|  |
| --- |
| Regional IT: Integration og Planlægning |
| Et praktikophold indenfor Region Syddanmarks Sundheds-IT |
| Denne rapport vil give et indblik i firma strukturen, processen og opgaverne varetaget i Integration og Planlægnings afdelingen. Derudover vil der blive givet et indblik i de opgaver udført af Christopher Buch, som praktikant i denne afdeling af Regional IT.  Denne praktikrapport skrevet af Christopher V. Buch, er underlagt fortroligheds erklæring og må derfor ikke ligges til offentlig skue senere. |

10-10-2014

Indhold

[Indledning 2](#_Toc399841039)

[Regional ITs struktur 2](#_Toc399841040)

[Arbejdet 2](#_Toc399841041)

[Databaserne 3](#_Toc399841042)

[Simple Queries 3](#_Toc399841043)

[SQL funktioner 3](#_Toc399841044)

[Rsfn\_UnixTimeToDateTime 4](#_Toc399841045)

[Rsfn\_DateTimeToUnixTime 5](#_Toc399841046)

[SQL Views 5](#_Toc399841047)

[Stored Procedures 6](#_Toc399841048)

[God praksis 6](#_Toc399841049)

[Insert ”route” procedure 8](#_Toc399841050)

[Insert i System procedure 9](#_Toc399841051)

[Statistic proceduren 9](#_Toc399841052)

[Reporting Services 10](#_Toc399841053)

[Udbytte 10](#_Toc399841054)

[Konklusion 10](#_Toc399841055)

# Indledning

Region Syddanmark er Danmarks tredje største region, med mere end 1,2 millioner indbyggere. I denne region er der flere store hospitaler: Sydvestjysk Sygehus I Esbjerg, Sygehus Lillebælt, Sygehus Sønderjylland I Sønderborg og Odense Universitetshospital.

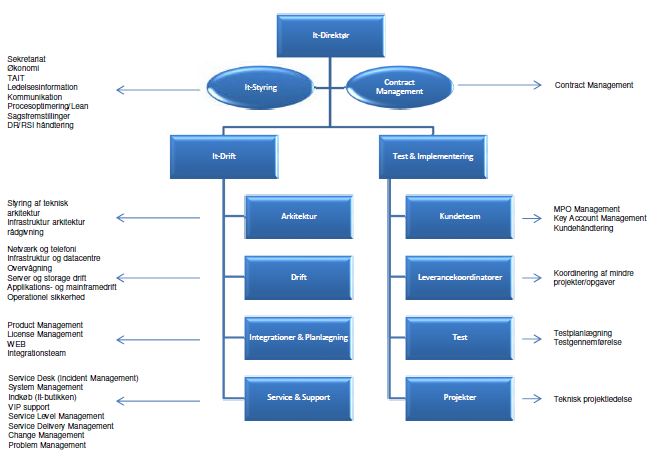
Sygehusene har hver en IT infrastruktur og har meddelelser af alt fra booking tider, til patient journaler, som skal sendes til forskellige undersystemer på enten hospitalet selv, en lokal klinik, et andet hospital, et laboratorie eller andet inden for sundhedssektoren.

Softwaren, som lægerne, sygeplejerskerne, laboranter og andet personel på sygehusene skal benytte, til dels udvikles, udvides og vedligeholdes af regionens sundheds IT-stab. Til denne brede IT-stab var jeg praktikant i afdelingen Integration og Planlægning. En afdeling ledet af Funktionschef Henrik Juul Andreasen, hjulpet af Chefkonsulent Morten Lyhne Thuesen og Teknisk Koordinator Thorkild Friis.

I denne rapport vil jeg gennemgå noget af den software, som arbejdes med i Integration og planlægningsafdelingen. Jeg vil med dette prøve at give et overblik over hvilke opgaver, samt hvordan og hvorfor de varetages af Integration og planlægning.

Jeg vil herefter komme ind på det jeg arbejdede med, i praktik perioden mandag den 11. august 2014 til 10. oktober 2014. Hvilket programmeringssprog jeg benyttede, hvilket udviklingsmiljø jeg programmerede det i og hvilke opgaver jeg lavede i for virksomheden i praktikperioden.

