IPA

Individuelle Praktische Arbeit

SQL-Scheduler

Autor: Romano Sabbatella

Druckdatum: 02.05.2017

Fachvorgesetzter:	
Experte:	
Zweit-Experte:	
Valid-Expertin:	
Berufsbildner:	
Version:	1.0

Inhalt

1 Vorwort

1.1 Einleitung

Jeder Informatiker macht am Ende seiner vierjährigen Lehre eine Individuelle Praktische Arbeit, kurz IPA. Diese IPA prüft sein erworbenes Wissen und Können über die vier Jahre Lehre und soll beweisen, dass er in der Lage ist als Informatiker EFZ betitelt zu werden.

Ich habe jedoch einen, nicht alltäglichen, Werdegang hinter mir. Dies ist meine zweite IPA. Die erste IPA habe ich als Landschaftsgärtner gemacht; die Doku damals entsprach überhaupt nicht dem Umfang dieser IPA, da dies auch Handwerklicher Natur war.
Auch hatte ich in dieser Informatiker Ausbildung nicht vier Jahre Lehre wie die meisten, sondern zwei, da ich in Zürich Altstetten die Berufslehre für Erwachsene, im ZLI, absolviere. Diese Lehre ist für Quereinsteiger und Personen welche schon länger auf diesem Gebiet arbeiten, ohne einen Abschluss gemacht zu haben.

2.0 Teil 1: Umfeld und Ablauf

2.1 Titel der Facharbeit

SQL-Scheduler

2.2 Ausgangslage

Die ERP-Lösung „aplix“ aus unserem Hause benötigt für den Betrieb zunehmend mehr terminierte Skripte, welche regelmässig laufen müssen. Heute werden diese Skripte über den SQL-Server-Agent terminiert ausgeführt. Der Aufwand diese Skripte zu hinterlegen und zu überwachen wird jedoch zunehmend grösser. Insbesondere die zeitliche Koordination von Jobs, welche nicht gleichzeitig laufen sollen, gestaltet sich aufwendig und fehleranfällig.

2.3 Detaillierte Aufgabenstellung

Es soll ein SQL-Scheduler programmiert werden. Dieser ermöglicht es eine gespeicherte Prozedur (Stored-Procedure) zu einem definierten Zeitpunkt oder in einem bestimmten Intervall auszuführen.

2.3.1 Jobs

Für die Ausführung einer Prozedur zu einem definierten Zeitpunkt bzw. in einem bestimmten Intervall, kann ein Job im SQL-Scheduler erfasst werden. Der Job hat einen Namen, eine Beschreibung, ein oder mehrere Terminierungen, ein Protokoll und enthält den Namen einer auszuführenden gespeicherten Prozedur (Stored-Procedure) inkl. Schema (z.B.

dbo.sp_meineProzedur). Zusätzlich können ein oder mehrere Datenbanknamen hinterlegt werden, auf denen die gespeicherte Prozedur ausgeführt werden soll.

2.3.2 Terminierung

Jeder Job hat ein Start- und Enddatum (inkl. Uhrzeit). Das Enddatum ist optional. Der Job wird frühestens am Startdatum das erste Mal ausgeführt und wird nach dem Enddatum nicht mehr gestartet. Bereits gestartete Jobs werden bei Erreichen des Enddatums nicht während der Ausführung gestoppt, sondern laufen über das Enddatum hinaus, bis sie fertig sind.

Für die Terminierung gibt es zwei Varianten. Bei der ersten Variante kann auf eine bestimmte Uhrzeit terminiert werden und an welchen Wochentagen der Job läuft (z.B. Montag, Mittwoch und Freitag jeweils 14.00 Uhr). Die zweite Variante erlaubt eine Terminierung in einem Intervall (in Minuten) angegeben, beginnend ab dem Startdatum des Jobs (z.B. alle 60min ab 01.04.2017, 14.30 Uhr).

Sollte ein Job für eine Datenbank so lange laufen, dass laut Zeitplan wieder ein Start fällig wäre, wird der SQL-Scheduler den bereits laufenden Job erst zu Ende laufen lassen, bevor er nochmal gestartet wird (s.a. „Parallelität“).

2.3.3 Parallelität

Das parallele Ausführen von Jobs ist erlaubt und erwünscht, sofern es zwei verschiedene Datenbanken betrifft (s.a. «Jobs»). Auf der gleichen Datenbank soll, zu einem beliebigen Zeitpunkt, jeder Job nur einmal laufen.

Da die Entwicklung eines Konstrukts zur parallele Ausführung von SQL-Code den Rahmen dieser Arbeit sprengt, darf für diesen spezifischen Fall ein fertiges Script verwendet werden, welches eine parallele Ausführung ermöglicht. Dabei sind Lösungen, welche ohne Zusatzprogramme auskommen, also innerhalb des SQL-Server laufen, zu favorisieren. Eine Möglichkeit ist es, den SQL Service Broker zu verwenden, für welchen es im Internet fertige Lösungen zum parallelen Ausführen von SQL-Scripts gibt.

2.3.4 Reihenfolge

Die Jobs können so eingestellt werden, dass deren Ausführungsreihenfolge sichergestellt ist (z.B. über eine Priorität oder eine Beziehung zum «Vorgänger»). Der nachfolgende Job wird erst dann gestartet, wenn der Vorgänger beendet ist. Es reicht jeweils den Status vom unmittelbaren Vorgänger-Job zu prüfen, es muss nicht die ganze Struktur aufgelöst und geprüft werden.

Dass bei den Beziehungen zwischen den Jobs keine Rekursion entsteht, ist Sache des Anwenders und muss vom SQL-Scheduler nicht geprüft bzw. verhindert werden.

2.3.5 Protokollierung

Der Scheduler protokolliert das Start- und Enddatum für jeden Job den er aufruft. Sollten Ausnahmen (Exceptions, Errors) auftreten, werden diese zusammen mit den Daten vom Job-Aufruf in einem Protokoll gespeichert. Enthalten sind mindestens Job-Name, Datenbank, Name der aufgerufenen Prozedur, Startzeit, Datum und Uhrzeit der Ausnahme, Fehler Ja/Nein, detaillierte Fehlermeldung.

Damit bei Anpassungen der Jobs die historischen Daten nicht beeinflusst werden, sollen die Werte in einer separaten Protokoll-Tabelle geführt werden.

2.3.6 Speicherort der Objekte

Alle Tabellen, Prozeduren und Funktionen die der SQL-Scheduler benötigt, sollen in einer separaten Datenbank namens „SQLScheduler“ enthalten sein.

2.3.7 Auszuführende Prozeduren

Die durch den Scheduler auszuführenden Prozeduren befinden sich in der Regel in den Datenbanken des Aplix-ERP und nicht in der Datenbank vom SQL-Scheduler (eine pro Kunde, jedoch üblicherweise mehrere Datenbanken auf dem gleichen SQL-Server).

Der Scheduler muss prüfen, ob die auszuführende Prozedur auch tatsächlich bei der Kundendatenbank enthalten ist. Falls nicht, wird ein entsprechender Fehler im Protokoll des SQL-Scheduler eingetragen.

Die Datenbank des SQL-Scheduler befindet sich auf der gleichen SQL-Server-Instanz wie die Kundendatenbanken. Es darf vorausgesetzt werden, dass die Datenbanken direkt angesprochen werden können (z.B. Kundendatenbank.dbo.sp_meinBackupScript). Damit die Prozeduren in der Kundendatenbank auch ausgeführt werden können, darf ein Benutzer erstellt bzw. benutzt werden, der die nötigen Rechte auf alle Kundendatenbanken besitzt (z.B. «sa»). Dies ist sicherheitstechnisch kritisch und wird ggf. zu einem späteren Zeitpunkt angepasst, was jedoch nicht Teil der IPA ist.

2.3.8 Auslöser für den Start der Jobs (Trigger)

Da in SQL kein «Timer» verfügbar ist, darf im Rahmen der Arbeit für den Start des SQL-Scheduler ein Job im SQL-Server-Agent eingefügt werden, der z.B. jede Minute den selbst

programmierten SQL-Scheduler startet. Dies wird in Zukunft evtl. geändert, ist jedoch nicht Teil der IPA.

Um den SQL-Scheduler bzw. die darin konfigurierten Jobs zu starten, soll der Aufruf einer einzelnen gespeicherten Prozedur (Stored-Procedure) genügen.

Es soll auf zusätzliche Schleifen, welche regelmässig auf auszuführende Jobs prüfen, innerhalb des SQL-Scheduler verzichtet werden (z.B. mit WAITFOR oder ähnliches).

2.3.9 Testen

Die einzelnen Funktionen und Prozeduren sollen mit Hilfe von Skripts getestet werden.

2.3.10 Bedienung und Konfiguration

Die Bedienung findet direkt über SQL-Befehle und/oder das SQL-Management-Studio statt. Eine graphische Benutzerschnittstelle oder dergleichen ist nicht Teil der Arbeit.

2.3.11 Dokumentation

Für das Datenmodell des SQL-Scheduler soll ein ERM erstellt werden.

Die wichtigsten Prozesse sollen mit Flussdiagrammen dokumentiert werden.

2.3.12 Benutzeranleitung und Beispiele

Es sollen eine Benutzeranleitung und Beispielskripte für das Einfügen, Bearbeiten und Löschen von Jobs erstellt werden. Die Anleitung richtet sich nur an Entwickler und soll nur spezifischen Aspekte für den Umgang mit dem SQL-Scheduler beinhalten, jedoch keine Grundlagen zu SQL.

2.4 Mittel und Methoden

Für die Entwicklung steht ein SQL-Server 2014, SQL-Server-Profiler und das SQL-Server-Management-Studio (SSMS) zur Verfügung.

2.5 Vorkenntnisse

Der Umgang mit der Datenbank von „aplix“ konnte in verschiedenen Projekten und Supportfällen während der Ausbildungszeit trainiert werden. Dabei konnten auch die nötigen Werkzeuge (SQL-Server, SQL-Server-Management-Studio) kennen gelernt werden.

2.6 Vorarbeiten

Es sind keine Vorarbeiten notwendig.

2.7 Neue Lerninhalte

Es ist eine neue Datenbank zu erstellen inkl. der nötigen Tabellenstruktur. Bisher wurde primär auf bestehenden Datenbank gearbeitet. Die Architektur muss ebenfalls erarbeitet werden, was zuvor nicht nötig war.

2.8 Arbeiten der letzten 6 Monate

Neben der Erstellung verschiedener Auswertungen und Berichte (SQL-Abfragen, Berichtdesign), mussten auch einzelne Funktionen und Prozeduren auf dem SQL-Server entworfen und programmiert werden. Es mussten ferner im Support Fehler in gespeicherten Prozeduren, Funktionen und Ansichten lokalisiert und behoben werden.

2.9 Projektorganisation

Abbildung 1 Projektorganisation dieser IPA

Die Organisation dieser Projektarbeit sieht wie folgt aus:

- Die Aufgabenstellung wurde von … verfasst, dem Fachvorgesetzten.
- Diese Aufgabenstellung wurde von der Valid-Expertin Petra Seitz validiert.
- Der Prüfungskandidat, welcher diese Aufgabe umsetzt, ist Romano Sabbatella.
- Der Experte …, der Zweit-Experte …, sowie der Fachvorgesetzte … werden diese IPA kontrollieren und bewerten.

2.10 Zeitplan

2.11 Meilensteine

Hier werden die Meilensteine, sowie welche Kriterien erfüllt sein müssen um jene zu erreichen, detailliert beschrieben.

2.11.1 Planung und Entscheidung abgeschlossen

Dieser Meilenstein ist erreicht sobald der Entscheid über die Art, Vorgehensweise und Machbarkeit gefällt wurde. Folgende Dokumente müssen erstellt sein:

- Zeitplan der Individuellen Praktischen Arbeit:
  Dieser zeigt den zeitlichen Verlauf, mit welchem diese Arbeit geplant wurde.
- ER-Model
  Das Entity Relationship Model zeigt die Tabellen und Struktur auf, welche für diese Arbeit erstellt werden müssen. Auch die Beziehungen der Tabellen, sowie die Datentypen der Attribute werden hier dargestellt und verdeutlicht.
- ER-Diagramm
  Das Entity Relationship Diagramm zeigt den Ablauf, welche die jeweiligen Prozeduren machen werden. Auch werden hier Entscheidungen, welche innerhalb von Prozeduren gemacht werden, für den Kunden so dargestellt, dass dieser keinerlei Programmierkenntnisse braucht.
- Entscheidung wurde gefällt
  Es wurde die Entscheidung über Machbarkeit und Dauer des Projekts festgelegt. Diese Entscheidung wird dokumentiert mit den Gedanken welche hinter diesem Entschluss liegen.

2.11.2 Programmierung, sowie Manual schreiben abgeschlossen

Dieser Meilenstein ist erreicht sobald die Programmierung abgeschlossen und die Installation- und die Benutzeranleitung erstellt wurde. Es werden folgende Dokumente erstellt:

- Skripts für die Erstellung der Datenbank und deren Tabellen.
- Prozeduren
  Diese beinhalten die gesamte Funktionalität des Schedulers.
- Prozeduren für die Parallelität.
  Diese wurden nicht von mir erstellt, werden jedoch beiliegen. Diese helfen dem Verständnis der Parallelität.
- Benutzerhandbuch
  Hier wird für den Benutzer aufgezeigt, in diesem Fall der Programmierer, wie die Einträge in der Datenbank aussehen müssen um sein Ziel zu erreichen.

- Installationsanleitung
  In dieser wird beschrieben welche Voraussetzungen erfüllt sein müssen. Auch wird veranschaulicht wie man die Datenbank sowie die Skripte anlegt sodass die Funktionalität sichergestellt ist.

2.11.3 Testen abgeschlossen

Dieser Meilenstein ist erreicht, sobald die Testdaten erstellt und die Funktionalität ausgiebig getestet wurde. Die festgestellten Fehler wurden behoben und die Tests noch einmal durchgeführt. Dieser Vorgang wird wiederholt, bis die erwarteten Resultate erreicht sind. Hierzu können folgende Dokumente anfallen:

- Skripts
  Es werden evtl. Tabellen sowie Prozeduren benötigt um die Test durchzuführen.
- Skripts mit Testdaten
  Testdaten müssen in die Tabellen eingefügt werden um die Tests korrekt durchführen zu können.
- Dokumentation der Tests
  Dies beinhaltet eine klare Dokumentation des Testumfeldes auf welchem getestet wird, sowie die Testmittel welche benötigt wurden um die Tests zu machen. Es wird hier auch eine Aufstellung der Testfälle dokumentiert. Falls mehrfach getestet wird, werden die Tests mehrfach in der Dokumentation dargestellt.

2.11.4 Auswertung der geleisteten Arbeit abgeschlossen

Dieser Meilenstein ist erreicht, sobald die geleistete Arbeit mit der Planung verglichen worden ist. Dieses Resultat wird in der Dokumentation, inklusive der Reflexion der geleisteten Arbeit, festgehalten. Hier wird in der Dokumentation folgendes anfallen:

- Vergleich der Planung und Realisierung
  Hier werden Planung und Realisierung verglichen und die Unterschiede dokumentiert. Es wird auch darauf eingegangen, wieso diese Unterschiede entstanden sind. Es wird versucht dem Leser zu vermitteln was nicht beachtet wurde, um beim nächsten Mal auf diese Punkte besser zu achten.
- Reflexion
  In dieser wird dokumentiert wie die Aufgabe von meinem Standpunkt aus erledigt wurde. Dazu gehören auch Eigenkritik und Entscheidungen welche gut gelungen sind. Auch wird festgehalten was ich bei einem nächsten Mal anders machen würde.

2.11.5 Dokumentation abgeschlossen

Dieser Meilenstein bildet das Ende des Projektes. Dieser wurde erreicht sobald die Dokumentation fertiggestellt, auf Rechtschreibung und Inhalt geprüft und geheftet wurde.

2.12 Arbeitsprotokoll

2.12.1 Tag 1 - Dienstag 18.04.2017

Arbeitszeit	8h/8h
Geplante Arbeit	• Aufgabenstellung analysiert • Zeitplan erstellen • ERM- und ER-Diagramm erstellen
Erledigte Arbeit	Bevor ich mit der Analyse und der Planung meiner IPA begonnen habe, habe ich die Aufgabenstellung meines Fachvorgesetzten in die Dokumentation kopiert. Anschliessend habe ich die Aufgabenstellung gründlich analysiert und Fragen an meinen Fachvorgesetzten gestellt. Diese können in der Hilfestellung nachgelesen werden. Als die Fragen geklärt waren, habe ich mit dem ER-Model begonnen. Ich hatte bereits eine Vorstellung wie dies aussehen sollte. Da dies noch nicht ganz durchgeplant war musste ich verschiedene Ansätze verfolgen und auch für eine Entscheiden. Die Entscheidung werde ich morgen fällen müssen. Ich habe einen Abend Zeit darüber nachzudenken.
Zeitplan	Ich bin mit den geplanten Arbeiten fertig geworden und liege gut im Zeitplan.
Probleme	Auch am ersten Tag können Probleme auftauchen. Dies ist heute aber nicht passiert.
Lösungen	Da es keine Probleme gab erübrigt sich dies.
Hilfestellungen	Nachdem ich die Aufgabenstellung analysiert hatte, ergaben sich zwei Fragen. Eine waren die Diagramme welche gefordert werden. Ich verstand nicht ganz was mein Fachvorgesetzter unter „Prüfung auf auszuführende Jobs“ und „Ausführung eines Jobs“ verstand. Nachdem dies geklärt war, stellte sich noch eine weitere Frage. In der Aufgabenstellung wird von einem ERM (Entity-Relationship-Model) gesprochen wobei im Bewertungsbogen ein ER-Diagramm (Entity-Relationship-Diagramm) gefordert wird. Nach einer Absprache war klar, dass beides gemacht werden sollte.
Reflexion	Es lief gut für den ersten Tag. Ich bin zufrieden wie ich heute vorangekommen bin. Ich musste eine Entscheidung in der Tabellenstruktur fällen, welche mir nicht leicht fiel. Die eine Variante war mit weniger Aufwand verbunden, jedoch war diese mir nicht flexibel genug. Ich entschied mich somit für eine Tabelle und Relation mehr. Ich schaue motiviert auf morgen.

2.12.2 Tag 2 - Mittwoch 19.04.2017

Arbeitszeit	8.5h/8h
Geplante Arbeit	• Planung überprüfen und entscheiden • Datenbank und Tabellen erstellen • Expertengespräch • Meilenstein 1 erreicht
Erledigte Arbeit	Ich habe gestern Abend über meine Planung nachgedacht und heute Morgen nochmals analysiert ob ich mit meinem gestrigen Zeitplan, den Flowcharts der „Prozedur“ Abläufe und dem Entity-Relationship-Model zufrieden bin. Nun musste ich mich auch zwischen zwei Varianten der Umsetzung entscheiden. Diese Entscheidung wurde in Kapitel 3.5 dokumentiert. Ich habe mir das ‚GO‘ gegeben um weiter zu fahren, somit ist der erste Meilenstein erreicht. Das Script „CreateDatabaseSQLScheduler.sql“ wurde erstellt. Dies löscht die Datenbank, falls vorhanden, und erstellt diese wieder auf dem SQL-Server. Das erstellte Skript „CreateTables.sql“ erstellt die Tabellen auf der Datenbank, mit deren Beziehungen zueinander. Das Skript wurde so angelegt, dass bei einem weiteren Ausführen die Tabellen gelöscht und wieder frisch erstellt werden. Gegen Ende des Tages besuchte mich ... und wir durften uns kennenlernen. Es gab ein konstruktives Gespräch, für welches ich sehr dankbar bin.
Zeitplan	Ich bin mit der geplanten Arbeit fertig geworden und konnte mir bereits Gedanken für morgen machen.
Probleme	Es sind keine Probleme aufgetreten
Lösungen	Da es keine Probleme gab, ist hier nichts Erwähnenswertes.
Hilfestellungen	Ich habe von meinem Fachvorgesetzten ein Logo der Firma Boreas, in Originalgrösse angefordert, da ich nicht genügend Rechte habe, um auf diesen Ordner zugreifen zu können.
Reflexion	Der Tag ging wie im Flug vorbei. Es lief super. Ich konnte sehr viele Inputs von ... mitnehmen und werde diese in meine Dokumentation einfliessen lassen. Auch konnte er mir Fehler, welche sich jetzt schon langsam einschlichen, zeigen, die ich verbessern muss. Ich werde mich morgen daran setzen.

