WIRTSCHAFTSUNIVERSITÄT WIEN
Vienna University of Economics and Business

Bachelorarbeit

Deutscher Titel der Bachelorarbeit	Evaluierung des Datenbanksystems SAP HANA als Data Warehouse für Analysen und Reports
Englischer Titel der Bachelorarbeit	Evaluation of the Database System SAP HANA as a Data Warehouse for Analyzes and Reports
Verfasser/in Familienname, Vorname(n)	Burgstaller Andreas
Matrikelnummer	11720941
Studium	Bachelorstudium Wirtschafts- und Sozialwissenschaften
Beurteiler/in Titel, Vorname(n), Familienname	ao.Univ.Prof. Dr. Rony G. Flatscher

Hiermit versichere ich, dass

1. ich die vorliegende Bachelorarbeit selbständig und ohne Verwendung unerlaubter Hilfsmittel verfasst habe. Alle Inhalte, die direkt oder indirekt aus fremden Quellen entnommen sind, sind durch entsprechende Quellenangaben gekennzeichnet.

2. die vorliegende Arbeit bisher weder im In- noch im Ausland zur Beurteilung vorgelegt bzw. veröffentlicht worden ist.

3. diese Arbeit mit der beurteilten bzw. in elektronischer Form eingereichten Bachelorarbeit übereinstimmt.

4. (nur bei Gruppenarbeiten): die vorliegende Arbeit gemeinsam mit

   entstanden ist. Die Teilleistungen der einzelnen Personen sind kenntlich gemacht, ebenso wie jene Passagen, die gemeinsam erarbeitet wurden.

Datum

Unterschrift

Abstract

Das Sammeln, Verwalten und Auswerten von Informationen hat einen immer größer werdenden Stellenwert für Unternehmen jeglicher Art. Diese hohen Anforderungen bringen klassische Datenbanken, die für kurze und simple Transaktionen optimiert wurden, an ihre Grenzen. Infolgedessen wurde die Entwicklung sogenannter Data Warehouses forciert. Dieser neue Ansatz von Datenbanksystemen ermöglicht es, komplexe Transaktionen effizienter abzuarbeiten und bessere und exaktere Auswertungen zu erstellen. Die vorliegende Bachelorarbeit gibt zunächst einen Überblick über die technologischen Eigenschaften und Funktionalitäten von Data Warehouses für verschiedenste Anwendungsszenarien. Anschließend wird erläutert, wie ein Data Warehouse aufgebaut ist, wie Datensätze implementiert werden und wie daraus Analysen und Reports erstellt werden. Dabei wird das Hauptaugenmerk auf Softwareressourcen des Anbieters SAP gelegt, dessen Schwerpunkt in der Entwicklung von ERP Systemen liegt. Besonders das Datenbank System SAP HANA, mithilfe dessen ein Data Warehouse entwickelt werden soll, steht im Mittelpunkt dieser Arbeit. Als Forschungsmethode der Bachelorarbeit wird ein Proof of Concept eingesetzt. Ziel ist ein lauffähiger Prototyp für ein Data Warehouse, das von verschiedenen Quellen Daten beziehen kann. Zusätzlich wird für die Beschaffung von Beispieldaten ein Object Rexx Web Scraper entwickelt, der direkt mit dem Data Warehouse kommuniziert. Abschließend sollen die Datensätze im Data Warehouse für beispielhafte Analysen und Reports verwendet werden, um den Nutzen zu verdeutlichen.

Vorwort

Mein Dank gilt Prof. Dr. Rony G. Flatscher für die Betreuung und immer freundliche Unterstützung bei der Erstellung dieser Bachelorarbeit, sowie der Wirtschaftsuniversität Wien.

Gender Erklärung

Aus Gründen der besseren Lesbarkeit wird in dieser Seminararbeit die Sprachform des generischen Maskulinums angewendet. Es wird an dieser Stelle darauf hingewiesen, dass die ausschließliche Verwendung der männlichen Form geschlechtsunabhängig verstanden werden soll.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

1 Einleitung

Das Kernthema dieser Bachelorarbeit liegt auf der Entwicklung und Verwaltung eines Data Warehouse Systemen. Die Forschungsfrage lautet, eignet sich das Datenbanksystems SAP HANA als Data Warehouse für Analysen und Reports.

1.1 Motivation

Im digitalen Zeitalter hat sich die Speicherung von Informationsflüsse zu einem selbstverständlichen Prozesse in Unternehmen entwickelt. Das führt dazu das die Menge der gespeicherten Daten kontinuierlich steigt. Allein die Speicherung erzeugt keinen Wettbewerbsvorteil, sondern die Auswertung und Interpretation der Informationen. Damit eine solche Analyse der Daten möglich ist, müssen entsprechende Rahmenbedingungen gegeben sein. Ein in der Literatur verbreitetes Konzept betreffend Datenhaltung, optimiert für Auswertungen, ist der Ansatz des Data Warehouses.

1.2 Zielsetzung

Ziel dieser Bachelorarbeit ist die Beantwortung der Forschungsfrage mit der Methode des Proof of Concepts. Dabei wird ein Prototyp entwickelt der Auskunft über die Machbarkeit geben soll. Die Ergebnisse der Arbeit sollen Auskunft darüber geben, ob das Datenbanksystems SAP HANA als Data Warehouse für Analysen und Reports geeignet ist.

1.3 Vorgehen

Für die Beantwortung der Forschungsfrage wird als erster Schritt ein allgemeiner Überblick über Datenspeicherung und Data Warehousing in der Literatur gegeben. Aufbauend auf der Theorie werden die Anforderungen und Rahmenbedingungen für den Prototypen definiert. Im letzten Kapitel wird der Prototyp anhand der erarbeiten Konzepte implementiert und die Ergebnisse präsentiert. Anhand der vorgezeigten Beispiele soll die Machbarkeit veranschaulicht werden und reflektiert werden, inwieweit sich das SAP HANA System für die Problemstellung eignet.

2 Grundlagen der Datenspeicherung

Dieses Kapitel deckt die theoretischen Grundlagen über Datenspeicher ab. Als Einstieg wird erläutert, was Informationen sind und wie sie in eine digitale Form gebracht werden. Anschließend wird erläutert, was ein Datenbanksystem ist, in welche Komponenten es unterteilt werden kann und wie diese zusammenspielen. Weiters werden zwei Ansätze für Datenbankmodelle vorgestellt, die für das Verständnis der nachfolgenden Kapitel hilfreich sind. Abschließend werden konzeptionelle Modellierungsmethoden präsentiert, die bei der Entwicklung von Datenbanksystemen Anwendung finden.

2.1 Informationen und Daten

Im digitalen Zeitalter haben Daten einen neuen Stellenwert bekommen und werden folglich als wertvolle Unternehmensressource angesehen. Dadurch, dass die Sammlung, Speicherung und Auswertung in der Regel computerunterstützt verläuft, ist es notwendig, dass diese in digitaler Form verfügbar sind. Grundsätzlich kann zwischen analogen und digitalen Daten unterschieden werden. Der Hauptaspekt von analogen Daten ist, dass sie durch kontinuierliche Funktionen repräsentiert werden und die Informationen dadurch stufenlos sind. (Hansen et al., 2015) Ein einfaches Beispiel für eine Quelle von analogen Daten ist das Quecksilberthermometer, bei dem die Temperatur über die Höhe des Flüssigkeitsstands definiert wird. (Hansen et al., 2015) Für die Weiterverarbeitung von Computern stellt diese Stufenlosigkeit der Informationen ein Problem dar, weshalb eine Transformation der Daten notwendig ist. Durch ein Abtasten der analogen Daten in einem festgelegten zeitlichen Intervall kann jedem Messpunkt ein digitaler Messwert zugeordnet werden. Im nächsten Schritt können diese digitalen Daten von Computern weiterverarbeitet werden, wobei sie rechnerintern durch Bitfolgen repräsentiert werden. Mithilfe von sogenannten Zeichensätzen ist es dem Rechner möglich Texte zu interpretieren. In der Regel wird ein Zeichen durch ein Byte repräsentiert, wodurch 256 verschiedene Zeichen dargestellt werden können. (Hansen et al., 2015) Einheitliche Zeichensätze haben den

Vorteil, dass sie den Austausch von Informationen zwischen Rechnern von verschiedenen Herstellern deutlich vereinfachen. Einer der ersten international genormter Zeichensätze ist der American Standard Code for Information Interchange (ASCII). (Hansen et al., 2015)

2.2 Datenbanksysteme

Ein Datenbanksystem ermöglicht eine computergestützte Datenverarbeitung von Informationen und besteht im Kern aus einer Datenbank und einer dazugehörigen Datenverwaltung. Eine Datenbank ist ein zentraler Datenbestand, welcher über festgelegte Zugriffsverfahren Benutzern und Anwendungsprogrammen Daten zur Verfügung stellt. Verwaltet werden diese Anfragen von einem Datenbankverwaltungssystem (DBMS). Das DBMS übernimmt das Abrufen und Speichern der Daten und sorgt dafür, dass die Datenstrukturen und Zugriffsrechte eingehalten werden. Dadurch haben alle Anwendungsprogramme eine einheitliche Schnittstelle und müssen keine Ressourcen für die Datenverwaltung aufbringen.

2.2.1 ACID Prinzipien

Im betrieblichen Kontext werden Datenbanksysteme von mehreren Benutzern verwendet, welche Lese- und Schreiboperationen auf die Unternehmensdaten ausführen. Diese Transaktionen der Benutzer können zeitgleich erfolgen, was bei falschem Verhalten des DBMS Konflikte in den Datensätzen erzeugen kann. Damit solche Zustände aufgelöst werden können, müssen vorab Grundregeln und Mechanismen definiert werden. Ein konventioneller Ansatz hierfür sind die ACID-Prinzipien für Datenbanksysteme. (Hansen et al., 2015)

Atomicity (Atomarität)

Eine Transaktion eines Benutzers besteht aus einzelnen Operationen. Wenn eine der Operationen fehlschlägt, ist es notwendig alle Änderungen der Transaktion rückgängig zu machen. Das Prinzip der Atomarität ist folglich nur erfüllt, wenn alle Einzelschritte einer Transaktion komplett oder gar nicht ausgeführt werden.

Consistency (Konsistenz)
Die Korrektheit und Widerspruchsfreiheit aller Informationen einer Datenbank wird als konsistenter Zustand bezeichnet. Wird eine Transaktion ausgeführt die Datensätze einfügt, verändert oder löscht, muss sichergestellt werden, dass nach gültigem Abschluss weiterhin eine konsistente Datenbank vorhanden ist.

Isolation (Abgrenzung)
Grundsätzlich können Datenbanksysteme von mehreren Benutzern gleichzeitig verwendet werden. Mithilfe von Sperrmechanismen für die Datensätze wird gewährleistet, dass bei parallelem Lesen und Schreiben keine Probleme auftreten. Das Prinzip der Abgrenzung sorgt folglich dafür, dass Transaktionen von Benutzern sich nicht gegenseitig beeinflussen und für den Anwender unsichtbar bleiben.

Durability (Dauerhaftigkeit)
Die Eigenschaft der Dauerhaftigkeit besagt, dass Änderungen aus einer korrekt ausgeführten und konsistenten Transaktion dauerhaft in der Datenbank gespeichert sein müssen. Sobald Datensätze gespeichert sind, dürfen Speicherausfälle oder Systemabstürze kein Grund für Verlust von Informationen sein. Sichergestellt wird dies mithilfe von Logdateien, anhand welcher die Datensätze vollständig reproduziert werden können im Falle eines Systemfehlers.

2.2.2 ANSI-SPARC-Dreischichtenmodell

Das ANSI-SPARC Dreischichtenmodell wurde 1975 von einem amerikanischen Normgremium entwickelt und gilt seitdem als Standard für die Architektur von Datenbanksystemen. Der Ansatz sieht vor die verschiedenen Teilbereiche eines Informationssystems zu entkoppeln. Die dadurch erreichte Datenunabhängigkeit ermöglicht es, dass Änderungen auf einer Ebene keine Abstimmung mit den anderen Schichten benötigen. Das folgende Konzept des Dreischichtmodells ist in der Literatur von Hansen et al. (2015) nachzulesen.

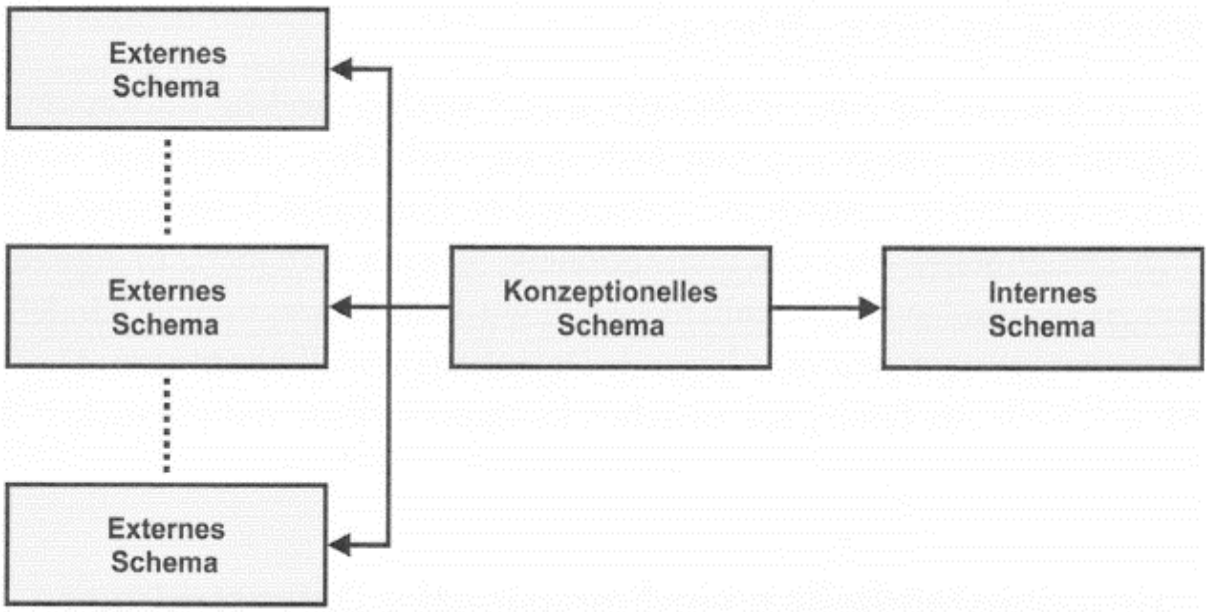

Abbildung 1: ANSI-3-Schichten-Konzept (Hahne, 2014, S. 27)

Auf der konzeptionellen Ebene wird in einem zweistufigen Verfahren das vollständige Informationssystem abgebildet. Im ersten Schritt wird das konzeptionelle Datenbankmodell abgeleitet, welches sämtliche Daten enthält und zusätzlich beschreibt, in welcher Beziehung diese zueinanderstehen. Dieses Modell ist noch nicht abgestimmt auf das konkrete Datenbankmodell des Datenbanksystems. Durch eine Ausrichtung auf dieses wird das logische Datenbankmodell, auch implementatives Modell genannt, abgeleitet. Dieses Modell berücksichtigt zusätzlich zu den Daten und Beziehungen die gegebenen Speicherstrukturen. Im Gegensatz zum konzeptionellen Modell wird es deshalb auch vom Datenbanksystem unterstützt.

Auf der externen Schicht werden angepasste Sichten für Benutzer und Applikationen zur Verfügung gestellt. Die jeweiligen Sichten bilden eine Teilmenge des konzeptionellen Schemas ab. Welche Daten darin enthalten sind, ist vom konkreten Anwendungsfall abhängig. Die Stückelung des Gesamtschemas bewirkt, dass nur notwendige Daten weitergegeben werden und die Sichten dadurch schlanker und übersichtlicher werden. Zusätzlich kann durch Rechteverwaltung das Abrufen von sensiblen Daten, wie z.B. Kundendaten, eingeschränkt werden.

Die interne Ebene befasst sich mit der physikalischen Implementierung der konzeptionellen Ebene. Hierbei werden beispielsweise die Zugriffsverfahren, die Benutzerverwaltung, die Sicherheitsmechanismen und die Formen der Datenspeicherung definiert. Entscheidungen auf dieser Ebene haben Auswirkung auf die Performance des Datenbanksystems. Grundsätzliche Ziele sind die optimale Nutzung der Speicherkapazitäten und die Minimierung der Zugriffszeiten. Verwaltet wird diese Ebene von einem Datenbankadministrator.

2.3 Datenbankmodelle

Es gibt unterschiedliche Ansätze für Datenbankmodelle, welche sich in den Strukturen und Beziehungen für die Datenspeicherung unterscheiden. Für eine Entscheidung ist es deshalb notwendig, die konkreten Anforderungen an das Datenbanksystem zu kennen. Dadurch kann das Datenbankmodell besser auf den konkreten Anwendungsfall abgestimmt werden und Performanceverlust vermieden werden. Im Folgenden werden zwei Ansätze erläutert, die für die kommenden Kapitel relevant sind.

2.3.1 Relationales Datenbankmodell

Das relationale Datenbankmodell zeichnet sich dadurch aus, dass die Speicherung der Daten in Form von Tabellen erfolgt. Es wurde Anfang der siebziger Jahre von Edgar Frank Codd entwickelt und ist wegen seiner sehr detaillierten und verständlichen Darstellung eines der häufigsten eingesetzten Modelle für Datenbanksysteme. (Fasel & Meier, 2016)

Grundsätzlich stellt jede Datenbanktabelle eine Entität mit einem Namen und entsprechenden Attributen dar, wobei die Spalten die Attribute darstellen und jede Zeile, auch Tupel genannt, einen eigener Dateneintrag repräsentiert. (Hansen et al., 2015) Für eine eindeutige Identifikation der einzelnen Tupel ist es notwendig ein Attribut als Primärschlüssel festzulegen. Dabei ist zu beachten, dass sich der Wert des Primärschlüssels in der Tabelle nicht wiederholen darf. Die Beziehungen zwischen den Entitäten werden mithilfe von Fremdschlüsseln dargestellt.

Tabelle Mitarbeiter und *Abteilung* zeigen ein Beispiel für eine Modellierung einer Mitarbeiterliste und deren dazugehörigen Abteilungsliste anhand des relationalen Datenmodells, wobei die Primärschlüssel unterstrichen sind. Die Zugehörigkeit der Mitarbeiter zu den jeweiligen Abteilungen wird mit dem Fremdschlüssel Abteilungs-Nr. repräsentiert.

Mitarbeiter-Nr.	Vorname	Nachname	Abteilungs-Nr.	Gehalt
M1	Irene	Müller	A2	55.000
M2	Eva	Schulze	A1	90.000
M3	Klaus	Huber	A2	60.000
M4	Martin	Meier	A3	35.000

Tabelle 1 Mitarbeiter: Relationale Darstellung einer Mitarbeiterliste

Abteilungs-Nr.	Name	Standort
A1	Leitung	Berlin
A2	Verkauf	Wien
A3	Buchhaltung	München

Tabelle 2 Abteilung: Relationale Darstellung einer Abteilungsliste

Für relationale Datenbanksysteme wird die transaktionsbasierte Sprache SQL verwendet. Sie ist wichtigste relationale Abfragesprache und ist durch die ISO normiert. (Fasel & Meier, 2016) Mithilfe von SQL kann die Datenbank definiert, bearbeitet und Abfragen auf die Daten getätigt werden. Ein besonderer Vorteil der Sprache ist die simple Syntax, die ohne Vorkenntnisse gelesen werden kann. Folgendes Beispiel zeigt eine SQL-Abfrage, die nach allen Mitarbeitern sucht, deren Gehalt kleiner als 50.000 ist.

```
SELECT *
FROM Mitarbeiter
WHERE Gehalt < 50000;
```

Quellcode 1: SELECT Statement Beispiel

Die Rückgabe dieser Abfrage ist die gesamte Zeile des Mitarbeiters M4. Das Symbol * legt fest, dass alle Spaltenattribute aufgelistet werden sollen. Alternativ können auch einzelne Rückgabewerte angegeben werden.

2.3.2 Multidimensionales Datenbankmodell

Das multidimensionale Datenbankmodell fand seine erste Anwendung als Analysewerkzeug für betriebswirtschaftliche Kennzahlen, wo es eine unterstützende Rolle für Entscheidungsprozesse einnahm. Die Besonderheit dieses Konzeptes ist die Betrachtung der Gewinne, Umsätze oder Verluste aus verschiedenen Perspektiven, wie z.B. Zeit, Ort oder Produkt. Die Perspektiven können dabei als Dimensionen bezeichnet werden, die es ermöglichen, die Kennzahlen in einen multidimensionalen Raum abzubilden. Die Dimensionen sind hierbei die Koordinaten, die gemeinsam auf einen Datenwert verweisen. Für eine genauere Navigation im Datenwürfel können die Dimensionen in Ebenen unterteilt werden, welche auch hierarchisch geordnet werden können. Beispiel hierfür wäre die die Aufteilung der Dimension Zeit in Tag, Monat, Quartal und Jahr. Zur besseren Veranschaulichung wird das multidimensionale Datenmodell in der Literatur häufig als Würfel abgebildet. (Köppen et al., 2012)

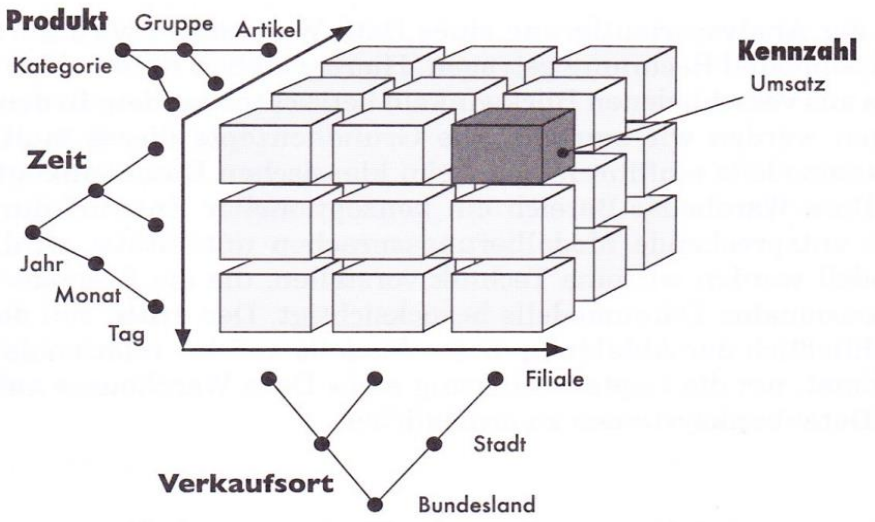

Abbildung 2: Datenwürfel (Köppen et al., 2012, S. 43)

In *Abbildung 2* entspricht jede Kante des Würfels einer Perspektive wodurch sich drei Dimensionen ergeben. Einschränkung zur Anzahl der Dimensionen für einen Datenwürfel gibt es keine, wobei diese Würfel als Hyperwürfel oder Hyper-Cube bezeichnet werden. (Adusei et al.,2019) Für die Speicherung der Datensätze eines Würfels werden multidimensionale Arrays verwendet. (Köppen et al., 2012)

2.4 Konzeptionelle Modellierungsmethoden

Wie bereits im Kapitel 2.3 erläutert, dienen konzeptionelle Datenmodelle dazu, das vollständige Informationssystem mit all seinen Daten und Beziehungen abzubilden. Für die Modellierung dieser Datenmodelle wurden verschiedene Notationen entwickelt. Im Folgenden werden die wichtigsten Standards präsentiert.

