



Brilliant
& Simple

DSci

Field of Data Science

Paul Julitz

Notizen

03.01.2020

Inhaltsverzeichnis

I Gesprächsnotizen	8
1 Events	9
1.1 Qlik Tag	9
2 Disposition - Fachliche und Anwendungsgrundlagen	13
2.1 Allgemein	13
2.2 Statistik ST Dbst MA IST	14
2.3 Plan und Dispo	16
2.4 Dienstbestandteile	16
2.4.1 Statistik: ST Dbst MA Ist BEG	17
3 DSci Infrastruktur	21
3.1 Power BI Gespräch	21
3.2 Qlik Rahmenbedingungen	24
3.3 AWS Vertriebsplattform (EuLe)	25
3.4 DB Modular Cloud	26
3.4.1 Beispiel Avi	26
3.4.2 Beispiel Konto von Roberto	30
3.4.3 Anforderungen	31
3.4.4 Dokumente	31
3.4.5 Schutzbedarf feststellung	31
3.4.6 BEAM Konto	32
3.4.7 Für was?	33
II Power BI	34
1 Working with M (Power Query) in Power BI	35
1.0.1 Variable	35
1.0.2 Variable name	35
1.0.3 Zuweisen / Syntax	36
1.0.4 number, text, logical	37
1.0.5 null	37
1.0.6 Type	37
1.0.7 Intrinsische Werte	38
1.1 Strukturierte Variablen	38
1.1.1 List	38
1.1.2 Record	38
1.1.3 Table	38
1.1.4 Beispiele	39
1.2 Function	39
1.2.1 Implicit function	39
1.2.2 Explicit function	39
1.2.3 Bewertungen (Conditions)	40
1.3 Spezielle	42
1.3.1 Empty Variable	42
1.3.2 Each	43
1.3.3 try otherwise	44
2 Business Intelligence: Part I - Power Query	44

2.1	Introduction to Excel Tools	44
2.2	Query Editor - Tabellenblätter	45
2.2.1	Home	46
2.2.2	Transform	46
3	Business Intelligence: Part II - Data Modeling 101	49
3.1	Data Normalization	49
3.2	Data Model	49
3.2.1	Primary and Foreign Key	49
3.2.2	Beziehungen	50
3.2.2.1	Star Model	51
3.2.2.2	Snowflake Model	51
3.2.3	Mehrere Daten Tabellen	52
3.2.4	Filter	52
3.2.5	Hide fields form client tools	54
3.2.6	Hierarchy	55
4	Business Intelligence: Power Pivot and DAX	56
4.1	Power Pivot 101	56
4.1.1	Oberfläche von Power Pivot	56
4.1.2	Berechnungen	56
4.1.3	Funktion mit normalen Tabellen	56
4.1.4	Calculated Columns	58
4.1.5	Implicit Funktionen	59
4.1.6	Explicit Funktionen - Measures	59
4.1.7	Filterfunktion	61
4.2	Basic DAX Function	61
4.2.1	Syntax und Operatoren	61
4.2.2	Function	62
4.2.2.1	Switch Function	63
4.2.2.2	Text Funktionen	64
4.3	Advanced DAX Function	64
4.3.1	Calculate	64
4.3.2	Filter	65
4.3.3	All	66
4.3.4	Iterator (X) - SumX()	67
4.3.5	Iterator (X)- RankX()	68
4.3.6	Time-Intelligence Function	69
5	Advanced Microsoft Power BI	70
5.1	Filter	70
5.1.1	Filter für Measure und DAX Funktionen	70
5.1.2	All-Funktion	70
5.1.3	Filter-Funktion	71
5.2	Introducing the X-Factor	73
5.2.1	SUMX()	73
5.2.2	COUNTX()	73
5.2.3	RANK.EQ() und RANKX()	73

III	VBA	75
1	Das VBA-Tutorial	76
1.1	Grundlagen	76
1.1.1	Debug.Print	76
1.1.2	Array	76
1.2	Objekte	76
1.2.1	Klassen	76
1.2.2	Methoden	78
1.2.3	Eigenschaft	78
1.2.3.1	Property	78

1.2.3.2	Property Prozedur	79
1.2.3.3	Unterobjekte	80
1.2.4	Auflistung	81
1.2.5	Ereignisse	82
1.2.6	Objektanweisung	83
1.2.7	Me-Object	84
1.2.8	Objektmodell	85
1.3	Fehlerbehandlung	86
1.4	Kurzer Einblick Formular	87
1.5	Microsoft Access	87
1.5.1	Marcos, Module	87
1.5.2	Formulare, Reports	87
1.5.3	Tabellen	88
2	VBA: Access	89
2.1	Sammelsurium	89
2.2	Introduction VBA Basic	89
2.2.1	VBA Syntax	89
2.2.2	Object Modell for Access	90
2.2.3	VBA Editor	91
2.2.4	Processor: Subroutine and function	91
2.2.5	Processor: Function	92
2.2.6	Set Statement	92
2.2.7	First Code-Marco	92
2.3	Variables, Constants, Calculation	94
2.4	Add Logic to your VBA Code	94
2.4.1	Iterator-Prozeduren	94
2.5	Debug Code	95
2.6	Manipulate Database Object using DoCmd Object	95
2.6.1	Open Objects	95
2.6.2	Close Objects	96
2.6.3	RunCommand Export	96
2.7	Read and manipulate Table Data	96
2.7.1	Add New	96
2.7.2	Edit	96
2.7.3	Data Perservation	96
2.7.4	TableDef	97
2.8	Manipulate a Database using the Application Object	97
2.8.1	Function to Summarize	97
2.8.2	DLookup()	97
2.9	Control Forms and Reports	97
2.9.1	Sinnvolle Einschränkungen	97
2.9.2	Verfeinerung	98

IV	SQL	99
1	SQL - Conceptual	100
1.1	Keys	100
1.1.1	Primäry Key	100
1.1.2	Composite Key	100
1.1.3	Forgein Key	100
1.1.4	Composite Key	101
1.1.5	Compound Key	101
1.2	Normilization	101
1.2.1	1NF - Atomicity	101
1.2.2	2NF - Partial Dependency	102
1.2.3	3NF - Transitive Dependency	103
1.3	Junction	103

1.3.1	One-to-One Relationship	103
1.3.2	One-to-Many Relationship	104
1.3.3	Many-to-Many Relationship	104
1.4	Data Integrity	104
2	SQL - Introduction	104
2.1	SELECT Statement	105
2.1.1	FROM - Input table	106
2.1.2	WHERE - Where row is evaluated	106
2.1.3	GROUP BY - zusammenfassen von Spalten	107
2.1.4	ORDER BY - sort table	107
2.1.5	LIMIT - Who many Rows	109
2.1.6	OFFSET - Ausschluss der ersten Zeilen	109
2.2	Concepts	109
2.2.1	Function Types	109
2.2.1.1	Count()	110
2.2.1.2	Aggregatfunktionen	110
2.2.2	DISTINCT Clause	110
2.2.3	CREATE TABLE ... ()	111
2.2.4	DROP TABLE	112
2.2.5	INSERT INTO ... ()()	112
2.2.6	UPDATE ... SET ... WHERE	113
2.2.7	DELETE FROM ... WHERE	113
2.2.8	NULL Value	113
2.2.9	DEFAULT	113
2.2.10	UNIQUE	114
2.2.11	ALTER TABLE ... ADD	114
2.2.12	CASE - Conditional Expression	114
2.2.13	Ansteuern von Spalten	115
2.3	Joints	115
2.3.1	Join on or more tables	115
2.3.2	Kinds of Joins	116
2.4	Selection of Function	116
2.4.1	Find the lenght of a string	117
2.4.2	Substring of Field of Date	117
2.4.3	Count()- Group by (aggregate function)	117
2.4.3.1	Count(), Group by - same coloum	117
2.4.3.2	Count(), Group by - different coloum	117
2.4.4	Mix	118
2.5	Transaction	118
2.6	Triggers	119
2.7	Subselects	120
2.7.1	IN Operator	120
2.7.2	View - temporay table	121
3	Object-Oriented Design	121
3.1	Objekt Orientierte Grundlagen	121
3.1.1	Beispiel	121
3.1.2	Objekte	122
3.1.3	Klassen	122
3.1.4	Abstraction, Encapsulation, Inheritance, Polymorphismen	124
3.1.4.1	Abstraction	124
3.1.4.2	Encapsulation	124
3.1.4.3	Inheritance	124
3.1.4.4	Polymorphism	125
3.2	Class Diagramm	126
3.2.1	Creating Class Diagramm	126
3.2.2	Construktur	126

3.2.3	Destruktor	127
3.2.4	Static / Instanz Variable/ Methode	127
3.3	Inheritance and Composition	128
3.3.1	Abstract class/ Methode	129
3.3.2	Interfaces	130
3.3.3	Aggregation / Composition	131
3.3.4	Zusammenfassung Relationen	132
3.4	Software Development	132

V Python 134

1	Python Introduction	135
1.1	Running Python from VS Code	135
1.1.1	Befehle	135
1.1.2	Launch Json File	135
1.1.3	Internal Console instead of Terminal	135
1.2	Basis Concepts	135
1.2.1	Variables and Expressions	135
1.2.2	Function	136
1.2.3	Conditionals	137
1.2.4	Loop	137
1.3	OOP - Class	138
1.3.1	Structure of a Class	138
1.3.2	Inheritance and Override	138
1.3.3	Methode Overload	139
1.3.4	The self Argument	140
1.3.5	Attributes	140
1.3.5.1	Ohne Constructur init ist falsch	140
1.3.5.2	Mit Constructor init ist richtig	140
1.3.5.3	Attribut einer Klasse direkt zuordnen.	140
1.3.6	Call methode from another Class	141
1.3.7	Public, Private and Protected	142
1.3.8	From ... import	143
1.4	Modul Formating Date and Time	143
1.4.1	Retriev information form datetime module with date, time and datetime class	143
1.4.2	Formating	144
1.4.3	Timedelta	145
1.4.4	Calendar	146
1.4.4.1	Beispiele	146
1.5	Working with files	147
1.5.1	Open(File,Mode)	147
1.5.2	OS and OS.Path	148
1.5.3	Shutil and Zip	148
1.6	Working with Webdata	149
1.6.1	Urllib Request	149
1.6.2	JSON	150
1.6.3	HTML Parser (Plus: itertools)	153
1.6.4	xml	154
2	Python with Excel	154
2.1	Pandas	154

VI GitHub 155

1	Konzepte	156
1.1	Befehle	156
1.2	Unterschied zwischen git bash, cmd und gui	156
1.3	Staging Area / Index / Cache	157

1.4	Committing	159
2	Example: github-slideshow	160
2.1	Clone Respository and Create a Branch	160
2.2	Check all the branches of one repository	160
2.3	Chance branch name	161
2.4	Change aktiv branch	161
2.5	Push new local branch	161
2.6	Check the status of a branch	161
2.7	Create a new file	161
2.8	Update Branch	162
2.9	Pull Request	162
2.10	Delete local branch	163
2.11	Delete remote branch	164
2.12	Prune deletet remote branches	164
3	Designing your Github Page	164
VII	Qlik Sense	165
1	Sammelsurium - Qlik Sense	166
1.1	Load	166
1.2	Incremental Loading	166
1.3	Data connection types	166
2	3 - Udemy - Certificate in Qlik Sense Analytics Development	167
2.1	Budgeting and KPI (goal) setting	167
2.1.1	DevHub	167
2.1.2	Von Hub bis Sheet	168
2.1.3	Selection	171
2.2	Qlik Key	171
2.2.1	Composite Key	171
2.2.2	Synthetic Key	174
2.2.3	Speicherung	175
VIII	Machine Learning	176
1	Konzepte	177
1.1	Anwendungsfall	177
2	Unsupervised Learning	177
3	Reinforcement Learning	177
4	Supervised Learning	177
IX	Anhang	178
A	Abkürzungsverzeichnis	179
B	Glossar	182
C	Literatur	184

Teil I

Gesprächsnotizen

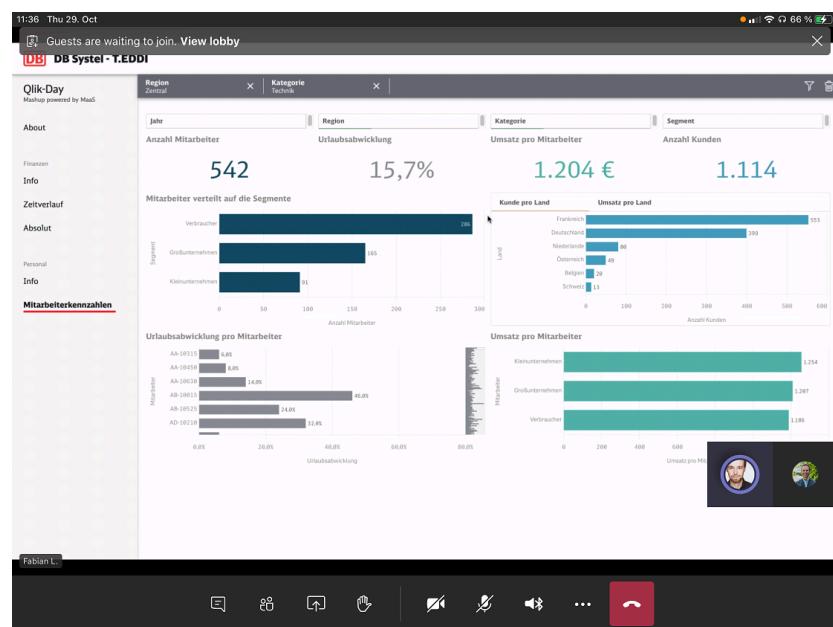
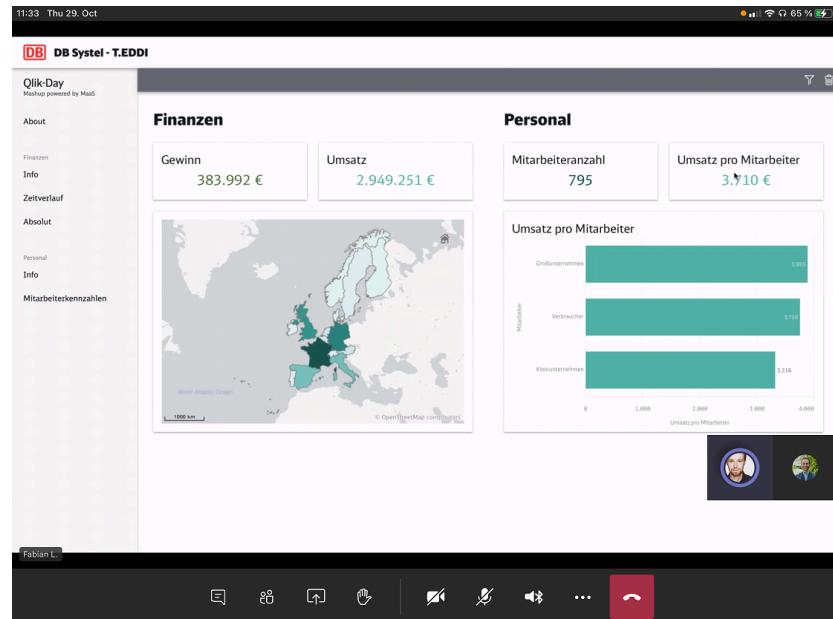
1 Events

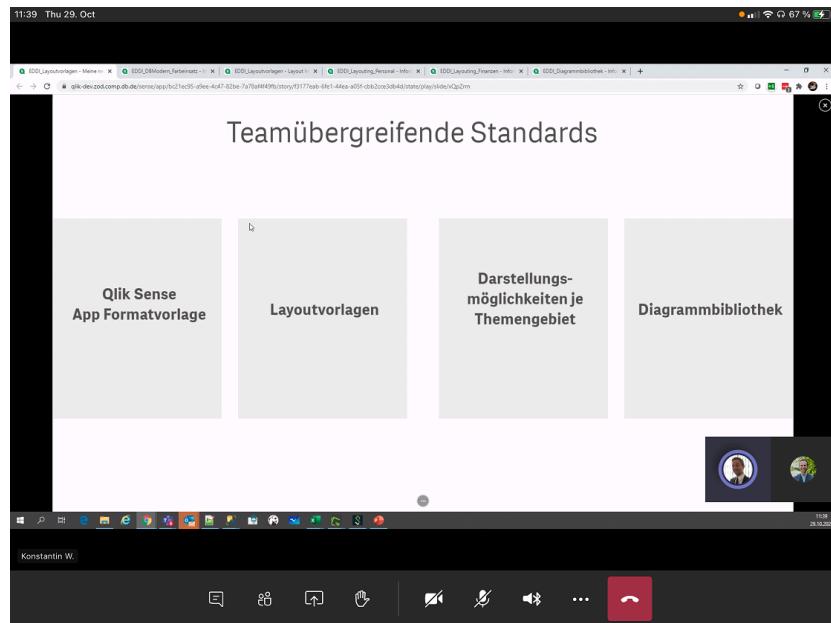
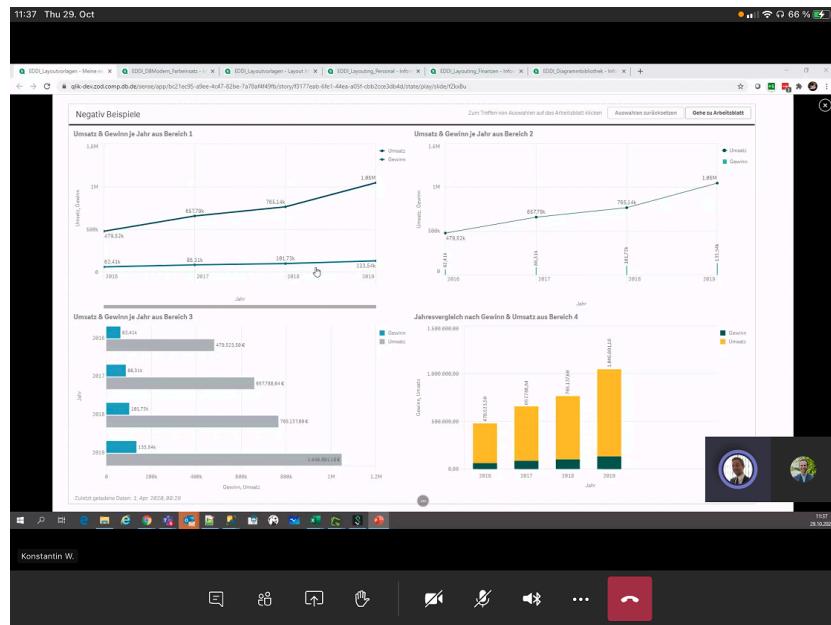
1.1 Qlik Tag

Datum: 30.10.2020

Uhrzeit: 10:30 Uhr

Qlik Layout. Wie können Informationen besser vermittelt werden.





11:41 Thu 29. Oct

Informationen zur Applikation: DBModern_Farbeinsatz

Übersicht Farben Innerhalb des Themes DBModern:

Cool Gray 200	rgb(215,220,225)
Cool Gray 400	rgb(135,140,150)
Cool Gray 500	rgb(100,105,115)
Cyan 500	rgb(48,150,200)
Cyan 700	rgb(0,106,150)
Cyan 800	rgb(0,75,100)
Light Green 200	rgb(48,116,20)
Turquoise 200	rgb(198,226,229)
Turquoise 400	rgb(0,181,174)
Turquoise 800	rgb(0,87,82)
Red 600	rgb(197,6,29)
Red 700	rgb(155,0,14)
Yellow 600	rgb(255,187,0)
Yellow 700	rgb(255,155,0)

Besonderheit:

In dieser App wird ein Überblick über die aktuell im Theme **DBModern** verwendeten Farben gegeben. Zur Übersicht werden alle rgb-Werte sowie die Namen der Farbe aus dem Marketingportal angegeben. Auf der Seite Farbverorschläge findet man dann einen Überblick, wie die Farben innerhalb des Themes eingesetzt werden können.

Die Farben wurden aus dem Marketingportal des Konzerns entnommen und entsprechen somit den aktuell gültigen Richtlinien.

Neben der Auswahl der einzelnen Farben gibt es die Möglichkeit, Farbgradienten für die Einfließung nach oben zu erstellen. Hierbei ist zu beachten, dass die Farben auf die jeweiligen Werte innerhalb der Kennzahl aufgeteilt werden.

Es gibt die Möglichkeit nach blau, grün, türkis oder orange Töne einzufiltern.

Im Theme wird ein Hintergrund verwendet, damit sich die einzelnen Qlik-Objekte besser voneinander absetzen.

Datenquellen:

Es handelt sich bei den verwendeten Daten um Testdaten, daher wird keine Aktualisierung des Datenstandes angezeigt.

Klassische Ampel Farben

Moderne Ampel Farben

Constantin W.

11:43 Thu 29. Oct

Vorschläge für Farbeinsatz

Eine einfache KPI ohne Bezeichnung der Kennzahl soll in grau dargestellt werden.

KPIs können mit Grenzwerten bestückt werden für eine Werte. In diesem Fall wäre positiv.

Einführung nach einer Kennzahl erfolgt in grau. Diese Baukästen, Legenden werden unten angezeigt.

Gesamtwert
1.474.625,46 €

Delta
1.474.625,46 €

Delta
1.474.625,46 €

Vergleich Vorschau vs. Vorjahr. Vier Balken sind das Vorjahr und die Vorschau.

Ist vs. Plan. Werte werden in blau Tönen dargestellt.

Ist vs. Plan. Werte werden unten angezeigt.

Verbraucher
796,81%
Produktionsmittel
402,27%

Einführung nach einer Kennzahl kann in blau Tönen in Balkendiagrammen, Legenden unten angezeigt.

Ein gestapeltes Balkendiagramm mit klassischen Ampelfarben kann wie folgt aussehen.

Ein gestapeltes Balkendiagramm mit modernen Ampelfarben kann wie folgt aussehen.

Vorschau gestapeltes Balkendiagramm mit klassischen Ampelfarben

Vorschau gestapeltes Balkendiagramm mit modernen Ampelfarben

gestapeltes Balkendiagramm

Constantin W.

11:48 Thu 29. Oct

Gewinn
383.992 €

Umsatz
2.949.251 €

Anzahl Produkte
1.912

Kategorie

Land/Region	Wert
Deutschland	549.421 €
Frankreich	565.120 €
Italien	393.110 €
Spanien	281.312 €
Australien	219.684 €
Malta	270.981 €
Europa/Rest	188.888 €
Österreich	99.703 €
Brasilien	81.212 €

11:50 Thu 29. Oct

Bestellstatus	Kategorie	Region	Segment
Gewinn	384k	Umsatz	2.95M
Anzahl Produkte	1.91k	Gewinn	384k
Anzahl Produkte	1.91k	Gewinn	384k

Umsatz pro Bestelldatum

Bahndatagrenn

Bestellstatus/Year

Bestellstatus/Region

Bestellstatus/Bestellstatus/Year

Umsatz pro Quartal

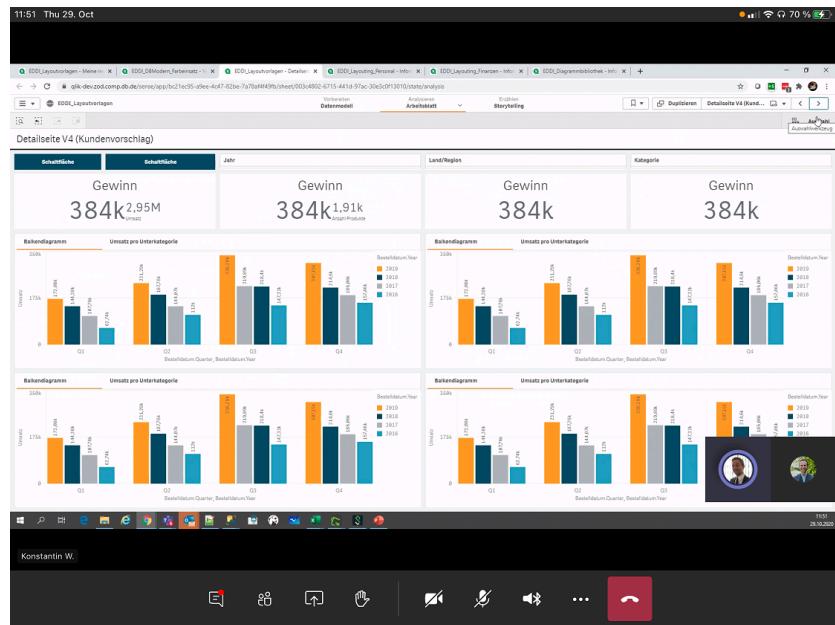
Umsatz pro Unternehmensgruppe

Umsatz

Kategorie

Region

Segment

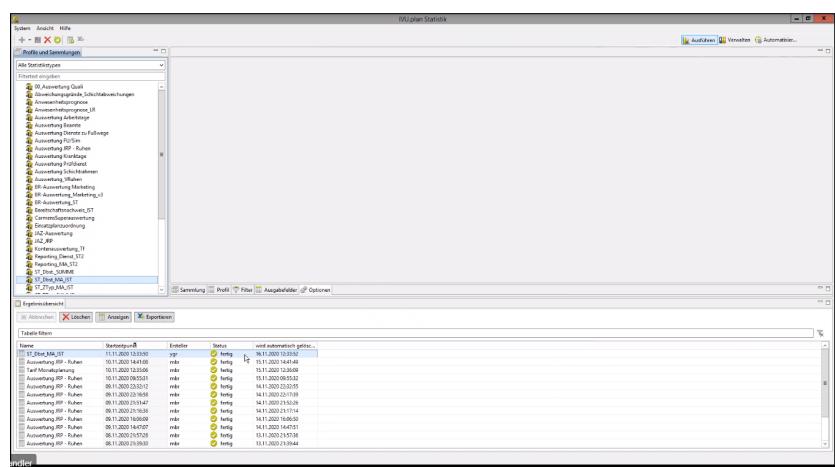


2 Disposition - Fachliche und Anwendungsgrundlagen

Gesprächspartner: Roy Kandler

2.1 Allgemein

- Die Firma **IVU Traffic Technologies AG** bietet für den DB Regio Konzern die Software integriertes Planungs- und Dispositionssystem (iPD) an. Diese ist ein umfassendes System, welches viel vorherige Einzelsysteme umfasst. Vorher war ein Teil Ausschnitt **MB.rail**.
 - Für die Auswertungen nutzen wir **IVU.plan Statistik**.



- Aktuell können 10 Schichten zu unterschiedlichen Anfangszeiten und Dauer ausgewählt werden.
 - Ein Dienst kann wie folgt aussehen.

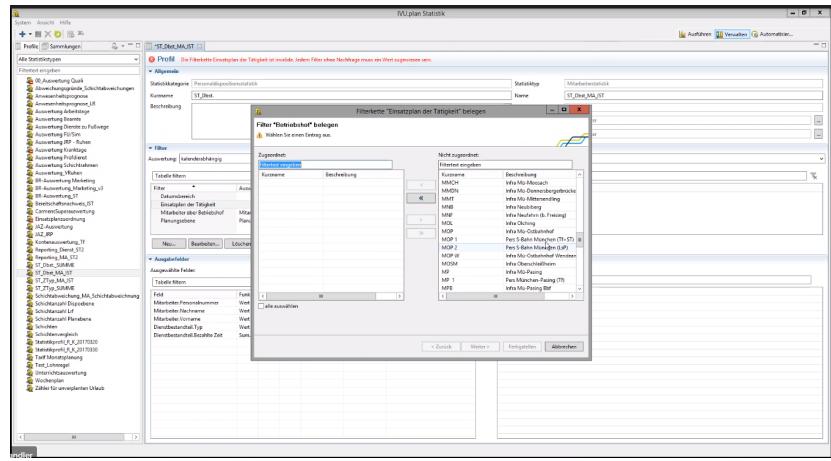
- Kurzname für den Dienstbestandteil **XPD**.
- Die Buchungen können bis zu 3 Tagen geändert werden.
- Stamm wird XPD bezeichnet.
- In iPD bestehen die Daten bis Dezember 2019. Das Archiv liegt bei Roy auf einem Laptop.

2.2 Statistik ST Dbst MA IST

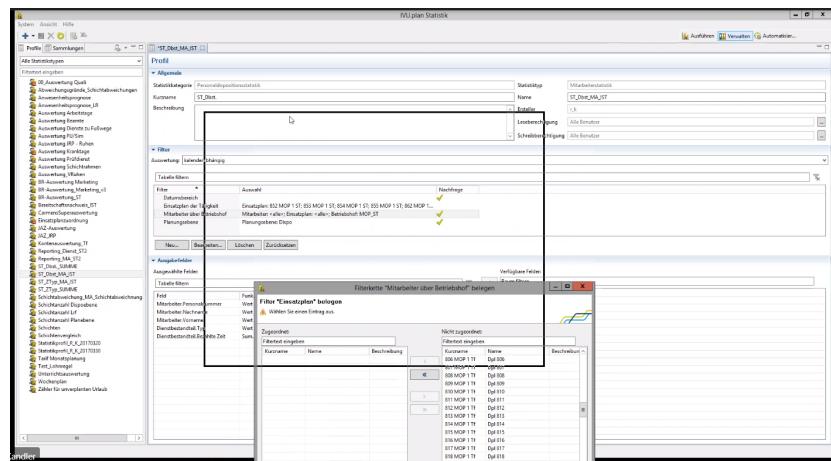
Die Liste aller Profile werden von Roy Kandler verwaltet.

Name	Statustyp	Ersteller	Status	vor 4 Minuten geladen
ST_ProfMA_IST	RP	rpk	fehl	19.11.2020 13:33:00
Auswertung RP_Fuhren	rpk	fehl	fehl	10.11.2020 14:14:08
Auswertung RP_Fuhren	rpk	fehl	fehl	10.11.2020 09:55:04
Auswertung RP_Fuhren	rpk	fehl	fehl	10.11.2020 09:55:01
Auswertung RP_Fuhren	rpk	fehl	fehl	09.11.2020 23:16:58
Auswertung RP_Fuhren	rpk	fehl	fehl	09.11.2020 23:16:47
Auswertung RP_Fuhren	rpk	fehl	fehl	09.11.2020 23:16:46
Auswertung RP_Fuhren	rpk	fehl	fehl	09.11.2020 16:06:08
Auswertung RP_Fuhren	rpk	fehl	fehl	09.11.2020 16:06:07
Auswertung RP_Fuhren	rpk	fehl	fehl	09.11.2020 15:57:28
Auswertung RP_Fuhren	rpk	fehl	fehl	09.11.2020 15:57:28

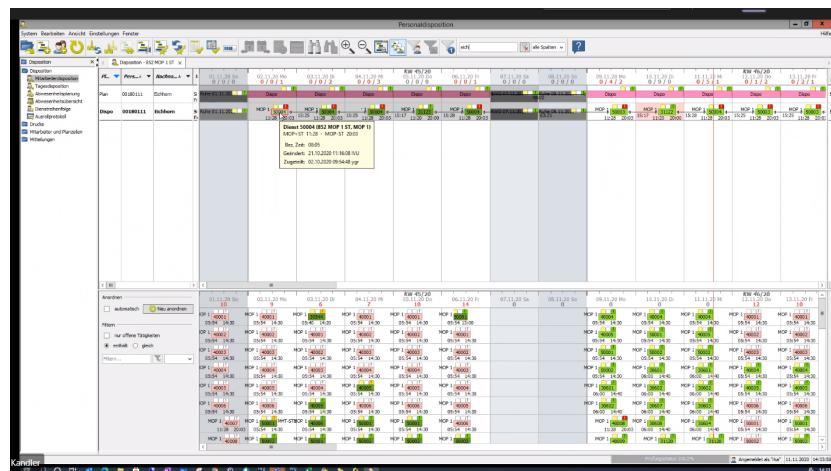
Die Ergebnisübersicht wird gezeigt, wann der Bericht geladen wird. Der Betriebshof ist **MOP 2**.



Die Einsatzpläne können ebenfalls konfiguriert werden.



Die Filtermöglichkeiten erstrecken sich auf den Einsatzplan der Tätigkeit, Mitarbeiter über Betriebshof und die Planungsebene Dispo



Die benötigten Dienstpläne sind

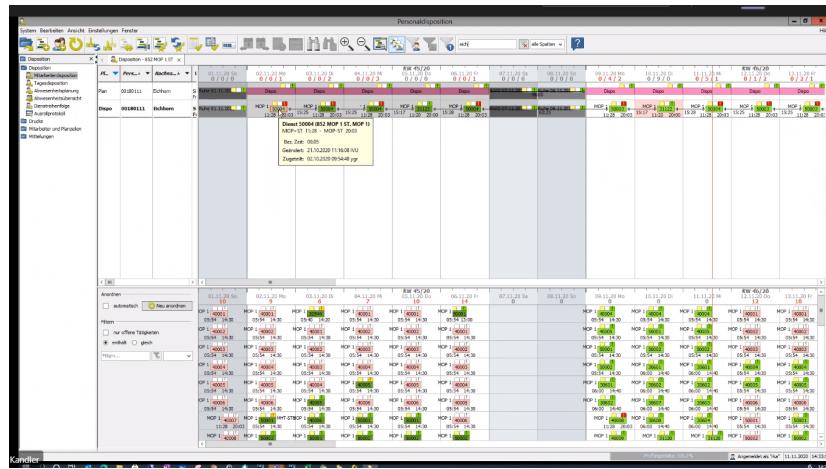
- 852
- 853
- 854

- 855
- 857* := Schichten; Kein XPD
- 862
- 864 := Bürozeiten (Dispo); Kein XPD
- 867 := Arbeiten nicht, werden nur geführt, Kein XPD
- 869 := Automatenguide; Kein XPD

Dies dient der feineren Untergliederung und der (vermutlichen) interen Kapazität der Dienstpläne.

2.3 Plan und Dispo

Der Dienstplan ist der Plan, welcher angelegt wird, als Sollplan. Die **Dispo** ist die **Schicht** und stellt damit den Habenplan dar. Diese Plannung übernimmt Thorsten und Yvonne.



2.4 Dienstbestandteile

A05PVsg Bürozeiten (Besprechung, Büro, Absprachen mit Dispo, Aussage bei Burgel für eine Zeugenaussage, Teamleitergespräche, MOSAIK, Englisch, Maskenverteilen)

A89K000 Teilkrank ohne Bezahlung auf Pausenbestanteil (UnPau) (Während der Schicht)

A89K100 Vollbezahlung (Während der Schicht)

AblHin * Frage: Eigentlich sollte es eine Nullbuchung sein. Puffer zwischen zwei weiteren Dienstbestandteilen.

AblRück * Nullbuchung (Wir dies auch zwischen PauRück und XPD gebucht?)

BEZPAU Bezahlte Pause, wenn die Pause nicht im Regeldienst genommen werden konnte.

PauHin 5 Minuten; Wegzeit zu Pause; Doppelfunktionär (10 Minuten, 20 Minuten, etc.)

PauRück Nullbuchung.)

TU * Tätigkeitsunterbrechnung (Bezahlte Pause unter Einsatzbereitschaft). Warum wird es für unsere Leute gebucht?

UNBPAU Unbezahlte Pause

ÜWegZ * Nullbuchung; Nur bei den Doppelzeitfunktionären.

XAd Abschlussdienst; Wenn die Kanzeldienst verlassen wird. (3 Minuten); Doppelfunktionäre

XAGI Automatenguides (Fahrgastzählung)/ Nur auf Anweisung von Christian (Abfrage: Wenn der Kollege in der Schicht davor oder danach XPD haben.)

XAufs Örtliche Aufsicht (Doppelfunktionäre)

XBüro Dispo-Bürozeit/ SOTIS

XME Melden Ende (1 Minuten), Am Ende einer Schicht

XMM Melden im Melderaum (3 Minuten), Am Anfang einer Schicht

XmVtK Melden auf der Kanzel (1 Minute; Dppelfunktionäre, Anmeldung in der Frühschicht/Spätschicht)

XPD Prüfdienst, Personal

XPV Zeit mit den Praxisvermittler: Geltkasse: 505; Buchungsgrund wie bei A05PVsg. Theoretisch werden die Lehrling A05PVsg auf der Habenseite, (Zeugena)

XRLenk Reisendenlenkung

XVd Vorbereitungsdienst (5 Minuten); Kanzelaufsperrn; Doppelfunktionäre

XZars Gang zur Zars; Früher gab es fixe Zeiten; Früher MT Auslesen

ZX27Fb * Betriebsdienst; Unterricht; Was beinhaltet es; WBT, RfU - Unterricht, etc.

ZX20Fb * Unklar

ZX41SU Gericht; Unbezahlte Zeit; Wenn eine Zeugenaussage benötigt wird

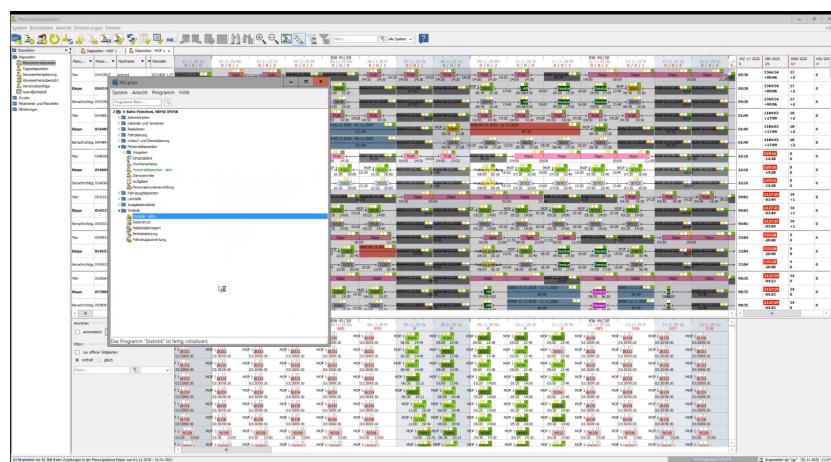
XÜ Übergabe; 2 Minuten; Doppelfunktionäre;

XWeg * Von XPD und A05PVsg; Wo sind die Wegezeiten hin.

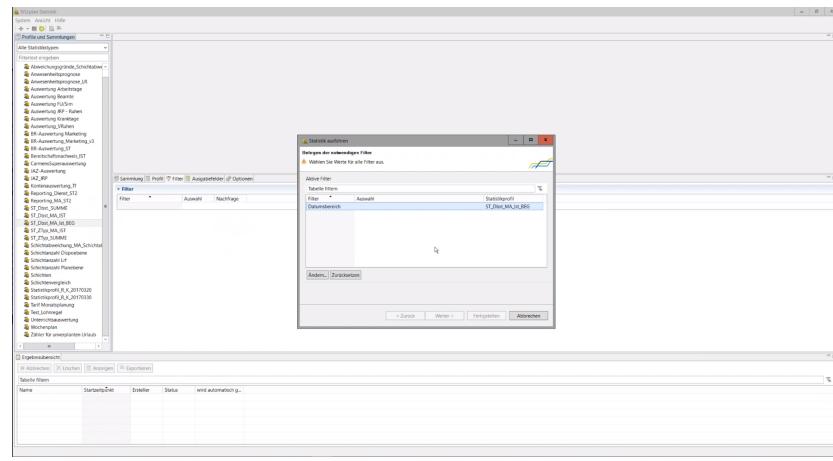
2.4.1 Statistik: ST Dbst MA Ist BEG

Mit der neuen Statistik soll eine Datenbasis geschafft werden, die für die Bayrische Eisenbahngesellschaft (BEG) Bericht dienen.

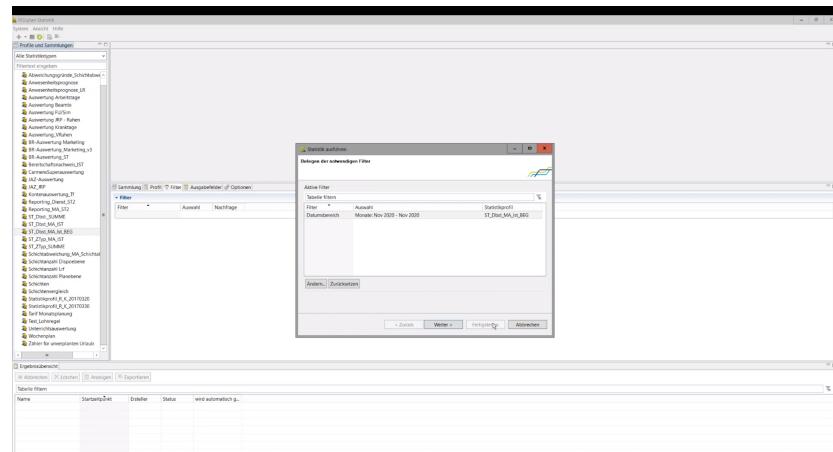
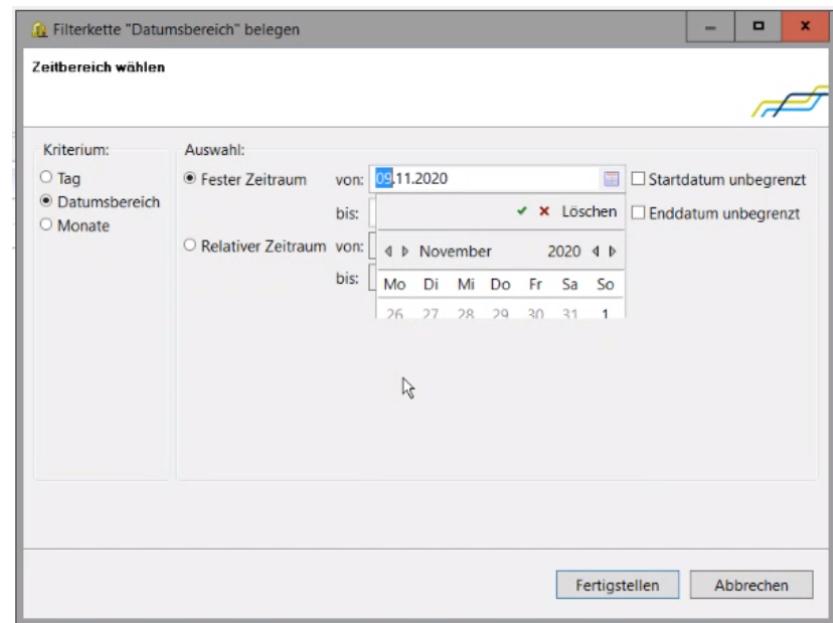
Über IVU.plan Statistik werden alle Statistik aufgerufen.



