as normal.

4. Create a new job for the SQL Server Agent.



5. For a basic SQL Agent Job, we are only configuring the pages named:
• General
• Steps
• Schedules

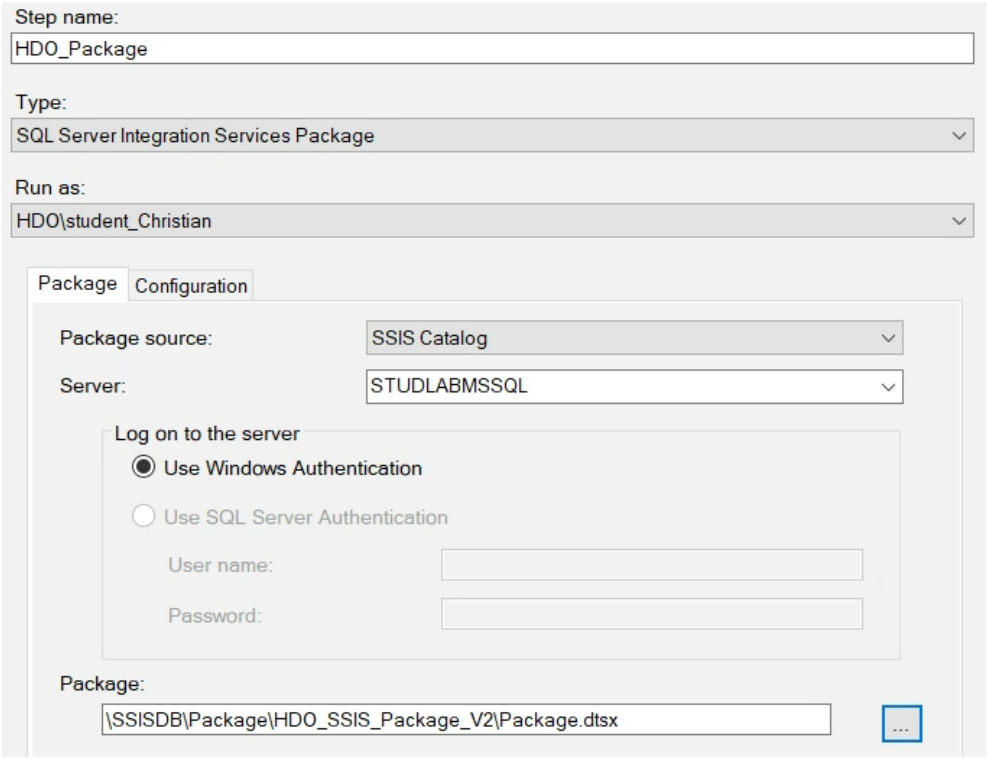
6. In "General", enter desired name, the SA user as owner. Make sure the box named "Enabled" is ticked.

7. In "Steps", create a new Step.

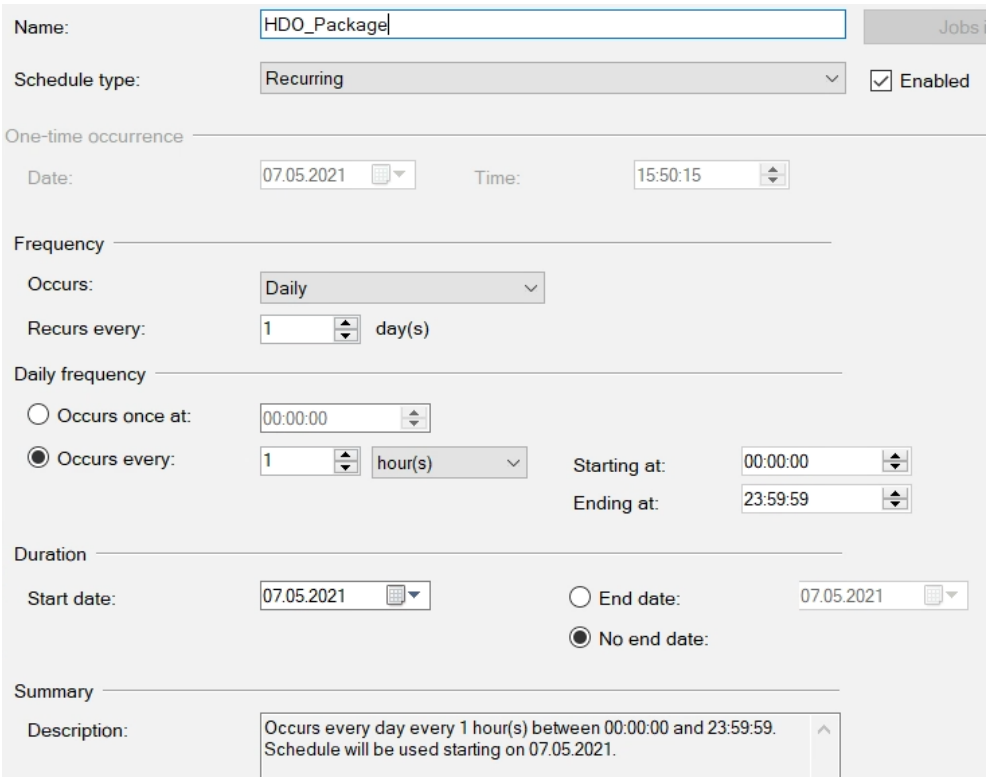
Make sure to:
• Change the "Run as" option to run the package as the proxy account.
• Change "Type" to SQL Server Integration Services Package
• As a package source, use the SSIS catalog. This is because the package is stored in the SSIS catalog (because of bullet point 3). Select the correct package to use.

Reminder: This configuration process might be bugged. Sometimes when selecting the SSIS Catalog as a package source it can't find the server in the drop-down list. If that is the case, select another package source, if the server appears in the drop-down list, select it. And then change the package source to the SSIS Catalog again. Then add the package using the three dots option.

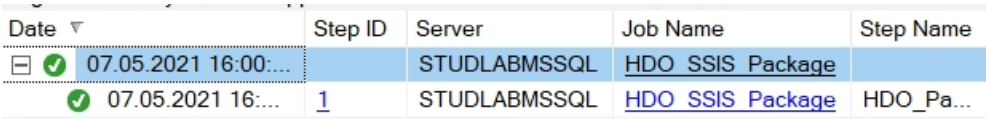
In the Advanced page, set:
• On success action to "Quit the job reporting success".
• On failure action to "Quit the job reporting failure".



8. In "Schedules", create a new schedule. This schedule will execute the job every day, each hour with no end date.



9. For the job that is created. Right-click on it and select "View History" to check if it ran successfully. Make sure that the job is Enabled.
Successful job:



Appendix I

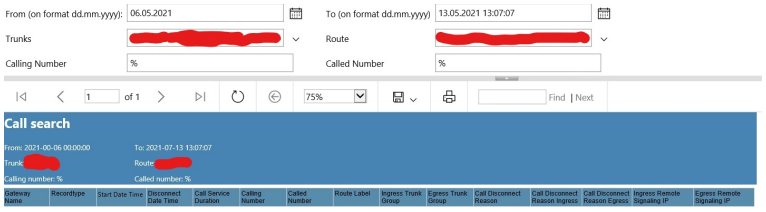
The SSRS reports and templates

The reports headers have the color *SteelBlue* in the background and the color *Ivory* in writing. The tables have the colors: in the background of the top text *#588bae*, in the background of the body is the color *AliceBlue*, and the writing is in black.