2.4.1 Entity Relationship Modell

Das Entity Relationship Modell (ER-Modell) ist eine Beschreibungsmethode, die dazu verwendet werden kann, um konzeptionelle Datenmodelle zu entwickeln. Durch die Entitätsorientierte Darstellung eignet sich das ER-Modell, um die Datenbasis von relationalen Datenbanksystemen darzustellen. In der Notation nach Chen besteht das ER-Modell aus drei Grundelementen, als da wären die Entitätstypen, Attribute und Beziehungstypen. (Adusei et al.,2019)

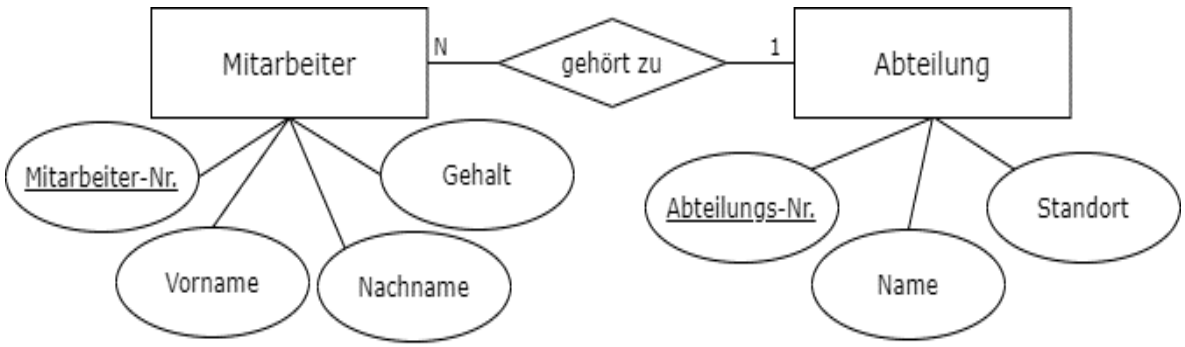

Abbildung 3: ER-Modell eines Mitarbeiters und dessen Abteilung

Abbildung 3 zeigt das Beispiel für eine Mitarbeiterliste und Abteilungsliste aus Kapitel 2.4.1 abgebildet in einem ER-Modell. Die einzelnen Attribute werden als Ovale dargestellt, wobei die Primärschlüssel unterstrichen werden. Die Beziehung der beiden Entitäten wird durch eine Raute repräsentiert, wobei die Kardinalität zusätzlich angegeben wird. In diesem Beispiel ist ein Mitarbeiter genau einer Abteilung zugehörig, im Gegensatz dazu kann eine Abteilung mehrere Mitarbeiter beinhalten. Dieses konzeptionelle Modell, dargestellt als ER-Modell, kann verwendet werden, um ein logisches

Datenmodell abzuleiten. Bei Verwendung eines relationalen Datenbanksystems erhält man dasselbe relationale Datenmodell wie *Tabelle Mitarbeiter* und *Tabelle Abteilung*. Bei der Transformation wurde die Beziehung der Entitäten, dargestellt als Raute, in einem Fremdschlüssel abgebildet und dem Mitarbeiter hinzugefügt.

2.4.2 ADAPT-Modellierung

Das ADAPT-Modell ist ein Beschreibungsmodell, welches speziell für die konzeptionelle Datenmodellierung von multidimensionalen Datenbankmodellen entwickelt wurde. Es ermöglicht eine grafische Darstellung der einzelnen Dimensionen und lässt eine Einteilung in Hierarchieebenen zu. Im Gegensatz zum ER-Modell ist diese Modellierungsform nicht geeignet für eine Auflistung der einzelnen Daten und deren Beziehungen, sondern wird verwendet um die fachlichen Anforderungen der Dimensionen festzuhalten. (Adusei et al.,2019) *Abbildung 4* zeigt ein Kunden und eine Produkt Dimension mit mehreren Hierarchien.

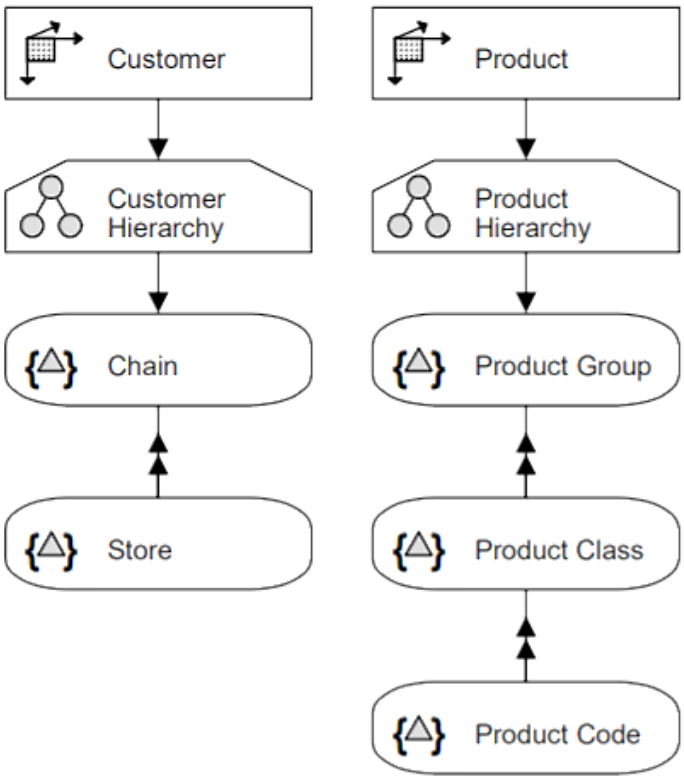

Abbildung 4 Kunden und Produkt Hierarchien (Hahne, 2014, S.79)

3 Data Warehouse

Der Begriff des Data Warehouse-Systems wird im allgemeinen Sprachgebrauch vielseitig verwendet, wobei er durch zwei Bereiche besonders geprägt wurde. Einerseits in den Bereichen der Informatik für ein Konzept eines Datenbanksystems und zweitens im betriebswirtschaftlichen Kontext als ein Analysewerkzeug zur Entscheidungshilfe. (Bauer, 2013) Das folgende Kapitel führt diese beiden Kontexte genauer aus und erläutert dabei die jeweiligen Anwendungsmöglichkeiten in denen das DWH eingesetzt werden kann. Weiters wird der Unterschied zu den konventionellen operationalen Datenbanksystemen verdeutlicht. Abschließend werden verschiedene Architekturansätze vorgestellt und evaluiert. Diese theoretischen Konzepte dieses Kapitels bilden die Grundlage für die Planung und Implementation des Prototyps.

3.1 Anwendungsgebiete

Technologische Fortschritte zwingen Unternehmen dazu ihre Arbeitsweisen zu überdenken und ihre Informationssysteme neu zu gestalten. Eine in steirischen Betrieben durchgeführte Studie ergab, dass 62% der Führungskräfte Digitalisierung explizit in ihre Unternehmensstrategie einbeziehen und 17% sie teilweise einbeziehen. (Rauter et al., 2021) Diese Entwicklungen sorgen dafür, dass das Speichern von Datenflüssen zu einem standardisierten Prozess geworden ist. Die dadurch wachsenden Datenbestände stellen für Unternehmen eine wertvolle Ressource darstellen, allerdings wird allein durch die Sammlung nicht das volle Potenzial ausgeschöpft. Die zentrale Motivation für die Entwicklung eines DWH beruht darauf, diese Daten aufzubereiten und auszuwerten.

Ziel der Auswertungen ist es, neue Erkenntnisse zu gewinnen und somit einen Wettbewerbsvorteil zu schaffen. Das klassische Anwendungsbeispiel für ein DWH ist die Betrachtung des Ist-Zustands, welche in Form von einem Bericht den Entscheidungsträgern monatlich vorgelegt werden kann. Eine mögliche Erweiterung dieses Systems ist die Integration von Planwerten.

Dadurch ist ein Vergleich des Ist- und Soll-Zustandes möglich, welcher Auskunft über die Erfüllungsgrade der einzelnen Kennzahlen gibt. Zusätzlich können Warnmechanismen, z.B. in Form von Ampelsystemen, implementiert werden, um Missstände frühzeitig zu erkennen und ein Entgegensteuern zu ermöglichen. (Bauer, 2013)

Ein weiteres Anwendungsgebiet vom DWH ist die Auswertung von komplexen Fragestellungen. Für solche umfangreichen Analysen werden verschiedene Datenquellen verwendet, die entsprechend transformiert werden, bevor sie ins DWH geladen werden können. (Bauer, 2013) Beispiel für eine solche Fragestellung wäre, wie sich verschiedene Marketingstrategien auf den Absatz auswirken. Dabei kann analysiert werden, wie effektiv die unterschiedlichen Werbekanäle auf die einzelnen demografischen Gruppen wirken. In diesem Fall würden die Kundengruppe und die Werbekanäle eine Dimension im DWH darstellen.

3.2 Merkmale eines Data Warehouses

Der US-amerikanische Informatiker Bill Inmon ist international für seine Arbeit am DWH bekannt und gilt in Fachkreisen auch als „Vater des Data Warehousing". (Devlin, 2018) Inmon selbst hat bereits in den frühen 90er Jahren Merkmale für DWH festgelegt, auf die heute noch verwiesen wird. (Devlin, 2018) Die folgenden Kriterien verdeutlichen den Kern des DWH-Konzeptes und helfen dabei, es von operativen Datenbanksystemen abzugrenzen.

Thema-orientiert

Im Gegensatz zu den operativen Systemen werden die Daten nicht prozessorientiert strukturiert, sondern auf die jeweilige Thematik ausgerichtet. (Devlin, 2018) Die Struktur im DHW ist darauf ausgelegt, die Fragen von Entscheidungsträgern zu beantworten. Bereits die Wahl der einzelnen Dimensionen grenzt das Themengebiet ein, da jede eine verdichtete Sammlung von Datensätzen eines Bereiches abbildet. Außerdem hat die Wahl der Datenquellen eine Auswirkung auf die Ergebnisse.

Integriert

Im Zuge des Extraktions-, Transformations- und Ladeprozesses werden Daten aus verschiedenen Quellen zusammengeführt. In der Literatur wird im Kontext der Datenintegration häufig der Begriff „single version of truth" verwendet. (Devlin, 2018) Dieser soll verdeutlichen, wie essenziell eine Standardisierung aller Datensätzen im DWH ist. Dabei geht es vor allem darum, einheitliche Messgrößen oder Definitionen für Kennzahlen zu verwenden. (Devlin, 2018) Allgemein verständliche Datensätze, wie z.B. Telefonnummern, können bei heterogenen Datenstrukturen Probleme bei der Auswertung verursachen.

Nicht-flüchtig

Ein Grundprinzip von einem DWH ist, dass einmal gespeicherte Datensätze das System nicht mehr verlassen. Im Gegensatz zu operativen Systemen gibt es im DWH keine Löschung von Datensätzen, weil das dem Ziel der lückenlosen Dokumentation widersprechen würde. (Devlin, 2018) Datensätze werden deshalb nur in Ausnahmefällen aus dem System entfernt.

Zeitlich veränderbar

Alle Datensätze in einem DWH benötigen eine zusätzliche Zeitkomponente, damit eine zeitliche Einordnung bei Auswertungen möglich ist. Weiters werden alte Datensätze nicht überschrieben, sondern entsprechend markiert und abgelegt, damit sie jederzeit wieder abrufbar sind. (Devlin, 2018) Dadurch soll gewährleistet werden, dass der gesamte Zeitraum der Daten betrachtet und analysiert werden kann.

3.3 Data Warehouse Architektur

Um das Konzept des DWH besser zu verdeutlichen, werden zuerst die technischen Unterschiede zu operativen Datenbanksystemen erläutert. Operative Produktivsysteme basieren auf dem Online Transaction Processing (OLTP) Konzept und sind für die schnelle Ausführung von kurzen Transaktionen optimiert. Klassisches Beispiel hierfür ist die Verbuchung eines Warenverkaufes im Datenbanksystem. In diesem Anwendungsfall entfalten

diese Systeme ihr volles Potenzial. Für komplexe statistische Abfragen sind sie jedoch nicht geeignet. Ein Grund dafür ist das sehr stark normalisierte Datenbankmodell, welches vorsieht, die Tabellen auf ihre kleinstmöglichen Entitäten aufzuspalten. Um die richtigen Informationen für die Auswertungen abzufragen, müssen die einzelnen Tabellen wieder verknüpft werden, was sich bei großen Datenmengen negativ auf die Performance auswirkt. Ein weiterer Nachteil ist, dass die analytischen Abfragen die Leistung des Produktivsystems beeinflussen können. Je komplexer die Abfragen werden, desto mehr Rechenleistung wird beansprucht, was zur Folge haben kann, dass das gesamte Datenbanksystem überlastet wird.

Aus diesem Grund wurde das DWH entwickelt, welches parallel zum operativen System läuft und ausschließlich für die Auswertung von Datensätzen eingesetzt wird. (Geisler, 2014) Die Daten kommen dabei aus einer oder mehreren Datenquellen. Der Vorgang der Dateninklusion in das DWH wird mithilfe eines ETL (Extract, Transform, Load) Prozesses realisiert. Als ersten Schritt werden die verschiedenen Datensätze geladen und bereinigt. Anschließend erfolgt die Transformation, in der die Daten in einen konsistenten einheitlichen Datenbestand umgewandelt werden. Als letzten Schritt werden die Daten in das DWH geladen. (Hahne, 2014)

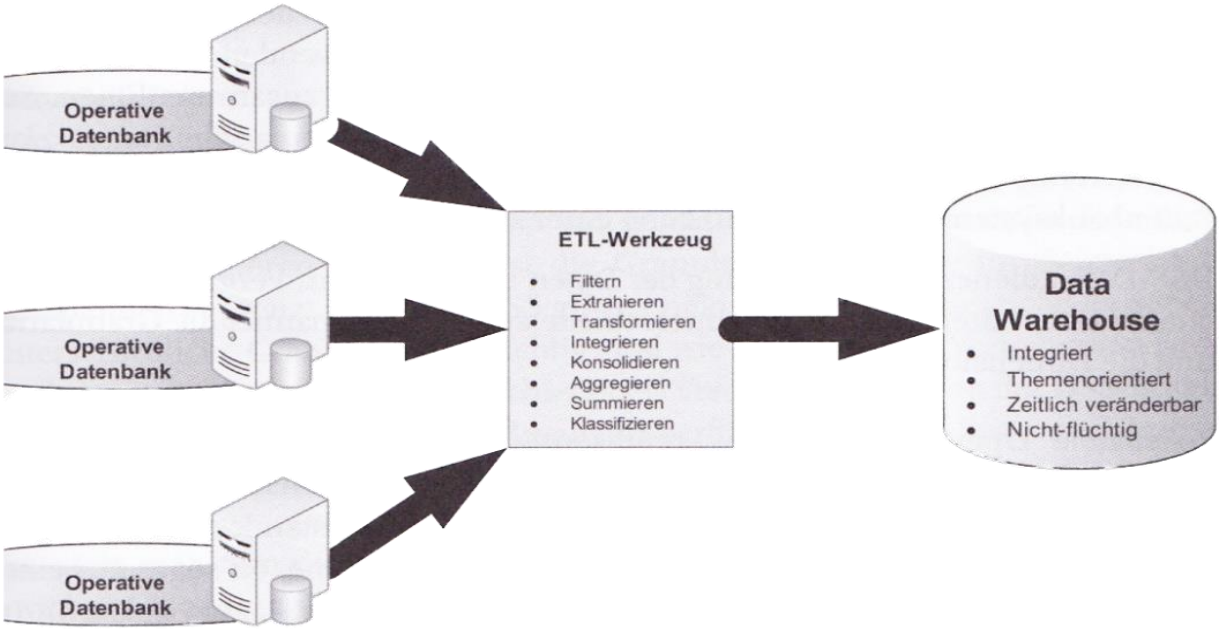

Abbildung 5: Architektur eines Data Warehouse (Geisler, 2014, S.399)

Abbildung 5 zeigt den Aufbau eines DWH Systems und veranschaulicht wie die Daten in das DWH gelangen. Nach der erfolgreichen Inklusion der Daten kann mit den Auswertungen begonnen werden, wobei die Applikationen und Benutzer die Anfragen direkt oder indirekt an das DWH stellen können. Häufig werden Online Analytical Processing (OLAP) Programme verwendet die den Endanwender dabei unterstützen, analytische Abfragen auf die Datensätze des DWH durchzuführen. (Hahne, 2014)

3.4 OLAP - Online Analytical Processing

Online Analytical Processing (OLAP) umfasst Systeme und Software die einen multidimensionalen Analyseansatz verfolgen und für die Auswertung größerer Datenmengen, wie sie im DWH vorliegen, vorgesehen sind. (Geisler, 2014) Zielsetzung dieses Ansatzes ist es, eine benutzerfreundliche Umgebung zu schaffen, die es Anwendern ermöglicht, komplexe Fragestellungen mit möglichst wenig Vorkenntnissen zu beantworten.

Das multidimensionale OLAP Modell wird auch als OLAP Würfel bezeichnet. Für eine Auswertung des Würfels werden in der Literatur folgende OLAP Operationen beschrieben, die den Nutzern verschiedene Sichten ermöglichen. (Köppen et al., 2012)

DRILL DOWN & ROLL UP

Mithilfe von Drill Down oder Roll Up kann zwischen den Hierarchieebenen der Dimensionen navigiert werden. Beispielsweise kann durch ein Drill Down auf die Dimension Ort von dem Attribut Land auf Bundesland geschlossen werden und somit eine feinere Granularität erreicht werden.

SLICE & DICE

Die Operationen Slice und Dice ermöglichen es, Teilausschnitte des OLAP Würfels zu selektieren. Mithilfe der Slice Methode wird eine Scheibe aus dem Datenwürfel ausgeschnitten. Dafür wählt man einen Messwert einer Dimension und erhält dann alle Datensätze, die diesem Wert entsprechen. Wird beispielsweise auf der Dimension Zeit das Jahr 2020 gewählt, erhält man eine zweidimensionale Tabelle mit allen Datensätzen des Jahres 2020.

Die Dice Operation arbeitet nach demselben Prinzip, nur dass in diesem Fall mehrere Dimensionswert festgelegt werden und deshalb ein multidimensionaler Würfel zurückgegeben wird.

PIVOT (ROTATE)
Die Pivot Operation wird verwendet, um den OLAP Würfel um die eigene Achse zu rotieren. Durch die Rotation verändert sich die Dimensionskombination, die der Betrachter sieht und es entsteht eine neue Perspektive auf die Datensätze.

DRILL ACROSS
Mithilfe der Drill Across Operation kann zwischen verschiedenen Datenwürfeln navigiert werden. Diese Operation wird verwendet, wenn es mehrere Datenquellen gibt, die in unterschiedlichen OLAP Würfeln abgebildet sind. Voraussetzung dafür sind einheitliche Dimensionsstrukturen zwischen den Datenwürfeln.

3.5 ROLAP - Relationale Implementierung

Der OLAP Ansatz basiert auf einem multidimensionalen Datenbankmodell, dennoch gibt es Implementierungskonzepte die eine relationale Speicherung der Daten ermöglichen. Dadurch, dass relationale Datenbanksysteme weitgehend standardisiert sind, werden sie von mehr Systemen unterstützt, was die Implementierung von relationalen OLAP Systeme anwenderfreundlich gestaltet. (Köppen et al., 2012)

Für die Implementation eines ROLAP-Systems reichen zumeist die Standardkenntnisse über SQL-Techniken. In Bezug auf Speicherverbrauch ist die multidimensionale Speicherung mittels Arrays effizienter. Bei der relationalen Speicherung müssen zusätzlich Fremdschlüssel angelegt werden, um die Relationen zwischen den Datensatz abzubilden. Dieser erhöhte Speicheraufwand kann dadurch ausgeglichen werden, dass die relationale Struktur flexibler ist und besser mit Veränderungen der Dimensionen umgehen kann. (Köppen et al., 2012)

3.5.1 Sternschema

Der Erfolg des DWH-Konzeptes baut auf zwei Grundprinzipien auf. Erstens auf die klare Abgrenzung zum Produktivsystem und zweitens auf die Modellierung der Daten in Sternen und Würfeln. (Kimball, 2002) Diese von Ralph Kimball entwickelte Abbildung der Daten in einem Sternenmuster wird Sternschema genannt.

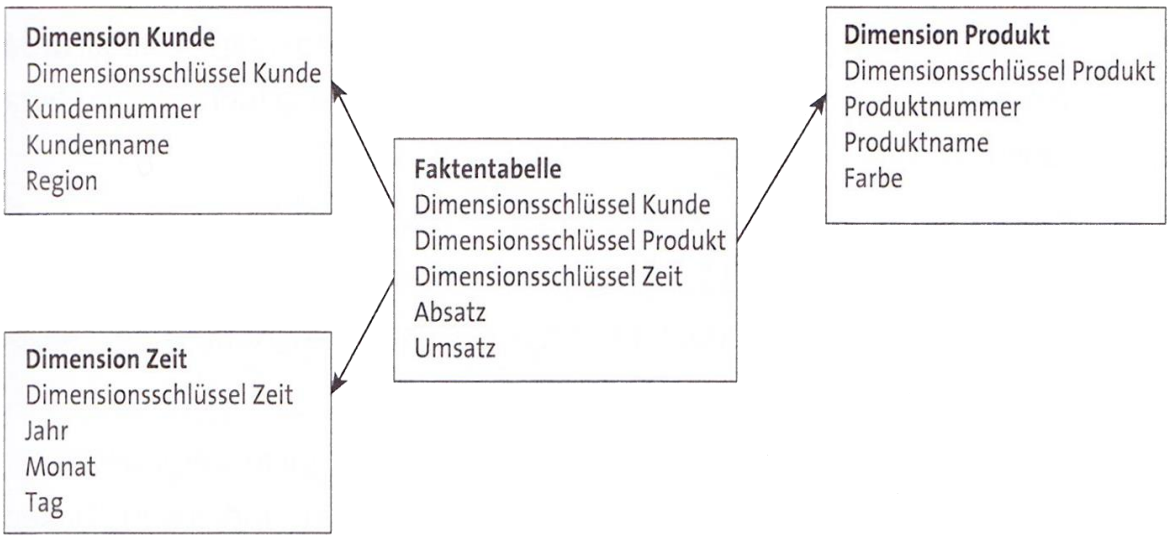

Abbildung 6: Sternschema (S.41 Adusei, Rötting, Yamada)

Abbildung 6 zeigt ein Beispiel für ein Sternschema. Bei diesem Ansatz steht im Zentrum eine Faktentabelle, die durch Relation mit sämtlichen Dimensionen verbunden ist. Diese Relationen werden in der Faktentabelle in Form von Fremdschlüsseln abgebildet, in der Regel werden dabei speicherschonende Integer Werte verwendet. (Adusei et al.,2019) Zusätzlich zu den Fremdschlüsseln können auch Kennzahlen enthalten sein. Die einzelnen Dimensionen werden dabei in einer denormalisierten Form abgebildet. Dadurch besteht in der klassischen Form des Sternschemas jede Dimension aus genau einer Tabelle. (Adusei et al.,2019) Vorteil des Sternschemas ist die einfache Struktur, welche einen hohen Grad an Flexibilität bietet. Änderungen an den Dimensionen können meistens allein durch das Hinzufügen von weiteren Attributen gelöst werden.

3.5.2 Galaxien Schema

Eine Modellierungsart, die auf dem Grundkonzept des Sternschemas aufbaut, ist das Galaxien Schema. Dieses Schema ermöglicht es mehrere Faktentabellen in einem Modell zu verwalten, weshalb es auch als Multifaktentabellen Schema bezeichnet wird. (Bauer, 2013) In diesem Ansatz stellt jede Faktentabelle eine eigene Galaxie dar und besitzt eine eigene Kombination an Dimensionen. (Adusei et al.,2019) Dadurch können Galaxien besser auf den jeweiligen Anwendungsfall angepasst werden und redundante Dimensionen verhindert werden.