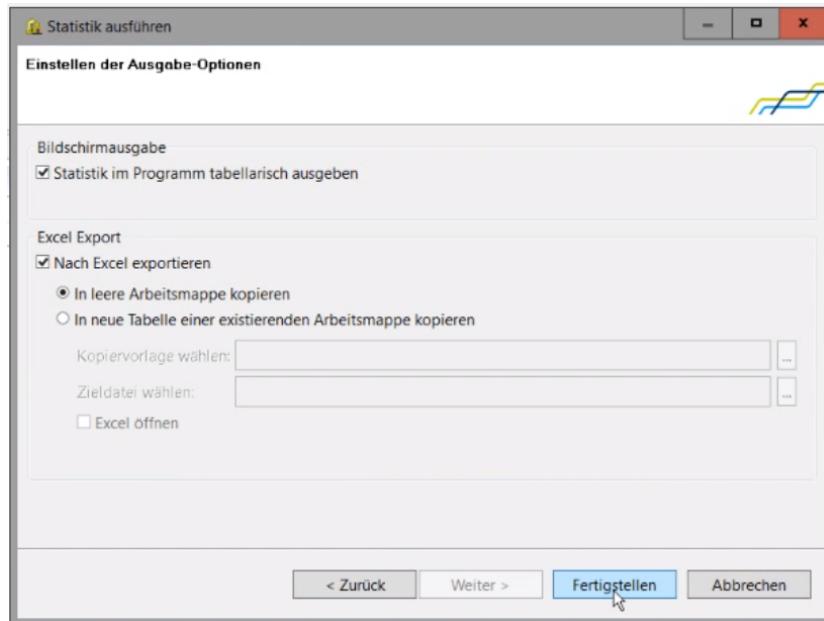
Im Modul wird der Bericht ST_Dbst_MA_Ist_BEG ausgewählt.



Im Filterbereich wird der letzte Monat ausgewählt.



Die Ablage muss noch definiert werden.



Der Bericht hat folgende Inhalte.

Microsoft Excel Screenshot showing a data table in a spreadsheet application.

The ribbon menu includes: Datei, Start, Einfügen, Seitenlayout, Formeln, Daten, Überprüfen.

The Einfügen tab is selected, showing options: Ausschneiden, Kopieren, Format übertragen.

The formula bar shows: A1, Gültigkeit.Datum.Datum (in dd.MM.yyyy HH:mm:ss).

The table has columns labeled A through F.

Column A contains numerical values from 10 to 63.

Column B contains dates and times, all starting at 0:00:00 on 01.11.2020.

Column C contains names: Bass, Ranko, PauRück, 0:00, ...

Column D contains names: Ranko, PauRück, UNBPAU, XME, ...

Column E contains names: PauHin, AblHin, XPD S1, UNBPAU, ...

Column F contains names: 0:05, 0:05, 0:05, 0:05, ...

The table continues with more rows, showing a pattern of names and times.

At the bottom of the screen, there are tabs: ST_Dbst_MA_Ist_BEG and ST_Dbst_MA_Ist_BEG_Parameter.

The page number 20 is visible in the bottom right corner.

Was noch unklar ist, warum es zwei Einsatzplan Spalten gibt und in der der Einsatzplan 857 fehlt.

A	B	C	D	E	F	G	H	I
1	Einsatzplan	Einsatzplan	Planungsebene	Mitarbeiter	Betriebshof	Betriebshof	Monate	ST_Dbst_MA_Ist_BEG
2	857 MOP 1 ST	855 MOP 1 ST	Dispo	00685610	MOP 1	MOP 1	Nov 2020	
3	855 MOP 1 ST	854 MOP 1 ST		5437629			Nov 2020	
4	854 MOP 1 ST	853 MOP 1 ST		50342443				
5	853 MOP 1 ST	852 MOP 1 ST		5437627				
6	852 MOP 1 ST	869 MOP 1 ST		05389378				
7	869 MOP 1 ST	864 MOP 1 ST		05451969				
8	864 MOP 1 ST	862 MOP 1 ST		05338344				
9	862 MOP 1 ST			05446779				
10				00814425				
11				05460678				
				05460678				

3 DSci Infrastruktur

3.1 Power BI Gespräch

Datum: 04.11.2020

Uhrzeit: 17:30 Uhr

Teilnehmer: Hermann Stelzl, Tobias Wolf (Tobia.to.Wolf@deutschebahn.com)

- Weitere Infos: [DB Planet](#)
- Zwei Fälle
 - **Anwendungsfall** bei Personenkreisen von > 15.
 - Team Nutzung bei Größen von 10 - 15. Hinweis: Zum Anfang, im Februar, wird vom PALM Prozess sich für beide Anwendungsfälle nichts ändern.

- DB internal

Abgrenzung persönliche/Team Nutzung vs. professionelle Anwendung 

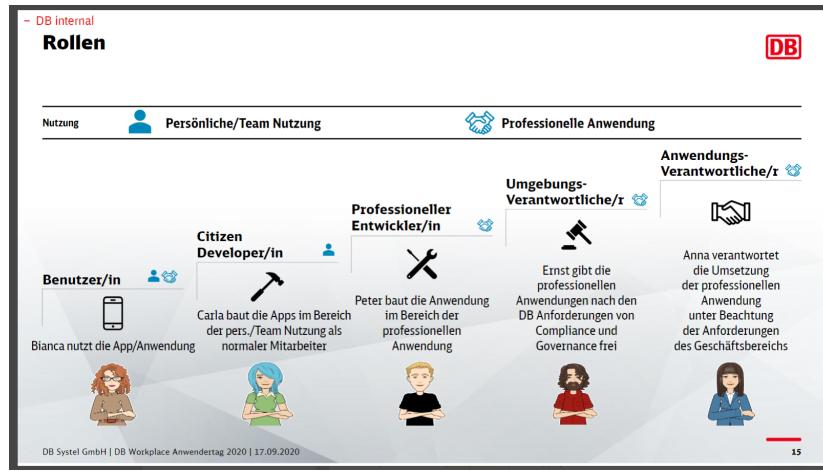
Nutzung	Persönliche/Team Nutzung	Professionelle Anwendung*
Merkmale	<ul style="list-style-type: none"> – Die App ist nicht relevant für Geschäftsprozesse oder für Geschäftsergebnisse – Sinn und Zweck ist die Produktivitätssteigerung einer einzelnen Person / eines Teams – Die App muss in keiner vereinbarten Qualität bereit gestellt werden – Es können keine Schäden bei Ausfall der Anwendung entstehen – Die App besitzt keine eigene Datenhaltung – Dient nicht dem primären Zweck der Verarbeitung personenbezogener Daten 	<ul style="list-style-type: none"> – Die Anwendung ist relevant für Geschäftsprozesse oder für Geschäftsergebnisse – Sie wird in einer vereinbarten Qualität und einem vereinbarten Service bereit gestellt – Es können Schäden entstehen, wenn die Anwendung nicht funktioniert – Die Anwendung hat eine eigene Datenhaltung – Sie steht einem größeren Benutzerkreis zur Verfügung
Umgebung	Standardumgebung	Separate Umgebungen
Mitbestimmung	Abgedeckt durch KBV DBO365	Mitbestimmung erforderlich

* Bereits bei Erfüllung eines der Merkmale handelt es sich um eine Anwendung

DB Systel GmbH | DB Workplace Anwendertag 2020 | 17.09.2020

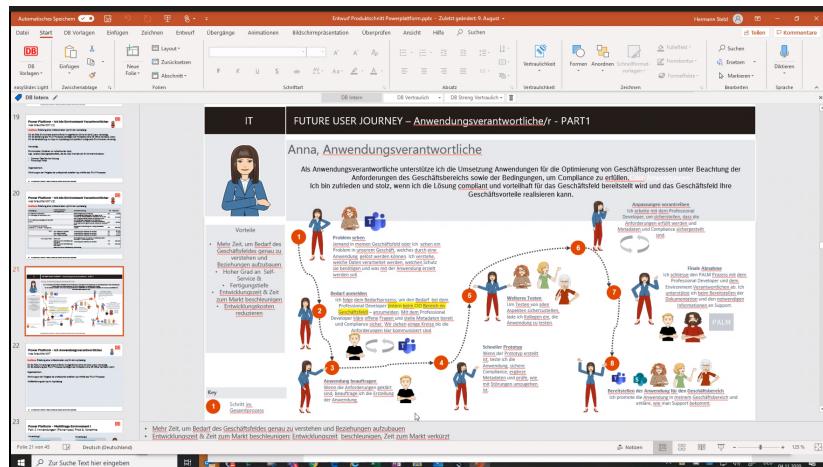
14

- User

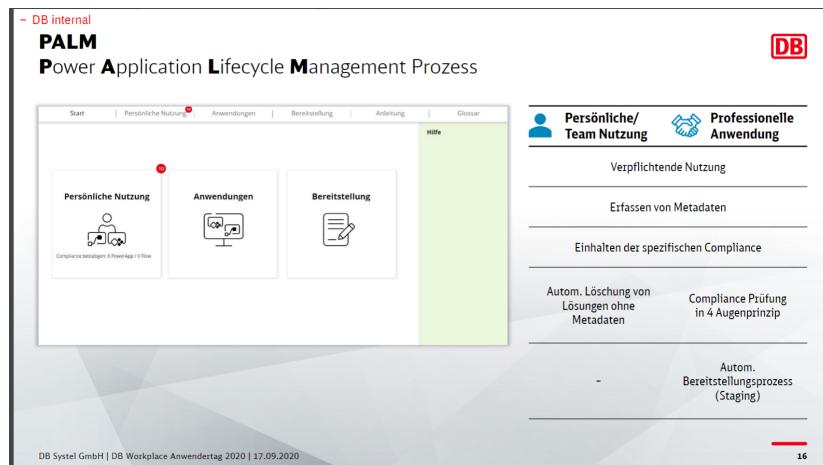


• Anwenderungsverantwortliche

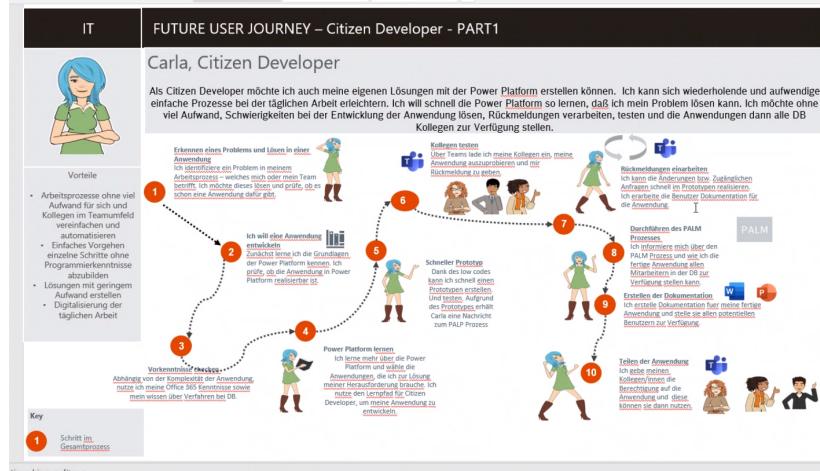
- Die **Professional Developer** lädt das Dashboard/ App in den Workspace (?) - deploy.
- Datenschutzerklärungen, BR-Beteiligung (Information), Governance (Bim-ID) muss geprüft werden.
- Die Hacken müssen gesetzt werden, und die App geht in Produktion.
- Der Gleiche Prozess muss auch bei Änderungen Ablaufen.



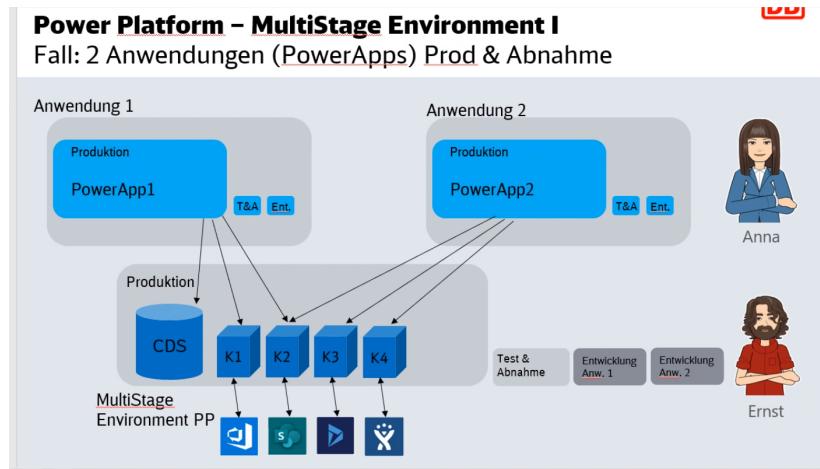
• PALM-Prozess



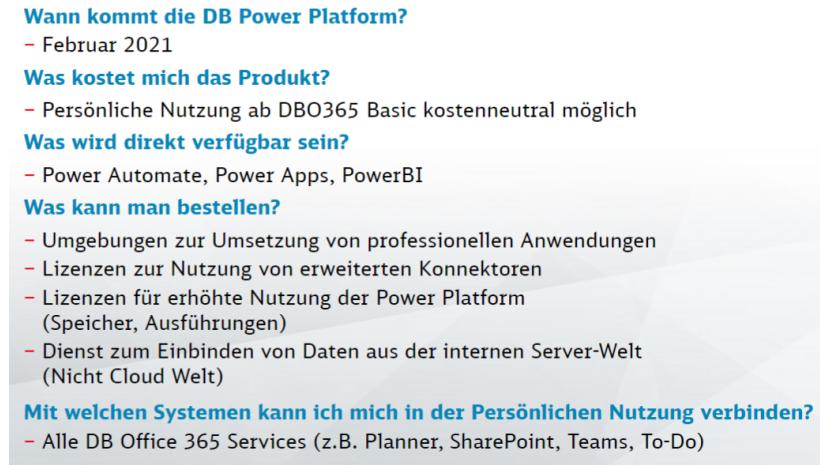
- Citizen Developer



- Es wird kein eigener Server benötigt. Es gibt eine MultiStage Environment



- Über die Conectoren werden alle Daten angebunden.



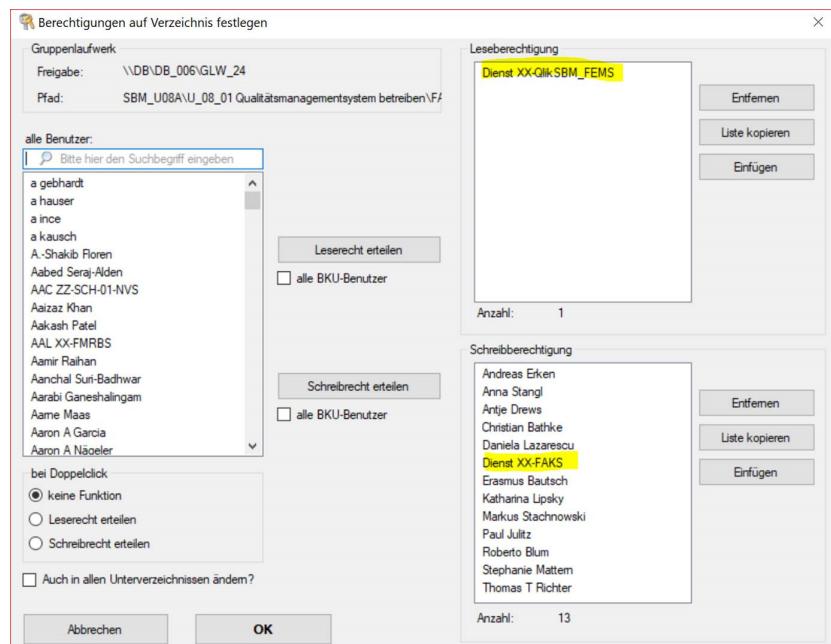
- Eine Analytics Anbindung, wie bei Amazon Web Service (AWS), soll über Azure Synaps funktionieren. In diesem Zusammenhang soll auch R oder Python Skripte abgespielt werden.

3.2 Qlik Rahmenbedingungen

- Wir haben 71 User.

- Entwicklerlizenzen haben wir 5.

- Der User **DienstQlikSBMFEMS** wurde von Cosmos Consult verwendet. Dieser greift auch auf den FAKS Ordner im BKU Netz zu, auf welchem die EBE Daten liegen.



- Rückantwort: Unterschied zwischen *qmc/useraccessallocations* und *qmc/user*?
- Der Rahmenvertrag für die S-Bahn München läuft über die Seriennummer



- Weitere Bestellung von **Site Token** (Entwicklerlizenzen) und **Konsument** erfolgt über Elisa.

The screenshot shows a software interface for managing purchases. The title bar says 'Normalbestellung 28925970 angelegt von BIANCA LEHNER'. The main area displays a grid of items with columns: S., Pos, K, P, Material, Kurztext, Bestellmenge, B..., T Lieferdatum, Nettopreis, W..., pro, B..., Waren... and a status column. The grid contains several entries for 'Qlik Sense Site Token for Prod. Site' and 'Qlik Sense Konsument'. The status column shows '10ST' for tokens and '10ST' for consumers. The bottom part of the screen shows a summary table with columns: Bestellmenge, B..., T Lieferdatum, Nettopreis, W..., pro, B..., Waren... and a status column. The total value is listed as '941,00 EUR'.

Für jede Instanz wird einen Einmal Gebühr fällig.

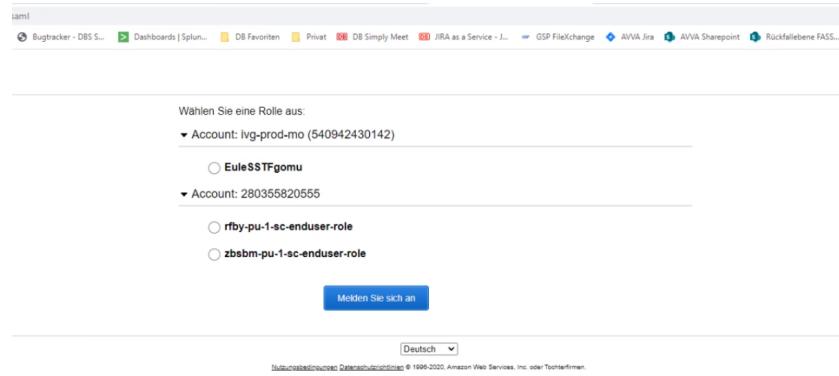
- Der Server *dbregio07.sv.db.de* wird von Regio verwaltet. Wer die Leistungsvereinbarung (LV) verwaltet, ist nicht klar.
 - Der direkte Ansprechpartner wäre Roberto
 - Jens Friedeman (Qualitätsmanagement)
 - Christian Heil
 - Servername: DBR07
- OneSource zieht um, und bittet Qlik Lizenzen an.

3.3 AWS Vertriebsplattform (EuLe)

DB Vertrieb hat die Plattform Erlös- und Leistungsdatenbank (EuLe). Auf befinden sich die Daten der Ticketautomaten. Das Konto läuft über EuLe.

The screenshot shows the AWS Management Console homepage. It features a search bar 'AWS Services' and a sidebar with 'Services finden' and 'Kürzlich besuchte Services' (including S3, Amazon SageMaker, and EC2). On the right, there are sections for 'Bleiben Sie mit Ihren AWS-Ressourcen unterwegs in Verbindung' (with a QR code for the mobile app) and 'Erkunden Sie AWS' (Amazon Redshift).

Bei Roberto hat mehrere Rollen und einen weiteren Account. Jede Anwendung verlangt eine eigene Rolle.



Die Daten für der Ticketautomaten befinden sich in Fgomu/final.

The screenshot shows the AWS S3 console. On the left, the navigation pane is open with "Amazon S3" selected. The main area shows a list of objects in the "final/" folder of the "Fgomu" bucket. The table has columns: Name, Typ, Letzte Änderung, Größe, and Speicherklasse. There are 676 objects listed, all of which are CSV files. The last object listed is "Fgomu_20190001_20190405_0511174652.csv". The URL for the folder is "https://s3.eu-central-1.amazonaws.com/540942430142.euleprod.default.output/Fgomu/final/".

3.4 DB Modular Cloud

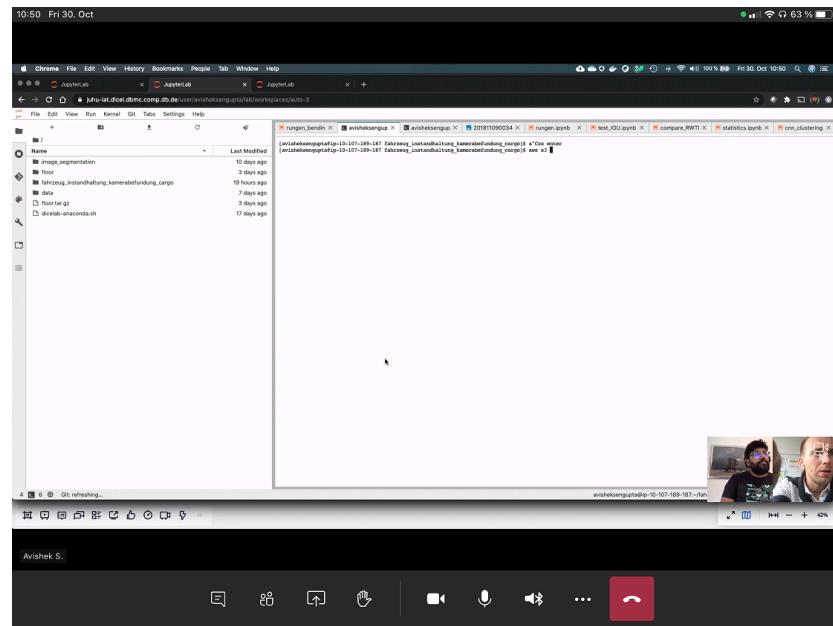
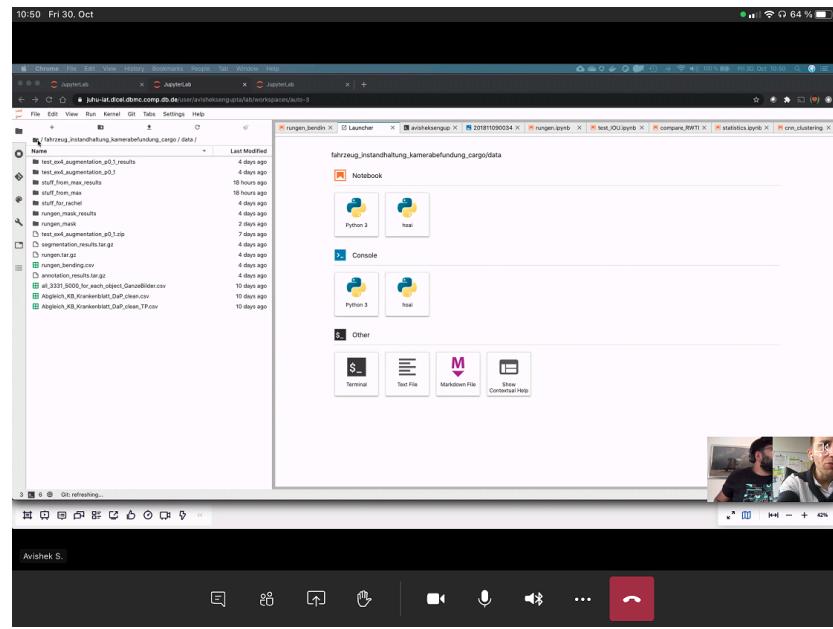
3.4.1 Beispiel Avi

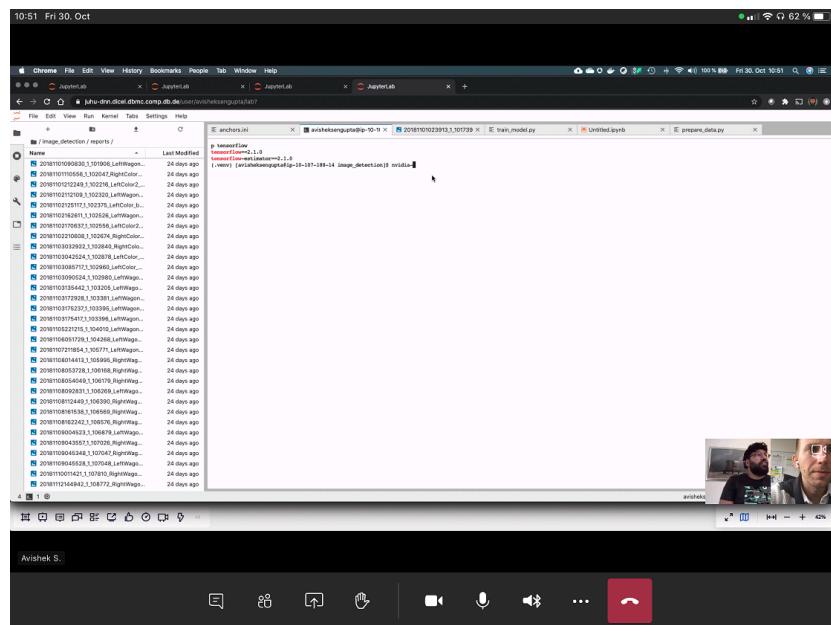
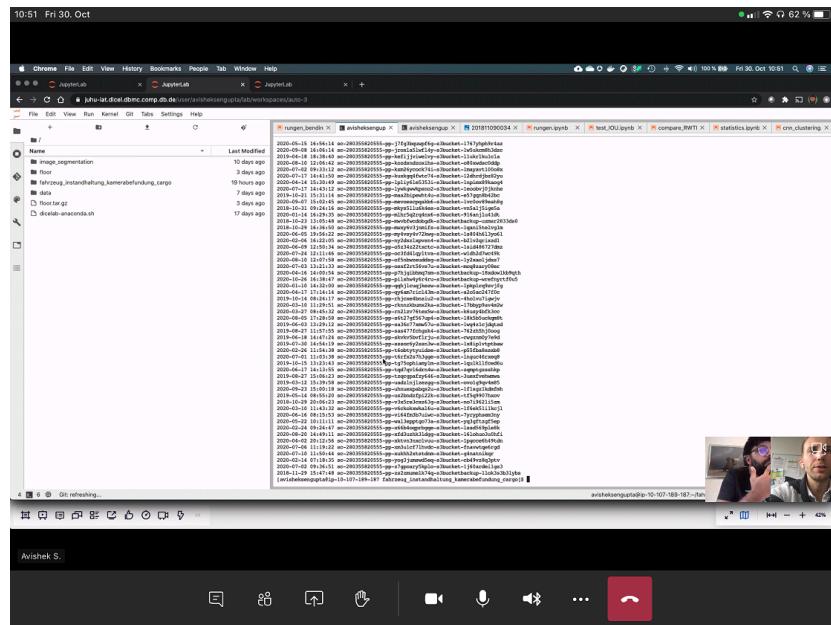
Datum: 30.10.2020

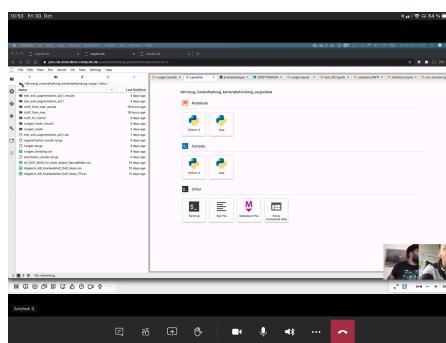
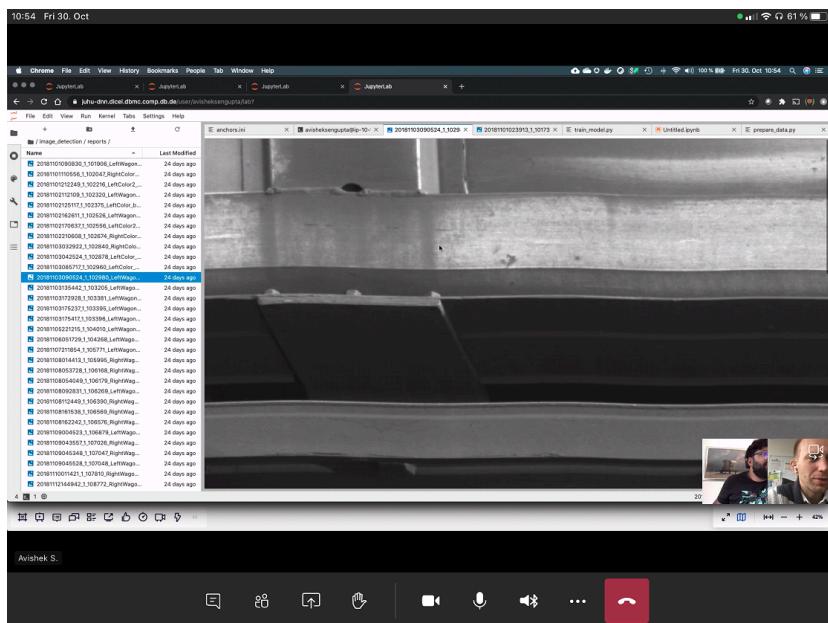
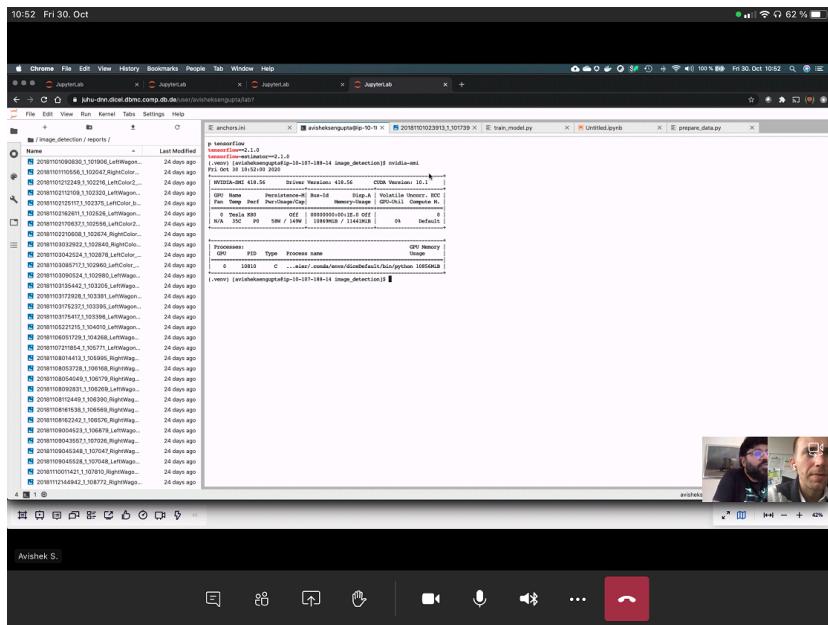
Uhrzeit: 10:30 Uhr

Person: Avishek Sen-Gupta

- Um ein AWS Account zu bekommen, benötigt es die Zustimmung eines Betriebsführungsteam. Ein solches Team kümmert sich um die Verwaltung, den Datenschutz und das die Abwicklung Konform läuft (Patches, Sicherheit, etc.). Damit ein Team selber, vision.AI, diese Verwaltung selber machen kann, bedarf es unteranderem einen Cloudführerschein.
- Linux Academy** vergibt AWS Test Konten.
- Ob eine Amazon Elastic Computing Cloud (EC2), **AWS.**, als virtuelle Datenverarbeitungsumgebung (Instance) gekauft werden, kann ist unklar.
- Der Aspekt mit Linux ist unklar; Hard-Client im DB System für zu Problemen; Hinter StageMaker läuft Linux
- Daniel Germanus ist Ansprechpartner, eine Umgebung über **Modular Cloud** zu bekommen.
 - House of AI Umgebung
 - KI-Lab
 - Jupyter Umgebung mit und ohne GPU







3.4.2 Beispiel Konto von Roberto

Die DB Modular Cloud (DB MC) ist ein Service der DB Systel. Dieser Service bietet skalierbare und selbstverwaltende Computer Plattformen durch AWS zur Verfügung zu stellen.

Das Angebot erstreckt sich auf

- Server-Varianten in verschiedenen Leistungsklassen (EC2).
 - Datenbank-Varianten ebenfalls in verschiedenen Leistungsklassen.

Auf der AWS Seite unter dem DB MC Konto, findet man die Angebote.

Produkte (21) Innenrechnung						
	Produktname	ID	Anbieter	Besitzer	Beschreibung	...
<small>Produktnamen</small>						
<input type="checkbox"/>	Backup - Database (RDS)	prod-c5d4jkpu8j	DB Modular Cloud	DB System GmbH		
<input type="checkbox"/>	Backup - Virtual Server (RSU)	prod-fvemrjhjnd	DB Modular Cloud	DB System GmbH	Create a backup of your virtual server at the current time.	
<input type="checkbox"/>	Datenbank (RDS) - MariaDB	prod-5t5eopun464g	DB Modular Cloud	DB System GmbH	Create a database - MariaDB (special features).	
<input type="checkbox"/>	Database (RDS) - MySQL	prod-7hewvqfhrvdy	DB Modular Cloud	DB System GmbH	Create a database - MySQL (special features).	
<input type="checkbox"/>	Database (RDS) - Oracle SE	prod-pur6vurh6c	DB Modular Cloud	DB System GmbH	Create a database - Oracle SE (special features).	
<input type="checkbox"/>	Database (RDS) - PostgreSQL	prod-p0d9qzqhpzpk	DB Modular Cloud	DB System GmbH	Create a database - PostgreSQL (special features).	
<input type="checkbox"/>	DBMC-Specific-Runtimes	prod-kvogqkldmky	DB Modular Cloud	DB System GmbH	Create customer specific runtimes.	
<input type="checkbox"/>	EC2 Auto Scaling (ASG)	prod-lockyqswkg	DB Modular Cloud	DB System GmbH	Provides an Auto Scaling group contains a collection of EC2 instances which can be treated as a logical grouping for the purposes of automatic scaling and management.	
<input type="checkbox"/>	Elastic File System (Amazon EFS)	prod-uklqktrntrwv	DB Modular Cloud	DB System GmbH	Provides a highly available, fully managed elastic NFS file system.	
<input type="checkbox"/>	Load Balancer	prod-uklqk721uclp	DB Modular Cloud	DB System GmbH	Create a load balancer including DNS-Addressing and SSL/TLS.	
<input type="checkbox"/>	Object Storage (S3-Bucket)	prod-q5lghqhz2wo	DB Modular Cloud	DB System GmbH	Creates an object store (S3 bucket) with access via a virtual server.	
<input type="checkbox"/>	Passwortcheck - User	prod-orwfr5t5k8juw	DB Modular Cloud	DB System GmbH	Resets your initial password.	
<input type="checkbox"/>	Security Group Rules	prod-spmed4skosua	DB Modular Cloud	DB System GmbH	You can add or remove rules for your security group, including port ranges, authorizing or revoking inbound access.	
<input type="checkbox"/>	Virtual Server (EC2) - AWS Linux	prod-pjtrqyptgl	DB Modular Cloud	DB System GmbH	Create a virtual server - AWS Linux (special features).	
<input type="checkbox"/>	Virtual Server (EC2) - AWS Linux 2	prod-ywbbdphdrfe	DB Modular Cloud	DB System GmbH	Create a virtual server - AWS Linux2 (special features).	

Roberto nutzt drei Produkte

Bereitgestellte Produkte (3) Informationen						
<input type="text" value="Bereitgestellte Produkte suchen"/> <input type="button" value="Zugrifffilter"/> <input type="button" value="Benutzer"/> <input type="button" value="Aktionen"/>				< 1 >		
Name	Erstellt	ID	Produktname	Versionname	Typ	Status
rfd_ab	Di, 15. Okt. 2019, 16:09:02 MESZ	pp-57itq7dqvw	Load Balancer	v1.4	CFN_STACK	Verfügbar
rFby	Mo, 20. Mai 2019, 15:49:50 MESZ	pp-zos4uhgx2nly	Security Group Rules	v11	CFN_STACK	Verfügbar
RFID_RegioBV	Mi, 9. Jan. 2019, 15:01:48 MESZ	pp-3t8uzhjgduuo	Virtual Server (EC2) - Windows	43.0.0	CFN_STACK	Verfügbar

Die Plattform nutzt die Amazon Pay-per-use-Ansatz.

Es gilt der Pay-per-use-Ansatz.	Das bedeutet: Sie zahlen nur so viel, wie Sie verbrauchen.
Unsere Preisindikatoren sind seit 06.08.2020 gültig.	 Wichtige Unterlagen
Preisübersicht Datenbanken (Single AZ)	Leistungsbeschreibung (DE) Stand 01.10.2020
Hinweis: Bei der Bereitstellung einer Multi-AZ Datenbank fallen die doppelten Kosten an (RDS-Preis aus der Übersicht mal zwei).	>jetzt herunterladen
> Download	
Preisübersicht Virtuelle Server	Preisübersichten für Datenbanken und Server 2020
Hinweis: Bitte beachten Sie, dass die Preise von den aktuellen AWS-Preisen abhängig sind und sich somit ständig, geringfügig ändern können.	>jetzt herunterladen
> Download	
Auto Scaling Group (ASG)	
Hinweis: Für das Produkt fallen keine zusätzlichen Kosten an.	
ALB-Preisindikatoren	Florian Widmann Product Owner DB System GmbH
Preis des Application Load Balancers (ALB) ist von mehreren Faktoren abhängig:	Weinmarktstr. 42-44 Raum C10094
- von der Laufzeit des ALBs sowie	 E-Mail
- von der voreinstellenden Menge der Load Balancer Capacity (LCU) pro Stunde.	 +49 69 265-7707
> Download	 DB Modular Cloud
- Im Dokument finden sich detaillierte Informationen zur Preisindikation und Rechenbeispiele für den ALB; erhalten Sie mit dem integrierten "ALB-Preisrechner" eine Preisindikation anhand der Parameter für den jeweiligen ALB.	
> Download	
Hinweis: Pro S3-Bucket wird eine fixe Gebühr von 5,00 € monatlich verbucht zzgl. der verbrauchten Speicherpreise (pay-per-use).	
Alle Anforderungen und Datenabrechnungen rufen Kosten hervor, die Sie direkt bei AWS nachlesen können.	
S3 Bucket (Simple Storage Service)	
Hinweis: Pro S3-Bucket wird eine fixe Gebühr von 5,00 € monatlich verbucht zzgl. der verbrauchten Speicherpreise (pay-per-use).	
Alle Anforderungen und Datenabrechnungen rufen Kosten hervor, die Sie direkt bei AWS nachlesen können.	
Beispielrechnung: 5,00 € (fix) + 0,0245 € (für die ersten 50 TB Daten) = 5,0245 €/Monat	
EFS Speicher (Elastic File System)	
Hinweis: Pro EFS Speicher wird eine fixe Gebühr von 5,00 € monatlich verbucht zzgl. der verbrauchten Speicherpreise (pay-per-use).	
Beispielrechnung: 5,00 € (fix) + 32 € (100 GB/Monat) = 37 €/Monat (zzgl. Backupkosten für EFS)	
LAMP-Stack Applikationsserver	
Hinweis: Es gilt der Pay-per-use-Ansatz. Das bedeutet: Sie zahlen nur so viel, wie Sie benötigen.	
Der Preis richtet sich nach den Preisen des virtuellen Servers und dem ALB.	
- EC2 (0,medium): ca. 57 € (4,00-17,00 Uhr) pro Monat	
- EC2 (0,medium): ca. 122€ bei 24x7 pro Monat	
- ALB (1,LCU3): ca. 52€ bei 24x7 pro Monat	

3.4.3 Anforderungen

- BEAM-ID
- APP-ID
- Prozess hat 3-4 Monate gedauert.