# Regional ITs struktur og infrastruktur



Figur : Region IT struktur. Kilde: regionsyddanmark.dk/dwn363741

Regional IT er opbygget I en struktur, set I mange større virksomheder, med en leder, diverse underledere og herunder medarbejdere.

Direktøren i Regional IT hedder Søren Lindegaard og hans opgaver består i kommunikation og bestemmelse af retninger, som har indflydelse på hele Regional IT. Dette kan være alt fra bestemmelser om hvilke systemer, som Regional IT skal køre, hvor hurtig disse skal rulles ud og hvorfra de skal indkøbes.

Under Direktøren er to afdelinger: IT-Drift og Test og Implementering

I IT drift er Flemming Brink afdelingschef.

IT-Drifts ansvar er bedst beskrevet fra regionens egen hjemmeside:

”Hovedansvaret i IT-Drift er at levere stabile og pålidelige it-services til forretningen med stor leverancesikkerhed og effektivitet. IT-Drift varetager forvaltning af servere og infrastruktur samt produkter og applikationer.

IT-Drift er ansvarlig for at sætte strategiske projekter fra Sundheds IT og taktiske projekter fra Regional IT. Desuden identificerer og prioriterer IT-Drift forbedringstiltag til it-strategien.

Derudover har IT-Drift ansvaret for at levere procedurer til Sundheds IT for idriftsættelse af projekter. Endelig rapporteres økonomien til It-Styring i forhold til drift, tillægsydelser og taktiske projekter.”   
Kilde: <http://regionsyddanmark.dk/wm345257>

Under IT-Drift er fire underafdelinger, med hver deres ansvar:

Arkitektur, som står for Teknisk Arkitektur for regionens Infrastruktur, samt rådgivning om samme.

Drift, som varetager Netværk og Telefoni, Infrastruktur og datacentre, samt overvågning af server og storagedrift, Applikation og mainframedrift og Operations Sikkerhed

Drift står blandt andet for implementering og vedligeholdelse af Regional IT’s Citrix løsning.

## Citrix

Citrix er et centraliserings løsning, som sætter programmer, som alle I Regional IT vil få brug for. Her tænkes programmer, som Microsoft Office-pakken, Adobe Reader og incident management Systemet MaximoSD. Citrix løsningen er valgt, som en hjælp til at holde programmer opdateret til de seneste versioner og de Udgaver af produkterne, som Regional IT har licenser til. På denne måde skal de enkelte ansatte ikke besvære sig med opdatering af produkterne, de skal ikke installeres individuelt, de kan opgraderes til brug til alle ansatte samtidig og hvis firmaet, som man har licenserne fra ønsker at undersøge, om man kun bruger de udgaver, som man har licens til, behøver de ikke undersøge hver enkelte computer og tablet, men kun den der ligger på den centrale server.

## MaximoSD

MaximoSD er som sagt et incident management system, baseret på en ITIL workflow.   
Et eksempel på benyttelse af MaximoSD, kan være når Integration og Planlægnings afdelingen får til opgave at udbedre problemer i software, som bruges af sundhedssektoren i regionen.  
I sådanne tilfælde, vil problemet først blive beskrevet i en problemudfyldelsesformular, hvor der skal beskrives hvor problemet opstår, med hvilket stykke software, hvem problemet opstod for, hvor hurtigt problemet skal løses og hvor meget påvirker det systemet hvor problemet er opstået.   
Påvirkning og hastighed hvorved problemer skal løses, indikeres med en talværdi mellem et og tre, hvor et vil være hurtigst muligt og en værdi på tre betyder at problemet kan løses når det er belejligt. Det samme gælder for påvirkning, hvor en værdi på et, betyder at det har stor indflydelse på systemet og er patient kritisk, mens en værdi på tre indikerer at der er tale om et mindre problem som ikke har særlig stor indflydelse på systemet som helhed.