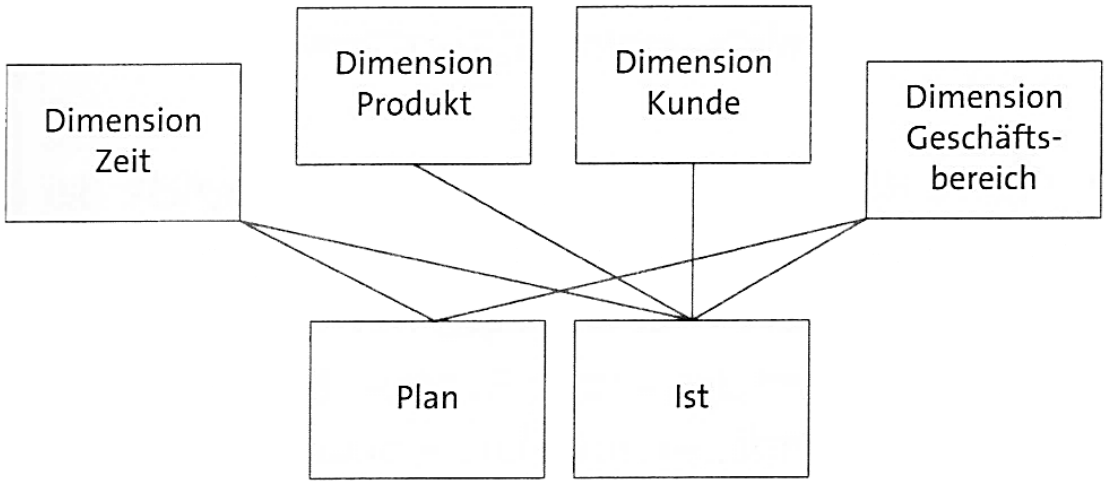

Abbildung 7: Galaxien Schema (S.43 Adusei, Rötting, Yamada)

Abbildung 7 zeigt ein Beispiel des Galaxien Schemas in dem die Daten in Plan- und Istwerte unterteilt sind. Grundsätzlich weisen Planwerte einen andern Detailgrad auf als Istwerte. Die Unterteilung in verschiedene Galaxien ermöglicht es, diese Unterschiede abzubilden. Durch die flexible Dimensionswahl der einzelnen Faktentabellen können die Datensätze an den Anwendungsfall angepasst werden und Nullstellen in den Dimensionstabellen vermieden werden. (Adusei et al.,2019) Besonders bei umfangreichen Datenmodellen sollte überprüft werden, ob eine Optimierung durch eine Aufspaltung in Galaxien möglich ist.

3.6 ETL-Prozess

Der ETL-Prozess ist dafür verantwortlich ein Datenbanksystem mit Daten zu versorgen. Er besteht aus Extraktion, Transformation und Speicherung der Daten. Die Implementation dieses Vorgangs ist sehr zeitintensiv und nimmt in der Regel den größten Aufwand bei der Entwicklung eines DWH-Systems in Anspruch. (Dhanda & Sharma, 2016)

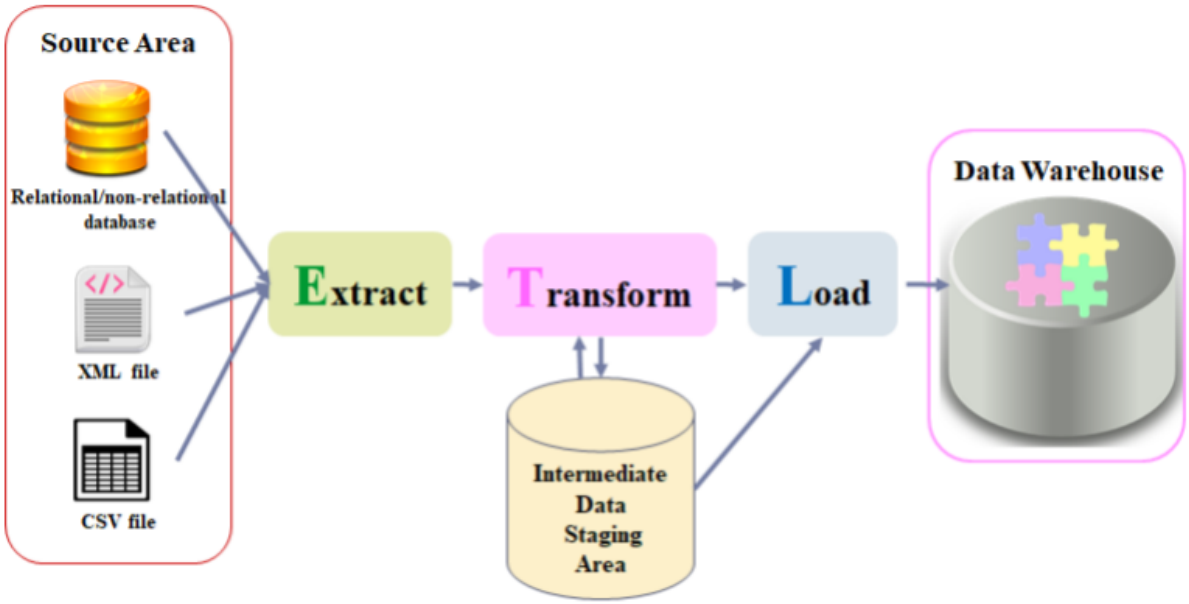

Abbildung 7: Aufbau eines ETL Prozesses (Homayouni, 2018, S.10)

Die Extraktion kann aus einer oder mehreren Datenquellen erfolgen. Die klassische Quelle sind operative Datenbanksysteme, darüber hinaus können Daten aus CSV-, XML- oder Textdateien stammen. Bei diesem Beschaffungsprozess wird zuerst nach allen relevanten Daten gefiltert und anschließend werden diese in eine für den Transformationsvorgang unterstützte Form gebracht.

Die Transformation der Rohdaten ist notwendig, damit die Datensätze dem Schema der Zieldatenbank entsprechen. Dabei kann generell zwischen syntaktischer und semantischer Transformation unterschieden werden. Erstere umfasst die formalen Anforderungen, wie einheitliche Datentypen.

Wesentliche Bestandteile der semantischen Transformation sind die Filterung, Eliminierung, Harmonisierung und Ergänzung von Datensätzen. Dabei steht vor allem der inhaltliche Aspekt im Fokus. Diese Schritte helfen dabei redundante, veraltete und fehlerhafte Daten auszusortieren und somit die Qualität der Daten langfristig zu verbessern. Der Transformationsprozess ist der entwicklungs- und rechenintensivste Teil des ETL-Prozesses. (Dhanda & Sharma, 2016) Der konkrete Aufwand hängt dabei von der Anzahl der Datenquellen und der gewünschten Datenqualität ab. Besonders unterschiedliche Strukturen und Dateitypen der Quellen erschweren es, die Informationen auf einen einheitlichen Standard zu bringen.

Sobald die Datensätze auf das Schema ausgerichtet sind, kann mit dem Ladeprozess begonnen werden. Grundsätzlich kann zwischen synchronem und asynchronem Laden unterschieden werden. (Dhanda & Sharma, 2016) Der synchrone Ansatz sieht vor, das DWH immer auf den aktuellen Stand zu halten. Dadurch werden Änderungen in den Datenquellen sofort an das DWH weitergeleitet. Im Gegensatz dazu erfolgen beim asynchronen Ansatz die Updates periodisch oder ereignisgesteuert. Welche Methode am besten geeignet ist, hängt von den Anforderungen an das DWH-System ab. Die Umsetzung eines Echtzeitsystems ist rechenintensiver und nicht in jedem Anwendungsfall sinnvoll. Beispielsweise ist für Auswertungen von historischen Quartalskennzahlen ein asynchroner Ladeprozess in einem täglichen Intervall vollkommen ausreichend. Weiters ist es empfehlenswert, die asynchrone Speicherung der Daten in betriebliche Ruhephasen, wie Betriebsschluss oder Wochenenden, einzuplanen.

3.7 In-Memory Datenbanken

Das klassische Speichermedium für Daten ist die Festplatte. Die wachsenden Datenmengen und rechenintensive Auswertungen bringen diese Technologie an ihre Grenzen. Aus diesem Grund geht der Trend in Richtung In-Memory Datenspeicherung. (Preuss, 2017)

Bei dieser Speichertechnik wird der Arbeitsspeicher anstatt den Festplatten verwendet. Ein weiterer Unterschied ist das Speicherformat, in dem die Daten abgelegt werden. Herkömmliche Datenbanken legen die Datensätze zeilenorientiert ab, wohingegen eine In-Memory Datenbank die Werte spaltenweise ablegt. Das ermöglicht ein Zusammenfügen gleicher Datenwerte, was bei großen Datenmengen die Performance deutlich verbessert. Besonders das DWH profitiert von dieser Komprimierung. Diese effizientere Nutzung der Rechenleistung macht die Implementation eines Echtzeitsystems umsetzbarer.

Da es sich beim Hauptspeicher um ein flüchtiges Medium handelt, würden die Daten bei einem Systemabsturz verloren gehen. In diesem Fall ist die Eigenschaft der Dauerhaftigkeit nicht erfüllt und es sind zusätzliche Maßnahmen notwendig. Möglich wäre eine Kombination aus regelmäßigen Momentaufnahmen und Transaktionsprotokollen in Form von Log-Dateien. (Preuss, 2017) Dadurch könnte im Falle eines Ausfalles der Ausschnitt geladen werden und die nicht erfassten Transaktionen mittels der Log-Dateien rekonstruiert werden.

4 Ressourcen

Kapitel 4 gibt eine Übersicht über die Software Ressourcen die für die Entwicklung des DWH-Systems, einschließlich des ETL-Prozesses, eingesetzt wurden. Dabei liegt der Fokus auf den Produkten des Softwareherstellers SAP. Neben den verwendeten Java Libraries nehmen diese den Großteil des entwickelten Prototyps ein.

4.1 SAP-Software

SAP ist ein börsennotierter Softwarekonzern mit Hauptsitz in Deutschland. Der Schwerpunkt des Unternehmens ist die Entwicklung von betriebswirtschaftlichen Softwareprodukten. Dabei besteht die Hauptproduktlinie aus Enterprise-Resource-Planning Systemen, welche es ermöglichen alle geschäftsrelevanten Unternehmensbereiche in ein Informationssystem abzubilden. Das neuste ERP Produkt ist SAP S4/HANA, welches auf dem hauseigenen SAP HANA Datenbanksystem läuft. (SAP, 2021a)

4.1.1 SAP HANA

SAP HANA ist ein Softwareprodukt von SAP das im Kern eine Implementation für ein spaltenbasiertes relationales DBMS bietet. Die Speicherung erfolgt dabei mittels einer In-Memory Datenbank, wodurch es möglich wird OLAP und OLTP Konzepte in einem System zu realisieren. (SAP, 2021b) Dieser Ansatz ermöglicht eine unkomplizierte Erweiterung des betrieblichen Informationssystems mit einem DWH.

Für die Implementation bietet SAP die Möglichkeit einer lokalen oder Cloud basierten Umsetzung. Die lokale Installation mit allen notwendigen OLAP Funktionen hat eine minimale Systemanforderung von 16GB RAM. Die lizenzfreie Entwicklung und Verwendung ist für Systeme bis zu 32GB möglich. (SAP, 2021b) Damit der Zugang der Projektarbeit nicht durch die minimalen Systemanforderungen eingeschränkt wird, wurde der Cloud basierte Ansatz gewählt. Die präsentierten Konzepte und Vorzeigebeispiele sind in beiden Versionen umsetzbar, ausschließlich der Installationsablauf und die Entwicklungsumgebung unterscheiden sich.

4.1.2 SAP HANA Cloud

Für diese Arbeit wird die Testversion von SAP HANA Cloud verwendet. Die freie Nutzung der Testversion ist 90 Tage möglich. Bei der Verwendung von HANA sind keine zusätzlichen Installationen auf dem eigenen System notwendig. Für die Entwicklung des OLAP Systems wird SAP Business Application Studio verwendet, welches als Browseranwendung in der HANA Cloud bereitgestellt wird. Weiters werden inkludierte Werkzeuge für das Verwalten des Datenbanksystems verwendet. Diese ermöglichen beispielsweise das Ausführen von SQL Skripten auf die Datenbankinstanzen um Tabellen zu erstellen oder Datensätze zu laden.

4.1.3 SAP Business Application Studio

SAP Business Application Studio ist eine in der HANA Cloud inkludierte Entwicklungsumgebung, die über Mozilla Firefox oder Google Chrome betrieben werden kann. (SAP, 2021c) Hauptmerkmal dieses Werkzeuges sind die Dev Spaces die für die unterschiedlichen Entwicklungsszenarien erstellt werden können. Diese Dev Spaces agieren als isolierte virtuelle Maschinen die sich in der HANA Cloud befinden und an den jeweiligen Anwendungsfall angepasst sind.

4.1.4 SAP Analytics Cloud

SAP Analytics Cloud ist ein in der Cloud verfügbares Softwaretool für die Visualisierung von Daten. Die Daten werden dabei entweder über einen Datei Import oder über eine direkte Verbindung zu einem Datenbanksystem bereitgestellt. In dieser Arbeit wird die Testversion verwendet, die eine freie Nutzung von bis zu 90 Tagen ermöglicht.

4.1.5 SAP HANA JDBC Driver

Für den Zugriff der ooRexx Applikationen auf das Datenmodell im HANA Cloud Datenbanksystem ist das Verwenden des SAP HANA JDBC Drivers notwendig. Das Einbinden der ngdbc JAR Datei ermöglicht es mit URL, Username und Password eine Connection zu erzeugen und SQL Statements auf die Datenbank auszuführen. (SAP, 2021d)

4.2 Open Object Rexx

Ein wesentlicher Bestandteil des Prototyps ist die Entwicklung eines eigenen ETL-Prozesses. Hierfür wurde die Skriptsprache Object Rexx eingesetzt. Diese ist eine vollwertige Programmiersprache, deren Hauptmerkmal ein leicht verständlicher und lesbarer Code ist. Darüber hinaus unterstützt es das verbreitete Konzept der objektorientierten Programmierung. Zusätzlich wird ein Open-Source-Projekt der Rexx Language Association namens Open Object Rexx verwendet. Diese bietet eine einfache Implementierung der Skriptsprache Object Rexx. (Rexx Language Association, 2021)

Zusätzlich zu Open Object Rexx wird für die Entwicklung das Framework BSF4ooRexx (Bean Scripting Framework für ooRexx) verwendet. Es ermöglicht die Interaktion mit der Java-Laufzeitumgebung, wodurch eine Implementierung von Java Klassen und Methoden ermöglicht wird. Darüber hinaus ist es möglich, den Object Rexx Code als Java Code zu tarnen, wodurch eine Kompatibilität zu Java unterstützten Technologien geschaffen wird. (Flatscher, 2016)

4.3 JSoup – HTML Parser

Für die Datenbeschaffung im Extraktionsschritt werden unterschiedliche Datenquellen verwendet. Sämtliche Daten werden dabei mittels Web Scraping Techniken gesammelt. Für diesen Vorgang wird die JSoup Java Library eingesetzt, die Klassen und Methoden für das Abrufen, Filtern und Manipulieren von HTML Webseite bereitstellt. (JSoup, 2021) Diese Ressourcen werden verwendet, um für verschiedene öffentlich zugängliche Webseiten Web Scraping Skripte zu entwickelt. Die Informationen die von den verschiedenen Webseiten bereitgestellt werden, weisen keine einheitliche HTML Struktur auf. Weshalb die einzelnen Skripte direkt auf die HTML Struktur der jeweiligen Seite ausgelegt sind. Dabei wird direkt auf die einzelnen HTML Elemente, bzw. Tags, zugegriffen und deren Inhalte oder Attribute abgerufen. Für die genaue Identifizierung der Elemente werden JSoup Methoden verwendet die es ermöglichen nach Klassennamen oder

HTML-Attributen zu filtern. Damit die bereitgestellten Web Scraping Skripte auf andere Webseiten ausgelegt werden können, muss diese Identifizierung und Zuordnung der der Elemente adaptiert werden.

4.4 JSON - Datenformat

Bei dem Textformat JSON (JavaScript Object Notation) handelt es sich um eine effiziente und für Menschen leicht lesbare Datenstruktur. Die Besonderheit dieser Struktur ist, dass sie auf den gängigen Programmierkonventionen aufbaut und dadurch sprachübergreifend einsetzbar ist. (JSON, 2021) Aus diesem Grund wird dieses Format verwendet um die gewonnenen Datensätze der einzelnen Web Scraper zwischenzuspeichern. Durch die Transformation in eine JSON Struktur werden die Daten vereinheitlicht und befinden sich in einem Format das universell weiterverarbeitet werden kann.

5 Fachliches Konzept des Prototyps

Der empirische Teil dieser Bachelorarbeit folgt dem Ansatz des Proof-of-Concepts. Diese spezielle Forschungsmethodik besteht im Kern aus der Realisierung eines theoretischen Konzeptes für eine vorab definierte Problemstellung. Ziel dieser Methode ist die Entwicklung eines funktionsfähigen Vorzeigebeispiels, das die Machbarkeit der theoretischen Konzepte demonstriert. Aufgabenstellung dieser Arbeit ist die Entwicklung eines Prototyps eines DWH Systems, das auf den SAP HANA Datenbanksystem basiert. Im folgenden Kapitel werden die allgemeinen formalen Anforderungen für den Prototypen festgelegt und die theoretischen Lösungsansätze für die Implementierung präsentiert.

5.1 Aufgabenstellung

Die Problemstellung, die es im Rahmen dieser Bachelorarbeit zu lösen gilt, ist die Entwicklung eines DWH Systems für Reporting- und Analysezwecke, ausgerichtet auf Mietwohnungen im Bundesland Wien. Das OLAP basierte DWH soll es ermöglichen, Immobiliendaten auszuwerten und grafisch darzustellen. Zusätzlich wird ein OLTP System entwickelt, welches die gesamten Informationen über die einzelnen Wohnungsimmobilien abbildet und ein einfaches Verwalten der Datensätze ermöglicht. Aufgabe dieses Systems ist es eine konsistente Datenbasis zu schaffen, die von dem OLAP System weiterverarbeitet werden kann. Die Implementierung der beiden Systeme soll in einem gemeinsamen Datenbanksystem realisiert werden. Für die Datenbeschaffung der Testdaten werden ooRexx Web Scraper entwickelt. Diese sind auf öffentlich zugängliche Mietwohnungsplattformen ausgerichtet und sammeln die essenziellen Daten der einzelnen Mietinserate. In einem weiteren Schritt werden diese Daten aufbereitet, transformiert und in das OLTP System integriert. Der gesamte Ablauf von der Datenbeschaffung bis hin zu dem Laden der Datensätze in das Datenbanksystem soll einen vollständigen ETL Prozess zwischen einer Datenquelle und einem Zielsystem darstellen.

5.1.1 Wesentliche Features

Folgende Funktionen müssen in der fertigen Lösung enthalten sein:

- Speichern und Verwalten von Mietwohnungsinseraten und allen dazugehörigen Informationen in einem Datenbanksystem.
- Speicherung der für analytische Zwecke relevanten Werte in einem DWH System.
- Beschaffung von Testdatensätzen von unterschiedlichen Mietwohnungsplattformen mittels Web Scraping.
- Vollständiger ETL Prozess, der die Datensätze aus unterschiedlichen Quellen lädt und in der Zieldatenbank vereinigt.
- Grafische Aufbereitung der Analysen und Berichte für den Anwender.

5.1.2 Qualitätsziele

Tabelle 3 zeigt die zentralen Qualitätsziele für die Softwareentwicklung nach der ISO 25010:2011. (Goulão et al., 2016)

Qualitätsziel	Motivation und Erläuterung
Hohe Kompatibilität (Interoperabilität)	Das Einbinden des Datenbanksystems in ein fremdes System soll mit angemessenen Aufwand verbunden sein
Modularer Aufbau (Änderbarkeit)	Ein modularer Aufbau soll das Austauschen oder Wiederverwenden von einzelnen Komponenten forcieren
Performance (Effizienz)	Die große Menge an Datensätzen macht eine effiziente Nutzung der Rechenleistungen erforderlich
Leichte Portierbarkeit (Übertragbarkeit)	Der Prototyp dient als Vorzeigebeispiel. Eine Installation in wenigen Schritten ist deshalb erstrebenswert.
Attraktivität (Usability)	Analysen und Reports sollen in grafisch ansprechenden Diagrammen dargestellt werden

Tabelle 3: Qualitätsziele des Prototyps

5.1.3 Rahmenbedingungen

Tabelle 4 enthält die definierten Rahmenbedingungen für den Prototypen:

Rahmenbedingungen	Beschreibung
Implementierung in Object REXX	Der ETL Prozess wird in der Skriptsprache Object Rexx realisiert
Plattform-unabhängig	Die Implementierung des Prototyps ist auf allen Betriebssystemen möglich
Moderate Hardwareausstattung	Die Implementierung des Prototyps ist auf allen marktüblichen Notebooks möglich

Tabelle 4: Rahmenbedingungen des Prototyps

5.2 Lösungsstrategie

Die Entwicklung des Prototyps kann in mehrere Teilbereiche unterteilt werden. *Abbildung 8* zeigt eine abstrakte Darstellung der Architektur und zeigt wie die einzelnen Komponenten miteinander interagieren. Die Hauptbausteine der fertigen Lösung sind der ETL-Prozess und das HANA Datenbanksystem in der Cloud. Das Datenbanksystem besteht dabei aus einem normalisierten Datenbankschema für die generelle Abbildung der

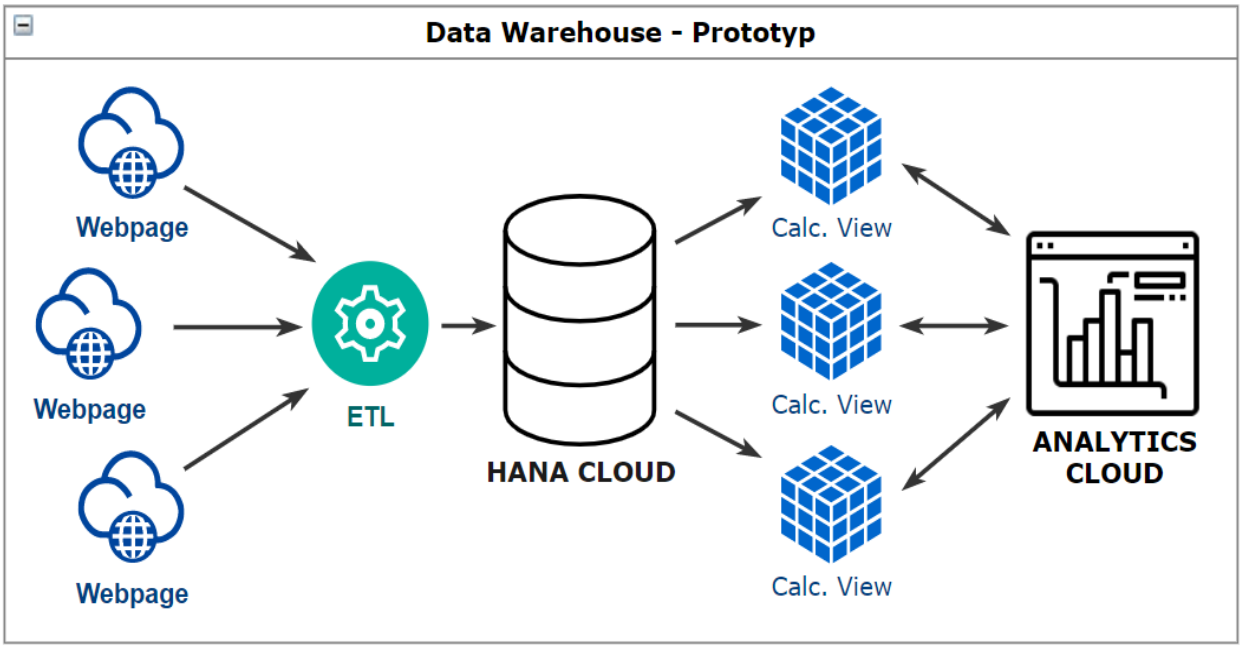

Abbildung 8: Lösungsstrategie des Prototyps

Datensätze und verschiedener OLAP Anwendungen, die die Datensätze für Analysezwecke aufbereiten. Dafür werden die von SAP bereitgestellten Calculation Views verwendet. In dem nächsten Schritt werden die Analysen in der Analytics Cloud grafisch aufbereitet. Die Verbindung erfolgt dabei mittels CSV Import oder einer Live Connection, bei der die Analytics Cloud direkt auf die aktuellen Datensätze zugreift. In der hier verwendeten Trial Version steht die Live Verbindung nicht zur Verfügung, weshalb die Datensätze manuell per CSV-Datei importiert werden.