3.4.4 Dokumente

Dokumente

3.4.5 Schutzbedarfsfeststellung

Um jedoch DB MC nutzen zu können, muss eine Schutzbedarfsfeststellung (SBF) durchgeführt werden, **Ablauf**. Die nächsten Punkte sind nicht ganz klar.

- Es gibt eine Topdomäne und Domäne über die SBF zu erstellen.

	Topdomäne	Domäne
1	Topdomäne	
2		Marketing
3		Kundemanagement
4		Vertrieb
5	Kundeninteraktion	Produktmanagement
6		Kundenkommunikation & Information
7		Abrechnung
8		Energiehandel & Beschaffung
9		Eisenbahnbetrieb
10		Infrastrukturdienstleistungen
11		Energiebetrieb
12	Infrastruktur	Infrastrukturmangement
13		Fahrplankonstruktion
14		Bahnhofsbetrieb
15		Transportnachfrage
16		Transportdurchführung
17	Transport	Transportsteuerung
18		Logistik-/Transportzusatzdienstleistungen
19		Finanzen, Bilanzen & Controlling
20		Strategie, Organisation & Qualitätsmanagement
21		Technologie & Umweltschutz
22		Personalmanagement
23		Einkauf
24	Querschnitt	ITK-Management & Dienstleistungen
		Compliance, Recht,

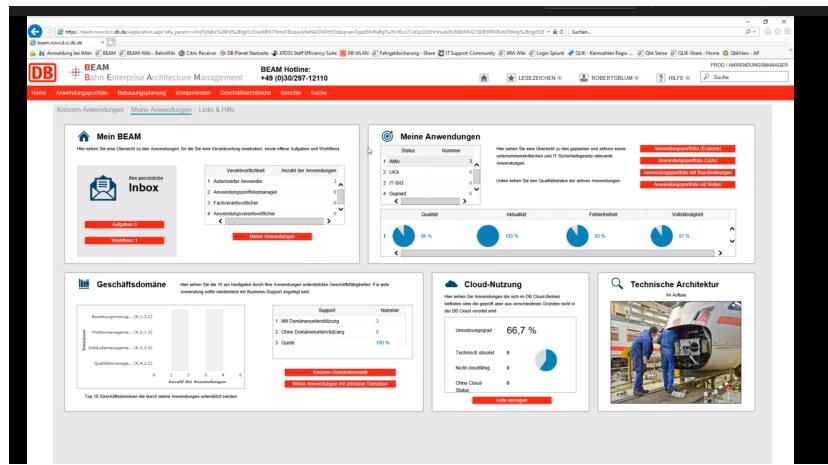
- Für die SBF wird die Checkliste benötigt, um die richtige Domäne zu finden.

	2.1.1 Produktmanagement
1.2.2 Fahrzeugeingang	
1.2.3 Abrechnung für Vermietung & Verkauf	
1.2.4 Service-Angebote	
1.3.1 Kunden- & Partnemanagement	
1.3.2 Berührungsmanagemen	
1.3.3 Kundenservice	
1.3.4 Kunden- & Partnernetzwerk	
1.4.1 Produktdaten	
1.4.2 Preis- & Tarifgestaltung	
1.4.3 Produktplanung & Entwicklung	
1.4.4 Produktions- & -austausch	
2.1.1 Strategisches Infrastrukturmangement	
2.1.2 Anlagenmanagement	
2.1.3 Infrastrukturdienstleistung	
2.1.4 Infrastrukturdienstleistung	
2.1.5 Energienmanagement	
2.1.6 Infrastrukturplanung	
2.2.1 Infrastrukturbearbeitung	
2.2.2 Infrastrukturdienstleistung	
2.2.3 Infrastrukturdienstleistung	
2.2.4 Fahrzeugplatzverteilung	
2.2.5 Operative Infrastrukturmangement	
2.2.6 Infrastrukturmangement	
2.2.7 Infrastrukturmangement	
2.2.8 Infrastrukturdienstleistung	
2.2.9 Infrastrukturdienstleistung	
2.2.10 Gebäudenagement	
2.2.11 Fahrzeugmanagement	
2.2.12 Fahrzeugspeisestock	
2.2.13 Personalentnahmung	
2.2.14 Ressourcenverteilung	
2.2.15 Ressourcenverteilung	
2.2.16 Kagnitivmanagement	
2.2.17 Energieverteilung	
2.2.18 Energieverteilung	
2.2.19 Energieninfrastrukturbetrieb & -disposition	
2.3.1 Auslastung	
2.3.2 Leistung	
2.3.3 Asset-Deckraum	
2.3.4 Kosten-Domäne	

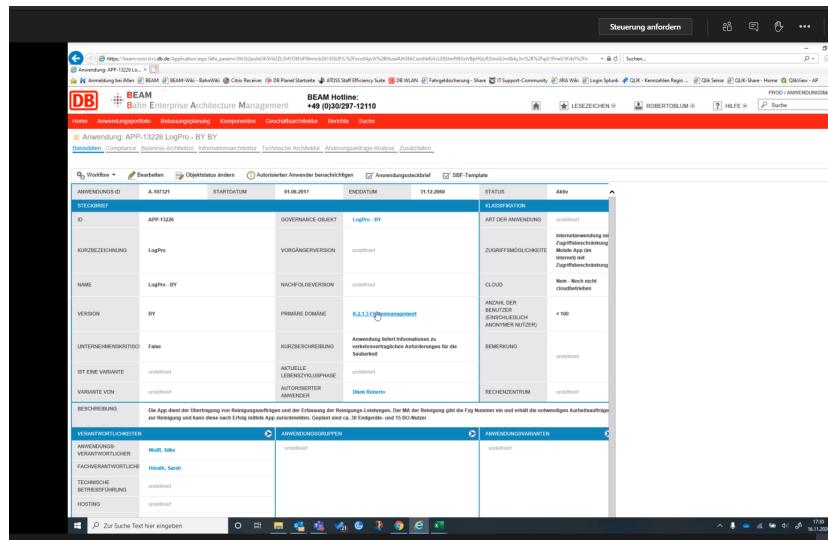
- Verantwortliche für eine BEAM Anwendung über iMAN. (Nutzerzugänger: Wer Zugänge zu meiner Anwendung hat.)

3.4.6 BEAM Konto

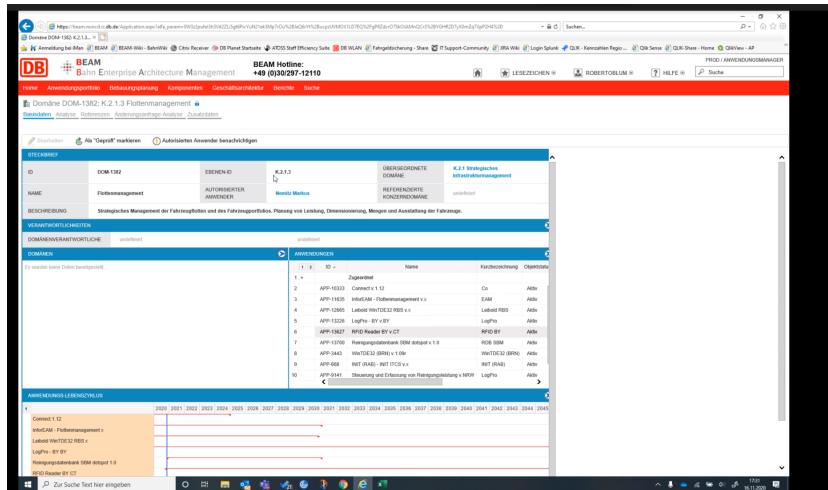
Das BEAM Konto bei Roberto sah wie folgt aus.



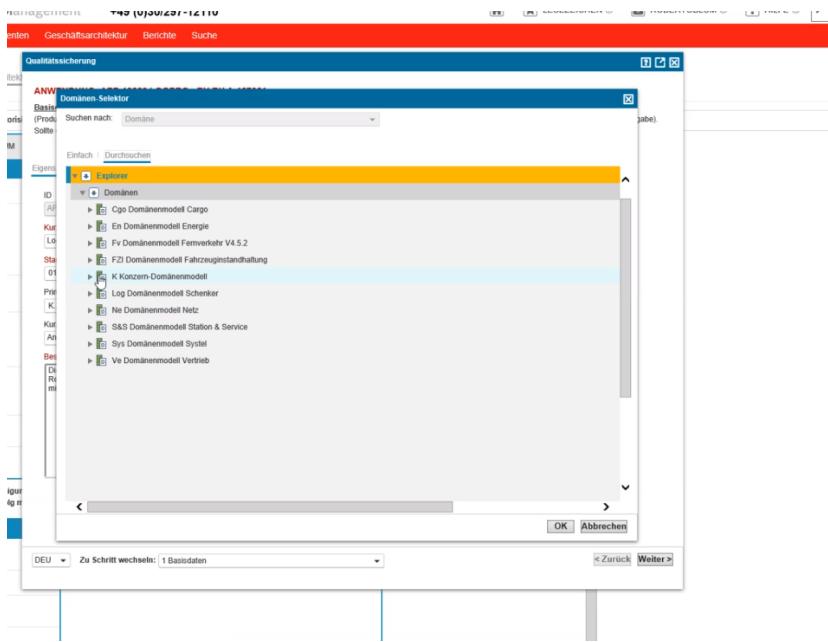
Die Anwendungen die hinterlegt werden müssen eine Primäre Domäne zugeordnet werden.



Über die Domäne kann eingesehen werden, welche Anwendungen darüber laufen.



Die Top Domänen lassen sich über den Domänen Selektor auswählen



3.4.7 Für was?

- Austausch mit Paigo. Der Datenaustausch mit Paigo soll zukünftig automatisch laufen. Der Anforderung ist, dass eine Secure File Transport Protocol (SFTP) Server angeboten wird. Über einen Simple Storage Services (S3) Bucket kann dies hergestellt werden, **SFTP for S3**.
- Berechnungsinstanz

iTime, gat Systeme (Dienstbestandteile) Ausschluss, Einschluss KSP, EDit, TerminalServer, isr ()

Teil II

Power BI

1 Working with M (Power Query) in Power BI

Funktion Eine **Funktion** ist ein mathematische Objekte, welches eine spezielle Relation, angewandt auf eine definierte Menge, ist. Dabei wir jedem Input genau ein Output zugewiesen. Um genau zusein, ist die Funktion die Abbildungsregel, das Objekte lautet Abbildung. Man nennt Abbildungen auch Funktionen, wenn sie ein ein Tupel aus Reelen Zahlen abbildet (Check). Beispiel: $f : \mathbb{R} \rightarrow \mathbb{R}$

Expression Eine **Expression** ist ein **algebraische** oder **syntakische Phrase**. In Power Query spricht man von Expression und meint damit ganze Zeilen Ausdrücke.

Value Es gibt verschiedene Datenstrukturen. Dabei können diese auch aus mehreren zusammengebaut sein.

Die Datentypen in Power Query sind ähnlich zu anderen Programmierungssprachen. Es gibt jedoch Ausnahmen. Ist kein Wert für eine Zelle vorhanden, so wird Type **Null**. Weiter Datentypen werden im Folgenden beschreiben. Hierbei werden spezielle Funktionen verwandt, um diesen Wert zu erzeugen. Dabei werden bestimmte Syntaxen verlangt.

1.0.1 Variable

Variablen werden bei erzeugen in *M* nicht deklariert. Eine automatische Zuweisung der Typen funktioniert über die Syntax. Für Funktionen können jedoch Typen für Inputvariablen und die Funktion selbst definiert werden.

Kind	Literal
<i>Null</i>	null
<i>Logical</i>	true false
<i>Number</i>	0 1 -1 1.5 2.3e-5
<i>Time</i>	#time(09,15,00)
<i>Date</i>	#date(2013,02,26)
<i>DateTime</i>	#datetime(2013,02,26, 09,15,00)
<i>DateTimeZone</i>	#datetimezone(2013,02,26, 09,15,00, 09,00)
<i>Duration</i>	#duration(0,1,30,0)
<i>Text</i>	"hello"
<i>Binary</i>	#binary("AQID")
<i>List</i>	{1, 2, 3}
<i>Record</i>	[A = 1, B = 2]
<i>Table</i>	#table({ "X", "Y"}, {{0,1}, {1,0}})
<i>Function</i>	(x) => x + 1
<i>Type</i>	type { number } type table [A = any, B = text]

Abbildung 1:

[Link](#)

1.0.2 Variable name

Variablen werden in Power Query, wie auch in anderen Sprachen, mit Hilfe einer Zeichenkette definiert. Es gibt jedoch die Besonderheit, das Variablennamen auch mit einem Leerzeichen definiert werden dürfen. Dafür muss der Name als String gesehen werden und mit einem # am Anfang versehen werden.

```
1 let
2   #"Das ist wirklich ein Variablenname!" = 13,
3   y = #"Das ist wirklich ein Variablenname!"
4 in
5   y // Returns 13
```

1.0.3 Zuweisen / Syntax

Für das Zuweisen eines Types zu einer Variablen wird der **as** Ausdruck verwandt. Wird eine Variable erzeugt ohne eine Funktion zu verwenden, erfolgt eine automatische Type-Zuweisung.

```
1 let
2   y = 1.3234 // type number
3   #"Variable 2" = "Text" // type text
4   #"Variable 3" = #date(1,1,1908) // Zuweisung erfolgt durch die intrinsische Funktion date
```

Der **as** Ausdruck wird verwendet, um Input und Output von Funktionen zu definieren.

```
1 let
2   Funktion1 = (Input as number) as number => 2 + 3
3 in
4   Funktion1
```

Wollen wir einen Typen einer Spalte einer Tabelle ändern, so nutzen wir die

Table.TransformColumnTypes
8/1/2019 • 2 minutes to read

Syntax

```
Table.TransformColumnTypes(table as table, typeTransformations as list, optional culture as nullable text) as table
```

About

Returns a table from the input `table` by applying the transform operation to the columns specified in the parameter `typeTransformations` (where format is {column name, type name}), using the specified culture in the parameter `culture`. If the column doesn't exist, an exception is thrown.

Example 1

Transform the number values in column [a] to text values from the table `(([a = 1, b = 2], [a = 3, b = 4]))`.

```
Table.TransformColumnTypes(Table.FromRecords(([{"a": 1, "b": 2}, {"a": 3, "b": 4}]), {"a", type text}, "en-US")
```

Abbildung 2:

Die Funktion lässt eine Tabelle als Input ein. Für die Transformation wir eine *Liste* benötigt. Diese benötigt jedoch für jeden Eintrag selber eine Liste, mit dem ersten Eintrag den Spaltennamen und den zweiten den zu bezeichneten Typ. Beispiel:

```
let
    Quelle = {1,2,3,5, "absa"}, 
    Quelle2 = #table({"A","B"},{{1,2},{1,3},{2,2}}),
    function = (table) => table[A],
    table2 = function(Quelle2)
in
    table2
```

Abbildung 3: Eine Spalte

Abbildung 4: Mehrere Spalten

Achtung Es können verschiedenen Syntaxen verwendet werden um eine Typ zu definieren.

Data Type	Syntax 1	Syntax 2
Whole Number	Int64.Type	-
Decimal Number	Number.Type	type number
Dates	DateType	type date
Text	Text.Type	type text
Binary		type binary
Date/Time		type datetime
Date/Time/Timezone		type datetimezone
Duration		type duration
Function		type function
List		type list
True/False		type logical
Record		type record
Any		type any
Any Non-Null		type anynonnull
None		type none
Null		type null
Type		type type

Abbildung 5: Mehrere Spalten

Dies sieht man auch im Beispiel mit mehreren Spalten. Die Integer Typumwandlung erfolgt mit dem "Int64.Type" dieser wird jedoch nicht "type Int64" geschrieben. Für andere Typen kann diese Logik aber angewandt werden.

1.0.4 number, text, logical

Variablen müssen

```

1 let
2   x = 5,
3   y = x + 5,
4   w = "Test Text",
5   z = true,
6 in
7   x // Returns

```

1.0.5 null

Variablen besitzen eine Ausprägung für einen Wert. Das Konzept von "any" und "null" wird hier nicht berücksichtigt. Diese typen können ebenfalls abgerufen werden. Dabei kann eine Funktion eine Input auf den type testen.

```

1 let
2   y = null // y ist vom Typ null
3   w = "null" // w ist vom Typ text
4   z = 0 // z ist vom Typ number
5 in
6   w // Return null

```

1.0.6 Type

Der Type **type** wird wie intrinsische Varialben mit einer Prefix deklariert. Der Prefix lautet "type" und der Variabletype wird in { }. Für Tabellen können die Typen für die Spalten gleich mit im Tabellenkopf vermerkt werden.

```

1 let
2   x = type {number}, // Die Variable x ist vom type type.
3   y = #table(
4     {Column1, Column2}, // Kopf der Tabelle ohne Typen
5     {
6       {2, "text"},      // Spalte 2, Wert text
7       {4, "hello"}    // Spalte 4, Wert hello

```

```

8         }
9     ),
10
11    w = #table(
12        type table [Column1 = type number, Column2 = type text],
13        \\ Kopf der Tabelle mit Typen
14        {
15            {2, "text"},
16            {4, "hello"}
17        }
18    )
19
20 in
21 y

```

1.0.7 Intrinsische Werte

Um Werte wie *date*, *duration*, *binary* etc. zu schaffen, werden spezielle **single intrinsic function** benötigt. Man erkennt diese daran, dass vor ihnen ein # steht.

```

1 let
2   x = #date(2019,2,2)
3 in
4   x // Returns 2019-2-2

```

1.1 Strukturierte Variablen

Es gibt drei verschiedene Datentypen, welche aus mehreren Komponenten bestehen
erden. [Link](#), [Link](#)

```

1 let
2   x = {a,2,abc} // List
3   y = [ // Record
4       variable1 = 1,
5       Item = "abc",
6       ID = 4
7   ],
8   w = #table(
9       {"Column1", "Column 2" },
10      {
11          {2, 4},
12          {"abs", 5},
13          {123j,#date(2019,2,2)}
14      }
15 in
16   x{3} // Returns abc
17   // y[Item] returns abs
18   // w{2}[Column1] returns 123j; 2 ist die zweite Zeile (Start 0)

```

1.1.1 List

Eine Liste ist ein einfaches Array, welches alle verschiedenen Typen von Objekten beinhalten kann.

1.1.2 Record

Ein Record ist ein Array, welches zu jedem Eintrag eine Indexnamen ausweisen kann. Es können auch Records von Records gemacht werden.

1.1.3 Table

Eine Tabelle teilt sich in den Tabellenkopf und die Zeilen auf. Um eine Table zu erstellen, muss die intrinsische Funktion `#table()` angewandt werden.

Bei der Erstellung der Tabellen, können gleich **typen** von Datenformaten gewählt werden, siehe Zuweisen/Syntax.

1.1.4 Beispiele

```

1 let
2   Source =
3 {
4   1,
5   "Bob",
6   DateTime.ToString(DateTime.Now(), "yyyy-MM-dd"),
7   [OrderID = 1, CustomerID = 1, Item = "Fishing rod", Price = 100.0]
8 }
9 in
10 Source

```

Das Result ist:

A List containing a Record	
1	
"Bob"	
2015-05-22	
OrderID	1
CustomerID	1
Item	"Fishing rod"
Price	100.0

Abbildung 6:

1.2 Function

Eigenen Funktionen können ebenfalls geschrieben werden. Es ist jedoch die sequenzielle Natur des M-Codes zu berücksichtigen. Eine Besonderheit zur Sprache C++ ist, dass die Typen für Input und Output Variablen nicht dekliniert werden müssen. Es wird dann von **impliziten Funktionen** gesprochen. Der Tye *any* wird dann als *Default* genutzt:

1.2.1 Implicite function

Name der Funktion = (Inputvariablen) => Expression, (1.1)

```

1 let
2   Add = (x, y) => x + y,
3   AddResults =
4   [
5     OnePlusTwo = Add(1,2) // Ergebnis ist 3
6     OnePlusOne = Add(1,1) // Ergebnis ist 2
7   ]
8 in
9   AddResults
/* 
10  OnePlusTwo 3
11  OnePlusOne 2
*/

```

1.2.2 Explicite function

Bei dieser Art von Funktionen müsse die Typen für die Input- wie Outputvariablen angegeben werden. Dies erfolgt im zweiten Block.

Name der Funktion = (Inputvariablen as "type") as "type" => Expression, (1.2)

Mit etwas geschickt, können auch komplexere Funktionen gebaut werden.

```
1 let
2   GreaterThanFive = (list) =>
3     let
4       OrderedList = List.Select(list, each _ >= 5)
5       /*
6       List.Select(list as list, selection as function) as list
7       */
8       Result = List.First(OrderedList)
9     in
10    Record = [
11      Ergebnis1 = GreaterThanFive({1,2,6}) // Returns 6
12      Ergebnis2 = GreaterThanFive({12,12,45,23}) // Returns 12
13    ]
14  in
15  Record
```

1.2.3 Bewertungen (Conditions)

Es ist essentielle für das Verstehen der meisten Funktionen, wie Tabllen, Listen, Records abgerufen und Teilmengen der Informationen selektiert werden können.

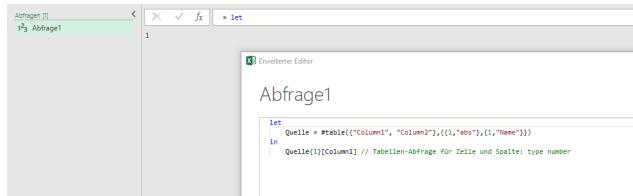


Abbildung 7: Tabelle - Zeile und Spalte (type cell-type)

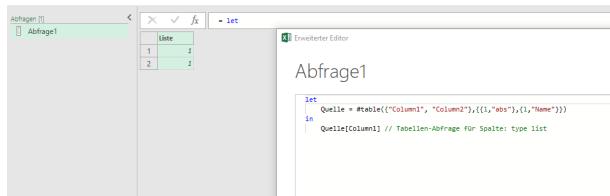


Abbildung 8: Tabelle - Spalte (type list)

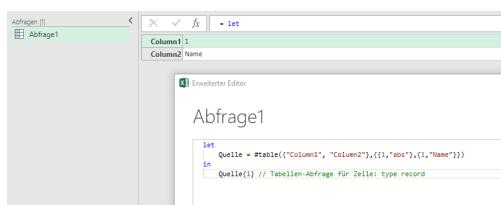


Abbildung 9: Tabelle - Zeile (type record)

Table

The screenshot shows the Power Query Editor interface. On the left, there's a preview pane with a single row labeled 'Abfrage1'. On the right, the 'Erweiterter Editor' (Advanced Editor) pane displays the following code:

```

let
    Quelle = [Eintrag1 = 1, Eintrag2 = "ABA"]
in
    Quelle{1}

```

A tooltip window titled 'Fehler in der Abfrage' (Error in query) is open over the 'Eintrag1' field, stating 'Details: Value = Eintrag1 Type=Type'.

Abbildung 10: Record - Zeile : Geht nicht!)

The screenshot shows the Power Query Editor interface. On the left, there's a preview pane with a single row labeled 'Abfrage1'. On the right, the 'Erweiterter Editor' (Advanced Editor) pane displays the following code:

```

let
    Quelle = [Eintrag1 = 1, Eintrag2 = "ABA"]
in
    Quelle{#Eintrag1}

```

Abbildung 11: Record - Eintrag [] (type cell-type
; Wichtig: Table.SelectRows()

Record

The screenshot shows the Power Query Editor interface. On the left, there's a preview pane with a single row labeled 'Abfrage1'. On the right, the 'Erweiterter Editor' (Advanced Editor) pane displays the following code:

```

let
    Quelle = { 1, "ABA" }
in
    Quelle{1}

```

Abbildung 12: Liste - Zeile (type cell-type)

List

Eine Funktion kann als Outputwert vom *type logic* sein, sie kann aber auch jeden anderen Wert annehmen. Wichtig ist jedoch, wird eine Logische Abfrage im Funktionskörper veranlasst und diese nicht verarbeitet, so nimmt die Funktion den Typen *logic* an.

Eine Funktion kann, wenn ihr eine Tabelle übergeben wird, eine Liste wieder geben, wenn aus der Tabelle eine Spalte mit `[]` ausgelesen wird, siehe 3. Wird die ausgelesene Liste verglichen mit einem einzelnen Wert, so wird die logische Abfrage *false* wiedergeben, da die beiden typen nicht gleich sind.

The screenshot shows the Power Query Editor interface. On the left, there's a preview pane with a single row labeled 'Abfrage1'. On the right, the 'Erweiterter Editor' (Advanced Editor) pane displays the following code:

```

let
    Quelle = {1,2,3,5, "abc"},
    Quelle = #table({"A","B"},{{1,2},{3,2},{2,2}}),
    function = (table) => table[A]=>,
    table2 = function(Quelle)
in
    table2

```

Abbildung 13:

Selbst wenn jeder Eintrag in der List den gleichen zuüberprüfenden Eintrag hat, gibt die Funktion *false* wieder

```

let
    Quelle = {1,2,3,4, "abs"},
    Quelle2 = #table([{"A","B"},{{2,2},{3,3},{2,2}}], function = [table] => table[A]{1}{2},
    table2 = function(Quelle2)
in
    table2

```

Abbildung 14:

Wird ein Abgleich mit einer List gemacht, welche die gleichen Einträge beinhaltet, so gibt die Funktion *true* wieder.

```

let
    Quelle = {1,2,3,4, "abs"},
    Quelle2 = #table([{"A","B"},{{2,2},{3,3},{2,2}}], function = [table] => table[A]{1}{2},
    table2 = function(Quelle2)
in
    table2

```

Abbildung 15:

Es sei vermerkt, es werden nicht die Typen abgefragt, sondern jeder Eintrag in der Liste wird verglichen.

```

let
    Quelle = {1,2,3,5, "abs"},
    Quelle2 = #table([{"A","B"},{{2,2},{3,3},{2,2}}], function = [table] => table[A]{1}{2},
    table2 = function(Quelle2)
in
    table2

```

Abbildung 16:

Im Abschnitt **each** wir über die eine spezielle Funktion gesprochen.

1.3 Spezielle

1.3.1 Empty Variable

Um den Ausdruck **each** verstehen zu können, sollte die Leere-Variable besprochen werden. In Python als auch M wir _ für Puffervariablen genutzt, welche später nicht mehr gebracht werden.

```

1 let
2   _ = Table.FromColumns({
3     {"Fred", "Mary", "Phil", "Sue"},
4     {1, 2, 3, 4},
5     {5, 2, 5, 1}},
6     {"Person", "Index", "Score"} // Tabellenkopf
7   )
8 in
9   _ // Die temporäre Variable heißt _ und ist vom Type table.

```

Um verschiedenen Daten von der Tabelle abzurufen, nutzen wir die Indexschreibweise:

$$\text{Tabellen-Variablen-Namen} \{ \text{Zeile} \} [\text{Spaltennamen}] \quad (1.3)$$

Für den Variablennamen _ gilt dies natürlich auch:

```

1 let
2   _ = Table.FromColumns({

```

```

3     {"Fred", "Mary", "Phil", "Sue"},  

4     {1, 2, 3, 4},  

5     {5, 2, 5, 1}},  

6     {"Person", "Index", "Score"} // Tabellenkopf  

7   )  

8 in  

9   _[Person] // Return Gesamte Spalte Person vom type list  

10  // _{2}[Person] // Return Zeile 2 der Spalte Person

```

Den temporären Variablennamen können wir auch weglassen, wenn Tabellen angesprochen werden. Dieser Prozess sollte für Listen nicht verwendet werden, da Power Query einen Zeilenverweis für Listen als neue Liste interpretiert:

```

1 let  

2   _ = {"ab", null, 6}  

3 in  

4   _{2} // Return null  

5   // _{2} // Erstellen einer weiteren Liste

```

Für Tabellen kann jedoch der `_` weggelassen werden.

```

1 let  

2   _ = Table.FromColumns({  

3     {"Fred", "Mary", "Phil", "Sue"},  

4     {1, 2, 3, 4},  

5     {5, 2, 5, 1}},  

6     {"Person", "Index", "Score"} // Tabellenkopf  

7   )  

8 in  

9   [Person] // Return Gesamte Spalte Person

```

1.3.2 Each

Die Funktion `each` ist eine einfache Funktion, welche die Syntax der Leeren Variable nutzt. Es gilt:

$$\text{each } x := (_) \Rightarrow _x \quad (1.4)$$

Der Begriff `each` ist vielleicht etwas irreführend, da nicht wirklich ein *loop*-Durchlauf gestartet wird. Vielmehr wird die `each`-Funktion genutzt, wenn die ummantelnde Funktion den loop-Durchlauf startet und `each` eine Überprüfung für jede der übergebenen Element machen soll.

Es ist sehr wichtig zwischen Tabellen und Records zu unterscheiden. Wir der `each`-Funktion eine Tabelle übergeben und sie soll bewerten, ob eine Wert in jeder Zeile steht, funktioniert dies nicht mit einem einzelnen Wert

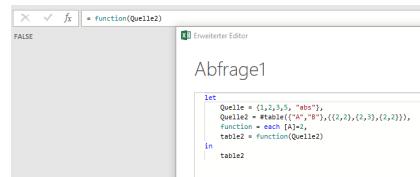


Abbildung 17:

Auch hier gilt, es muss mit einer Liste abgeglichen werden.

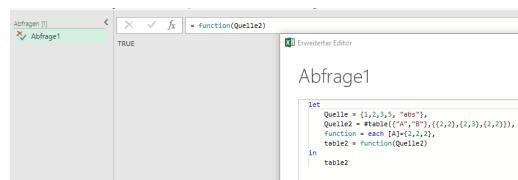


Abbildung 18:

Damit each *true* zurückgeben kann, muss die geprüfte List ebenfalls die richtigen Wert enthalten.

```

let
    Quelle = {1,2,3,4, "abc"}, 
    Quelle2 = #table({{"A","B"}},{{1,2},{2,3},{2,2}}),
    function = each [A]=2,
    table2 = Table.SelectRows(Quelle2, each [A]=2)
in
    table2

```

Abbildung 19:

Wird die Funktion **Table.SelectRows()** eingesetzt, so wird nicht eine Tabelle, sondern ein Record für die zu überprüfenden Zeile übergeben.

```

let
    Quelle = {1,2,3,4, "abc"}, 
    Quelle2 = #table({{"A","B"}},{{1,2},{2,3},{2,2}}),
    function = each [A]=2,
    table2 = Table.SelectRows(Quelle2, each [A]=2)
in
    table2

```

Abbildung 20:

Der *each* Operator nimmt das übergebenen Record-Element, welches aus der Tabelle extrahiert wurde und überprüft mit der Eintragssyntax `[]`, die auch die Spaltensyntax für Tabellen ist, den Wert der für die Zeile in der entsprechenden Spalte steht. Eine einfache logische Abfrage entscheidet dann, ob der Eintrag zu "jeder" Zeile in der entsprechenden Spalten richtig ist. Wenn ja, dann wird die Zeile ausgewählt, wenn nein, dann wird die Zeile gelöscht.

In der *Table.SelectRows()* Funktion wird jede Zeile als Record übergeben. Wie oben beschrieben, wird die Auswahl pro Zelleninhalt ausgewertet.

```

let
    Quelle = {1,2,3,4, "abc"}, 
    Quelle2 = #table({{"A","B"}},{{1,2},{2,3},{2,2}}),
    Quelle4 = [A=2],
    function = each [A]=2,
    table2 = Table.SelectRows(Quelle2, each [A]=2), // Erstelle Records, nicht Tabellen oder Listen!!!
in
    table2

```

Abbildung 21:

1.3.3 try otherwise

Das *try ... otherwise* Statement ist für besonders geeignet, um Fehler abzufangen. Wie das *if*-Statement werden zwei Szenarien festgelegt, welche ausgeführt werden, nach der Überprüfung einer Bedingung.

Das *try-otherwise* Statement fängt jedoch *Fehler* ab. Die Bedingung ist somit, wenn die Bedingung (Funktion) einen Fehler produziert, dann wird *otherwise* ausgeführt.

2 Business Intelligence: Part I - Power Query

2.1 Introduction to Excel Tools

Im ersten Teil des Kurses wird besprochen welche Methoden es in der Microsoft Power BI Umgebung gibt. In der Business Version sind einzelne Komponenten integriert, welche vollintegriert in der BI-Version sind. In der Übersicht wird gezeigt, wie die einzelnen Komponenten aufgebaut sind.

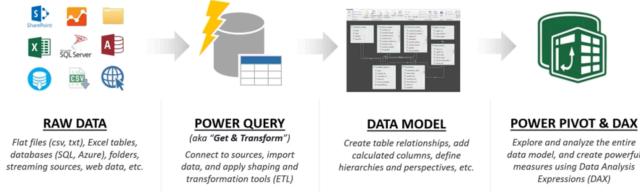


Abbildung 22:

- Datenbank**
- Access wird mehr als Datenbank gesehen. Welche Fragen noch offen sind, im Hinblick auf Power Query, ist: "Wie unterscheiden sich Access Queries von Power Query-Abfragen?".
 - Weitere Datenspeicherungen können in comma-separated values (.csv) oder Microsoft Excel Open XML Spreadsheet (.xlsx) Dateien liegen.
 - Es können aber auch Server wie SQL Server oder SharePoint angebunden werden.

Query Mit Hilfe der Abfragen können spezifische Fragen an die eingelesenen Daten gestellt werden. Diese werden in Excel geladen. Entweder werden die Daten direkt in Excel eingelesen und oder über Pivot Tabellen / Charts dargestellt oder sie können in das Datenmodell eingelesen werden. Per Query wird mit Prinzip Extract, Transform, Load (ETL) gearbeitet.

Datenmodell In dem Datenmodell werden Beziehungen zwischen verschiedenen Datenquellen oder spezifischer Tabellen bezogen. Die Daten werden mit einem speziellen Komprimierungsverfahren gespeichert. Gespeichert werden die Daten im Arbeitsspeicher, für die weitere Verarbeitung. Ergänzend zur Queries durch Power Query können auch weitere Spalten angefügt werden. Es ist wichtig zu vermerken, dass nur die Daten im Datenmodell gespeichert werden, die entweder selektiert durch die Query eingelesen wird oder direkt eingespeichert wird.

PowerPivot Dieses Option umfasst nicht nur die dynamischen Pivottabellen sondern auch die **Measures**. Power Pivot verwaltet nicht nur eine einfache Tabelle, sondern greift auf das komplette Datenmodell zu. Mit Measures können BI-Funktionen geschaffen werden. Diese werden mit Data Analysis Expressions (DAX) erstellt.

2.2 Query Editor - Tabellenblätter

Power Query bietet verschiedene Möglichkeiten an, Daten in Excel zu laden.

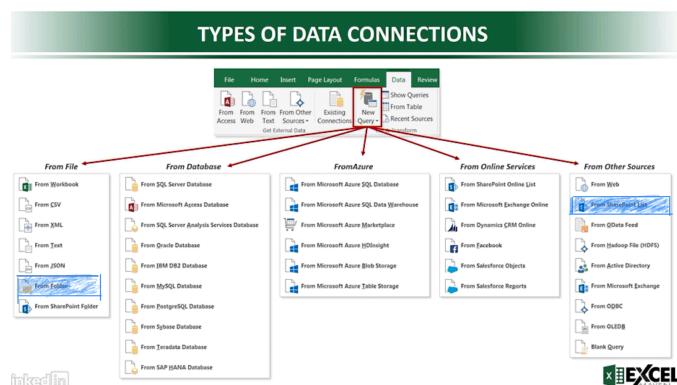


Abbildung 23:

Im ersten markierten Bereich wird drauf verwiesen, einen Pointer auf einen Ordner zeigen zu lassen. Ebenso ist es möglich, eine SharePoint-Seite anzusprechen.

Der Editor ist in drei Bereiche eingeteilt:

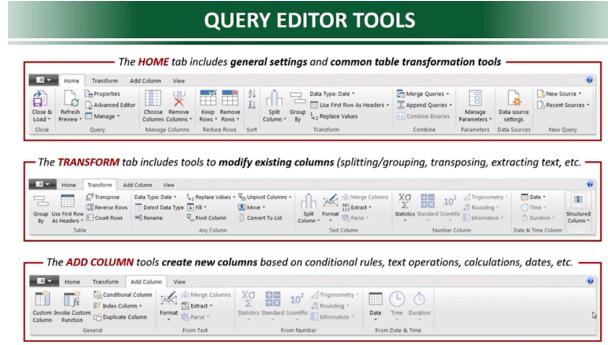


Abbildung 24:

- Home: Die Daten sind geladen und werden transformiert.
- Transform: Die Daten sind geladen und werden transformiert & verändert.
- Add Column: Die Daten sind geladen und werden transformiert & neue Daten werden angefügt.

2.2.1 Home

Remove Duplicates Diese Funktion ist nützlich, wenn man dynamische Indexfunktionen erstellen will. Zum Beispiel, kann für die

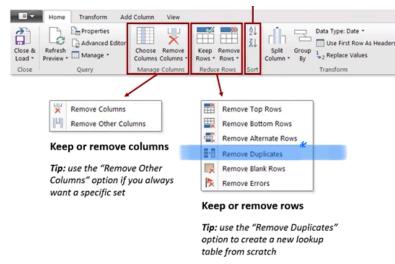


Abbildung 25:

Proper Name Es wird empfohlen die Queries auch als solche zu bezeichnen. Zum Beispiel "ETLName". Dies wird sich (vermutlich) bei Power Pivot erschließen.

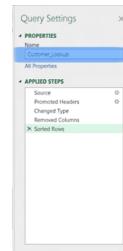


Abbildung 26:

2.2.2 Transform

Text Transformation In dem Bereich unter dem Tab können Text-Einträge verändert werden. Die Funktionalität richtet sich vorrangig an Spalten vom String Typ.

- Trim - Leerzeichen vor dem Eintrag können gelöscht werden.
- Prefix - Textteile können angefügt werden.

- Groß- und Kleinschreibung kann angepasst werden
- Beispiel: Die Namen in der Liste können angepasst werden, falls die Vor- und Nachnamen nicht richtig großgeschrieben sind.

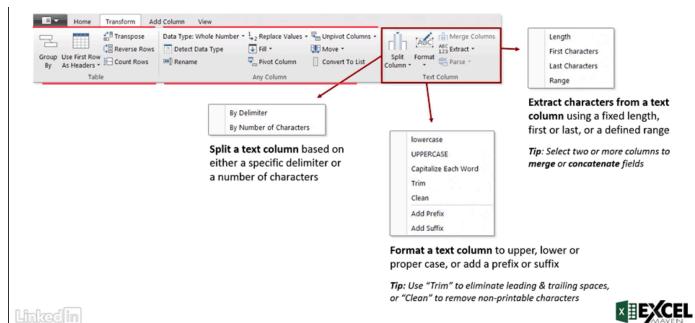


Abbildung 27:

Zahlen Transformation

- Statistiken geben Werte zurück. Eine oder mehrere Spalten werden reduziert auf einen Wert. Die Statistik-Funktionen werden nicht in dem *Add Column* Tabellenblatt angezeigt. **Idee:** Statistik können bestimmt werden und per Verbindung gespeichert werden. Daraufhin werden diese in Power Pivot verwendet, um mit ihnen weiter zu rechnen. Den Wert kann in eine *Liste* oder eine *Tabelle* konvertieren.

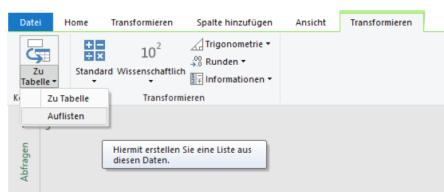


Abbildung 28:

- Standard, Wissenschaftliche oder andere Funktion für numerische Spalten lassen sich auf Spalten anwenden

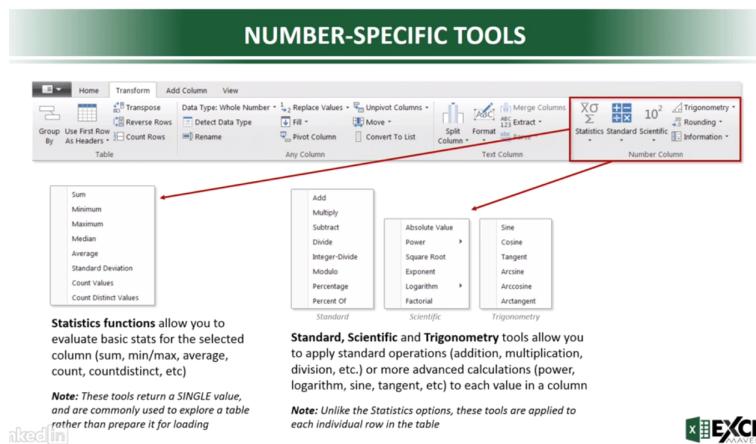


Abbildung 29:

- **List Tool** Die Werte der Tabelle können nicht direkt verändert werden. Es kann nicht einfach in eine Zelle gegangen werden und ein Wert verändert werden. Mit List können aber ganz neue Spalten, von einer Satt aus, erstellt werden.

