# Begränsningar

Till detta examenarbete fick vi endast tillgång till en kopia av produktionsdatabasen och tyvärr inte till varken back-end eller produktionsdatabasen. Detta medförde att vi var tvungna att ta hänsyn till vissa begränsningar.

* De lösningar vi kom fram till fick inte påverka någon funktionalitet utöver hastigheten på ”Bussines-intelligence” rapportgenereringen i back-enden.
* Lösningarna fick inte heller omdesigna databasen i stora drag då detta kunde påverka andra delar av systemet bortom vår kontroll.
* Och lösningen måste kunna implementeras utan att back-enden ska behöva modifieras.

Detta medförde att fokusen för vårt examensarbete skulle kretsa runt databasen.

**ALEX**

# Scope

Forskningen gällande databaser har de senaste årtionden exploderat och är mer eller mindre oändlig. Därför och med tanke på tidsbegränsningen behövde vi definiera ett scope.

De områden som vi kände kunde vara av relevans var:

* Query & execution plan tuning
* Index tuning
* Statistik för exekveringar
* In-memory lösningar

**ALEX**

# Research

Vi insåg snabbt att vår problemställning måste ha forskats om tidigare och vi bestämde därför att läsa på om andra tekniker som har utvecklats för att tackla liknande problem.

Detta gjordes för att se om relationsdatabasen vi skulle experimentera på kunde modifieras eller inspireras av system såsom distribuerade databaser, objektorienterade databaser och graph databaser såsom Neo4J.

**ALEX**

Utifrån de olika databassystem vi efterforskade fann vi en teknik som vi kände var ytterst relevant för vårat problem, närmare bestämt tabellpartitionering.

Tabellpartitionering är oftast relevant när tabellerna i databasen växer i storlek och kan leda till exekveringsproblem.

I SQL Servers finns det mer eller mindre två sätt att partitionera tabeller på, antingen manuellt eller använda sig utav en inbyggd funktion som partitionerar en existerande tabell.

Det manuella viset innebär att man skapar en view som agerar som en superklass till de avsedda tabellerna, Denna teknik kallas även för partitioned view där alla förfrågningar tas hand om utav viewn. Partitionering av datan måste även göras manuellt och placeras i fysiska tabeller som kopplas denna view.

Till skillnad från det manuella viset så innehåller den inbyggda funktionen inga fysiska tabellpartitioneringar utan servern automatiskt placerar rätt data i rätt partitioneringar som beror på villkoren för partitioneringen, t.ex. efter datum. Partitioneringarna är alltså under huven kan man säga och representeras endast men en fysisk tabell.

**ALEX**

# Hur skapas rapporterna?

Utifrån front-end interaktionen anropas först beck-enden som i sin tur anropar databasen. Back-enden anropar databasen med både queries och anropningar som säger åt databasen att exekvera stored procedures.

De flesta rapportgenereringarna består av stored procedures i databasen, detta är subrutiner som har skapats då dem ofta används och har till fördel att dem är cachade.

Efter att dessa stored procedures har exekverats sparas resultatet i en egen tabell för just denna rapport. Back-enden omvandlar sedan detta resultat till en visuell representation i front-enden.

**BYT BILD**

# Databasen

På bilden ser visuell representation av databasens tabeller och dess relationer, den högra cirkel är resultattabellerna. Databasen är av relationstypen och en Microsoft SQL Server med SQL språket MS SQL.

**BYT BILD**

Totalt finns det 254 tabeller och 59 stored procedures varav 30 av dem är direkt involverade med rapportgenereringen.

I medel är varje stored procedure beroende av 37 olika tabeller och uppbyggda mer eller mindre på samma sätt vad det gäller SELECT, INSERT, UPDATE och är på cirka 300 rader SQL vardera.

**ALEX**

# Analys av stored procedures

Vi valde att fokusera väldigt mycket på de stored procedures som ansågs vara ineffektiva. Till en början så tog vi tid på exekveringar där den värsta tog hela 7 minuter.

De stored procedures som tog längst tid stämde överens med de rapportgenereringarna som var mest kritiska enligt företaget och vi valde att koncentrera oss på de 7 värsta, med avseende på exekveringstid.

**BYT BILD**

Vi bröt sedan ner dessa subrutiner ännu mer genom att ta tid mellan de olika SQL block, SELECT, INSERT, UPDATE osv. för att få en tydligare bild var problemet låg. För det mesta var det INSERT-blocken som tog mest tid att exekvera men även vissa UPDATES.

**BYT BILD**

I dessa block kunde vi sedan med hjälp av subrutinernas execution plan, som är en visuell representation av exekveringen, identifiera vilka tabeller som visade sig vara problematiska.

**BYT BILD**

Med hjälp av denna analys kunde vi sedan experimentera lite med de tabeller som var av intresse. Vi gjorde en sorts simulering av tabellpartition för att se om denna teknik kunde hjälpa.   
All data i dessa tabeller hade datum för när dem var inlagda och därmed grupperade vi dem i år. Vi exekverade subrutinerna med parametrar för ett visst år, mätte exekveringstiden och sedan raderade data för ett visst år och gjorde om proceduren tills att endast data för det året vi sökte efter fanns kvar i tabellerna.

För att kunna göra detta och inte påverka dem existerande tabellerna, skapade vi två kopior av de tabeller som var intresse. En helt utan index och keys samt en exakt kopia innehållande de keys och index som originaltabellen innehöll.

**ALEX**

# Rekommendation

Baserat på de undersökningar och experiment vi har utfört så har vi ett par rekommendationer som bör genomföras och undersökas ytterligare för att uppnå en bättre rapportgenerering.

Databasens tabeller börjar växa så att det börjar bli problematiskt för rapportgenerering.   
Vår tabellexperimentation visade att tabellpartitionering är av intresse då företaget kommer att växa mer och mer i framtiden. Den inbyggda funktionen för tabellpartitioneringen finns tyvärr inte tillgänglig för den databasversionen som företaget idag har och vi rekommenderar därför att de applicerar en manuell variant såsom partitioned views tills de möjligtvis uppgraderar databasversionen.

**ALEX**

Vidare finns det diverse områden som också kan förbättras där användandet av olika datatyper behöver göras mer konsekvent. På grund av denna inkonsekvens av datatypsanvändande finns det också en väldigt låg kardinalitet genom hela subrutinernas tabeller.   
Kardinaliteten kan också bero på saknade index, gammal statistik, mycket liknande data m.m. och därför bör dessa områden ligga på ganska hög fokus för vidare utredning.

Eftersom Perfect It BeX AB har haft en ökning av kunder de senaste åren bör systemet förberedas för expansion. Idag finns det endast en fysisk server och bör expanderas till flera, då kan man t.ex. distribuera back-enden på en och databasen på en annan för att förminska prestanda belastningen.

Företaget bör också överväga att investera i Enterprise versionen då den möjliggör för företaget att använda Hekaton lösningen, inbyggd funktionalitet för tabellpartition och mer RAM-minne.

Denna investering är tyvärr dyr och fördelarna måste vägas mot kostnaden.