5.3 Konzeptionelles Datenmodell

Im Zuge der Entwicklung wurde ein Entity Relationship Modell erstellt, welches alle Informationen abbildet und einen ersten Ansatz eines konzeptionellen Datenbankmodells für das OLTP System darstellt.

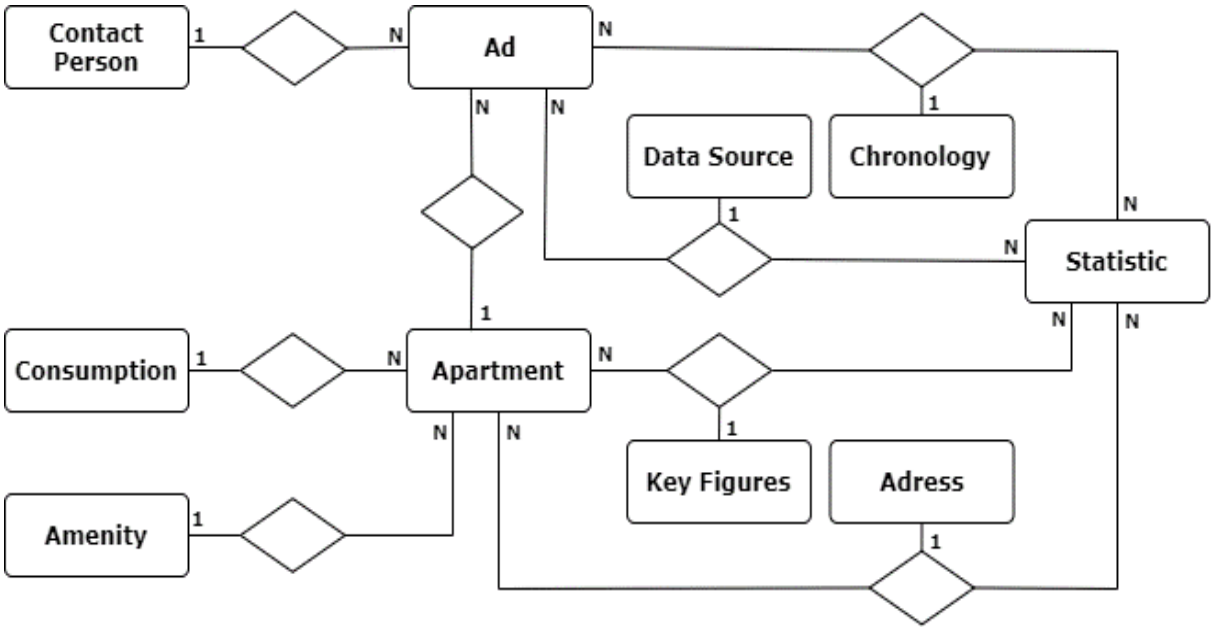

Abbildung 9: Konzeptionelles Datenbankmodell des Prototyps

Abbildung 9 gibt einen ersten Überblick über die einzelnen Entitäten und deren Relationen zueinander. Bei der Wahl der Struktur und den Beziehungen lag vor allem die Anwendbarkeit auf unterschiedliche Datenquellen im Fokus. Die Präsentation der Informationen der unterschiedlichen Webseiten ist nicht einheitlich. Deshalb ist es notwendig, ein möglichst flexibles Schema zu entwickeln, welches dennoch einen entsprechenden Detailgrad aufweist.

Tabelle 5 enthält eine ausführliche Beschreibung zu den einzelnen Entitäten und Auskunft darüber welche Art von Attributen diese enthalten.

Entität	Beschreibung
Ad	**Ad** bildet die einzelnen Angebote/Inserate der Mietwohnungsplattformen ab. Jedem Inserat wird genau eine Wohnung zugeordnet. Des Weiteren sind Metadaten über Zeit, Datenquelle und Kontakt des Inserates enthalten.
Apartment	**Apartment** stellt die eigentliche Mietwohnung dar. Dazu gehören Informationen über Adresse, Kosten, Raumaufteilung, Ausstattung und Energiewerte.
Contact Person	**Contact Person** enthält sämtliche Information bezüglich des Kontaktes für das Inserat. Dabei ist es egal, ob es sich um eine natürliche Person oder eine Organisation handelt.
Data Source	**Data Source** gibt Auskunft über die einzelnen Quellen, von denen Daten abgerufen werden. Für eine genaue Zuordnung wird jede URL dokumentiert, von der Daten gesammelt wurden.
Chronology	**Chronology** enthält zeitbezogenen Informationen, wie z.B. das Erstellungsdatum der Datensätze.
Key Figures	**Key Figures** ist eine Auslistung aller Kennzahlen bezüglich der einzelnen Mietwohnung. Dazu gehören Miete, Betriebskosten, Heizkosten, Kaution, Maklerprovision, etc.
Adress	**Adress** enthält die Adressdaten wie Land, Postleihzahl und Anschrift für eine Mietwohnung.
Consumption	**Consumption** beschreibt die Energieeffizienz der Mietwohnung. Das beinhaltet Daten über die Energieklasse, den Heizwert und die Art der Heizung.
Amenity	**Amenity** gibt Auskunft über die Ausstattung und besondere Merkmale einer Mietwohnung. Dazu zählt die Nutzung eines privaten Parkplatzes oder Kellerabteils.
Statistic	**Statistic** bildet extern bezogene statistische Daten über Mietimmobilien ab, welche Kennzahlen kombiniert mit Informationen über Ort, Zeit und Herkunft enthalten.

Tabelle 5: Beschreibung der Entitäten

5.4 ETL-Prozess

Die Entwicklung eines ETL-Prozesses für die Beschaffung und Integration von Datensätzen ist eine der Kernaufgaben dieser Arbeit. Dieser Prozess lässt sich in mehrere Arbeitsschritte unterteilen. Damit beim Zusammenführen der einzelnen ooRexx Applikationen alles korrekt abläuft, wurde der Ablauf vorab in der grafischen Spezifikationssprache Business Process Model and Notation (BPMN) abgebildet. Diese ermöglicht es, komplexe Geschäftsprozesse in einer standardisierten Form grafisch darzustellen, die sowohl für betriebswirtschaftliche als auch technische Akteure lesbar ist. (Object Management Group, 2021)

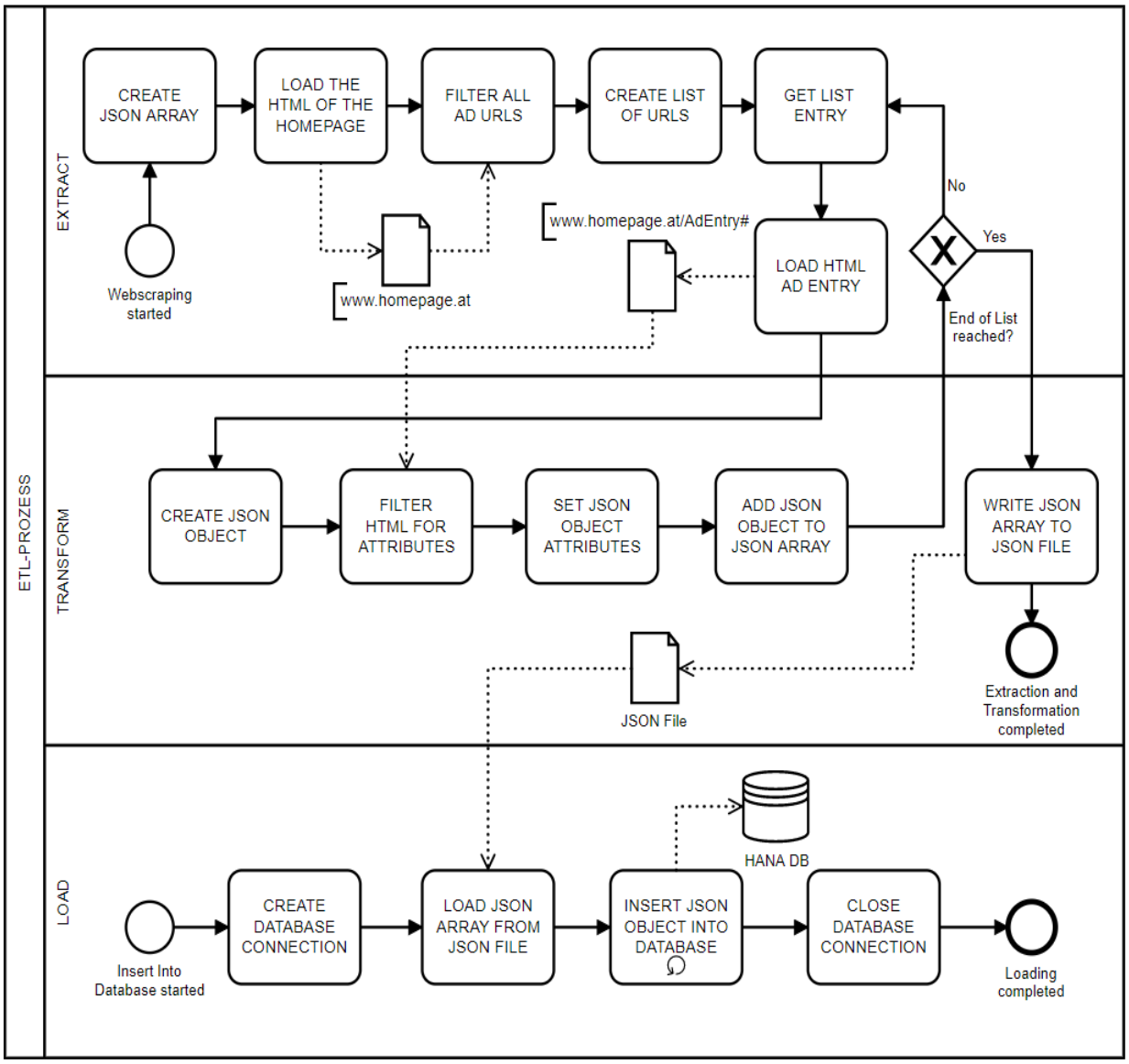

Abbildung 10: ETL Prozess ausgerichtet auf Mietwohnungsplattform

Abbildung 10 veranschaulicht den ETL-Prozess der für die Beschaffung, Transformation und Integration der Datensätze verantwortlich ist. In dieser Arbeit werden Informationen ausschließlich mittels Web Scraping Methoden gesammelt, welche von der Jsoup Library bereitgestellt werden. Für eine temporäre Speicherung der Daten wird die JSON Library verwendet.

5.4.1 Extraktion

Der erste Schritt des ETL-Prozesses behandelt die Extraktion der Daten aus den unterschiedlichen Quellen. Beim hier verwendeten Web Scraping werden die Inhalte der Webseiten im HTLM Format abgerufen, um anschließend einzelne Datensätze über die HTML Struktur zu selektieren. Der Zugriff auf die Webseiten erfolgt dabei über die URL. Als Hauptdatenquelle in diesem Projekt werden öffentliche Mietwohnungsplattformen verwendet, welche detaillierte Inserate für jede einzelne Wohnung bereitstellen. Jedes Inserat kann dabei über eine eindeutige URL abgerufen werden, die sich aus dem Hostnamen der Plattform und dem Dateipfad des Inserates ergibt. Als erster Schritt wird eine Liste aller Inserat URLs mittels Web Scraping von der jeweiligen Plattform extrahiert. Anschließend können in einem iterativen Verfahren die einzelnen Inserate abgerufen und die Datensätze gefiltert werden. Mit der Bereitstellung der Datengrundlage für den Transformationsprozess ist der Extraktionsprozess abgeschlossen.

5.4.2 Transformation

Die Aufgabe des Transformationsprozesses ist es, die unstrukturierten Daten, die durch den Extraktionsprozess gesammelt werden, auf einen einheitlichen Standard zu bringen. In syntaktischer Hinsicht bedeutet das, dass die Informationen auf einheitliche Datentypen gebracht werden müssen. Im Falle der Mietwohnungsinserate müssen die Kennzahlen, Adressdaten und Datumswerte in Datenformate transformiert werden, die mit sämtlichen Datenquellen kompatibel sind. Dagegen ist die semantische Transformation dafür zuständig, die inhaltlichen Widersprüche der Datenquellen zu beseitigen. Dabei entspringen die meisten Konflikte der Tatsache, dass die Informationen der einzelnen Mietwohnungsplattformen

unterschiedliche Granularitäten aufweisen. Zur Lösung der syntaktischen und semantischen Probleme werden die einzelnen Attribute aus der HTML Struktur gefiltert und anschließend in einem einheitlichen JSON Format, das auf den Entitäten von *Tabelle 5* basiert, zwischengespeichert. Das Ergebnis sind strukturierte und bereinigte Datensätze, die in das Datenbanksystem geladen werden können.

5.4.3 Laden

Im letzten Schritt des ETL-Prozesses werden die Datensätze in das SAP HANA Datenbanksystem integriert. Dieser Vorgang wird mittels eines SAP HANA Clients umgesetzt, der eine Verbindung zwischen der ooRexx Applikation und dem HANA System ermöglicht. Für die Verwendung des Clients muss lediglich der SAP HANA JDBC Driver eingebunden werden. Anschließend kann mit den entsprechenden Zugangsdaten eine Verbindung aufgebaut werden und die Daten können mittels INSERT INTO Statements eingefügt werden.

5.5 OLAP System

In den vorherigen Schritten wurde das Konzept für das OLTP Datenbanksystem und den ETL-Prozess beschrieben. Die letzte fehlende Komponente, die für den vollständigen Prototypen noch fehlt, ist das OLAP System. In SAP HANA werden OLAP Anwendungen in Form von Calculation Views umgesetzt. Diese können wiederrum in die drei Kategorien Attribute, Analytic und Calculation unterteilt werden. Die Entwicklung erfolgt dabei entweder skriptbasiert oder über eine grafische Modellierungsumgebung. In dieser Arbeit wurden alle Calculation Views mit dem Modellierungswerkzeug von Business Application Studio entworfen.

Attribute View

Attribute Views sind eine vereinfachte Ausprägung der anderen beiden Varianten. Sie dienen hauptsächlich dazu Attribute von unterschiedlichen Tabellen in eine Entität zu vereinen. Diese Entitäten können dann in weiterer Folge als Dimensionen in Analytic Views oder Calculation Views

eingebunden werden. (SAP, 2021e) Außerdem bieten sie die Möglichkeit die Hierarchieebenen frei zu modellieren. Dadurch kann der Grad der Granularität adaptiert werden und die Navigation in einem OLAP-Würfel präziser gestaltet werden.

Analytic View

Analytic Views verfügen über alle Funktionalitäten die in den Attribute Views verfügbar sind. Zusätzlich dazu bieten sie die Verwendung von Messwerten. (SAP, 2021f) Durch ihre Eigenschaften bieten sich Analytic Views für die Modellierung von OLAP-Würfeln an. Die Dimensionen werden dabei in Form von Attribute Views dargestellt die mittels eines Star-Join mit der Fakten Tabelle verknüpft werden.

Calculation View

Calculation Views sind eine erweiterte Form der Analytic Views. Sie ermögliche die Modellierung komplexer Sachverhalte und unterstützen dabei sämtliche Funktionen der Attribute und Analytic Views. (SAP, 2021g) Beispiel dafür ist der Vergleich von Plan- und Ist-Kennzahlen, welche in separaten Faktentabellen aufgeteilt sind. Calculation Views ermöglichen es verschiedene Faktentabellen miteinander zu verknüpfen.

Konzept für den Prototyp

Abbildung 11 zeigt einen abstrakten Aufbau des OLAP Systems der für die Realisierung des Prototyps vorgesehen ist. Das Konzept basiert dabei auf dem Galaxien-Schema erläutert in Kapitel 3. Das Schema beinhaltet die Faktentabellen IST und STATISTIC. Erstere stellt dabei einen OLAP-Würfel für Mietwohnungsdaten dar. Die Faktentabelle ist eine Kombination der Tabellenwerte AD, APARTMENT und KEY_FIGURES und verfügt über sämtliche Dimensionen. Die Faktentabelle STATISTIC ergibt sich aus den Tabellenwerten von STATISTIC und KEY_FIGURES und stellt einen OLAP-Würfel über die statistischen Datenwerte im System dar. Sie verfügt nur über relevante Dimensionen, wodurch Nullwerte in ansonsten redundanten Dimensionen vermieden werden.

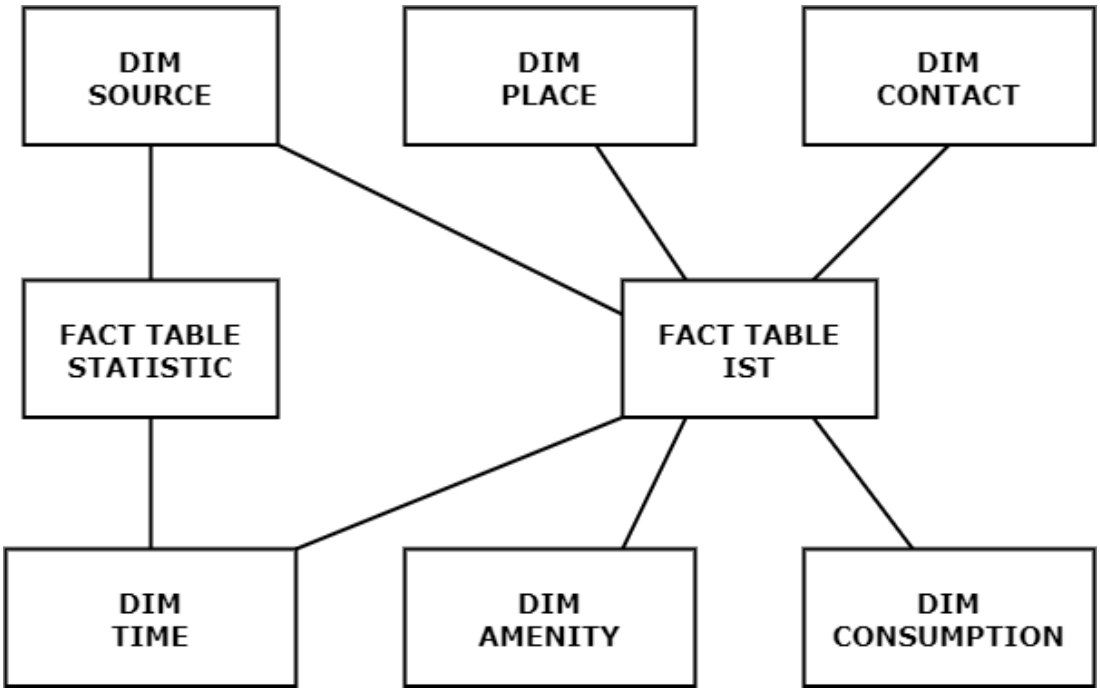

Abbildung 11: OLAP System nach dem Galaxien Schema

6 Implementierung des Prototyps

Aufbauend auf den theoretischen Konzepten aus dem vorherigen Kapitel wurde ein Prototyp entwickelt. Das folgende Kapitel befasst sich mit der Implementierung der ausgearbeiteten Konzepte und Modelle und präsentiert dafür beispielhafte Ausschnitte aus jedem Teilbereich. Ziel ist es die vorab definierten Funktionalitäten vorzuzeigen und dadurch die Machbarkeit eines DWH Systems in einer HANA Umgebung zu veranschaulichen.

6.1 OLTP System

Für die Erstellung des OLTP Systems musste das konzeptionelle Datenbankmodell auf das relationale und spaltenorientierte HANA Datenbanksystem ausgerichtet werden. Dafür mussten entsprechende Datentypen für die Attribute definiert werden und die Beziehungen zwischen den Entitäten durch Fremdschlüssel abgebildet werden. Das Ergebnis dieser Anpassungen ist das logische Datenbankmodell in *Abbildung 12*, welches von dem HANA Datenbanksystem unterstützt wird. Die Erstellung der Datenbankstruktur erfolgte mittels SQL Statements.

```
CREATE SCHEMA "IMMODB";
```

Quellcode 2: SQL CREATE SCHEMA Statement– IMMODB

```
CREATE COLUMN TABLE "IMMODB"."CONTACT_PERSON"
(contact_person_id INT NOT NULL GENERATED BY DEFAULT AS
IDENTITY UNIQUE, contact_info VARCHAR(45),
 phone_number VARCHAR(45),
email VARCHAR(45));
```

Quellcode 3: SQL CREATE TABLE Statement– CONTACT_PERSON

Quellcode 2 zeigt die Erstellung des Schemas IMMODB, welches als Container für sämtliche Tabellen dient. Anschließend wurden die einzelnen Tabellen mittels CREATE TABLE Statement erstellt. *Quellcode 3* zeigt das CREATE Statement der Datenbanktabelle für die Entität Contact Person. Der Tabellenschlüssel contact_person_id wird dabei automatisch mittels einer Sequenz erzeugt die den nächsten Zahlenwert, beginnend mit 1, vergibt.

```
CREATE COLUMN TABLE "IMMODB"."APARTMENT" ( apartment_id INT
NOT NULL GENERATED BY DEFAULT AS IDENTITY UNIQUE,
key_figures_id INT NULL,
adress_id INT NULL,
consumption_id INT NULL,
characteristic_id INT NULL,
room_count DOUBLE NULL,
construction_year INT NULL,
construction_type VARCHAR(45) NULL,
apartment_condition VARCHAR(45) NULL);
```

Quellcode 4: SQL CREATE TABLE Statement– APARTMENT

Quellcode 4 zeigt das CREATE Statements für die Erstellung der Tabelle APARTMENT. Wie bei der Tabelle Contact Person wird der Tabellenschlüssel mithilfe einer Sequenz erstellt. Weiters verfügt die Entität Apartment über mehrere Beziehungen zu anderen Entitäten die als Fremdschlüssel in den Attributten key_figures_id, adress_id, consumption_id und characteristic_id abgebildet wurden. Quellcode 5 zeigt die Zuweisung der einzelnen Fremdschlüsselbeziehungen auf die Tabellenattribute.

```
ALTER TABLE "IMMODB"."APARTMENT" ADD FOREIGN KEY (adress_id)
REFERENCES immodb.adress(adress_id)
ON DELETE CASCADE;

ALTER TABLE "IMMODB"."APARTMENT" ADD FOREIGN KEY
(key_figures_id) REFERENCES immodb.key_figures(key_figures_id)
ON DELETE CASCADE;

ALTER TABLE "IMMODB"."APARTMENT" ADD FOREIGN KEY
(consumption_id) REFERENCES immodb.consumption(consumption_id)
ON DELETE CASCADE;

ALTER TABLE "IMMODB"."APARTMENT" ADD FOREIGN KEY
(characteristic_id) REFERENCES immodb.characteristic
(characteristic_id) ON DELETE CASCADE;
```

Quellcode 5: ALTER TABLE Statement - Foreign Key Zuweisung

Diese Ausschnitte aus den SQL Statements veranschaulichen die Erstellung der Datenstruktur im HANA Datenbanksystem. Eine vollständige Auflistung der SQL Statements wurde im Anhang dieser Arbeit angefügt.