- Power M - List - Funktionen (siehe Dokument)

The screenshot shows the title 'List functions' and a subtitle '8/6/2019 • 8 minutes to read'. Below this, a paragraph explains that the Power Query Formula Language (informally known as "M") is a powerful **mashup query language** optimized for building queries that mashup data. It is a functional, case sensitive language similar to F#, which can be used with [Power Query](#) in Excel and [Power BI Desktop](#). To learn more, see the [Power Query Formula Language \(informally known as "M"\)](#).

Information	
FUNCTION	DESCRIPTION
List.Count	Returns the number of items in a list .
List.NonNullCount	Returns the number of items in a list excluding null values
List.IsEmpty	Returns whether a list is empty.

Selection	
FUNCTION	DESCRIPTION
ListAlternate	Returns a list with the items alternated from the original list based on a count, optional repeatInterval, and an optional

Abbildung 30:

- Auch hier kann man keine einzelnen Zellen verändern.

Wie kann man eine neue Tabelle in Power Query anlegen?

- Offen *Blank Power Query*
- Mit der Funktion = #date(2017, 1, 1) wird der erste Eintrag in die Liste gemacht.

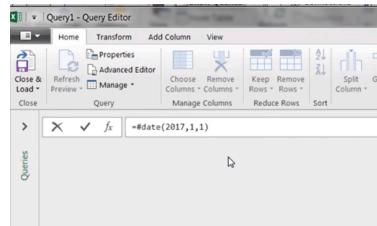


Abbildung 31:

Variablen werden in Power Query mit einem # begonnen.

```

let
    source = Web.Page(Web.Contents("http://www.imdb.com/chart/top"))
    Data0 = Source{0}[Data]
    #"Changed Type" = Table.TransformColumnTypes(Data0,{{"", type text}, {"Rank & Title", type text}, {"IMDb Rating", type number}})
    #"Removed Other Columns" = Table.SelectColumns(#"Changed Type",{"Rank & Title", "IMDb Rating"})
    #"Split Column by Delimiter" = Table.SplitColumn(#"Removed Other Columns", "Rank & Title", Splitter.SplitTextByEachDelimiter({";"}, 1, 1))
    #"Changed Type1" = Table.TransformColumnTypes(#"Split Column by Delimiter",{{"Rank & Title.1", Int64.Type}, {"Rank & Title.2", type text}, {"IMDb Rating", type number}})
    #"Renamed Columns" = Table.RenameColumns(#"Changed Type1",{{"Rank & Title.1", "Title"}, {"Rank & Title.2", "Text"}, {"IMDb Rating", "Rating"}, {"Renamed Column1", "Type64"}})
    #"Split Column by Delimiter1" = Table.SplitColumn(#"Renamed Columns", "Text", Splitter.SplitTextByEachDelimiter({";"}, 1, 1))
    #"Changed Type2" = Table.TransformColumnTypes(#"Split Column by Delimiter1",{{"Text", type text}, {"Rating", Int64.Type}})
    #"Replaced Value" = Table.ReplaceValue(#"Changed Type2","", "", Replacer.ReplaceText, {"Rank & Title.2.2"})
    #"Changed Type3" = Table.TransformColumnTypes(#"Replaced Value",{{"Rank & Title.2.2", Int64.Type}})
    #"Renamed Column1" = Table.RenameColumns(#"Changed Type3",{{"Rank & Title.2.2", "Year"}, {"Rank & Title.2.1", "Title"}, {"Rating", "Type64"}})
in
    #"Trimmed Text"

```

Abbildung 32:

Einfache Rechenoperationen sind auch möglich.

3 Business Intelligence: Part II - Data Modeling 101

3.1 Data Normalization

Um Datenmodell aufzubauen, ist es sinnvoll, dem Prinzip der **Separation** zu folgen. Daten die auf Sachverhalte mit verschiedenen Entstehungsgründen zurückzuführen sind, sollten auch in separate Tabellen stehen.

date	product_id	quantity
1/1/1999	869	5
1/1/1999	1472	3
1/1/1999	76	4
1/1/1999	320	3
1/1/1999	4	4
1/1/1999	952	4
1/1/1999	3222	4
1/1/1999	357	4
1/1/1999	3359	4
1/1/1999	3426	5
1/1/1999	190	4
1/1/1999	367	4
1/1/1999	250	5
1/1/1999	600	4
1/2/1999	702	5

day_of_month	month	year	weekday	week_of_year	month_name	quarter
1	1	1997	Wednesday	1	January	Q1
2	1	1997	Thursday	1	January	Q1
3	1	1997	Friday	1	January	Q1
4	1	1997	Saturday	1	January	Q1
5	1	1997	Sunday	1	January	Q1
6	1	1997	Monday	2	January	Q1

product_id	product_brand	product_name	product_sku	product_qty	product_desc	product_uom
1	Washington	Washington Berry Juice	90748503814	2.05	0.94	8.39
2	Washington	Washington Mango Drink	5842777305	0.83	0.4	11.1
3	Washington	Washington Cream Soda	5983352293	2.05	1.64	10.5
4	Washington	Washington Diet Soda	8561139469	2.10	0.77	6.56
5	Washington	Washington Cola	2080464796	1.15	0.37	11.8
6	Washington	Washington Lemonade	3000000000	2.05	0.93	1.8
7	Washington	Washington Orange Juice	8870532250	2.05	0.8	8.57

Abbildung 33:

In dem Beispiel sieht man, das die ergänzenden Daten zu dem Produkt in einer separaten Tabelle gespeichert werden sollte.

Wir können ebenfalls unterscheiden, zwischen **Daten Tabellen** und **LookUp table**. Was als Vermerk angebracht wird, ist sich Kalender Tabellen zu generieren. Dies ist deshalb sinnvoll, weil Power Pivot kein so leichte Oberfläche hat, wie Power Query um Daten zu extrahieren (Bis jetzt).

3.2 Data Model

3.2.1 Primary and Foreign Key

The primary key serves as a unique identifier. The foreign key allows to connect to the primary key.

date	product_id	quantity
1/1/1999	869	5
1/1/1999	1472	3
1/1/1999	76	4
1/1/1999	320	3
1/1/1999	4	4
1/1/1999	952	4
1/1/1999	3222	4
1/1/1999	517	4
1/1/1999	3359	4
1/1/1999	337	4
1/1/1999	3426	5
1/1/1999	190	4
1/1/1999	367	4
1/1/1999	250	5
1/1/1999	600	4
1/2/1999	702	5

day_of_month	month	year	weekday	week_of_year	month_name	quarter
1	1	1997	Wednesday	1	January	Q1
2	1	1997	Thursday	1	January	Q1
3	1	1997	Friday	1	January	Q1
4	1	1997	Saturday	1	January	Q1
5	1	1997	Sunday	1	January	Q1
6	1	1997	Monday	2	January	Q1

product_id	product_brand	product_name	product_sku	product_qty	product_desc	product_uom
1	Washington	Washington Berry Juice	90748503814	2.05	0.94	8.39
2	Washington	Washington Mango Drink	5842777305	0.83	0.4	11.1
3	Washington	Washington Cream Soda	5983352293	2.05	1.64	10.5
4	Washington	Washington Diet Soda	8561139469	2.10	0.77	6.56
5	Washington	Washington Cola	2080464796	1.15	0.37	11.8
6	Washington	Washington Lemonade	3000000000	2.05	0.93	1.8
7	Washington	Washington Orange Juice	8870532250	2.05	0.8	8.57

Abbildung 34:

Im Datendiagramm werden Beziehungen zwischen foreign und primary key gelegt. Dabei zieht man vom foreign key feld auf das primary key feld.

Welche andere Variante funktioniert, ist unter *Design* zu sehen. Dabei erstellt man Beziehungen zwischen den Tabellen. Dies ist in Anlehnung an Power Query Zusammenführung.

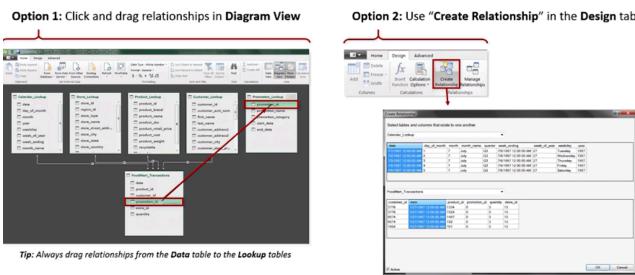


Abbildung 35:

3.2.2 Beziehungen

Beziehungen können auf zwei Arten erstellt werden. Die Beziehungen kann über die *Diagramm Ansicht*, durch ziehen der Einträge, erstellt werden. Zum anderen kann eine Beziehungen auch über den Entwurf-Tab erstellt werden

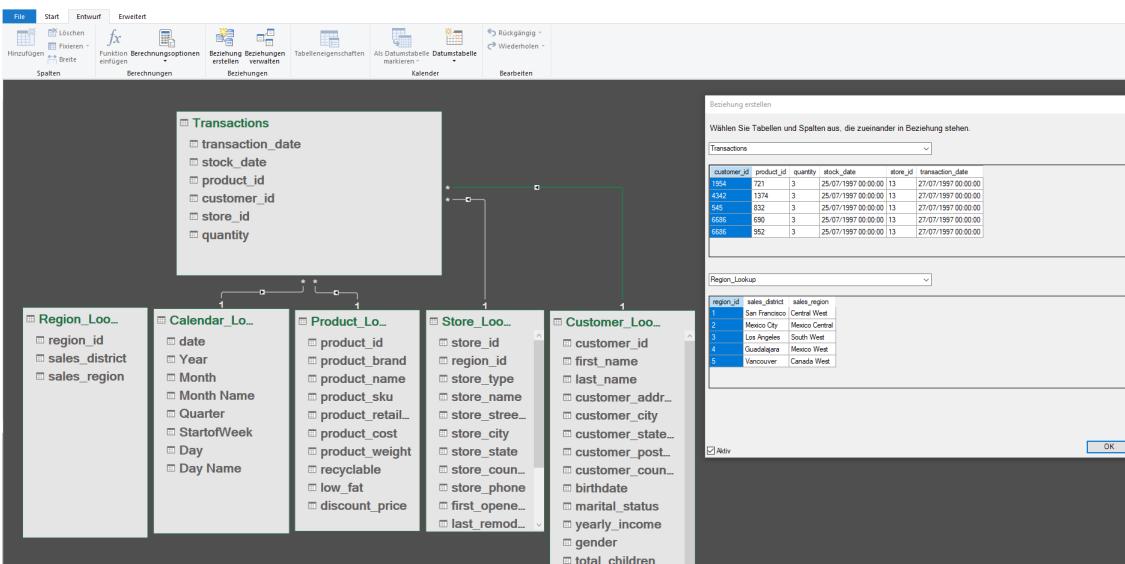


Abbildung 36:

In Power Query können verschiedene Schnittmengen gebildet werden. Es wird von *Merge* gesprochen. Es können jedoch nur $n : 1$ Beziehungen erstellt werden. Eine $m : n$ Beziehung von Power Pivot aktuelle nicht zur Verfügung gestellt. In Power Pivot kann nur eine Beziehungen zwischen zwei Tabellen erstellt werden. Eine weitere Beziehung wird dann als *inaktiv* verwaltet. Im Datendigramm wird dies mit einer gestrichelten Linie angezeigt.

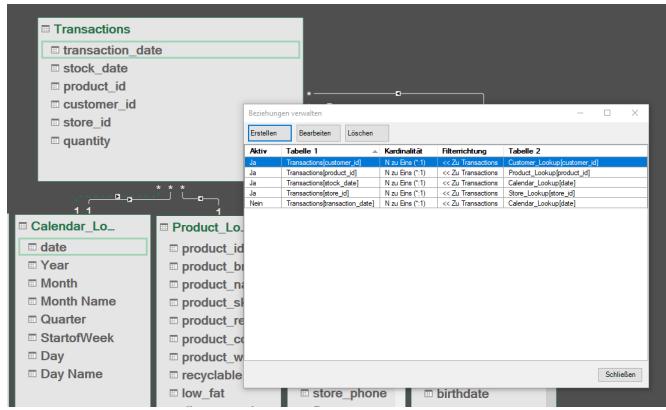


Abbildung 37:

3.2.2.1 Star Model In einem *star model* werden Beziehungen nur zwischen *primary key* Tabellen und *foreign key* Tabellen erstellt. Es bildet sich einen Sternenmuster aus Beziehungen heraus.

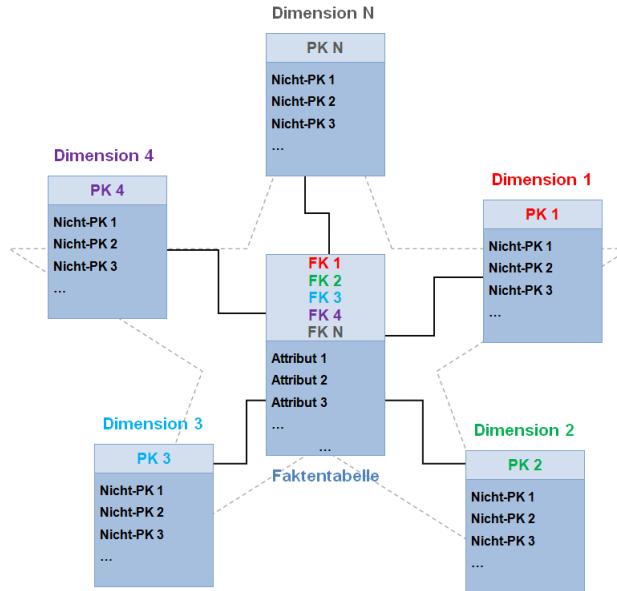


Abbildung 38:

3.2.2.2 Snowflake Model Ein *snowflake model* erlaubt Beziehungen zwischen *lookup tables*. Es entsteht ein immer weiter ausbreitendes Muster.

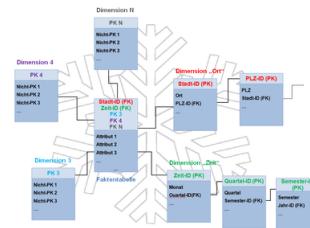


Abbildung 39:

In dem gewählten Beispiel wird **region_id** in der primären Tabelle mit **region_id** in der sekundären Tabelle verbunden. Eine verkettete Beziehung besteht jetzt zwischen der *Transaktionsliste* und der *Tabelle für Regionen*.

Dies wird über **store_id** geregelt.

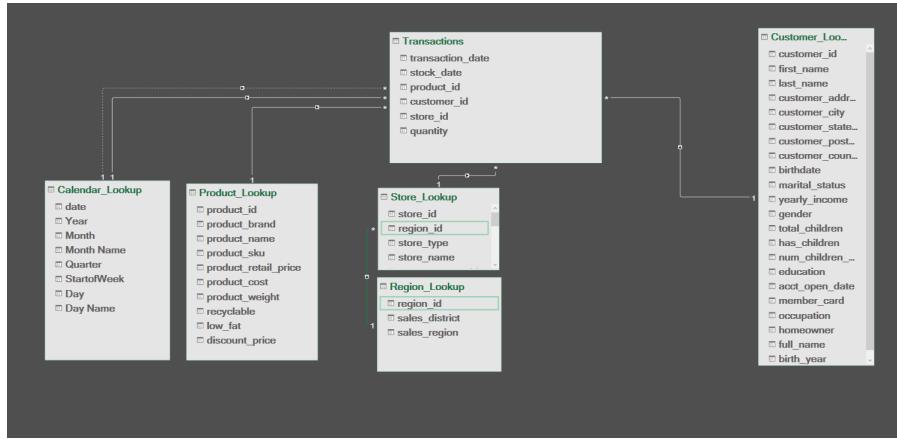


Abbildung 40:

3.2.3 Mehrere Daten Tabellen

Es wird stark empfohlen, zwischen *LookUp* und *Daten Tabellen* zu unterscheiden. Daten Tabellen werden aber nicht miteinander verbunden, sondern durch die $n : 1$ Verbindung zu den jeweiligen LookUp Tabellen verbunden. Dabei wird die Pivot-Filterfunktion besser nutzbar gemacht, wie im späteren Verlauf genauer gezeigt.

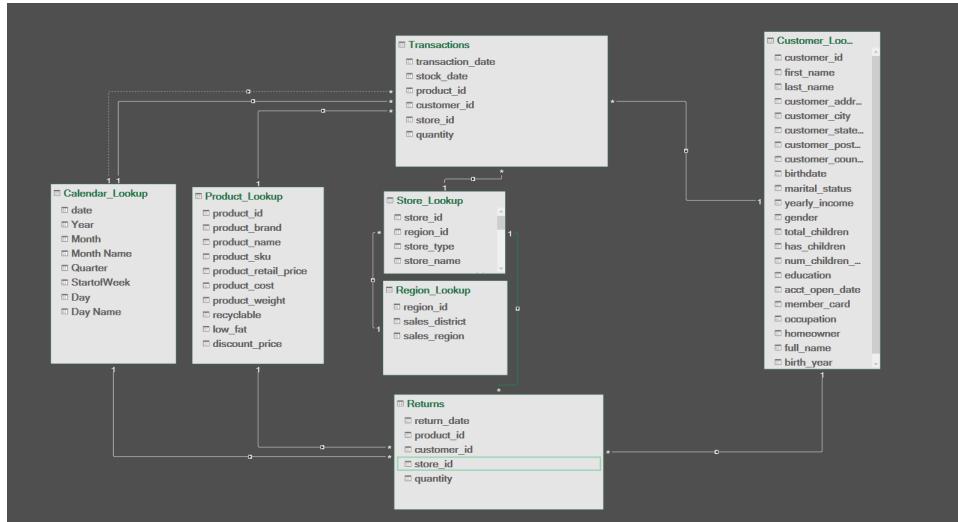


Abbildung 41:

3.2.4 Filter

Die Filterfunktion in den *PivotTable Fields* richtet sich nach den $n : 1$ Richtung. Es kann nur sinnvoll gefiltert von dem LookUp table. **Filter und Rows** müssen aus den LookUP Tabelle kommen! Die linke Seite kommt somit aus den Lookup Tabellen und die rechte Seite von den Daten Tabellen.

FILTER DIRECTION IS IMPORTANT (CONT.)

Calendar_Lookup filters flow "down" to both the Transactions and Returns tables, so we can filter or segment those metrics using any field from the Calendar table.

Filtering by date in the Transactions table yields incorrect, unfiltered values from the Returns table, since filter context cannot flow "upstream" to the Calendar table.

Abbildung 42:

In dem Beispiel gibt es zwei Daten Tabellen: Transaction und Return. Die spezifischen Werte können in die \sum **Values** Spalte eingefügt werden. Die Felder für **Filter** können mit weiteren Spalten für die ID's in den Lookup Tabellen eingefügt werden. Dafür eignet sich auch ein *kontinuierlicher Kalender*. Führt man die Abfrage durch Power Query durch, so kann man die aktuellen und relevantesten Daten filtern. Diese werden dann extrahiert und ein eigener Kalender mit den extrahierten unter Bewertungen kann geschaffen werden.

Abbildung 43:

In dem Bereich für **Spalten** und **Zeile** gilt ebenfalls, soll eine Verfeinerung der Daten aufgegriffen werden & mehrere Werte werden aus unterschiedlichen Tabellen abgegriffen, so muss die Granulierung für jeden Daten Tabelle erfolgen. Dabei muss die Spalte aus der Lookup Tabelle kommen.

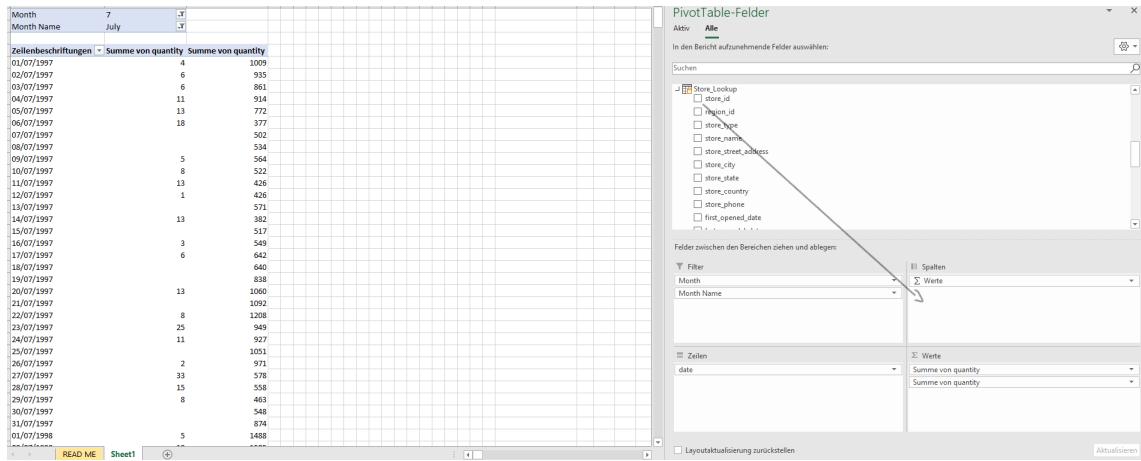


Abbildung 44:

Ein extrahieren aus einer der Daten Tabelle wird einen Fehler aufrufen, da die Filterung nicht über den Lookup Tabelle "durchfließt".

3.2.5 Hide fields form client tools

Im vorherigen Teil wurde der Fokus darauf gelegt, welche Bereich zum Filtern geeignet sind und welche nicht. Die Regel ist hier:

$$\text{Filter (Slice), Zeilen und Spalten enthalten } \textit{Foreign Key's}. \quad (3.1)$$

Ein fälschlicherweise Filtern eines primären Schlüssel (Spalte) wird verhindert, indem die primären Key's in der Daten Tabelle versteckt werden. Dies geschieht nicht im Modell selber, aber im Pivot Field Bereich.

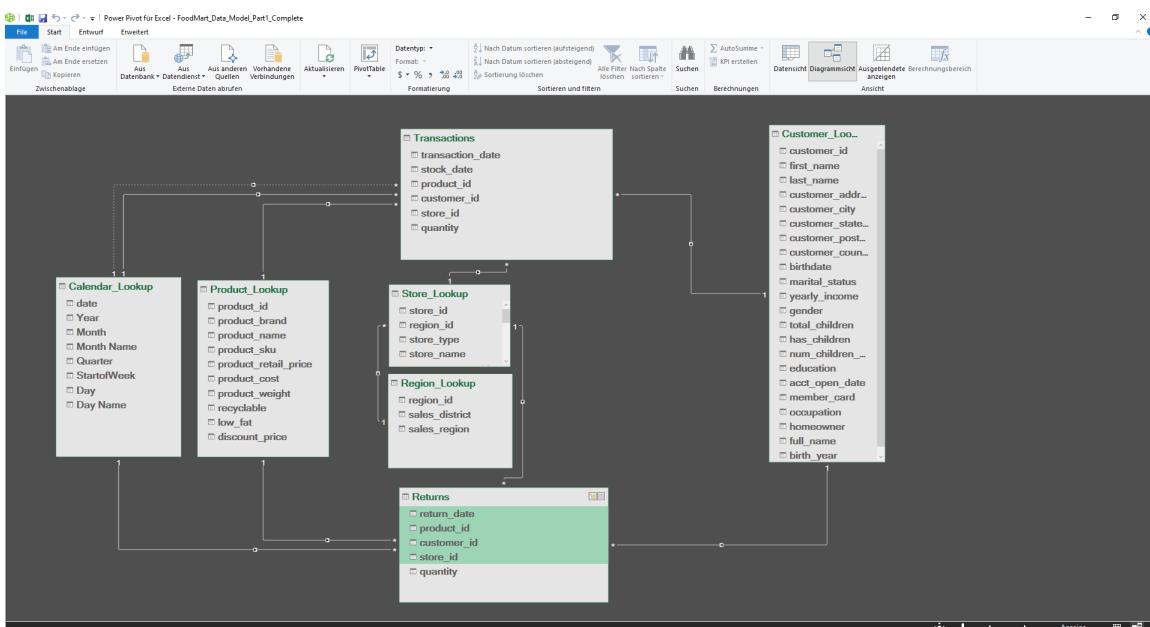


Abbildung 45:

Verwendet man die *auszublendenen Funktion*, werden die Key's nicht mehr angezeigt.

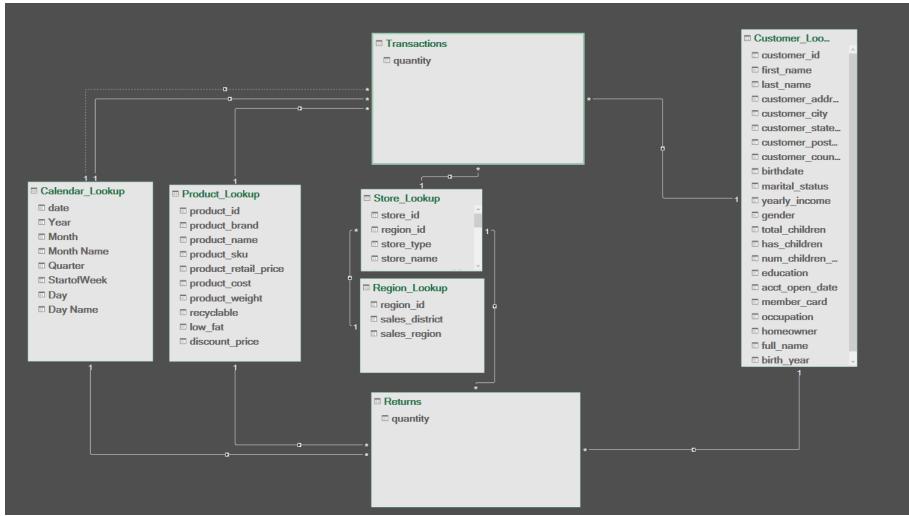


Abbildung 46:

Mit der Funktion an, wenden die Key's grau Schraffiert markiert.

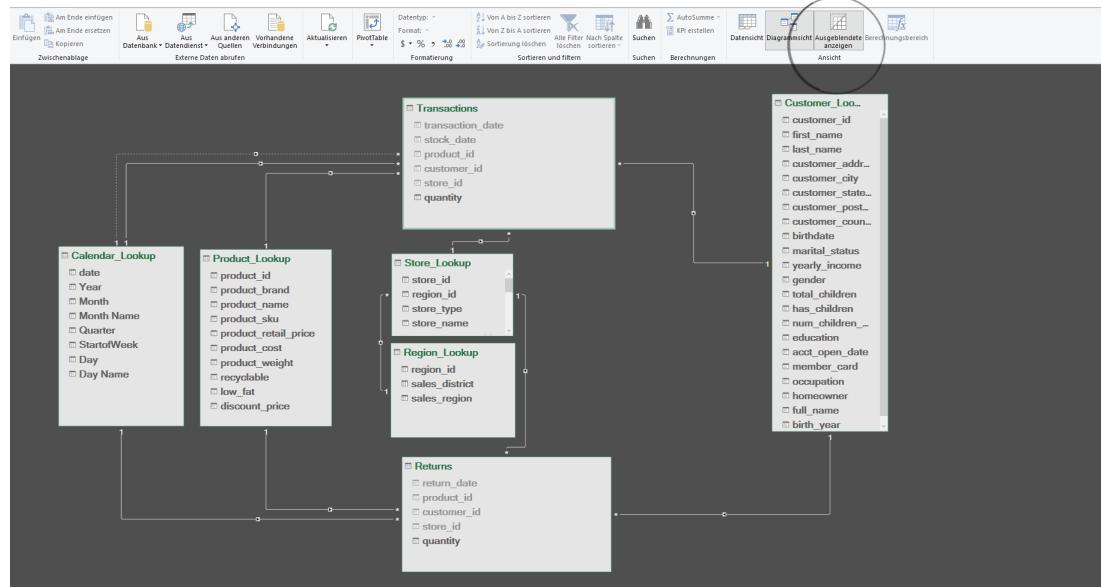


Abbildung 47:

3.2.6 Hierarchy

Mit Hilfe der Hierarchy können Gruppierungen von Foreign Key's erstellt werden, welche inhaltlich zusammenpassen. Dabei ist die Zeit eine naheliegende Tabelle.

Hierarchies are groups of nested columns that reflect multiple levels of granularity

- For example, a "Geography" hierarchy might include **Country**, **State**, and **City** columns
- Each hierarchy is treated as a **single item** in PivotTables and PivotCharts, allowing users to "drill up" and "drill down" through different levels of the hierarchy in a meaningful way



Abbildung 48:

Den Effekt wird im dritten Teil des Dreiteilers besprochen.

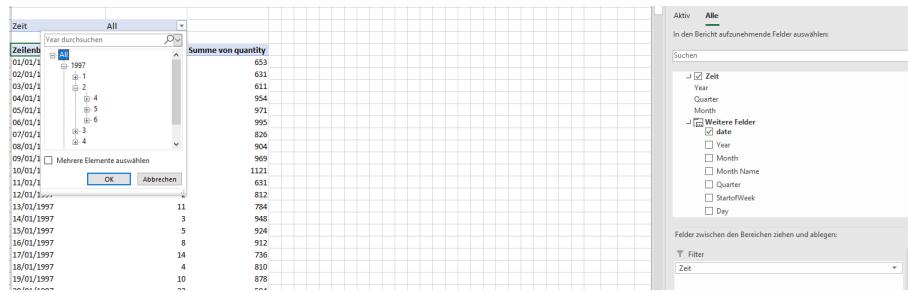


Abbildung 49:

4 Business Intelligence: Power Pivot and DAX

4.1 Power Pivot 101

4.1.1 Oberfläche von Power Pivot

Einige Objekte der Oberfläche wurden schon in den vorherigen Kursen behandelt. Die Resultate können am Ende der Prozesses per *Pivot Tabelle* Schaltfläche in Excel eingeführt werden.

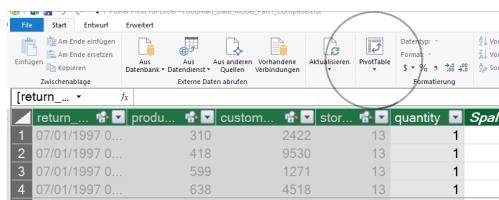


Abbildung 50:

Über diese Funktion können verschiedenen Kombinationen von Pivot Tabellen und Grafiken eingefügt werden.

4.1.2 Berechnungen

4.1.3 Funktion mit normalen Tabellen

Tabellen die nicht über das Datenmodell verarbeitet sind und direkt mit Power Pivot eingefügt werden können die die **Berechnete Feld** Funktion nicht nutzen. Diese sind ausgegraut.

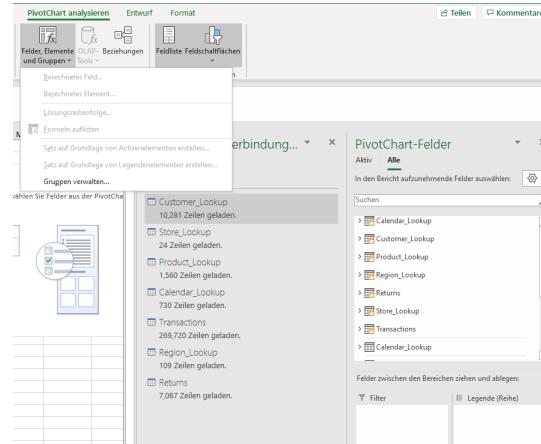


Abbildung 51:

Tabellen die nicht im Daten Modell liegen, können extra Felder zu der Tabelle hinzufügen ohne an den *Rohdaten* etwas zu verändern.

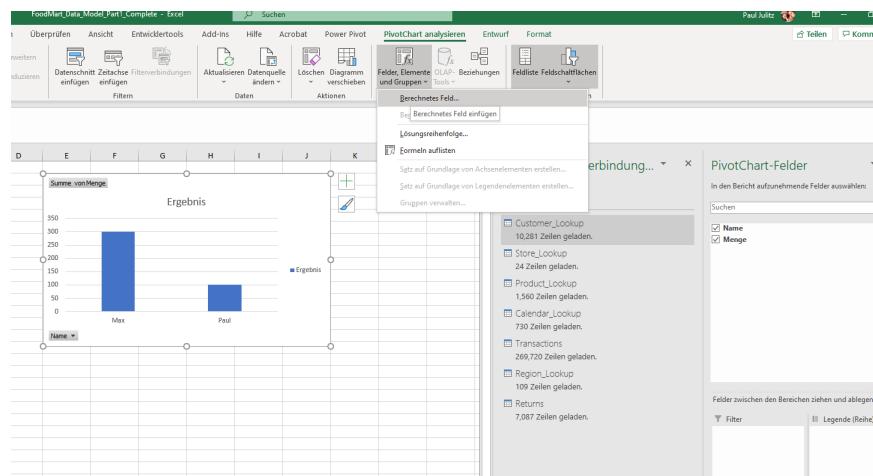


Abbildung 52: Öffnen

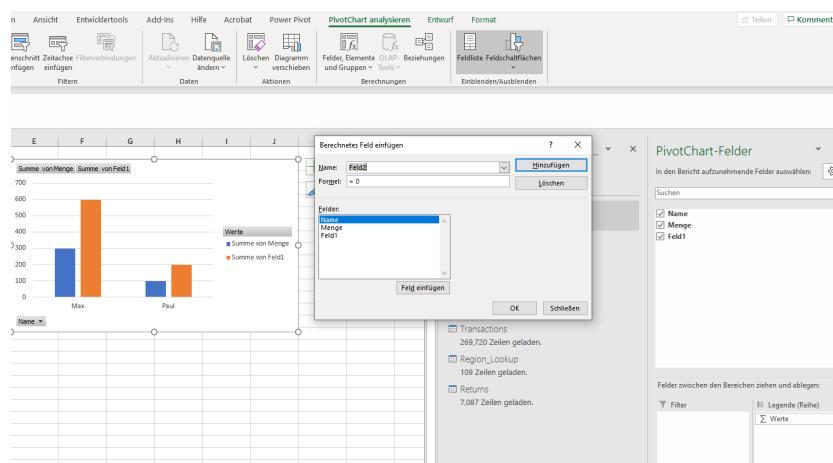


Abbildung 53: $f(x) = 2 \text{ mal Menge}$

4.1.4 Calculated Columns

Berechnende Spalten sind ähnlich aufgebaut, wie erweiternden Spalten in Tabellen. Es ist ratsam, diese gleich über Power Query anzulegen. Diese werden meist verwendet, um Slicer, Zeilen oder Spalten Granulierungen vorzunehmen.

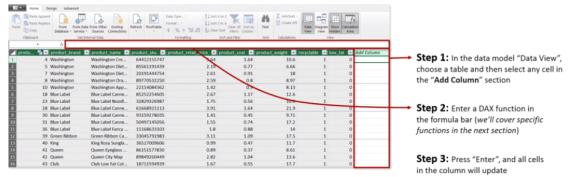


Abbildung 54:

Die benutzerdefinierten Spalten werden für einzelnen Tabellen verwendet. Die **Measures** werden für das gesamte Datenmodell verwendet. Alles was mit aggregierten Werten zu tun hat, wird darüber bestimmt.

	produ...	custom...	stor...	quantity	Berechnete Spalte 1	Spalte hinzufügen
310	2422	13	1	1	2	
418	9530	13	1	1	2	
599	1271	13	1	1	2	
638	4518	13	1	1	2	
1078	484	13	1	1	2	
75	8919	13	1	1	2	
107	4261	13	1	1	2	
189	3268	13	1	1	2	
391	4507	13	1	1	2	
475	1173	13	1	1	2	
569	625	13	1	1	2	
590	9929	13	1	1	2	
804	545	13	1	1	2	
959	8473	13	1	1	2	
992	8164	13	1	1	2	
1046	2548	13	1	1	2	
1049	4261	13	1	1	2	
1063	7416	13	1	1	2	
1226	4342	13	1	1	2	
1319	2548	13	1	1	2	
1393	4507	13	1	1	2	
1404	5782	13	1	1	2	

Abbildung 55:

Ebenso ist zu berücksichtigen, dass es aggregierte Werte geben kann, diese werden dann aber für jeden Zeile verwendet.

Abbildung 56:

Einige der DAX Funktionen sind gleich von der Syntax und den Eingabewerten wie in Excel.

total_chil...	num_children_at_...	educat...	acct_ope...	er...	occup...	homeo...	has_chil...	full_name...	birth_ye...	Spalte hinzufügen	
3	0	High Sch...	16/11/19	HOUR	Manual	N	Y	Bertha Ja...	1948		
3	0	High Sch...	05/05/19	MINUTE	Manual	N	Y	Ole Weldon	1931		
3	0	High Sch...	26/06/19	MONTH	Manual	N	Y	Paul Alcorn	1973		
2	0	High Sch...	09/02/19	QUARTER	Manual	N	Y	Jared Bust...	1910		
3	0	High Sch...	15/03/19	SECOND	Manual	N	Y	Margaret A...	1979		
4	0	High Sch...	02/03/19	WEEK	Manual	N	Y	Vanessa T...	1930		
2	0	High Sch...	04/06/19	YEAR	Manual	N	Y	Catherine ...	1966		
1	0	High Sch...	05/03/1992	...	Manual	N	Y	Stacey Cer...	1943		
4	0	High Sch...	17/05/1992	0...	Bronze	Manual	N	Y	Marlin Coriell	1933	
3	0	High Sch...	08/09/1992	0...	Bronze	Manual	N	Y	Deanna Sa...	1916	
4	0	High Sch...	21/02/1991	0...	Bronze	Manual	N	Y	Joseph Th...	1968	
4	0	High Sch...	12/06/1994	0...	Bronze	Manual	N	Y	Roberta St...	1919	
1	0	High Sch...	24/04/1992	0...	Bronze	Manual	N	Y	Paula Toml...	1948	
3	0	High Sch...	08/11/1991	0...	Bronze	Manual	N	Y	Troy Lipford	1953	

Abbildung 57:

4.1.5 Implicit Funktionen

Zieht man die Spalten von den Daten Tabellen in den *Werte Bereich* wir die Auswahl von verschiedenen Funktionen geben. Diese Funktionen nennt man **Implicit Function**.



Abbildung 58:

Die Funktionen sind fix vorgegeben. Warum oft darauf hingewiesen ist, *explicit function* zu nutzen, liegt daran, dass implizierte Funktionen fix und keine weiteren Änderungen an ihnen vorgenommen werden kann.

- Der Name der Funktion kann nicht verändert werden.
- Zahlenformate zu den Funktionen kann nicht geändert werden. Hinweis: Soll ein gewisses Zahlenformat beibehalten werden, bietet Power Pivot explizite Funktionen an, das Format fix zu halten. Dabei wird bei jeder weiteren Verwendung der Funktion das Dateiformat angepasst.

4.1.6 Explicit Funktionen - Measures

Einfache **explizierte Funktionen** werden in die Tabellen in Funktionsbereich geschrieben.

ID	quantity	Sum of quantity
1	120	120
2	230	230
3	340	340
4	450	450
5	560	560
6	670	670
7	780	780
8	890	890
9	1000	1000
10	1110	1110
11	1220	1220
12	1330	1330
13	1440	1440
14	1550	1550
15	1660	1660
16	1770	1770
17	1880	1880
18	1990	1990
19	2000	2000
20	2110	2110
21	2220	2220
22	2330	2330
23	2440	2440
24	2550	2550
25	2660	2660
26	2770	2770
27	2880	2880
28	2990	2990
29	3000	3000

Abbildung 59:

Bildet man die gleiche implizierte Funktion nach, so funktioniert sie auch in gleicher Weise.

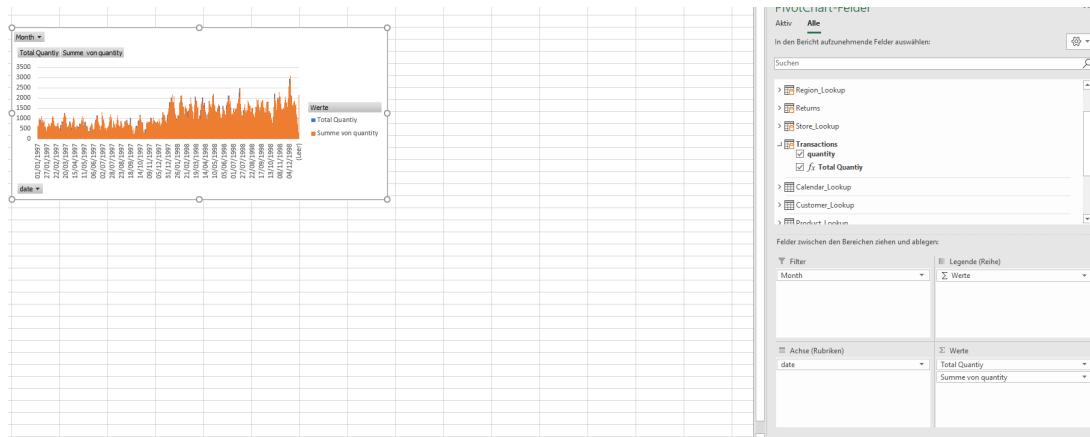


Abbildung 60:

Die Verwaltung der *expliziten Funktionen* kann auch über Excel unter *Power Pivot/ Measures* eingesehen werden.