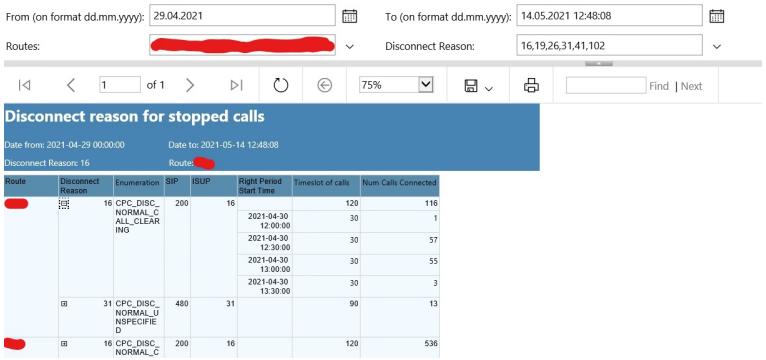
I.1 The reports

I.1.1 Call search

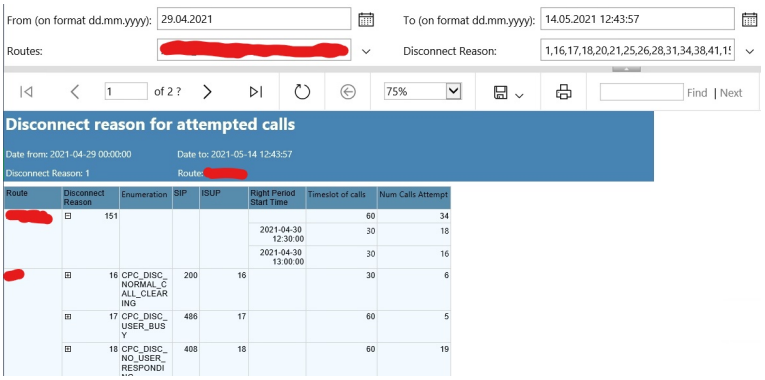
Because of the information that this report contains, it was chosen a date where it had no records.



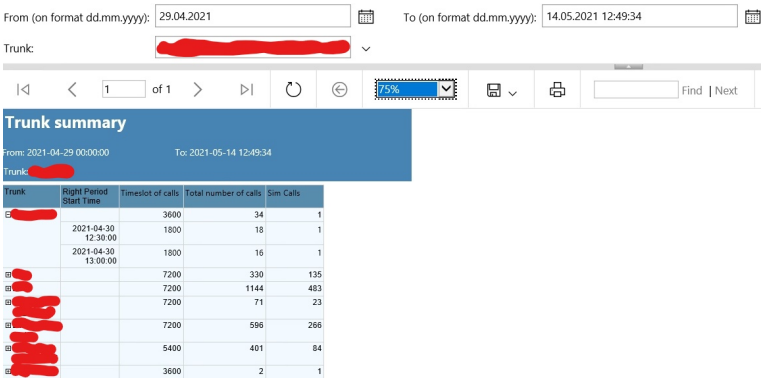
I.1.2 Disconnect reason for stopped calls



I.1.3 Disconnect reason for attempted calls

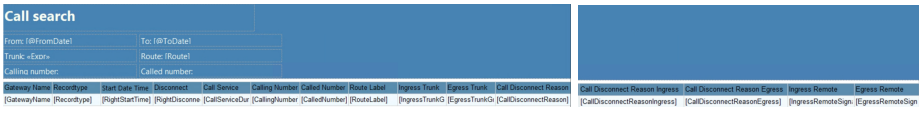


I.1.4 Trunk Summary

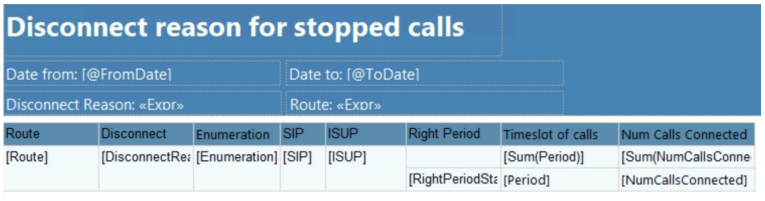


I.2 The templates

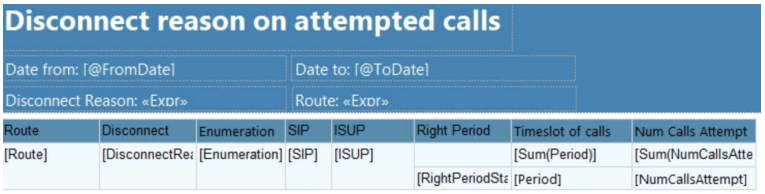
I.2.1 Call search



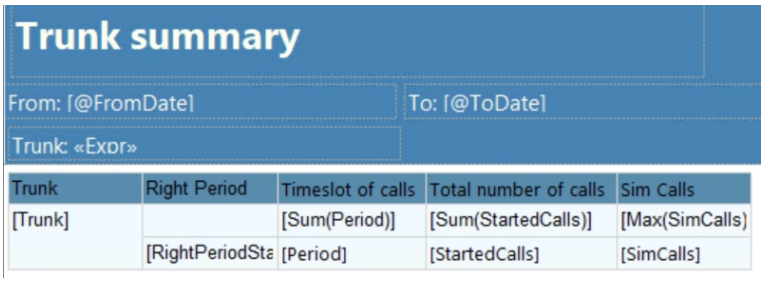
I.2.2 Disconnect reason for stopped calls



I.2.3 Disconnect reason for attempted calls



I.2.4 Trunk Summary



Appendix J

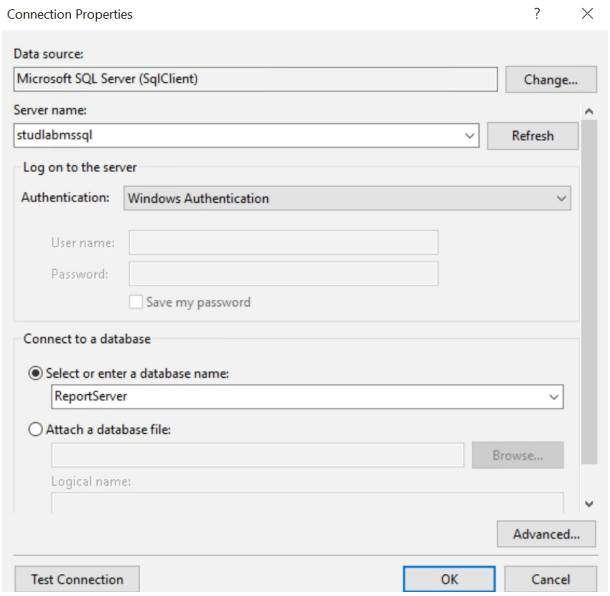
SSRS Configuration

J.1 Start a project

The first step is to open Visual studio, click on "File" in the top left corner, then click on new, and project. Search for “Report Server Project” in the search box. Then click on the box with the name “Report Server Project” and change the name or location if needed, then create.

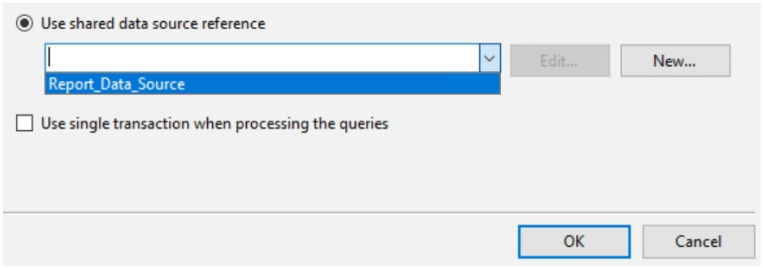
J.2 Data source

Right click on the "Data Sources" folder on the Report Data box to the left, click on "Use shared data source reference", then on "New...". Choose the Type of the Data source and click on "build", then fill in the Server name, and a database name, like in the picture of the data source configured to the reports:



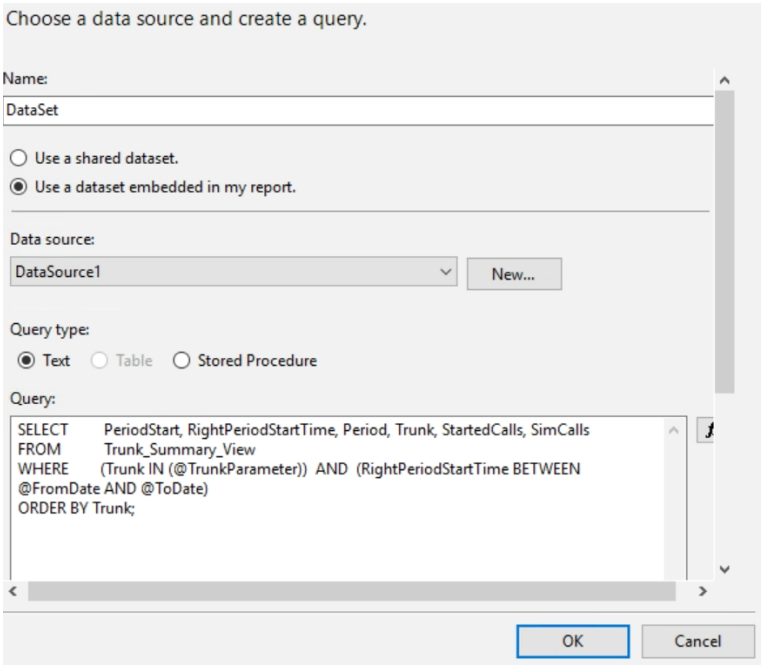
Then click OK and OK again.