2.12.3 Tag 3 – Donnerstag 20.04.2017

Arbeitszeit	8h/8h
Geplante Arbeit	• Prozedur erstellen • Dokumentieren
Erledigte Arbeit	Es wurden die Prozedur „sp_CheckToRunJobs.sql“ und „sp_RunJob.sql“ erstellt. Das „sp_CheckToRunJobs.sql“ ist insoweit

	fertig, dass noch der asynchrone Aufruf gemacht werden muss. Dazu benötige ich jedoch in einem nächsten Schritt noch Zeit, den Aufruf richtig zu erstellen. Das Skript „sp_RunJob.sql“ ist fast fertig. Dieses Skript soll später in eine Queue des SQL Service Broker kommen und von dort aufgerufen werden.
Zeitplan	Ich bin gut im Zeitplan. Ich merke jedoch, dass ich vermutlich bei der Dokumentation mehr Zeit brauche als geplant. Dies wird sich aber spätestens morgen herausstellen.
Probleme	Ich habe bei der Entwicklung der Skripts bemerkt, dass ich eine Spalte, „DayTime“, in der Tabelle „Schedules“ vergessen habe. Diese wird wichtig beim Ermitteln, welche Jobs ausgeführt werden sollen, da sie angibt um welche Uhrzeit an einem Tag das Skript gestartet werden soll.
Lösungen	Ich musste das ER-Model, sowie das „CreateTables.sql“, überarbeiten.
Hilfestellungen	Heute habe ich keine Hilfestellung gebraucht.
Reflexion	Das mit der vergessen Spalte war dumm von mir, eine Fehlplanung, welche behoben werden musste da die Funktionalität ansonsten nicht dem Ziel entspricht. Es stand explizit in der Aufgabenstellung, dass eine solche gebraucht wird. Zum Glück habe ich meine Skripts so geschrieben, dass ich die Änderung schnell durchführen konnte. Dies war beim Entwickeln mehr Arbeit, welche sich doch ausgezahlt hat. Ich bin weiter mit gutem Gefühl auf Kurs, jedoch hoffe ich keine anderen Fehler bei der Planung gemacht zu haben.

2.12.4 Tag 4 – Freitag 21.04.2017

Arbeitszeit	8.2h/8h
Geplante Arbeit	• Entwicklung der Prozedur abschliessen • Die „sp_RunJob.sql“ parallel aufrufen
Erledigte Arbeit	Die Entwicklung der Prozedur, welche ich gestern begonnen habe wurde insoweit fertig gestellt, dass nur der Parallele Aufruf der „sp_RunJob.sql“ noch gemacht werden muss. Ich habe mich in den Kommentar des Aufrufs eingelesen und auch den Code welcher dahintersteht begutachtet.
Zeitplan	Ich liege gut im Zeitplan. Evtl. muss für die Doku mehr Zeit investieren als geplant.
Probleme	Der Parallele Aufruf wird wahrscheinlich schwieriger als ich dachte. Das Skript welches mir zur Verfügung liegt, ist, so wie es aussieht, nicht für Datenbankübergreifende Aufrufe gemacht.

Lösungen	Ich werde ein oder zwei Test Skripte schreiben müssen, sowie eine Tabelle. Diese werde ich in einer Test-Datenbank integrieren um den Datenbankübergreifenden Aufruf zu testen, bevor ich diesen implementiere.
Hilfestellungen	Ich habe eine Dokumentation des Skripts zum Parallelen Aufruf zur Hilfe genommen. Es befindet sich in der Quellenangabe.
Reflexion	Der Tag ging recht schnell vorbei. Da ich wusste, wie ich vorgehen sollte, kam ich recht schnell voran. Ich bin unsicher wieviel Zeit ich tatsächlich brauche, um den parallelen Aufruf korrekt zu gestalten. Jedoch bin ich jetzt auch froh, mal eine kleine Auszeit in Form des Wochenendes, zu haben und über die Arbeit nachdenken zu können.

2.12.5 Tag 5 - Montag 24.04.2017

Arbeitszeit	8h/8h
Geplante Arbeit	• Paralleler Aufruf der Prozeduren
Erledigte Arbeit	Die Skripts, welche ich aus dem Internet hatte, mussten erst in der Datenbank laufen gelassen werden. Diese Skripts generierten eine Tabelle, eine Queue, einen Service, welcher an die Queue als Kommunikationspartner gehängt wurde, sowie zwei Prozeduren. Danach wurde der parallele Aufruf mühsam umgesetzt.
Zeitplan	Ich brauchte mehr Zeit als ich angenommen hatte. Zum Glück hatte ich hier grosszügig geplant und eine Reserve schon bei der Planung gegeben. Durch diese Reserve bin ich noch auf Kurs.
Probleme	Ich konnte zwar die „Jobs“ (Prozedur) auf einer Testdatenbank ansprechen, jedoch nicht parallel. Nach stundenlangem Versuchen, bemerkte ich, dass die Queue stehen geblieben war. Ich konnte mir dies nicht erklären bis ich auf die Idee kam, die Errors in die Testdatenbank zu schreiben. Dies klappte nicht, wieso auch immer. Plötzlich hatte ich pro Aufruf zwei Fehlermeldungen, was komisch war.
Lösungen	Ich entfernte die Insert- und Updatestatements und merkte, dass es zwar keinen Fehler mehr gab, jedoch der Job nicht ausgeführt wurde. Nach längerem durchstöbern der Microsoft Dokumentationen über Queues und Services bin ich auf ein Berechtigungsproblem gestossen. Der Service hatte schlicht kein Recht den Job auszuführen. Es musste die Datenbankeigenschaft „TRUSTWORTHY“ auf „ON“ gestellt werden.
Hilfestellungen	Es wurden für Nachforschungen über Queues und Services die offiziellen Microsoft-Dokumentationen verwendet.
Reflexion	Heute war „zäh“. Es ging kaum voran und es hat gedauert bis ich den

	Fehlern und Problemen auf die Schliche kam. Meine zeitliche Reserve für die Planung ist bei dieser Aufgabe aufgebraucht worden. Auch habe ich, einmal mehr, spüren dürfen, dass Skripts aus dem Internet nicht immer das Gelbe vom Ei sind. Jedoch habe ich heute viel gelernt. Das Sprichwort „Durch Fehler lernt man“ hat sich wieder mal bewahrheitet.

2.12.6 Tag 6 – Dienstag 25.04.2017

Arbeitszeit	8.5h/8h
Geplante Arbeit	• Usermanual erstellen • Installationsmanual erstellen • Expertenbesuch • Erstellen von Testdaten und Skripten
Erledigte Arbeit	Das Usermanual wurde so geschrieben, dass ein Informatiker in der Lage ist, den Scheduler wunschgemäss einzusetzen. Das Usermanual beinhaltet eine Auflistung der Attribute, deren Datentypen und eine Beschreibung was dieses Attribut bewirkt. Das Installationsmanual wurde recht knapp, da es hauptsächlich darum geht, die erstellten Skripte in einer bestimmten Reihenfolge auszuführen. Es wurde jedoch auf wichtige Punkte hingewiesen, welche befolgt oder kontrolliert werden müssen, bevor diese Applikation live gehen soll. Der Expertenbesuch von ... verlief, wie letztes Mal, sehr gut. Er gab mir wichtige Tipps für den weiteren Verlauf der IPA, sowie für die Präsentation und das Websummary. Ich konnte offene Fragen klären, welche im Raum standen. Abschliessend wurde ein Skript erstellt, welches eine Test-Datenbank, inklusive Tabellen und Prozeduren erstellt. Zudem wurde ein Insert-Statement für die SQLScheduler-Datenbank vorbereitet um die erstellten Jobs laufen zu lassen und somit die Funktionalität zu testen.
Zeitplan	Ich liege gut im Zeitplan.
Probleme	Wir mussten heute alle Geräte über einen Zeitraum von 30 Minuten herunterfahren, da in dem Häuserblock, in welchem unser Büro liegt, die Stromzähler ausgewechselt wurde. Dies ist auch der Grund für die halbe Stunde länger als geplant.
Lösungen	Während dieser halben Stunde habe ich die Systemgrenzen meines Systems auf ein Blattpapier skizziert, welches ich morgen digital noch zeichnen werde.
Hilfestellungen	Heute habe ich keine Hilfestellung gebraucht.

Reflexion	Ich bin zufrieden mit meiner heutigen Leistung. Der Elektriker zog mir allerdings einen Strich durch meinen Plan. Diese Pause, wenn man dies so nennen kann, tat allerdings recht gut. Ich bin zuversichtlich was das Testen angeht, bin jedoch nicht ganz sicher ob ich alles beachtet habe. Ich werde es in den nächsten Tagen sehen.

2.12.7 Tag 7 - Mittwoch 26.04.2017

Arbeitszeit	8h/8h
Geplante Arbeit	• Testdrehbuch schreiben • Testen mit Fehlerbehebung
Erledigte Arbeit	Das Testdrehbuch wurde erstellt. Auch wurde heute eine Stunde mehr in die Dokumentation investiert, um wieder im Zeitplan zu sein. Der Dokumentation wurden das Testumfeld, Testmittel und Testfälle angefügt.
Zeitplan	Ich konnte heute nicht wie geplant mit dem Testen beginnen, bin jedoch auf Kurs. Morgen wird sich zeigen wie gut mein Produkt sich macht. Ich hoffe es gibt nicht allzu viel was ich ausbessern muss.
Probleme	Heute sind keine Probleme aufgetreten.
Lösungen	Brauchte es keine.
Hilfestellungen	Brauchte ich heute nicht.
Reflexion	Ich bin zuversichtlich, dass mein Projekt dem entspricht, was ich mir darunter vorstelle. Da ich heute jedoch erst die Testfälle geschrieben habe, bin ich ein wenig unter Zeitdruck. Ich bin zuversichtlich, dass ich die Zeit einholen kann.

2.12.8 Tag 8 – Donnerstag 27.04.2017

Arbeitszeit	8h/8h
Geplante Arbeit	• Testen mit Fehlerbehebung
Erledigte Arbeit	Ich habe nach definierten Testfälle getestet und die Beobachtungen eingetragen. Zudem musste ich bei der Parallelität nochmals ans Überarbeiten, da dort Fehler aufgetaucht sind. Weiteres bei Probleme und Lösungen.
Zeitplan	Da Ich sehr viel Zeit für die Fehlerbehebung brauchte bin ich ein wenig in Verzug gekommen.
Probleme	Bei den Testfällen wurde bemerkt, dass es Probleme mit der Parallelität gibt. Es trat das Phänomen auf, dass man im Job mit einem „WAITFOR DELAY“ die Tabelle komplett sperrt. Es kam die Vermutung auf, dass der Service eine Transaktion startet.
Lösungen	Es wurden mit dem SQL-Profiler, welcher alle Aktionen auf einem

	SQL-Server aufzeichnen kann, die Transaktionen aufgezeichnet. Das Resultat war ernüchternd. Ausser ein paar wenige interne Prozesse des SQL-Servers wurden keine Transaktionen gefunden. Es kam der Verdacht auf, dass der Service nur eine gewisse Anzahl von Queue-Einträgen verarbeiten kann und somit limitiert ist. Nach Durchforsten der Microsoft-Dokumentationen bin ich nicht weiter gekommen. Also fing ich nochmals von vorne an und bemerkte, die gesuchte Limitation befindet sich nicht auf dem Service, sondern auf der Queue. Diese schränkt die Anzahl der Empfänger ein. Nachdem ich diese Eigenschaft erhöht hatte, klappte alles wie gewünscht.
Hilfestellungen	Microsoft-Dokumentationen über Services und Queues.
Reflexion	Ich lerne jeden Tag mehr dazu. Heute hatte ich zwar viel Zeit gebraucht um die Dokumentationen zu lesen, bin jedoch aber auch froh, den Fehler gefunden zu haben. Ich hätte von Anfang an von vorne anfangen sollen zu suchen ich war zu fest der Überzeugung, der Service sei das Problem.

2.12.9 Tag 9 – Freitag 28.04.2017

Arbeitszeit	8h/8h
Geplante Arbeit	• Umsetzung mit Planung vergleichen • Reflexion schreiben • Dokumentieren
Erledigte Arbeit	Ich habe die Planung anhand von Diagrammen und Modellen mit der Realisierung verglichen, und meine Beobachtungen zu Papier gebracht. Anschliessend habe ich mich und meine geleistete Arbeit begutachtet und dies in meiner Dokumentation niedergeschrieben. Dadurch, dass ich heute „nur“ Dokumentiere, hatte ich auch die Gelegenheit diese nochmals durchzuschauen und zu verbessern.
Zeitplan	Ich bin gut im Zeitplan.
Probleme	Heute wurden keine Probleme entdeckt.
Lösungen	Brauchte ich keine.
Hilfestellungen	Keine.
Reflexion	Ich konnte mich sehr stark auf die Dokumentation fixieren, was mir recht war. Mein schriftliches Deutsch ist nicht wirklich gut, daher war ich froh ein wenig mehr Zeit für die Dokumentation zu haben. Ich werde dieses Dokument von meiner Mutter gegenlesen lassen. Sie ist grammatikalisch sehr viel besser als ich, jedoch versteht sie von der Informatik sehr wenig. Wir werden beide das Dokument grammatikalisch auf Kurs bringen.

2.12.10 Tag 10 - Dienstag 02.05.2017

Arbeitszeit	8h/8h
Geplante Arbeit	• Schlusswort schreiben • Dokumentation überprüfen
Erledigte Arbeit	Das Schlusswort wurde verfasst. Anschliessend wurde die Dokumentation auf Fehler überprüft, abgeschlossen, Ausgedruckt und geheftet. Diese Version wurde als PDF auf PkOrg geladen.
Zeitplan	Ich bin von der Zeit her gut fertig geworden. Zum Glück hatte ich mir am Schluss eine Reserve eingeplant. welche ich nutzen konnte. Bei der Doku hatte ich mich verschetzt.
Probleme	Es traten keine Probleme auf.
Lösungen	Dies erübrigt sich.
Hilfestellungen	Da ich zuhause keine Bindemaschine habe, musste ich dies bei meiner Mutter binden.
Reflexion	Der Schlussspurt war erfolgreich. Ich bin mit meiner Arbeit zufrieden und bereite mich nun auf die Präsentation und den Vortrag vor.

3 Teil 2: Projekt

3.1 Kurzfassung des IPA-Berichtes

3.1.1 Ausgangssituation

Unsere ERP-Lösung „Aplix-Handel“ hat ein sogenanntes MIS (Management Information System), welches die Daten Kundenspezifisch darstellt. Da das Aufbereiten sehr viel Rechenleistung benötigt, wird dies über Nacht gemacht, wenn niemand aktiv auf der Datenbank arbeitet. Wenn dies tagsüber gemacht wird, würde der User dies in Form eines massiven Performanceverlustes massiv spüren. Der MS-SQL Server bietet jedoch den SQL Server-Agent, mit welchem wir in der Lage sind SQL-Skripts automatisch zu starten, erst ab der Web-Version an. Da wir auch kleine Kunden haben, welche nicht über die finanziellen Mittel verfügen, um sich einen teuren SQL-Server zu unterhalten, wurde schon länger nach einer alternativen Lösung gesucht. Die Express Edition ist zwar kostenlos, verfügt aber nicht über einen SQL Server-Agent, was diese Edition bis anhin nicht als Alternative attraktiv machte. Dies sollte sich nun ändern, sodass wir auch für kleinere Betriebe attraktiv werden. Im Rahmen dieser IPA sollte nun ein SQL-Scheduler programmiert werden, welcher im späteren Verlauf, auf SQL-Server Express-Editionen laufen wird.

3.1.2 Umsetzung

Für die Umsetzung dieser Arbeit wurde nach IPERKA vorgegangen. Die Aufgabenstellung wurde erst analysiert und Fragen geklärt. Anschliessend wurde ein Zeitplan erstellt, sowie Diagramme erstellt, welche zur Planung der Arbeit und den Skripts behilflich waren. Anhand dieser Planungen wurde die Arbeit realisiert und getestet. Zum Schluss der IPA wurde ein Fazit gezogen und meine Arbeit reflektiert.

3.1.3 Ergebnis

Es wurde erfolgreich ein SQL-Scheduler erstellt, welcher parallel Prozeduren auf Kundendatenbanken laufen lässt. Dies ist sehr speziell für SQL, da dies unter normalen Umständen nicht parallel ausgeführt wird. Es wurde gewährleistet das Serverweit maximal 5 Prozeduren parallel ausgeführt werden, dies ist aber jederzeit änderbar.

3.2 Projektmethode

Für diese Individuelle Praktische Arbeit musste eine geeignete Projektmethode gewählt werden, welche sich für dieses Projekt eignet. Da Romano Sabbatella von der Berufsschule her gut mit IPERKA vertraut ist und sich diese Projektmethode auch gut für ein Projekt in dieser Grösse eignet, fiel der Entscheid auf IPERKA. Jede Phase in IPERKA beinhaltet bestimmte Fragen, welche beantwortet werden sollten. Die Phasen, sowie deren Fragen, sehen wie folgt aus:

Informieren

- Was ist das Ziel dieses Projekts?
- Was sind die technischen Voraussetzungen dieses Projektes?
- Was sind die Vorgaben an welche man sich halten muss?

Planen

- Wie sieht der Zeitplan aus?
- Welche Arten gibt es wie das Projekt realisiert werden kann?

Entscheiden

- Ist die Planung zeitlich machbar?
- Ist die Planung der Arbeit machbar?

Realisieren

- Wurde alles Geplante umgesetzt?

Kontrollieren

- Wurde alles wie geplant umgesetzt?
- Ist die Umsetzung qualitativ gut?
- Treten noch Fehler auf?

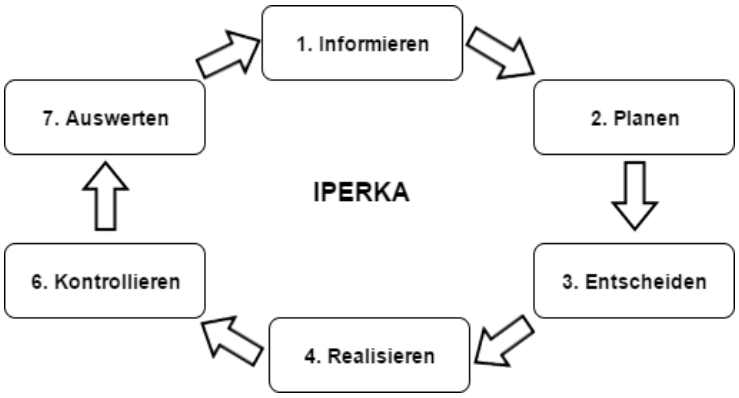

Abbildung 2 Ablauf von IPERKA

Auswerten

- Was wurde gut gemacht?
- Was wurde schlecht gemacht?
- Was wurde daraus gelernt?

3.3 Informieren

3.3.1 Umgebung

Das Produkt wird auf einem SQL-Server 2012 Standard-Edition laufen. Dies kann und wird sich nach der IPA aber noch ändern. Das Ziel dieser IPA ist, dass der Scheduler auf der genannten Version läuft. Im Moment ist es so, dass alle unserer Kunden sich mindestens einen SQL Server mit der Standard-Edition zulegen müssen, damit alle Funktionalitäten gewährleistet sind. Das „Aplix"-System braucht den SQL Server Agent, welcher zu einer bestimmten Zeit die nötigen Skripte, sogenannte „Jobs", anstösst, um zum Beispiel rechenintensive Berechnungen auszuführen. Der SQL Server Agent ist allerdings erst ab der Standardedition verfügbar, siehe Darstellung.