Når et problem er registreret i MaximoSD og problemet er noget der skal rettes af Integration og planlægning, kan blandt andet funktionschef Henrik Juul Andreasen, chefkonsulent Morten Thyesen og teknisk koordinator Thorkild Friis tage højde for, hvem der kan udbedre problemet. De uddeler så opgaven til den eller de, som kan løse det.

Når problemet er løst, efter bedste kunnen af de som tog sig af opgaven, registeres en change. I en change, beskrives ændringen der er foretaget for at løse problemet og ny version af program eller ny kodedel til program, vedlægges til dette.

Henrik Juul Andreasen

Heriblandt godkendelse af indkøb af udstyr, som afdeling skal have foretaget. Dette kan være alt fra en godkendelse af indkøb af en Bærbar computer til få tusinde kroner, til indkøb af nye servere, der kan koste mere end 100.000 danske kroner. Foruden dette, skal han også holde styr

Cosmic, Cloverleaf (message broker software), Citrix

ITIL, Prince2

Kanban, tavle opgaver

# Arbejdet

Arbejdet I Integration og planlægnings afdeling har for mig bestået af kodning af Transact-SQL opgaver i Microsoft SQL Server Management Studio (SSMS).

SQL Server Management Studio

Transact-SQL, hvad er T-SQL?

Diagrammer: EER og flowdiagrammer

Subversion

## Databaserne

Cloverleaf og Medcom

## Simple Queries

De første opgaver bestod i at foretage simple queries. Den første skulle gøres mellem tre tabeller, hvori den ene, var en relationstabel mellem de to andre. Tabel1 ville have information omkring modtageren af brevet, samt et id matchende til denne, Tabel2 havde så information omkring meddelelsestype, samt til hvilken lokation meddelelsen skulle sendes. Relations tabellen indeholdt information der henviste til, om hvorvidt en meddelelse var kommet igennem. Såfremt der var foreign keys fra tabel1 og tabel2 i samme række, betød det at meddelelsen var kommet igennem.

Queryen gjorde så en join over alle tre tabeller og kunne ved en full join vise både meddelelser der var kommet igennem og ikke igennem. Dette blev der lavet et udtræk på, ved blot at gemme resultatet af den specifikke query.

Der blev desuden lavet et udtræk, hvori der var benyttet inner join og select distinct, så at kun meddelelser der var kommet igennem og meddelelser der var unikke ville blive vist. Det vil sige meddelelser, som har forskellige værdier i de tre felter.

Disse udtræk skulle så gives videre til en anden udvikler, som skulle bruge dem i forbindelse med et af regionens hospitaler.

## SQL funktioner

Det næste der skulle laves, var to SQL funktioner, der skulle konvertere en dato kolonne, enten fra unix-time eller til unix time. Unix time er en tidsenhed, der måler antal sekunder der er gået fra 1. januar 1970 til et givet tidspunkt længere fremme i tiden.

For at SQL ved hvor henne man vil gemme en funktion, det vil sige til hvilken database en funktion skal tilhøre, er det god praksis i starten at angive den navnet på den database der skal bruges. Til de funktioner jeg har skrevet, har jeg derfor i starten skrevet ”use CloRSDDevil”, efterfulgt af ”go”. Denne praksis sikrer at funktionen sættes til den rigtige database. Såfremt det ikke var skrevet, kunne funktionen have endt under Master databasen og ville dermed ikke være tilgængelig til brug, hvis jeg ville benytte den med en anden database.

En funktion i SQL skabes ved at forklare SSMS i en query, at den skal ”create function”. Dette efterfølges af funktionens navn, input parameter og parametertype. Derefter skal funktionen have forklaret, hvad den skal returnere, eksempelvis en integer, en nvarchar, en datetime eller sågar en tabel.