For the reports: right click on the Data sources folder, then on "Add Data Source...", then click on "Use shared data source reference" and click on the down pointing arrow, then on the name of the shared data source and OK, like in the figure below:



J.3 Data sets

Right-click on the "Datasets" folder, click on add Dataset, click on "Use embedded in my report", and choose a name. Then paste a chosen query based on which report the data set are made in; the queries are below the picture. The data set in the picture is made to the Trunk Summary report.



In the text window, use a query, the queries used for the data sets in this report are the four:

Call search

```
SELECT TOP(1000) GatewayName, AccountingID, Recordtype, RightStartTime,
↪  RightDisconnectDateTime, CallServiceDuration, CallingNumber, CalledNumber,
↪  RouteLabel, IngressTrunkGroup, EgressTrunkGroup,
↪  CallDisconnectReason,CallDisconnectReasonIngress, CallDisconnectReasonEgress,
↪  IngressRemoteSignalingIP, EgressRemoteSignalingIP, RightInsertDateTime
FROM CDR_RAW_View

WHERE ((EgressTrunkGroup IN (@TrunkGroup)) OR (IngressTrunkGroup IN (@TrunkGroup)))
↪  AND  (RouteLabel IN (@RouteParameter)) AND (CallingNumber LIKE
↪  '%'+@CallingNumberParameter+'%') AND  (CalledNumber LIKE
↪  '%'+@CalledNumberParameter+'%') AND (RightStartTime BETWEEN @FromDate AND
↪  @ToDate)
```

Disconnect reason stopped calls

```
SELECT DisconnectReason, NumCallsConnected , RightPeriodStartTime, Period, Route,
↪  Enumeration, ISUP, SIP
FROM Route_Summary_With_Discause_View
WHERE (NumCallsConnected > 0) AND (DisconnectReason IN
↪  (@DisconnectReasonParameter)) AND (Route IN (@RouteParameter)) AND
↪  (RightPeriodStartTime BETWEEN @FromDate AND @ToDate)

ORDER BY Route;
```

Disconnect reason attempted calls

```
SELECT DisconnectReason, NumCallsAttempt, RightPeriodStartTime, Period, Route,
↪  Enumeration, ISUP, SIP
FROM Route_Summary_With_Discause_View
WHERE (NumCallsAttempt > 0) AND (DisconnectReason IN (@DisconnectReasonParameter))
↪  AND (Route IN (@RouteParameter))
AND (RightPeriodStartTime BETWEEN @FromDate AND @ToDate)

ORDER BY Route;
```

Trunk Summary

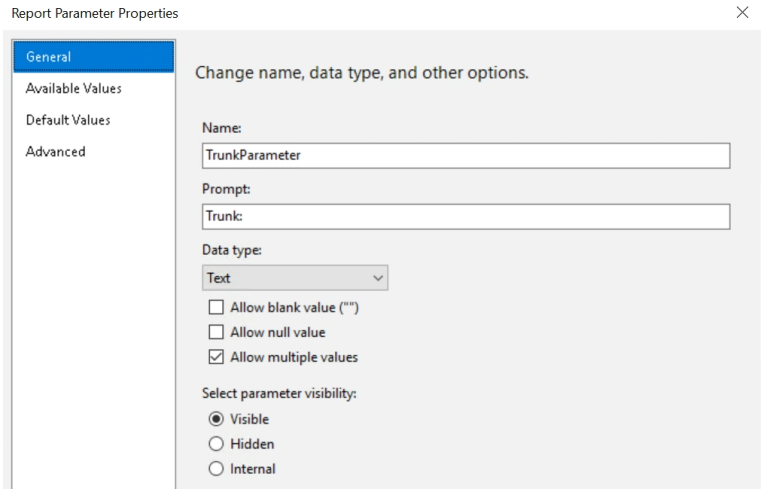
```
SELECT PeriodStart, RightPeriodStartTime, Period, Trunk, StartedCalls, SimCalls
FROM Trunk_Summary_View
WHERE (Trunk IN (@TrunkParameter))  AND  (RightPeriodStartTime BETWEEN @FromDate
↪  AND @ToDate)
ORDER BY Trunk;
```

J.4 Parameters

When running the queries from the data set [Appendix J.3], the parameters will be added in the folder parameters. For all the parameters: right click on the parameters name in the parameters folder, click on the "Parameter Prorteties", then check if the parameters is set to be the right type.

J.4.1 Multiple value

For allowing multiple values to the parameters, find the Parameters folder and right click on the parameters name then open Parameter properties and check on the mark "Allow multiple values". In the reports, this is done in the parameters: Route, Trunk and Disconnect reason.



J.4.2 Drop Down lists

For the parameters with multiple values, a drop down list is made. The list is made by making an embedded data set for each parameter and selecting distinct values as seen in the codes below.

Call search's drop down datasets

```
SELECT DISTINCT IngressTrunkGroup FROM CDR_RAW_View
WHERE  IngressTrunkGroup IS NOT NULL
UNION
SELECT DISTINCT EgressTrunkGroup FROM CDR_RAW_View
WHERE  EgressTrunkGroup IS NOT NULL
ORDER BY IngressTrunkGroup;
```

```
SELECT DISTINCT RouteLabel
FROM   CDR_RAW_View
WHERE  RouteLabel IS NOT NULL
ORDER BY RouteLabel;
```

Disconnect reason for stopped calls' drop down datasets

```
SELECT DISTINCT Route
FROM   Route_Summary_With_Discause_View
WHERE  (NumCallsConnected > 0)  AND (Route IS NOT NULL)
ORDER BY Route;
```

```
SELECT DISTINCT DisconnectReason
FROM   Route_Summary_With_Discause_View
WHERE  (NumCallsConnected > 0)  AND (DisconnectReason IS NOT NULL)
ORDER BY DisconnectReason;
```

Disconnect reason for attempted calls' drop down datasets

```
SELECT DISTINCT Route
FROM   Route_Summary_With_Discause_View
WHERE  (NumCallsAttempt > 0)  AND (Route IS NOT NULL)
ORDER BY Route;
```

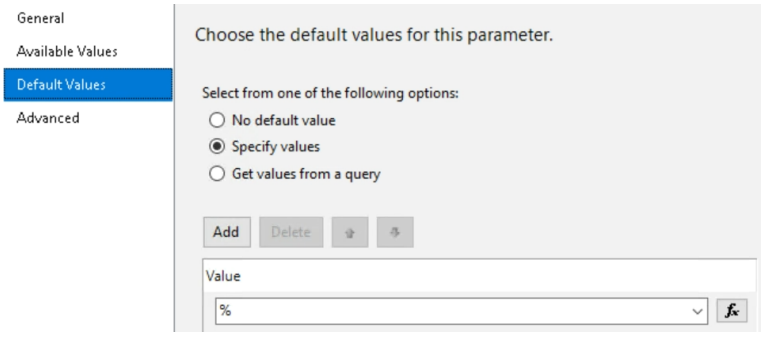
```
SELECT DISTINCT DisconnectReason
FROM   Route_Summary_With_Discause_View
WHERE  (NumCallsAttempt > 0)  AND (DisconnectReason IS NOT NULL)
ORDER BY DisconnectReason;
```