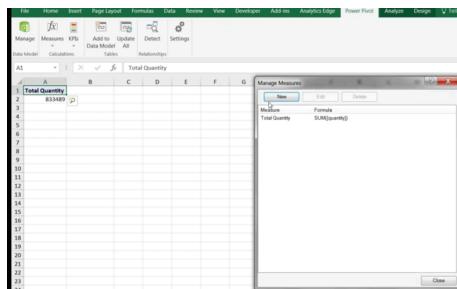


Abbildung 61:

Wir ein *Measures* erstellt, kann dies ebenfalls über das Eingabefeld erfolgen.

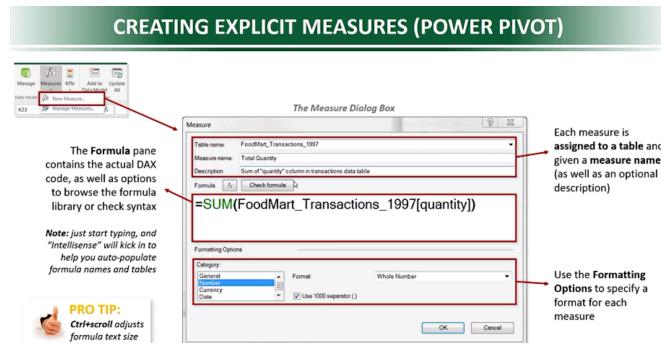


Abbildung 62:

Die Zuweisung hat nicht damit zu tun, dass nur Werte aus der Tabelle verwendet werden können, sondern damit zu welcher Tabelle es im Pivot Field angezeigt wird. Die *expliziten Funktionen* werden deshalb bevorzugt, weil

- eine genau Bezeichnung möglich ist.
- Measures aus verschiedenen Tabellen zusammengeführt werden können.
- Datentypen angepasst werden können.

A screenshot of the Microsoft Power BI interface. On the left is a PivotTable with data from row 1 to 23. The columns include 'Zeilenschriftarten' (row 1), 'Total Quantity' (row 1), 'Summe von quantity' (row 1), 'Zeilenschriftarten' (row 2), 'Total Quantity' (row 2), and 'H' through 'M'. The data shows various cities and their total quantities. Row 24 is a blank row. To the right of the PivotTable is the 'PivotTable-Felder' pane, which lists several lookup tables and their fields. Below this is a 'Filter' section where 'sales_district' is selected under 'Zeilen'.

Abbildung 63:

4.1.7 Filterfunktion

Die Art und Weise wie Filter (Slicer), Zeilen und Spalten verwendet werden können, kann als Wasserfall-Diagramm verstanden werden. Die Filter werden in den Lookup Tabellen gesetzt. Über die Key's werden die Tabellen, welche in Beziehung zu der Lookup Tabelle stehen gefiltert. Soll ein Filter für mehrere Daten Tabellen angelegt werden, wo werden nämlich alle Tabellen gefiltert, welche in Beziehung stehen. Ein lokaler Slicer kann im Zweifelsfall über eine eigene Spalte in der Datentabelle erstellt werden. Die Filterfunktion erstreckt sich dabei über alle Spalten, die in der Lookup Tabelle zu finden sind, selbst wenn sie in der ursprünglichen Tabelle nicht angefügt wurden.

Erzeugte Slicer müssen mit anderen Verweisen verbunden werden, sodass die Daten upgedatet werden sollen. Sonst bleiben auch verknüpfte Koordinaten-Kategorien separat.

4.2 Basic DAX Function

4.2.1 Syntax und Operatoren

Die DAX Sprache ist eine Funktionssprache. Dies bedeutet, ein Measure muss eine Funktion enthalten, die eine Form der Aggregations zu lässt. Hingegen können Berechnende Spalte ohne Funktionen auskommen und einfache Operatoren verwenden.

Die gängigsten Operatoren für DAX sind:

DAX OPERATORS					
Arithmetic Operator	Meaning	Example	Comparison Operator	Meaning	Example
+	Addition	$2 + 7$	=	Equal to	$[City] = "Boston"$
-	Subtraction	$5 - 3$	>	Greater than	$[Quantity] > 10$
*	Multiplication	$2 * 6$	<	Less than	$[Quantity] < 10$
/	Division	$4 / 2$	>=	Greater than or equal to	$[Unit_Price] >= 2.5$
^a	Exponent	$2 ^ 5$	<=	Less than or equal to	$[Unit_Price] <= 2.5$
			<>	Not equal to	$[Country] <> "Mexico"$
Text/Logical Operator					
&	Concatenates two values to produce one text string				$[City] & " " & [State]$
&&	Create an AND condition between two logical expressions				$([State] = "MA") \&& ([Quantity] > 10)$
(double pipe)	Create an OR condition between two logical expressions				$([State] = "MA") ([State] = "CT")$
IN	Creates a logical OR condition based on a given list (using curly brackets)				$'Store Lookup'[State] IN {"MA", "CT", "NY"}$



Abbildung 64:

Für *and* wird **&&** verwendet, und für *or* werden zwei vertikale Trennstriche verwendet. Für die Syntax der Funktionen kann Power Query verwandt werden. Die Referenz zu einer Spalte in einer Tabelle funktioniert

gleich. Variablen Namen werden aber anders gehandhabt. Namen mit Leerzeichen werden mit einfachen Anführungsstrichen umschlossen.

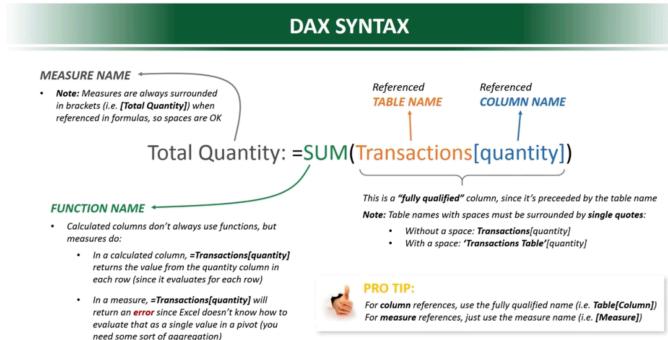


Abbildung 65:

Die Funktionen ähneln den die aus Excel und Power Query bekannt sind. Im weiteren Verlauf wird aber noch verstärkt auf einzelne Funktionen eingegangen.

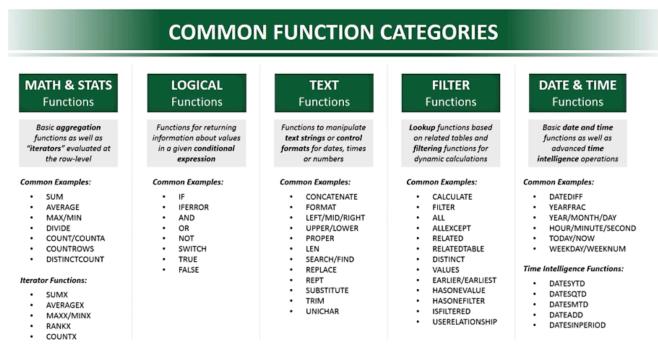


Abbildung 66:

4.2.2 Function

Für numerische Werte können viele verschiedenen Spalten hinzugefügt werden.

Abbildung 67:

Werden mehrere Spalten hinzugefügt, so können Sortierungen über die Zeilenfilterung durchgeführt werden.

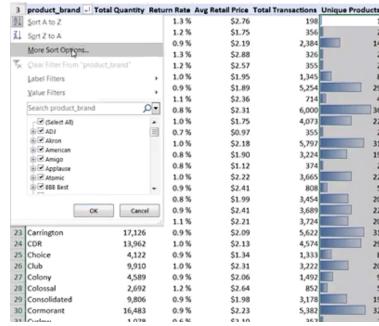


Abbildung 68:



Abbildung 69:

Einfache Logische Funktionen verhalten sich wie schon oben beschrieben. Bei *And()* und *Or()* Funktionen können auch mit den dazugehörigen Operatoren beschrieben werden. Sollen mehrere Menge zusammengefasst werden, nimmt man die Operatoren.



Abbildung 70:

4.2.2.1 Switch Function Ein einfacher Austausch von Werten.

SWITCH & SWITCH(TRUE)

SWITCH()	Evaluates an expression against a list of values and returns one of multiple possible result expressions
-----------------	--

=SWITCH(<expression>, <value1>, <result1>, <value2>, <result2>, ... <else>)

↑

Any DAX expression that returns a single scalar value, evaluated multiple times (for each row/constant)

List of **values** produced by the expression, each paired with a **result** to return for rows/cases that match

Examples:

- `=SWITCH(Calendar_Lookup[month_num], "January", "February", etc..)`
- `=SWITCH(TRUE(), [retail_price]<5, "Low Price", AND([retail_price]>=5, [retail_price]<20, "Med Price", AND([retail_price]>=20, [retail_price]<50, "High Price", "Premium Price"))`

Abbildung 71:

Switch erlaubt auch eine Variante, in der mehrere Überprüfungen vorgenommen werden können. In der ersten Übergabe wird *true()* übergeben.

TEXT FUNCTIONS		
LEN()	Returns the number of characters in a string	=LEN(<text>)
CONCATENATE()	Joins two text strings into one	=CONCATENATE(<text1>, <text2>)
LEFT/MID/ RIGHT()	Returns a number of characters from the start/middle/end of a text string	=LEFT/RIGHT(<text>, <num_chars>) =MID(<text>, <start_num>, <num_chars>)
UPPER/LOWER/ PROPER()	Converts letters in a string to upper/lower/proper case	=UPPER/LOWER/PROPER(<text>)
SUBSTITUTE()	Replaces an instance of existing text with new text in a string	=SUBSTITUTE(<text>, <old_text>, <new_text>, <instance>)
SEARCH()	Returns the position where a specified string or character is found, reading left to right	=SEARCH(<find_text>, <within_text>, <start_num>, <NotFoundValue>)

Abbildung 72:

4.2.2.2 Text Funktionen

4.3 Advanced DAX Function

Die bisherigen Funktionen finden sich in einer oder anderen Form in Excel oder auch der M-Query Sprache wieder. Es gibt aber auch komplexerer Funktion in DAX. Die *Calculate* Funktion. Diese erlaubt komplizierte Berechnungen, basierend an verschiedenen Filtern. Dies ist nützlich, wenn Berechnungen nur für bestimmte Felder ausgeführt werden sollen.

CALCULATE	
CALCULATE()	Evaluates a given expression or formula under a set of defined filters
=CALCULATE(<expression>, <filter1>, <filter2>, ...)	
Name of an existing measure or a formula for a valid measure	List of simple Boolean (True/False) filter expressions (note: these require simple, fixed values; you cannot create filters based on measures)
Examples: • [Total Transactions] • SUM[Transactions[quantity]]	Examples: • Store_Lookup[store_country] = "USA" • Calendar[Year] = 1998 • Transactions[quantity] >= 5
 PRO TIP: CALCULATE works just like SUMIF or COUNTIF, except it can evaluate measures based on ANY sort of calculation (not just a sum, count, etc); it may help to think of it like "CALCULATEIF"	

Abbildung 73:

4.3.1 Calculate

Dies Funktion nimmt in als erstes Argument eine Variable vom Typ table auf. Als zweites Argument wird ein Filter eingesetzte. Dabei wird die Tabelle, Stream up, gefiltert. Dieser Filter wird dann weiter gegeben und die Berechnungen im ersten übergebenen Tabelle oder Tabellen berechnet. Es ist eine verbesserte Variante der *Sumif*-Funktion. Die Funktionsweise für DAX Funktionen ist aber wichtig zu verstehen. Die Filter in der dazugehörigen Tabellen werden aktiviert und diese Filter werden für das Measure komplett im Datenmodell weitergegeben. Die Berechnungen mit *n*-Tabellenspalten im ersten Bereich werden dann durchgeführt, unter Berücksichtigung der Filter.

Wie man sieht werden vorherige, übergebene Filter überschrieben, weswegen in jeder Zeile der gleiche Wert steht.

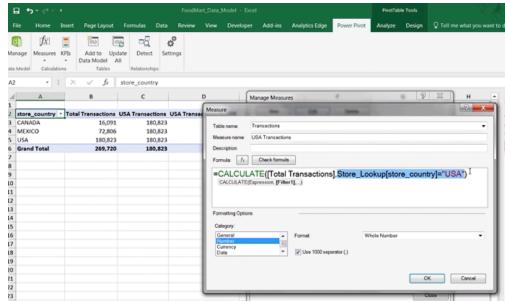


Abbildung 74:

Der Filter, als zweites Argument, kann auch eine Tabelle enthalten. In diesem Fall wird die Tabelle, die gefiltert wird, ersetzt und die Tabelle, welche durch die Filter-Funktion gefiltert wurde, ersetzt temporär die angegebene Tabelle.

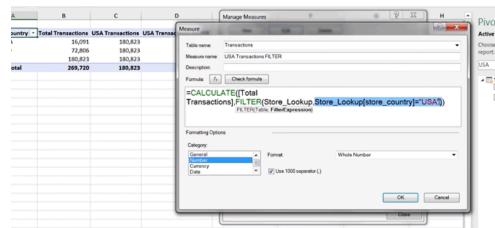


Abbildung 75:

Mit der Filter-Funktion existieren die gesuchten Filter nicht mehr und die Zeilen bleiben gleich.

store_country	Total Transactions	USA Transactions	USA Transactions FILTER
CANADA	36,091	180,823	180,823
MEXICO	77,806	180,823	180,823
USA	180,823	180,823	180,823
Grand Total	260,720	180,823	180,823

Abbildung 76:

4.3.2 Filter

Die Filterfunktion erhält als Input eine Tabelle. Die Filterfunktion wird mit Angaben von Spezifikation von Spalten erstellt. Der Funktion gibt eine Tabelle wieder.

FILTER

FILTER() Returns a table that represents a subset of another table or expression

=FILTER(<table>, <filter expression>)

Table to be filtered

Examples:

- `Store_Lookup`
- `Product_Lookup`

A Boolean (True/False) filter expression to be evaluated for each row of the table

Examples:

- `Store_Lookup[store_country]=\"USA\"`
- `Calendar[Year]>1998`
- `[retail_price]>AVERAGE[retail_price]`

HEY THIS IS IMPORTANT!

FILTER is used to add filter context on top of what's already defined by the PivotTable layout.

Since FILTER returns a table (as opposed to a scalar), it's almost always used as an input to other functions, like enabling more complex filtering options within a CALCULATE function (or passing a filtered table to an iterator like SUMX).

Abbildung 77:

Wie kann man **Ergänzende-Slicer** in eine ein Datenmodell einfügen, ohne es mit den restlichen Tabellen zu verbinden → mit der Filterfunktion. Die Tabelle Der Filter in der Filterfunktion kann durch eine extra-eingefügte Tabelle gebunden werden.

1. Die MAX-Funktion gibt nur einen Wert wieder:

MAX

04/19/2019 • 2 minutes to read • [Edit](#)

Returns the largest value in a column, or between two scalar expressions.

Syntax

```
DAX
MAX(<column>)
```

```
DAX
MAX(<expression1>, <expression2>)
```

Parameters

Term	Definition
column	The column in which you want to find the largest value.
expression	Any DAX expression which returns a single value.

Abbildung 78:

Die Einbindung läuft über zwei Measure.

PRO TIP: FILTERING WITH DISCONNECTED SLICERS (PART 2)

STEP 4: Place your new table on the pivot as a slicer:

STEP 5: Create a measure to capture the slicer selection, then reference it in a **FILTER** statement within **CALCULATE**:

The **Transactions Under Price Threshold** measure calculates Total Transactions when the product price is below the selected threshold

Abbildung 79:

Hinweis: Dies Funktion wäre auch sinnvoll, wenn es um die Prämienberechnung geht. In dem man die einzelnen **Threshold** angibt. Zusätzlich, es können auch Slicer direkt über die Ansicht der Pivot-Tabellen angezeigt werden.

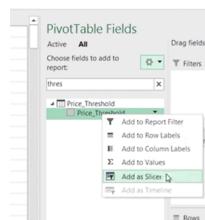


Abbildung 80:

4.3.3 All

Die All-Funktion ignoriert die bisher angewendeten Filter zu einer Tabelle. Dies erlaubt Berechnungen vorzunehmen, welche nicht ändern sollen, wenn interaktiv Filter angewandt werden.



=ALL(<table> or <column>, [column1], [column2],...)

The table or column that you want to clear filters on

List of columns that you want to clear filters on (optional)

Notes:

- If your first parameter is a table, you can't specify additional columns
- All columns must include the table name, and come from the same table

Examples:

- Transactions
- Product_Lookup[product_brand]

Examples:

- Customer_Lookup[customer_city], Customer_Lookup[customer_country]
- Product_Lookup[product_name]

Abbildung 81:

Angenommen, es ist verlangt, eine Tabelle mit prozentualen Verhältnissen zu zeigen.

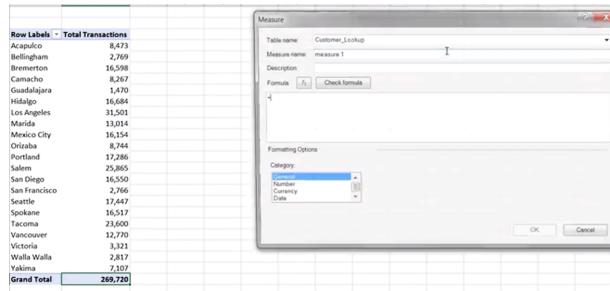


Abbildung 82:

Dies ist nur möglich, wenn die Summe am Ende gleich bleibt, wenn man möchte, dass durch Selektion die Prozente sich nicht verändern. Die All-Funktion erlaubt, die Verhältnisse bei zu behalten, selbst wenn die Zeilen gefiltert werden. Zum Beispiel könnte das Measures heißen:

$$StatisProzent = \frac{[Transaction]}{\text{Sum}(All([Transaction]))} \quad (4.1)$$

4.3.4 Iterator (X) - SumX()

Das Grundprinzip von Measure ist, dass die Funktion einen Wert und keine Tabelle oder Spalte wieder gibt. Eine Aggregation von Text ist somit im einzelnen nicht möglich, außer, eine klvere Funktion wird geschrieben.

Die Steuerung wird über die Filter gesteuert!

Die Iteratorfunktionen sind nur erweiterte Funktionen ihrer zugrundeliegenden, einfachen Funktionen. Als Beispiel, die *Sum()* Funktion kann nur ein Aggregat einer Spalte sein.

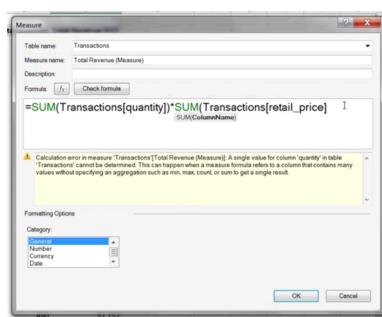


Abbildung 83:

Dabei wird erst die komplette erste Spalte und danach die komplette zweite Spalte aggregiert. Erst dann, werden die beiden Skalare miteinander multipliziert. Bei der Iterator Funktion wird, wird Zeile für Zeile summiert und im Anschluss aggregiert. Dies macht auch die *Calculate()* Funktion. Dabei wird nur zusätzlich ein Filter benötigt.

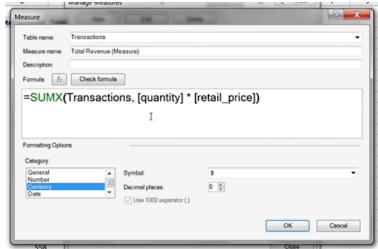


Abbildung 84:

Die *SumX()* Funktion verlangt eine Tabelle, von welcher die Berechnungen ausgehen. Es kann aber mit der *Related()* Funktion Daten aus anderen Tabellen gezogen werden.

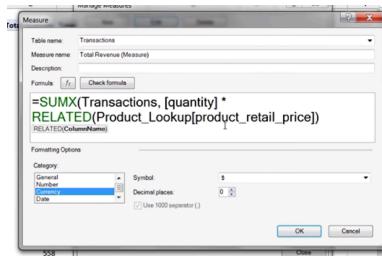


Abbildung 85:

4.3.5 Iterator (X)- RankX()

Die *RankX()* Funktion ist eine iterative Funktion, weil sie erst die komplette Spalte durchlaufen muss, bevor jeder Wert in Reihe gebracht werden. Die Funktion gibt eine Liste von Zahlen wieder. Diese gibt an.

RANKX

12/10/2018 • 2 minutes to read •

Returns the ranking of a number in a list of numbers for each row in the *table* argument.

Syntax

DAX	Copy
RANKX(<table>, <expression>[, <value>[, <order>[, <ties>]]])	

Abbildung 86:

Es ist dabei wichtig, zu berücksichtigen, dass die Filterfunktion einen Einfluss auf die Funktion hat. Die Funktion *All()* erlaubt, die Filter auszublocken und einen Ordnung der Werte zu ermöglichen, obwohl sie gefiltert sind.

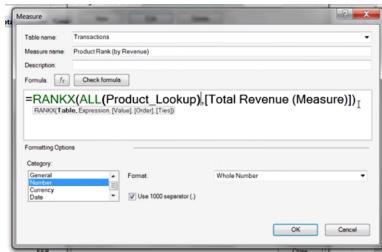


Abbildung 87:

Weitere Unterteilungen kann wie folgt aussehen:

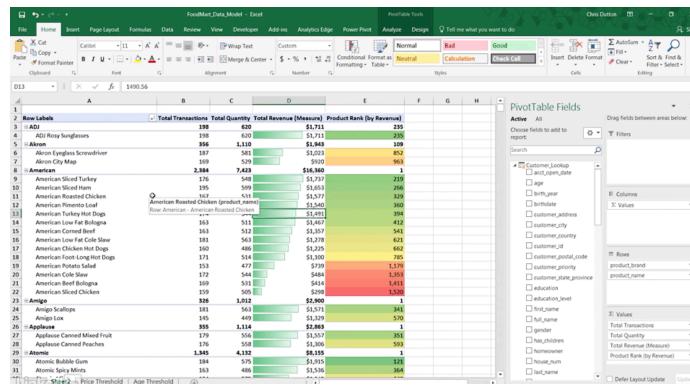


Abbildung 88:

4.3.6 Time-Intelligence Function

Diese Funktionen verhalten sich ähnlich zu denen in M .

TIME INTELLIGENCE FORMULAS

Time Intelligence functions allow you to easily calculate common time comparisons:

Performance To-Date	=CALCULATE(<measure>, DATESYTD(Calendar[Date])) Use DATESQTD for Quarters or DATESMTD for Months
Previous Period	=CALCULATE(<measure>, DATEADD(Calendar[Date], -1, MONTH)) Select an interval (DAY, MONTH, QUARTER, or YEAR) and the # of intervals to compare (i.e. previous month, rolling 10-day)
Running Total	=CALCULATE(<measure>, DATESINPERIOD(Calendar[Date], MAX(Calendar[Date]), -10, DAY))

PRO TIP: To calculate a moving average, use the running total calculation above and divide by the # of intervals!

Abbildung 89:

Die erste Funktion erlaubt in Abschnitten zu untergliedern. Dabei wird ein spezifischer Filter für das Datum verwandt. Nachdem die ersten Funktionen geschrieben sind, können die Funktionen einfach miteinander kombiniert werden. Die Funktion *DateAdd()* erlaubt:

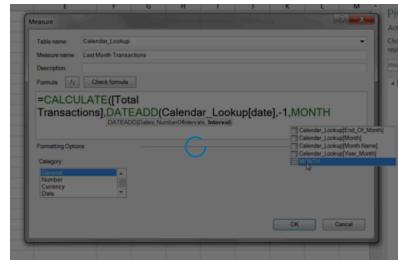


Abbildung 90:

Setzt man die Funktion ohne mit mit ins Verhältnis, so ergibt sich:

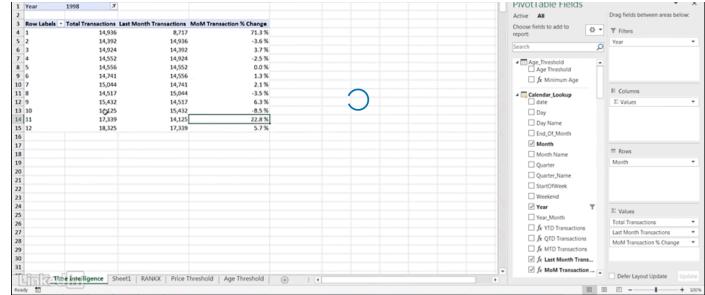


Abbildung 91:

Gleiches gilt auch für die Funktion für Perioden *Dateinperiode()*. Eine fixe Zeitspanne kann somit erstellt werden.

5 Advanced Microsoft Power BI

5.1 Filter

5.1.1 Filter für Measure und DAX Funktionen

Es können zwei große Kategorien von Filter unterschieden werden:

- Pivot-Koordinaten-Filter (PKF)
 - Bei gleichen Spalten in der gleichen Tabelle, wird der Filter geblockt und so betrachtet, als wurde er nicht implementiert.
- DAX-Filter-Option
 - Filter werden von außen nach innen weitergegeben.
 - Bei Spalten die in Beziehung stehen, muss aufgepasst werden, ob PKF von der Up-Stream oder Down-Stream Seite kommt. Entweder wird die der PKF geblockt oder führt dazu, dass die restliche Datenbasis leer wird.

5.1.2 All-Funktion

- Die *All()* blockt alle Filter, die von anderen DAX Funktionen übergeben wurden und den angegebenen Bereich (spezifische Tabelle oder Spalte) betreffen.
- Die *All()* blockt alle Filter, die von anderen PKF übergeben wurden und den angegebenen Bereich (spezifische Tabelle oder Spalte) betreffen. **Achtung:** Der Fokus liegt auf die Filterung, nicht auf den Bereich der Berechnet wird. Wo

```

1 =Calculate(Sum([Anzahl_Fahrgäste]))
2 /Calculate(Sum([Anzahl_Fahrgäste]),All(ETL_FGZ_Pruefer[Anzahl_Fahrgäste]))
```

hilft nicht, wenn die PKF aus der Datumstabelle kommen. Sollen die PKF für die Daten geblockt werden, so muss Folgendes angewandt werden.

```
1 =Calculate(Sum([Anzahl_Fahrgäste]))  
2 /Calculate(Sum([Anzahl_Fahrgäste]),All('Calendar'[Date]))
```

- Der Rückgabewert ist eine Tabellen, aber *Calculate()* erkennt die Funktion an, und beitet nicht nur Boolean sondern auch Tabellen Input für DFO.

5.1.3 Filter-Funktion

- Die Funktion *Filter()* erlaubt kompliziertere Bedingungen zu formulieren. Im Vergleich zu DFO können folgenden Funktionen mit aufgegriffen werden:

– DAX

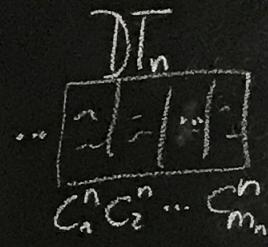
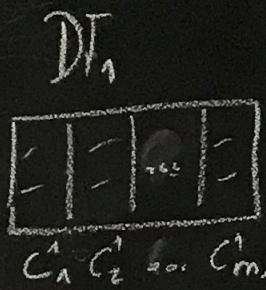
- * Die Filterbedingung erlaubt nicht für

```
1 =Calculate(Sum([Anzahl_Fahrgäste]),  
2 'Calendar'[Date]=Min(ETL_FGZ_Pruefer[Datum]))
```

- * Die *Filter()* erlaubt für

```
1 =Calculate(Sum([Anzahl Fahrgäste]),  
2 Filter('Calendar','Calendar'[Date]=Min(ETL_FGZ_Pruefer[Datum])))
```

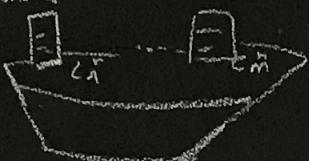
Dabei ist wichtig zu berücksichtigen, dass die PKF für den gleichen Bereich den Filter überschreiben.



$$D\bar{T} = \left\{ C_{n,m}^1, C_{n,m}^2, \dots, C_{n,m}^n \right\}; n, m \in \mathbb{N}$$

DAX-Function: $DAX_i(\square_{g_i}, V_i)$ $V_i = \{\text{Skalar, DAX Expression, Tabel, Column}\}$

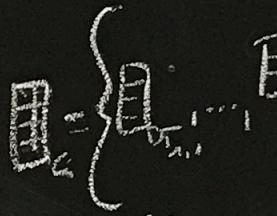
Measure: $M(DAX_1(\square_{g_1}, V_1), \dots, DAX_m(\square_{g_m}, V_m)) \in \mathbb{R}$



PT = Point Table



K_{ij} = Koordinaten
z.B. $K_{1,1}, \dots, K_{m,n}$



Filter: K_C : DE Filter Was wird geblockt? $K_C \rightarrow M(\cdot)$

I. $M(\cdot) \xrightarrow{PT} K_C = \emptyset \rightarrow M(\cdot)$ Berechnung der DI unter PT $\xrightarrow{*} (ohne D\bar{T}_C \text{-Filter})$

II. $M(\cdot) \xrightarrow{PT} K_C \neq \emptyset \rightarrow K_C \text{-Filter} \rightarrow F.F \rightarrow F.F \xrightarrow{*} M_C$

III. $M(\cdot) \xrightarrow{PT} K_C \neq \emptyset \rightarrow K_C \text{ gleich } D\bar{T}_C \text{-Filter} \rightarrow K_C \text{ und } D\bar{T}_C$

Mit Hilfe einer Umschließung der *Calculate()* Funktion kann das PKF geblockt werden.

```
1   =Calculate(
2     Calculate(
3       Sum([Anzahl Fahrgäste]),
4       Filter('Calendar',
5         'Calendar'[Date]=Min(ETL_FGZ_Pruefer[Datum])
6       )
7     ),
8     All('Calendar'[Date])
9   )
```

- Measure
- Vergleiche von Tabellenspalten

```
1   CALCULATE(Sum([Anzahl Fahrgäste]),Filter('Calendar','Calendar'[Date]='
2     Calendar'[Date]))
```

- Wie schon oben erwähnt, werden PKF nicht geblockt und können somit direkt angewandt werden.

5.2 Introducing the X-Factor

5.2.1 SUMX()

Die *SUMX()* addiert von Zeile zu Zeile den gesetzten Ausdruck. Wird als Ausdruck nur der Spaltenverweis verwendet, so ist *SUMX()* und *SUM()* gleich. Erst, wenn zeilenweise Multiplikation oder komplexere Berechnungen benötigen werden, die nicht linear, unterscheiden sich die Funktionen. Betrachten wir also nicht lineare Ausdrücke in der der *SUMX()*, so kommt es zu unterschieden.

5.2.2 COUNTX()

Wie gerade beschrieben, werden Ausdrücke auf Zeilenebene bewertet.

5.2.3 RANK.EQ() und RANKX()

Die *X* – Funktion funktionieren überwiegend als erweiterte Funktion ihrer normalen Varianten. Dabei werden Filtermöglichkeiten erlaubt und Expression ermöglicht.

- *Rang.EQ(Value, Spalte, Order)* gibt die Rang eines spezifischen Wertes *Value* wieder.

```
1   = Calculate(
2     Rank.EQ(
3       50, 'Spezifischer Wert'
4       [Anzahl Fahrgäste], 'In welcher Spalte gesucht wird
5       ASC ' Absteigende Reihenfolge
6     )
7   )
```

Der gesuchte Wert 50 wird in der Spalte *[Anzahl Fahrgäste]* gesucht. Dabei werden PKF nicht geblockt. Das bedeutet, in dem Beispiel, dass *[Date]* ein PKF ist, wird der Filter weiter gegeben und für jeden angewandten Filter der Measure berechnet.

Zeilenschriftrüfung	Measure 1	Measure 2
17.03.2020	107	45
18.03.2020	52	17
19.03.2020	44	22
20.03.2020	31	
21.03.2020	31	
22.03.2020	57	
23.03.2020	41	
24.03.2020	51	
25.03.2020	60	
26.03.2020	49	
27.03.2020	39	
28.03.2020	13	
29.03.2020	24	
30.03.2020	181	112
31.03.2020	102	55
01.04.2020	66	33
02.04.2020	53	
03.04.2020	105	47
04.04.2020	47	42
05.04.2020	85	
06.04.2020	216	
07.04.2020	168	48
08.04.2020	170	70
Gesamtergebnis	1.726	945

Abbildung 93: *Rank.EQ()* unter PKF

In der Zeile 17.03.2020 wird der Filter weitergegeben und *Rang.EQ()* daraufhin angewendet.

- Für den Wert kann auch eine Measure verwendet werden. Als Beispiel kann auch der angelesene *MAX()* Wert verwendet werden.
- Eine andere Alternative ist es, eine Spalte einzufügen und *Rang.EQ()* zu verwenden, um jeden Wert einen Rang zuzuweisen.

Ze... [= Rank.EQ([Anzahl Fahrgäste],[Anzahl Fahrgäste],ASC)]	E... [= Rank.EQ([Anzahl Fahrgäste],[Anzahl Fahrgäste],ASC)]	A... [= Rank.EQ([Anzahl Fahrgäste],[Anzahl Fahrgäste],ASC)]	Anzahl Fah... [= Rank.EQ([Anzahl Fahrgäste],[Anzahl Fahrgäste],ASC)]	Durchl. Ausl... [= Rank.EQ([Anzahl Fahrgäste],[Anzahl Fahrgäste],ASC)]	Zeitintervall... [= Rank.EQ([Anzahl Fahrgäste],[Anzahl Fahrgäste],ASC)]	Zeitintervall... [= Rank.EQ([Anzahl Fahrgäste],[Anzahl Fahrgäste],ASC)]	Hauptverke... [= Rank.EQ([Anzahl Fahrgäste],[Anzahl Fahrgäste],ASC)]	Ansteckungs... [= Rank.EQ([Anzahl Fahrgäste],[Anzahl Fahrgäste],ASC)]	R... [= Rank.EQ([Anzahl Fahrgäste],[Anzahl Fahrgäste],ASC)]	Einzelne_Zunge... [= Rank.EQ([Anzahl Fahrgäste],[Anzahl Fahrgäste],ASC)]	Berechnete Spalte 1... [= Rank.EQ([Anzahl Fahrgäste],[Anzahl Fahrgäste],ASC)]
1	St. Marien... Ostbah...	47	23,5	10:00	10:15 Sonstige Zeiten	(1) Gering	Stadtteil... TRUE		897		
2	St. Marien... Ostbah...	64	32	13:30	13:30 Sonstige Zeiten	(1) Gering	Stadtteil... TRUE		1135		
2	Neuperl... Ostbah...	55	27,5	14:30	14:30 Sonstige Zeiten	(1) Gering	Stadtteil... TRUE		1015		
2	Neufahrn Ostbah...	89	44,5	20:00	20:00 Sonstige Zeiten	(1) Gering	Stadtteil... TRUE		1394		
2	Ebersh... Ostbah...	83	41,5	10:00	10:00 Sonstige Zeiten	(1) Gering	Stadtteil... TRUE		1326		
2	Tutting Ostbah...	85	42,5	13:00	13:15 Sonstige Zeiten	(1) Gering	Stadtteil... TRUE		1352		
2	Hohen... Ostbah...	100	50	15:00	15:00 Sonstige Zeiten	(1) Gering	Stadtteil... TRUE		1473		
2	Harthaus... Ostbah...	104	52	20:00	20:00 Sonstige Zeiten	(1) Gering	Stadtteil... TRUE		1494		
2	Wächter... Ostbah...	28	14	20:00	20:15 Sonstige Zeiten	(1) Gering	Stadtteil... TRUE		571		
2	Flughaf... Ostbah...	84	42	09:00	09:00 Sonstige Zeiten	(1) Gering	Stadtteil... TRUE		1338		
2	Fasang... Ostbah...	15	7,5	20:30	20:30 Sonstige Zeiten	(1) Gering	Stadtteil... TRUE		261		
2	Eichenau Ostbah...	114	57	13:30	13:30 Sonstige Zeiten	(1) Gering	Stadtteil... TRUE		1540		
2	St. Mat... Ostbah...	21	10,5	19:30	19:30 Sonstige Zeiten	(1) Gering	Stadtteil... TRUE		403		
2	St. Marien... Ostbah...	22	11	19:30	19:30 Sonstige Zeiten	(1) Gering	Stadtteil... TRUE		427		
2	St. Marien... Ostbah...	10	5	19:30	19:30 Sonstige Zeiten	(1) Gering	Stadtteil... TRUE		161		
2	Heimer... Ostbah...	10	5	19:30	19:30 Sonstige Zeiten	(1) Gering	Stadtteil... TRUE		161		
2	Feldm... Ostbah...	78	39	13:00	13:15 Sonstige Zeiten	(1) Gering	Stadtteil... TRUE		1281		
2	Aying Ostbah...	33	16,5	15:00	15:00 Sonstige Zeiten	(1) Gering	Stadtteil... TRUE		648		
2	Flughaf... Ostbah...	4	2	13:30	13:45 Sonstige Zeiten	(1) Gering	Stadtteil... TRUE		25		
2	Fasang... Ostbah...	17	8,5	13:30	13:30 Sonstige Zeiten	(1) Gering	Stadtteil... TRUE		308		
2	Ge-Mari... Ostbah...	12	6,5	20:00	20:00 Sonstige Zeiten	(1) Gering	Stadtteil... TRUE		708		

Abbildung 94: *Rank.EQ()* unter als Spaltenwert.PKF

Teil III

VBA

1 Das VBA-Tutorial

1.1 Grundlagen

1.1.1 Debug.Print

Diese Funktion *Debug* in Visual Basic Application (VBA) ist ein Objekt, welches zwei Methoden besitzt. Diese sind *Print* und *Assert*. Die *Print* Methode erlaubt das einfache Wiedergeben der Werte von Variablen in VBA. Der Vorteil gegenüber *MsgBox* ist, dass eine Bestätigung der Ausgabe erfolgen muss. Ebenso bleibt die Darstellung im Direktfenster nach Beendigung der Prozedur erhalten.

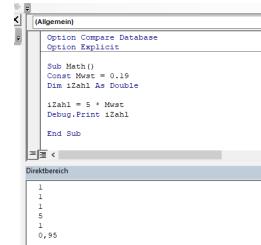


Abbildung 95:

1.1.2 Array

Ein Array in VBA wird über () indiziert. Dabei kann ein Array jeglichen Datentypen enthalten.

```
1 Option Compare Database
2 Option Explicit
3
4 Sub Math()
5
6     Dim sAusgabe(2) As String
7
8     sAusgabe(0) = "Test im Feld 0"
9     Debug.Print sAusgabe(0) ' Test im Feld 0
10
11    sAusgabe(1) = "Test im Feld 1" 'Test im Feld 1
12    Debug.Print sAusgabe(1)
13
14 End Sub
```

Die Dimensionalisierung des Arrays kann auch über ein Variable erfolgt. Ebenso kann das Array auch neu definiert werden. Dafür wird *ReDim* verwendet. Dim iAnzahl As Integer

```
1 Sub Math()
2
3     iAnzahl = 3
4     ReDim sFeld(iAnzahl)
5     sFeld(0) = "Feld 0"
6     Debug.Print sFeld(0) ' Feld 0
7
8     iAnzahl = 4
9     ReDim sFeld(iAnzahl)
10    Debug.Print sFeld(0) ' Leer
11
12 End Sub
```

1.2 Objekte

1.2.1 Klassen

Wie auch in der Objekt-Orientierte-Programmierung (OOP) können über VBA Klassen erstellt werden. Dabei wird das *Klassen-Module* verwendet. Der Name des Klassenmoduls im *Klassenmodule* Ordner ist das Objekt.

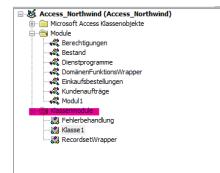


Abbildung 96:

Am Beispiel des *Autos* wird die Klasse **Auto** geschaffen. Die Klasse ist zwar leer, aber in VBA wird sie schon als Klasse erkannt. Dies kann nachvollzogen werden, wenn in einem *Standardmodul* eine Variable vom Typ der Klasse Auto erzeugt wird.

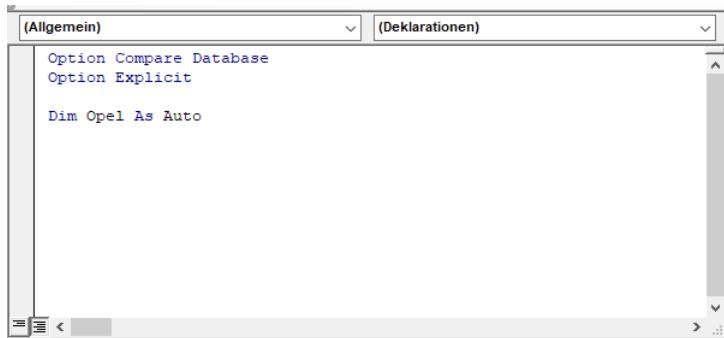


Abbildung 97:

Das Symbole

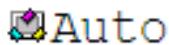


Abbildung 98:

zeigt dabei Auto als Objekt an. **Wichtig:** Die Dimensionierung einer Objektvariablen ist nur ein *Verweis*. Die Erzeugung des Objekts wird durch

Set \cdots = *New* Auto (1.1)

instanziert. Sollen mehrere Auto erzeugt werden, so lässt sich dies durch weitere Objektverweise erzeugen.

```
1 Dim Opel As Auto  
2 Dim VW As Auto  
3 Dim Ferrari As Auto
```

Wie schon erwähnt, sind die Objekte noch nicht instanziert. Es folgt:

```
1 'Objektverweise
2 Dim Opel As Auto
3 Dim VW As Auto
4 Dim Ferrari As Auto
5
6 'Instanzisierung
7 Set Opel = New Auto
8 Set VW = New Auto
9 Set Ferrari As Auto
```

Der Vorgang kann ebenso auch in einer Zeile erfolgen.

```
1 Dim Opel As New Auto
```

Man spricht auch hier von *Objektdeklaration*.