Verwaltungstools

Funktion	Enterprise	Standard	Web	Express mit Advanced Services	Express
SQL Management Objects (SMO)	ja	Benutzerkontenst euerung	Benutzerkontenst euerung	Benutzerkontensteuer ung	ja
SQL-Konfigurations-Manager	ja	Benutzerkontenst euerung	Benutzerkontenst euerung	Benutzerkontensteuer ung	ja
SQL CMD (Command Prompt Tool – Eingabeaufforderungstool)	ja	Benutzerkontenst euerung	Benutzerkontenst euerung	Benutzerkontensteuer ung	Ja
Distributed Replay – Administratortool	Ja	Benutzerkontenst euerung	Benutzerkontenst euerung	Benutzerkontensteuer ung	Nein
Distributed Replay – Client	Ja	Benutzerkontenst euerung	Benutzerkontenst euerung	Nein	Nein
Distributed Replay - Controller	Ja (bis zu 16 Clients)	Ja (1 Client)	Ja (1 Client)	Nein	Nein
SQL Profiler	ja	Ja	Nein [1]	Nein [1]	Nein [1]
SQL Server-Agent	ja	Benutzerkontenst euerung	Benutzerkontenst euerung	Nein	Nein
Microsoft System Center Operations Manager Management Pack	ja	Benutzerkontenst euerung	Benutzerkontenst euerung	Nein	Nein
Datenbankoptimierungsratgeber (DTA)	Ja	Ja [2]	Ja [2]	Nein	Nein

Abbildung 3 Verwaltungs Tools der verschiedenen SQL-Server Editionen

Die Lizenzierung eines solchen Servers ist recht teuer und kann nicht von jedem Betrieb bezahlt werden, da eine Lizenz pro Prozessorkern mindestens 3'717 US Dollar / Jahr kostet. Somit wurde SQL-Server-Agent eine Alternative gesucht. Aus dieser Suche entstand diese IPA.

3.3.2 Aufgabenstellung analysieren

Das Produkt, welches erstellt wird, trägt den Namen „SQL-Scheduler". Dieses beinhaltet eine Datenbank, in dieser die benötigten Tabellen deren Struktur nicht definiert ist. Diese Datenbank und deren Tabellen werden durch den Kandidaten, Romano Sabbatella, erarbeitet und definiert. Die Tabellen dürfen keine ‚n' zu ‚n' Beziehungen aufweisen. Jene

müssen in einer Zwischentabelle aufgelöst werden. Der Kandidat wird mindestens zwei Prozeduren erstellen, in welchen die Jobs, welche auszuführen sind, zuerst kontrolliert werden. Das heisst, es wird geschaut ob die Datenbank vorhanden ist, in dieser das Skript und eine Prozedur vorhanden ist und ob dieser Job ausgeführt werden muss. Dies wird in den Tabellen mit einer Terminierung definiert. Diese kann über zwei Varianten passieren. Die erste ist eine Terminierung auf Wochentag und Tageszeit, bei welchem der Job ausgeführt werden soll. Die Zweite Variante ist eine Intervall Terminierung. Hierbei wird dem Scheduler ein Intervall gegeben in welchem Zeitabstand, in Minuten, der Job ausgeführt werden soll. Der Unterschied wird durch eine Spalte auf einer der Tabellen definiert, welche im Benutzerhandbuch genauer beschrieben ist. Die Ausführung der Jobs wird parallel laufen. Dies stellt eine Herausforderung dar, da SQL Prozedural ist. Das heisst, es wird ein Befehl gesendet und auf eine Antwort gewartet. Parallel heisst, die Befehle werden gleichzeitig ausgeführt. Die Skripts für die parallele Ausführung sind nicht Bestandteil der IPA, daher konnte sich der Kandidat im Internet an einer bestehenden Lösung bedienen. Es ist jedoch wichtig, dass der Kandidat den groben Ablauf dieser Skripts versteht. Zusammenfassend ist das Ziel des Projektes einen SQLScheduler, nach Aufgabenstellung, welcher Datenbankübergreifend Prozeduren startet. Die Lösung wird auf einem SQL-Server 2012 Standard-Edition laufen.

3.4 Planen

3.4.1 Datensicherung / Versionsverwaltung

In der Firma Boreas wird keine Versionsverwaltung betrieben. Einzig wird in den Skripts, welche in den SQL-Server hinzugefügt werden, in die Header jeweils die Version (Datum der Änderung), die Initialen und der Beschrieb der Änderung geschrieben.

```
-- ============================================================================================================
-- E R S T E L L E N
-- ============================================================================================================
CREATE PROCEDURE [dbo].[sp_CopyProdukt_OVL]
   @i_ProduktID int,  --@VonProdukt
   @i_ProduktIDNeu int --@NachProdukt
AS
-- ------------------------------------------------------------------------------
-- Entwickler..... AS
-- Funktion....... Wird nach sp_CopyProdukt aufgerufen
-- Parameter...... @i_ProduktID    Vorlage Produkt
--                 @i_ProduktIDNeu Neues Produkt
-- Rueckgabe......
-- Verwendung..... Produktmaske in aplix
-- Versionsinfo... 2012.02.05 / AS : Erstellung
--                 2016.11.24 / RS : Da der Trigger auf der Tlb Produkte Preise erstellt und im sp_CopyProdukt
--                                   auch Preise erstellt werden die doppelten Preise gelöscht
-- Fehlercodes....
-- ------------------------------------------------------------------------------
```

Abbildung 4 Veranschaulichung der Versionen eines Skriptes der Firma Boreas

Dies ist für die Arbeit des Kandidaten nicht gerade von Vorteil, da eine Versionsverwaltung Pflicht ist. Man kann damit beweisen, dass die Dokumente mindestens einmal am Tag abgespeichert wurden, welche man bearbeitet hat. Es wird eine Business Dropbox

verwendet, welche sich automatisch auf alle Geräte synchronisiert. Die Versionsverwaltung wurde nun so gelöst, dass pro Tag einen Ordner mit dem Inhalt des letzten Tages erstellt wurde. Neue Dokumente wurden jeweils in den Ordner vom Erstellungstag gelegt. Um sicher zu gehen, wurden jeweils am Ende eines Arbeitstages die Ordner auf eine Externe-Harddisk kopiert, welche der Kandidat mit nach Hause nahm um den Backup räumlich von den Daten zu trennen.

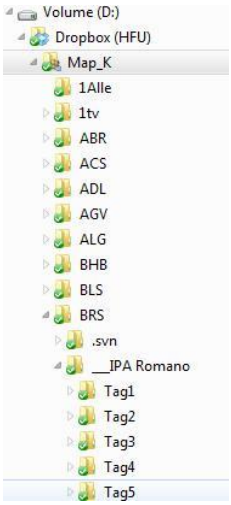

Abbildung 6 Speichern der Dokumente pro Tag

Abbildung 5 Veranschaulichung des Backups

3.4.2 ER-Model

Der erste Schritt der Planung ist das Erstellen eines Entity-Relationship-Model, welches innerhalb der Datenbank die Tabellen und ihre Beziehungen darstellt. Alle Beziehung welche eine ‚n' zu ‚n' Beziehung haben, sollen in einer Zwischentabelle aufgelöst werden. Innerhalb einer Tabelle werden die Felder mit deren Datentypen genauer beschrieben, sodass die ganze Tabellenstruktur auf einen Blick sichtbar ist. Es wurde analysiert ob ‚n' zu ‚n' Beziehungen bestehen und ob diese aufgelöst werden müssen. Es wurden keine ‚n' zu ‚n' Beziehungen gefunden, nach dem die grobe Idee des ER-Models erstellt wurde. Die Tabellen werden danach in der Realisierung analog des geplanten ER-Models erstellt.

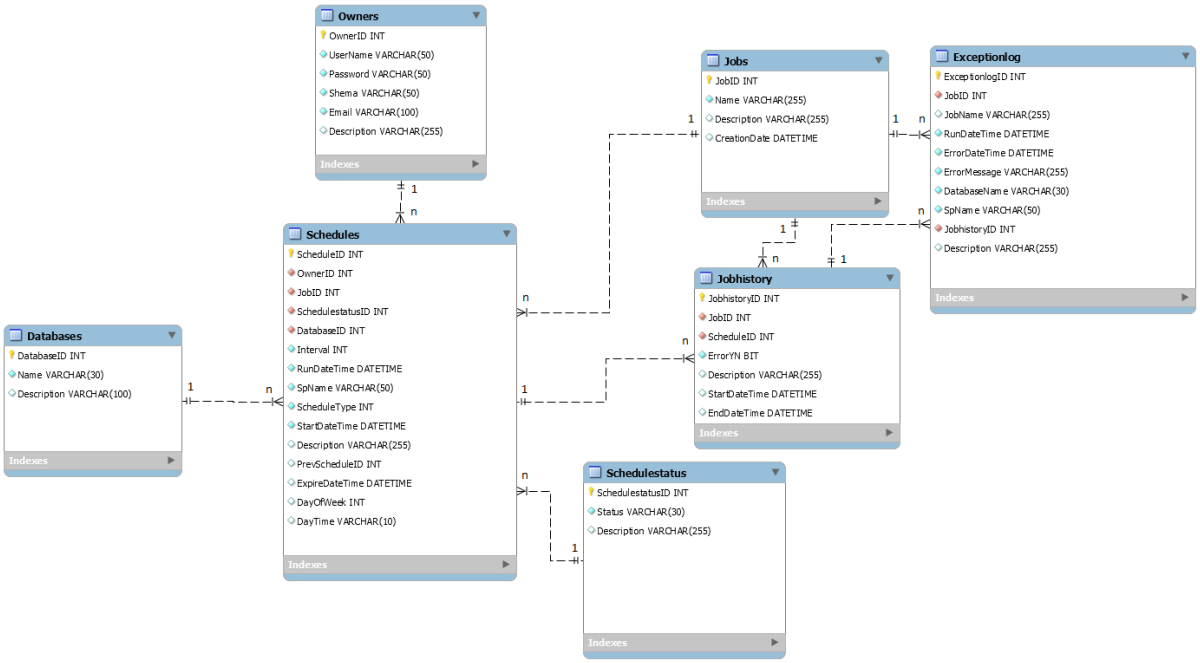

Abbildung 7 Entity-Relation-Model

3.4.3 ER-Diagramm

Das Entity-Relation-Diagramm sieht wie ein Klassendiagramm aus. Das bedeutet, es werden Beziehungen untereinander mit Worten dargestellt. Zum Beispiel „Owner owns Schedule“, wobei der „Owner“ und „Schedule“ zwei Tabellen sind und „owns“ ihre Beziehung zueinander. Auch werden die wichtigsten Attribute der einzelnen Tabelle angegeben und verdeutlicht. Dieses Diagramm konnte fast analog des ER-Models gemacht werden.

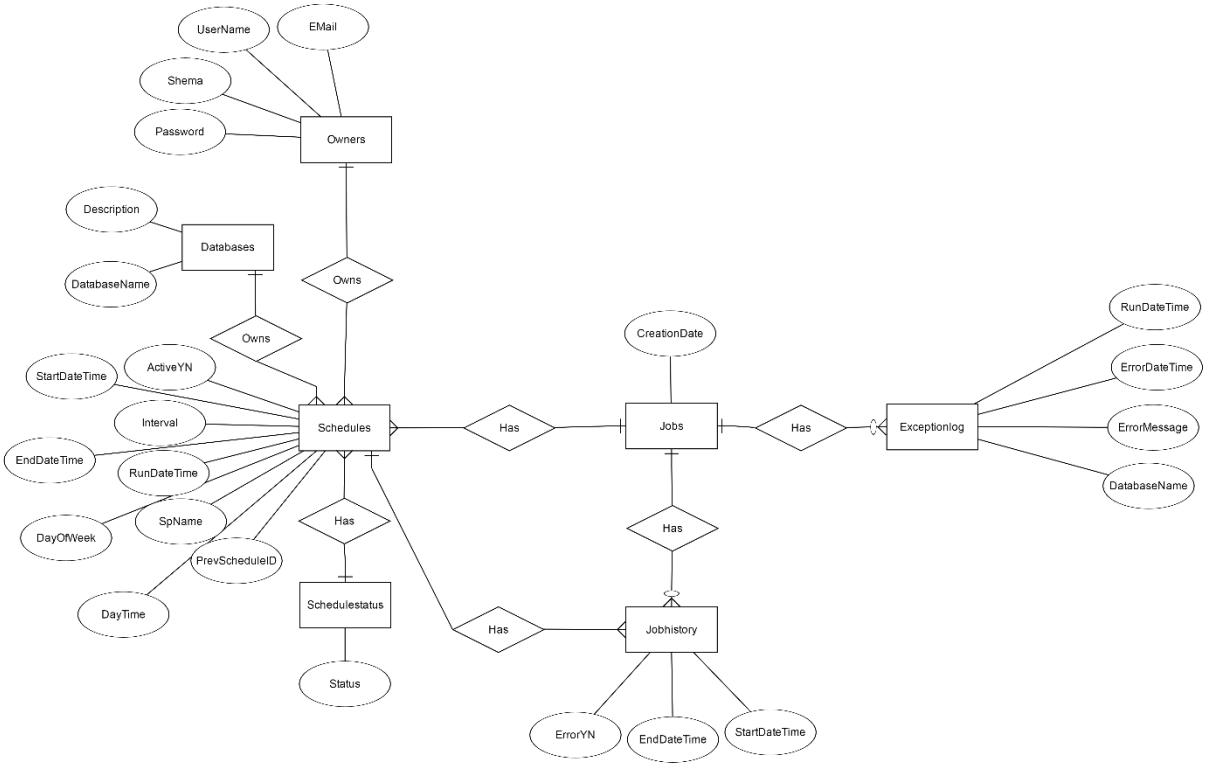

Abbildung 8 Entity-Relationship-Diagramm

3.4.4 Flowchart-Diagramm „Prüfung auf auszuführende Jobs“

Der nächste Schritt, bei der Planung ist, ein Flowchart-Diagramm, welches den Ablauf und die Prüfung auf auszuführende Jobs beschreibt. Es werden Entscheidungen innerhalb der Prozedur so aufgezeigt, dass diese für jedermann verständlich sind. Es wird alles genauestens beschrieben und mit Pfeilen verbunden, wann man wo durchkommt und welche Aktionen wo durgeführt werden. Dieses Flowchart wird vom Kandidaten für die Struktur der Prozedur, welche Entwickelt wird, verwendet.

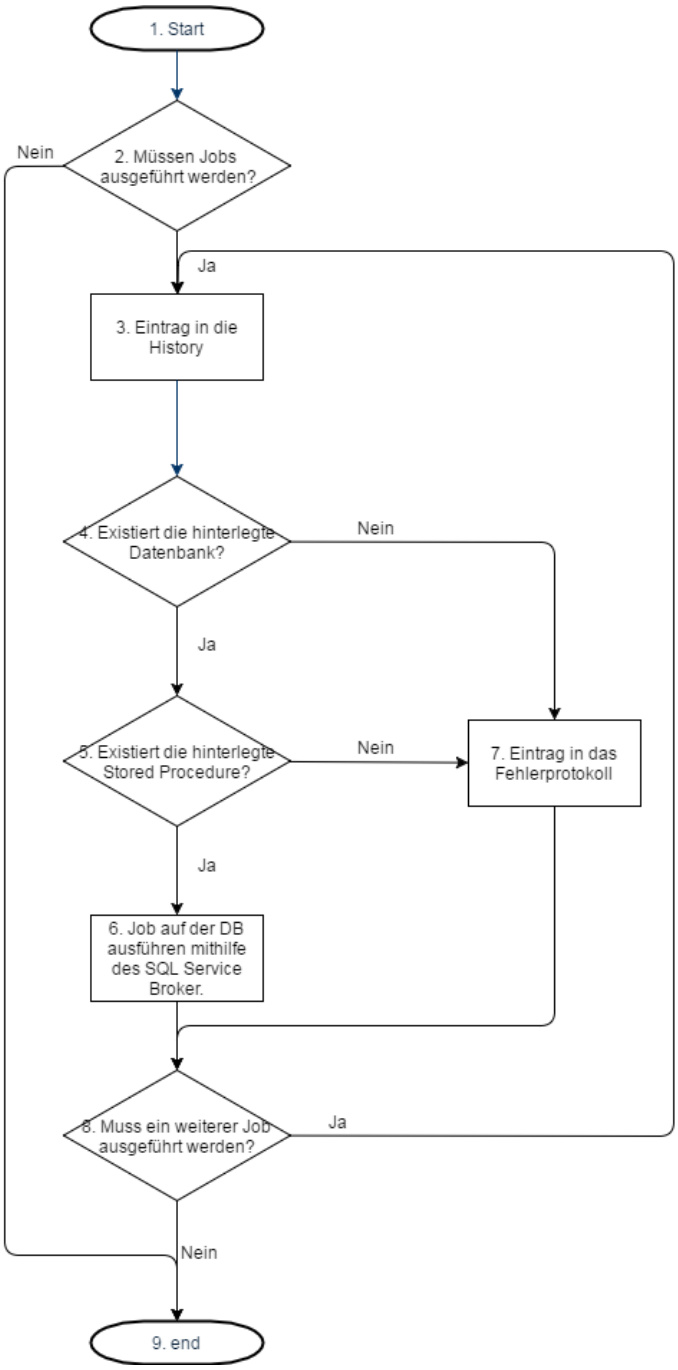

Abbildung 9 Flowchart-Diagramm "Prüfung auf auszuführende Jobs"

3.4.5 Flowchart-Diagramm „Ausführung eines Jobs“

Dieses Flowchart-Diagramm zeigt auf was passiert nachdem der Job gestartet wird. Es beschreibt eine eigene Prozedur, welche parallel aufgerufen wird. Hier werden auch wieder bestimmte Kontrollen gemacht, welche im Diagramm Schritt für Schritt veranschaulicht werden. Dieses Diagramm sollte, wie das vorherige, übersichtlich und gut verständlich sein, ohne das nötige Vorwissen.

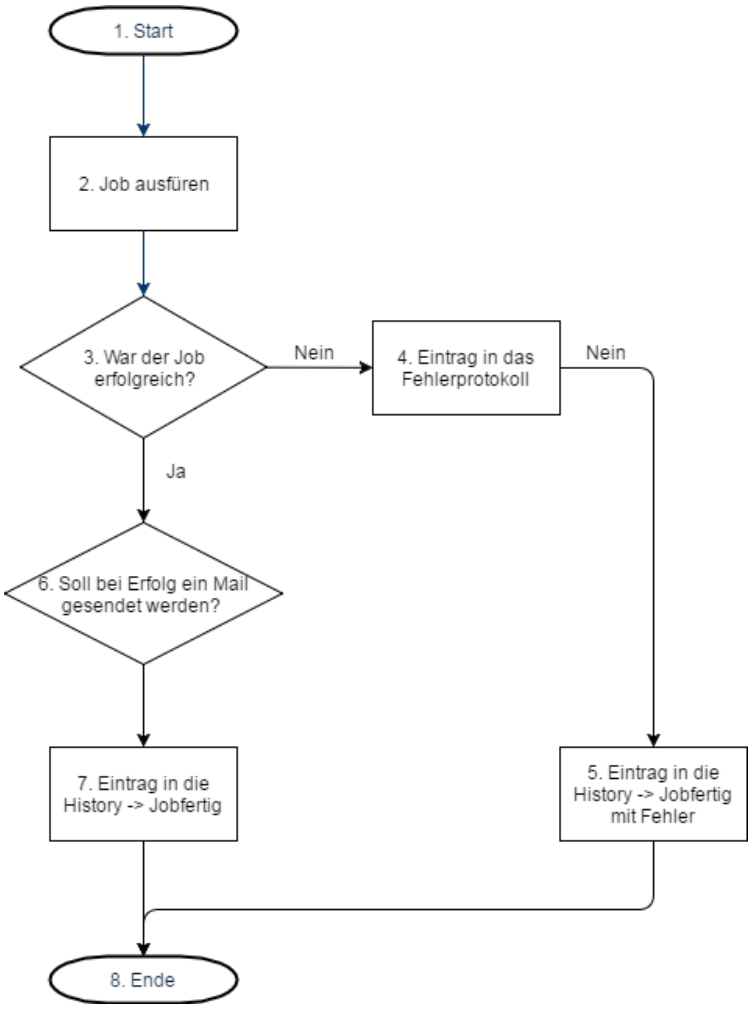

Abbildung 10 Flowchart-Diagramm "Ausführung eines Jobs"

3.4.6 Systemgrenzen und Zusammenhänge

Hier wird die Systemgrenze, meines Projektes, konkret definiert.