Trunk Summary's drop down datasets

```
SELECT DISTINCT Trunk
FROM Trunk_Summary_View
WHERE  Trunk IS NOT NULL
ORDER BY Trunk;
```

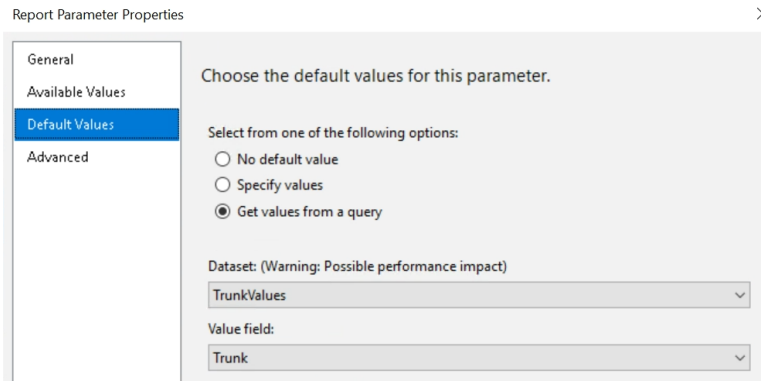
J.4.3 Default value

To set the default value for the wildcard and the date, open Parameter properties and click on default values and choose "Specify values" and click on "Add", like in the picture below. For the wildcards set the value to %.For the dates, set the value of "From date" 7 days before today by writing "=DateAdd("d",-7,Today())" in the value box and for the "To date" choose the default value "=NOW".



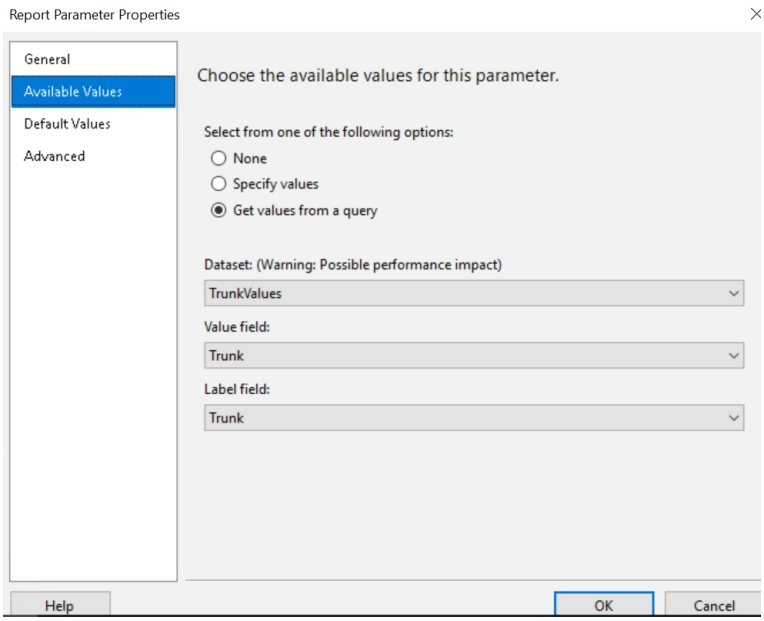
Since the rest of the parameters uses drop down lists, open Parameter properties and click on default values and choose "Get values from a query" and click on the

boxes and choose the dataset that were made for the parameter and choose the Value field for the parameter.



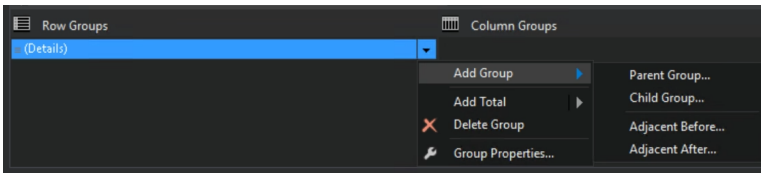
J.4.4 Available value

The only parameters that uses available values in these reports, are the ones that uses drop down lists. Open Parameter properties and click on Available Values and choose "Get values from a query" and click on the boxes and choose the dataset that were made for the parameter and choose the Value field and Label field for the parameter.

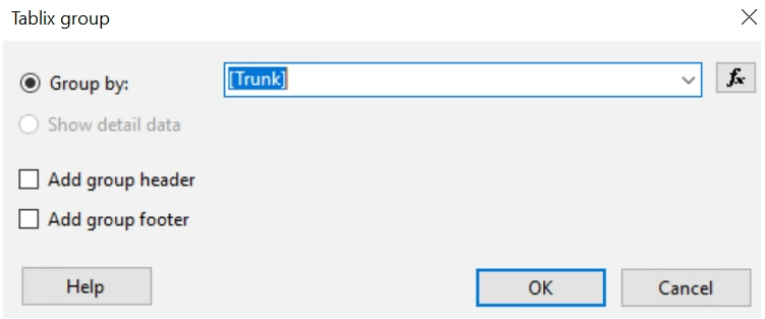


J.5 Grouping tables and adding total

Go to “Row Groups” under the edit report page, then click on “(Details)” in Row Groups, and Add Group. Here you can choose to add “Parent Group” or “Child group”. In the report Trunk Summary it uses the trunk as a Parent group, and it has no child group. A “Child Group” is always next to the “Parent Group”, an “Child Group” is set the reports "Disconnect reason for stopped calls" and in "Disconnect reason for attempted calls", where the “Enumeration”, “SIP”, and “ISUP” is all “Child Groups” of “Disconnect reason”, which is a “Parent Group” under the "Parent Group" "Route". Click on either parent or child group.

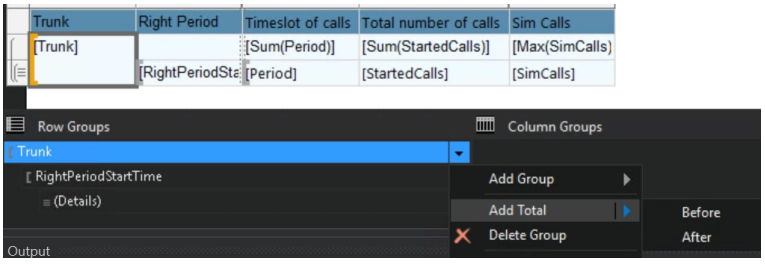


Then the “Tablix group” window pops up, choose the field/column or Expression to group by.



J.5.1 Adding total

To show sum, or max in the report, go to the “Row Groups” and click on the group you want the numbers for, and click on “Add Total”. Here it you can choose “Before” and “After”. “Before” shows the results before a groups text starts, and is used in the reports: Disconnect reason for stopped calls, Disconnect reason for attempted calls and Trunk Summary.

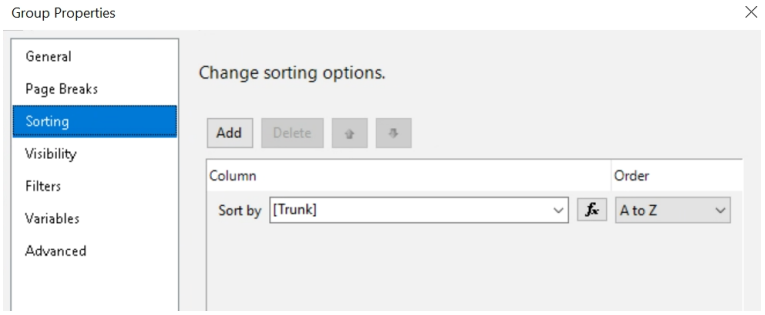


After clicking on “Before” or “After”, it automatically comes up as [Sum(...)] on the rows that have a numeric value. However, this can be changed to others like MAX, which is used in the trunk Summary report for SimCalls.

J.5.2 Sorting in groups

In Row groups, click on the group that you want to be hidden and open group properties.

In group properties; click on "Sorting", then "Add", and choose what to sort by.