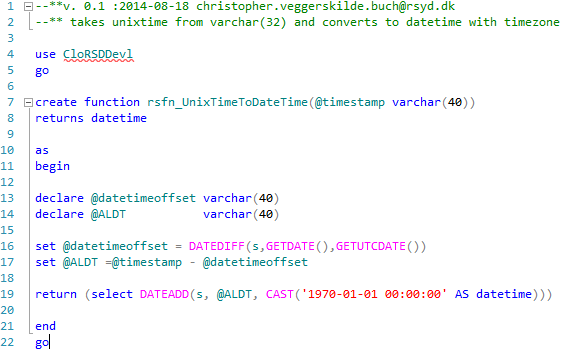
Efter man har defineret, hvad den skal returnere, kan man så begynde sin kode efter at have skrevet ”as begin”. Når man har skrevet koden, som funktionen skal udføre, afsluttes med at angive hvad den skal returnere.

Der skal også gøres opmærksom på at for hver ”begin”(begynd), skal der være en ”end”(slut). Skal funktionen køre umiddelbart efter, kan der efterfølgende skrives ”go”, som gør SSMS opmærksom på at det her kode skal køres før alt der måtte komme efter.

Når funktionerne efterfølgende skal bruges, skriver man i et query vindue:   
”exec function [funktionsnavn(input parameter)]” uden de firkantede parenteser og uden citationstegn.

Dette gælder også, såfremt man skal kalde databasens funktion fra C# eller andet programmeringssprog.

### Rsfn\_UnixTimeToDateTime

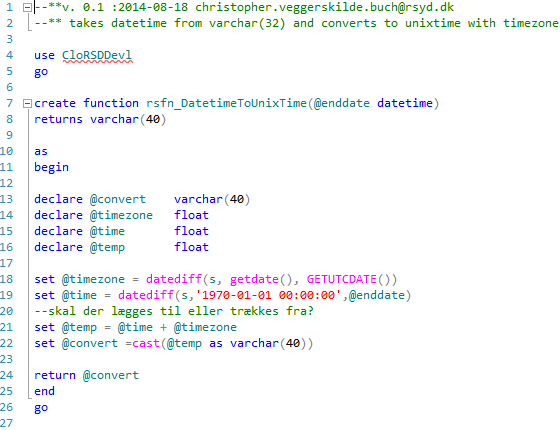


Den første funktion kaldet rsfn\_UnixTimeToDateTime tog en unixtid ind, hvorefter den i en variabel kaldet @datetimeoffset gemmer forskellen mellem UTC tid og lokal tid.

Den næste variabel (@ALDT) tager så tidsforskellen gemt i @datetimeoffset og trækker fra vores inputs tid

Afsluttende, returnerer funktionen en tid i normale tidsenheder, beregnet ved hjælp af den indbyggede funktion dateadd, som skal bruge en tidsenhed (eksempelvis sekunder). Derudover, skal den også bruge vores tid i unixtime og et starttidspunkt, som den skal regne den endelige tid ud fra. I dette tilfælde sætter vi starttidspunktet til at være 1. januar 1970, efter som det er en unix tid, som vi konverterer.

### Rsfn\_DateTimeToUnixTime



Denne funktion gør det modsatte af funktionen i sidste afsnit; Det vil sige den konverter en normal dato til en unix dato.

I funktionen er det første der sker, at der i en variabel gemmes tidsforskellen, mellem lokaltid og utc-tid. Dette gøres som i den første funktion, for at man senere kan tage højde for tidsforskellen mellem tiderne i beregningen af den endelige tid.

Efter at have gemt forskellen, er det næste skridt en konvertering til unix-tid. Dette gøres også ved hjælp af den indbyggede funktion datediff.

Når tiden er konverteret, skal der nu modsat den første lægges tidsforskellen til, hvor man før træk den fra.

Afsluttende returnerer man resultatet, så brugeren nu kan se den tilsvarende dato tid i unix-tid.

## SQL Views

Det næste der skulle laves var et view. Det er en funktionalitet, hvorved man kan give en ikke-administrator bruger adgang til at se visse tabeller i en database, uden egentlig at give dem adgang til andet end blot at kigge på indholdet af disse tabeller og deres fælles data. I et view kan man sågar administrere hvilke kolonner i tabellerne, som brugeren skal have adgang til at se.

I det view jeg skulle lave for Integrations teamet i Regional IT, skulle der vises tre tabeller: t\_organisationunit\_lettertype\_messagedirection (routes), t\_organisationunit og t\_system.