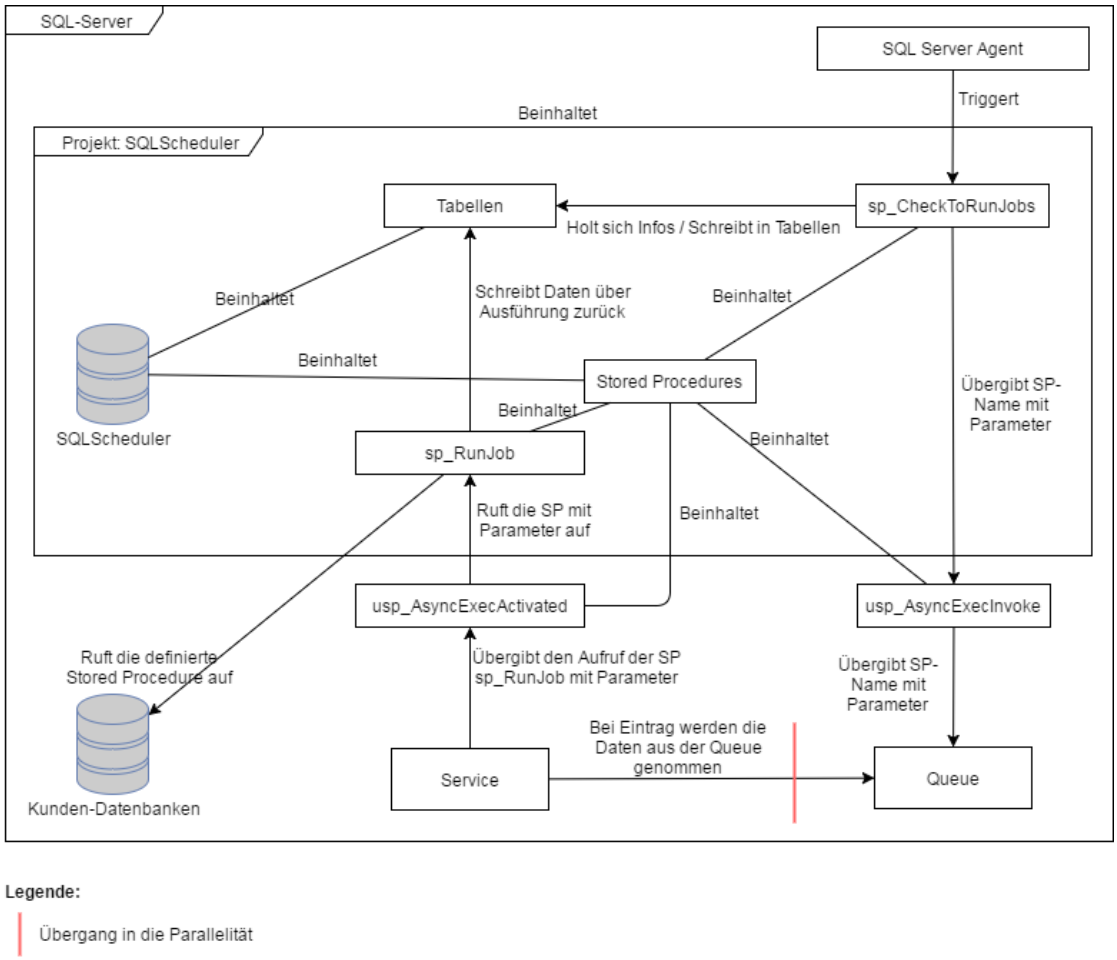

Abbildung 11 Veranschaulichung der Systemgrenzen und deren Zusammenhänge

In diesem Projekt wird die Datenbank SQLScheduler erstellt und die darin enthaltenen Tabellen und Prozeduren.
Die Darstellung erklärt wie die einzelnen Komponenten miteinander arbeiten und welche Komponente wo existiert. Als Erstes triggert der SQL-Server-Agent die Prozedur sp_CheckToRunJobs. Diese schaut nun, über Abfragen auf die Tabellen, ob ein Job laufen muss und ruft die usp_AsyncExecInvoke mit einem Prozeduraufruf, einen String, als Übergabeparameter. Dieser String wird nun in die Queue geschoben und dieser Aufruf ist damit vorbei. Jetzt kommt die Parallelität ins Spiel. Die sp_CheckToRunJobs läuft weiter, da der String in die Queue gelegt wurde. Nun kommt der SQL Broker Service, im Diagram der Service, ins Spiel. Dieser kontrolliert in regelmässigen Abständen ob ein Eintrag in der Queue wartet. Dieser Eintrag wird nun genommen und vom Service mit einem hinterlegten Skript abgearbeitet. Dem Service wurde die Prozedur usp_AsyncExecActivated hinterlegt. Diese ruft nun die Prozedur, welche ihr als String übergeben wurde, auf. Diese Prozedur ist in

jedem Fall sp_RunJob. Diese liest und schreibt wieder auf die Tabellen der SQLScheduler Datenbank. Auch ist diese Prozedur zuständig für den Aufruf anderee Prozeduren auf den Kundendatenbanken.

3.5 Entscheiden

3.5.1 Planung überprüfen und auf Machbarkeit entscheiden

Es gibt im SQL-Server zwei, mir bekannte Wege, um SQL-Skripts parallel laufen zu lassen. Der erste Weg ist mit dem SQL-Agent-Service. Dieser lässt sich mit einer Maske steuern. Der Agent hat den Vorteil, dass er ein ‚GUI' hat, in welchem er schnell einen Überblick über die gewünschten und terminierten Ausführungen hat. Die Tabellenstruktur ist jedoch sehr verwirrend aufgebaut, dafür ist das ‚GUI' sehr strukturiert und übersichtlich.
Der zweite Weg führt über den SQL-Service-Broker; dieser lässt uns eigene Services, auf der gewünschten Datenbank, laufen. Der Nachteil dieser Variante ist, dass man den Service selbst programmieren muss. Das heisst es muss eine Prozedur erstellt und diese dem Service zugewiesen werden. Diese Variante gibt insgesamt mehr Arbeit, da diese Prozedur auch erst erstellt werden muss.

Das Kriterium, welches mich am meisten für eine der Varianten überzeugte, war die Tatsache, dass der SQL-Server-Agent nicht auf allen Editionen zur Verfügung steht. Die Express-Edition, welche nichts kostet, hat nicht alle Features welche kostenpflichtige Versionen haben. So auch der SQL-Server-Agent. Der Service-Broker ist jedoch in allen Editionen verfügbar.

Ich habe mich somit für die SQL-Service-Broker Variante entschieden, welche ich in dieser Individuellen Praktischen Arbeit umsetzen werde.

3.6 Realisieren

3.6.1 Datenbank

Das Datenbank-Skript, „CreateDatabaseSQLScheduler.sql" wurde so angelegt, dass wenn man dasselbe Skript nochmals laufen lässt, die Datenbank, falls vorhanden, gelöscht und wieder erstellt, wird. Sehr wichtig war die Trustworthy Eigenschaft auf der SQLScheduler Datenbank sowie auf den Datenbanken auf denen Jobs ausgeführt werden sollen, auf ‚ON' zu stellen. Dieses ist eine Eigenschaft, welche den externe Assemblys (externer Code) die Berechtigung gibt, auf der Datenbank Code auszuführen (ON) oder nicht (OFF). Da wir einen Service für das Ausführen der Skripts verwenden und dieser als extern gilt, muss diese Eigenschaft auf ON gestellt sein.

```
IF EXISTS (SELECT * FROM sys.databases WHERE name = 'SQLScheduler') BEGIN
    DROP DATABASE SQLScheduler
END
/*
-- Create the database SQLScheduler
*/
CREATE DATABASE SQLScheduler
GO
ALTER DATABASE SQLScheduler SET TRUSTWORTHY ON
GO
```

Abbildung 12 Löschen und Erstellen der Datenbank

3.6.2 Tabellen

Das Skript, „CreateTables.sql“, welches die Tabellen erstellt, wurde, so geschrieben, dass wenn das Skript ein zweites Mal läuft, die Fremdschlüssel und Tabellen gelöscht und neu erstellt werden.

```
-- Drop the Table Jobhistory if it exists
IF EXISTS (SELECT * FROM sys.objects WHERE object_id = OBJECT_ID(N'Jobhistory') AND TYPE IN (N'U')) BEGIN
    -- Drop Constraint
    ALTER TABLE Jobhistory DROP CONSTRAINT Jobhistory_Jobs
    DROP TABLE Jobhistory
END
```

Abbildung 13 Löschen der Constrains

```
-- Create the Table Jobhistory
CREATE TABLE Jobhistory(
    JobhistoryID int NOT NULL IDENTITY(1,1) PRIMARY KEY,
    JobID int NOT NULL,
    ErrorYN bit NOT NULL,
    Description varchar(255),
    StartDateTime datetime,
    EndDateTime datetime
)
```

Abbildung 14 Erstellen der Jobhistory Tabelle

Den Fremdschlüsseln musste einen Namen gegeben werden um genau jene bei einem wiederholten Durchlauf des Skriptes wieder zu löschen. Es gibt einen Fall, wenn die Datenbank nicht vorhanden ist, dass das Skript einen Fehler wirft.

3.6.3 sp_CheckToRunJobs

Diese Prozedur regelt das Ausführen der Jobs. Es kontrolliert pro Aufruf ob ein Job auf einer Datenbank ausgeführt werden muss. Dazu wurde ein sogenannter Cursor verwendet. Dieser erlaubt ein Resultset einer Abfrage zu speichern und mit Hilfe einer Schleife Datensatz für Datensatz abzuarbeiten.

```
/*
 -- Declare cursor
 */
DECLARE crsJobs CURSOR LOCAL FOR
    (SELECT
        JobID, ScheduleID, SchedulestatusID, ScheduleType
    FROM
        Schedules
    WHERE
        RunDateTime < @Now
        and (SELECT Status FROM Schedulestatus WHERE Schedules.SchedulestatusID = Schedulestatus.SchedulestatusID) <> 'Disabled'
    ) FOR READ ONLY
 -- open cursor
OPEN crsJobs
 -- fetch the first row
FETCH NEXT FROM crsJobs INTO @JobID, @ScheduleID, @ScheduleStatus, @IntervallType
 -- get thru all the jobs
WHILE @@FETCH_STATUS = 0 BEGIN
```

Abbildung 15 Veranschaulichung eines Cursors

So kann jeder Datensatz weiter auf Kriterien geprüft werden, ob dieser Schedule tatsächlich schon laufen soll oder mit Intervall. Auch wird geprüft, ob der Job einen Vorgänger hat welcher am laufen ist. Falls falsche Informationen hinterlegt sind, oder gar keine, wird ein Fehlerstring zusammengestellt, welcher am Ende in die Exceptionlog Tabelle geschrieben wird und auf den Eintrag in der Jobhistory Tabelle verweist. Ob Erfolg oder Misserfolg, es wird immer die nächste Startzeit für den Job berechnet, sofern es sich um einen Schedule des Typs Intervall handelt. Bei wöchentlich fixierten Zeiten wird nur die RunDateTime einen Tag nach vorne geschoben, sodass dieser nicht zweimal am selben Tag läuft.

3.6.4 sp_RunJob

Diese Prozedur wird mit gewissen Parametern von der Prozedur „sp_CheckToRunJobs.sql“ in die Queue gegeben, um danach vom Service aufgerufen und ausgeführt, werden. Da dieses Skript auf einer anderen Datenbank läuft, muss der komplette Pfad der Tabellen bei einem Select, Insert, Update und Delete angegeben sein. Ansonsten würde es diese Tabellen auf anderen Datenbanken nicht geben und die Statements würden Fehler produzieren.

```
-- get the runtime
SELECT @RunDateTime = StartDateTime FROM SQLScheduler.dbo.Jobhistory WHERE JobhistoryID = @JobhistoryID
```

Abbildung 16 Veranschaulichung der Abfrage mit Datenbank.Schema.Tabelle

Dieses Skript ruft wieder ein weiteres Skript auf, welches sich auf der Kundendatenbank befindet und den eigentlichen Job bildet. Sobald das Skript auf der Datenbank gelaufen ist, wird „sp_RunJob.sql“ Einträge in die Jobhistory, sowie im Fehlerfall auch in die Tabelle Exceptionlog, schreiben.

3.6.5 Trigger HandleInputs

Es wurde ein Trigger entwickelt, welcher falsche Benutzereingaben abfängt. Es wird auf Plausibilität der Daten geprüft. Die Einträge, welche als Beispiel auf Intervall gesetzt sind und einen Intervall von -1 haben können nicht stimmen.

3.6.6 Parallelität

Die Skripts wurden nicht vom Kandidaten erstellt, jedoch ist Wichtig zu erwähnen, dass es Anpassungen gab. Es geht, wie im oberen Abschnitt, um die Create-, Update-, Insert- und Delete-Statements. Da auch dieses Skript, welches massgebend ist für den Service, extern läuft, ist es wichtig den kompletten Pfad bei allen Statements zu verwenden.

```
update SQLScheduler.dbo.AsyncExecResults set
    [start_time] = @starttime
    , [finish_time] = @finishTime
    , [error_number] = @execErrorNumber
    , [error_message] = @execErrorMessage
    where [token] = @token;
```

Abbildung 17 Veranschaulichung eines Update-Statements mit Datenbank.Schema.Tabelle

Es musste, damit die Parallelität einwandfrei funktioniert, die Datenbank-Eigenschaft Trustworthy auf ‚On' gestellt werden. Dies wurde auf der Datenbank „SQLScheduler" und auf der Test-Datenbank gemacht. Zudem wurden alle Create, Insert, Update und Delete Statements in Try / Catch Blöcke gestellt, da es Probleme gab mit diesen Statements und der Eintrag in der Queue hängen geblieben ist und somit alles blockiert hatte.

3.6.7 User-Manual

Darstellungsdefinition:

- *Spalten welche Pflicht sind werden so dargestellt.*
- *Spalten welche nicht Pflicht sind werden so dargestellt.*
- *Spaltenname Datentyp*
  Beschrieb der Spalte

Das Einfügen eines Jobs muss wie folgt ausgeführt werden und die Reihenfolge muss eingehalten werden:

Als erstes Muss ein Job definiert werden, auf der Tabelle „Jobs". Da zum Beispiel der Job „Datensicherung" auch bei anderen Kunden auf derselben Datenbank läuft, kann es sein, dass es schon einen Job gibt, welcher beschreibt was gemacht werden soll.
Folgende Spalten werden beim Erstellen eines Datensatzes in der Schedule Tabelle eingetragen:

- Name VARCHAR(255)
  Hier wird der Name des Jobs eingetragen. Beispielsweise „Datensicherung".
- Description VARCHAR(255)
  Hier kann eine Beschreibung oder Anmerkungen zum Job hinterlegt werden.

Weiter muss ein Owner vorhanden sein, in welchem Benutzername auf der Datenbank, sowie das Schema zu diesem User, Passwort und Mail hinterlegt ist. Dieser wird in der Tabelle „Owners“ angelegt. Das Mail ist in diesem Stadium noch nicht wichtig, jedoch könnte nach der IPA die Funktionalität entwickelt werden, dass im Fehler- oder Erfolgsfall ein Mail versendet wird. Dies ist jedoch nicht Teil dieser Individuellen Praktischen Arbeit.
Folgende Spalten sind Pflicht und müssen beim Erstellen eines Datensatzes in der Jobs Tabelle eingetragen sein:

- UserName VARCHAR(50)
  Dies ist der Benutzername, welcher für die Sicherheit entscheidend ist. Dieser Benutzer sollte nur Zugriff auf die Kundendatenbank haben.
- Password VARCHAR(50)
  Das Passwort gehört zum UserName.
- Shema VARCAHR(50)
  Das Schema ist auch ein Teil der Sicherheitsmassnamen. Diese Spalte beinhaltet das Schema auf welches der Benutzer auf der Kundendatenbank Zugriff hat.
- Email VARCHAR(100)
  Hier wird die Kunden-Emailadresse hinterlegt. Diese Person hinter der Mailadresse ist für die Kommunikation zwischen uns und dem Kunden zuständig.
- Description VARCHAR(255)
  Hier kann eine Beschreibung oder Anmerkungen zum Job hinterlegt werden.

Zudem muss eine Datenbank zum jeweiligen Kunden vorhanden sein, diese wird in der Tabelle „Databases“ hinterlegt.
Folgende Spalten sind Pflicht und müssen beim Erstellen eines Datensatzes in der Databases Tabelle eingetragen sein:

- Name VARCHAR(30)
  Diese Spalte wird benötigt um den Datenbanknamen der Kundendatenbank zu speichern.
- Description VARCHAR(255)
  Hier kann eine Beschreibung oder Anmerkungen zum Job hinterlegt werden.

Nun, da wir alle benötigten Referenzen haben, wird der Schedule erstellt. Der Schedule ist der eigentliche Job. Hier wird hinterlegt, welche Prozedur wann, in welchem Intervall, bei welchem Kunden laufen wird.
Folgende Spalten werden beim Erstellen eines Datensatzes in der Schedule Tabelle eingetragen:

- OwnerID INT
  Dies ist die ID des Owners.
- JobID INT
  Dies ist die ID des Jobs unter welchem der Schedule läuft.

- SchedulestatusID INT
  Diese ID beschreibt auf der Schedulestatus-Tabelle in welchem Status sich dieser Schedule befindet. Es gibt folgende Status:
  - „Ready“ Der Schedule ist ‘ready to go’.
  - „Running“ In diesem Status läuft der Schedule.
  - „Disabled“ Dieser Status erhalten Schedules welche nicht mehr laufen.
- DatabaseID INT
  Dies ist die ID der Datenbank.
- Intervall INT
  Bei Intervall Schedules wird hier der Zeitabstand in Minuten angegeben, bei wöchentlichen auszuführenden Schedules wird 0 eingetragen.
- RunDateTime DATETIME
  Hier wird das Datum eingetragen wann der Schedule laufen soll. Am Anfang sollte hier der Eintrag analog des StartDateTime sein, nachher wird dies berechnet und verändert sich wie gewünscht.
- SpName VARCHAR(50)
  Dies ist der Prozedurname jener Prozedur die auf der Kundendatenbank ausgeführt werden soll. Diese muss sich auf der Kundendatenbank befinden.
- ScheduleType INT
  Hier wird eine Zahl eingetragen, welche so implementiert wurden Es sind zwei Varianten implementiert worden:
  - 1
    Dies bedeutet für den SQLScheduler dass dieser Schedule mit einer Minuten-Intervall läuft. Hier wird kein Wochentag beachtet.
  - 2
    Dies bedeutet der Schedule wurde auf einen Wochentag und eine Startzeit definiert und wird, sobald dieser Zeitpunkt in der Woche erreicht wird, ausgeführt.
- StartDateTime DATETIME
  Dieses Datum ist dafür zuständig, dass der Schedule nicht vor diesem Datum gestartet wird. Dies ermöglicht eine Terminierung in die Zukunft.
- Description VARCHAR(255)
  Hier können Kommentare oder Beschreibungen eingefügt werden.
- PrevScheduleID INT
  Falls der Schedule abhängig von einem anderen Schedule ist, wird hier die ID vom Vorgänger eingetragen, ansonsten steht hier NULL.
- ExpireDateTime DATETIME
  Hier wird ein Enddatum des Schedules eingetragen. Falls nichts steht, wird der Schedule bis auf weiteres laufen.
- DayOfWeek INT
  Wenn der ScheduleType 2 ist, wöchentliche Terminierung, muss hier der Wochentag

eingetragen werden, in Nummern 1 bis 7, wann der Schedule laufen soll. Wichtig ist zu beachten, dass auf unserer Datenbank Tag 1 der Sonntag ist. Dies muss aber von System zu System überprüft werden, da dies eine lokale Einstellung auf dem Server auf welchem der SQL Server läuft, ist.
Beispielsweise will man den Montag eintragen, so muss 2 in diese Spalte geschrieben werden. Bei Freitag muss 6 eingetragen werden.

- DayTime VARCHAR(10)
  Bei ScheduleType 2 wird hier die Tageszeit eingetragen, wann der Job laufen soll. Dieser wird als String in die Spalte eingetragen nach folgendem Schema [hhmm]. Wichtig dabei ist zu beachten, dass auch die Stunden immer zweistellig sind. So muss zum Beispiel 2:00 Uhr wie folgt eingetragen werden: ‚0200'. Es darf kein Leerschlag oder andere Zeichen in diese Spalte.