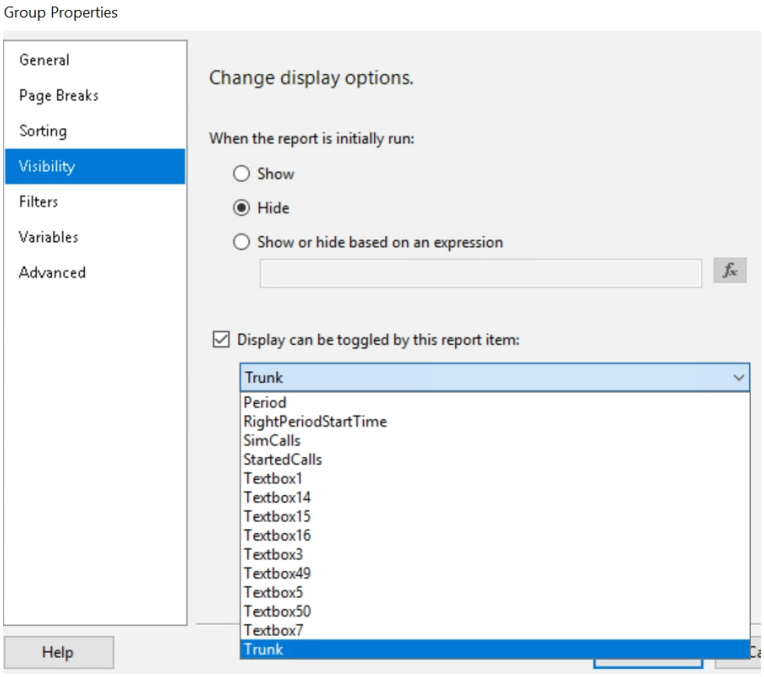


Trunk summary are sorted by Trunk, both Disconnect reason for attempted calls and Disconnect reason for stopped calls are sorted by routes in the route group and further sorted by Disconnect reason.

J.6 Enable drill down

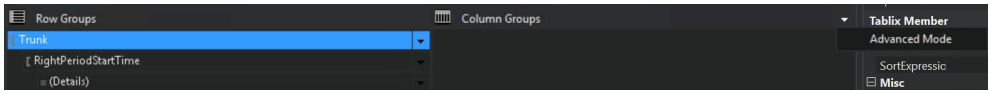
In Row groups, click on the group that you want to be hidden and open group properties.

In group properties; go to visibility, and choose "Hide" then check off the mark before the text "Display can be toggled by this report item". Then click on the down pointed array and choose an item. This item is where the choice to show the rows will be.

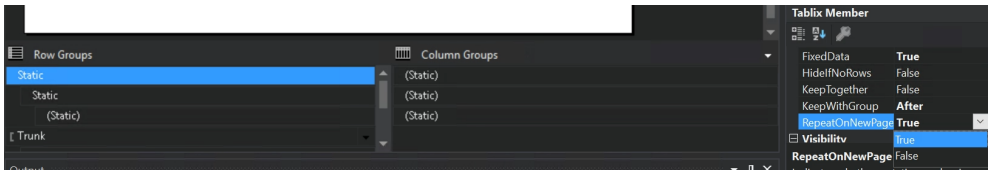


J.7 Keep top text row on new page and while scrolling

Click on the arrow beside columns groups, and click on advanced mode, as seen in the picture below.



Click on the upper text (Static), then go to the properties on the side.



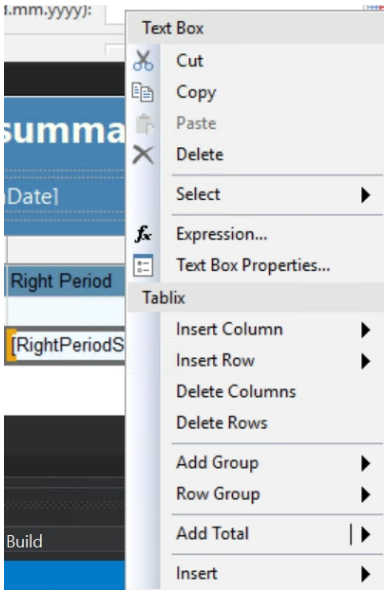
To make the top text appear on each page; click on RepeatOnNewPage and change to True. To make the top text to appear on the screen while scrolling down in the report; scroll up in propertis until the FixedData comes up, then change to True.

J.8 Show only top 1000 rows

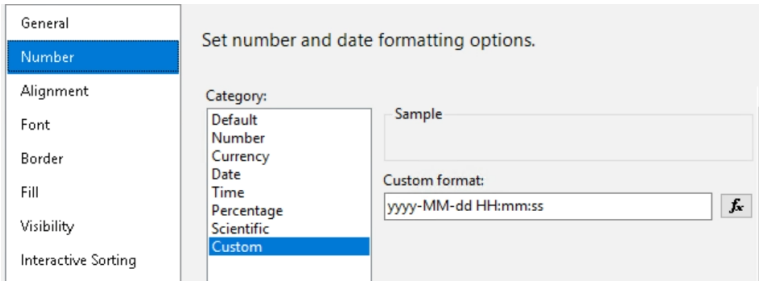
There are different ways to do this, the way done in this report is by opening the dataset, go to the query and write TOP(1000) after SELECT. Like in section J.3 in the query for Call search.

J.9 Change how to display date/time

Right click on the box that contains the date and/or time, then click on "Text Box Properties".



Click on the Number box and choose a category and a date/time setup. In this report the date and time is displayed on the format: yyyy-MM-dd HH:mm:ss