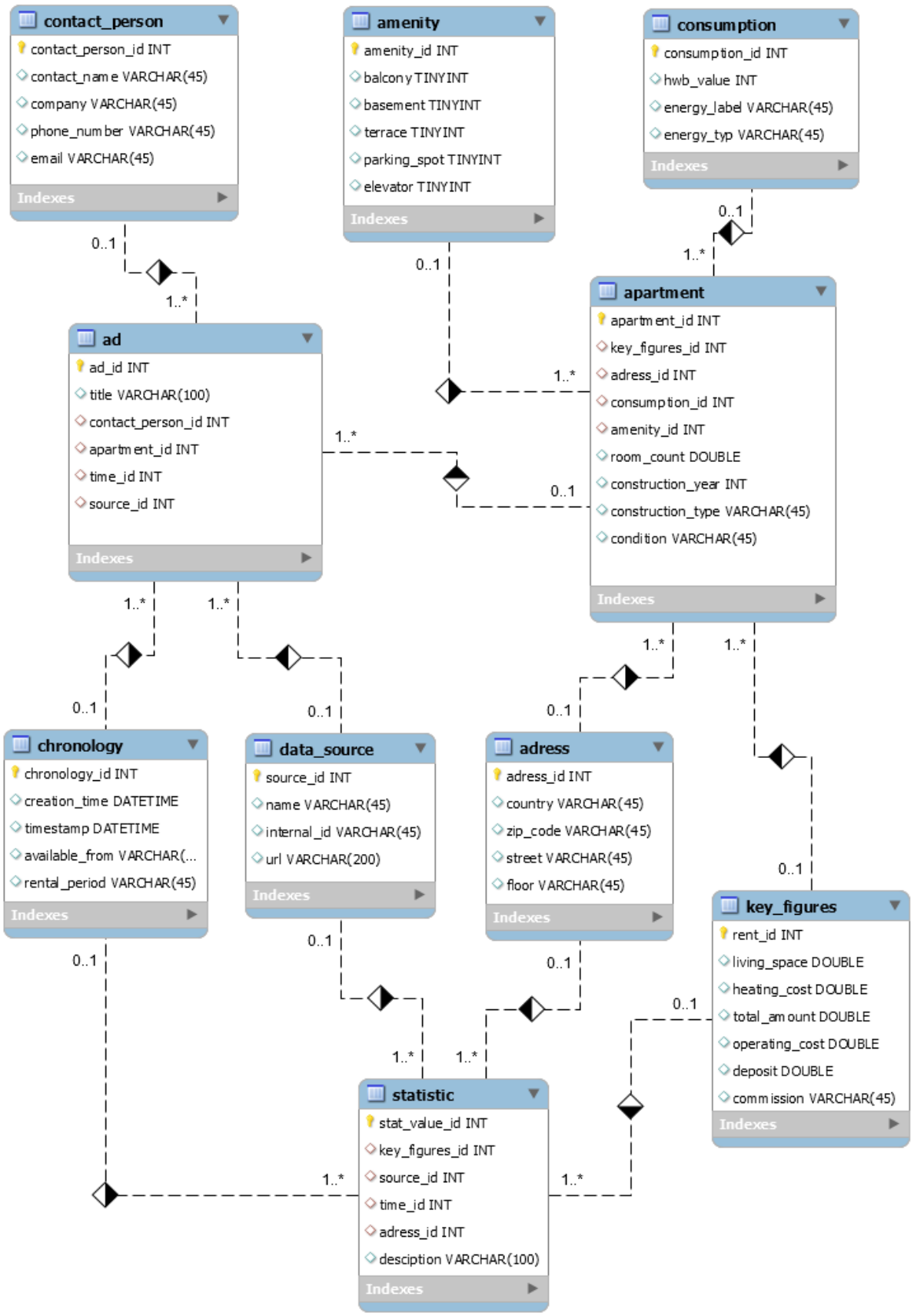

Abbildung 12: Logisches Datenbankmodell des Prototyps

6.2 ETL-Prozess

Die folgenden Programmausschnitte veranschaulichen die Funktionalitäten des entwickelten ETL-Prozesses. Das Web Scraping ist dabei auf die Mietplattform FindMyHome ausgelegt. Der vollständige Programmcode ist im Anhang angefügt.

Quellcode 6 zeigt das Startskript für die Extraktion und Transformation. Die Routine startWebscraping_FHM startet das Web Scraping für die Zielwebseite. Der Routine wird die URL der Startseite und der Pfad für eine Auflistung der Inserate übergeben. Hierbei ist anzumerken, dass Mietplattformen in der Regel über Suchfunktionen verfügen, die es ermöglicht eine Liste der Inserate anzuzeigen. Rückgabewert ist eine Aufzählung von Inserat Details, aufbereitet in einem strukturierten JSON Format. Mithilfe der Routine writeJSONtoFile werden diese Daten in ein JSON File gespeichert. Der Speicherpfad ist dabei das aktuelle Arbeitsverzeichnis, welches über System~getProperty(„user.dir“) ermittelt wird.

```
-- Homepage and Path of the Ad List
homepage = ("https://www.findmyhome.at")
adlist_path = („/immo/wohnung-mieten/wien“)

-- start ROUTINE for Web Scraping of findmyhome.at
CALL startWebscraping_FMH homepage, adlist_path
entryList = result

-- get current working directory
System = bsf.import("java.lang.System")
current_folder = System~getProperty("user.dir")

-- JSON File storage path
path = current_folder"\temp_data.json"

-- ROUTINE to write JSON Array to File
CALL writeJSONtoFile entryList~toString, path

::REQUIRES "BSF.CLS"
```

Quellcode 6: Start der Extraktion und Transformation

6.2.1 Extraktion der Inserat URLs

Quellcode 7 zeigt die Routine startWebscraping_FMH. Als erster Schritt wird ein JSON Array für die Speicherung der Einträge erstellt. Danach wird mittels URL und Jsoup Methode das HTML Dokument der Inserats Liste abgerufen. Das erhaltene Dokument enthält Verlinkungen zu allen Inseraten, welche mit Jsoup Methoden gefiltert werden können. In diesem Beispiel erfolgt die Filterung über das Klasseattribute „obj_list". Die erhaltene Liste an Elementen wird mit einer Schleife durchlaufen und die Inserat Referenz abgerufen. Die vollständige Inserats URL wird anschließend an die Routine scrapeAdEntry_FHM übergeben. Diese ruft ein Inserat auf und gibt die Informationen als JSON Objekt zurück. Abschließend werden die Einträge an das JSON Array angefügt und die gesamte Liste wird zurückgegeben.

```
::ROUTINE startWebscraping_FMH
  PARSE ARG homepage, adlist_path
    --create JSON Array
    entryList = .bsf~new("org.json.JSONArray")

    -- get HTML Document of the Ad List
    Jsoup = bsf.import("org.jsoup.Jsoup")
    mainpage = Jsoup~connect(homepage""adlist_path)~get()

    -- Filter all Elements from Class "obj_list"
    adlist = mainpage~getElementsByClass("obj_list")

    -- Loop through all Elements and extract the Ad URLs
    DO j = 0 TO (adlist~size - 1)
        adobj = adlist~get(j)~getElementsByTag("a")
        adref = adobj~first()~attr("href") - get Ad Path

        -- CALL ROUTINE to load each Ad and return JSON Object
        IF (adref \= "") THEN
            DO
                link = homepage""adref - assemble Ad URL
                CALL scrapeAdEntry_FHM link
                entryList~put(result) - add JSON Object
            END
    END
    RETURN entryList
```

Quellcode 7: Routine startWebscraping_FMH

6.2.2 Filterung und Transformation

Die Routine scrapeAdEntry_FMH wird genutzt um die einzelnen Inserate und deren Attribute aufzurufen. Dabei wird für jedes Inserat ein JSON Objekt erstellt, indem die Werte gespeichert werden. Als Eingabeparameter bekommt die Routine eine URL die direkt auf die Webseite des Inserates zeigt.

```
::ROUTINE scrapeAdEntry_FMH
  PARSE ARG url

    --Load the HTML Document of the Ad Entry
    Jsoup = bsf.import("org.jsoup.Jsoup")
    adpage = Jsoup~connect(url)~get()
```

Quellcode 8: Routine scrapeAdEntry_FMH – Aufrufen eines Inserats

```
    --Extract & Transform for TABLE KEY_FIGURES
    key_figures = .Key_Figures~NEW()

    --COLUMN <DOUBLE> TOTAL_AMOUNT
    temp = adpage~select("tr:contains(Monatliche
                          Gesamtkosten:) td")~last()

    IF (temp \= .nil & temp \= "") THEN
        DO
            temp = temp~ownText()~delWord(1,1)
            temp = temp~changeStr(".","")
            temp = temp~changeStr(",",".")
            key_figures~total_amount= temp
        END
    ELSE
        DO
            key_figures~total_amount= "NULL"
        END
```

Quellcode 9: Routine scrapeAdEntry_FMH – Filterung eines Attributes

Das HTML Dokument des Inserates kann anschließend gefiltert und die Werte zugewiesen werden. Damit eine Struktur gegeben ist, wurden für alle Entität aus *Tabelle 5* Klassen mit dazugehörigen Attributen erzeugt. *Quellcode 9* zeigt die Erzeugung der Klasse Key_Figures und der Zuweisung des Attributes total_amount. Die Klasse Key_Figures ist in *Quellcode 11*

abgebildet. Der Wert des Attributes wird dabei mittels der select Methode von Jsoup gefiltert, welche es ermöglicht komplexe Suchoperationen durchzuführen. Der gefilterte Wert wird anschließend bereinigt und auf einen einheitlichen Datentyp gebracht. Beim Attribut total_amount handelt es sich um die Gesamtmiete angegeben in Euro. Im Datenbankschema ist der Wert als DOUBLE definiert, weshalb das Währungszeichen und die Tausenderpunkte entfernt werden müssen. Außerdem muss das Dezimaltrennzeichen von einem Beistrich zu einem Punkt geändert werden.

Nachdem alle Entitäten erzeugt und die entsprechenden Attribute zugewiesen wurden, können diese dem JSON Objekt hinzugefügt werden. Dabei wird jede Entität in Form eines eigenen JSON Objekt angefügt. Für die Erzeugung der JSON Objekte wird die exportJSON Methode aus *Quellcode 11* verwendet. Diese Methode ist in allen Klassen der Entitäten enthalten und soll eine einheitliche JSON Struktur gewährleisten. *Quellcode 10* zeigt die Erzeugung und Rückgabe des JSON Objektes.

```
--create a JSON Object for the Ad Entry
jsonAdEntry = .bsf~new("org.json.JSONObject")

-- ADD the Entities to the JSON Object
jsonAdEntry~~put("adress", adress~exportJSON)
jsonAdEntry~~put("data_source", data_source~exportJSON)
jsonAdEntry~~put("chronology", chronology~exportJSON)
jsonAdEntry~~put("characteristic",
                 characteristic~exportJSON)
jsonAdEntry~~put("key_figures", key_figures~exportJSON)
jsonAdEntry~~put("apartment", apartment~exportJSON)
jsonAdEntry~~put("consumption", consumption~exportJSON)
jsonAdEntry~~put("contact_person",
                 contact_person~exportJSON)
jsonAdEntry~~put("ad", ad~exportJSON)

-- print JSON Object in Console
SAY jsonAdEntry~toString

RETURN jsonAdEntry
```

Quellcode 10: Routine scrapeAdEntry_FMH – Zuweisung zum JSON Objekt

```
::CLASS Key_Figures
  ::METHOD  living_space ATTRIBUTE
  ::METHOD  total_amount ATTRIBUTE
  ::METHOD  operating_cost ATTRIBUTE
  ::METHOD  heating_cost ATTRIBUTE
  ::METHOD  deposit ATTRIBUTE
  ::METHOD  commission ATTRIBUTE
  ::METHOD exportJSON

    jsonData = .bsf~new("org.json.JSONObject")

    jsonData~~put("total_amount", self~total_amount)
    jsonData~~put("living_space", self~living_space)
    jsonData~~put("operating_cost", self~operating_cost)
    jsonData~~put("heating_cost", self~heating_cost)
    jsonData~~put("deposit", self~deposit)
    jsonData~~put("commission", self~commission)

    RETURN jsonData
```

Quellcode 11: Klasse Key_Figures

6.2.3 Temporäre Speicherung der Daten

Die Routine writeJSONtoFile ist der letzte Schritt des Transformationsprozesses. Ausgeführt wird die Routine in Quellcode 6, nachdem alle Datensätze gesammelt und in Form eines JSON Array zurückgegeben wurden. Dieses JSON Array wird in Textform in ein JSON File gespeichert und wird beim Einlesen wieder als solches erkannt.

```
::ROUTINE writeJSONtoFile
  PARSE ARG input, path

    -- create FileWriter Instance
    fileWriter = bsf.import("java.io.FileWriter")

    file = fileWriter~NEW(path,.false)

    file~write(input)
    file~flush
    file~close
```

Quellcode 12: Routine writeJSONtoFile

6.2.4 Datenbank Verbindung

Für das Laden der Datensätze in die HANA Datenbank muss zuerst eine Connection definieren werden. In *Quellcode 13* wird eine Instance der Klasse SAPConnection erstellt und die Verbindungsinformationen über den Konstruktor übergeben. *Quellcode 15* zeigt die Klasse SAPConnection. Bei der Initialisierung der Klasse werden die Parameter verwendet um eine Verbindung zur Datenbank herzustellen.

```
-- SAP HANA Connection Information
url = "jdbc:sap://<SQL-Endpoint>"
user = "<DB-User>"
pw = "<DB-Password>"

-- connect to SAP HANA Database
conImmoDB = .SAPConnection~NEW(url,dbuser,dbpw)
```

Quellcode 13: Datenbank Verbindung einrichten

Nachdem eine Verbindung zur HANA Datenbank aufgebaut wurde, werden die Datensätze aus dem JSON File eingelesen. *Quellcode 14* zeigt wie aus den eingelesenen Datensätzen ein JSON Array erzeugt wird. Anschließend werden die Listeneinträge mit einer Schleife durchlaufen und die JSON Objekte der einzelnen Entitäten abgerufen. Das Einfügen der Datensätze in die Datenbank erfolgt über die Methoden der SAPConnection. In diesen werden die Werte in vorgefertigte Insert Statement eingefügt und diese ausgeführt.

```
-- get current project directory
System = bsf.import("java.lang.System")
current_folder = System~getProperty("user.dir")

-- import Ressources
FileReader = bsf.import("java.io.FileReader")
FileBuffered = bsf.import("java.io.BufferedReader")
JsonArray = bsf.import("org.json.JSONArray")

-- read the JSON Array from JSON File
reader = FileReader~NEW(current_folder"\temp_data.json ")
bfr = FileBuffered~NEW(reader)
content = bfr~readLine()
```

```
IF (content \= .nil | content \= "")THEN
    DO
        adList = JsonArray~NEW(content)
        DO i = 0 TO (adList~length - 1)
            adEntry = adList~get(i)

            data_source = adEntry~get("data_source")
            conImmoDB~insertIntoDATA_SOURCE(
                        data_source~get("name"),
                        data_source~get("internal_id"),
                        data_source~get("url"))

            chronology = adEntry~get("chronology")
            conImmoDB~insertIntoCHRONOLOGY(
                        chronology~get("creation_time"),
                        chronology~get("available_from"),
                        chronology~get("rental_period"))
            .
            .
            .
             ad = adEntry~get("ad")
             conImmoDB~insertIntoAD(ad~get("title"))

        END

    END
```

Quellcode 14: Einlesen und Speichern der JSON Daten

```
::CLASS SAPConnection
    ::METHOD conn ATTRIBUTE
    ::METHOD init
        USE ARG url, dbuser, dbpw
        DriverManager = bsf.import("java.sql.DriverManager")
        self~conn = DriverManager~getConnection(url,user,pw)
        SAY "Connected success"
        SAY "HANA Url:" url
        SAY "User: "dbuser
    ::METHOD insertIntoDATA_SOURCE
      EXPOSE conn
      USE ARG source_name, internal_id, url
      stmt=conn~createStatement
      stmt~executeUpdate("INSERT INTO IMMODB.DATA_SOURCE
       (SOURCE_NAME, INTERNAL_ID, URL) VALUES
       ('"source_name"', '"internal_id"', '"url"');")
```

```
::METHOD insertIntoCHRONOLOGY
  EXPOSE conn
  USE ARG ad_creation_time, available_from, rental_period
  stmt=conn~createStatement
  stmt~executeUpdate("INSERT INTO IMMODB.CHRONOLOGY
  (AD_CREATION_TIME, ENTRY_CREATION_TIME,
  AVAILABLE_FROM,RENTAL_PERIOD)
  VALUES('"ad_creation_time"',
  CURRENT_TIMESTAMP, '"available_from"',
  '"rental_period"') ;")
::METHOD insertIntoCONTACT_PERSON
  EXPOSE conn
  USE ARG contact_info , phone_number ,  email
  stmt=conn~createStatement
  stmt~executeUpdate("INSERT INTO IMMODB.CONTACT_PERSON
  ( CONTACT_INFO ,  PHONE_NUMBER ,  EMAIL )
  VALUES('"contact_info"', '"phone_number"', '"email"');")
::METHOD insertIntoADRESS
  EXPOSE conn
  USE ARG  country ,  zip_code ,  street ,  floor
  stmt=conn~createStatement
  stmt~executeUpdate("INSERT INTO IMMODB.ADRESS
  ( COUNTRY ,  ZIP_CODE ,  STREET ,  FLOOR ) VALUES
  ('"country"', '"zip_code"', '"street"', '"floor"');")
  .
  .
  .
::METHOD insertIntoAD
  EXPOSE conn
  USE ARG title
  stmt=conn~createStatement
  stmt~executeUpdate("INSERT INTO IMMODB.AD
  ( TITLE , CONTACT_PERSON_ID , APARTMENT_ID,
   CHRONOLOGY_ID ,  DATA_SOURCE_ID ) VALUES ('"title"',
  (SELECT MAX(CONTACT_PERSON_ID)FROM
  IMMODB.CONTACT_PERSON),
   (SELECT MAX( APARTMENT_ID)FROM IMMODB.APARTMENT),
  (SELECT MAX(CHRONOLOGY_ID)FROM IMMODB.CHRONOLOGY),
  ( SELECT MAX( DATA_SOURCE_ID )FROM  IMMODB .
  DATA_SOURCE)
  );")
```

Quellcode 15: Klasse SAPConnection

6.3 OLAP System

Die Entwicklung des OLAP Systems erfolgte mit SAP Business Application Studio. Diese Anwendung ist in der SAP HANA Cloud integriert und kann über dem Browser benutzt werden. Die folgenden OLAP Anwendungen wurden in Form von Calculation Views mit dem grafischen Modellierungswerkzeug von SAP Business Application Studio erstellt.

6.3.1 Dimensionen

Die Navigation in einem OLAP-Würfel erfolgt mithilfe der Dimensionen. In SAP HANA kann eine Dimension entweder direkt durch eine Tabelle oder durch eine Attribute View dargestellt werden. Für die Dimensionen der folgenden OLAP-Würfel wurden Attribute Views für jede Entität entwickelt, bei denen für die Auswertungen redundante Spalten der Tabellen nicht inkludiert wurden. Weiters wurden, wenn möglich Hierarchien erstellt um eine bessere Navigation im Würfel zu ermöglich. *Abbildung 13* zeigt die Hierarchieebenen der Dimension PLACE, die die Adressdaten abbildet.

Level	Column	Level Type	Order By	Sort Direction
1	COUNTRY	REGULAR	COUNTRY	Ascending
2	ZIP_CODE	REGULAR	ZIP_CODE	Ascending
3	STREET	REGULAR	STREET	Ascending

Abbildung 13: Hierarchieebenen Dimension PLACE

6.3.2 Fakten Tabellen

Eine Faktentabelle beinhaltet Kennzahlen und Schlüsselwerte für die Dimensionen. Für das OLAP System dieses Projektes wurden die Faktentabellen T_FACT_IST und T_FACT_STAT entwickelt. T_FACT_IST bildet sämtliche Mietwohnungsdaten ab und enthält deshalb die Schlüsselwerte aus den Tabellen AD und APARTMENT. Beide Faktentabellen enthalten Kennzahlen die aus der Tabelle KEY_FIGURES bezogen werden, wobei das Attribut PricePerSquareMeter eine kalkulierte Spalte aus TOTAL_AMOUNT dividiert durch LIVING_SPACE ist.

Abbildung 14 und *Abbildung 15* veranschaulichen das Zusammenführen der Tabellen für die Erstellung der beiden Faktentabellen mittels Join Operation.

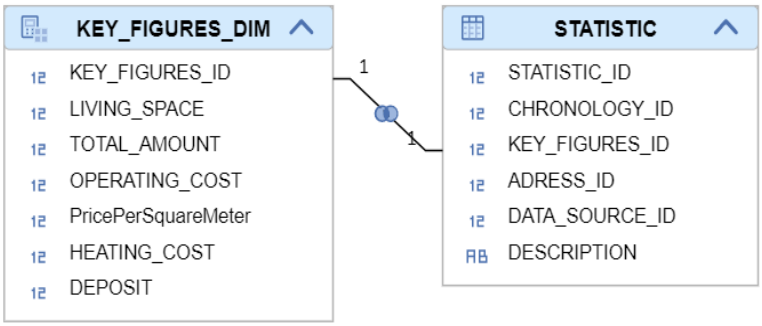

Abbildung 15: Join Operation für Faktentabelle T_FACT_STAT

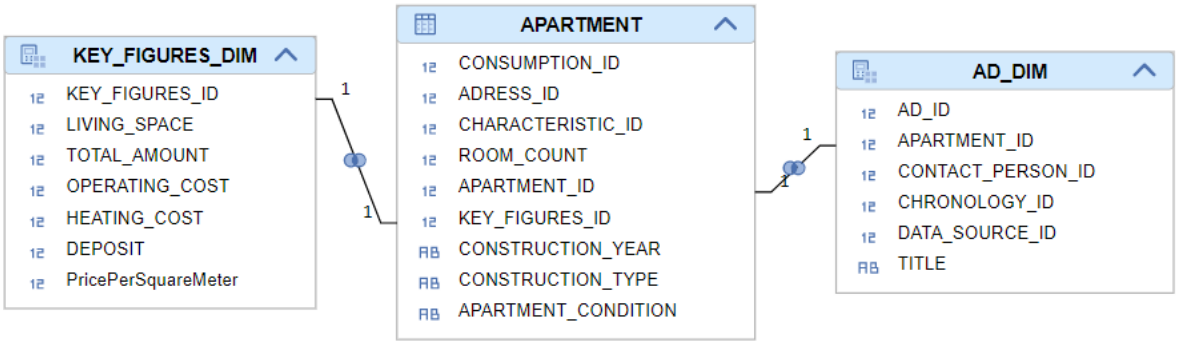

Abbildung 14: Join Operation für Faktentabelle T_FACT_IST

6.3.3 Calculation View – FACT SHEET STAT

Für die Präsentation der statistischen Datensätze im System wird die Calculation View FACT_SHEET_STAT erzeugt. Diese ergibt sich aus der Faktentabelle T_FACT_STAT und den beiden Dimensionen SOURCE und TIME. Die Verknüpfung der Faktentabelle und den Dimensionen erfolgt mittels Star Join.