Die weitern Tabellen, Jobhistory und Exceptionlog sind reine Auswertungstabellen. Der Benutzer wird hier keine Einträge erstellen.
Die Tabelle Jobhistory beinhaltet folgende Spalten:

- JobhistoryID
  Die ID der Jobhistory
- JobID
  Die ID des ausgeführten Jobs
- ScheduleID
  ID des Schedules von welchem dies die History ist.
- ErrorYN
  Es gibt hier zwei mögliche Varianten, nämlich:
  - 0
    Die Ausführung wurde wie geplant durchgeführt.
  - 1
    Es trat während der Verarbeitung ein Fehler auf, welcher in der Tabelle Exceptionlog mit der JobhistoryID gesucht werden kann.
- Description
  Dies ist eine Spalte, welche vom Benutzer gebraucht werden kann.
- StartDateTime
  Dies ist die Startzeit der Ausführung des Schedules.
- EndDateTime
  Dies ist der Endzeitpunkt der Ausführung des Schedules.

Die Tabelle Exceptionlog sieht im Aufbau folgendermassen aus:

- ExceptionlogID
  Dies ist die ID der Exception.

- JobID
  Hier wird die ID des Jobs eingetragen.
- JobName
  Hier wird der Name des Jobs eingetragen
- RunDateTime
  Diese ist die Startzeit der Ausführung des Schedules
- ErrorDateTime
  Hier wird die Zeit, zu welcher der Fehler aufgetreten ist, dokumentiert.
- ErrorMessage
  Die Meldung, welche vom SQL-Server geworfen wird, wird hier eingetragen.
- DatabaseName
  Der Datenbankname des gescheiterten Schedules wird hier eingetragen.
- SpName
  Die Prozedur, welche für den Fehler verantwortlich ist, wird hier vermerkt.
- JobhistoryID
  Die Jobhistory, welche nun einen Error vermerkt hat, wird hier eingetragen.
- Description
  Diese Spalte kann vom Benutzer verwendet werden.

3.6.8 Installations-Manual

Systemvoraussetzungen:

- Mindestens einen SQL-Server 2012, Version 12.0.4232, Standard Edition (Andere Versionen und Editionen wurden nicht getestet)
- Microsoft SQL-Management-Studio

Anleitung:

Es müssen folgende Skripts in der angegebenen Reihenfolge mit dem SQL-Management-Studio ausgeführt werden:

- CreateDatabase.sql
- CreateTables.sql
- TriggerHandleInputs.sql
- usp_AsyncExecActivated.sql
- usp_AsyncExecInvoke.sql
- asy_CreateTables.sql
- asy_CreateQueueAndService.sql
- sp_RunJob.sql
- sp_CheckToRunJobs.sql

3.7 Kontrollieren

3.7.1 Test-Umfeld

Das Test-Umfeld beschreibt die Rahmenbedingungen, welche für die auszuführenden Tests gelten.
Es wird auf einem Microsoft-SQL-Server 2012 Version 12.0.4232, Standardedition, getestet. Dieser verfügt über einen SQL-Server-Agent um die erstellten Prozeduren zu triggern. Für die Verbindung auf den SQL-Server wird ein SQL-Management-Studio, Version 13.0.16106.4, verwendet.

Es wird auf einer Test-Datenbank getestet, welche eigens dazu erstellt wurde. Diese Datenbank ist zwar online, jedoch nicht produktiv im Einsatz. Es werden auch nicht gültige Werte geprüft.

Es werden dem Tester Skripts gegeben, welche dafür zuständig sind, dass die Schedules im richtigen Zustand sind. Er bekommt auch pro Test einen Befehl um die Tests auszuwerten.

Es wurden zwei Prozeduren auf der Testumgebung erstellt. Die Prozedur „sp_CheckSuccess“ schreibt einen Datensatz direkt in die Tabelle TestResult. Die Prozedur „sp_CheckSuccessDelay“ wartet 20 Sekunden bis der Eintrag gemacht wird.

Fich3.7.2 Testmittel

Folgendes wird für die Tests gebraucht:

- Microsoft-SQL-Server 2012, Version 12.0.4232.
- SQL-Server-Agent.
- SQL-Management-Studio, Version 13.0.16106.4.
- Die erstellte Datenbank SQLScheduler.
- Die erstellte Datenbank IpaTestDB.
- Die erstellten Tabellen auf der SQLScheduler Datenbank.
- Die erstellte Tabelle auf der IpaTestDB Datenbank.
- Die erstellte Prozedur „sp_CheckToRunJobs“ auf der SQLScheduler Datenbank.
- Die erstellte Prozedur „sp_CheckToRunJobs“ auf der SQLScheduler Datenbank.
- Die angepasste Prozedur „usp_AsyncExecActivated“ auf der SQLScheduler Datenbank.
- Die angepasste Prozedur „usp_AsyncExecInvoke“ auf der SQLScheduler Datenbank.
- Die erstellte Queue auf der SQLScheduler Datenbank.
- Den erstellten Service auf der SQLScheduler Datenbank.

Der Tester wird nun mithilfe von Skripts, welche für ihn vorbereitet wurden, die Testfälle bearbeiten. Die Skripts sind so aufgebaut, dass neue Einträge in die Schedules Tabelle gemacht werden. Nachdem diese Einträge gemacht wurden, wird die Prozedur sp_CheckToRunJobs aufgerufen, gegebenenfalls mehrfach, und eine Auswertung der gemachten Jobs erstellt. Der Tester wird nun die Beobachtungen auf der ausgegebenen Datensätze, im SQL-Management-Studio, machen. Anschliessend werden die Beobachtungen dokumentiert.

3.7.3 Test-Fälle und Auswertung des ersten Testes

Tester: Romano Sabbatella			Datum: 2017.04.27	
Nr.	**Testfall-Beschreibung**	**Skriptname**	**Erwartetes Resultat**	**Test erfolgreich Ja/Nein**
1	Wird der Intervall, welcher dem Schedule hinterlegt wurde, auch eingehalten?	Case1.sql	Es werden pro ScheduleID je drei Einträge erwartet, mit je dem Intervall Unterschied in der Runtime.	Ja
2	Startet das Schedule mit wöchentlichem Zyklus, samt Zeitterminierung, zum richtigen Zeitpunkt?	Case2.sql	Das erste Resultat ist der durchgelaufene Schedule, welcher für am Morgen um 1:00 Uhr terminiert war und nun neu auf morgen terminiert wurde. Das Zweite Resultat ist die Jobhistory. Hier darf nur ein Eintrag kommen, da der zweite Schedule nicht laufen durfte.	Ja
3	Wird bei Fehler in die Exceptionlog-Tabelle geschrieben?	Case3.sql	Es gibt einen Eintrag mir den wichtigsten Informationen in der Exceptionlog Tabelle.	Ja
4	Wird beim Fehlerfall das RunDateTime trotzdem wieder berechnet?	Case4.sql	Das erste Resultat ist der Schedule vor dem bearbeiten. Das zweite ist der neu berechnete Schedule, das dritte ist die Exception Tabelle. Das erste Resultat ist nicht dasselbe Datum wie das Resultat 2 und im Resultat 3 befindet sich ein Eintrag.	Ja
5	Wird bei einem Fehlerfall die JobhistoryID richtig auf der Exceptionlog Tabelle hinterlegt?	Case5.sql	Die JobhistoryID auf der Tabelle Exceptionlog ist die ID der Jobhistory.	Ja

6	Wird das parallele Ausführen zweier Schedules gleichzeitig auf derselben Datenbank richtig ausgeführt?	Case6.sql	Das Resultat mit der Beschreibung „without delay“ sollte zwischen dem „CreationDate“ und den „FinishDate“ des Resultates, mit der Beschreibung „with delay“ liegen.	Nein
7	Wird die Ausführung eines Schedule, welcher einen Vorgänger hat der noch läuft, verhindert?	Case7.sql	Die erste Prozedur wird durchgeführt, wobei die zweite erst bei einem weiteren Durchlauf startet.	Ja
8	Was passiert wenn der Wochentag auf 0 gesetzt ist und der ScheduleTyp auf 2?	Case8.sql	Das Skript wirft einen Fehler.	Ja
9	Was passiert wenn der Wochentag auf 8 gesetzt ist und der ScheduleTyp auf 2?	Case9.sql	Das Skript wirft einen Fehler.	Ja
10	Was passiert wenn der Intervall auf 0 gesetzt ist und der ScheduleTyp auf 1 (Intervall)?	Case10.sql	Das Skript wirft einen Fehler.	Ja
11	Was passiert wenn der ScheduleTyp auf 3 gesetzt wird?	Case11.sql	Das Skript wirft einen Fehler.	Ja
12	Was passiert wenn der ScheduleTyp auf 0 gesetzt wird?	Case12.sql	Das Skript wirft einen Fehler.	Ja

3.7.4 Erkannte Fehler

Das parallele Ausführen zweier Prozeduren machte Probleme. Es wurde nicht wie erwartet Parallel sondern Sequentiell ausgeführt. Der Fehler wurde, mittels SQL-Server-Profiler gesucht, da die Vermutung war, es habe etwas mit Transaktionen zu tun. Nach durchforsten der Dokumentationen wurde eine Fehlkonfiguration der Queue gefunden, welche nur einen Empfänger der Nachricht vorsah. Dies wurde nun auf 5 gesetzt und die Tests nochmals durchgeführt.

3.4.5 Test-Fälle und Auswertung des zweiten Testes

Tester: Romano Sabbatella			Datum: 2017.04.27	
Nr.	**Testfall-Beschreibung**	**Skriptname**	**Erwartetes Resultat**	**Test erfolgreich Ja/Nein**
1	Wird der Intervall, welcher dem Schedule hinterlegt wurde, auch eingehalten?	Case1.sql	Es werden pro ScheduleID je drei Einträge erwartet mit je dem Intervall Unterschied in der Runtime.	Ja
2	Startet das Schedule mit wöchentlichem Zyklus, samt Zeitterminierung, zum richtigen Zeitpunkt?	Case2.sql	Das erste Resultat ist der durchgelaufene Schedule, welcher für am Morgen um 1:00 Uhr terminiert war und nun neu auf morgen terminiert wurde. Das Zweite Resultat ist die Jobhistory. Hier darf nur ein Eintrag kommen, da der zweite Schedule nicht laufen durfte.	Ja
3	Wird bei Fehler in die Exceptionlog-Tabelle geschrieben?	Case3.sql	Es gibt einen Eintrag mir den wichtigsten Informationen in der Exceptionlog Tabelle.	Ja

4	Wird beim Fehlerfall das RunDateTime trotzdem wieder berechnet?	Case4.sql	Das erste Resultat ist der Schedule vor dem bearbeiten, das zweite ist der neu berechnete Schedule, das dritte ist die Exception Tabelle. Das erste Resultat ist nicht dasselbe Datum wie das Resultat 2 und im Resultat 3 befindet sich ein Eintrag.	Ja
5	Wird bei einem Fehlerfall die JobhistoryID richtig auf der Exceptionlog Tabelle hinterlegt?	Case5.sql	Die JobhistoryID auf der Tabelle Exceptionlog ist die ID der Jobhistory.	Ja
6	Wird das parallele Ausführen zweier Schedules gleichzeitig auf derselben Datenbank richtig ausgeführt?	Case6.sql	Das Resultat mit der Beschreibung „without delay“ sollte zwischen dem „CreationDate“ und den „FinishDate“ des Resultates, mit der Beschreibung „with delay“ liegen.	Ja
7	Wird mit der Ausführung eines Schedule, welcher einen Vorgänger hat der noch läuft, verhindert?	Case7.sql	Die erste Prozedur wird durchgeführt, wobei die zweite erst bei einem weiteren Durchlauf startet.	Ja
8	Was passiert wenn der Wochentag auf 0 gesetzt ist und der ScheduleTyp auf 2?	Case8.sql	Das Skript wirft einen Fehler.	Ja
9	Was passiert wenn der Wochentag auf 8 gesetzt ist und der ScheduleTyp auf 2?	Case9.sql	Das Skript wirft einen Fehler.	Ja
10	Was passiert wenn der Intervall auf 0 gesetzt ist und der ScheduleTyp auf 1 (Intervall)?	Case10.sql	Das Skript wirft einen Fehler.	Ja

11	Was passiert wenn der ScheduleTyp auf 3 gesetzt wird?	Case11.sql	Das Skript wirft einen Fehler.	Ja
12	Was passiert wenn der ScheduleTyp auf 0 gesetzt wird?	Case12.sql	Das Skript wirft einen Fehler.	Ja

3.7 Auswertung

3.7.1 Vergleichen der Planung und Umsetzung

Es wurde alles ausgeführt wie geplant. Die grössten Unterschiede, oder besser gesagt Abweichungen, hat es bei der Implementierung der Parallelität gegeben. Es wurde in der Planung grosszügig Zeit eingeplant, welche schlussendlich doch nicht ausreichte diese Fehlerfrei zu implementieren. In der Testphase wurden Verbesserungen gemacht, welche zur Umsetzung der Parallelität gehörten, jedoch während der Realisierung nicht wahrgenommen wurden. Die Realisierung ansonsten ging recht gut und zügig voran. Die Planung im Allgemeinen, besonders das Entity-Relationship-Model, erleichterte die Realisierung insgeheim. Gegen den Schluss konnte, dank guter und sauberer Planung, noch viel Zeit in die Dokumentation investiert werden.

Das Fazit zu dieser Individuellen Praktischen Arbeit ist, dass sich eine gute Planung auszahlt. Es wird viel Zeit für die Planung gebraucht. Es zahlt sich aus in der Realisierung, weil man dort viel schneller vorankommt, da die Problematiken während der Planung schon angedacht wurden.

3.7.2 Arbeiten nach Abschluss der IPA

Im Zusammenhang mit meiner Individuellen-Praktischen-Arbeit gibt es keine Verbesserungen. Diese Arbeit wurde im Gedanken entwickelt, einen Scheduler auf der kostenfreien SQL-Server Express-Edition zur Verfügung zu haben. Dies wird im Anschluss der IPA von uns getestet. Sobald dies einwandfrei funktioniert, werden wir anfangen unsere Applikation, bei kleinen Betrieben, mit SQL-Server Express-Editionen anzubieten.

3.7.3 Reflexion

Ich bin sehr froh, diese Arbeit abschliessen zu können. Es war ein sehr spannendes und informatives Projekt, auch wenn nicht immer alles so lief wie gewollt. Es gab einige Probleme, in der Implementierung der Parallelität. Durch diese Probleme kam ich immer wieder ins Schwitzen, da ich nur begrenzte Zeit zur Verfügung hatte. Es gab Abende, wo ich an nichts anderes Denken konnte. Trotzdem hat mir diese Arbeit gefallen und neue Wege innerhalb von SQL-Servern aufgezeigt, da paralleles Arbeiten in SQL nicht alltäglich ist. Auch wurde mir aufgezeigt, dass es noch Vieles gibt, was ich noch nicht weiss, auch wenn ich tagtäglich mit SQL zu tun habe. Der Abschluss dieser Arbeit ist ein weiterer Meilenstein in meinem Leben. Falls mich heute jemand fragen würde was ich anders machen würde; ich würde es genau gleich angehen.

4 Schlusswort

Diese Arbeit war wohl das grösste und wichtigste Projekt während meiner Ausbildung zum Informatiker, Fachrichtung Applikationsentwicklung. Dadurch, dass mein Alltag im Betrieb mehr mit Fehlersuche, Korrekturen und Anpassungen überschattet wird, war es eine sehr schöne Erfahrung einmal ein Projekt im grösseren Stil machen zu dürfen. Die „normale“ Zeit für die erwähnten Tätigkeiten ist im Maximum einen halben bis ganzen Tag. Ich hatte auch in der Schule noch kein Projekt, welches länger als einen Tag ging. Daher war diese Herausforderung einen so langen Zeitraum hoch konzentriert zu arbeiten, neu für mich. Es war nicht einfach so viele Seiten auf Papier zu bringen. Nachdem ich den Einstieg in dieses Projekt gefunden hatte, kam ich sehr gut voran.

Ich habe sehr vieles in diesem Projekt gelernt. In der Schule behandelt man die Projektplanung eher als etwas Theoretisches, was ja durchaus stimmt. Dies nun aber anzuwenden und mit der Realisierung zu verbinden, war ein tolles Erlebnis, welches ich sehr genoss. Ich bin sehr zufrieden mit meinen getroffenen Entscheidungen, sowie meine Herangehensweise an das Projekt.

Ich möchte mich bei allen Beteiligten für den reibungslosen Ablauf dieses Projektes bedanken. Speziell möchte ich meinem Fachvorgesetzten, …, meinen Dank aussprechen. Nicht nur, dass er mir dieses Projekt anvertraut hat, er hat mich seit einem Jahr mit bestem Wissen und Gewissen ausgebildet und auf meinem Berufsweg begleitet.

5 Glossar

Aplix
Aplix heisst die ERP-Software welche die Firma Boreas AG vertreibt.

Applikation
Eine Applikation ist ein Programm welches für einen bestimmten Zweck entwickelt wurde.

Assembly
Programmcode welcher bereit ist zur Ausführung.

Attribute
Spaltenbezeichnungen innerhalt von Tabellen.

Constraints
Constraints bezeichnen in einer Datenbank Tabellenbeziehungen untereinander.

Cursor
Der Cursor führt eine Abfrage auf Tabellen aus und speichert das Resultat in sich.

ER-Model

Das Entity-Relationship-Model ist eine Art der Darstellung der Tabellen innerhalb einer Datenbank. Es zeigt die Beziehungstypen, Spaltennamen und deren Detantyp auf.

ERP-Lösung
Das Enterprice-Resource-Planning-Lösung, auf Deutsch Geschäftsressourcenplanung, ist eine Betriebswirtschaftliche Softwarelösung zur Abwicklung von Geschäftsprozessen.

Errors
Auf Deutsch: Fehler.

Exceptionlog
Dies ist eine erstellte Tabelle innerhalb der erstellten Datenbank. Auf Deutsch heisst dies Fehlerprotokoll.

GUI
Das Graphical User Interface, kurz GUI, ist die Grafischebenutzeroberfläche eines Programms.

IPERKA
IPERKA (Informieren, Planen, Entscheiden,Realisieren, Kontrollieren, Auswerten) ist eine Projektmanagement-Methode.

‚n' zu ‚n' Beziehung
‚n' zu ‚n' Beziehungen sind im englischen unter ‚many to many' bekannt. Diese sagt aus, dass zum Beispiel viele Personen viele Freunde haben.

Prozedural
Prozedurale Programmierung hält sich Strikt an den Ablauf der Anweisungen.

Resultset
Dies ist ein Resultat welches aus mehrere Datensätzen oder auch Spalten besteht.

Scheduler
Der Scheduler ist Disponent von terminierten Aufträgen. Er ist in der Lage an einem festgelegten Zeitpunkt eine bestimmte Aufgabe zu erledigen.

Skripts
Skripts ist eine Textstruktur und ist mit einem Drehbuch zu vergleichen.

SQL-Profiler
Dies ist ein Tool von Microsoft welches mit dem SQL-Management-Studio kommt. Es erlaubt dem Benutzer alle Aktivitäten auf einem SQL-Server zu überwachen.

SQL-Server
Ein SQL-Server ist ein Datenbankmanagementsystem.

SQL-Service-Broker
Der SQL-Service-Broker bietet die Möglichkeit einer Warteschlangen-Anwendung innerhalb des SQL-Servers.

String
Dies ist ein Datentyp und beschreibt eine Zeichenkette.

Trustworthy
Dies ist eine Eigenschaft auf Microsoft SQL-Datenbanken welche verhindert, dass schädliche Programme auf die Datenbank zugreifen können.

Websummary
Eine Zusammenfassung für die Prüfungskommission mit Text und einer Grafik.