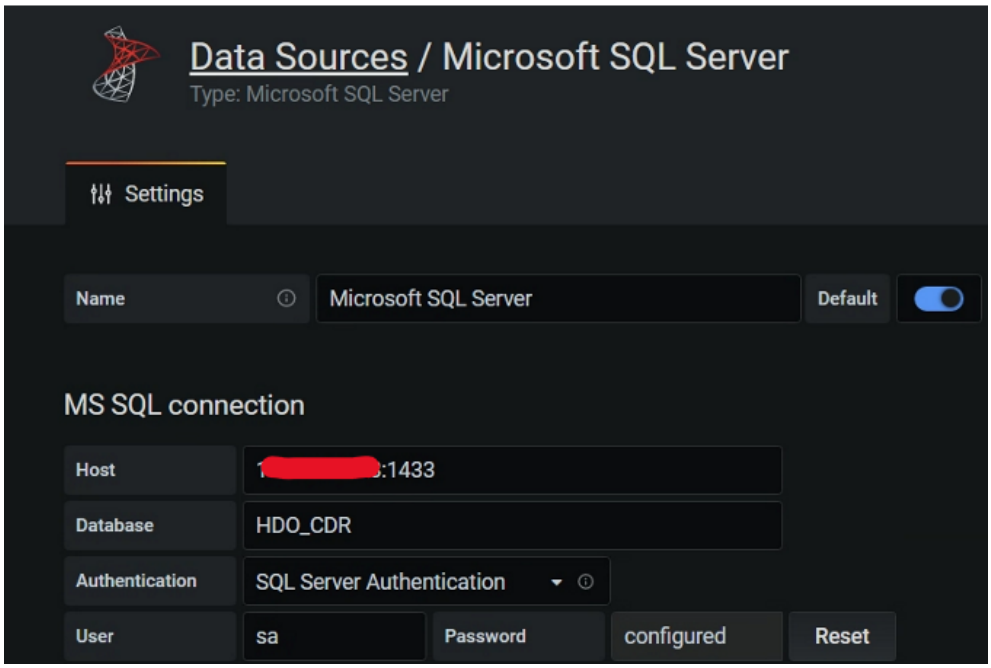


Appendix K

Grafana Configuration

K.1 Data Source

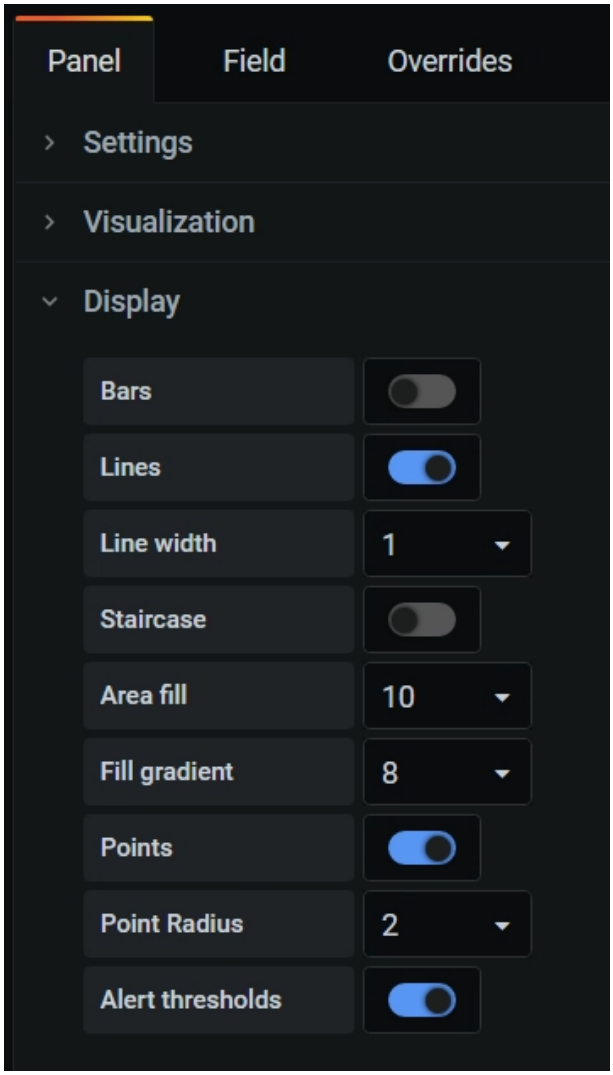
Within Grafana, add the relevant data source:
• Host implies to the IP address + port number of the machine running Grafana
• Port 1433 is representing MS-SQL



K.2 Dashboard

Create a dashboard, within that, add in total eight panels. The following implies to all the panels:
• "Display" configuration is set to the following for every panel except panel 2

(Recordtype). Keep "Display" default for panel 2.



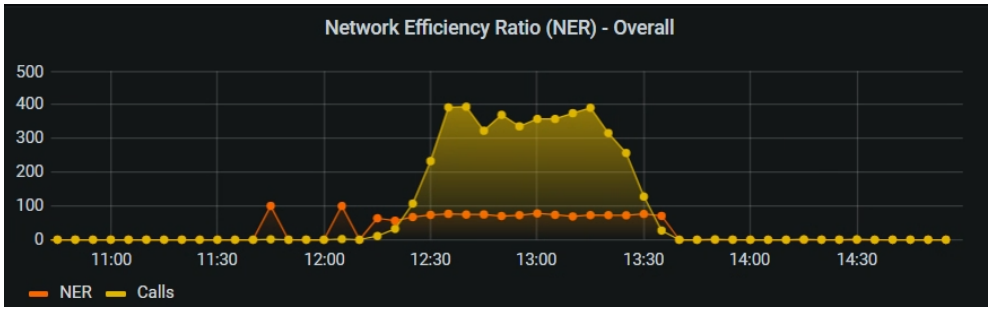
The following implies to all the panels:
• For visualization of the query, "Graph" is used for every panel except panel 2 (Recordtype). In panel 2, "Bar gauge" is used.

Network Efficiency Ratio (NER) - Overall:

- Panel 1
- Configuration:

```
SELECT
  $__timeGroup(StartDateTime, '5m', 0) AS time,
100 * ((sum(case when CallDisconnectReason = 16 then 1 else 0 end) +
        sum(case when CallDisconnectReason = 17 then 1 else 0 end) +
        sum(case when CallDisconnectReason = 18 then 1 else 0 end) +
        sum(case when CallDisconnectReason = 19 then 1 else 0 end) +
        sum(case when CallDisconnectReason = 21 then 1 else 0 end))
        / ((CAST((count(Recordtype)) as FLOAT)))) as NER,
        (count(Recordtype)) as Calls
FROM HDO.CDR_RAW
WHERE $__timeFilter(StartDateTime)
GROUP BY $__timeGroup(StartDateTime, '5m')
ORDER BY 1
```

Resulting panel:



Recordtype:

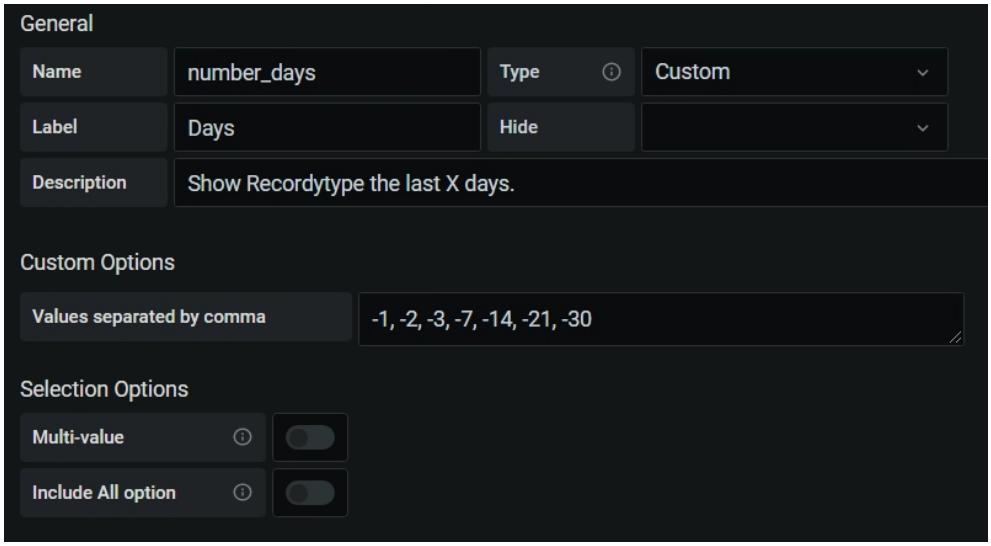
- Panel 2
- Configuration:

```
SELECT count(Recordtype) as Total_Calls,
sum(case when Recordtype = 'STOP' then 1 else 0 end) AS STOP,
sum(case when Recordtype = 'ATTEMPT' then 1 else 0 end) AS ATTEMPT
FROM HDO.CDR_RAW
WHERE StartDateTime >= DATEADD(DAY, $number_days, GETDATE())
AND StartDateTime <= GETDATE()
```

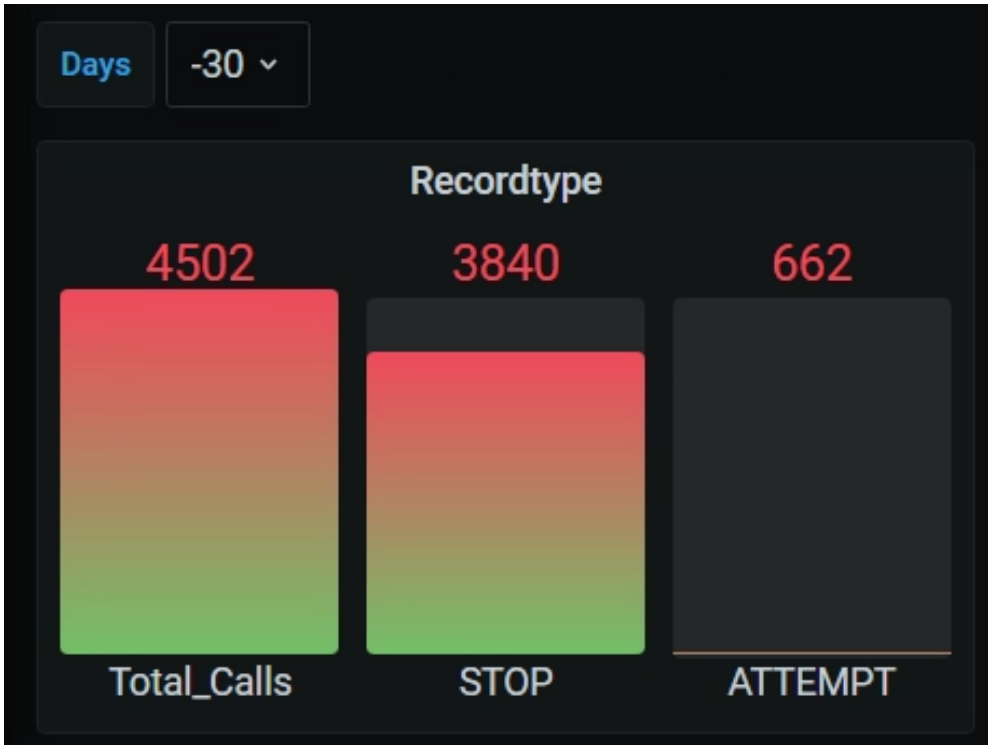
The goal of the Recordtype panel is to visualize gauges that represent total calls, then show how many of the total calls were STOP records and how many were ATTEMPT records. The panel must be dynamic to display data in certain time intervals, the intervals are 1, 2, 3, 7, 14, 21, and 30 days. To make that possible, a dashboard variable is created with the necessary values. In this case, the variable is named "number_days" and it is used within the query as shown in the configuration above.