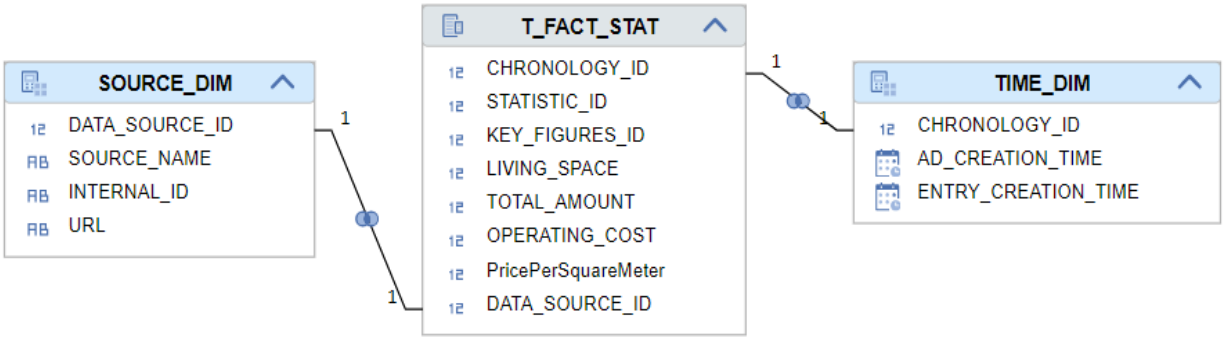

Abbildung 16: Calculation View FACT SHEET STAT – Star Join

6.3.4 Calculation View – FACT SHEET IST

Die Information über die Mietwohnungen von den Inserats Plattformen werden in der Calculation View FACT_SHEET_IST dargestellt. Vergleichbar wie bei FACT_SHEET_STAT werden die Dimensionen mittels eines Star Joins verknüpft mit der Faktentabelle verknüpft.

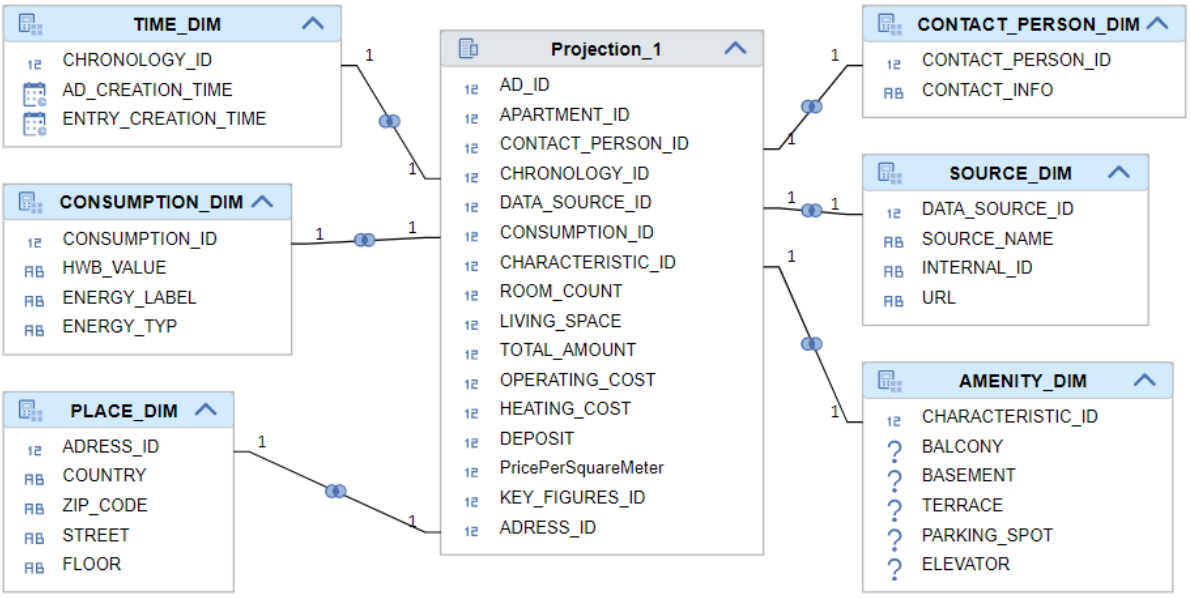

Abbildung 17: Calculation View FACT_SHEET_IST – Star Join

6.3.5 Calculation View – FACT SHEET COMP

Durch das Galaxienschema werden die Datensätze in zwei unterschiedliche Calculation Views dargestellt. Damit dennoch eine Auswertung über beide Galaxien möglich ist, wird die Calculation View FACT_SHEET_COMP erzeugt. Durch die Verbindung mittels eines Union Join werden die Datensätze miteinander kombiniert und können gemeinsam analysiert werden.

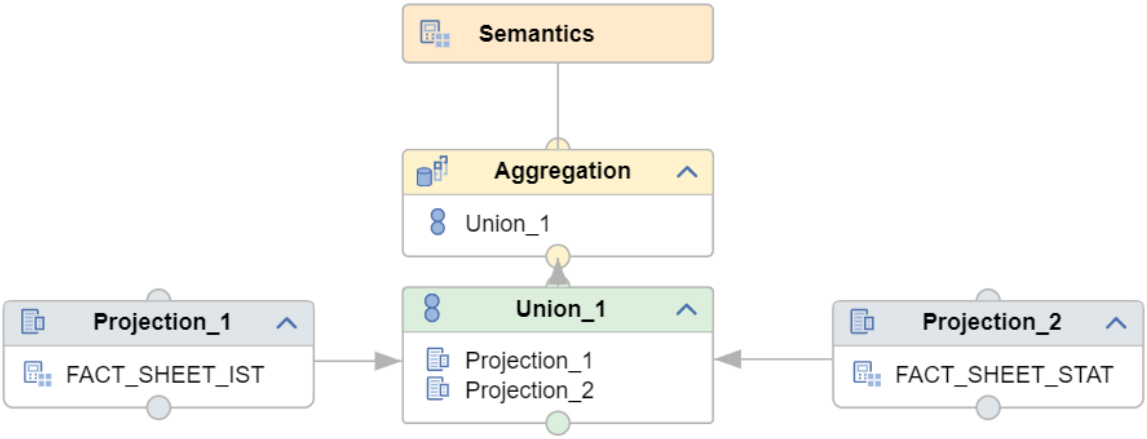

Abbildung 18: Verknüpfung der Calculation Views mit Union Join

6.4 Analytics Cloud

Die grafische Darstellung der einzelnen Calculation Views erfolgte mittels SAP Analytics Cloud. Dafür wurden die Daten der Calculation Views in Form von CSV-Dateien aus dem Datenbanksystem exportiert und in das Analyse Tool importiert. Die folgenden Beispiele veranschaulichen die Visualisierungsmethoden die Analytics Cloud bereitstellt.

Abbildung 19: Ergebnisse der SQL-Aggregationsfunktionen

Abbildung 19 zeigt unterschiedliche Anwendungsmöglichkeiten wie die SQL-Aggregationsfunktionen COUNT, AVG, MIN und MAX in einem Dashboard eingebunden werden können.

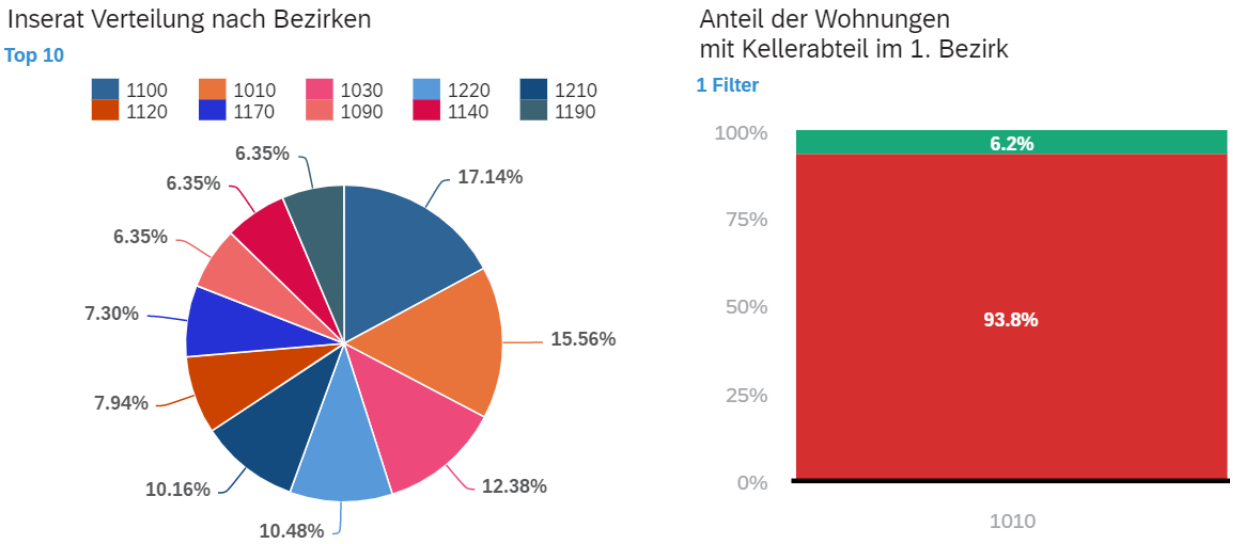

Abbildung 20: Torten und Säulendiagramm

Abbildung 20 zeigt zwei Diagrammarten, die für Analysen zur Verfügung stehen. In der linken Darstellung wurde ein Tortendiagramm verwendet um die Verteilung der Inserate auf die Bezirke darzustellen. Für die Abfrage wurden die Datensätze nach der Dimension PLACE mit der Hierarchieebene ZIP_CODE selektiert. Außerdem wurde ein Filter eingebaut der das Ergebnis

auf die 10 häufigsten Bezirke beschränkt. Die rechte Abbildung zeigt ein gestapeltes Säulendiagramm. Hierbei wurden eine SLICE Operation auf die PLACE Dimension des Würfels ausgeführt und der ZIP_CODE 1010, stellvertretend für den 1.Bezirk, festgelegt. Anschließend wurde mit einer COUNT Funktion auf die Dimension AMENITY die Anzahl der Wohnungen mit und ohne Kellerabteil gezählt.

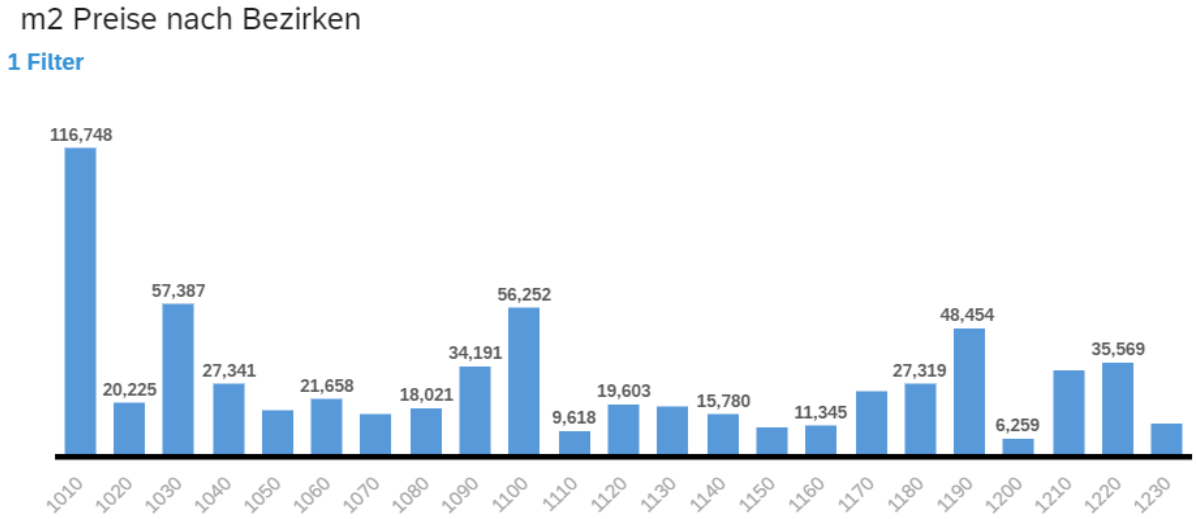

Abbildung 21: Durchschnittlicher Quadratmeterpreis nach Bezirken

Abbilddung 21 zeigt eine Auflistung der durchschnittlichen Quadratmeterpreise abgebildet in einem Säulendiagramm.

Abbildung 22: Vergleich der Ist und Statistik Quadratmeterpreise

Abbildung 22 zeigt einen Vergleich der durchschnittlichen Quadratmeterpreise der Ist und Statistik Werte. Die Daten stammen dabei aus der Calculation View FACT_SHEET_COMP, welche sowohl die Kennzahlen der Statistik Werte als auch der Ist Werte enthält. Geordnet wurde das Ergebnis dabei nach der Dimension TIME.

Conclusio

Anhand der erarbeiteten Konzepte aus der Literatur wurde eine Demo Anwendung für ein DWH System in einer SAP HANA Datenbankumgebung realisiert. Dabei wurde das OLTP und OLAP System in ein Datenbanksystem kombiniert, wobei die Produktivdatenbank mittels SQL Statements erzeugt und die verschiedenen OLAP Würfel in Form von Calculation Views dargestellt wurden. Das gesamte DWH System basiert auf dem SAP HANA Datenbanksystem und es wurden außer für die grafische Darstellung der Daten keine externen Ressourcen verwendet. Die Darstellung wurde in der SAP Analytics Cloud realisiert, wobei durch die Verwendung der Trial Version nicht alle Funktionalitäten verfügbar waren und deshalb eine Erstellung von Analysen und Reports mit Echtzeitverbindung nicht möglich war. Die Integration der Daten, die durch Web Scraping Skripte gesammelt wurden, erfolgte über den SAP JDBC Client. Durch die Verwendung der JDBC Datenbankschnittstelle ist eine Kompatibilität des DWH Systems mit allen Java Anwendungen gewährleistet. Bezüglich der Performance des Systems können anhand dieser Arbeit keine Aussagen getroffen werden. Die Rechenleistung der SAP HANA Cloud in der Trial Version wird für alle Nutzer geteilt, weshalb nicht ersichtlich ist wie viele Arbeitsspeicher jeder Datenbankinstanz zur Verfügung steht. Abschließend kann gesagt werden, dass das SAP HANA Datenbanksystem die Problemstellung erfüllt und sich somit als DWH für Analysen und Reports eignet.

Literaturverzeichnis

Anane Adusei, Dickson/Rötting, Ingo/Yamada, Stefan (2019). *SAP HANA - Datenmodellierung* (1. Auflage.). Bonn: Rheinwerk Verlag. https://permalink.obvsg.at/wuw/AC15293112

Bauer, Andreas (2013). *Data-Warehouse-Systeme: Architektur, Entwicklung, Anwendung* (4., überarb. und erw. Aufl..). Heidelberg: dpunkt-Verl. https://permalink.obvsg.at/wuw/AC10886959

Dhanda, P., & Sharma, N. (2016). Extract transform load data with ETL tools. International Journal of Advanced Research in Computer Science, 7(3) Retrieved from https://search.proquest.com/scholarly-journals/extract-transform-load-data-with-etl-tools/docview/1813053993/se-2?accountid=29104

Devlin, B. (2018). *Thirty years of data warehousing. Business Intelligence Journal*, 23, 12-24. Retrieved from https://search.proquest.com/magazines/thirty-years-data-warehousing/docview/2055020688/se-2?accountid=29104

Fasel, Daniel/Meier, Andreas (2016). *Big Data: Grundlagen, Systeme und Nutzungspotenziale*. Wiesbaden: Springer Vieweg. https://permalink.obvsg.at/wuw/AC13270822

Flatscher, Rony (2016). *AutoJava-BSF4ooRexx-01.* Abgerufen März 31, 2021, von http://wi.wu.ac.at:8002/rgf/wu/lehre/autojava/material/foils/AutoJava-BSF4ooRexx-01.pdf

Geisler, Frank (2014*). Datenbanken: Grundlagen und Design* (5., aktualisierte und erw. Aufl..). Heidelberg: mitp. https://permalink.obvsg.at/wuw/AC11761212

Goulão, M., Amaral, V., & Mernik, M. (2016). Quality in model-driven engineering: A tertiary study. Software Quality Journal, 24(3), 601-633. doi:http://dx.doi.org/10.1007/s11219-016-9324-8

Hahne, Michael (2014). *Modellierung von Business-Intelligence-Systemen: Leitfaden für erfolgreiche Projekte auf Basis flexibler Data-Warehouse-Architekturen* (1. Aufl..). Heidelberg: dpunkt-Verl. https://permalink.obvsg.at/wuw/AC12049669

Hansen, Hans Robert/Mendling, Jan/Neumann, Gustaf (2015). *Wirtschaftsinformatik: Grundlagen und Anwendungen* (11., völlig neu bearb. Aufl..). Berlin: de Gruyter. http://dx.doi.org/10.1515/9783110335293

Homayouni, H. (2018). *An approach for testing the extract-transform-load process in data warehouse systems* (Order No. 10689298). Available from Publicly Available Content Database. (2055766176). Retrieved from https://search.proquest.com/dissertations-theses/approach-testing-extract-transform-load-process/docview/2055766176/se-2?accountid=29104

JSON (2021). *Introducing JSON*. Abgerufen März 31, 2021, von https://www.json.org/json-en.html

JSoup (2021). *Jsoup: JAVA HTML Parser (January 29, 2021)*. Abgerufen März 31, 2021, von https://jsoup.org/

JSoup (2021). *Use selector-syntax to find elements*. Abgerufen März 31, 2021, von https://jsoup.org/cookbook/extracting-data/selector-syntax

Kimball, R. (2002). *Two powerful ideas. Intelligent Enterprise*, 5(15), 30-33. Retrieved from https://search.proquest.com/trade-journals/two-powerful-ideas/docview/200572396/se-2?accountid=29104

Köppen, Veit/Saake, Gunter/Sattler, Kai-Uwe (2012*). Data Warehouse: Technologien* (1. Aufl..). Heidelberg: mitp. https://permalink.obvsg.at/wuw/AC09592613

Object Management Group (2021). *GRAPICAL NOTATIONS FOR BUSINESS PROCESSES*. Abgerufen März 31, 2021, von https://www.omg.org/bpmn/index.htm

Preuss, Peter (2017). *In-Memory-Datenbank SAP HANA*. Wiesbaden: Springer Fachmedien Wiesbaden Imprint: Springer Gabler. http://dx.doi.org/10.1007/978-3-658-18603-6

Rauter Romana Ass-Prof Dr, Lerch, A., Lederer-Hutsteiner, T. M., Klinger, S. D., Mayr, A. M., Dr, Gutounig Robert, M. D., & Pammer-Schindler Viktoria Assoc-Prof Dr. (2021). *Digital und/oder analog? zusammenarbeit am arbeitsplatz aus der perspektive österreichischer unternehmen. Wirtschaftsinformatik & Management*, 13(1), 48-58. doi:http://dx.doi.org/10.1365/s35764-020-00307-6

Rexx Language Association (2021). *About Open Object Rexx*. Abgerufen März 31, 2021, von https://www.oorexx.org/about.html

SAP (2021a). *SAP Global Corporate Affairs (January 29, 2021)*. Abgerufen März 31, 2021, von https://www.sap.com/documents/2017/04/4666ecdd-b67c-0010-82c7-eda71af511fa.html

SAP (2021b). *What is SAP HANA?*. Abgerufen März 31, 2021, von https://www.sap.com/products/hana/what-is-sap-hana.html#overview

SAP (2021c). *What is SAP Business Application Studio?*. Abgerufen März 31, 2021, von https://help.sap.com/viewer/9d1db9835307451daa8c930fbd9ab264/Cloud/en-US/8f46c6e6f86641cc900871c903761fd4.html

SAP (2021d). *Connect to SAP HANA via JDBC?*. Abgerufen März 31, 2021, von

https://help.sap.com/viewer/f1b440ded6144a54ada97ff95dac7adf/2.8/en-US/ff15928cf5594d78b841fbbe649f04b4.html

SAP (2021e). Using Attribute Views. Abgerufen März 31, 2021, von https://help.sap.com/viewer/fc5ace7a367c434190a8047881f92ed8/2.0.03/en-US/6136aec486fa4dc380f78adbb9de47df.html

SAP (2021f). Using Analytic Views. Abgerufen März 31, 2021, von https://help.sap.com/viewer/fc5ace7a367c434190a8047881f92ed8/2.0.03/en-US/46816a454ed947e8a163e86b29068377.html

SAP (2021g). Using Calculation Views. Abgerufen März 31, 2021, von https://help.sap.com/viewer/fc5ace7a367c434190a8047881f92ed8/2.0.03/en-US/d60ad1f0bb571014af49c9db1740d68c.html