6 Versionsverzeichnis

Version	Name	Datum	Beschreibung der Tätigkeit
0.1	Romano Sabbatella	18.04.2017	- Aufgabenstellung aus PkOrg übernommen - Kapitel 1, 2.9 – 2.11, 3.3 erfasst - Arbeitsprotokoll Tag 1
0.2	Romano Sabbatella	19.04.2017	- Kapitel 3.1.1, 3.2, 3.4 erfasst - Arbeitsprotokoll Tag 2
0.3	Romano Sabbatella	20.04.2017	- Kapitel 3.5 erfasst - Arbeitsprotokoll Tag 3
0.4	Romano Sabbatella	21.04.2017	- Arbeitsprotokoll Tag 4
0.5	Romano Sabbatella	24.04.2017	- Kapitel 3.6.1 – 3.6.6 erfasst - Arbeitsprotokoll Tag 5
0.6	Romano Sabbatella	25.04.2017	- Kapitel 3.6.6 – 3.6.8 erfasst - Arbeitsprotokoll Tag 6
0.7	Romano Sabbatella	26.04.2017	- Arbeitsprotokoll Tag 7
0.8	Romano Sabbatella	27.04.2017	- Kapitel 3.7 erfasst - Arbeitsprotokoll Tag 8
0.9	Romano Sabbatella	28.04.2017	- Kapitel 3.7 erfasst - Arbeitsprotokoll Tag 9
1.0	Romano Sabbatella	02.05.2017	- Kapitel 3.8 , 4 - 8 erfasst - Arbeitsprotokoll Tag 10

7 Abbildungsverzeichnis

8 Quellenverzeichnis

Bild des MS-SQL-Server Logos
http://www.iperiusbackup.net/wp-content/uploads/2016/05/1768.sql_logo.png
Abrufdatum: 19.04.2017 8:30 Uhr

Asynchroner Aufruf für Stored Procedure auf MS-SQL-Server
http://rusanu.com/2009/08/05/asynchronous-procedure-execution/
Abrufdatum: 21.04.2017 11:00 Uhr

SQL Server Editionen Informationen
https://msdn.microsoft.com/de-ch/library/cc645993.aspx
Abrufdatum: 24.04.2017 10:00 Uhr

SQL-Server Service Broker Dokumentation
https://msdn.microsoft.com/de-de/library/bb522893.aspx
Abrufdatum : 25.04.2017 13:40 Uhr

SQL-Server Queue Dokumentation
https://msdn.microsoft.com/de-ch/library/ms190495.aspx
Abrufdatum : 25.04.2017 13:40 Uhr

9 Anhang

9.1 CreateDatabase.sql

```
/*
------------------------------------------------------------------------------------------
---------------
Autor:.............Romano Sabbatella
Date:..............2017.04.19
Description:.......Creates the database SQLScheduler
Versions:..........2017.04.19 / RS Creat script
...................xxxx.xx.xx / xx xxxxxx
------------------------------------------------------------------------------------------
---------------
*/

USE master
/*
-- Drop the database if it exist
*/
IF EXISTS (SELECT * FROM sys.databases WHERE name = 'SQLScheduler') BEGIN
       DROP DATABASE SQLScheduler
END
/*
-- Create the database SQLScheduler
*/
CREATE DATABASE SQLScheduler
GO
ALTER DATABASE SQLScheduler SET TRUSTWORTHY ON
GO
```

9.2 CreateTables.sql

```
/*
------------------------------------------------------------------------------------------
---------------
Autor:.............Romano Sabbatella
Date:..............2017.04.19
Description:.......This script creates all the tables you need for the SQLScheduler
Versions:..........2017.04.19 / RS Creat script
...................2017.04.20 / RS Added some fields in the table scheduler
...................xxxx.xx.xx / xx
------------------------------------------------------------------------------------------
---------------
*/
-- Use the Database
USE SQLScheduler

/*
-- Drop all tables
*/

-- Drop the Table Jobprotocol if it exists
IF EXISTS (SELECT * FROM sys.objects WHERE object_id = OBJECT_ID(N'Exceptionlog') AND TYPE
IN (N'U')) BEGIN
       -- Drop Constraint
       ALTER TABLE Exceptionlog Drop CONSTRAINT Exceptionlog_Jobs
       ALTER TABLE Exceptionlog Drop CONSTRAINT Exceptionlog_Jobhistory
       DROP TABLE Exceptionlog
END

-- Drop the Table Jobhistory if it exists
IF EXISTS (SELECT * FROM sys.objects WHERE object_id = OBJECT_ID(N'Jobhistory') AND TYPE IN
(N'U')) BEGIN
```

```
	-- Drop Constraint
	ALTER TABLE Jobhistory DROP CONSTRAINT Jobhistory_Jobs
	ALTER TABLE Jobhistory DROP CONSTRAINT Jobhistory_Schedules
	DROP TABLE Jobhistory
END

-- Drop the Table Schedules if it exists
IF EXISTS (SELECT * FROM sys.objects WHERE object_id = OBJECT_ID(N'Schedules') AND TYPE IN
(N'U')) BEGIN
	-- Drop Constraint
	ALTER TABLE Schedules DROP CONSTRAINT Schedules_Owners
	ALTER TABLE Schedules DROP CONSTRAINT Schedules_Jobs
	ALTER TABLE Schedules DROP CONSTRAINT Schedules_ScheduleStatus
	ALTER TABLE Schedules DROP CONSTRAINT Schedules_Databases
	ALTER TABLE Schedules DROP CONSTRAINT Schedules_Schedules
	DROP TABLE Schedules
END

-- Drop the Table Owners if it exists
IF EXISTS (SELECT * FROM sys.objects WHERE object_id = OBJECT_ID(N'Owners') AND TYPE IN
(N'U')) BEGIN
	DROP TABLE Owners
END

-- Drop the Table Databases if it exists
IF EXISTS (SELECT * FROM sys.objects WHERE object_id = OBJECT_ID(N'Databases') AND TYPE IN
(N'U')) BEGIN
	DROP TABLE [Databases]
END

-- Drop the Table Schedulestatus if it exists
IF EXISTS (SELECT * FROM sys.objects WHERE object_id = OBJECT_ID(N'Schedulestatus') AND
TYPE IN (N'U')) BEGIN
	DROP TABLE Schedulestatus
END

-- Drop the Table Jobs if it exists
IF EXISTS (SELECT * FROM sys.objects WHERE object_id = OBJECT_ID(N'Jobs') AND TYPE IN
(N'U')) BEGIN
	DROP TABLE Jobs
END

/*
-- TABLE Jobs
*/
--Create the table Databases
CREATE TABLE [Databases](
	DatabaseID int NOT NULL IDENTITY(1,1) PRIMARY KEY,
	[Name] varchar(30) NOT NULL,
	[Description] varchar(255)
)

-- Create the Table Jobs
CREATE TABLE Jobs(
	JobID int NOT NULL IDENTITY(1,1) PRIMARY KEY,
	[Name] varchar(255),
	[Description] varchar(255),
	CreationDate datetime
)

/*
-- TABLE Jobhistory
*/
-- Create the Table Jobhistory
CREATE TABLE Jobhistory(
	JobhistoryID int NOT NULL IDENTITY(1,1) PRIMARY KEY,
```

```
	JobID int NOT NULL,
	ScheduleID int NOT NULL,
	ErrorYN bit NOT NULL,
	[Description] varchar(255),
	StartDateTime datetime,
	EndDateTime datetime
)

/*
-- TABLE Jobprotocol
*/
-- Create the Table Jobhistory
CREATE TABLE Exceptionlog(
	ExceptionLogID int NOT NULL IDENTITY(1,1) PRIMARY KEY,
	JobID int NOT NULL,
	JobName varchar(255),
	RunDateTime datetime,
	ErrorDateTime datetime NOT NULL,
	ErrorMessage varchar(max) NOT NULL,
	[Name] varchar(30) NOT NULL,
	SpName varchar(50) NOT NULL,
	JobhistoryID int NOT NULL,
	[Description] varchar(255)
)

/*
-- TABLE Owners
*/
-- Create the Table Owners
CREATE TABLE Owners(
	OwnerID int NOT NULL IDENTITY(1,1) PRIMARY KEY,
	UserName varchar(50) NOT NULL,
	[Password] varchar(50) NOT NULL,
	Shema varchar(50) NOT NULL,
	[Description] varchar(255),
	EMail varchar(100)
)

/*
-- TABLE Schedules
*/
-- Create the Table Schedules
CREATE TABLE Schedules(
	ScheduleID int NOT NULL IDENTITY(1,1) PRIMARY KEY,
	OwnerID int NOT NULL,
	JobID int NOT NULL,
	SchedulestatusID int NOT NULL,
	Intervall int NOT NULL,
	RunDateTime datetime NOT NULL,
	DatabaseID int NOT NULL,
	SpName varchar(50) NOT NULL,
	ScheduleTyp int NOT NULL,
	StartDateTime datetime NOT NULL,
	[Description] varchar(255),
	PrevScheduleID int,
	ExpireDateTime datetime,
	[DayOfWeek] int,
	DayTime varchar(10)
)

/*
-- TABLE Schedulestatus
*/
-- Create the Table Schedulestatus
CREATE TABLE Schedulestatus(
	SchedulestatusID int NOT NULL IDENTITY(1,1) PRIMARY KEY,
```

```
	[Status] varchar(30) NOT NULL,
	[Description] varchar(255)
)

/*
-- ADD Foreign Keys
*/
-- Table Schedules
ALTER TABLE Schedules ADD CONSTRAINT Schedules_Owners FOREIGN KEY (OwnerID) REFERENCES
Owners(OwnerID)
ALTER TABLE Schedules ADD CONSTRAINT Schedules_Jobs FOREIGN KEY (JobID) REFERENCES
Jobs(JobID)
ALTER TABLE Schedules ADD CONSTRAINT Schedules_Schedules FOREIGN KEY (PrevScheduleID)
REFERENCES Schedules(ScheduleID)
ALTER TABLE Schedules ADD CONSTRAINT Schedules_ScheduleStatus FOREIGN KEY
(SchedulestatusID) REFERENCES Schedulestatus(SchedulestatusID)
ALTER TABLE Schedules ADD CONSTRAINT Schedules_Databases FOREIGN KEY (DatabaseID)
REFERENCES [Databases](DatabaseID)
-- Table Jobhistory
ALTER TABLE Jobhistory ADD CONSTRAINT Jobhistory_Jobs FOREIGN KEY (JobID) REFERENCES
Jobs(JobID)
ALTER TABLE Jobhistory ADD CONSTRAINT Jobhistory_Schedules FOREIGN KEY (ScheduleID)
REFERENCES Schedules(ScheduleID)
-- Table Jobprotocol
ALTER TABLE Exceptionlog ADD CONSTRAINT Exceptionlog_Jobs FOREIGN KEY (JobId) REFERENCES
Jobs(JobID)
ALTER TABLE Exceptionlog ADD CONSTRAINT Exceptionlog_Jobhistory FOREIGN KEY (JobhistoryID)
REFERENCES Jobhistory(JobhistoryID)
```

9.3 TriggerHandleInputs.sql

```
/*
-------------------------------------------------------------------------------------------
---------------
Autor:.................Romano Sabbatella
Date:..................2017.04.20
Description:..........This script creates a trigger on the datebase sql scheduler.
.....................This trigger handles wrong inputs from User
Versions:.............2017.04.20 / RS Creat script
.....................xxxx.xx.xx / xx
-------------------------------------------------------------------------------------------
---------------
*/
CREATE TRIGGER [dbo].[Schedules_HandelInserts] ON [dbo].[Schedules] FOR UPDATE, INSERT
AS
-------------------------------------------------------------
-- prepare
-------------------------------------------------------------
DECLARE @ScheduleID int
DECLARE @Weekday int
DECLARE @ScheduleTyp int
DECLARE @Intervall int
-------------------------------------------------------------
-- work
-------------------------------------------------------------
DECLARE crsEintraege CURSOR FOR (SELECT ScheduleID, [DayOfWeek], ScheduleTyp, Intervall
FROM inserted) FOR READ ONLY
OPEN crsEintraege
FETCH NEXT FROM crsEintraege INTO @ScheduleID, @Weekday, @ScheduleTyp, @Intervall
WHILE @@FETCH_STATUS = 0 BEGIN
	-- weekday can be NULL
	SELECT @Weekday = ISNULL(@Weekday, 0)
	-- check scheduletype
	IF @ScheduleTyp > 2 or @ScheduleTyp < 1 BEGIN
		DELETE Schedules WHERE ScheduleID = @ScheduleID
```

```
                RAISERROR (15600,-1,-1, 'The ScheduleTyp is not valid! It has to be 1 or 2')
        END
        -- check intervall
        IF @ScheduleTyp = 1 and @Intervall < 1 BEGIN
                DELETE Schedules WHERE ScheduleID = @ScheduleID
                RAISERROR (15600,-1,-1, 'The interval can not be lower than 1 if the
ScheduleTyp is 1 (Intervall)')
        END
        -- check dayofweek
        IF @Weekday > 7 or @Weekday < 1 and @ScheduleTyp = 2 BEGIN
                DELETE Schedules WHERE ScheduleID = @ScheduleID
                RAISERROR (15600,-1,-1, 'The Weekday is not a valid number between 1-7!')

        END
    /*
    -- next entry
    */
    FETCH NEXT FROM crsEintraege INTO @ScheduleID, @Weekday, @ScheduleTyp, @Intervall
    CONTINUE
END
CLOSE crsEintraege
DEALLOCATE crsEintraege
GO

ALTER TABLE [dbo].[Schedules] ENABLE TRIGGER [Schedules_HandelInserts]
GO
```

9.4 asy_CreateTables.sql

```
/*
-------------------------------------------------------------------------------------------
---------------
Autor:................
Date:.................2017.04.21
Description:..........This script is for the asyncron call of sps
Versions:.............2017.04.21 / RS Create the script
.......................xxxx.xx.xx / xx
-------------------------------------------------------------------------------------------
---------------
*/

/*
-- Result Table
*/
-- select * from AsyncExecQueue


DROP TABLE [AsyncExecResults]
GO
CREATE TABLE [AsyncExecResults] (
        [token] uniqueidentifier primary key
        , [submit_time] datetime not null
        , [start_time] datetime null
        , [finish_time] datetime null
        , [error_number]      int null
        , [error_message] nvarchar(2048) null);
GO
```

9.5 asy_CreateQueueAndService.sql

```
/*
-------------------------------------------------------------------------------------------
---------------
Autor:................
Date:.................2017.04.21
Description:..........This script is for the asyncron call of sps
```

```
Versions:.............2017.04.21 / RS Create the Asynch
.....................xxxx.xx.xx / xx
-------------------------------------------------------------------------------------------
---------------
*/
/*
-- Queue
*/

DROP SERVICE [AsyncExecService]
GO
DROP QUEUE [AsyncExecQueue]
GO


CREATE QUEUE [AsyncExecQueue]
GO

/*
-- Service
*/
CREATE SERVICE [AsyncExecService] ON QUEUE [AsyncExecQueue] ([DEFAULT]);
GO

/*
-- Alter the queue after you created the sps
*/
ALTER QUEUE [AsyncExecQueue]
    with activation (
    procedure_name = [usp_AsyncExecActivated]
    , max_queue_readers = 5
    , execute as owner
    , status = on);
GO
```

9.6 usp_AsyncExecInvoke.sql

```
/*
-------------------------------------------------------------------------------------------
---------------
Autor:................http://rusanu.com/2009/08/05/asynchronous-procedure-execution/
Date:.................2017.04.21
Description:..........This script is a stored procedure for the asyncron call of sps
Versions:.............2017.04.21 / RS copy pasted it
.....................xxxx.xx.xx / xx
-------------------------------------------------------------------------------------------
---------------
*/
DROP PROCEDURE [usp_AsyncExecInvoke]
GO

CREATE PROCEDURE [usp_AsyncExecInvoke]
    @procedureName sysname
    , @token uniqueidentifier output
as
begin
    declare @h uniqueidentifier
           , @xmlBody xml
        , @trancount int;
    set nocount on;

       --set @trancount = @@trancount;
 --   if @trancount = 0
 --       begin transaction
 --   else
```

```
        --save transaction usp_AsyncExecInvoke;
    begin try
        begin dialog conversation @h
            from service [AsyncExecService]
            to service N'AsyncExecService', 'current database'
            with encryption = off;
        select @token = [conversation_id]
            from sys.conversation_endpoints
            where [conversation_handle] = @h;
        select @xmlBody = (
            select @procedureName as [name]
            for xml path('procedure'), type);
              -- insert into SQLScheduler.dbo.result ([description])
values(convert(varchar(8000),@xmlBody));
        send on conversation @h (@xmlBody);
              /*
              -- Muss kontrolliert werden wann auf einem anderen Server
              */
        insert into SQLScheduler.dbo.AsyncExecResults
            ([token], [submit_time])
            values
            (@token, getdate());
    --if @trancount = 0
    --    commit;
    end try
    begin catch
        declare @error int
            , @message nvarchar(2048)
            , @xactState smallint;
        select @error = ERROR_NUMBER()
            , @message = ERROR_MESSAGE()
            , @xactState = XACT_STATE();
        --if @xactState = -1
        --    rollback;
        --if @xactState = 1 and @trancount = 0
        --    rollback
        --if @xactState = 1 and @trancount > 0
        --    rollback transaction usp_my_procedure_name;

        raiserror(N'Error: %i, %s', 16, 1, @error, @message);
    end catch
end
GO
```

9.7 usp_AsyncExecActivated.sql

```
/*
------------------------------------------------------------------------------------------
---------------
Autor:.................http://rusanu.com/2009/08/05/asynchronous-procedure-execution/
Date:..................2017.04.21
Description:..........This script is needed for the asyncron call of sps
Versions:.............2017.04.21 / RS copy pasted it
.....................xxxx.xx.xx / xx
------------------------------------------------------------------------------------------
---------------
*/
drop procedure usp_AsyncExecActivated
go

create procedure usp_AsyncExecActivated
as
begin
    set nocount on;
    declare @h uniqueidentifier
        , @messageTypeName sysname
```

```
        , @messageBody varbinary(max)
        , @xmlBody xml
        , @procedureName sysname
        , @startTime datetime
        , @finishTime datetime
        , @execErrorNumber int
        , @execErrorMessage nvarchar(2048)
        , @xactState smallint
        , @token uniqueidentifier;

    -- begin transaction;
    begin try;
              --insert into result ([description]) values ('recieving');
        receive top(1)
            @h = [conversation_handle]
            , @messageTypeName = [message_type_name]
            , @messageBody = [message_body]
            from [AsyncExecQueue];
                    -- insert into result ([description]) values (@messageBody);
        if (@h is not null)
        begin
            if (@messageTypeName = N'DEFAULT')
            begin
                            -- insert into result ([description]) values ('async');
                -- The DEFAULT message type is a procedure invocation.
                -- Extract the name of the procedure from the message body.
                --
                select @xmlBody = CAST(@messageBody as xml);
                select @procedureName = @xmlBody.value(
                    '(//procedure/name)[1]'
                    , 'sysname');

                -- save transaction usp_AsyncExec_procedure;
                select @startTime = GETDATE();
                begin try
                                    -- insert into result ([description]) values
(@procedureName);
                                    DECLARE @cmd varchar(8000);
                                    SELECT @cmd = 'exec ' + @procedureName;

                                    exec (@cmd)
                    -- exec @procedureName;
                                    -- insert into result ([description]) values
(@procedureName + 'finish');
                end try
                begin catch
                -- This catch block tries to deal with failures of the procedure execution
                -- If possible it rolls back to the savepoint created earlier, allowing
                -- the activated procedure to continue. If the executed procedure
                -- raises an error with severity 16 or higher, it will doom the transaction
                -- and thus rollback the RECEIVE. Such case will be a poison message,
                -- resulting in the queue disabling.
                --
                            -- insert into SQLScheduler.dbo.result ([description]) values
('Catch');
                select @execErrorNumber = ERROR_NUMBER(),
                    @execErrorMessage = ERROR_MESSAGE()
                --    @xactState = XACT_STATE();
                --if (@xactState = -1)
                --begin
                --    -- rollback;
                --    raiserror(N'Unrecoverable error in procedure %s: %i: %s', 16, 10,
                --        @procedureName, @execErrorNumber, @execErrorMessage);
                --end
                --else if (@xactState = 1)
                --begin
```

```
                --    rollback transaction usp_AsyncExec_procedure;
                --end
                end catch

                select @finishTime = GETDATE();
                select @token = [conversation_id]
                    from sys.conversation_endpoints
                    where [conversation_handle] = @h;
                if (@token is null)
                begin
                    raiserror(N'Internal consistency error: conversation not found', 16,
20);
                end
                update SQLScheduler.dbo.AsyncExecResults set
                    [start_time] = @starttime
                    , [finish_time] = @finishTime
                    , [error_number] = @execErrorNumber
                    , [error_message] = @execErrorMessage
                    where [token] = @token;
                if (0 = @@ROWCOUNT)
                begin
                    raiserror(N'Internal consistency error: token not found', 16, 30);
                end
                end conversation @h;
            end
            else if (@messageTypeName =
N'http://schemas.microsoft.com/SQL/ServiceBroker/EndDialog')
            begin
                end conversation @h;
            end
            else if (@messageTypeName =
N'http://schemas.microsoft.com/SQL/ServiceBroker/Error')
            begin
                declare @errorNumber int
                    , @errorMessage nvarchar(4000);
                select @xmlBody = CAST(@messageBody as xml);
                with xmlnamespaces (DEFAULT
N'http://schemas.microsoft.com/SQL/ServiceBroker/Error')
                select @errorNumber = @xmlBody.value ('(/Error/Code)[1]', 'INT'),
                    @errorMessage = @xmlBody.value ('(/Error/Description)[1]',
'NVARCHAR(4000)');
                -- Update the request with the received error
                select @token = [conversation_id]
                    from sys.conversation_endpoints
                    where [conversation_handle] = @h;
                                /*
                                -- Muss bei anderer DB geändert werden
                                */
                update SQLScheduler.dbo.AsyncExecResults set
                    [error_number] = @errorNumber
                    , [error_message] = @errorMessage
                    where [token] = @token;
                end conversation @h;
             end
           else
           begin
                raiserror(N'Received unexpected message type: %s', 16, 50,
@messageTypeName);
            end
        end
        -- commit;
    end try
    begin catch
        declare @error int
            , @message nvarchar(2048);
        select @error = ERROR_NUMBER()
```

```
            , @message = ERROR_MESSAGE()
        --    , @xactState = XACT_STATE();
        --if (@xactState <> 0)
        --begin
        --    rollback;
        --end;
        raiserror(N'Error: %i, %s', 1, 60,  @error, @message) with log;
    end catch
end
go
```