Within the dashboard, select: dashboard settings -> variables -> new

- Configure the variable as following:



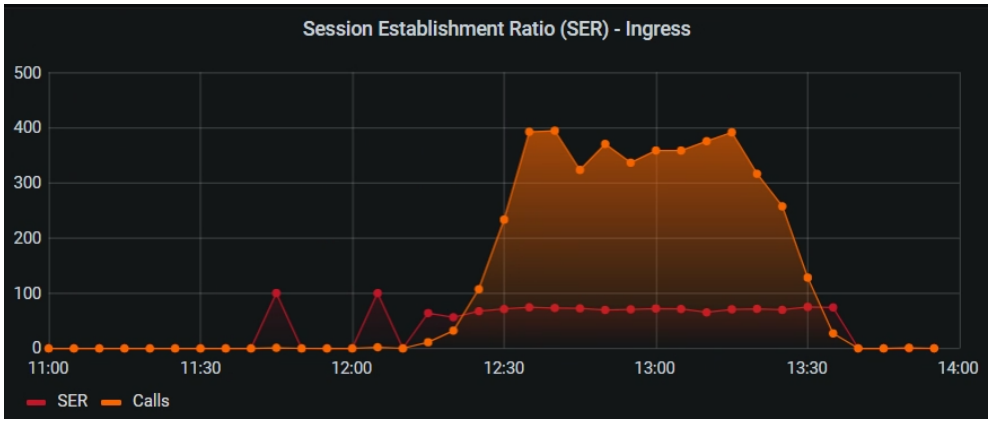
Resulting panel:
- Show Recordtype data for the last 30 days



Session Establishment Ratio (SER) - Ingress:
- Panel 3
- Configuration:

```
SELECT
  $__timeGroup(StartDateTime, '5m', 0) as time,
  100 * ((sum(case when CallDisconnectReasonIngress = 200 then 1 else 0 end)) /
  ((CAST((count(Recordtype)) as FLOAT)) - sum(CASE WHEN
  ↪  CAST(LEFT(CallDisconnectReasonIngress, 1) AS INT) = 3 then 1 else 0 end))) as
  ↪  SER,
  (count(Recordtype)) as Calls
FROM HDO.CDR_RAW
WHERE $__timeFilter(StartDateTime)
GROUP BY $__timeGroup(StartDateTime, '5m')
ORDER BY 1
```

Resulting panel:

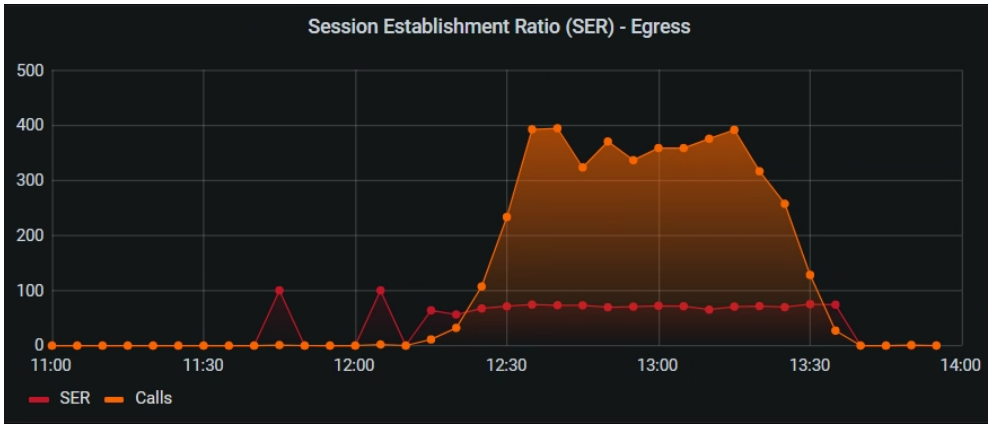


Session Establishment Ratio (SER) - Egress:

- Panel 4
- Configuration:

```
SELECT
  $__timeGroup(StartDateTime, '5m', 0) as time,
  100 * ((sum(case when CallDisconnectReasonEgress = 200 then 1 else 0 end)) /
  ↪  ((CAST((count(Recordtype)) as FLOAT)) - sum(CASE WHEN
  ↪  CAST(LEFT(CallDisconnectReasonEgress, 1) AS INT) = 3 then 1 else 0 end))) as
  ↪  SER,
  (count(Recordtype)) as Calls
FROM HDO.CDR_RAW
WHERE $__timeFilter(StartDateTime)
GROUP BY $__timeGroup(StartDateTime, '5m')
ORDER BY 1
```

Resulting panel:

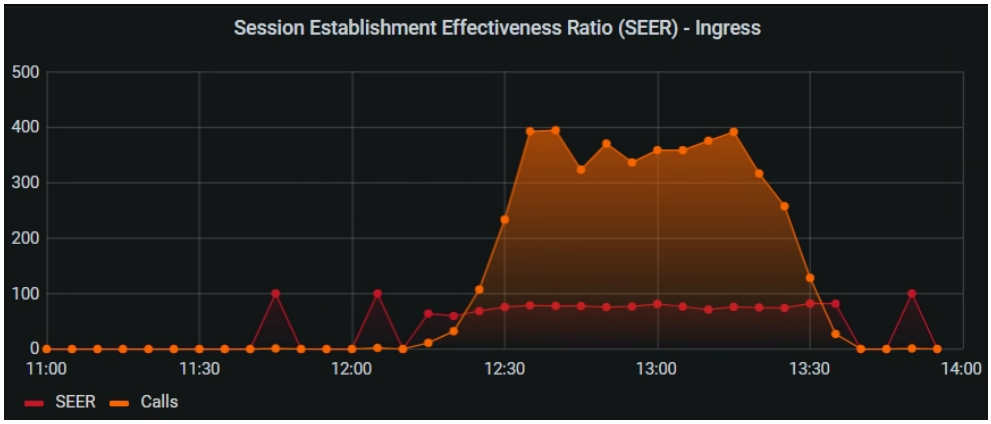


Session Establishment Effectiveness Ratio (SEER) - Ingress:

- Panel 5
- Configuration:

```
SELECT
  $__timeGroup(StartDateTime, '5m', 0) as time,
   100 * ((sum(case when CallDisconnectReasonIngress = 200 then 1 else 0 end) +
    sum(case when CallDisconnectReasonIngress = 480 then 1 else 0 end) +
    sum(case when CallDisconnectReasonIngress = 486 then 1 else 0 end) +
    sum(case when CallDisconnectReasonIngress = 600 then 1 else 0 end) +
    sum(case when CallDisconnectReasonIngress = 603 then 1 else 0 end)) /
    ↪  ((CAST((count(Recordtype)) as FLOAT)) -
    (sum(CASE WHEN CAST(LEFT(CallDisconnectReasonIngress, 1) AS INT) = 3 then 1
    ↪  else 0 end)))) as SEER,
    (count(Recordtype)) as Calls
FROM HDO.CDR_RAW
WHERE $__timeFilter(StartDateTime)
GROUP BY $__timeGroup(StartDateTime, '5m')
ORDER BY 1
```

Resulting panel:



Session Establishment Effectiveness Ratio (SEER) - Egress:

- Panel 6
- Configuration:

```
SELECT
  $__timeGroup(StartDateTime, '5m', 0) as time,
    100 * ((sum(case when CallDisconnectReasonEgress = 200 then 1 else 0 end) +
    sum(case when CallDisconnectReasonEgress = 480 then 1 else 0 end) +
    sum(case when CallDisconnectReasonEgress = 486 then 1 else 0 end) +
    sum(case when CallDisconnectReasonEgress = 600 then 1 else 0 end) +
    sum(case when CallDisconnectReasonEgress = 603 then 1 else 0 end)) /
    ↪  ((CAST((count(Recordtype)) as FLOAT)) -
    (sum(CASE WHEN CAST(LEFT(CallDisconnectReasonEgress, 1) AS INT) = 3 then 1 else
    ↪  0 end)))) as SEER,
    (count(Recordtype)) as Calls
FROM HDO.CDR_RAW
WHERE $__timeFilter(StartDateTime)
GROUP BY $__timeGroup(StartDateTime, '5m')
ORDER BY 1
```

Resulting panel:

Session Defects Ratio (SDR) - Ingress:

- Panel 7
- Configuration:

```
SELECT
  $__timeGroup(StartDateTime, '5m', 0) as time,
100 *
((sum(case when CallDisconnectReasonIngress = 500 then 1 else 0 end) +
sum(case when CallDisconnectReasonIngress = 503 then 1 else 0 end) +
sum(case when CallDisconnectReasonIngress = 504 then 1 else 0 end)) /
  (CAST((count(Recordtype)) as FLOAT))) as SDR
FROM HDO.CDR_RAW
WHERE $__timeFilter(StartDateTime)
GROUP BY $__timeGroup(StartDateTime, '5m')
ORDER BY 1
```

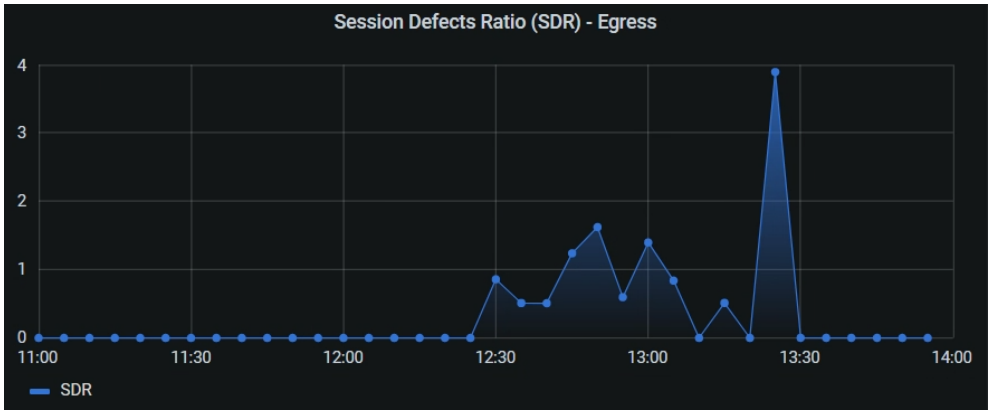
Resulting panel:

Session Defects Ratio (SDR) - Egress:

- Panel 8
- Configuration:

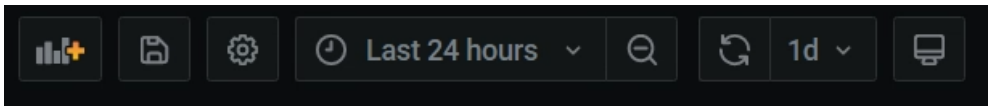
```
SELECT
  $__timeGroup(StartDateTime, '5m', 0) as time,
100 * ((sum(case when CallDisconnectReasonEgress = 500 then 1 else 0 end) +
sum(case when CallDisconnectReasonEgress = 503 then 1 else 0 end) +
sum(case when CallDisconnectReasonEgress = 504 then 1 else 0 end)) /
  (CAST((count(Recordtype)) as FLOAT))) as SDR
FROM HDO.CDR_RAW
WHERE $__timeFilter(StartDateTime)
GROUP BY $__timeGroup(StartDateTime, '5m')
ORDER BY 1
```

Resulting panel:



K.3 Dashboard time handling

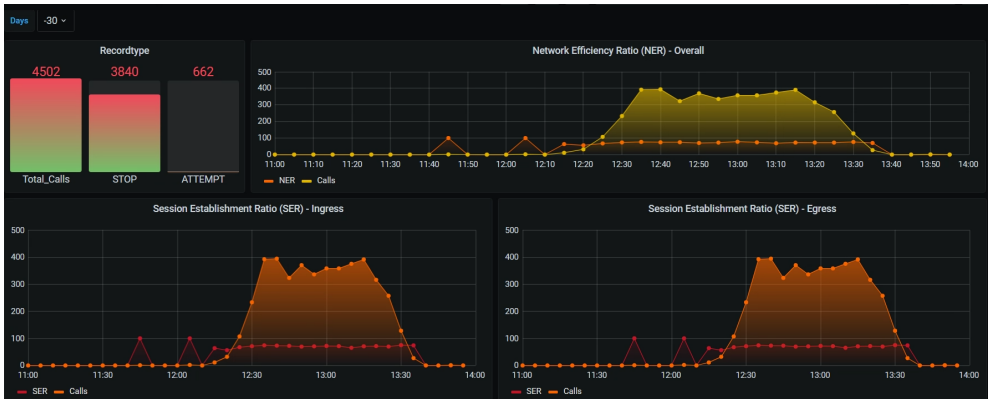
In the top right corner of the dashboard, among other things, there are several settings related to time management:

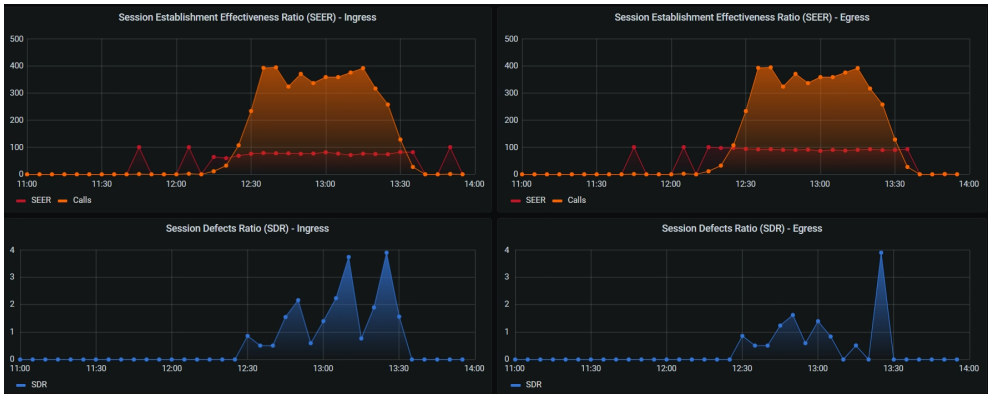


To achieve the desired dashboard, edit the time range for when data is visualized. In this case, the time range is relative and it is set to the last 24 hours. Since data in the database are in the UTC+00.00 format (because of the SBC), it is necessary to manipulate the time zone. If it is summertime, set the time zone to UTC+02.00. If it is wintertime, set the time zone to UTC+01.00. In addition, the dashboard is set to automatically update the panels every day. This will ensure less manual interactions with the dashboard.

K.4 Resulting dashboard

For the figure below. The relative time range is not set to the last 24 hours. This is only for testing purposes, in the production environment relative time range will be set to the last 24 hours.


Days
-30
Recordtype
4502
3840
662
Total_Calls
STOP
ATTEMPT
Network Efficiency Ratio (NER) - Overall
NER
Calls
Session Establishment Ratio (SER) - Ingress
SER
Calls
Session Establishment Ratio (SER) - Egress
SER
Calls


Session Establishment Effectiveness Ratio (SEER) - Ingress
SEER
Calls
Session Establishment Effectiveness Ratio (SEER) - Egress
SEER
Calls
Session Defects Ratio (SDR) - Ingress
SDR
Session Defects Ratio (SDR) - Egress
SDR

Jølsett. M, Stene. C, Östergren. P

Monitoring and reporting KPIs in an EMS phone system

NTNU
Kunnskap for en bedre verden