1.2.2 Methoden

Methoden werden in VBA mit dem Symbole



Abbildung 99:

dargestellt.

Für die Klasse `Auto` wird die Methode `hupen` definiert. Diese ist der Klasse zu gewissen und kann über die Klasse abgerufen werden. Eine Methode wird mit einer Prozedur definiert.

```
Access_Northwind - Auto (Code)
[Allgemein] hupen
Option Compare Database
Option Explicit

Public Function hupen(Optional Sekunden As Integer = 2) As Boolean
    'Funktionsweise
    MsgBox Sekunden
    'Wiedergabe
    hupen = True
End Function

Access_Northwind - Modul1 (Code)
[Allgemein] Autofahren
Option Compare Database
Option Explicit

Public Sub Autofahren()
    'Objektverweis
    Dim Opel As Auto
    'Objektklasse
    Set Opel = New Auto
    'Auszählen
    Opel.hupen (1)
    Opel.hupen
End Sub
```

Abbildung 100:

Die Funktion (Mehtode) `hupen()` verwendet eine Optionale Variable. Dabei wird mit `= 2` der voreingestellt Wert definiert, welcher verwendet wird, wenn die kein Wert übergeben wird. Oder kurzgesagt: *Default*. Die Syntax wie Variablen eingegeben werden, kann mit `()` oder ohne erfolgen.

1.2.3 Eigenschaft

1.2.3.1 Property Zusätzlich zu den Methoden einer Klasse, können auch Eigenschaften definiert werden. Im einfachsten Fall sind dies Intrinsische Variablen. Es können aber auch weitere Objekte als Eigenschaft definiert werden.

Eigenschaften werden in mit Intellisense mit dem Symbole:



Abbildung 101:

dargestellt. Damit werden einzelnen Variablen, Unterobjekte und Property Prozeduren angesprochen. Wird das Objekt `Opel` der Klasse `Auto` über ein Modul initialisiert, so stehen alle Eigenschaften und Methoden.

```
Access_Northwind - Modul1 (Code)
[Allgemein] Autofahren
Option Compare Database
Option Explicit

Public Sub Autofahren()
    'Objektverweis
    Dim Opel As Auto
    Dim strFarbeObject As String
    'Objektklasse
    Set Opel = New Auto
    'Auszählen
    Opel.hupen (1)
    Opel.hupen
    Opel.strFarbe = "Grau"
    Debug.Print Opel.strFarbe
End Sub
```

Abbildung 102:

1.2.3.2 Property Prozedur Die Eigenschaften zu einer Klasse können **public** oder **privat** sein. Wenn sie öffentlich hinterlegt sind, können diese über den Klassenverweis direkt geändert werden. Variablen die **private** sind, können nur über die Klasse selber geändert werden.

Die **Property Let/ Get** Prozedure hilft die Eingabe von Daten für die Eigenschaften zu verwalten. Dies bittet sich vorwiegend für Variablen vom Typ **private** an. Eine mögliche Variante die Verwaltung zu steuern, ist die privaten Variablen mit dem Prefix **pv** zu versehen. Die Prozedur hingegen, kann den Variablennamen tragen ohne Prefix. Gleichnamigkeit der Variable und der dazugehörigen Prozedur ist nicht vorgesehen. Es ist aber möglich, das **Get** und **Let** den gleichen Namen tragen. Wie schon erwähnt, werden die Prozeduren als Eigenschaften geführt und nicht als Methoden angezeigt.

- Verwendet man nur **Property Let**, so ist eine Konfiguration der Variable vorgesehen, aber ein Auslesen nicht.
- **Property Set** erlaubt eine Adressierung von Referenzen von Objekten.
- Verwendet man nur **Property Get**, so ist eine **private** Variable schreibgeschützt.

Am Beispiel des Objektes *Opel* werden die Eigenschaften über die **Property** Prozedur verwaltet. Bei **Get** wird auch der Rückgabewert definiert. Dabei verhält sich **Get** wie eine Funktion und **Let** wie ein **Sub**.

The screenshot shows two side-by-side code editors in Microsoft Access:

- Left Editor (Access_Northwind - Auto (Code)):**

```

Option Compare Database
Option Explicit

Public strFarbe As String
Private bytTempo As Byte
Private bytSperre As Boolean

Public Function hupen(Optional Sekunden As Integer = 2) As Boolean
    'Funktionsweise
    Debug.Print Sekunden
    'Wiedergabe
    hupen = True
End Function

Public Property Let Let_Geschwindigkeit(Tempo As Byte)
    If Tempo > 250 Then
        bytTempo = 250
        bytSperre = True
    Else
        bytTempo = Tempo
        bytSperre = False
    End If
End Property

Public Property Get Get_Geschwindigkeit() As Long
    Get_Geschwindigkeit = bytTempo
    'The default value for a number is 0.
End Property

```
- Right Editor (Access_Northwind - Modul1 (Code)):**

```

Option Compare Database
Option Explicit

Public Sub AutoFahren()
    'Objektverweis
    Dim Opel As Auto
    'Objektedeklaration
    Set Opel = New Auto
    'Auszählen
    Opel.hupen (1)
    Opel.Let_Geschwindigkeit = 170
    Opel.strFarbe = "Schwarz"
    Debug.Print Opel.strFarbe
    Opel.Get_Geschwindigkeit
End Sub

```

Below the editors, a **Direktbereich** window shows the variable values:

```

1
2
Schwarz
170

```

Abbildung 103:

Für die Deklaration von Eigenschaften wird meist die **Property** Prozedur verwendet. Eine Deklaration außerhalb einer Prozedur ist nicht in VBA vorgesehen.

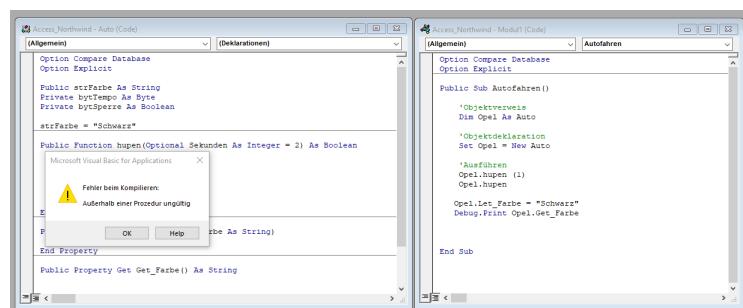


Abbildung 104: Keine Deklaration der Variablen außerhalb einer Prozedur

Innerhalb der Methoden kann dies jedoch vorgenommen werden.

```

'Property #####
Public Property Let Let_Farbe(eigFarbe As String)
    strFarbe = eigFarbe
End Property

Public Property Get Get_Farbe() As String
    Get_Farbe = strFarbe
End Property

```

Nachbereich

1	
2	
Schwarz	

Abbildung 105: Deklaration der Variablen mit Hilfe von [Property](#)

Erfolgt die Deklaration in dem *Modul*, so ist nichts weiter zu tun. Der große Vorteil bei der [Property](#) Prozedur ist, dass für [Let](#) und [Get](#) der gleiche Name verwendet werden kann. Es kann somit eine [private](#) Variable definiert werden, und diese über eine [Get](#) und [Let](#) abgerufen und verändert werden.

```

'Access_Northwind - Auto [Code]
[Alignment] Tempo [PropertyLet]

Option Compare Database
Option Explicit

Public strFarbe As String
Private bytTempo As Byte
Private myAirbags As New Collection

'Property #####
Public Property Let Tempo(getTempo As Byte)
    bytTempo = getTempo
End Property

Public Property Get Tempo() As Byte
    Tempo = bytTempo
End Property

```

Abbildung 106:

1.2.3.3 Unterobjekte Das Ziel soll sein, das ein Objekt ebenfalls aus mehreren Unterobjekten bestehen kann. Dafür werden Eigenschaften verwandt, welche als Objektvariablen definiert werden.

Die Klasse Radio wird mit einer Eigenschaft und einer Methode gefüllt.

```

'Access_Northwind - Radio [Code]
[Alignment] Name [PropertyLet]

Option Compare Database

Public mName As String

Public Property Let Name(pName As String)
    mName = pName
End Property

Public Property Get Name() As String
    Name = mName
End Property

Public Function einschalten() As Boolean
    Debug.Print "Radio ist eingeschalten."
    einschalten = True
End Function

```

Abbildung 107:

In der Klasse Auto wird die Eigenschaft *ppRadio* vom Typ [Radio](#) deklariert. Damit zu jedem Objekt ein Radio initialisiert wird, wird die *Class_Initialize* Prozedur aufgerufen. Diese wird immer angestoßen, wenn ein neues Objekt initialisiert wird.

```

'Access_Northwind - Auto [Code]
Class Initialize

Option Compare Database
Option Explicit

'Deklaration Variablen (Welche gibt es)
'.....
Public Name As String
Private ppRadio As Radio

'Get and Define
'.....
'Initialise Class
Private Sub Class_Initialize()
    Set ppRadio = New Radio
End Sub

'Data Property Object Radio
Public Property Get Radio() As Radio
    Set Radio = ppRadio
End Property

```

Abbildung 108:

Mit dem Ausdruck `SetppRadio = NewRadio` wird das Unterobjekt initialisiert. Die Get-Prozedur gibt den Verweis wieder. Dabei ist die Get-Prozedur vom Typ Radio. Mit `SetRadio = ppRadio` wird der Verweis an die Funktion übergeben. Hier müsste nochmal geprüft werden, ob Set benötigt wird. Am Ende kann das Radio mit seinen Eigenschaften und Methoden zu jedem initialisierten Auto abgerufen werden.

```
Autofahren - Modul (Code) | Automobil
[Allgemein] | Autofahren
Option Compare Database
Option Explicit

Public Sub Autofahren()
    'Objektverweise
    Dim Cpel As Auto
    Set Cpel = New Auto

    'Text
    Cpel.Name = "Objektor"
    Debug.Print Cpel.Name
    Cpel.Radio.Name = "Sadianone"
    Debug.Print Cpel.Radio.Name

End Sub
```

Abbildung 109:

1.2.4 Auflistung

Eine **Auflistung** ist ein Objekt, welches vor definierte Methoden hat:

- Item - Diese erlaubt eine Zugriff auf die Elemente in *Collection* Objekt.
 - Count - Zählt wie viele Elemente eine Auflistung besitzt.
 - Add - Fügt ein Element der Auflistung hinzu.
 - Remove - Entfernt ein Element aus der Auflistung.

```
Access_NonWingIt - Modul1 (Code)
(Allgemein)                                     ▾ Autofahren
Option Compare Database
Option Explicit

Public Sub Autofahren()
    'Objektwechsel
    Dim Opel As Auto
    Set Opel = New Auto

    'Test
    Opel.Name = "Opel"
    Debug.Print Opel.Name
    Opel.Radio.Name = "RadioName"
    Debug.Print Opel.Radio.Name
    Opel.Airbags.Count = 5
    Debug.Print Opel.Airbags.Count
    Debug.Print Opel.Airbags
End Sub
```

Abbildung 110:

- 1

An Hand des Auto-Beispiels muss Auflistung deklariert:

```
Public Property Get Airbags() As Collection
    Set Airbags = pvAirbags
End Property
```

Abbildung 111:

und initialisiert

```
Access Northwest - Auto (Code)
[Allgemein] Airbagseinbaus

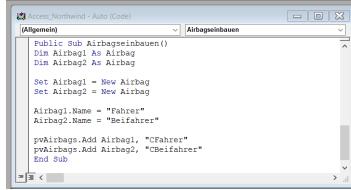
Public Name As String
Private pwRadio As Radio
Private pwAirbags As Collection

'*Set and Define.....'
'*Initialize Class
Private Sub Class_Initialize()
    Set pwRadio = New Radio
    Set pwAirbags = New Collection
End Sub

'*Sets Property Object Radio
```

Abbildung 112:

werden. In dem genannten Beispiel sollen zwei Airbags eingebaut werden. Dafür greift die Klasse Auto auf die Klasse Airbag zu und speichert diese im Objekt Airbags.



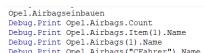
```

Public Sub Airbagseinbauen()
    Dim Airbag1 As Airbag
    Dim Airbag2 As Airbag
    Set Airbag1 = New Airbag
    Set Airbag2 = New Airbag
    Airbag1.Name = "Fahrer"
    Airbag2.Name = "Beifahrer"
    pvaAirbags.Add Airbag1, "Fahrer"
    pvaAirbags.Add Airbag2, "Beifahrer"
End Sub

```

Abbildung 113:

Über das Modul kann die Auflistung angesteuert werden, dafür gibt es verschiedene Syntaxen für eine Aufrufung der einzelne Elemente.



```

Opel.Airbagseinbauen
Debug.Print Opel.Airbags.Count
Debug.Print Opel.Airbags.Item(1).Name
Debug.Print Opel.Airbags(1).Name
Debug.Print Opel.Airbags("Fahrer").Name

```

Abbildung 114:

1.2.5 Ereignisse

Ereignisse sind Prozeduren, welche nicht über das Objekt selbst abgerufen werden können, sondern durch Vorgänge mit dem Objekt hervorgerufen werden. Zwei Prozeduren haben wir schon kennengelernt. Für **Ereignisse** wird das Symbol

Ereignisse

Abbildung 115:

verwendet. Im Objektkatalog wird diese neben Eigenschaften und Methoden angezeigt.



Abbildung 116:

Unter einem VBA-Application Objekt findet auf der linken Seite des Drop-Down

1.2.6 Objektanweisung

Zwei Anweisungen, **With... End With** und **For Each**, sind für das Referenzieren von Objekten wichtig. Es kann ein spezifisches Objekt angesteuert mit der With-Umgebung:

```
1 With Objekt  
2 .Methode  
3 .Eigenschaft  
4 .Objekt.Methode  
5 End With
```

Als Beispiel

```
1 Private Sub AddCustomer()  
2 Dim theCustomer As New Customer  
3  
4 With theCustomer  
5 .Name = "Coho Vineyard"  
6 .URL = "http://www.cohovineyard.com/"  
7 .City = "Redmond"  
8 End With  
9  
10 With theCustomer.Comments  
11 .Add("First comment.")  
12 .Add("Second comment.")  
13 End With  
14 End Sub  
15  
16 Public Class Customer  
17 Public Property Name As String  
18 Public Property City As String  
19 Public Property URL As String  
20  
21 Public Property Comments As New List(Of String)  
22 End Class
```

Mit dem **For Each** Loop können die verschiedenen Elemente in einer Auflistung angesteuert werden.

```
For Each Element-Objekt in Auflistung  
'Statement  
Next
```

Dies geschieht solange bis alle Objekt in der Auflistung durchlaufen sind. Anhand des Beispiels zum Airbagseinbau können muss ein Airbag Objekt zu erst separat deklariert werden.

```
1 Public Sub Autofahren()  
2  
3 'Objektverweis  
4 Dim Opel As Auto  
5 Set Opel = New Auto  
6 Dim LAirbag As Airbag  
7  
8  
9 'Test  
10 Opel.Name = "Objekt"  
11 Debug.Print Opel.Name  
12 Opel.Radio.Name = "Radioname"  
13 Debug.Print Opel.Radio.Name  
14 Opel.Airbagseinbauen  
15 Debug.Print Opel.Airbags.Count  
16 Debug.Print Opel.Airbags.Item(1).Name  
17 Debug.Print Opel.Airbags(1).Name  
18 Debug.Print Opel.Airbags("CFahrer").Name  
19 Debug.Print Opel.Airbags.Item(2).Name  
20 Debug.Print Opel.Airbags.Item(2).einbauen  
21 For Each LAirbag In Opel.Airbags  
22 Debug.Print LAirbag.Name 'Fahrer, Beifahrer  
23 Next  
24 End Sub
```

1.2.7 Me-Object

Wenn Makros geschrieben werden, kann dies in zwei Formen getan werden:

- Modules
- Class-Modules
 - Mit einem Interface
 - Ohne Interface

Es gibt also Klassen-Module, welche ein Interface-Design mit sich bringen. Das *ME*-Objekt hilft, die Adressierung in einem Module zu verallgemeinern. In Excel werden Tabellenblätter als *Excel-Klassenobjekte* in VBA. Am Beispiel von Northwind wird gezeigt:

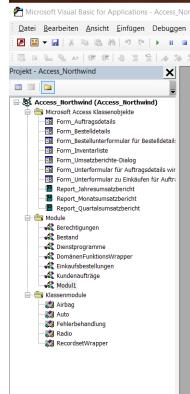


Abbildung 117:

Access weiß Form und Reports als Klassenobjekte aus. Die weiteren Module wurden ergänzt und die Klassenmodul sind die Objekte, welcher selber gewählt wurden. Wird ein Marco geschrieben, welches sich in einem Klassenmodul befindet, so kann das Objekt direkt mit *ME*. angesteuert werden.

Am Beispiel von Excel wird ein Wert "Hello Friends" der Zelle "A1" des Datenblattes "Data Sheet" gespeichert.

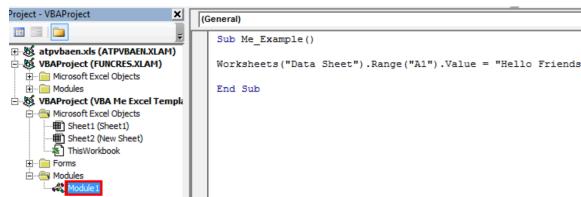


Abbildung 118:

Dieser Code könnte jedoch auch im Microsoft Excel Objekt *Sheet2 (Data Sheet)* (Achtung: Im Bild steht "New Sheet") eingetragen werden. Die Adressierung des Objekts mit

¹ Worksheet("Data Sheet").

könnte auch über

¹ Me.

erfolgen. Intellisense erkennt dies ebenfalls, und weiß die zugehörigen Methoden und Ereignisse aus.

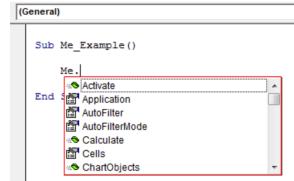


Abbildung 119:

Das Sub im Klassenobjekt selbst sieht dann wie folgt aus:

```

1 Sub Me()
2 Me.Range("A1").Value = "Hello Friends"
3 End Sub

```

Der Vorteil der sich daraus ergibt, ist, dass der Code leichter wiederverwendet werden kann. Gleichermaßen gilt für *Userforms*. Ein Userform ist ein instanziertes Objekt. Objekt-Eigenschaften können erzeugt werden und haben eine visuelle Erscheinungsbild. Die Ansteuerung kann in einem Userform-Module ebenfalls über *ME*. erfolgen.

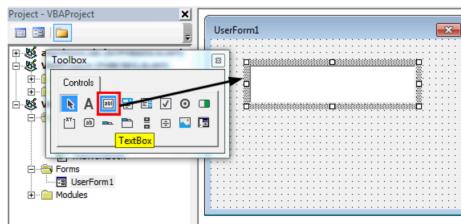


Abbildung 120:

```

1 Private Sub TextBox_Change()
2   UserForm1.TextBox1.Text = "Welcome to VBA!"
3 End Sub

```

oder

```

1 Private Sub TextBox_Change()
2   ME.TextBox1.Text = "Welcome to VBA!"
3 End Sub

```

mit dem Ergebnis

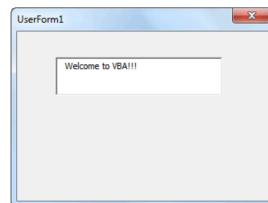


Abbildung 121:

1.2.8 Objektmodell

Bei der Programmierung eigener Objekte, ist es sinnvoller die Objektstruktur zu dokumentieren. Im Fall von Norhwind würde diese wie folgt aussehen:

Elemente des Auto-Objekts		
Ein Auto-Objekt verweist auf ein bestimmtes Auto .		
Eigenschaften	Name	Beschreibung
<input checked="" type="checkbox"/>	abgesenkt	Zeigt an, ob die Geschwindigkeit abgesenkt wurde. Boolean, schreibgeschützt.
<input checked="" type="checkbox"/>	Airbags	Gibt die Airbags-Auflistung des Objekts zurück.
<input checked="" type="checkbox"/>	Farbe	Farbe des Autos. String mit Lese-/Schreibzugriff.
<input checked="" type="checkbox"/>	Geschwindigkeit	Ganzzahl, die die aktuelle Geschwindigkeit angibt. Lese-/Schreibzugriff.
<input checked="" type="checkbox"/>	Radio	Gibt das Radio-Objekt zurück.
Methoden	Name	Beschreibung
<input checked="" type="checkbox"/>	hupen	Verursacht ein Geräusch.
Ergebnisse	Name	Beschreibung
<input checked="" type="checkbox"/>	Initialisierung	Wird beim Initialisieren des Autos ausgeführt.

Abbildung 122:

1.3 Fehlerbehandlung

Error Resume Next Fehler in der Zeile werden ignoriert. Die Prozedur wird daraufhin so ausgeführt, als wäre die Zeile nicht existent.

On Error go to Tritt ein Fehler auf, wir auf an einen definierten Punkt im Code gesprungen. Hier muss aufgepasst werden, dass die Übersichtlichkeit gewahrt wird, und die Funktion sollte nur zur Fehlerbehebung verwendet werden. Der Verweis kann ein Wort oder Zahl gefolgt von einem :.

```

1 Function GibFehler() As Double
2 Dim i As Double
3
4 On Error GoTo Eingabefehler
5
6 i = 1 / InputBox("Geben Sie eine Zahl ein")
7
8 GibFehler = i
9
10 Exit Function
11
12 Eingabefehler:
13 GibFehler = 0
14
15 End Function

```

Resume Mit Resume kann wieder zurück in den Code gesprungen werden. Zum Beispiel wird eine Schleife, basierend an dem Fehler weiter durchlaufen. Vermerk: **Err** Objekt fängt den Fehler einer Zeile ab. Mit **Err.Number** kann der Fehlerwert wiedergegeben werden.

```

1
2 Function GibFehler()
3 Dim i As Byte
4
5 On Error GoTo Eingabefehler
6
7 i = InputBox("Geben Sie eine Zahl ein")
8 i = (i ^ 2 + 1) / i
9
10 GibFehler = i
11
12 Ende:
13 Exit Function
14
15 Eingabefehler:
16 Select Case Err.Number
17     Case 6 'Überlauf (negativer Wert)
18         Resume
19     Case 11 'Division durch "0"
20         i = 0
21         Resume Next
22     Case 13 'Typen unverträglich (Text)
23         Resume Ende
24 End Select
25

```

1.4 Kurzer Einblick Formular

Combobox Zufügen der Werte für eine Combobox.

```

1 Privat Sub UserForm1_Initialize()
2 Dim Liste(2) as Variant
3
4 Liste(0) = "A"
5 Liste(1) = "B"
6 Liste(2) = "C"
7
8 Me.ComboBox1.List = Liste
9 Me.ComboBox1.AddItem ("D")
10
11 End Sub

```

1.5 Microsoft Access

1.5.1 Marcos, Module

Access bietet die Möglichkeit **Marcoaktionen** über ein Click-und-Drop-Modus zu generieren. Die Funktionen sind jedoch im Vergleich zu **Modulen** starr. Alle Makroaktionen sind in VBA über das **DoCmd** Objekt abrufbar.

1.5.2 Formulare, Reports

- Formulare werden über das das Auflistungen **Forms** angesteuert. Die Steuerelemente zugehörig zu einer Form werden über **Control** verwaltet (Collection).

```
1 Debug.Print Application.Forms(0).Control("txtName").Name
```

- Ereignisse** Die Steuerungselement eines Formulares können auf viele der Ereignisse reagieren.

Am Beispiel von Nordwind enthält das *Formular Inventarliste* viele Steuerelemente.

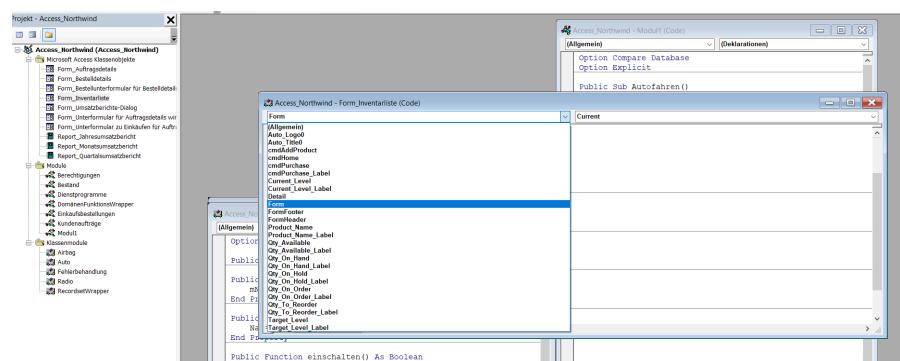


Abbildung 123:

Jedes dieser Element kann auf eine Vielzahl von **Ereignissen**.

Elemente des Auto-Objekts		
Ein Auto-Objekt verweist auf ein bestimmtes Auto .		
Eigenschaften	Name	Beschreibung
<input checked="" type="checkbox"/> abgespult		Zeigt an, ob die Geschwindigkeit abgespult wurde. Boolean, schreibgeschützt.
<input checked="" type="checkbox"/> Airbags		Gibt die Airbags-Auflistung des Objekts zurück.
<input checked="" type="checkbox"/> Farbe		Farbe des Autos. String mit Lese-/Schreibzugriff.
<input checked="" type="checkbox"/> Geschwindigkeit		Ganzzahl, die die aktuelle Geschwindigkeit angibt. Lese-/Schreibzugriff.
<input checked="" type="checkbox"/> Radio		Gibt das Radio-Objekt zurück.
Methoden	Name	Beschreibung
<input checked="" type="checkbox"/> hupen		Verursacht ein Geräusch.
Ergebnisse	Name	Beschreibung
<input checked="" type="checkbox"/> Initialise		Wird beim Initialisieren des Autos ausgeführt.

Abbildung 124:

```

1 Option Compare Database
2 Option Explicit
3
4
5 Private Sub Form_Current()
6 Me.txtNachname.BackColor = RGB(255, 128, 128)
7 End Sub
8
9 Private Sub txtNachname_AfterUpdate()
10 Me.txtNachname.BackColor = RGB(128, 128, 255)
11 End Sub

```

1.5.3 Tabellen

- Der Zugriff auf Daten wird von Access nicht unterstützt. Dies geschieht alles über Datenbank Objekt Bibliotheken. Dabei ist die wichtigste für den einfachen Gebrauch **DAO** (Database Object).
- Für die Datenbasis in der eigenen Access-Datenbank wird gilt: DAO.Database = Currentdb
- Um ein spezifischen Datentabelle zuladen wird DAO.Recordset = Currentdb.OpenRecordset("Name")

2 VBA: Access

2.1 Sammelsurium

- Close Recordset am Ende einer Prozedur: Recordset.Close Set Recordset to nothing
- Close Database am Ende einer Prozedur, Set Variable to nothing
- Recordset.Move(Rows, [Optional]Bookmark): Rows ist ein Long-Parameter der notwendig ist.
-
- Recordset.MoveNext() Bewegt sich
- Recordset.AddNew(): Fügt eine neue Zeile zum Datenset und macht die zugefügte Zeile zum *Current Record*.
- EOF: End of File gibt solange das Ende nicht erreicht ist den Wert `false` wieder.
- Idee: Ein Form anlegen, welcher über die verschiedensten Forms und Reports ansteuert. Ebenso, sollten die Buttons so angelegt werden, dass die man zurückspringen kann (Me-Objekt). Plus: Es sollten die anderen offenen Tabs geschlossen werden.
- DbEngine - DAO Objekt ist das übergeordnete Objekte. Es gibt auch keine Möglichkeit mehrere Instanzen davon zu erzeugen. Es existieren zwei große Auflistungen : Workspaces und Errors.
- Idee: Start-Seite: Hier werden alle wichtigen Informationen und Verlinkungen angezeigt.
- Über die Eigenschaften der Aktuellen Datenbank kann der Name, sowie weitere Eigenschaften angepasst werden.

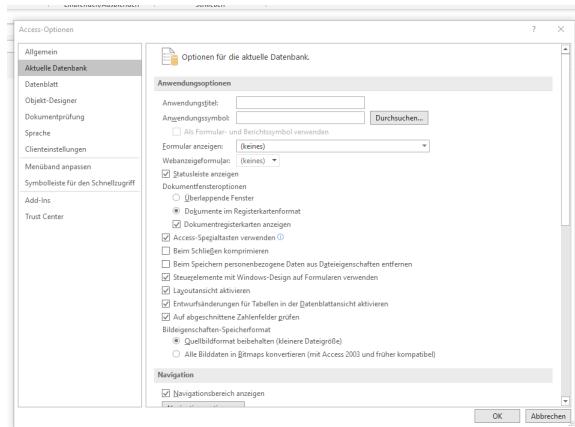


Abbildung 125:

2.2 Introduction VBA Basic

2.2.1 VBA Syntax

Method siehe OOP. Diese ist eine spezifische Funktion. Diese ist nur anwendbar für ein vorgegebenes Object oder einer Familie von vererbten Objekten.

Function A function is a piece of code that is called by name. It can be passed data to operate on (i.e. the parameters) and can optionally return data (the return value). All data that is passed to a function is explicitly passed.

Statement Die ist ein einzeiliger Ausdruck. Dieser drückt meist eine Aktion oder eine Definition aus.

```
1  Dim obj As Object ' one statement
2  ...
3  If x = true then x = 1 else 5 ' one statement
```

Um mehrere Aktionen miteinander zu verbinden, wir : verwendet.

```
1   ' ...
2   rsDataSet.Close:           Set rsDataSet= Nothing
3   ' ...
```

Mit der *With* Funktion kann man mehrere statements zusammenfassen. Dabei ist es egal, das das statement aus mehreren Methoden abrufen besteht.

```
1   ' ...
2   With Currentdb.Sheet("Tabelle1").Range("B1")
3     .Color = blue
4     .Value = "This is a message."
5     .Foreground = Brushes.DarkSeaGreen
6     .Background = Brushes.LightYellow
7   End With
8   ' ...
```

Soll das statement unterbrochen werden, so verwendet man _.

2.2.2 Object Modell for Access

Über die Seite zur Dokumentation für die Datenobjekte für VBA im Office Context bietet eine umfassende Sammlung aller Objekte die im Access Kontext wichtig sind.

The screenshot shows a Microsoft Docs page titled 'Access VBA reference'. The page is dated 10/10/2018 and has a reading time of 2 minutes. It features a sidebar with navigation links for various Office applications like Microsoft Word, Excel, and Project. The main content area contains a note about developing solutions across multiple platforms and provides links to 'Concepts' and 'Object model reference' sections. There are also 'See also' and 'Support and feedback' links at the bottom.

Abbildung 126:

Es wird zwischen *Concepts* und *Objects* unterschieden. Im folgenden wird sich überwiegend auf Objekte fokussiert.

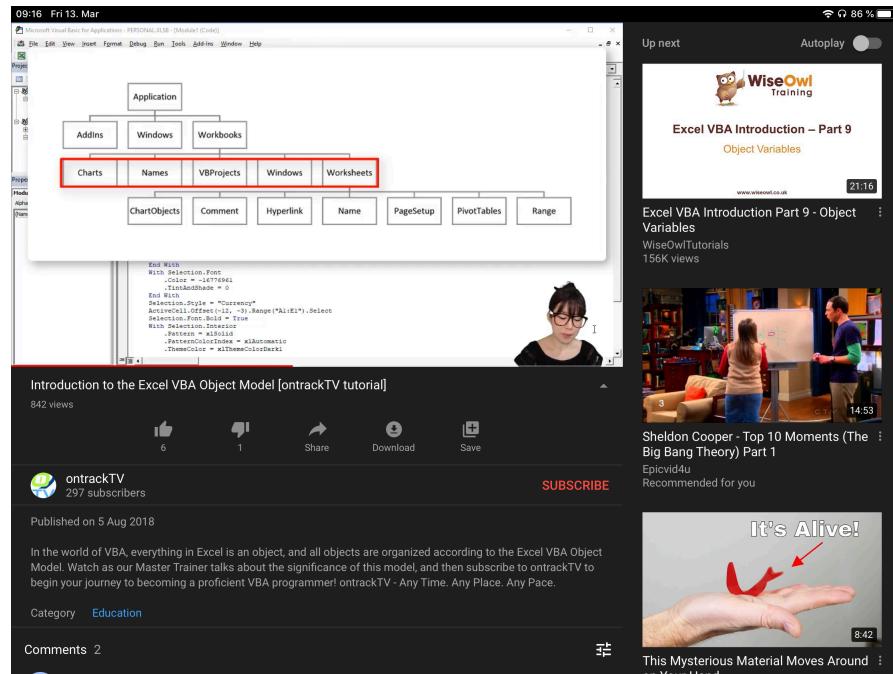


Abbildung 127:

Auf der object model wird vermutlich im späteren Verlauf nochmal eingegangen. Aktuelle wird im Kurse weiter vorangeschritten, um vielleicht das fehlende Verständnis über das Model aufzuarbeiten.

2.2.3 VBA Editor

Der Editor bietet über **F2** die Möglichkeit an, die Hilfefunktion abzurufen.

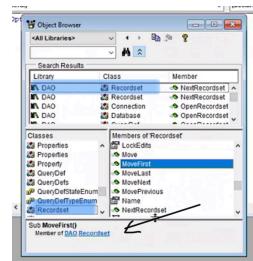


Abbildung 128:

Eine der einfachsten *Expressions* ist die Message Box. Die Funktion dazu lautet

```

1 Sub MessageBox ()
2     MsgBox "Hello Paul."
3 End Sub

```

2.2.4 Processor: Subroutine and function

In diesem Kontext ist es sinnvoll darüber zu sprechen, dass VBA zwischen Objekt-Module und Module unterscheidet. Subroutinen können Werte übertragen bekommen.

```

1 Sub DatabaseName ()
2     Dim sDbName As String ' Variablennamen können mit einem Kleinbuchstaben beginnen.
3     sDbName = "Paul"
4 End Sub

```

```
5
6 Sub ShowDatabaseName (sDbName As String)
7     MsgBox sDbName
8 End Sub
```

2.2.5 Processor: Function

Der Unterschied zur *Sub* ist, dass eine *function* einen Wert zurückgeben kann. Ein *Sub* nicht. The *return* ist called by the function name. Das bedeutet am Ende der Prozedur wird ein eine Ausgabe definiert. Diese muss mit den Namen der Funktion betitelt werden.

```
1 Function MultiplikationPy (dZahl As double)
2     MultiplikationPy = dZahl*3.14
3 End function
4
5 Sub Show ()
6     ' MsgBox MultiplikationPy(5); Dies wird nicht funktionieren, da die Funktion ein double
7     ' Wert benötigt.
8     Dim dZahl As double
9     dZahl = 5
10    MsgBox MulitplikationPy(dZahl)
11 End Sub
```

2.2.6 Set Statement

The *Set* statement is like a pointe in *C++*. VBA hat eine Objekt-Hierarchy aufgebaut. Dabei können innerhalb dieser Hierarchy Objekte angesteuert werden - es wird auf sie verweisen. Mit *Set* kann daraufhin innerhalb dieser Struktur Verweise geschaffen werden. Diese werden dann genutzt um Methoden abzurufen, auf weitere Objekte zu verweisen oder Werte auszulesen. Damit ist *Set* zu unterscheiden von dem einfachen Datentypen, wie *string*, *bool*. Die Verweise können zum einen mit einem einfachen *Object* Typ deklariert werden, zum anderen können aber auch spezifischer Typen angegeben werden.

```
1 Dim wbObjekt As Workbooks ' Variable wird initialisiert
2 Set Application.Workbooks("Name.xlsxm")
```

Wir eine neue Instant eines Objektes erzeugt, wird *New* verwendet. Bei der Deklarierung, *Set*, wird noch kein Objekt erzeugt, erst mit *Set*

```
1 Dim myChildForms(1 to 4) As Form1
2 Set myChildForms(1) = New Form1
3 Set myChildForms(2) = New Form1
4 Set myChildForms(3) = New Form1
5 Set myChildForms(4) = New Form1
```

Es können natürlich mehrere Referenzen zu einem Objekt geschaffen werden.

```
1 Dim wbObjekt As Workbooks
2 Set wb = Application.Workbooks("Name.xlsxm")
3 Set wb2 = Application.Workbooks("Name.xlsxm")
```

2.2.7 First Code-Marco

Wir ein Command Button erstellt, kann dieser über die Funktion *Event Builder* angesteuert werden.

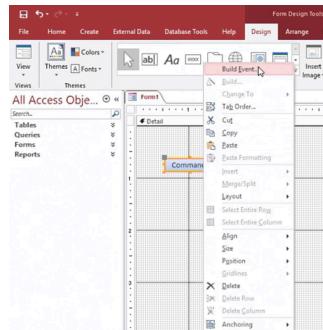


Abbildung 129:

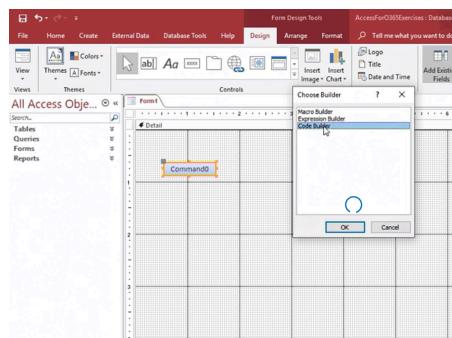


Abbildung 130:

Es wird ein Makro (Code) erstellt.

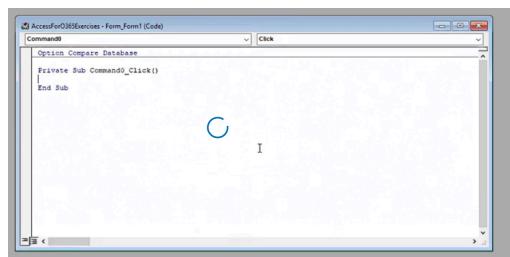


Abbildung 131:

Die Option *Option Compare Database* hat die Funktion, dass die folgenden Prozeduren auf Groß- und Klein-schreibung bei einem String-Vergleich nicht berücksichtigt wird. Ebenso wird das Modul unter dem Formblatt abgespeichert.

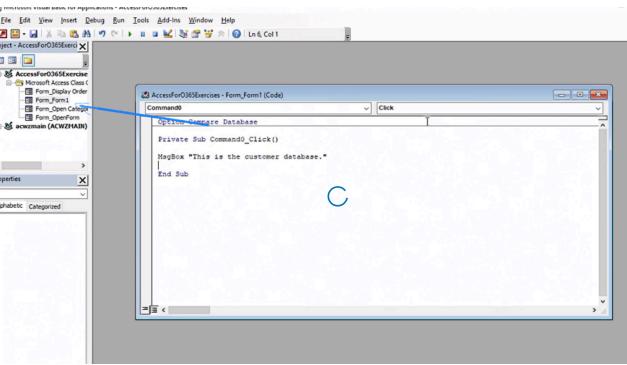


Abbildung 132:

So weit ich es verstanden habe, ist ein Form Objekt ein angesteuertes Object welches sich in Forms Objekt befindet. Methoden die für Form vorgesehen werden, können erst durch ein genaues ansteuerten eines einzelnen Form ausgeführt werden.

```

1 Sub AddressObject()
2     Dim myForm As Form
3     Set myForm = Forms("Product") ' Use of a different object
4     MsgBox myForm.Name
5     Set myForm = Nothing ' Releases the connection
6 End Sub

```

2.3 Variables, Constants, Calculation

Die Option *Option Explicit* zwingt, dass alle Variablen deklariert werden müssen. Variablen die außerhalb von Prozeduren stehen, werden als globale Variablen definiert und sind für das gesamte Modul verfügbar.

```

1 Dim iNumber As Integer = 20 ' Global
2 Const iNumber2 As Integer = 30 ' Global und behält den konstanten Wert
3 Static iNumber3 As Integer = 40 ' Global und behält den Wert über mehrere Ausführungen
4 Sub Main()
5
6 End Sub

```

2.4 Add Logic to your VBA Code

2.4.1 Iterator-Prozeduren

- For ... to ... Next: Integer Loop-Schleife
- For each ... in ... Next: Iteriert durch eine Array durch.

```

1 Sub ForEach()
2     Dim varItem As Variant
3     Dim varArray As Variant
4
5     varArray = Array("Eins", "Zwei", "Drei")
6
7     For each varItem in varArray
8         MsgBox (varItem)
9     Next
10
11 End Sub

```

Für das Form-Collection kann ebenfalls so angesteuert werden.

```

1 Sub ForEachForm()
2     Dim myF As Form
3

```

```

4 For Each myF In Forms
5 MsgBox (myF.Name)
6 Next
7
8 End Sub

```

- Do until ... Loop:

- Do while ... Loop:

Es gibt verschiedene Syntaxen/ Methoden um auf Spalten in einem Recordset zu verweisen.