9.8 sp_CheckToRunJobs.sql

```
/*
-------------------------------------------------------------------------------------------------
---------------
Autor:................Romano Sabbatella
Date:.................2017.04.19
Description:..........This script checks if there's a job which must be scheduled.
......................If there's one it calls another procedure which handels the call of
the job
Parameter:............None
Return parameter:.....None
Version:..............2017.04.19 / RS Creat the base structure of the script
......................2017.04.20 / RS Created the script with all functionalities except
the parallel call of the sp
......................2017.04.26 / RS Had some issues with the parameters as string. I had
to add ' + '
......................xxxx.xx.xx / xx
-------------------------------------------------------------------------------------------------
---------------
*/
-- exec sp_CheckToRunJobs
/*
-- Drop the procedure
*/
DROP PROCEDURE [dbo].[sp_CheckToRunJobs]
GO

/*
-- Create the procedure
*/
CREATE PROCEDURE [dbo].[sp_CheckToRunJobs]
AS

/*
-- Declarations
*/
DECLARE @JobID int
DECLARE @Now DateTime
DECLARE @ErrorMsg varchar(255)
DECLARE @Database varchar(30)
DECLARE @ScheduleID int
DECLARE @OwnerID int
DECLARE @SpName varchar(50)
DECLARE @Shema varchar(50)
DECLARE @SpPath varchar(100)
DECLARE @RunDateTime datetime
DECLARE @ErrorDateTime datetime
DECLARE @JobName varchar(255)
DECLARE @StatusID int
DECLARE @DependsFromID int
DECLARE @PrevStatusID int
DECLARE @ScheduleStatus varchar(30)
DECLARE @Waiting bit
DECLARE @HistoryID int
```

```
DECLARE @ExecString varchar(255)
DECLARE @IntervallType int
DECLARE @Intervall int
DECLARE @NextRunDateTime datetime
DECLARE @Hour int
DECLARE @Minutes int
DECLARE @Daytime varchar(10)
/*
-- Initialisation
*/
SELECT @ScheduleID = 0
SELECT @Now = GETDATE()
SELECT @ErrorMsg = ''
SELECT @OwnerID = 0
/*
-- Declare cursor
*/
DECLARE crsJobs CURSOR LOCAL FOR
	(SELECT
		JobID, ScheduleID, SchedulestatusID, ScheduleTyp
	FROM
		Schedules
	WHERE
		RunDateTime < @Now
		and (SELECT Status FROM Schedulestatus WHERE Schedules.SchedulestatusID =
Schedulestatus.SchedulestatusID) <> 'Disabled'
		and ISNULL(ExpireDateTime,GETDATE()+1) > @Now
		and StartDateTime < @Now
	) ORDER BY RunDateTime FOR READ ONLY
-- open cursor
OPEN crsJobs
-- fetch the first row
FETCH NEXT FROM crsJobs INTO @JobID, @ScheduleID, @ScheduleStatus, @IntervallType
-- get thru all the jobs
WHILE @@FETCH_STATUS = 0 BEGIN
	/*
	-- Reset some values
	*/
	SELECT @Intervall = 0
	SELECT @JobName = ''
	SELECT @RunDateTime = GETDATE()
	SELECT @ErrorMsg = ''
	SELECT @Database = ''
	SELECT @SpName = ''
	SELECT @Shema = ''
	SELECT @SpPath = ''
	SELECT @ExecString = ''
	SELECT @OwnerID = 0
	SELECT @StatusID = 0
	SELECT @HistoryID = 0
	SELECT @DependsFromID = 0
	SELECT @Waiting = 0
	SELECT @Hour = 0
	SELECT @Minutes = 0
	SELECT @Daytime = ''
	/*
	-- Check if the interval is day of the week
	*/
	IF @IntervallType = 2 BEGIN
		IF (SELECT [DayOfWeek] FROM Schedules WHERE ScheduleID = @ScheduleID) <>
DATEPART(dw, GETDATE()) BEGIN
			SELECT @Waiting = 1
		END
		ELSE BEGIN
			-- get the time string
```

```
                        SELECT @Daytime = DayTime FROM Schedules WHERE ScheduleID =
@ScheduleID
                        IF ISNULL(@Daytime, '') = '' BEGIN
                                SELECT @ErrorMsg = @ErrorMsg + 'Die Tageszeit ist nicht
eingetragen. '
                        END
                        ELSE BEGIN
                                -- get the hour and minute from the time string
                                SELECT @Hour = convert(int,LEFT(@Daytime, 2))
                                SELECT @Minutes = convert(int,Right(@Daytime, 2))
                                -- check if the hour it should run is already here
                                IF @Hour > DATEPART(hh,@Now) BEGIN
                                        SELECT @Waiting = 1
                                END
                                -- check if the minute it should run is already here
                                IF @Minutes > DATEPART(n,@Now) BEGIN
                                        SELECT @Waiting = 1
                                END
                        END
                END
        END
        /*
        -- Check if the job infront is finished
        */
        SELECT @DependsFromID = PrevScheduleID FROM Schedules WHERE ScheduleID = @ScheduleID
        IF ISNULL(@DependsFromID,0) > 0 BEGIN
                SELECT @PrevStatusID = SchedulestatusID FROM Schedules WHERE ScheduleID =
@DependsFromID
                IF (SELECT Status FROM Schedulestatus WHERE SchedulestatusID = @PrevStatusID)
= 'Running' BEGIN
                        SELECT @Waiting = 1
                END
        END
        /*
        -- Just run the job if there's no previous depending job or the job allredy finished
        */
        IF @Waiting = 0 BEGIN
                /*
                -- Write an entry into the history
                */
                --we need a tmp table to save the output
                CREATE TABLE #TmpHistory(
                        HistroyID int
                )
                --insert the first details into history
                INSERT INTO Jobhistory
                        (JobID, ScheduleID, ErrorYN, StartDateTime)
                        OUTPUT inserted.JobhistoryID INTO #TmpHistory
                VALUES
                        (@JobID, @ScheduleID, 0, @RunDateTime)
                -- get the historyid
                SELECT @HistoryID = (SELECT TOP 1 HistroyID FROM #TmpHistory)
                -- drop the table
                DROP TABLE #TmpHistory
                /*
                -- Check if all needed informations are there
                */
                -- Database
                SELECT
                        @Database = ISNULL([Databases].[Name],'')
                FROM
                        [Databases]
                        JOIN Schedules ON [Databases].DatabaseID = Schedules.DatabaseID
                WHERE
                        Schedules.ScheduleID = @ScheduleID
                -- check if the name has at least 1 char
```

```
		IF NOT EXISTS (SELECT * FROM sys.databases WHERE name = @Database) BEGIN
			SELECT @ErrorDateTime = GETDATE()
			SELECT @ErrorMsg = @ErrorMsg + 'Die gewünschte Datenbank ist nicht
vorhanden. '
		END
		--OwnerID
		SELECT @OwnerID = OwnerID FROM Schedules WHERE ScheduleID = @ScheduleID
		-- SP
		SELECT @SpName = ISNULL(SpName,'') FROM Schedules WHERE ScheduleID =
@ScheduleID
		-- check if the spname is at least 1 char
		IF LEN(@SpName) = 0 BEGIN
			SELECT @ErrorDateTime = GETDATE()
			SELECT @ErrorMsg = @ErrorMsg + 'Der SP-Namen ist nicht eingetragen. '
		END
		-- shema
		SELECT @Shema = ISNULL(Shema,'') FROM Owners WHERE OwnerID = @OwnerID
		-- check if the shema name is at least 1 char
		IF LEN(@Shema) = 0 BEGIN
			SELECT @ErrorDateTime = GETDATE()
			SELECT @ErrorMsg = @ErrorMsg + 'Der Shema-Namen ist nicht eingetragen.
'
		END
		-- concat the string
		SELECT @SpPath = @Database + '.' + @Shema + '.' + @SpName
		-- check if the sp exists
		IF OBJECT_ID(@SpPath) IS NOT NULL BEGIN
			/*
			-- update the status to running
			*/
			UPDATE Schedules SET SchedulestatusID = (SELECT SchedulestatusID FROM
Schedulestatus WHERE [Status] = 'Running') WHERE ScheduleID = @ScheduleID
			-- get the exec string
			SELECT @ExecString = 'sp_RunJob ' + convert(varchar(255),@JobID) + ',
' + convert(varchar(255),@ScheduleID) + ', ' + convert(varchar(255),@HistoryID) + ', ' +
'''' + @SpPath + ''''
			/*
			-- Execute the next sp
			*/
			DECLARE @token uniqueidentifier
			BEGIN TRY
				EXEC usp_AsyncExecInvoke @ExecString, @token output
			END TRY
			BEGIN CATCH
				-- Build error msg
				SELECT @ErrorDateTime = GETDATE()
				SELECT @ErrorMsg = @ErrorMsg + 'Die SP wurde nicht in die Queue
gestellt. Der ExecString ist: ' + @ExecString + '. '
			END CATCH
		END
		ELSE BEGIN
			-- build error msg
			SELECT @ErrorDateTime = GETDATE()
			SELECT @ErrorMsg = @ErrorMsg + 'Die gewünschte Sp existiert unter dem
Pfad ' + @SpPath + ' nicht. '
		END
		/*
		-- write the error message into the protocol
		*/
		IF LEN(@ErrorMsg) > 0 BEGIN
			-- get the job name for error report
			SELECT @JobName = [Name] FROM Jobs WHERE JobID = @JobID
			/*
			-- write an entry into the protocol table and save it
			*/
			INSERT INTO Exceptionlog
```

```
                        (JobID, RunDateTime, ErrorDateTime, ErrorMessage, [Name],
SpName, JobName, JobhistoryID)
                VALUES
                        (@JobID, @RunDateTime, @ErrorDateTime, @ErrorMsg, @Database,
@SpName, @JobName, @HistoryID)
                /*
                -- write the entry into the history table with error flag
                */
                Update
                        Jobhistory
                SET
                        ErrorYN = 1, EndDateTime = @ErrorDateTime
                WHERE
                        JobhistoryID = @HistoryID
        END
        /*
        -- set the new interval
        */
        SELECT @IntervallType = ScheduleTyp FROM Schedules WHERE ScheduleID =
@ScheduleID
        -- if the interval is in minutes
        IF @IntervallType = 1 BEGIN
                -- get the interval length
                SELECT @Intervall = Intervall FROM Schedules WHERE ScheduleID =
@ScheduleID
                SELECT @NextRunDateTime = DATEADD(mi,@Intervall, (SELECT RunDateTime
FROM Schedules WHERE ScheduleID = @ScheduleID))
        END
        IF @IntervallType = 2 BEGIN
                -- set the rundate in the future and out of today because its weekly
                SELECT @NextRunDateTime = DATEADD(d, 1, @Now)
        END
        -- set the new run date
        UPDATE Schedules SET RunDateTime = @NextRunDateTime WHERE ScheduleID =
@ScheduleID
    END
    -- next fetch
    FETCH NEXT FROM crsJobs INTO @JobID, @ScheduleID, @ScheduleStatus, @IntervallType
END
--close the cursor
CLOSE crsJobs
-- get rid of the memory region
DEALLOCATE crsJobs

GO
```

9.9 usp_AsyncExecActivated.sql

```
/*
-------------------------------------------------------------------------------------------
---------------
Autor:................Romano Sabbatella
Date:.................2017.04.20
Description:..........This script runs the job with help of the SQL Broker
.....................
Parameter:............@ScheduleID   the ID of the schedule which has to run
.....................@HistoryID    the ID of the History entry
.....................@SpPath              Contains the path of the sp which it has to run
Return parameter:.....None
Version:..............2017.04.20 / RS Creat the base structure of the script
.....................2017.04.21 / RS Corrected some mistakes
.....................2017.04.24 / RS surrounded inserts and updates with try / catch
because of the queue so it runs,
.....................                 otherwise the queue will crash and stop working
.....................xxxx.xx.xx / xx xxxxxx
```

```
-------------------------------------------------------------------------------------------
---------------
*/
/*
-- Drop the procedure
*/
DROP PROCEDURE [dbo].[sp_RunJob]
GO

/*
-- Create the procedure
*/
CREATE PROCEDURE [dbo].[sp_RunJob]
	(
	@i_JobID int,
	@i_ScheduleID int,
	@i_HitoryID int,
	@i_SpPath varchar(100)
	)
AS
/*
-- Declarations
*/
DECLARE @JobID int
DECLARE @JobhistoryID int
DECLARE @ScheduleID int
DECLARE @SpPath varchar(100)
DECLARE @Error bit
DECLARE @ErrorMsg varchar(max)
DECLARE @Description varchar(100)
DECLARE @ExceptionlogID int
DECLARE @JobName varchar(255)
DECLARE @RunDateTime datetime
DECLARE @DateTime datetime
DECLARE @DatabaseName varchar(30)
DECLARE @SpName varchar(50)
/*
-- Initialisation
*/
--to values i got
SELECT @JobID = @i_JobID
SELECT @JobhistoryID = @i_HitoryID
SELECT @ScheduleID = @i_ScheduleID
SELECT @SpPath = @i_SpPath
-- new values
SELECT @DatabaseName = ''
SELECT @SpName = ''
SELECT @ErrorMsg = ''
SELECT @Description = ''
SELECT @Error = 0
SELECT @JobName = ''
/*
-- execute the sp
*/
BEGIN TRY
	exec @SpPath
	SELECT @DateTime = GETDATE()
END TRY
/*
-- Error handling if something went wrong
*/
BEGIN CATCH
	BEGIN TRY
	SELECT @ErrorMsg = ERROR_MESSAGE()
	SELECT @DateTime = GETDATE()
	-- get the job name
```

```
        SELECT @JobName = [Name] FROM SQLScheduler.dbo.Jobs WHERE JobID = @JobID
        -- get the databasename
        SELECT
                @DatabaseName = [Databases].[Name]
        FROM
                SQLScheduler.dbo.Schedules
                JOIN [Databases] ON SQLScheduler.dbo.Schedules.DatabaseID =
[Databases].DatabaseID
        -- get the runtime
        SELECT @RunDateTime = StartDateTime FROM SQLScheduler.dbo.Jobhistory WHERE
JobhistoryID = @JobhistoryID
        -- get the sp name
        SELECT @SpName = SpName FROM SQLScheduler.dbo.Schedules WHERE ScheduleID =
@ScheduleID
        -- get the jobname
        SELECT @JobName = Name FROM SQLScheduler.dbo.Jobs WHERE JobID = @JobID
        /*
        -- Create tmptable for output
        */
        -- insert error into protocoll
        INSERT INTO SQLScheduler.dbo.Exceptionlog
                (JobID, JobName, RunDateTime, ErrorDateTime, ErrorMessage, [Name], SpName,
JobhistoryID)
        VALUES
                (@JobID, @JobName, @RunDateTime, @DateTime, @ErrorMsg, @DatabaseName,
@SpName, @JobhistoryID)
        -- set Error
        SELECT @Error = 1
        END TRY
        BEGIN CATCH
                -- if everthing goes wrong nothing happens here
                -- otherwise the queue will crash
        END CATCH
END CATCH
/*
-- Update the History
*/
BEGIN TRY
        UPDATE
                SQLScheduler.dbo.Jobhistory
        SET
                ErrorYN = @Error, [Description] = @Description, EndDateTime = @DateTime
        WHERE
                JobhistoryID = @JobhistoryID
END TRY
BEGIN CATCH
        -- if everthing goes wrong nothing happens here
        -- otherwise the queue will crash
END CATCH
/*
-- Update the schedule status
*/
BEGIN TRY
        UPDATE SQLScheduler.dbo.Schedules SET SchedulestatusID = (SELECT SchedulestatusID
FROM Schedulestatus WHERE [Status] = 'Ready') WHERE ScheduleID = @ScheduleID
END TRY
BEGIN CATCH
        -- if everthing goes wrong nothing happens here
        -- otherwise the queue will crash
END CATCH

GO
```

9.10 CreateTestDatabase.sql

```
/*
```

```
------------------------------------------------------------------------------------------------
---------------
Autor:.................Romano Sabbatella
Date:..................2017.04.20
Description:..........This script creates test databases
......................
Parameter:............None
Return parameter:.....None
......................2017.04.20 / RS create the script
......................2017.04.25 / RS Added tables into the databases
......................xxxx.xx.xx / xx
------------------------------------------------------------------------------------------------
---------------
*/
/*
-- master
*/
USE master
/*
-- TestDB1
*/
-- drop database
IF EXISTS (SELECT * FROM sys.databases WHERE name = 'IpaTestDB') BEGIN
	DROP DATABASE IpaTestDB
END
GO

-- create database
CREATE DATABASE IpaTestDB
GO
ALTER DATABASE IpaTestDB SET TRUSTWORTHY ON
GO
```

9.11 CreateTestScripts.sql

```
/*
------------------------------------------------------------------------------------------------
---------------
Autor:.................Romano Sabbatella
Date:..................2017.04.25
Description:..........This script generates stored procedures on the IpaTestDB1
......................
Parameter:............None
Return parameter:.....None
......................2017.04.25 / RS create the script
......................xxxx.xx.xx / xx
------------------------------------------------------------------------------------------------
---------------
*/
USE IpaTestDB
-- table
IF EXISTS (SELECT * FROM sys.objects WHERE object_id = OBJECT_ID(N'TestResult') AND TYPE IN
(N'U')) BEGIN
	-- Drop table
	DROP TABLE TestResult
END
--create table
CREATE TABLE TestResult(
	Id int identity(1,1) primary key not null,
	success bit,
	[description] varchar(300),
	CreationDate datetime,
	FinishDate datetime
)
GO
-- drop procedure
```

```
DROP PROCEDURE sp_CheckSuccess
GO
-- create p_CheckSuccess
CREATE PROCEDURE sp_CheckSuccess
AS

	INSERT INTO TestResult (success, [description],CreationDate) VALUES (1, 'Without
delay', GETDATE())

GO
-- drop procedure
DROP PROCEDURE sp_CheckSuccessDelay
GO
-- create p_CheckSuccess
CREATE PROCEDURE sp_CheckSuccessDelay
AS
	CREATE TABLE #Tmp(
		ID int
	)

	INSERT INTO
		TestResult (success, [description],CreationDate)
	OUTPUT inserted.Id INTO #Tmp
	VALUES
		(1, 'With delay', GETDATE())


	WAITFOR DELAY '00:00:02'
	UPDATE TestResult SET
		[description] = 'with delay', FinishDate = GETDATE()
	WHERE
		Id = (SELECT ID FROM #Tmp)

GO

-- drop procedure
DROP PROCEDURE sp_CheckSuccessParam
GO
-- create p_CheckSuccess
CREATE PROCEDURE sp_CheckSuccessParam(
	@i_desc varchar(255)
)
AS

INSERT INTO TestResult (success, [description]) VALUES (1, 'With param: ' + @i_desc)

GO
```