Anhang

CREATE TABLE – SQL Statement

```
CREATE COLUMN TABLE "IMMODB"."CONTACT_PERSON" (contact_person_id INT
NOT NULL GENERATED BY DEFAULT AS IDENTITY UNIQUE, contact_info
VARCHAR(100) , phone_number VARCHAR(100) ,email VARCHAR(100));

CREATE COLUMN TABLE "IMMODB"."KEY_FIGURES" (key_figures_id INT NOT
NULL GENERATED BY DEFAULT AS IDENTITY UNIQUE, living_space DOUBLE
NULL,  total_amount DOUBLE NULL,  operating_cost DOUBLE NULL,
heating_cost DOUBLE NULL,  deposit DOUBLE NULL,  commission
NVARCHAR(200) NULL);

CREATE COLUMN TABLE "IMMODB"."ADRESS" (adress_id INT NOT NULL
GENERATED BY DEFAULT AS IDENTITY UNIQUE,  country VARCHAR(45) NULL,
zip_code VARCHAR(45) NULL,  street VARCHAR(100) NULL,  floor
VARCHAR(45) NULL);

CREATE COLUMN TABLE "IMMODB"."CONSUMPTION" (consumption_id INT NOT
NULL GENERATED BY DEFAULT AS IDENTITY UNIQUE,  hwb_value
VARCHAR(100) NULL,  energy_label VARCHAR(45) NULL,  energy_typ
VARCHAR(100) NULL);

CREATE COLUMN TABLE "IMMODB"."CHARACTERISTIC" ( characteristic_id
INT  NOT NULL GENERATED BY DEFAULT AS IDENTITY UNIQUE,  balcony
BOOLEAN NULL,  basement BOOLEAN NULL,  terrace BOOLEAN NULL,
parking_spot BOOLEAN NULL,  elevator BOOLEAN NULL);

CREATE COLUMN TABLE "IMMODB"."APARTMENT" (apartment_id INT NOT NULL
GENERATED BY DEFAULT AS IDENTITY UNIQUE, key_figures_id INT NULL,
adress_id INT NULL,  consumption_id INT NULL,  characteristic_id INT
NULL,  room_count DOUBLE NULL, construction_year VARCHAR(100) NULL,
construction_type VARCHAR(100) NULL,  apartment_condition
VARCHAR(100) NULL);

ALTER TABLE "IMMODB"."APARTMENT" ADD FOREIGN KEY
(adress_id) REFERENCES immodb.adress(adress_id)
ON DELETE CASCADE;

ALTER TABLE "IMMODB"."APARTMENT" ADD FOREIGN KEY (key_figures_id)
REFERENCES immodb.key_figures(key_figures_id)
ON DELETE CASCADE;

ALTER TABLE "IMMODB"."APARTMENT" ADD FOREIGN KEY (consumption_id)
REFERENCES immodb.consumption(consumption_id)
ON DELETE CASCADE;

ALTER TABLE "IMMODB"."APARTMENT" ADD FOREIGN KEY (characteristic_id)
REFERENCES immodb.characteristic(characteristic_id)
ON DELETE CASCADE;

CREATE COLUMN TABLE "IMMODB"."CHRONOLOGY" (chronology_id INT NULL
GENERATED BY DEFAULT AS IDENTITY UNIQUE, ad_creation_time SECONDDATE
```

```
NULL, entry_creation_time SECONDDATE  NULL,  available_from
VARCHAR(100) NULL,  rental_period VARCHAR(100) NULL);
CREATE COLUMN TABLE "IMMODB"."DATA_SOURCE" (  data_source_id INT NOT
NULL GENERATED BY DEFAULT AS IDENTITY UNIQUE,  source_name
VARCHAR(45) NULL,  internal_id VARCHAR(45) NULL,  url VARCHAR(200)
NULL);

CREATE COLUMN TABLE "IMMODB"."AD" ( ad_id INT NOT NULL GENERATED BY
DEFAULT AS IDENTITY UNIQUE,  title NVARCHAR(200) NULL,
contact_person_id INT NULL,  apartment_id INT NULL,  chronology_id
INT NULL,  data_source_id INT NULL);

ALTER TABLE "IMMODB"."AD" ADD FOREIGN KEY(contact_person_id)
REFERENCES immodb.contact_person(contact_person_id)
ON DELETE CASCADE;

ALTER TABLE "IMMODB"."AD" ADD FOREIGN KEY (apartment_id)
REFERENCES immodb.apartment(apartment_id)
ON DELETE CASCADE;

ALTER TABLE "IMMODB"."AD" ADD FOREIGN KEY (chronology_id)
REFERENCES immodb.chronology(chronology_id)
ON DELETE CASCADE;

ALTER TABLE "IMMODB"."AD" ADD FOREIGN KEY (data_source_id)
REFERENCES immodb.data_source(data_source_id)
ON DELETE CASCADE;

CREATE COLUMN TABLE "IMMODB"."STATISTIC" (statistic_id INT NOT NULL
GENERATED BY DEFAULT AS IDENTITY UNIQUE, description VARCHAR(200),
adress_id INT NULL,  key_figures_id INT NULL,  chronology_id INT
NULL,  data_source_id INT NULL);

ALTER TABLE "IMMODB"."STATISTIC" ADD FOREIGN KEY
(adress_id) REFERENCES immodb.adress(adress_id)
ON DELETE CASCADE;

ALTER TABLE "IMMODB"."STATISTIC" ADD FOREIGN KEY (key_figures_id)
REFERENCES immodb.key_figures(key_figures_id)
ON DELETE CASCADE;

ALTER TABLE "IMMODB"."STATISTIC" ADD FOREIGN KEY (chronology_id)
REFERENCES immodb.chronology(chronology_id)
ON DELETE CASCADE;

ALTER TABLE "IMMODB"."STATISTIC" ADD FOREIGN KEY
(data_source_id)REFERENCES immodb.data_source(data_source_id)
ON DELETE CASCADE;
```

CREATE USER – SQL Statement

```
CREATE USER IMMOUSR PASSWORD "Immo2021" SET USERGROUP DEFAULT;
ALTER USER IMMOUSR DISABLE PASSWORD LIFETIME;

CREATE ROLE CCROLE;
GRANT SELECT, SELECT METADATA ON SCHEMA "IMMODB" TO CCROLE WITH
GRANT OPTION;
GRANT CCROLE to IMMOUSR WITH ADMIN OPTION;
```

INSERT INTO – SQL Statement

```
INSERT INTO "IMMODB"."DATA_SOURCE"("SOURCE_NAME", "URL")
VALUES('Name',  'https://www.test.at');

INSERT INTO IMMODB.CHRONOLOGY(AD_CREATION_TIME, ENTRY_CREATION_TIME)
VALUES('0001-01-01T01:01:01', CURRENT_TIMESTAMP);

INSERT INTO "IMMODB"."CONTACT_PERSON"("CONTACT_INFO",
"PHONE_NUMBER", "EMAIL") VALUES('Max Mustermann', '+43', '@');

INSERT INTO "IMMODB"."ADRESS"("COUNTRY", "ZIP_CODE", "STREET",
"FLOOR") VALUES('Country', 'ZIP Code', 'Street', 'Floor');

INSERT INTO "IMMODB"."CHARACTERISTIC"("BALCONY", "BASEMENT",
"TERRACE", "PARKING_SPOT", "ELEVATOR") VALUES(TRUE, TRUE, TRUE,
TRUE, TRUE);

INSERT INTO "IMMODB"."KEY_FIGURES"("LIVING_SPACE","TOTAL_AMOUNT",
"OPERATING_COST") VALUES(99, 99, 99);

INSERT INTO "IMMODB"."CONSUMPTION"("HWB_VALUE", "ENERGY_LABEL",
"ENERGY_TYP") VALUES(99, 'Label', 'Typ');

INSERT INTO "IMMODB"."APARTMENT"("KEY_FIGURES_ID", "ADRESS_ID",
"CONSUMPTION_ID", "CHARACTERISTIC_ID", "ROOM_COUNT",
"CONSTRUCTION_YEAR", "CONSTRUCTION_TYPE", "APARTMENT_CONDITION")
VALUES(  ( SELECT MAX("KEY_FIGURES_ID")   FROM
"IMMODB"."KEY_FIGURES"), ( SELECT MAX("ADRESS_ID")FROM
"IMMODB"."ADRESS"), ( SELECT MAX("CONSUMPTION_ID")FROM
"IMMODB"."CONSUMPTION"),( SELECT MAX("CONSUMPTION_ID")FROM
"IMMODB"."CONSUMPTION"), 99, 2021, 'Type', 'Condition');

INSERT INTO "IMMODB"."AD"("TITLE", "CONTACT_PERSON_ID",
"APARTMENT_ID", "CHRONOLOGY_ID", "DATA_SOURCE_ID") VALUES(
    'Titel', ( SELECT MAX("CONTACT_PERSON_ID")FROM
"IMMODB"."CONTACT_PERSON" ), ( SELECT MAX("APARTMENT_ID")FROM
"IMMODB"."APARTMENT" ), ( SELECT MAX("CHRONOLOGY_ID")FROM
"IMMODB"."CHRONOLOGY" ), ( SELECT MAX("DATA_SOURCE_ID")FROM
"IMMODB"."DATA_SOURCE" ));

INSERT INTO IMMODB.STATISTIC(DESCRIPTION, ADRESS_ID, KEY_FIGURES_ID,
CHRONOLOGY_ID, DATA_SOURCE_ID) VALUES(   'Description', ( SELECT
MAX(ADRESS_ID)FROM IMMODB.ADRESS ),       ( SELECT
MAX(KEY_FIGURES_ID)FROM IMMODB.KEY_FIGURES ),    ( SELECT
MAX(CHRONOLOGY_ID)FROM IMMODB.CHRONOLOGY ),( SELECT
MAX(DATA_SOURCE_ID)FROM IMMODB.DATA_SOURCE ));
```

startWebScraper_FMH.rexx

```
-------------------------------------------------------------------------
test_mode = .true -- set false to scrape the total number of ad
entries

homepage = ("https://www.findmyhome.at")
adlist_path = ("/immo/wohnung-mieten/wien")

CALL startWebscraping_FMH homepage, adlist_path, test_mode
entryList = result

System = bsf.import("java.lang.System")
current_folder = System~getProperty("user.dir")

path = current_folder"\output\temp_ist.json"
CALL writeJSONtoFile entryList~toString, path

-------------------------------------------------------------------------
::REQUIRES "BSF.CLS"
-------------------------------------------------------------------------
--Routine to write the JSON Array into a File
::ROUTINE writeJSONtoFile
  PARSE ARG input, path

    fileWriter = bsf.import("java.io.FileWriter")
    file = fileWriter~NEW(path,.false)
    file~write(input)
    file~flush
    file~close

-------------------------------------------------------------------------
--Routine to start the ETL-Process for the website www.findmyhome.at
::ROUTINE startWebscraping_FMH
    PARSE ARG homepage, adlist_path, test_mode
        --import Jsoup Ressources
        Jsoup = bsf.import("org.jsoup.Jsoup")

        entryList = .bsf~new("org.json.JSONArray") --create JSON
Array

        SAY "Start Webscraping:" homepage""adlist_path

        mainpage = Jsoup~connect(homepage""adlist_path)~get()

        IF test_mode THEN
            DO
                adCount = 200 -- only scrape the first 200 ad
entries
            END
        ELSE
            DO
                --get the total number of ad entries and set the
adCount
```

```
                adCount = mainpage~getElementsByClass("hidden-md
hidden-lg col-xs-12")~first()~getElementsByTag("b")~first()
                adCount  = adCount~ownText()~delWord(2,2)
            END


        DO i = 0 TO adCount BY 20
            mainpage =
Jsoup~connect(homepage""adlist_path"?entry="i)~get()
            adlist = mainpage~getElementsByClass("obj_list")

            DO j = 0 TO (adlist~size - 1)
                adref =
adlist~get(j)~getElementsByTag("a")~first()~attr("href")
                IF (adref \= "") THEN
                    DO
                        link = homepage""adref
                        CALL scrapeAdEntry_FMH link -- returns JSON
Object
                        entryList~put(result) --add JSON Object to
Array
                    END
            END
        END

        RETURN entryList

------------------------------------------------------------------------
--Routine for the Extraction and Transformation of a single
Findmyhome Entry
::ROUTINE scrapeAdEntry_FMH
    PARSE ARG url

    --Load the HTML Document of the Ad Entry
    Jsoup = bsf.import("org.jsoup.Jsoup")
    adpage = Jsoup~connect(url)~get()

    --Extraction of the nesseccary values from the HTML code
    --Transformation of the Data into Objects

    --Extract & Transform for  TABLE DATA_SOURCE
    data_source = .data_source~NEW()
    --COLUMN <NVARCHAR> SOURCE_NAME
    data_source~name="FindMyHome"
    --COLUMN <NVARCHAR> URL (value already collected)
    data_source~url=url
    --COLUMN <NVARCHAR> INTERNAL_ID
    temp = adpage~getElementById("khandyanzeigenMobile")
    IF ( temp \= .nil & value \= "") THEN
        DO

data_source~internal_id=temp~getElementsByTag("b")~last()~ownText()
        END
    ELSE
        DO
```

```
            data_source~internal_id=""
        END

    --Extract & Transform for TABLE CHRONOLOGY
    chronology = .chronology~NEW()
    --COLUMN AD_CREATION_TIME <SECONDDATE>
    temp =
adpage~getElementsByAttributeValue("name","openimmo_updated_at")~fir
st()
    IF (temp \= .nil & temp \= "") THEN
        DO
            chronology~creation_time=temp~attr("content")
        END
    ELSE
        DO
            chronology~creation_time="0001-01-01 00:00:00.0000000"

        END
    --COLUMN AVAILABLE_FROM <DATE>
    temp = adpage~select("tr:contains(Beziehbar ab) td")~last()
    IF ( temp \= .nil & temp \= "") THEN
        DO
            chronology~available_from = temp~ownText()
        END
    ELSE
        DO
            chronology~available_from= ""
        END
    --COLUMN RENTAL_PERIOD <NVARCHAR>
    temp = adpage~select("tr:contains(Mietdauer) td")~last()
    IF ( temp \= .nil & temp \= "") THEN
        DO
            chronology~rental_period = temp~ownText()
        END
    ELSE
        DO
            chronology~rental_period= "nicht befristet"
        END

    --Extract & Transform for TABLE CONTACT_PERSON
    contact_person = .contact_person~NEW()
    --COLUMN CONTACT_INFO <NVARCHAR>
    temp =
adpage~getElementById("telKontakt")~getElementsByTag("b")~first()
    IF ( temp \= .nil & temp \= "") THEN
        DO
            IF (temp \= "Telefon Allgemein:")THEN
            DO
                contact_person~contact_info = temp~ownText()
            END
        END
    ELSE
        DO
            contact_person~contact_info= ""
        END
```

```
    --COLUMN PHONE_NUMBER <NVARCHAR>
    temp = adpage~getElementById("khandyanzeigenMobile")
    IF ( temp \= .nil & temp \= "") THEN
        DO
            temp =
adpage~getElementById("khandyanzeigenMobile")~getElementsByTag("b")~
first()
            IF (temp \= "Telefon Allgemein:")THEN
                DO
                    temp =
adpage~getElementById("khandyanzeigenMobile")~html()
                    temp = temp~delStr(39)
                    temp = temp~delWord(1,2)
                    contact_person~phone_number = temp
                END
        END
    ELSE
        DO
            contact_person~phone_number= ""
        END
    --COLUMN EMAIL <NVARCHAR>  not included on this page
    contact_person~email = ""

    --Extract & Transform for TABLE ADRESS
    adress = .adress~NEW()
    --COLUMN <NVARCHAR> COUNTRY (only entries from Vienna!)
    adress~country= "AUT"
    --COLUMN <NVARCHAR> ZIP_CODE
    temp = adpage~getElementsByClass("immo_header_value")~first()
    IF (temp \= .nil & temp \= "") THEN
        DO
            adress~zip_code=temp~ownText()
        END
    ELSE
        DO
            adress~zip_code=""
        END
    --COLUMN STREET <NVARCHAR> (no structured information about
street)
    adress~street= ""
    --COLUMN FLOOR <NVARCHAR>
    temp = adpage~select("tr:contains(Stock) td")~last()
    IF (temp \= .nil & temp \= "") THEN
        DO
            adress~floor=temp~ownText()
        END
    ELSE
        DO
            adress~floor=""
        END

    --Extract & Transform for TABLE CHARACTERISTIC
    characteristic = .characteristic~NEW()
    --COLUMN BALCONY <BOOLEAN>
    temp = adpage~select("tr:contains(Balkon) td")~last()
```

```
    IF (temp \= .nil & temp \= "") THEN
        DO
            characteristic~balcony= "TRUE"
        END
    ELSE
        DO
            characteristic~balcony= "FALSE"
        END
    --COLUMN BASEMENT <BOOLEAN>
    temp = adpage~select("tr:contains(Keller) td")~last()
    IF (temp \= .nil & temp \= "") THEN
        DO
            characteristic~basement="TRUE"
        END
    ELSE
        DO
            characteristic~basement= "FALSE"
        END
    --COLUMN TERRACE <BOOLEAN>
    temp = adpage~select("tr:contains(Terrassen) td")~last()
    IF (temp \= .nil & temp \= "") THEN
        DO
            characteristic~terrace= "TRUE"
        END
    ELSE
        DO
            characteristic~terrace="FALSE"
        END
    -- PARKING_SPOT <BOOLEAN> (no information about parking spaces)
    characteristic~parking_spot= "NULL"
    --COLUMN ELEVATROR <BOOLEAN>
    temp = adpage~select("tr:contains(Lift) td")~last()
    IF (temp \= .nil & temp \= "") THEN
        DO
            characteristic~elavator= "TRUE"
        END
    ELSE
        DO
            characteristic~elavator= "FALSE"
        END

    --Extract & Transform for TABLE KEY_FIGURES
    key_figures = .key_figures~NEW()
    --COLUMN <DOUBLE> TOTAL_AMOUNT
    temp = adpage~select("tr:contains(Monatliche Gesamtkosten:)
td")~last()
    IF (temp \= .nil & temp \= "") THEN
        DO
            temp = temp~ownText()~delWord(1,1)
            temp = temp~changeStr(".","")
            temp = temp~changeStr(",",".")
            key_figures~total_amount= temp
        END
    ELSE
        DO
```

```
            key_figures~total_amount= "NULL"
        END
    SAY url
    --COLUMN LIVING_SPACE <INT>
        temp = adpage~getElementsByClass("immo_header_value")
        IF (temp \= .nil & temp \= "" & temp~size > 2) THEN
            DO
                temp = adpage~getElementsByClass
("immo_header_value")~get(2)
                temp = temp~ownText()~delWord(2)
                temp = temp~changeStr(".","")
                temp = temp~changeStr(",",".")
                key_figures~living_space= temp
            END
        ELSE
            DO
                key_figures~living_space= "NULL"
            END
    --COLUMN <DOUBLE> OPERATING_COST (no information about heating
cost)
    temp = adpage~select("tr:contains(Betriebskosten (inkl. MwSt):)
td")~last()
        IF (temp \= .nil & temp \= "") THEN
            DO
                temp = temp~ownText()~delWord(1,1)
                temp = temp~changeStr(".","")
                temp = temp~changeStr(",",".")
                key_figures~operating_cost= temp
            END
        ELSE
            DO
                key_figures~operating_cost= "NULL"
            END
    --COLUMN <DOUBLE> HEATING_COST (no information about heating
cost)
    key_figures~heating_cost="NULL"
    --COLUMN <DOUBLE> DEPOSIT
    temp = adpage~select("tr:contains(Kaution:) td")~last()
        IF (temp \= .nil & temp \= "") THEN
            DO
                temp = temp~ownText()~delWord(1,1)
                temp = temp~changeStr(".","")
                temp = temp~changeStr(",",".")
                key_figures~deposit= temp
            END
        ELSE
            DO
                key_figures~deposit= "NULL"
            END
    --COLUMN  <NVARCHAR> COMMISSION
    temp = adpage~select("tr:contains(Provision) td")~first()
        IF (temp \= .nil & temp \= "") THEN
            DO
                temp = temp~ownText()~delWord(1,1)
                key_figures~commission= temp
```

```
            END
        ELSE
            DO
                key_figures~commission= ""
            END

    --Extract & Transform for TABLE CONSUMPTION
    consumption = .consumption~NEW()
    --COLUMN HWB_VALUE <INT> AND ENERGY_LABEL <NVARCHAR>
    temp = adpage~getElementById("energieausweis")
    IF (temp \= .nil & temp \= "") THEN
        DO
            temp = adpage~getElementById("energieausweis")~ownText
            parse var temp temp1 "kWh/m2/Jahr: " temp2 " Klasse"
temp3 ": " temp4
            consumption~hwb_value= temp2~delWord(2)
            consumption~energy_label= SUBSTR(temp4,1,1)
        END
    ELSE
        DO
            consumption~hwb_value= ""
            consumption~energy_label= ""
        END
    --COLUMN ENERGY_TYPE <NVARCHAR>
    temp = adpage~select("tr:contains(Heizung) td")~last()
    IF (temp \= .nil & temp \= "") THEN
        DO
            consumption~energy_type= temp~ownText()
        END
    ELSE
        DO
            consumption~energy_type= ""
        END

    --Extract & Transform for TABLE APARTMENT
    apartment = .apartment~NEW()
    --COLUMN ROOM_COUNT <INT>
    temp = adpage~select("tr:contains(Zimmer) td")~last()
    IF (temp \= .nil & temp \= "") THEN
        DO
            temp = temp~ownText()
            apartment~room_count= temp

        END
    ELSE
        DO
            apartment~room_count= "NULL"
        END
    --COLUMN CONSTRUCTION_YEAR <NVARCHAR>
    temp = adpage~select("tr:contains(Baujahr) td")~last()
    IF (temp \= .nil & temp \= "") THEN
        DO
            apartment~construction_year= temp~ownText()
        END
    ELSE
```

```
        DO
            apartment~construction_year= ""
        END
    --COLUMN CONSTRUCTION_TYPE <NVARCHAR>
    temp = adpage~select("tr:contains(Alt- oder Neubau) td")~last()
    IF (temp \= .nil & temp \= "") THEN
        DO
            apartment~construction_type= temp~ownText()
        END
    ELSE
        DO
            apartment~construction_type= ""
        END
    --COLUMN CONDITION <NVARCHAR>
    temp = adpage~select("tr:contains(Zustand) td")~last()
    IF (temp \= .nil & temp \= "") THEN
        DO
            apartment~condition= temp~ownText()
        END
    ELSE
        DO
            apartment~condition= ""
        END

    --Extract & Transform for TABLE AD
    ad = .ad~NEW()
    --COLUMN <NVARCHAR> TITLE
    temp = adpage~getElementsByClass("col-xs-12 col-sm-12 col-md-12
col-lg-12 margin-top-10")~first()~getElementsByTag("h1")~first()
    IF (temp \= .nil & temp \= "") THEN
        DO
            ad~title= temp~ownText()
        END
    ELSE
        DO
            ad~title= ""
        END
    --COLUMN <NVARCHAR> DESCRIPTION
    ad~description = ""

    --create a JSON Object for the Ad Entry
    jsonAdEntry = .bsf~new("org.json.JSONObject")
    -- Add the Data to the JSONObject
    jsonAdEntry~~put("adress", adress~exportJSON)
    jsonAdEntry~~put("data_source", data_source~exportJSON)
    jsonAdEntry~~put("chronology", chronology~exportJSON)
    jsonAdEntry~~put("characteristic", characteristic~exportJSON)
    jsonAdEntry~~put("key_figures", key_figures~exportJSON)
    jsonAdEntry~~put("apartment", apartment~exportJSON)
    jsonAdEntry~~put("consumption", consumption~exportJSON)
    jsonAdEntry~~put("contact_person", contact_person~exportJSON)
    jsonAdEntry~~put("ad", ad~exportJSON)


    SAY jsonAdEntry~toString -- print JSON Objekt in console
```

```
    RETURN jsonAdEntry

------------------------------------------------------------------------
::CLASS data_source
    ::METHOD name ATTRIBUTE
    ::METHOD internal_id ATTRIBUTE
    ::METHOD url ATTRIBUTE
    ::METHOD exportCSV
        say self~name";"self~internal_id";"self~"url"
    ::METHOD exportJSON
        jsonObject = bsf.import("org.json.JSONObject")
        jsonArray = bsf.import("org.json.JSONArray")
        jsonData = jsonObject~NEW
        jsonData~~put("name", self~name)~~put("internal_id",
self~internal_id)~~put("url", self~url)
        RETURN jsonData

------------------------------------------------------------------------
::CLASS chronology
    ::METHOD  creation_time ATTRIBUTE
    ::METHOD  available_from ATTRIBUTE
    ::METHOD  rental_period ATTRIBUTE
    ::METHOD exportCSV
        say
self~creation_time";"self~available_from";"self~rental_period
    ::METHOD exportJSON
            jsonObject = bsf.import("org.json.JSONObject")
            jsonArray = bsf.import("org.json.JSONArray")
            jsonData = jsonObject~NEW
            jsonData~~put("creation_time",
self~creation_time)~~put("available_from",
self~available_from)~~put("rental_period", self~rental_period)
            RETURN jsonData

------------------------------------------------------------------------
::CLASS contact_person
    ::METHOD contact_info ATTRIBUTE
    ::METHOD phone_number ATTRIBUTE
    ::METHOD email ATTRIBUTE
    ::METHOD exportCSV
            say self~contact_info";"self~phone_number";"self~email
    ::METHOD exportJSON
            jsonObject = bsf.import("org.json.JSONObject")
            jsonArray = bsf.import("org.json.JSONArray")
            jsonData = jsonObject~NEW
            jsonData~~put("contact_info",
self~contact_info)~~put("phone_number",
self~phone_number)~~put("email", self~email)
            RETURN jsonData

------------------------------------------------------------------------
::CLASS adress
    ::METHOD  country ATTRIBUTE
    ::METHOD  zip_code ATTRIBUTE
```

```
    ::METHOD  street ATTRIBUTE
    ::METHOD  floor ATTRIBUTE
    ::METHOD exportCSV
            say
self~country";"self~zip_code";"self~street";"self~floor
    ::METHOD exportJSON
            jsonObject = bsf.import("org.json.JSONObject")
            jsonArray = bsf.import("org.json.JSONArray")
            jsonData = jsonObject~NEW
            jsonData~~put("country", self~country)~~put("zip_code",
self~zip_code)~~put("street", self~street)~~put("floor", self~floor)
            RETURN jsonData

------------------------------------------------------------------------
::CLASS characteristic
    ::METHOD  balcony ATTRIBUTE
    ::METHOD  basement ATTRIBUTE
    ::METHOD  terrace ATTRIBUTE
    ::METHOD  parking_spot ATTRIBUTE
    ::METHOD  elavator ATTRIBUTE
    ::METHOD exportCSV
             say
self~balcony";"self~basement";"self~terrace";"self~parking_spot";"se
lf~elavator
    ::METHOD exportJSON
            jsonObject = bsf.import("org.json.JSONObject")
            jsonArray = bsf.import("org.json.JSONArray")
            jsonData = jsonObject~NEW
            jsonData~~put("balcony", self~balcony)~~put("basement",
self~basement)~~put("terrace", self~terrace)~~put("parking_spot",
self~parking_spot)~~put("elavator", self~elavator)
            RETURN jsonData

------------------------------------------------------------------------
::CLASS key_figures
    ::METHOD  living_space ATTRIBUTE
    ::METHOD  total_amount ATTRIBUTE
    ::METHOD  operating_cost ATTRIBUTE
    ::METHOD  heating_cost ATTRIBUTE
    ::METHOD  deposit ATTRIBUTE
    ::METHOD  commission ATTRIBUTE
    ::METHOD exportCSV
            say
self~living_space";"self~total_amount";"self~operating_cost";"self~h
eating_cost";"self~deposit";"self~commission
    ::METHOD exportJSON
            jsonObject = bsf.import("org.json.JSONObject")
            jsonArray = bsf.import("org.json.JSONArray")
            jsonData = jsonObject~NEW
            jsonData~~put("total_amount",
self~total_amount)~~put("living_space",
self~living_space)~~put("operating_cost",
self~operating_cost)~~put("heating_cost",
self~heating_cost)~~put("deposit", self~deposit)~~put("commission",
self~commission)
```

```
            RETURN jsonData

------------------------------------------------------------------------
::CLASS consumption
    ::METHOD  hwb_value ATTRIBUTE
    ::METHOD  energy_label ATTRIBUTE
    ::METHOD  energy_type ATTRIBUTE
    ::METHOD exportCSV
            say
self~hwb_value";"self~energy_label";"self~energy_type
    ::METHOD exportJSON
            jsonObject = bsf.import("org.json.JSONObject")
            jsonArray = bsf.import("org.json.JSONArray")
            jsonData = jsonObject~NEW
            jsonData~~put("hwb_value",
self~hwb_value)~~put("energy_label",
self~energy_label)~~put("energy_type", self~energy_type)
            RETURN jsonData

------------------------------------------------------------------------
::CLASS apartment
    ::METHOD  room_count ATTRIBUTE
    ::METHOD  construction_year ATTRIBUTE
    ::METHOD  construction_type ATTRIBUTE
    ::METHOD  condition ATTRIBUTE
    ::METHOD exportCSV
            say
self~room_count";"self~construction_year";"self~construction_type";"
self~condition
    ::METHOD exportJSON
            jsonObject = bsf.import("org.json.JSONObject")
            jsonArray = bsf.import("org.json.JSONArray")
            jsonData = jsonObject~NEW
            jsonData~~put("room_count",
self~room_count)~~put("construction_year",
self~construction_year)~~put("construction_type",
self~construction_type)~~put("condition", self~condition)
            RETURN jsonData

------------------------------------------------------------------------
::CLASS ad
    ::METHOD  title ATTRIBUTE
    ::METHOD  description ATTRIBUTE
    ::METHOD exportCSV
            say self~title";"self~description
    ::METHOD exportJSON
            jsonObject = bsf.import("org.json.JSONObject")
            jsonArray = bsf.import("org.json.JSONArray")
            jsonData = jsonObject~NEW
            jsonData~~put("title", self~title)~~put("description",
self~description)
            RETURN jsonData

------------------------------------------------------------------------
```