Routes tabellen fortæller os noget om hvor meddelelserne sendt igennem cloverleaf kommer fra, hvor de skal hen og generel trafik information, så som hvorvidt meddelelsen er intern eller extern.

t\_system fortæller os noget om systemerne, som meddelelserne sendes fra og til, med information omkring systemernes navn, beskrivelse og kontrol parametre.

t\_organisationunit er så en forlængelse af routes, i det den udspecificerer hvem afsenderen er og hvem modtageren er. Den indeholder dermed information omkring hvilke steder meddelelserne fysisk sendes hen. Det vil sige til hvilken person, på hvilken afdeling sendes de her meddelelser og fra hvem er de sendt.

Der skulle altså laves et view mellem disse. At lave sådan et view, gøres i SSMS ved i et query window at skrive create view, efterfulgt af navnet man ønsker at give sit view, endt med et ”as”.

Efter dette skrive man den query, som man ønsker at oprette et view på, startende med select, efterfulgt af kolonne navnene man vil vise. Derefter angives tabellerne, som kolonnerne eksisterer i, med almen ”join” statements, der forklarer hvordan tabellerne hænger sammen. Til sidst kan der også skrives en ”where” clause, der specificerer nogle bestemte ting den skal kigge på. Det er dog ikke nødvendigt at inkludere en ”where” clause til et view.

Når man har angivet alle kolonner man ønsker at se i sin select og tabeller i from og eventuelt specifikke paremetre i where, afsluttes med ”go”, for at forklare at opgaven ovenfor skal udføres.

Eksekverer man dette, vil der på den angivne database blive oprettet et view, under [databasenavn]->[views].

## Stored Procedures

Det jeg brugte mest tid på i praktik perioden, var at skrive stored procedures. Stored procedures er som navnet hentyder gemte procedurer, procedurer der udføres på serversiden, af den specifikke database. Procedurer kan forsimple opgaver både direkte på serversiden, men også i eventuelle programmer, som kobler op til serveren og databasen. Et af formålene, med en stored procedure, er at gøre det lettere for udviklere af programmer, som skal koble sig op mod databasen, at gøre de operationer der er nødvendige i forhold til programmet. Eksempelvis vil man ved hjælp af en stored procedure kunne spare en del linje koder i c#, hvis man skal indsætte eller update en eller flere tabeller i en database. Stored procedures kan kodes sådan at programudvikleren blot i sit program, skal angive at databasen skal eksekvere proceduren, med dens parametre, hvorefter den vil indsætte eller opdatere tabellerne. Program udvikleren, behøver dermed ikke kode hvilke tabeller der skal indsættes i eller opdateres i databasen.

En anden ting er at det kan mindske netværksbelastningen, da i stedet for at programmet skal til at tjekke, om hvorvidt noget information passer og hvorvidt det eksisterer allerede og blot skal opdateres, kan dette gøres på database-siden, hvorved der behøves færre transaktioner mellem programmet og databasen.

Stored procedures kan kodes til at foretage CRUD, enten alle kommandoerne eller enkelte af dem.

Procedurerne minder meget i deres opbygning om klasser i c#, i det de kan have en række parametre i starten som skal angives med navn og variabeltype, svarende til controlleren og efterfølgende have kode-delen af klassen.

I omfanget af praktiktiden, blev der udarbejdet seks procedurer, hvoraf de tre var procedurer, som skulle køres i en anden procedure. I de følgende afsnit vil jeg beskrive dem alle.

### God praksis

Blandt de første ting der skulle læres, når der skulle kodes procedurer, var hvordan man ved hjælp af god kode skik kunne teste for mange problemer, som kunne opstå ved kørsel af procedurer.

Til dette har Regional IT udviklet en standard, for hvordan procedurer bør kodes.