9.12 CreateTestScripts.sql

```
/*
-----------------------------------------------------------------------------------------
---------------
Autor:.................Romano Sabbatella
Date:..................2017.04.20
Description:..........This script creates test data into the tables
.......................!!Important!! Run the insert statement manually because the IDs of
the foreign keys are maybe different !!!
Parameter:............None
Return parameter:.....None
.......................2017.04.20 / RS Create the script
.......................xxxx.xx.xx / xx
-----------------------------------------------------------------------------------------
---------------
*/
-- exec sp_CheckToRunJobs
```

```
--
/*
-- use the right database
*/
USE SQLScheduler
/*
-- Table databases
*/
INSERT INTO [Databases]
       (Name, Description)
VALUES
       ('IpaTestDB', 'TestDB')

/*
-- Table Owner
*/
INSERT INTO Owners
       (UserName, Password, Shema, Email, Description)
VALUES
       ('sa', 'password1', 'dbo', 'support@boreas.ch', 'testacc')

/*
-- table jobs
*/

INSERT INTO Jobs
       (Name, Description, CreationDate)
VALUES
       ('Datenpflege', 'Dieser Job pflegt Daten', GETDATE()),
       ('Testfälle', 'Dieser Job dividiert durch 0', GETDATE()-2),
       ('Warten auf schön Wetter', 'Dieser Job dauert', GETDATE()-1)

/*
-- table schedulestatus
*/
INSERT INTO Schedulestatus
       (Status, Description)
VALUES
       ('Ready', 'The Schedule is ready to go'),
       ('Running', 'The Schedule is currently bussy'),
       ('Disabled', 'The Schedule is disabled')


/*
-- table Schedules
!! have a look at the IDs and run it maually because thery're maybe not the same !!

select * from Owners
select * from Jobs
select * from Schedulestatus
select * from Databases

select * from Schedules


*/
/*
-- disable all in table
-- UPDATE Schedules SET ScheduleStatusID = 3


INSERT INTO Schedules
       (OwnerID, JobID, SchedulestatusID, DatabaseID, Intervall, RunDateTime, SpName,
ScheduleType, StartDateTime, [Description], PrevScheduleID, ExpireDateTime, [DayOfWeek],
DayTime)
VALUES
```

```
	(1, 2, 1, 1, 1, GETDATE(), 'sp_CheckSuccess', 1, GETDATE()-1, 'Der Job sollte
laufen', 0, NULL, NULL, NULL),
	(1, 1, 1, 1, 2, GETDATE(), 'sp_CheckSuccessDelay', 1, GETDATE()-1, 'Der Job sollte
laufen', 0, NULL, NULL, NULL),
	(1, 1, 1, 1, 2, GETDATE(), 'sp_CheckSuccess', 1, GETDATE()+100, 'Der Job sollte
nicht laufen', 0, NULL, NULL, NULL),
	(1, 1, 1, 1, 60, GETDATE(), 'sp_CheckSuccess', 1, GETDATE(), 'Der Job sollte
laufen', 0, NULL, NULL, NULL),
	(1, 1, 1, 1, 0, GETDATE(), 'sp_CheckSuccess', 2, GETDATE()-1, 'Der Job sollte
laufen', 0, NULL, 2, '1100'),
	(1, 1, 1, 1, 0, GETDATE(), 'sp_CheckSuccess', 2, GETDATE()+1, 'Der Job sollte nicht
laufen', 0, NULL, NULL, '1200')

*/
```

9.13 Case1.sql

```
/*
-------------------------------------------------------------------------------------------
---------------
Autor:.................Romano Sabbatella
Date:..................2017.04.27
Description:..........This script is for test case 1
Parameter:............None
Return parameter:.....None
.....................2017.04.27 / RS Create the script
.....................xxxx.xx.xx / xx
-------------------------------------------------------------------------------------------
---------------
*/
-- Insert the data
INSERT INTO Schedules
	(OwnerID, JobID, SchedulestatusID, DatabaseID, Intervall, RunDateTime, SpName,
ScheduleTyp, StartDateTime, [Description], PrevScheduleID, ExpireDateTime, [DayOfWeek],
DayTime)
VALUES
	(1, 2, 1, 1, 1, GETDATE()-1, 'sp_CheckSuccess', 1, GETDATE()-1, 'Der Job sollte
laufen', NULL, NULL, NULL, NULL),
	(1, 1, 1, 1, 2, GETDATE()-1, 'sp_CheckSuccess', 1, GETDATE()-1, 'Der Job sollte
laufen', NULL, NULL, NULL, NULL)

-- create temp table
CREATE TABLE #Tmp(
	ScheduleID int,
	Runtime datetime,
	Intervall int
)
-- execute the check to run jobs three times
exec sp_CheckToRunJobs
WAITFOR DELAY '00:00:02'
-- select into tmp db
INSERT INTO #Tmp (ScheduleID, Runtime, Intervall) (SELECT ScheduleID, RunDateTime,
Intervall FROM Schedules)

--exec
exec sp_CheckToRunJobs
WAITFOR DELAY '00:00:02'
-- select into tmp db
INSERT INTO #Tmp (ScheduleID, Runtime, Intervall) (SELECT ScheduleID, RunDateTime,
Intervall FROM Schedules)

-- exec
exec sp_CheckToRunJobs
INSERT INTO #Tmp (ScheduleID, Runtime, Intervall) (SELECT ScheduleID, RunDateTime,
Intervall FROM Schedules)
```

```
-- get the data
SELECT * FROM #Tmp ORDER BY ScheduleID, Runtime ASC

-- drop table
DROP TABLE #Tmp

DELETE Schedules
```

9.14 Case2.sql

```
/*
------------------------------------------------------------------------------------------------
---------------
Autor:.............Romano Sabbatella
Date:..............2017.04.27
Description:.......This script is for test case 2
Parameter:.........None
Return parameter:..None
..................2017.04.27 / RS Create the script
..................xxxx.xx.xx / xx
------------------------------------------------------------------------------------------------
---------------
*/
-- make sure the table is empty
DELETE Jobhistory
DELETE Schedules
-- Insert the data
INSERT INTO Schedules
       (OwnerID, JobID, SchedulestatusID, DatabaseID, Intervall, RunDateTime, SpName,
ScheduleTyp, StartDateTime, [Description], PrevScheduleID, ExpireDateTime, [DayOfWeek],
DayTime)
VALUES
       (1, 2, 1, 1, 0, GETDATE()-1, 'sp_CheckSuccess', 2, GETDATE()-1, 'Der Job sollte
laufen', NULL, NULL, DATEPART(dw,GETDATE()), '0100'),
       (1, 1, 1, 1, 0, GETDATE()-1, 'sp_CheckSuccess', 2, GETDATE()-1, 'Der Job sollte
laufen', NULL, NULL, DATEPART(dw,GETDATE()), '2359')
-- execute the check to run jobs
exec sp_CheckToRunJobs
WAITFOR DELAY '00:00:02'
-- check if the runtime is set to tomorrow
SELECT * FROM Schedules WHERE RunDateTime > GETDATE()
-- check if there's an entry in the Jobhistory table
SELECT * FROM Jobhistory
-- delete the table entrys
DELETE Jobhistory
DELETE Schedules
```

9.15 Case3.sql

```
/*
------------------------------------------------------------------------------------------------
---------------
Autor:.............Romano Sabbatella
Date:..............2017.04.27
Description:.......This script is for test case 3
Parameter:.........None
Return parameter:..None
..................2017.04.27 / RS Create the script
..................xxxx.xx.xx / xx
------------------------------------------------------------------------------------------------
---------------
*/
-- make sure the table is empty
DELETE Exceptionlog
DELETE Jobhistory
```

```
DELETE Schedules
-- Insert the data
INSERT INTO Schedules
       (OwnerID, JobID, SchedulestatusID, DatabaseID, Intervall, RunDateTime, SpName,
ScheduleTyp, StartDateTime, [Description], PrevScheduleID, ExpireDateTime, [DayOfWeek],
DayTime)
VALUES
       (1, 2, 1, 1, 1, GETDATE()-1, 'sp_CheckFail', 1, GETDATE()-1, 'Der Job sollte
laufen', NULL, NULL, NULL, NULL)
-- execute the check to run jobs
exec sp_CheckToRunJobs
WAITFOR DELAY '00:00:02'
-- get the error
SELECT * FROM Exceptionlog
-- delete the entrys
DELETE Exceptionlog
DELETE Jobhistory
DELETE Schedules
```

9.16 Case4.sql

```
/*
------------------------------------------------------------------------------------------
---------------
Autor:.................Romano Sabbatella
Date:..................2017.04.27
Description:..........This script is for test case 4
Parameter:............None
Return parameter:.....None
.......................2017.04.27 / RS Create the script
.......................xxxx.xx.xx / xx
------------------------------------------------------------------------------------------
---------------
*/
-- make sure the table is empty
DELETE Exceptionlog
DELETE Jobhistory
DELETE Schedules
-- tmp table
-- Insert the data
INSERT INTO Schedules
       (OwnerID, JobID, SchedulestatusID, DatabaseID, Intervall, RunDateTime, SpName,
ScheduleTyp, StartDateTime, [Description], PrevScheduleID, ExpireDateTime, [DayOfWeek],
DayTime)
VALUES
       (1, 2, 1, 1, 60, GETDATE()-1, 'sp_CheckFail', 1, GETDATE()-1, 'Der Job sollte
laufen', NULL, NULL, NULL, NULL)
-- get the date from schedules
SELECT RunDateTime FROM Schedules
-- execute the check to run jobs
exec sp_CheckToRunJobs
WAITFOR DELAY '00:00:02'
-- get the new runtime
SELECT RunDateTime FROM Schedules
-- get the error
SELECT * FROM Exceptionlog
-- delete the entrys
DELETE Exceptionlog
DELETE Jobhistory
DELETE Schedules
```

9.17 Case5.sql

```
/*
------------------------------------------------------------------------------------------
---------------
```

```
Autor:.................Romano Sabbatella
Date:..................2017.04.27
Description:..........This script is for test case 5
Parameter:............None
Return parameter:.....None
.......................2017.04.27 / RS Create the script
.......................xxxx.xx.xx / xx
-------------------------------------------------------------------------------------------------
----------------
*/
-- make sure the table is empty
DELETE Exceptionlog
DELETE Jobhistory
DELETE Schedules
-- Insert the data
INSERT INTO Schedules
       (OwnerID, JobID, SchedulestatusID, DatabaseID, Intervall, RunDateTime, SpName,
ScheduleTyp, StartDateTime, [Description], PrevScheduleID, ExpireDateTime, [DayOfWeek],
DayTime)
VALUES
       (1, 2, 1, 1, 60, GETDATE()-1, 'sp_CheckFail', 1, GETDATE()-1, 'Der Job sollte
laufen', NULL, NULL, NULL, NULL)
-- get the date from schedules
SELECT * FROM Schedules
-- execute the check to run jobs
exec sp_CheckToRunJobs
WAITFOR DELAY '00:00:02'
-- get the new runtime
SELECT * FROM Schedules
-- get the error
SELECT * FROM Exceptionlog
-- delete the entrys
DELETE Exceptionlog
DELETE Jobhistory
DELETE Schedules
```

9.18 Case6.sql

```
/*
-------------------------------------------------------------------------------------------------
----------------
Autor:.................Romano Sabbatella
Date:..................2017.04.27
Description:..........This script is for test case 5
Parameter:............None
Return parameter:.....None
.......................2017.04.27 / RS Create the script
.......................xxxx.xx.xx / xx
-------------------------------------------------------------------------------------------------
----------------
*/
-- make sure the table is empty
DELETE Exceptionlog
DELETE Jobhistory
DELETE Schedules
DELETE IpaTestDB.dbo.TestResult
-- Insert the data
INSERT INTO Schedules
       (OwnerID, JobID, SchedulestatusID, DatabaseID, Intervall, RunDateTime, SpName,
ScheduleTyp, StartDateTime, [Description], PrevScheduleID, ExpireDateTime, [DayOfWeek],
DayTime)
VALUES
       (1, 2, 1, 1, 1, GETDATE()-2, 'sp_CheckSuccessDelay', 1, GETDATE()-1, 'Der Job sollte
laufen', NULL, NULL, NULL, NULL),
       (1, 2, 1, 1, 1, GETDATE()-1, 'sp_CheckSuccess', 1, GETDATE()-1, 'Der Job sollte
laufen', NULL, NULL, NULL, NULL)
```

```
-- execute the check to run jobs
exec sp_CheckToRunJobs
WAITFOR DELAY '00:00:02'
-- get the result
SELECT * FROM IpaTestDB.dbo.TestResult
WAITFOR DELAY '00:00:30'
-- get the result
SELECT * FROM IpaTestDB.dbo.TestResult
-- delete the entrys
DELETE Exceptionlog
DELETE Jobhistory
DELETE Schedules
DELETE IpaTestDB.dbo.TestResult
```

9.19 Case7.sql

```
/*
------------------------------------------------------------------------------------------------
---------------
Autor:.................Romano Sabbatella
Date:..................2017.04.27
Description:..........This script is for test case 6
Parameter:............None
Return parameter:.....None
.....................2017.04.27 / RS Create the script
.....................xxxx.xx.xx / xx
------------------------------------------------------------------------------------------------
---------------
*/
-- make sure the table is empty
DELETE Exceptionlog
DELETE Schedules
DELETE Jobhistory
-- get a variable
DECLARE @ScheduleID int
-- get a temp table
CREATE TABLE #Tmp(
       id int
)
-- Insert the data
INSERT INTO Schedules
       (OwnerID, JobID, SchedulestatusID, DatabaseID, Intervall, RunDateTime, SpName,
ScheduleTyp, StartDateTime, [Description], PrevScheduleID, ExpireDateTime, [DayOfWeek],
DayTime)
OUTPUT
       inserted.ScheduleID INTO #Tmp
VALUES
       (1, 2, 1, 1, 1, GETDATE()-2, 'sp_CheckSuccessDelay', 1, GETDATE()-1, 'Der Job sollte
laufen', NULL, NULL, NULL, NULL)
--get the id
SELECT @ScheduleID = (SELECT TOP 1 ID FROM #Tmp)
-- drop the temp table
DROP TABLE #Tmp
-- Insert the data
INSERT INTO Schedules
       (OwnerID, JobID, SchedulestatusID, DatabaseID, Intervall, RunDateTime, SpName,
ScheduleTyp, StartDateTime, [Description], PrevScheduleID, ExpireDateTime, [DayOfWeek],
DayTime)
VALUES
       (1, 2, 1, 1, 1, GETDATE()-1, 'sp_CheckSuccess', 1, GETDATE()-1, 'Der Job sollte
laufen', @ScheduleID, NULL, NULL, NULL)
-- execute the check to run jobs
exec sp_CheckToRunJobs
WAITFOR DELAY '00:00:02'
SELECT * FROM Jobhistory
-- make sure the table is empty
```

```
DELETE Exceptionlog
DELETE Schedules
DELETE Jobhistory
```

9.20 Case8.sql

```
/*
------------------------------------------------------------------------------------------------
---------------
Autor:.............Romano Sabbatella
Date:..............2017.04.27
Description:.......This script is for test case 8
Parameter:.........None
Return parameter:..None
..................2017.04.27 / RS Create the script
..................xxxx.xx.xx / xx
------------------------------------------------------------------------------------------------
---------------
*/
-- make sure the table is empty
DELETE Schedules
-- Insert the data
INSERT INTO Schedules
       (OwnerID, JobID, SchedulestatusID, DatabaseID, Intervall, RunDateTime, SpName,
ScheduleTyp, StartDateTime, [Description], PrevScheduleID, ExpireDateTime, [DayOfWeek],
DayTime)
VALUES
       (1, 2, 1, 1, 1, GETDATE()-2, 'sp_CheckSuccessDelay', 2, GETDATE()-1, 'Der Job sollte
laufen', NULL, NULL, 0, NULL)
-- make sure the table is empty
DELETE Schedules
```

9.21 Case9.sql

```
/*
------------------------------------------------------------------------------------------------
---------------
Autor:.............Romano Sabbatella
Date:..............2017.04.27
Description:.......This script is for test case 9
Parameter:.........None
Return parameter:..None
..................2017.04.27 / RS Create the script
..................xxxx.xx.xx / xx
------------------------------------------------------------------------------------------------
---------------
*/
-- make sure the table is empty
DELETE Schedules
-- Insert the data
INSERT INTO Schedules
       (OwnerID, JobID, SchedulestatusID, DatabaseID, Intervall, RunDateTime, SpName,
ScheduleTyp, StartDateTime, [Description], PrevScheduleID, ExpireDateTime, [DayOfWeek],
DayTime)
VALUES
       (1, 2, 1, 1, 1, GETDATE()-2, 'sp_CheckSuccessDelay', 2, GETDATE()-1, 'Der Job sollte
laufen', NULL, NULL, 8, NULL)
-- make sure the table is empty
DELETE Schedules
```

9.22 Case10.sql

```
/*
------------------------------------------------------------------------------------------------
---------------
```

```
Autor:.............Romano Sabbatella
Date:..............2017.04.27
Description:.......This script is for test case 10
Parameter:.........None
Return parameter:....None
..................2017.04.27 / RS Create the script
..................xxxx.xx.xx / xx
-------------------------------------------------------------------------------------------
---------------
*/
-- make sure the table is empty
DELETE Schedules
-- Insert the data
INSERT INTO Schedules
       (OwnerID, JobID, SchedulestatusID, DatabaseID, Intervall, RunDateTime, SpName,
ScheduleTyp, StartDateTime, [Description], PrevScheduleID, ExpireDateTime, [DayOfWeek],
DayTime)
VALUES
       (1, 2, 1, 1, 0, GETDATE()-2, 'sp_CheckSuccessDelay', 1, GETDATE()-1, 'Der Job sollte
laufen', NULL, NULL, NULL, NULL)
-- make sure the table is empty
DELETE Schedules
```

9.23 Case11.sql

```
/*
-------------------------------------------------------------------------------------------
---------------
Autor:.............Romano Sabbatella
Date:..............2017.04.27
Description:.......This script is for test case 11
Parameter:.........None
Return parameter:....None
..................2017.04.27 / RS Create the script
..................xxxx.xx.xx / xx
-------------------------------------------------------------------------------------------
---------------
*/
-- make sure the table is empty
DELETE Schedules
-- Insert the data
INSERT INTO Schedules
       (OwnerID, JobID, SchedulestatusID, DatabaseID, Intervall, RunDateTime, SpName,
ScheduleTyp, StartDateTime, [Description], PrevScheduleID, ExpireDateTime, [DayOfWeek],
DayTime)
VALUES
       (1, 2, 1, 1, 0, GETDATE()-2, 'sp_CheckSuccessDelay', 3, GETDATE()-1, 'Der Job sollte
laufen', NULL, NULL, NULL, NULL)
-- make sure the table is empty
DELETE Schedules
```

9.24 Case12.sql

```
/*
-------------------------------------------------------------------------------------------
---------------
Autor:.............Romano Sabbatella
Date:..............2017.04.27
Description:.......This script is for test case 12
Parameter:.........None
Return parameter:....None
..................2017.04.27 / RS Create the script
..................xxxx.xx.xx / xx
-------------------------------------------------------------------------------------------
---------------
*/
```

```
-- make sure the table is empty
DELETE Schedules
-- Insert the data
INSERT INTO Schedules
       (OwnerID, JobID, SchedulestatusID, DatabaseID, Intervall, RunDateTime, SpName,
ScheduleTyp, StartDateTime, [Description], PrevScheduleID, ExpireDateTime, [DayOfWeek],
DayTime)
VALUES
       (1, 2, 1, 1, 0, GETDATE()-2, 'sp_CheckSuccessDelay', 0, GETDATE()-1, 'Der Job sollte
laufen', NULL, NULL, NULL, NULL)
-- make sure the table is empty
DELETE Schedules
```