- Itemmethode: myR!["Nachname"]
- Fields: myR.Fields("Nachname")

Der Verweis start immer am Anfang eines Tabelle. Der Loop muss jedoch separat gesteuert werden. In M wird die über die Funktion gesteuert und per each wird jede Zeile umgewandelt. Wir die *Recordset.AddNew()* Methode gestartet, zeigt der Zeiger am Ende der Tabelle.

```

1 Sub Ausgabe_Print()
2     Dim recTest As Recordset
3     Dim i As Long
4     Set recTest = Currentdb.OpenRecordset("Tabelle")
5
6     Do while i < recTest.RecordCount ' Oder recTest.EOF (Endoffile)
7         Debug.Print recTest.Fields("Nachname") ' Alle Daten in der Spalte Nachname werden
8             ausgegeben.
9         i = i + 1
10        recTest.Move
11    End Sub

```

- If else; elseif
- Case Statement

2.5 Debug Code

- Siehe unter Fehlerbehandlung/VBA-Tutorial.
- Breakpoint. Klickt man am linken Rand im Editor wird ein roter Punkt gesetzt. Dieser stellt ein Breakpoint da.

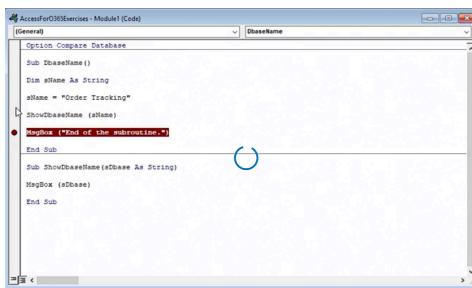


Abbildung 133:

- Add Watch

2.6 Manipulate Database Object using DoCmd Object

2.6.1 Open Objects

DoCmd OpenForm Diese Funktion bietet viele Option die Form in spezieller Weise zu öffnen: DoCmd.OpenForm

DoCmd OpenRecord Ein Record wird gleich an den Drucker gesendet. Um nur ein Report sich anzeigen zu lassen, muss die Funktion abgeändert werden.

DoCmd OpenTable Mit DoCmd.OpenTable "Name" wird die dazugehörige Tabelle geöffnet.

Die verschiedenen Ansichtsmodi helfen das Objekt im richtigen Modus zu öffnen.

2.6.2 Close Objects

Um ein Objekt zu schließen, kann

```
1 DoCmd.Close acForm "Name"
```

Unter **ac** wird das Objekt spezifiziert und über den Namen direkt angesteuert.

2.6.3 RunCommand Export

Unter DoCmd.RundCommand befinden sich die Menüoptionen. Diese können über diese Methode aufrufen. Am Beispiel Daten zu exportieren, wird der Befehl acCmdExportExcel ausgewählt.

```
1 DoCmd.RunCommand (acCmdExportExcel)
```

2.7 Read and manipulate Table Data

2.7.1 Add New

Es ist wichtig, dass nach dem die Methode AddNew() ausgewählt und die Eingaben getätigter wurden, die Update() Methode aufgerufen wird, um die Eingaben zu speichern.

2.7.2 Edit

Die Methode Edit() erlaubt einen spezifischen Datensatz zu bearbeiten. Es ist dabei wichtig vorher die entsprechende Zeile zu finden. Eine Methode, die spezifische Datensätze sucht, ist FindFirst(). Ebenfalls wird UpDate() benötigt.

2.7.3 Data Perservation

Um den Nutzer einzubinden, ob die geänderten Einträge auch wirklich verwendet werden sollen, wird gern die Transaktions-Methoden verwendet.

- BeginTrans()
- CommitTrans()
- RollbackTrans()

Diese Methoden können für verschiedenen Objekte angewandt werden. Es bietet sich an, .Workspace(0) zu verwenden. Ebenso, kann der Nutzer aufmerksam gemacht werden, ob er die Änderungen anwenden möchte.

```
1 Sub Preserv()
2     Dim worVar As Workspace
3     Dim recVar As Recordset
4     Set worVar = DAO.DBEngine.Workspace(0)
5     Set recVar = DAO.Currentdb.OpenRecordset("Name")
6
7     worVar.BeginTrans()
8
9     Do Until recVar.EOF
10
11     'Stuff
12
13     Loop
14
15     If MsgBox ("Do you want the Change?", vbQuestion + vbYesNo) = vbYes Then
```

```

16     worVar.CommitTrans()
17 Else
18     worVar.RollbackTrans()
19
20 End Preserv

```

2.7.4 TableDef

Diese Objekt ist unter Currentdb zufinden. Mit Hilfe dieses Objektes können unter anderem Eigenschaften eines Feldes abgerufen werden.

2.8 Manipulate a Database using the Application Object

2.8.1 Function to Summarize

- DAvg() - Nimmt den Average über eine Spalte hinweg: DAvg("Quanität")
- DMax(), DCount(), DMin(), etc.
- DFirst(), DLast() - Bei diesen beiden Funktionen wird die Tabelle als String verwendet: DFirst(SSpaltenname", "Tabellennamen")

2.8.2 DLookup()

Die DLookUp() Funktion gibt die Werte aus einer anderen Tabelle oder Querie wieder.

```

1 DLookUp(
2   "[Spalte_Wiedergabe_Sekundär]", //-- Der Wert der gesucht wird, aus einer meist anderen
   //-- Tabelle/ Query.
3   "Tabelle_Sekundär", //-- Andere Tabelle/ Query
4   "[Spalte_Suche_Sekundär] =" & [Spalte_Suche_Primär]) //-- Vergleiche Spalten aus der zu
   //-- suchenden Tabelle und aus der erstellten Tabelle.

```

2.9 Control Forms and Reports

2.9.1 Sinnvolle Einschränkungen

- Damit Nutzer nicht Daten einfach ändern, löschen oder hinzufügen, kann dies direkt über Eigenschaften der Forms gesteuert werden. Gesteuert werden die Eigenschaften am Besten, wenn das Objekt geladen wird.

```

1 Sub Form_Load()
2   Me.AllowAdditions = False
3   Me.AllowEdits = False
4   Me.AllowDeletions = False
5 End Sub

```

- Ebenso kann die Filterfunktion deaktiviert werden. Dies können auch in Formular angewandt werden.

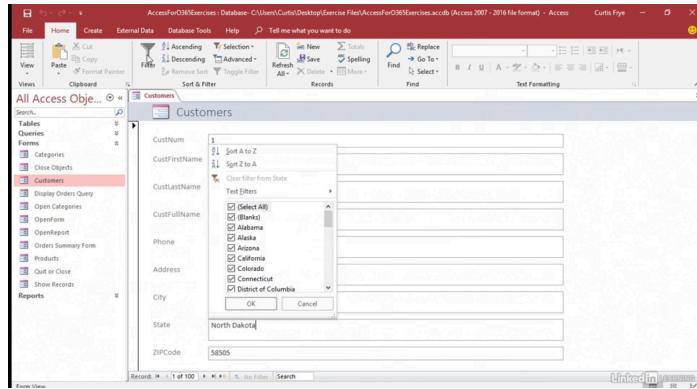


Abbildung 134:

Um diese Funktion nicht mehr zu erlauben.

```

1 Sub Form_Load()
2 Me.AllowFilter = False
3 End Sub

```

Wir das Formular neu geladen, so ist die Filter-Funktion ausgeblendet.

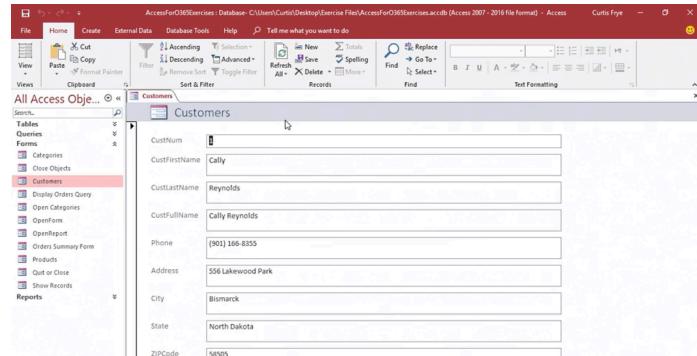


Abbildung 135:

2.9.2 Verfeinerung

Dynamische Beschriftung Der Hintergrund, sowie die Überschrift eines Formulars und Report können direkt über das `Me.` Objekt angesteuert werden.

Aktualisierung des Formulars Wird direkt ein Wert in der Tabelle hinterlegt, so ändert dies nicht automatisch die abgefragten Daten im dazugehörigen Formular. Mit `Me.Requery` wird unter `Form_Activate()` das Formular eine neue Abfrage starten, sobald es das aktive Fenster wird.

Teil IV

SQL

1 SQL - Conceptual

On: Youtube

1.1 Keys

1.1.1 Primäry Key

Ein Primärer Schlüssel ist eine Spalte, welche jede Zeile in einer Tabelle eindeutig definiert. Dieser Schlüssel unterliegt dabei zwei Eigenschaften

- Eindeutigkeit (Unique)
- Nicht Null (No Null)
- Nicht änderbar (No changing)

Es wird dafür meist eine Identitätsspalte eingefügt, welche eine Indexierung vornimmt. Diese muss aber nicht zwangsläufig der "Primäre Key" sein.

1.1.2 Composite Key

Eine Tabelle, in welcher keine Spalte für sich eine eindeutige Identifikationsmöglichkeit darstellt, benötigt ein Composite Key. Dieser beinhaltet mindestens eine zwei Spaltenreferenzen. Diese Kombination aus Spalten ergibt eine Eindeutigkeit einer jeden Zeile.

Composite Key	
ClassStudents	
ClassID	StudentID
7	3084
7	3072
203	3024

Abbildung 136:

In dem Beispiel ist jede Zeile identifizierbar, wenn ein Schlüssel aus beiden Spalten gebildet wird.

1.1.3 Forgein Key

Eine Tabelle, welche Werte aus einer anderen Tabelle abfragt, benötigte eine Referenz zu dieser benötigten Tabelle. Die Tabelle, aus der Werte benötigt werden, fällt unter der Kategorie **Eltern**. Die Tabelle, welche Werte aus der Eltern Tabelle benötigt, fällt unter der Kategorie **Kind**.

In dem Fall, dass eine Kind Tabelle eindeutige Zeile in der Eltern Tabelle referieren muss, wird ein **Forgein Key** benötigt. Dieser muss

- nicht null sein,
- eindeutig die Eltern Tabelle referieren.

Es ist jedoch möglich einen null Eintrag in der Forgein Key Spalte zu besitzen, welche nicht zur Spalte der Eltern Tabelle referiert. Daraus folgt, dass der Forgein Key nicht eindeutig sein muss und null sein kann. Diese ist ein Adopiv-Kind, welches nicht zu einer zu einer gewissen Zeile in der Primärschlüssel Spalte der Elterntabelle referiert.

1.1.4 Composite Key

Wenn eine Tabelle zugrunde liegt, in welcher keine Spalte für sich als Primär Key fungiert, so wird eine Primär Key oder auch Composite Key aus mehreren Spalten gebildet.

In der Tabelle mit Studenten-ID und Class-ID, gibt es keinen Primären Schlüssel. Die Spalten müssen verbunden werden, sodass ein eindeutiger Schlüssel für die Tabelle geschaffen wird.

ClassStudents	
classID	studentID
7	3084
7	3072
202	3024

Abbildung 137:

Es kann nämlich sein, dass in der Spalte mehrere gleiche Werte stehen.

Composite Key	
ClassStudents	
classID	studentID
7	3084
7	3072
202	3024

Abbildung 138:

Ein Composite Key ist somit ein Primäry Key, welcher sich über mehrer Spalten erstreckt.

1.1.5 Compound Key

Ein Compound Key ist fast gleich zum Composite Key mit dem Unterschied, dass die verbundenen Spalten selber wieder Foreign Key sind, und somit zu anderen Tabellen verweisen.



Abbildung 139:

1.2 Normalization

1.2.1 1NF - Atomicity

In diesem Kontext wir von Entities und Attributes gesprochen. Dabei werden Entities in Tabellen umgesetzt und Attributes als Spalten definiert. Die Atomizierung beschreibt, dass die Zielsetzung einer Tabelle sind in der Wahl der Spalten niederschlagen soll. Das bedeutet, Werte in der Spalte mit dem Hauptbezug keine mehreren Werte besitzen soll. In anderen Spalten sind wiederkehrende Werte akzeptable. In anderen Hauptbezug-Spalten/Spalten ist dies nicht akzeptable. Der Primärkey ist damit jedoch nicht zwangsläufig gemeint.

Wenn einem Instanz einer Tabelle mehrere Werte zugeordnet werden müssen, dann sollte dies in einer ergänzenden Tabelle geschehen, welche den Hauptzweck besitzt, das Attribute abzudecken. Jedoch gilt auch hier, dass Eindeutigkeit benötigt wird.

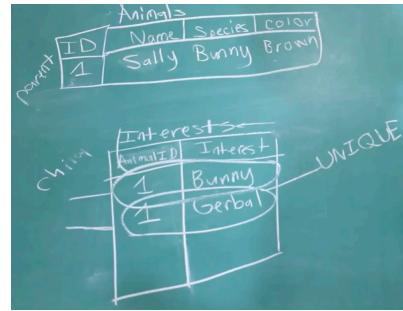


Abbildung 140:

Am Beispiel zeigt sich, dass Interests verwendet wird, Interessen für verschiedene Tiere zu speichern. Dabei kann ein Tier mehrere Interessen haben, jedoch in jeder Zeile steht nur ein Tier mit einem eindeutigen Interesse.

Eine Tabelle befindet sich in 1NF,

- wenn ein oder mehrere Attribute kein Duplikat eines anderen Attributes ist.
- wenn Instanzen eines Attributes keine Gruppierung des zugehörigen Attributes ist.

Das bedeutet, dass mehrere Ausprägungen eines Attributes nicht in einer Tabelle stehen sollen. Ebenso sollten auch diese auch nicht in einer Zelle gruppiert stehen. Diese sollen eindeutige Einträge in eine weitere Tabelle übertragen werden.

Problem:

Customers			Product		
Cust_ID	CustName	Phone	... other attributes	Product	Product
1	Fulton, Bruce	555-1212		Printer	Fax
2	Jones, Bill	555-1313		Fax	Modem

or:

Customers			Product	
Cust_ID	CustName	Phone	... other attributes	Product
1	Fulton, Bruce	555-1212		Printer
1	Fulton, Bruce	555-1212		Fax
1	Fulton, Bruce	555-1212		Modem
2	Jones, Bill	555-1313		Fax

or:

Customers			Product	
Cust_ID	CustName	Phone	... other attributes	Product
1	Fulton, Bruce	555-1212		Printer, Fax, Modem
2	Jones, Bill	555-1313		Fax

Abbildung 141:

Die Lösung zu dem Problem kann sein, dass eine Junction Tabelle erstellt wird.

Solution:

Customers				Products	
CustID	FName	LName	Phone	Prod_ID	Prod_Name
1	Bruce	Fulton	555-1212	1	Printer
2	Bill	Jones	555-1313	2	Fax

Junction Table

Customer_Products	
Cust_ID	Prod_ID
1	1
1	2
1	3
2	2

Abbildung 142:

Eine Juncion Tabelle wird nicht benötigt, wenn die ausgegliederte Tabelle sich nicht vollständig auflisten lässt.

1.2.2 2NF - Partial Dependency

Die zweite Normalisierung setzt die erste voraus. Als weiteren Punkt müssen alle Non-Key Attributes sich auf den Primären Schlüssel beziehen. Bei einem Primären Key, welcher aus einer Spalte besteht, ist 2NF gleich 1NF.

In einer Tabelle, welche einen Compound Key besitzen, müssen sich alle Non-Attribute auf alle Bestandteile des Primären Key beziehen. Im folgenden Beispiel ist dies nicht gegeben.

TABLE_PURCHASE_DETAIL		
Customer ID	Store ID	Purchase Location
1	1	Los Angeles
1	3	San Francisco
2	1	Los Angeles
3	2	New York
4	3	San Francisco

Abbildung 143:

Mit Hilfe der 2NF wir entweder die Tabelle getrennt oder die Daten nicht extra mit einbezogen.

TABLE_PURCHASE		TABLE_STORE	
Customer ID	Store ID	Store ID	Purchase Location
1	1	1	Los Angeles
1	3	2	New York
2	1	3	San Francisco
3	2		
4	3		

Abbildung 144:

1.2.3 3NF - Transitive Dependency

Die dritte Normalisierung setzt sich aus den beiden vorherigen zusammen. Aufbauend auf das vorherige folgt, dass ein Non-Key Attribute sich nicht auf ein anderes Non-Key Attribute beziehen soll.

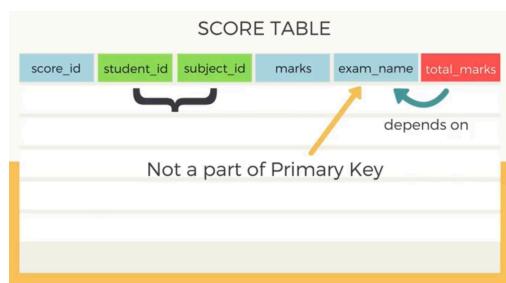


Abbildung 145:

Die gesamt Anzahl der Punkte ist Abhängig von der Art des Test. Diese Information sollte ausgeliedert werden.



Abbildung 146:

1.3 Junction

1.3.1 One-to-One Relationship

Trival.

1.3.2 One-to-Many Relationship

Trival.

1.3.3 Many-to-Many Relationship

Um eine Beziehung zwischen zwei Primärkeys herzustellen, welche in einer Many-to-Many Beziehung stehen, benötigt es eine intermediären Tabelle. In dieser besteht in keiner der Foreign Key Spalten eine Eindeutigkeit.

Am Beispiel der Beziehung zwischen Autorinnen und Büchern lässt sich zeigen, dass es eines zwei One-to-Many Beziehungen benötigt, um eine Many-to-Many Beziehung abzubauen.

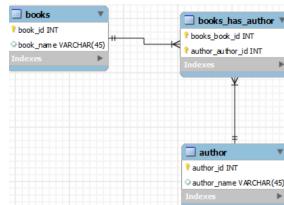


Abbildung 147:

Die Autorinnen können mehrere Bücher verfasst werden. Die Bücher können mehrer Autorinnen haben. In der Junction Tabelle wird die Permutation dargestellt, ohne dabei eine Tabelle zu erzeugen, in welcher mehrere Attribute von einem abgeleitet werden.

1.4 Data Integrity

Daten Integrität befasst sich mit Frage, wie können Datenbanken eine gewisse Stabilität und Verlässlichkeit gewähren. Dabei werden die vorherigen beschriebenen Themen verwendet. Daten Integrität lässt sich in drei Bereiche aufteilen.

- Entry Integrity
- Reference Integrity
- Domain Integrity

Die erste beiden nutzen als Grundlage die Primär und Foreign Keys.

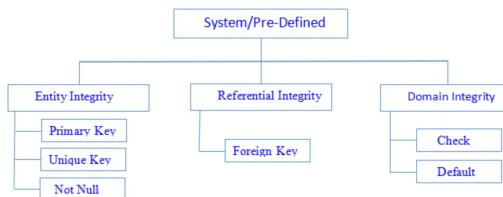


Abbildung 148:

Domain Integrity beschäftigt sich damit, das Einträge in die Tabellen die richtigen Formate und Rahmenbedingungen (Constraints) erfüllen.

2 SQL - Introduction

On: LinkedInLearning Eine Relational Database ist eine DB, in welcher Informationen in zweidimensionale Tabellen gespeichert werden.

2.1 SELECT Statement

Soll eine komplette Tabelle bezogen werden, so wird das Sternchen * verwendet. In der Beispieldatenbank befindet sich die Tabelle Countries. Sollen alle Spalten und Wert der Tabelle bezogen werden, wird * verwendet, um alle Spalten der Tabelle auszuwählen:

```
1 SELECT *
2 From Customers;
```

Result:						
CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK

Abbildung 149:

Ein Teilmenge der Spalten wird über die genaue Adressierung angesteuert. Die Interne Ordnung legt dabei die Reihenfolge der Spalten in der übergebenen Tabelle fest.

```
1 SELECT CustomerID, Adress
2 From Customers;
```

Result:	
CustomerID	Address
1	Obere Str. 57
2	Avda. de la Constitución 2222
3	Mataderos 2312
4	120 Hanover Sq.
5	Berguvvägen 8
6	Forsterstr. 57
7	24, place Kléber

Abbildung 150:

Wie unten beschrieben, können die Bezeichnungen der Spalten durch AS geändert werden.

```
1 SELECT CustomerID AS Index, Adress
2 From Customers;
```

Result:

Number of Records: 91

ID	'Adresse'
1	Obere Str. 57
2	Avda. de la Constitución 2222
3	Mataderos 2312
4	120 Hanover Sq.
5	Berguvvägen 8
6	Forsterstr. 57
7	24, place Kléber

Abbildung 151:

Für die Dauer der Abfrage kann der Spaltennamen und oder der Tabellennamen, der generierten Tabelle, mit **AS** geändert werden.

```
1 SELECT CustomerID AS ID, Address AS 'Adresse'  
2 From Customers;
```

Result:

Number of Records: 91

ID	'Adresse'
1	Obere Str. 57
2	Avda. de la Constitución 2222
3	Mataderos 2312
4	120 Hanover Sq.
5	Berguvvägen 8
6	Forsterstr. 57
7	24, place Kléber

Abbildung 152:

Mit Apostrophen können auch Schlüsselwörter als Spaltennamen verwendet werden.

2.1.1 FROM - Input table

Mit **FROM** wir die Quelle der SQL Anfrage bestimmt. Dies kann eine einfache Tabelle sein oder auch Tabelle, welche **Joins** zu anderen Tabellen besitzt.

2.1.2 WHERE - Where row is evaluated

Die Clause **WHERE** ist besser im Englischen zu verstehen. Select the following rows **where ... are equal to**

....

```
1 SELECT *  
2 FROM Customers  
3 WHERE City = 'Berlin';
```

Number of Records: 1

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany

Abbildung 153:

Es kann auch nach einzelnen Begriffen gesucht werden. Mit `Like` kann nach einzelnen Begriffen gesucht werden.

- Mit % können Begriffe einzeln betrachtet werden.
- Mit '%...%' wird der Begriff gesucht, egal was davor oder danach steht.
- Mit '%...' wird der Begriff gesucht, egal was davor steht.
- Mit '...%' wird der Begriff gesucht, egal was danach steht.

Ebenso können einzelnen Buchen an gewissen Stellen gesucht werden. Dafür nutzt man _.

```
1 SELECT Country
2 FROM Customers
3 WHERE Name Like '%Tim%' // Anfang oder Ende ist egal.
```

```
1 SELECT Country
2 FROM Customers
3 WHERE Name Like '_i%' // Zweites Zeichen ist i
```

Um mehrere Bedingungen festzulegen, wird der `IN()` Operator festgelegt.

```
1 SELECT Country
2 FROM Customers
3 WHERE Name IN('West', 'Ost')
```

2.1.3 GROUP BY - zusammenfassen von Spalten

Im einfachen Fall wird diese Clause genutzt um eindeutige Werte einer Spalte auszugeben.

```
1 SELECT Country
2 FROM Customers
3 GROUP BY Country
```

Result:	
Number of Records: 21	
Country	
Argentina	
Austria	
Belgium	
Brazil	
Canada	
Denmark	
Finland	
France	

Abbildung 154:

2.1.4 ORDER BY - sort table

Zum Ordnen einer Tabelle wird `Order by` verwendet. Danach wir der Spaltennamen eingetragen, nach dem absteigend geordnet werden soll.

```
1 SELECT *
2 From Country
3 Order by Name;
```

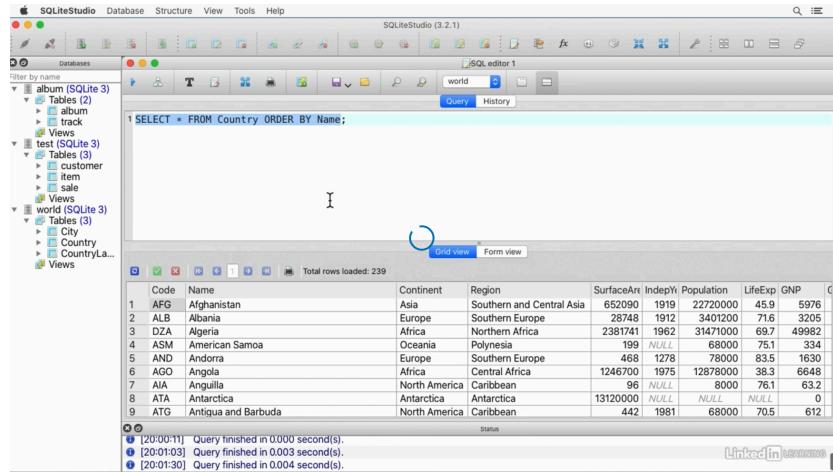


Abbildung 155:

Anstatt * können auch einzelne Spaltennamen adressiert werden.

```

1 SELECT      Name , Code
2 From       Country
3 Order by   Name;
```

Es wird bei nicht wie bei M Formula Language [...] benötigt, um Spalten zu adressieren.

Wie schon erwähnt, soll die Spaltenbezeichnung geändert werden, kann dies für jede einzelne Spalte getan werden.

```

1 SELECT      Name AS "Name Neu" , Code AS "Country_Code"
2 From       Country
3 Order by   Name_Neu;
```

In diesem Fall verhält es sich ähnlich wie in M Formula Language. Für einen String als Spaltennamen werden Anführungszeichen verwendet.

```

1 SELECT      Name AS Name2 , Code
2 From       Country
3 Order by   Name2;
```

Mit Hilfe einer Aggregatfunktion können Werte für die gruppierte Spalte ausgegeben werden.

```

1 SELECT Count(Quantity) , ProductID
2 FROM OrderDetails
3 Group by ProductID;
```

Number of Records: 77	
ProductID	Expr1001
1	8
2	11
3	2
4	5
5	4
6	2
7	2
8	2

Abbildung 156:

Die Anzahl der Aggregationen ist nicht beschränkt.

```
1 SELECT SUM(Quantity), Count(ProductID)
2 FROM OrderDetails
3 GROUP BY ProductID;
```

Zu Vermerken ist, dass nur die Spalten generiert werden, welche auch durch `SELECT` ausgewählt werden.

```
1 SELECT ProductID, SUM(Quantity) AS SUM_Quantity, Count(ProductID) AS Count_Qunatity
2 FROM OrderDetails
3 GROUP BY ProductID;
```

Number of Records: 77

ProductID	SUM_Quantity	Count_Qunatity
1	159	8
2	341	11
3	80	2
4	107	5
5	129	4
6	36	2
7	25	2
8	140	2

Abbildung 157:

2.1.5 LIMIT - Who many Rows

Mit dem Ausdruck `Limit` kann die Anzahl der Zeilen reglementiert werden. Dieser Ausdruck kommt nach `Order BY`.

```
1 SELECT Name, Contient, Region // column name
2 From Country // table name
3 WHERE Country = 'Germany'
4 ORDER BY Name
5 LIMIT 5
```

2.1.6 OFFSET - Ausschluss der ersten Zeilen

Die Auswahl der Zeilen kann auch noch der Indexierung der Tabelle erfolgen. Mit dem Begriff `OFFSET` wird eine Auswahl der der letzten $m - n$ Zeilen getroffen. Wobei m die Anzahl der gesamten übergeben Tabellenanzahl ist und n der zu reduzierenden Zeilen ist.

```
1 SELECT Name, Contient, Region // column name
2 From Country // table name
3 WHERE Region = 'West'
4 ORDER BY Name
5 OFFSET 5 // Ersten 5 Zeilen werden nicht berücksichtigt.
```

2.2 Concepts

2.2.1 Function Types

Aggregatfunktionen haben die folgende Syntax

```
1 SELECT Funktionstype(Column) From Table
```

2.2.1.1 Count() Die Funktion **Count()** zählt die Zeilen der generierten Tabellen. Es können dabei zwischen der ganzen Tabelle und einzelnen Spalten unterschieden werden. Wählt man einzelnen Spalten aus, werden nicht leere Zeilen gezählt.

Count() angewandt auf die gesamt Tabelle

```
1 SELECT Count(*)
2 From Country // table name
```

generiert eine Tabelle mit dem Spaltentitel "Count(*)", in welcher in der ersten Zeile der gesuchte Wert steht.

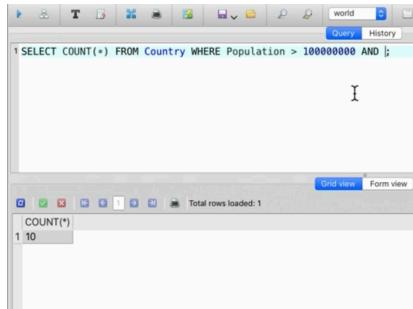


Abbildung 158:

Die Einschränkung auf eine gewisse Spalte, gibt die Anzahl der Zeilen, welche nicht null sind, wieder.

```
1 SELECT Count(Region)
2 From Country
```

Ebenso können weitere Einschränkungen und Überprüfungen an der Tabelle vorgenommen werden.

```
1 SELECT Count(Region), Contient
2 From Country
3 WHERE Contient = 'Europe'
```

Manche Anpassungen haben kein Effekt auf das Ergebnis der Funktion.

```
1 SELECT Count(*)
2 From Country // table name
3 ORDER BY Name
```

Sollen unterschiedliche Einträge verglichen werden, wird **DISTINCT** als Zusatz verwendet, siehe **DISTINCT Clause**

2.2.1.2 Aggregatfunktionen Zu den Aggregatfunktion gehört

- AVG()
- MAX(), MIN()
- SUM(),
- Count()

2.2.2 DISTINCT Clause

Innerhalb einer Aggregatfunktion werden nur alle ungleichen, nicht leeren Zeilen ausgewählt.

In der Funktion **Count()** angewandt, werden nur Zeilen gezählt, welche nicht gleich sind.

```
1 SELECT Count(DISTINCT Quantity)
2 FROM OrderDetails;
```

Die Clause kann auch vor einem Spaltennamen in der **SELECT** Clause angewandt werden.

```
1 SELECT DISTINCT Quantity
2 FROM OrderDetails;
```

Quantity
1
2
3
4
5

Abbildung 159:

Werden weitere Spalten hinzugefügt, bezieht sich die Eindeutigkeit auf die gesamte Zeilenbreite der Tabelle.

```
1 SELECT DISTINCT Quantity, ProductID
2 FROM OrderDetails;
```

Quantity	ProductID
1	19
1	37
1	69
2	13
2	17
2	26
2	40

Abbildung 160:

Im Vergleich:

```
1 SELECT Quantity, ProductID
2 FROM OrderDetails
3 ORDER BY Quantity;
```

sind 518 Zeilen generiert worden, hingegen nur 433 unter Berücksichtigung der Eindeutigkeit.

Quantity	ProductID
1	69
1	37
1	19
2	17
2	56

Abbildung 161:

2.2.3 CREATE TABLE ... ()

Um eine Tabelle zu erstellen, wird **CREATE Table** verwendet. Die Schema hinter diesem Ausdruck beschreibt die Spaltennamen und Typen. Dabei wird zuerst der Spaltenname genannt und danach der Typ.

```
1 CREATE TABLE Tabellenname (
2 Spaltenname_1 TEXT,
3 Spaltenname_2 INTEGER,
4 Spaltenname_3 BOOLEAN
5 );
```

Neue Werte können über **INSERT INTO** eingepflegt werden.

```
1 INSERT INTO Tabellenname  
2 (Spaltenname_1, Spaltenname_2)  
3 Value ('text', 4, True);
```

2.2.4 DROP TABLE ...

```
1 DROP TABLE Test;
```

Wenn die Tabelle nicht existiert, kommt es zu einem Fehler. Dieser kann umgangen werden, indem **IF EXISTS** vor dem Tabellennamen geschrieben wird.

```
1 DROP TABLE IF EXISTS Test;
```

2.2.5 INSERT INTO ... ()()

Daten können auch an Tabellen angefügt werden. Die **INSERT INTO** Clause baut sich in drei Teile auf

- Tabllenbezug
- Spaltenbezug
- Werte

```
1 INSERT INTO Customers  
2 (Spalten_1, Spalte_2)  
3 VALUE ('Test', 2);
```

Wir die Spalte nicht erwähnt, wir der Null-Wert eingesetzt.

```
1 INSERT INTO Customers  
2 (Spalte_2)  
3 VALUE (2);
```

Werte aus anderen Tabellen können über **SELECT** eingefügt werden. Anstatt **Value** wird das Select Statement verwendet.

```
1 Creat table Test  
2 (a integer, b text, c text);  
3 INSERT INTO Test  
4 Select ID, Name, Adress From Customer;  
5 Select * From Test;
```

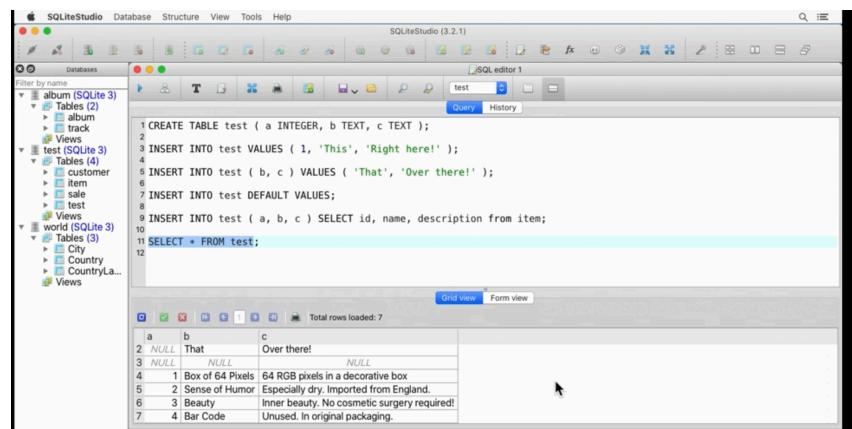


Abbildung 162:

2.2.6 UPDATE ... SET ... WHERE

Update wirkt auf die gesamte Tabelle. Daher ist es wichtig, die Bedingung oder Bedingungen festzulegen, welche Zeilen direkt angesprochen werden sollen. Der Aufbau des Ausdruckes zieht wie folgt aus:

- Tabellenbezug
- SET Spaltenbezug mit Änderung
- WHERE Bedingung

```
1 UPDATE Customers
2 SET Spalte_1 = 'Test', Spalte_3 = 'Test'
3 WHERE Spalte_2 = 'Tesst';
```

Wenn keine Bedingung angegeben ist, werden alle Werte in einer spezifischen Spalte geändert.

```
1 UPDATE Customers
2 SET Spalte_1 = 'Test';
```

2.2.7 DELETE FROM ... WHERE

Unterscheidet sich zu [UPDATE](#) in der Kategorie, dass ganze Zeilen gelöscht werden, nicht nur einzelne Werte. Dies kann über [UPDATE](#) erfolgen.

```
1 DELETE
2 FROM Customers
3 WHERE id = 5;
```

In dem obigen Beispiel wird die Zeile mit Index 5 gelöscht.

2.2.8 NULL Value

Um Zellen auf den Eintrag [null](#) zu testen, kann in Structured Query Language (SQL) nicht nach null gesucht werden.

```
1 WHERE id = null // Gilt nie. Die Tabelle bleibt leer.
```

Damit nach [null](#) Werte gesucht werden kann, muss folgende Überprüfung erfolgen:

```
1 WHERE id IS NULL
```

Die Negation lautet [IS NOT NULL](#).

Diese Beschreibung wird auch verwendet, wenn Datentypen für Spalten festgelegt werden.

```
1 CREATE TABLE test (
2 a Integer NOT NULL,
3 b Text NOT NULL,
4 c Text
5 );
6
7 INSERT INTO test Value
8 (1, 'abs', 'sde');
9
10 INSERT INTO test (b) Values ('asd'); // Ruft einen Fehler hervor, weil a NOT NULL fordert.
```

2.2.9 DEFAULT

Wie [NOT NULL](#) kann eine Default Bedingung erstellt werden.

```
1 DROP TABLE IF EXIST test;
2 CREATE TABLE test (
3 a Integer DEFAULT 0,
4 b Text NOT NULL,
5 c Text
6 );
```

2.2.10 UNIQUE

Siehe [DEFAULT](#)

```
1 DROP TABLE IF EXIST test;
2 CREATE TABLE test (
3     a Integer UNIQUE,
4     b Text NOT NULL,
5     c Text
6 );
```

2.2.11 ALTER TABLE ... ADD

Um Spalten zu einer Tabelle hinzuzufügen, wird die [ALTER TABLE ... ADD](#) Funktion verwendet.

```
1 CREATE TABLE test (
2     a Integer UNIQUE,
3     b Text NOT NULL,
4     c Text
5 );
6
7 ALTER TABLE test ADD d test;
```

Ebenso kann auch hier gleich ein [Default](#) Wert mit [ADD](#) d test DEFAULT 'panda' hinterlegt werden.

2.2.12 CASE - Conditional Expression

Um einen Werte in einer Spalte zu überprüfen, kann [CASE](#) verwendet werden. Im einfachen Fall wird eine Überprüfung der Werte einer Spalte durchgeführt. Mit [CASE](#) können mehrere Abfragen gestellt werden. Das Resultat kann in die gleiche Spalte oder eine andere geschrieben werden.

Die Werte, die gespeichert werden sollen, werden unter [End](#) abgespeichert. Der Aufbau der Funktion ist.

```
1 CASE (Spalte) WHEN (Bewertung) THEN (...)
```

```
2 ELSE (...)
```

```
3 END (Speicherort)
```

```
1 SELECT department, salary, name
2 CASE department WHEN 50 THEN 1,5*salary
3 WHEN 60 THEN 2,0*salary
4 END 'Finaly_Salary'
5 From Employee;
```

```

SELECT first_name, department_id, salary,
       CASE department_id WHEN 50 THEN 1.5*salary
                           WHEN 12 THEN 2.0*salary
                           ELSE salary
       END "REVISED SALARY"
FROM Employee;

```

FIRST_NAME	DEPARTMENT_ID	salary	REVISED SALARY
Vipul	50	30000	45000
Amit	12	27000	54000
Satish	12	3500	7000
Harshey	23	7300	7300
Archit	50	2950	4425

5 rows returned in 0.00 seconds [CSV Export](#)

Abbildung 163:

Die Funktion lässt zwei Möglichkeiten zu:

- ```

1 CASE (Spalte) WHEN (Bewertung) THEN (...)

2 ELSE (...)

3 END AS (Neuer Speicherort)
```

---



---

```

1 CASE WHEN (Spalten bezogene Bewertung) THEN (...)

2 ELSE (...)

3 END AS (Neuer Spaltenname)
```

---

### 2.2.13 Ansteuern von Spalten

In SQLite wird die Syntax [] für Spalten verwandt. Dabei können Spaltennamen auch ohne Eckige Klammern geschrieben werden, wenn der Tabellennamen angegeben wird.

---

```

1 SELECT Substr([a],1,2)
2 From t;
3 -- oder
4 SELECT Substr(a,1,2)
5 From t;
```

---

## 2.3 Joints

Die Relationsfunktion fußen auf den bestehenden Relation der Mengenlehre.

### 2.3.1 Join on or more tables

Die Joins sind wie folgt aufgebaut:

- Select aller Spalten die angezeigt werden sollen. Dabei können die Spalten aus den verbundenen Tabellen angesteuert werden. Ebenso können auch die Spalten der zugrundeliegenden Tabelle angezeigt werden.
  - Jede Tabelle kann einzeln eingesteuert werden *Tabellename*. Die Auswahl der Spalten erfolgt über den Operator ..

- Die Möglichkeit alle Spalten anzusprechen erfolgt ebenso den bekannten Operator \*.

---

```
1 Select customer.name AS "Customer-Name", item.name AS "Item-Name", sale.*
```

---

- Mit **From** wird die zugrundeliegende Tabelle angesteuert.

- Es können jetzt mehrere **Joins** angefügt werden. Der Aufbau eines **Joins** ist, dass an Ersterstelle die angefügten Tabelle steht. Danach wird festgelegt, über welchen Schnittpunkt die Tabellen verbunden werden.

---

```
1 Join customer ON sale.customer_id = customer.id
2 Join item ON sale.item_id = item.id;
```

---

- Weitere Funktionalitäten können angefügt werden

---

```
1 WHERE sale.id = 2;
```

---

Mit Joinsfunktion erlaubt so, dass verschiedene Datensätze verbunden werden.