#### Testcases

Hertil kommer den del at man også bør teste om procedurerne virker efter intentionen og der ikke opstår fejl på grund af koden, når de benyttes. Dette involverer blandt at man efter at have installeret proceduren, på sin database afprøver om der ved udfyldning af parametre, rigtig og forkert får de forventede resultater. Får jeg en fejl meddelelse, hvis jeg lader et felt være null, som ikke må være null? Får jeg en fejl meddelelse, hvis et felt indeholder en værdi der ikke passer med en tidligere værdi i samme kolonne, såfremt det der indsættes, skal matche noget der allerede eksisterer i en kolonne. Det samme gælder for procedurer, som skal frembringe data, frem for at indsætte, her kan det også være nødvendigt at undersøge om det der er indtastet af brugeren, matcher i den tilsvarende kolonne.

På den måde minimerer vi chancen for at når proceduren senere skal køres, på den rigtige server, laver uventede fejl.

#### Error logging

Til regionens databaser er der af Thorkild Friis, udviklet en procedure, som tager fejl meddelelser fra procedurer der følger regionens standard og indsætter disse i en tabel kaldet error log.

Errorlog virker kun, hvis man i procedurerne benytter exception handling, i form af try-catch, hvor der i catch-delen skal ligge et stykke kode, som sikrer at fejl der sker i try-delen, bliver kørt igennem en procedure, som sikrer at alle fejl meddelelser, som sker ved kørsel af procedurer, bliver indsat i tabellen errorlog.

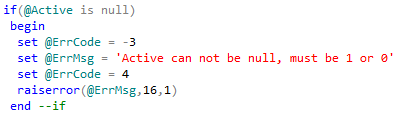
Ideen med denne, er at man senere kan gå ind og se, hvilke fejl der er opstået i løbet af kørsel af procedurerne der skal benyttes af regionen. Dette hjælper udviklere af selve procedurerne, både når de selv udvikler procedurerne, men også når andre benytter procedurerne og der opstår problemer eller fejl. I begge tilfælde, vil udvikleren af proceduren ved hjælp af errorlog være i stand til at se hvor fejlen opstod og hvorfor den opstod.

#### Error code

Error code er en int variabel der bør være i alle procedurer og som i sidste ende, skal sendes med ind i errorlog, for at give udvikleren en ide om hvor fejlen opstod.

For hver fejl der kan ske, bør der sættes en error code. Hvis error code er et negativ tal, betyder det at problemet er i koden, hvor et positivt tal forklarer at problemet er på netværket eller serveren.

Et eksempel på et sted hvor man i en insert procedure kan sætte en fejl meddelelse, er ved felter der ikke må være null, når der skal indsættes. Dette kan skrives som følgende kode stykke.

 Kodestykket til venstre viser en fejlhandling i en procedure, jeg senere i dokumentet vil diskutere. Det essentielle i kodestykket til venstre er strukturen. Koden viser at input parameteret @Active, ikke må være null, når proceduren køres. Såfremt den er null, skal den sætte @Errcode til -3, @ErrMsg til den røde tekst og derefter skal den benytte den indbyggede funktionalitet Raiserror, til at sende fejl meddelelsen, sammen med et fejl niveau 16, som fortæller noget om fejlens alvorlighed. Et niveau på 16 og en tilstand på 1, forklarer sql server at hvis fejlen sker, skal den helt stoppe med at eksekvere koden i try-delen, og hoppe til catch, hvorefter man kan logge fejlen til Errorlog tabellen.

### Insert ”route” procedure

Formålet med insert route proceduren er at den skal hjælpe med at kunne indsætte nye ”routes” det vil sige indsættelse af information i tabellerne t\_OrganisationUnit og t\_OrganisationUnit\_LetterType\_Messagedirection. Disse to tabeller er beskrevet i afsnittet SQL Views.

Opbygningen af proceduren var en kompliceret affære på daværende tidspunkt, da min viden omkring stored procedures var begrænset til kort information jeg havde fået i løbet af mine tidligere semestre, samt procedurer til cloverleaf databasen udviklet af Thorkild Friis.