startWebScraper_Statistic.rexx

```
-------------------------------------------------------------------------
url = "https://www.statistik.at/web_de/statistiken/menschen_und_ge-
sellschaft/wohnen/wohnkosten/110836.html"

CALL startWebscraping_Statistic url
entryList = result

System = bsf.import("java.lang.System")
current_folder = System~getProperty("user.dir")

path = current_folder"\output\temp_statistic.json"
CALL writeJSONtoFile entryList~toString, path

-------------------------------------------------------------------------
::REQUIRES "BSF.CLS"
-------------------------------------------------------------------------
::ROUTINE writeJSONtoFile
  PARSE ARG input, path

    fileWriter = bsf.import("java.io.FileWriter")
    file = fileWriter~NEW(path,.false)
    file~write(input)
    file~flush
    file~close

-------------------------------------------------------------------------
::ROUTINE startWebscraping_Statistic
    PARSE ARG url
        Jsoup = bsf.import("org.jsoup.Jsoup") --import Jsoup
ressources
        entryList = .bsf~new("org.json.JSONArray") --create JSON Ar-
ray

        --load the HTML Document of the statistic source
        SAY "Start Webscraping:" url
        mainpage = Jsoup~connect(url)~get()

        --Extract & Transform for  TABLE STATISTIC
        statistic = .statistic~NEW
        --COLUMN <NVARCHAR> DESCRIPTION
        statistic~description = mainpage~getElementsBy-
Class("header")~first()~ownText

        --Extract the Table with the statistic values
        statTable = mainpage~getElementsByClass("body")~first()
        valueList = statTable~getElementsByTag("tr")

        --Extract & Transform for  TABLE DATA_SOURCE
        data_source = .data_source~NEW()
        data_source~name="Statistic Austria"
        data_source~internal_id="none"
        data_source~url=url
```

```
        --Extract & Transform for  TABLE ADRESS
        adress = .adress~NEW()
        adress~country = "AUT"
        adress~zip_code = ""
        adress~street = ""
        adress~floor = ""

        DO i = 0 TO (15) -- iterate through rows 2005 till 2020
            statElement = valueList~get(i)--get current row

            --Extract & Transform for TABLE CHRONOLOGY
            chronology = .chronology~NEW()
            chronology~creation_time = statElement~getEle-
mentsByTag("th")~first()~ownText"-01-01 00:00:00.0000000"
            chronology~available_from = ""
            chronology~rental_period = ""

            --Extract & Transform for  TABLE KEY_FIGURES
            temp = statElement~toString
            temp = temp~changeStr(",",".")
            parse var temp colum1 "<td>" colum2 "</td>" "<td>"
colum3 "</td>" "<td>" colum4 "</td>" "<td>" colum5 "</td>" "<td>"
colum6 "</td>" "<td>"
            key_figures = .key_figures~NEW
            key_figures~total_amount = colum2
            key_figures~living_space = colum2/colum3
            key_figures~operating_cost = colum6
            key_figures~heating_cost = "NULL"
            key_figures~deposit = "NULL"
            key_figures~commission = "NULL"

            --create a JSON Object for the Ad Entry
            jsonEntry = .bsf~new("org.json.JSONObject")
            -- add the Data to the JSONObject
            jsonEntry~~put("data_source", data_source~exportJSON)
            jsonEntry~~put("adress", adress~exportJSON)
            jsonEntry~~put("key_figures", key_figures~exportJSON)
            jsonEntry~~put("chronology", chronology~exportJSON)
            jsonEntry~~put("statistic", statistic~exportJSON)

            SAY jsonEntry~toString -- print JSON String in console
            entryList~~put(jsonEntry) -- add JSON Object to Array
        END

        RETURN entryList

------------------------------------------------------------------------
::CLASS data_source
    ::METHOD name ATTRIBUTE
    ::METHOD internal_id ATTRIBUTE
    ::METHOD url ATTRIBUTE
    ::METHOD exportCSV
        say self~name";"self~internal_id";"self~"url"
    ::METHOD exportJSON
```

```
        jsonObject = bsf.import("org.json.JSONObject")
        jsonArray = bsf.import("org.json.JSONArray")

        jsonData = jsonObject~NEW
        jsonData~~put("name", self~name)~~put("internal_id",
self~internal_id)~~put("url", self~url)
        RETURN jsonData

------------------------------------------------------------------------
::CLASS chronology
    ::METHOD  creation_time ATTRIBUTE
    ::METHOD  available_from ATTRIBUTE
    ::METHOD  rental_period ATTRIBUTE
    ::METHOD exportCSV
        say self~creation_time";"self~availa-
ble_from";"self~rental_period
    ::METHOD exportJSON
            jsonObject = bsf.import("org.json.JSONObject")
            jsonArray = bsf.import("org.json.JSONArray")

            jsonData = jsonObject~NEW
            jsonData~~put("creation_time", self~crea-
tion_time)~~put("available_from", self~availa-
ble_from)~~put("rental_period", self~rental_period)
            RETURN jsonData

------------------------------------------------------------------------
::CLASS key_figures
    ::METHOD  living_space ATTRIBUTE
    ::METHOD  total_amount ATTRIBUTE
    ::METHOD  operating_cost ATTRIBUTE
    ::METHOD  heating_cost ATTRIBUTE
    ::METHOD  deposit ATTRIBUTE
    ::METHOD  commission ATTRIBUTE
    ::METHOD exportCSV
            say self~living_space";"self~total_amount";"self~operat-
ing_cost";"self~heating_cost";"self~deposit";"self~commission
    ::METHOD exportJSON
            jsonObject = bsf.import("org.json.JSONObject")
            jsonArray = bsf.import("org.json.JSONArray")

            jsonData = jsonObject~NEW
            jsonData~~put("total_amount", self~to-
tal_amount)~~put("living_space", self~living_space)~~put("operat-
ing_cost", self~operating_cost)~~put("heating_cost", self~heat-
ing_cost)~~put("deposit", self~deposit)~~put("commission", self~com-
mission)
            RETURN jsonData

------------------------------------------------------------------------
::CLASS statistic
    ::METHOD  description ATTRIBUTE
    ::METHOD exportCSV
            say self~description
    ::METHOD exportJSON
```

```
            jsonObject = bsf.import("org.json.JSONObject")
            jsonArray = bsf.import("org.json.JSONArray")

            jsonData = jsonObject~NEW
            jsonData~~put("description", self~description)
            RETURN jsonData

------------------------------------------------------------------------
::CLASS adress
    ::METHOD  country ATTRIBUTE
    ::METHOD  zip_code ATTRIBUTE
    ::METHOD  street ATTRIBUTE
    ::METHOD  floor ATTRIBUTE
    ::METHOD exportCSV
            say self~coun-
try";"self~zip_code";"self~street";"self~floor
    ::METHOD exportJSON
            jsonObject = bsf.import("org.json.JSONObject")
            jsonArray = bsf.import("org.json.JSONArray")

            jsonData = jsonObject~NEW
            jsonData~~put("country", self~country)~~put("zip_code",
self~zip_code)~~put("street", self~street)~~put("floor", self~floor)
            RETURN jsonData

------------------------------------------------------------------------
```

insertDataIntoDB.rexx

```
-------------------------------------------------------------------
-- import ressources
FileReader = bsf.import("java.io.FileReader")
FileBuffered = bsf.import("java.io.BufferedReader")
JsonArray = bsf.import("org.json.JSONArray")

-- SAP HANA Connection Information
url = "jdbc:sap://<SQL-Endpoint>"
user = "<DB-User>"
pw = "<DB-Password"

-- connect to SAP HANA Database
conImmoDB = .sapConnection~NEW(url,user,pw)

-- get current project directory
System = bsf.import("java.lang.System")
current_folder = System~getProperty("user.dir")

-- read the ist value JSON Array from JSON File
reader = FileReader~NEW(current_folder"\output\temp_ist.json ")
bfr = FileBuffered~NEW(reader)
content = bfr~readLine()

if (content \= .nil | content \= "")THEN
    DO
        adList = JsonArray~NEW(content) -- create the JSON Array
        DO i = 0 TO (adList~length - 1) -- iterate through all JSON
Entries
            adEntry = adList~get(i)
            SAY  "INSERT "adEntry~get("data_source")~get("url")

            -- get data_source JSON Object and call method for IN-
SERT INTO
            data_source = adEntry~get("data_source")
            conImmoDB~insertInto-
DATA_SOURCE(data_source~get("name"),data_source~get("inter-
nal_id"),data_source~get("url"))

            -- get chronology JSON Object and call method for INSERT
INTO
            chronology = adEntry~get("chronology")
            conImmoDB~insertIntoCHRONOLOGY(chronology~get("crea-
tion_time"),chronology~get("available_from"),chronol-
ogy~get("rental_period"))

            -- get contact_person JSON Object and call method for
INSERT INTO
            contact_person = adEntry~get("contact_person")
            conImmoDB~insertIntoCONTACT_PERSON(contact_per-
son~get("contact_info"),contact_person~get("phone_number"),con-
tact_person~get("email"))
```

```
            -- get adress JSON Object and call method for INSERT
INTO
            adress = adEntry~get("adress")
            conImmoDB~insertIntoADRESS(adress~get("coun-
try"),adress~get("zip_code"),adress~get("street"),adress~get("floor"
))

            -- get characteristic JSON Object and call method for
INSERT INTO
            characteristic = adEntry~get("characteristic")
            conImmoDB~insertIntoCHARACTERISTIC(characteris-
tic~get("balcony"),characteristic~get("basement"),characteris-
tic~get("terrace"),characteristic~get("parking_spot"),characteris-
tic~get("elavator"))

            -- get key_figures JSON Object and call method for IN-
SERT INTO
            key_figures = adEntry~get("key_figures")
            conImmoDB~insertIntoKEY_FIGURES(key_figures~get("liv-
ing_space"),key_figures~get("total_amount"),key_figures~get("operat-
ing_cost"),key_figures~get("heating_cost"),key_figures~get("de-
posit"),key_figures~get("commission"),)

            -- get consumption JSON Object and call method INSERT
INTO
            consumption = adEntry~get("consumption")
            conImmoDB~insertIntoCONSUMPTION(consump-
tion~get("hwb_value"),consumption~get("energy_label"),consump-
tion~get("energy_type"))

            -- get apartment JSON Object and call method for INSERT
INTO
            apartment = adEntry~get("apartment")
            conImmoDB~insertIntoAPARTMENT(apart-
ment~get("room_count"),apartment~get("construction_year"),apart-
ment~get("construction_type"),apartment~get("condition"),)

            -- get apartment JSON Object and call method for INSERT
INTO
            ad = adEntry~get("ad")
            conImmoDB~insertIntoAD(ad~get("title"))
        END
    END

-- read the statistic value JSON Array from JSON File
reader =FileReader~NEW(current_folder"\output\temp_statistic.json ")
bfr = FileBuffered~NEW(reader)
content = bfr~readLine()

if (content \= .nil | content \= "")THEN
    DO
        adList = JsonArray~NEW(content) --create the JSON Array
        DO i = 0 TO (adList~length - 1) -- iterate through all JSON
Entries
            adEntry = adList~get(i)
```

```
            SAY  "INSERT "adEntry~get("data_source")~get("url")

            -- get data_source JSON Object and call method for IN-
SERT INTO
            data_source = adEntry~get("data_source")
            conImmoDB~insertInto-
DATA_SOURCE(data_source~get("name"),data_source~get("inter-
nal_id"),data_source~get("url"))

            -- get chronology JSON Object and call method for INSERT
INTO
            chronology = adEntry~get("chronology")
            conImmoDB~insertIntoCHRONOLOGY(chronology~get("crea-
tion_time"),chronology~get("available_from"),chronol-
ogy~get("rental_period"))

            -- get adress JSON Object and call method for INSERT
INTO
            adress = adEntry~get("adress")
            conImmoDB~insertIntoADRESS(adress~get("coun-
try"),adress~get("zip_code"),adress~get("street"),adress~get("floor"
))

            -- get key_figures JSON Object and call method for IN-
SERT INTO
            key_figures = adEntry~get("key_figures")
            conImmoDB~insertIntoKEY_FIGURES(key_figures~get("liv-
ing_space"),key_figures~get("total_amount"),key_figures~get("operat-
ing_cost"),key_figures~get("heating_cost"),key_figures~get("de-
posit"),key_figures~get("commission"),)

            -- get key_figures JSON Object and call method for IN-
SERT INTO
            statistic = adEntry~get("statistic")
            conImmoDB~insertIntoSTATISTIC(statistic~get("descrip-
tion"))

        END
    END

------------------------------------------------------------------------
::REQUIRES "BSF.CLS"
------------------------------------------------------------------------
::CLASS sapConnection
    ::METHOD conn ATTRIBUTE
    ::METHOD init
        USE ARG url, dbuser, dbpw
        DriverManager = bsf.import("java.sql.DriverManager")
        self~conn = DriverManager~getConnection(url,dbuser,dbpw)
        SAY "Connected success"
        SAY "HANA Url:" url
        SAY "User: "dbuser
    ::METHOD insertIntoDATA_SOURCE
        EXPOSE conn
        USE ARG source_name, internal_id, url
```

```
        stmt=conn~createStatement
        stmt~executeUpdate("INSERT INTO IM-
MODB.DATA_SOURCE(SOURCE_NAME, INTERNAL_ID, URL) VAL-
UES('"source_name"', '"internal_id"', '"url"');")
    ::METHOD insertIntoCHRONOLOGY
        EXPOSE conn
        USE ARG ad_creation_time,  available_from, rental_period
        stmt=conn~createStatement
        stmt~executeUpdate("INSERT INTO IMMODB.CHRONOLOGY(AD_CREA-
TION_TIME, ENTRY_CREATION_TIME, AVAILABLE_FROM, RENTAL_PERIOD) VAL-
UES('"ad_creation_time"', CURRENT_TIMESTAMP, '"available_from"',
'"rental_period"') ;")
    ::METHOD insertIntoCONTACT_PERSON
        EXPOSE conn
        USE ARG contact_info , phone_number ,  email
        stmt=conn~createStatement
        stmt~executeUpdate("INSERT INTO IMMODB.CONTACT_PERSON( CON-
TACT_INFO ,  PHONE_NUMBER ,  EMAIL ) VALUES('"contact_info"',
'"phone_number"', '"email"');")
    ::METHOD insertIntoADRESS
        EXPOSE conn
        USE ARG  country ,  zip_code ,  street ,  floor
        stmt=conn~createStatement
        stmt~executeUpdate("INSERT INTO IMMODB.ADRESS( COUNTRY ,
ZIP_CODE ,  STREET ,  FLOOR ) VALUES('"country"', '"zip_code"',
'"street"', '"floor"');")
    ::METHOD insertIntoCHARACTERISTIC
        EXPOSE conn
        USE ARG balcony ,  basement ,  terrace ,  parking_spot ,
elevator
        stmt=conn~createStatement
        stmt~executeUpdate("INSERT INTO IMMODB.CHARACTERISTIC( BAL-
CONY ,  BASEMENT ,  TERRACE ,  PARKING_SPOT ,  ELEVATOR )  VAL-
UES("balcony", "basement", "terrace", "parking_spot", "elevator");")
    ::METHOD insertIntoKEY_FIGURES
        EXPOSE conn
        USE ARG living_space, total_amount ,  operating_cost ,
heating_cost ,  deposit ,  commission
        IF (living_space == "" | living_space == "0.00 " | liv-
ing_space == 0)THEN living_space="NULL"
        IF (total_amount == "" | living_space == "0.00 " | liv-
ing_space == 0)THEN total_amount="NULL"
        IF (operating_cost == "")THEN operating_cost="NULL"
        IF (heating_cost == "")THEN heating_cost="NULL"
        IF (deposit == "")THEN deposit="NULL"
        IF (commission \= "")THEN commission = SUBSTR(commis-
sion,1,200)
        stmt=conn~createStatement
        stmt~executeUpdate("INSERT INTO IMMODB.KEY_FIGURES(LIV-
ING_SPACE, TOTAL_AMOUNT ,  OPERATING_COST ,  HEATING_COST ,  DEPOSIT
,  COMMISSION ) VALUES("living_space","total_amount", "operat-
ing_cost", "heating_cost", "deposit", '"commission"');")
    ::METHOD insertIntoCONSUMPTION
        EXPOSE conn
        USE ARG hwb_value ,  energy_label ,  energy_typ
```

```
        stmt=conn~createStatement
        stmt~executeUpdate("INSERT INTO IMMODB.CONSUMPTION(
HWB_VALUE ,  ENERGY_LABEL ,  ENERGY_TYP ) VALUES('"hwb_value"',
'"energy_label"', '"energy_typ"');")
    ::METHOD insertIntoSTATISTIC
            EXPOSE conn
            USE ARG description
            description = SUBSTR(description,1,100)
            stmt=conn~createStatement
            stmt~executeUpdate("INSERT INTO IMMODB.STATISTIC(DE-
SCRIPTION, ADRESS_ID, KEY_FIGURES_ID, CHRONOLOGY_ID, DATA_SOURCE_ID)
VALUES(    '"description"',   ( SELECT MAX(ADRESS_ID)FROM IM-
MODB.ADRESS ),   ( SELECT MAX(KEY_FIGURES_ID)FROM IMMODB.KEY_FIGURES
), ( SELECT MAX(CHRONOLOGY_ID)FROM IMMODB.CHRONOLOGY ),   ( SELECT
MAX(DATA_SOURCE_ID)FROM IMMODB.DATA_SOURCE ));")
    ::METHOD insertIntoAPARTMENT
        EXPOSE conn
        USE ARG room_count ,   construction_year ,  construc-
tion_type ,  apartment_condition
        IF (room_count == "")THEN room_count="NULL"
        stmt=conn~createStatement
        stmt~executeUpdate("INSERT INTO IMMODB.APARTMENT( KEY_FIG-
URES_ID ,  ADRESS_ID ,  CONSUMPTION_ID ,  CHARACTERISTIC_ID ,
ROOM_COUNT ,  CONSTRUCTION_YEAR ,  CONSTRUCTION_TYPE ,  APART-
MENT_CONDITION ) VALUES(( SELECT MAX( KEY_FIGURES_ID )    FROM  IM-
MODB . KEY_FIGURES ), ( SELECT MAX( ADRESS_ID )FROM  IMMODB . ADRESS
),( SELECT MAX( CONSUMPTION_ID )FROM  IMMODB . CONSUMPTION ),( SE-
LECT MAX( CONSUMPTION_ID )FROM  IMMODB . CONSUMPTION
),"room_count",'"construction_year"','"construction_type"','"apart-
ment_condition"');")
    ::METHOD insertIntoAD
        EXPOSE conn
        USE ARG title
        stmt=conn~createStatement
        stmt~executeUpdate("INSERT INTO IMMODB.AD( TITLE ,  CON-
TACT_PERSON_ID ,  APARTMENT_ID ,  CHRONOLOGY_ID ,  DATA_SOURCE_ID )
VALUES('"title"',( SELECT MAX( CONTACT_PERSON_ID )FROM  IMMODB .
CONTACT_PERSON  ), ( SELECT MAX( APARTMENT_ID )FROM  IMMODB . APART-
MENT  ), ( SELECT MAX( CHRONOLOGY_ID )FROM  IMMODB . CHRONOLOGY  ),(
SELECT MAX( DATA_SOURCE_ID )FROM  IMMODB . DATA_SOURCE)  );")

------------------------------------------------------------------------
```