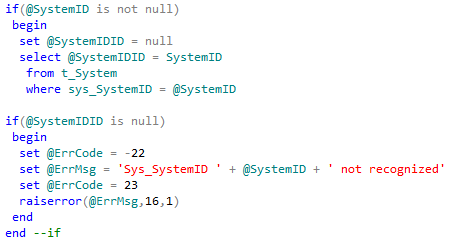
Sidstnævnte gav mig en skabelon for hvordan procedurer kunne se ud, men ikkepræcist hvordan procedurerne jeg skulle lave, ville se ud.

Informationen, som jeg havde fået at vide omkring hvad proceduren skulle indeholde var heller ikke det helt vilde. Alt hvad jeg havde fået af forhåndsinformation, var blot at den skulle være i stand til at indsætte ”routes”. Jeg havde på daværende tidspunkt ikke engang forstået meget omkring den logiske opbygning af de to tabeller.

Jeg snakkede dog en del med Thorkild Friis omkring det og fik forklaret meget af hvad jeg havde behov for at vide, men søgte også meget information online, for at få forklaret hvad forskellige ting gjorde og hvordan man bedst gjorde ting der skulle gøres.

Proceduren er opbygget sådan at den ved kørsel, starter med at undersøge om input parametre, som ikke må være null, er null.

Eksemplet til venstre viser et sådant tilfælde. Hvis @SystemIDID ikke har en værdi, så giv den værdien af af SystemID fra t\_System.

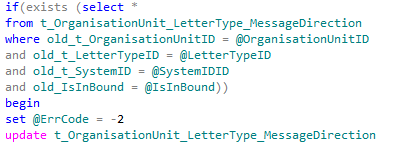
Efter det ovenstående kodestykke er kørt, testes derpå hvorvidt variablen kaldet @SystemID er null. Hvis denne IKKE er null, skal der testes på, om hvorvidt det der står i variablen, passer med noget der allerede står i t\_System tabellen, i kolonnen sys\_SystemID. Dette gøres som vist ved kodestykket herunder.

Ovenstående billede viser en typisk test, på input parametre, som skal matche noget eksisterende og ikke må være tilfældigt. Såfremt at @SystemID ikke er null, så sæt @SystemIDID til null. Herefter så sæt @SystemIDID lig med SystemID i tabellen t\_System, hvor sys\_SystemID er lig med @SystemID.

Det vil sige, såfremt at værdien i @SystemID matcher noget i kolonnen sys\_SystemID i tabellen t\_System, så tag værdien i samme række, fra kolonnen SystemID og indsæt denne i variablen @SystemIDID

Hvis der intet sted i kolonnen sys\_SystemID står det der er indsat i variablen @SystemID, vil variablen @SystemIDID forblive null. Dermed eksisterer den indtastede værdi ikke i nogen tabel og er dermed ugyldig. Af denne grund forklarer vi så brugeren, ved hjælp af stykket under, at det han eller hun indtastede i parameteret @SystemID ikke er genkendt.

Når alle disse tests på om input parametre er null og om de er gyldige, er foretaget, skal der undersøges hvorvidt en række allerede eksisterer i tabellen, som matcher fire af vores input parametre.



Routes tabellen har fire foreign keys, som svarer til fire tilsvarende input parametre i proceduren. Såfremt indholdet i de fire værdier i input parametre svarer til noget der står på en række i routes tabellen, skal den ikke forsøge at indsætte en ny række, men blot opdatere den eksisterende række i tabellen.

Såfremt de fire input parametre ikke alle eksister på samme række i tabellen, foretages en insert.

### Insert i System procedure

Proceduren til indsættelse af rækker i t\_system ligner meget proceduren til indsættelse af rækker i routes. Forskellene på de to procedurer ligger i hvor meget der skal testes på og hvor mange kolonner der skal indsættes på.

Derudover besidder t\_System kun en to foreign keys, hvor en af dem er en reference til sig selv.

Foruden disse små forskelle, er der ikke noget nyt i denne procedure.

### Statistic proceduren

Opbygning af under procedurerne

#### Rssp\_VarIntStatistic

#### Rssp\_DateStatistic

#### Rssp\_MultiStat

Benyttelse af indbygget procedure sp\_SelectSQL

# Reporting Services

# Udbytte

Hvad har jeg fået ud af opholdet på praktik pladsen?

# Konklusion