---

```
1 --Ziel ist es die Namen aus der customer Tabelle zu laden.
2 Select customer.name AS "Customer-Name", item.name AS "Item-Name", sale.*
3 From sale
4 Join customer ON sale.customer_id = customer.id
5 Join item ON sale.item_id = item.id
6 ;
```

---

|   | Customer-Name | Item-Name        | id | item_id | customer_id | date       | quantity | price |
|---|---------------|------------------|----|---------|-------------|------------|----------|-------|
| 1 | Mary Smith    | Box of 64 Pixels | 1  | 1       | 2           | 2009-02-27 | 3        | 2995  |
| 2 | Mary Smith    | Sense of Humor   | 2  | 2       | 2           | 2009-02-27 | 1        | 1995  |
| 3 | Bill Smith    | Box of 64 Pixels | 3  | 1       | 1           | 2009-02-28 | 1        | 2995  |
| 4 | Bob Smith     | Bar Code         | 4  | 4       | 3           | 2009-02-28 | 2        | 999   |
| 5 | Mary Smith    | Box of 64 Pixels | 5  | 1       | 2           | 2009-02-28 | 1        | 2995  |

Abbildung 164:

### 2.3.2 Kinds of Joins

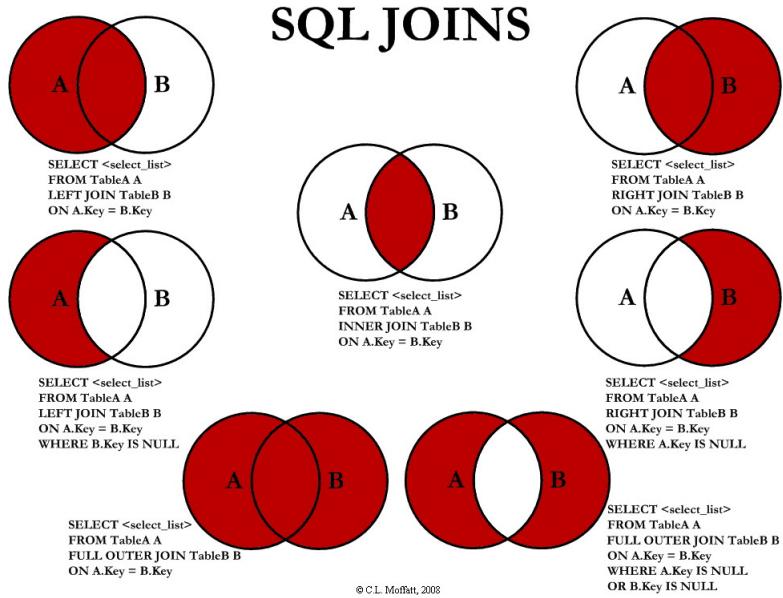


Abbildung 165:

## 2.4 Selection of Function

#### 2.4.1 Find the lenght of a string

Die Funktion `LENGTH()` gibt die Länge eines String-Values wieder.

```
1 -- Bestimmen der Länge einer Spalte: Leerzeichen werden auch mitgezählt.
2 Select id, name, Length(name) AS 'Länge der Spalte Name'
3 From customer
4 ;
```

|   | id | name       | Länge der Spalte Name |
|---|----|------------|-----------------------|
| 1 | 1  | Bill Smith | 10                    |
| 2 | 2  | Mary Smith | 10                    |
| 3 | 3  | Bob Smith  | 9                     |

Abbildung 166:

#### 2.4.2 Substring of Field of Date

Datumseinträge können in Teil Einträge zerlegt werden.

```
1 Select date,
2 Substr(date,1,4) AS Year,
3 Substr(date,6,2) AS Month,
4 Substr(date,9,2) AS Day
5 From sale
6 ;
```

|   | date       | Year | Month | Day |
|---|------------|------|-------|-----|
| 1 | 2009-02-27 | 2009 | 02    | 27  |
| 2 | 2009-02-27 | 2009 | 02    | 27  |
| 3 | 2009-02-28 | 2009 | 02    | 28  |
| 4 | 2009-02-28 | 2009 | 02    | 28  |
| 5 | 2009-02-28 | 2009 | 02    | 28  |

Abbildung 167:

#### 2.4.3 Count()- Group by (aggregate function)

Die `Count()` Funktion auf eine Spalte oder eine Tabelle zählt die nicht leeren Einträge. Angewandt auf eine Tabelle, wir die Anzahl der Zeilen dargestellt. Erst mit der Kombination `Group by` wirkt die Funktion wie ein Funktion, welche ein Parameter übergeben bekommen wird.

##### 2.4.3.1 Count(), Group by - same coloum

Für jeden aggregierte Eintrag (grouped), wir die Anzahl ermittelt. Ohne dies zählt die `Count()` Funktion nur die Zeilen einer Tabelle.

```
1 Select state, Count(state)
2 From customer
3 Group by state
4 ;
```

##### 2.4.3.2 Count(), Group by - different coloum

Die `Count()` Funktion funktioniert, nicht anders, wenn sie auf eine spezifische Spalte angewandt wird. Die Funktion zählt nur die aggegierten Zeilen, für jede Tabelle.

```
1 Select state, Count(address) -- Wie viel unterschiedliche Adress gibt es in jedem Staat.
2 From customer
3 Group by state
4 ;
```

#### 2.4.4 Mix

**Trim** Die Trim-Funktionen werden weiter unterteilt. Der Kern ist, dass entweder Leerzeichen oder andere Zeichen aus dem Textfeld entfernt werden. Dabei unterscheidet **Trim()** in linksseitige **LTrim** und rechtseitige **RTrim** Funktionen.

**Case Function** Strings können in groß oder klein Darstellung der ASCII-Zeichen umgewandelt werden.

**Typeof()** gibt den Typ des Eintrages wieder.

**Datetime** Die **Datetime()** Funktion nimmt als Input Variablen *Strings*. Das aktuelle Datum wird mit '*now*' beschrieben.

---

```
1 Select Datetime('now','+1 day'),
```

---

## 2.5 Transaction

Zugriffe auf die Datenbank können in Packete geschnürt werden. Diese Transaktionen Starten und Enden mit einem Befehl. Dieser kann sich von SQL Datenbanksystem zu Datenbanksystem anders sein. Transaktionspackete sichern

- Daten Integrität
- Performance

In SQLite gilt

---

```
1 Begin Transaction;
2 ...
3 End Transaction;
```

---

Geht etwas schief, oder werden bestimmte Kriterien nicht erfüllt, so kann mit **Rollback** die bisherige Transaktion rückgängig gemacht werden.

---

```
1 Begin Transaction;
2 ...
3 Rollback;
4 ...
5 End Transaction;
```

---

Als Beispiel dient die folgende Übung:

---

```
1 -- 02 transactions
2 -- test.db
3
4 CREATE TABLE widgetInventory (
5 id INTEGER PRIMARY KEY,
6 description TEXT,
7 onhand INTEGER NOT NULL
8);
9
10 CREATE TABLE widgetSales (
11 id INTEGER PRIMARY KEY,
12 inv_id INTEGER,
13 quan INTEGER,
14 price INTEGER
15);
16
17 INSERT INTO widgetInventory (description, onhand) VALUES ('rock', 25);
18 INSERT INTO widgetInventory (description, onhand) VALUES ('paper', 25);
19 INSERT INTO widgetInventory (description, onhand) VALUES ('scissors', 25);
20
21 SELECT * FROM widgetInventory;
22 SELECT * FROM widgetSales;
23
24 BEGIN TRANSACTION;
25 INSERT INTO widgetSales (inv_id, quan, price) VALUES (1, 5, 500);
26 UPDATE widgetInventory SET onhand = (onhand - 5) WHERE id = 1;
```

```

27 END TRANSACTION;
28
29 BEGIN TRANSACTION;
30 INSERT INTO widgetInventory (description, onhand) VALUES ('toy', 25);
31 ROLLBACK;
32 SELECT * FROM widgetInventory;

```

---

## 2.6 Triggers

Triggers erlauben bestimmt Aktionen durchzuführen, wenn ein Ereignis an einer Tabelle durchgeführt wird. Die verschiedenen Trigger werden in SQLite an die Tabelle angefügt.

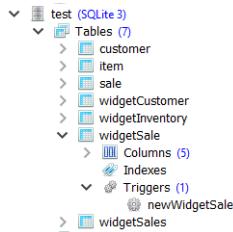


Abbildung 168:

Ein Trigger wird mit dem Namen, den Auslöser und den Bezug initialisiert.

```

1 Create TRIGGER [name] [Event] ON [Tabelle]
2 Begin
3 ...
4 End
5 ;

```

---

Im gewählten Beispiel ist der Auslöser das Einfügen einer neuen Zeile.

```

1 Create TRIGGER NewCustomerID After INSERT ON widgetSales
2 Begin
3 UPDATE widgetCustomer -- Die zu modifizierte Tabelle
4 SET last_order_id = NEW.id -- New. Ist das Objekt, welche generiert wird, wenn eine
 neue Zeile in widgetSales eingelegt wird.
5 WHERE widgetCustomer.id = NEW.customer_id --
6 ;
7 End
8 ;

```

---

Die Trigger Funktion kann verwendet werden, für das Ergänzen von Daten. Diese können beispielsweise nach dem Einfügen einer Zeile ausgelöst werden.

```

1 -- 01 update triggers
2 -- test.db
3 Drop Table if exists widgetSale;
4 Drop Table if exists widgetCustomer;
5
6 CREATE TABLE widgetCustomer (id INTEGER PRIMARY KEY, name TEXT, last_order_id INT, stamp
 TEXT);
7 CREATE TABLE widgetSale (id INTEGER PRIMARY KEY, item_id INT, customer_id INT, quan INT,
 price INT, stamp TEXT);
8
9 INSERT INTO widgetCustomer (name) VALUES ('Bob');
10 INSERT INTO widgetCustomer (name) VALUES ('Sally');
11 INSERT INTO widgetCustomer (name) VALUES ('Fred');
12
13 SELECT * FROM widgetCustomer;
14 Select * From widgetSale;
15
16 CREATE TRIGGER stampSale After Insert ON widgetSale
17 BEGIN
18 UPDATE widgetSale SET stamp = Datetime('now') WHERE id = NEW.id;
19 UPDATE widgetCustomer Set last_order_id = New.id, stamp = DateTIme('now')

```

```

20 Where widgetCustomer.id = New.customer_id;
21 END
22 ;
23
24 INSERT INTO widgetSale (item_id, customer_id, quan, price) VALUES (1, 3, 5, 1995);
25 INSERT INTO widgetSale (item_id, customer_id, quan, price) VALUES (2, 2, 3, 1495);
26 INSERT INTO widgetSale (item_id, customer_id, quan, price) VALUES (3, 1, 1, 2995);
27 SELECT * FROM widgetSale;
28 SELECT * FROM widgetCustomer;

```

---

Was zu beachten ist, dass der **Trigger** selbst ausgeführt werden muss. Dabei wird dieser in unter

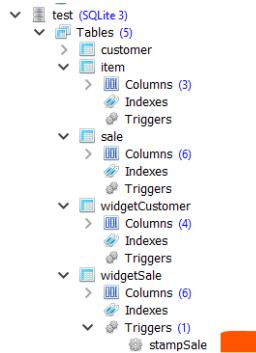


Abbildung 169:

abgespeichert. Das Objekt **NEW** steuert den Record der letzten Zeile der vorgegebenen Tabelle, in dem Fall *widgetSale*. Die Spalten werden dann als Attribute des Objektes **New** gesehen.

## 2.7 Subselects

Abfragen können ebenfalls als temporäre Tabelle betrachtet werden.

```

1 CREATE TABLE t (a TEXT, b TEXT);
2 INSERT INTO t VALUES ('NY0123', 'US4567');
3 INSERT INTO t VALUES ('AZ9437', 'GB1234');
4 INSERT INTO t VALUES ('CA1279', 'FR5678');
5 SELECT * FROM t;
6
7 -- Trennen der Strings
8 SELECT SUBSTR([a],1,2) AS State,
9 SUBSTR([b],1,2) AS Country,
10 SUBSTR([a],3) AS State_Code,
11 SUBSTR([b],3) AS Country_Code
12 From t;
13
14 Select * From Country;
15
16 -- Suche Country-Name mit SELECT als Tabelle
17 SELECT Country.Name, st.Country_Code
18 From (SELECT SUBSTR([a],1,2) AS State,
19 SUBSTR([b],1,2) AS Country,
20 SUBSTR([a], 3) AS State_Code,
21 SUBSTR([a], 3) AS Country_Code
22 From t) AS st
23 Join Country
24 On Country.Code2 = st.Country;
25
26 DROP TABLE t;

```

---

### 2.7.1 IN Operator

Mit dem In-Operator können mehrere Bedingungen geprüft werden. Das bedeutet

---

```

1 SELECT *
2 From Country
3 WHERE Name IN(Deutschland, Japan, Frankreich); -- Die Syntax ist nicht = In(...)
```

---

In der Spalte [Name] wird nach den Werten *Deutschland*, *Japan* oder *Frankreich* gesucht. Das gleiche Ergebnis ist auch erreicht, wenn zwei **OR** Operatoren verwendet werden würden. Mit Subselect kann eine Liste als Speicherort für multiple Bedingungen verwendet werden.

---

```

1 SELECT *
2 From Country
3 WHERE Code2 IN((SELECT SUBSTR([b],1,2) AS Country_Code, SUBSTR([b],1,1))
4 From t))
5 ;
```

---

Die Bezugstabelle muss als Liste vorliegen. Das bedeutet, eine Tabelle kann nicht vom IN() Operator verwendet werden.

## 2.7.2 View - temporary table

Subselect können in View gespeichert werden. Diese befinden sich als Unterpunkt unter **Tabellen**.



Abbildung 170:

Über *Drop View Beispielname* werden diese entfernt. Der Befehl zur Generierung sieht wie folgt aus:

---

```

1 CREATE VIEW view_name AS
2 SELECT column1, column2, ...
3 FROM table_name
4 WHERE condition;
```

---

Beispiel *track*:

|    | id | album | ic                                    | title | track number | duration |
|----|----|-------|---------------------------------------|-------|--------------|----------|
| 1  | 1  | 1     | Bright Lights Big City                |       | 1            | 320      |
| 2  | 2  | 1     | One More Life                         |       | 2            | 244      |
| 3  | 3  | 1     | Basin Street Blues                    |       | 5            | 296      |
| 4  | 4  | 1     | California                            |       | 3            | 205      |
| 5  | 5  | 1     | Take Five                             |       | 4            | 238      |
| 6  | 6  | 1     | Georgia On My Mind                    |       | 6            | 280      |
| 7  | 7  | 1     | Rainy Day Blues                       |       | 7            | 343      |
| 8  | 8  | 1     | It's Only Rock'n Roll A Little Bit    |       | 8            | 256      |
| 9  | 9  | 1     | Ain't Nobody's Business               |       | 9            | 447      |
| 10 | 10 | 1     | I That's All                          |       | 10           | 368      |
| 11 | 11 | 1     | Don't Goode                           |       | 11           | 255      |
| 12 | 12 | 1     | Lover Man                             |       | 2            | 185      |
| 13 | 13 | 1     | Blue Suede Shoes                      |       | 3            | 266      |
| 14 | 14 | 1     | Yesterdays                            |       | 4            | 469      |
| 15 | 15 | 1     | Blue Suede Shoes                      |       | 5            | 150      |
| 16 | 16 | 11    | The Queen                             |       | 5            | 150      |
| 17 | 17 | 11    | Sat. Pepper's Lonely Hearts Club Band |       | 6            | 76       |

Abbildung 171:

Die Bewertung der Dauer wird in der View Tabelle *DurationView* gespeichert.

## 3 Object-Oriented Design

### 3.1 Objekt Orientierte Grundlagen

#### 3.1.1 Beispiel

Die Grundlagen für OOP werden in dem Kurs besprochen. Es soll dabei darum gehen, die Logik hinter den Prinzipien von OOP zu verstehen.

Es wird versucht die Vorteile zu beschreiben und die Abwegung zwischen verschiedenen Konzepten wie Funktionales-Programmierung aufzuzeigen.

Ein kurzes Einführungsbeispiel ist das Kuchen-Backen. Wir können das Backen eines Kuchens als Abfolge verschiedener Prozesse beschreiben, welches zu dem Resultat, einen fertiger Kuchen, führt. Beschreiben wir jedoch den Backvorgang mit OOP, dann werden die verschiedenen Gerätschaften beschrieben, mit dem was man mit ihnen tun kann.

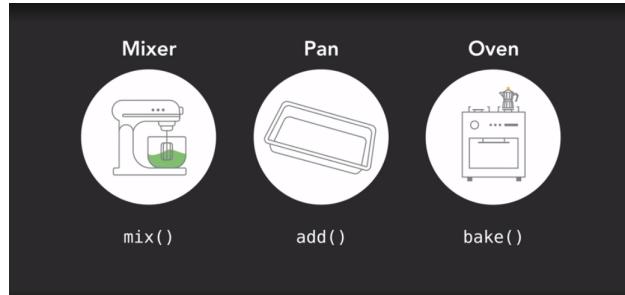


Abbildung 172:

### 3.1.2 Objekte

Es wird zuerst ein **Objekt** beschrieben. Diese kann verschiedenen **Attribute** besitzen. Oft wird für Attribute auch andere Begrifflichkeiten verwendet:

$$\text{Attribute} \cong \text{Eigenschaften, Status, Variable, Daten etc}^1 \quad (3.1)$$

Im Abstraktenraum sprechen wir von Objekten die Attribute besitzen und mit denen man **Methoden** vollziehen kann. Auch hier werden oft verschiedenen Begrifflichkeiten verwendet:

$$\text{Methoden} \cong \text{Operationen, Funktionen etc.} \quad (3.2)$$

Eine Methode ist dabei nur verwenbar von einem Objekt der zugehörigen Klasse, in welcher die Methode definiert wurde. Zwei Objekte vom Typ Konto können wir folgt aussehen:

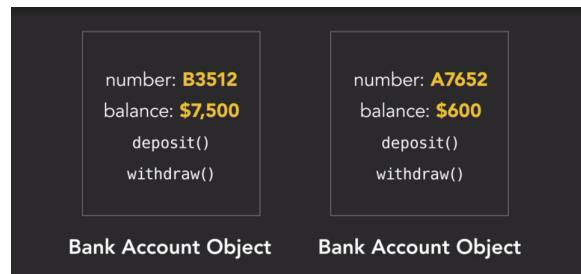


Abbildung 173:

Auf das Objekte Bankkonto mit der Nummer *B3512* können wir die Operation *'deposit()'* ausführen. Dies gilt für alle Bankkonten.

### 3.1.3 Klassen

Die Blaupausen für die Objekte sind die **Klassen**. Eine Klasse enthält alle Bestandteile für ein Objekte. Ein Objekt wird erst durch eine Klasse erstellt, somit kann eine Klasse mehrere Objekte besitzen. Die Klasse besitzt einen Namen:

$$\text{Typ} \cong \text{Name} \quad (3.3)$$

und Attribute (Properties) sowie Methoden (Operations)

## Class Components



Eine Klasse wird meist in kompakterer Schreibweise geschrieben:



Abbildung 174:

Wird ein Objekt erzeugt, so nennt man den Vorgang **Instanzierung**. Ein Objekt wird auch manchmal **Instanz** genannt.



Abbildung 175:

Die verschiedenen Sprachen habe bereit verschiedene Classen definiert. So zum Beispiel ist ein String, Boolen oder andere Datentypen Classen in den verschiedenen Sprachen. Diese Klassen werden zu **Bibliotheken** zusammengefasst.

## Frameworks and Libraries

- Java Class Library
- .NET Framework BCL
- C++ Standard Library
- Ruby Standard Library
- Python Standard Library

Abbildung 176:

### 3.1.4 Abstraction, Encapsulation, Inheritance, Polymorphismen

**3.1.4.1 Abstraction** Zum Beschreiben einer Klasse werden nur die wichtigsten Informationen benötigt und die in der einfachsten Form.

**3.1.4.2 Encapsulation** Wir ein Objekt erstellt (instanziert), dann werden die Attribute definiert. Wollen wir aber auf die Attribute zugreifen, so wird der Zugang nur mit Hilfe von Methoden erlaubt. Das bedeutet, soll das Attribut geändert werden, so muss die dafür definierte Methode aufgerufen werden.

In dem Beispiel wollen wir ein Keks aus dem Objekt Keksdose entnehmen, dafür wird jedoch die Methoden `requestCookie()`:

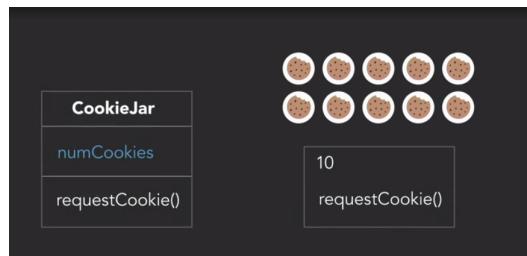


Abbildung 177:

Im Code wird dies durch den Ausdruck **private** hervorgerufen.

```
1 class Account {
2 private int account_number;
3 private int account_balance;
4
5 public void showData() {
6 //code to show data
7 }
8
9 public void deposit(int a) {
10 if (a < 0) {
11 //show error
12 } else
13 account_balance = account_balance + a;
14 }
15 }
```

Eine Variable kann somit nicht außerhalb der Klasse aufgerufen oder verändert werden. Die Variable ist nur in der Klasse zugänglich. Wollten wir zum Beispiel einen Konto aufrufen und einen negativen Betrag von 100 Euro eintrag, würde dies nicht gehen.

```
1 Account a = new Account();
2 a.account_number = -100;
```

Diese Operation könnte somit nicht ausgeführt werden, weil die Variable nicht außerhalb der Klasse verändert werden kann, sondern nur folgendes geht:

```
1 Account a = new Account();
2 a.deposit(-100);
```

Der gesamte Code kann so als Kapsel verstanden werden.

**3.1.4.3 Inheritance** Die **Vererbung** von Klassen-Komponenten ist eine sinnvolle Funktion, um weitere Abstraktionsebenen zu erlauben:

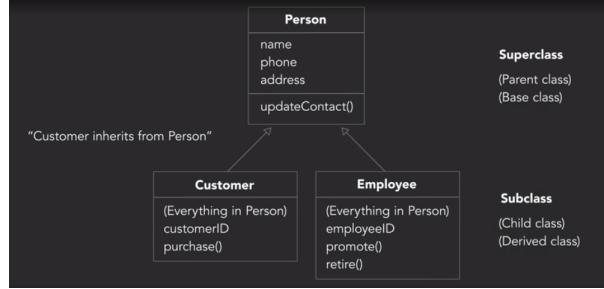


Abbildung 178:

Eine Klasse kann auch von mehreren **Superklassen** eine Vererbung erhalten:

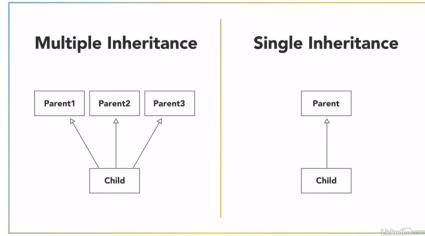


Abbildung 179:

---

```

1 Class A(...); //Base class
2 Class B : public A(...); //B derived from A
3 Class C : public B(...); //C derived from B

```

---

**3.1.4.4 Polymorphism** Hier wird zwischen **Statischer** und **Dynamischer Polymorphism** unterschieden. Wie diese direkt unterschieden werden, hängt von vom Complierprozess ab. Zu jeder der beiden Kategorien lässt sich aber besondere Methode definieren:

- Method Overload: Mit Hilfe dieses Prinzip können verschiedene Methoden mit der gleichen Signatur oder Namen definiert werden. Es muss aber möglich sein, Anhand der Inputanzahl oder Typen die Methoden zu differenzieren.

---

```

1 void sum (int a , int b);
2 void sum (int a , int b, int c);
3 void sum (float a, double b);

```

---

- Method Overriding: Soll eine Klasse abgeleitet von einer Superklasse werden, so kann es sein, das nicht alle Methoden genau so verwendet werden sollten. Damit aber nicht die ganze Klasse neu aufgesetzt werden muss, schreibt man nur die Methode um, die anzupassen ist.

---

```

1 class X{
2 public int sum(){
3 // some code
4 }
5 }
6
7 class Y extends X{
8 public int sum(){
9 //overridden method
10 //signature is same
11 }
12 }

```

---

## 3.2 Class Diagramm

### 3.2.1 Creating Class Diagramm

In dem UML Diagramm definieren wir die Variablen/ Attribute zuerst. Hinter dem Doppelpunkt definieren wir den Typ der Variablen und mit dem Gleichheitszeichen definieren wir den *Defaultvalue*.

| ClassName                      |
|--------------------------------|
| callSign: String = "Excellent" |
| shieldActive: Boolean          |
| position: Coordinate           |
| Methoden                       |

Für die Methoden werden meistens Verben in ersten Teil der Bezeichnung für eine Methode verwendet.

| Spaceship                             |
|---------------------------------------|
| callSign: String = "Excellent"        |
| shieldActive: Boolean                 |
| position: Coordinate                  |
| getPosition(): Coordinate             |
| setPosition(integer): Coordinate      |
| move(): Coodinate                     |
| lowerShield(integer, string): Boolean |

In der Klammer werden die Inputvariablen definiert, die die Methode benötigt. Weiter wird auch hier der Wiedergabetyp, hinter dem Doppelpunkt, definiert.

Um **Private** Klassifizierung in der Klasse zu definieren, werden **Plus** und **Minus** genutzt. Ein Plus signalisiert, dass das Attribut oder Methode **public** ist und ein Minus, dass das Attribut oder die Methode **private** ist.

| Spaceship                               |
|-----------------------------------------|
| + callSign: String = "Excellent"        |
| - shieldActive: Boolean                 |
| - position: Coordinate                  |
| + getPosition(): Coordinate             |
| - setPosition(integer): Coordinate      |
| + move(): Coodinate                     |
| + lowerShield(integer, string): Boolean |

### 3.2.2 Konstruktor

Ein **Konstruktor** ist eine Methode, welche beim erzeugen der Klasse aufgerufen wird. Wollen wir ein Objekt erzeugen (instanzieren), verwenden wir

```
1 class X{
2 public int sum(){
3 // some code
4 }
5 }
6
7 X myX = new X();
```

Damit nicht die Defaultvalues für die Attribute verwendet werden, sondern wie für *callSign = "Excellent"*, wird der Konstruktor verwendet.

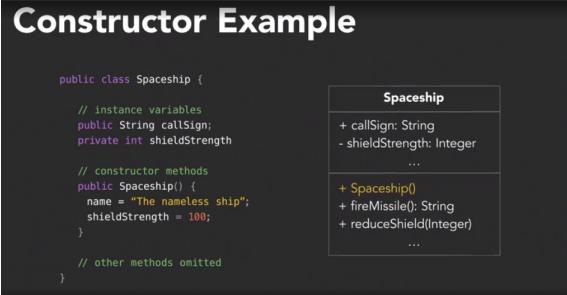


Abbildung 180:

Hinweis: Es soll

---

```

1 public Spaceship() {
2 callSign = "The nameless ship";
3 NOT name
4 shieldStrength = 100;
5 }

```

---

heißen. Es können durch die Fähigkeit von *Overloading* mehrere Konstruktoren erstellt werden. Diese müssen sich jedoch in ihren Inputvariablen unterscheiden lassen:

---

```

1 public class Spaceship {
2 // instance variables
3 public String callSign;
4 private int shieldStrength;
5
6 // konstruktor
7 public Spaceship() {
8 callSign = "The nameless ship";
9 shieldStrength = 100;
10 }
11
12 // overloading konstruktor
13 public Spaceship(String name) {
14 callSign = name
15 shieldStrength = 200;
16
17 // other methods ommitted
18 }

```

---

Instanziert man das Objekt, so folgt, dass man verschiedene Konstruktoren aufrufen kann.

---

```

1 Spaceship yourShip = new Spaceship(); // Erzeugt das "Default" Objekt
2 Spaceship myShip = new Spaceship("Enterprise"); // Ruft den Overload Konstruktor auf.

```

---

### 3.2.3 Destruktor

Solle in Objekt gelöscht werden, so verwendet man den Destruktor.

### 3.2.4 Static / Instanz Variable/ Methode

Es gibt zwei Arten von Variable, die **Instanz Varialben** und die **Statischen Variablen**, in einer Klasse.

**Instanz Variable** Dies Form der Variable wir für jedes Objekt in einer Klasse angelegt. Erzeugen wir 3 Objekte in einer Klasse, so erzeugen wir auch 3 *Instanzen Variablen*, wenn diese in der Klasse angelegt wurden.

---

```

1 public class Spaceship {
2 // Instanz Variable
3 public String callSign;
4 private int shieldStrength;
5 // [...]
6
7 }

```

---

```

8 // Create two objects
9 Spaceship ship1 = new Spaceship();
10 Spaceship ship2 = new Spaceship();

```

---

Soll die zugängliche Variable *callSign* geändert werden, so wird das Objekt und dann die *Instanz Variable* aufgerufen:

---

```
1 ship1.callSign = "Excalibur";
```

---

**Static Variable** Die Form der Variablen ist dann nötig, wenn Variablen für mehrere Objekte definiert werden sollen, diese aber nicht für jedes einzelne Objekt geändert werden können. Dies sind mit Globalen Variablen vergleichbar, nur dass sie nicht außerhalb der Klasse existieren. Ändert man eine *statische Variable*, so ändert man den Wert für jedes Objekt der zugehörigen Klasse.

---

```

1 public class Spaceship {
2 // Static variables
3 public static float toughness;
4 [...]

```

---

Soll der Wert der zugänglichen Variable, *toughness* geändert werden, wir die Klassen NICHT ein bestimmtes Objekt aufgerufen.

---

```
1 Spaceship.toughness = 0.85; // In jeder Klasse wird der Wert angepasst.
```

---

**Static Method** Aufbauend auf dem Prinzip von *Statischen Variablen* können **Static Methods** nur von der Klasse aufgerufen werden. Das Grundprinzip ist, dass auch Statische Variablen nur über Statische Methoden aufgerufen und verändert werden. Ein Konstruktor kann so, zwar eine Initialisierung der Instanzvariablen erzeugen, aber für weitere Änderungen ist es ratsam, die Methoden zu nutzen, um Variablen zu ändern.

---

```

1 public class Spaceship {
2 // Static variables
3 public static float toughness;
4 [...]
5 public static increaseDifficulty(float t){
6 toughness += t
7 }

```

---

Eine *Static Method* kann nur eine *Static Variable* verändern und vom *Objekt* aufgerufen werden.

---

```

1 Spaceship.increaseDifficulty(0.2); // In jeder Klasse wird der Wert
2 // wenn die Initialisierung von 0.8 erfolgreich war.

```

---

**Static Notierung** In dem UML Diagramm werden statische Objekte mit einem Unterschrich ausgewiesen.

| Spaceship                               |
|-----------------------------------------|
| + callSign: String = "Excellent"        |
| - shieldActive: Boolean                 |
| - position: Coordinate                  |
| + toughness: float                      |
| + getPosition(): Coordinate             |
| - setPosition(integer): Coordinate      |
| + move(): Coordinate                    |
| + lowerShield(integer, string): Boolean |
| + increaseDifficulty(float)             |

### 3.3 Inheritance and Composition

Es ist nicht ratsam, nach *Vererbung* zu suchen und verschiedenen Ebenen von Beziehungen zu erstellen. **Inheritance** "will reveal is self". **Overriding** ist ein essentieller Teil der Vererbung. Ebenso ist es möglich weitere

Methoden einer Klasse zuzufügen.

In C++ wird der Doppelpunkt verwendet, um eine Klasse einer Superklasse zuzuweisen:

```
1 public class CargoShuttle : Spaceship {
2 \\\ CargoShuttle inherent all from Spaceship
3 }
```

Beim Beschreiben der hergeleiteten Klasse, wird von Sprache zu Sprache unterschied, welche Syntax zu verwenden ist. In Java wird der Prefix **super**, in C# **base**, verwendet. **Hinweis:** In einer Klasse, kann auch eine Methode beschrieben werden, die ein Objekt der Klasse intanziert.

Im folgenden Beispiel

```
1 class left {
2 public:
3 void foo();
4 };
5
6 class right {
7 public:
8 void foo();
9 };
10
11 class bottom : public left, public right {
12 public:
13 void foo()
14 {
15 //base::foo(); // ambiguous
16 left::foo();
17 right::foo();
18
19 // and when foo() is not called for 'this':
20 bottom b;
21 b.left::foo(); // calls b.foo() from 'left'
22 b.right::foo(); // call b.foo() from 'right'
23 }
24 };
```

wird der Synatz `::` benötigt, um auf die Superklasse zu referieren. In C++ kann auch auf zwei Superklassen referiert werden. Andere Klassen verweisen, in der Regel nur auf eine Superklasse.

### 3.3.1 Abstract class/ Methode

Eine **Abstrakte Klasse** ist eine Form der Klasse, welche eine Initialisierung zu lässt. Der Zweck dieser Form ist, dass die Klasse als Super Klasse dient. Dabei kann eine Abstrakte Klasse auch selber eine Subklasse sein, jedoch wird von dieser Ebenen mindestens eine weitere Ebene erschlossen.

```
1 public abstract class GraphicObject {
2 // declare variables
3 // declare nonabstract methods
4 abstract void draw();
5 }
```

Eine **Abstrakte Methode** wird ohne Implementation geschrieben. D. h., hinter (...) wird nur ; gesetzt. In der Subklasse wird diese Methode dann implementiert: *Override*. Wird die Methode nicht implementiert, so muss die Sub Klasse auch abstrakt sein.

Die Methoden die zu überschreiben sind, werden dann, im Normalfall, als abstrakt dekliniert. Selbst, wenn diese nicht notwendig ist, hilft es, die Intention in dem programmierten Text besser zu vermitteln.

```
1 abstract class GraphicObject {
2 int x, y;
3 ...
4 void moveTo(int newX, int newY) {
5 ...
6 }
7 }
```

```

7 abstract void draw();
8 abstract void resize();
9
10
11 }
12 class Circle extends GraphicObject {
13 void draw() {
14 // Overriding
15 }
16 void resize() {
17 // Overriding
18 }
19 }
20 class Rectangle extends GraphicObject {
21 void draw() {
22 // Overriding
23 }
24 void resize() {
25 // Overriding
26 }
27

```

Schlussfolgerung: Eine **abstract class** ist nur zur Vererbung angelegt. Objekte können auch nicht über diese Klasse instantiiert werden.

### 3.3.2 Interfaces

Es gibt einige Unterschied zwischen *Abstract class* und **Interface**. Auf dem ersten Blick wirken beide jedoch sehr ähnlich. Das Interface wirkt auch wie extreme Variante der Abstrakten Klasse. Die Zielsetzung mit der man die Klassen erstellt, sind aber andere.

**Abstrakte Klasse** wird verwendet, wenn Klassen verschiedenen Beziehungen miteinander haben und eine Hierarchische Struktur besitzen. Bei C++ können Klassen von mehreren Superklassen die Eigenschaften erhalten, bei den meisten anderen Sprachen jedoch nicht - Hierarchische Struktur.

**Interface** wird verwendet, wenn Fähigkeiten übertragen werden sollen. Die Fähigkeit *move()* und *draw()* lässt sich besser jeweiligen Interface Klassen zuordnen, als eine hierachische Abstrakte Klasse zu erzeugen.

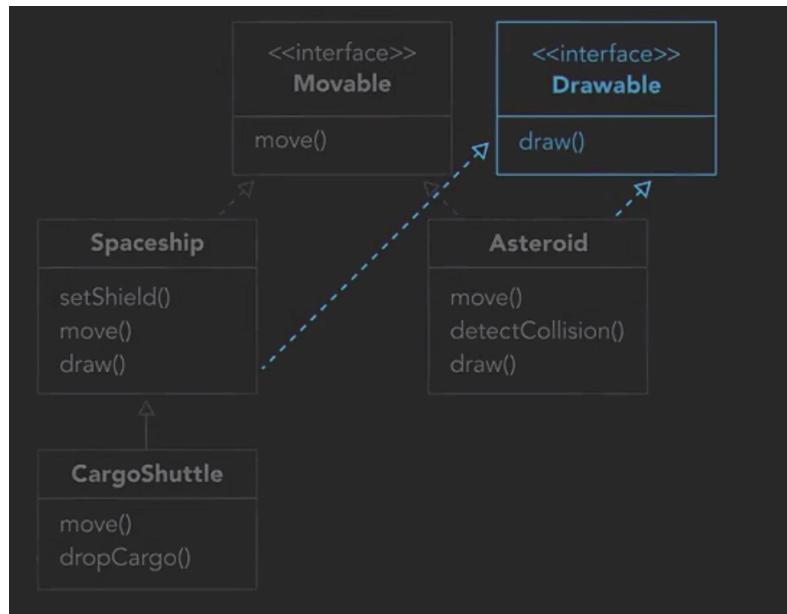


Abbildung 181:

In dem UML Diagramm werden Interface mit gepunkteten Linien gezeichnet.

Eigenschaften

- Alle Variablen und Methoden sind **public, static**. ≠ In einer Abstrakten Klasse kann eine Methode abstrakt sein.

```

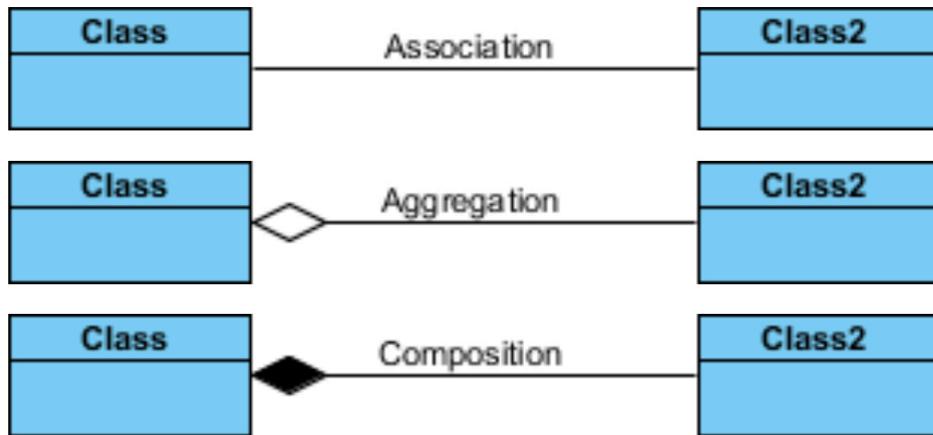
1 interface Moveable {
2 // methode signate
3 void move(int x, int y);
4
5 class X implements Moveable {
6 // Implementation for interface method
7 public void move(int x, int y){
8 // Implementation code
9 }
10 // Additional methods
11 }

```

- Alle Methoden besitzen keine Implementation.
- Keine Methode muss mit **public** oder **private** deklariert werden.

### 3.3.3 Aggregation / Composition

Diese beiden Relationen beziehen sich nicht so sehr auf eine besondere Syntax, sondern auf die Gestaltung des Codes.



The figure below shows a generalization.



Abbildung 182:

- Die Relation **Aggregation** wird so verstanden, dass ein Subobjekt, in einer Gruppenabhängigkeit einbezogen wird. Löst sich diese jedoch auf, bleibt das erzeugt Objekt erhalten.
- Die Realation **Composition** wird so verstanden, dass ein Subobjekt, in einer Gruppenabhängigkeit einbezogen wird. Löst sich diese jedoch auf, so hört das erzeugt Objekt auf zu existieren.

Zum Beispiel: Ein Spaceship kann teil einer Fleet sein. Verschwindet die Fleet, so verschwindet das Spaceship nicht zwangsläufigerweise.

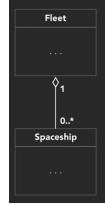


Abbildung 183:

Zum Beispiel: Ein Auto besteht aus einem Motor und Reifen. Wenn das Auto nicht mehr ist, wird der Motor nicht mehr benötigt.

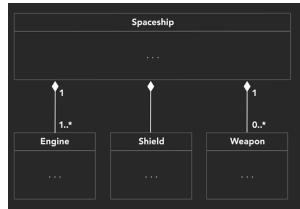


Abbildung 184:

**Hinweis** Mit Hilfe der UML Beziehungsanalyse soll die Programmiererin erkennen, welche Objekte mit einem *Destruktor* gelöscht werden müssen und welche nicht.

### 3.3.4 Zusammenfassung Relationen



Abbildung 185:

## 3.4 Software Development

| Object-Oriented Languages |             |                 |         |                |                  |  |
|---------------------------|-------------|-----------------|---------|----------------|------------------|--|
| Language                  | Inheritance | Call to Super   | Typing  | Interfaces     | Abstract Classes |  |
| Java                      | single      | super           | static  | Yes            | Yes              |  |
| C#                        | single      | base            | static  | Yes            | Yes              |  |
| Python                    | multiple    | super           | dynamic | Abstract Class | Yes              |  |
| Swift                     | single      | super           | static  | Protocols      | No               |  |
| C++                       | multiple    | name of class:: | static  | Abstract Class | Yes              |  |
| Ruby                      | mixins      | super           | dynamic | n/a            | n/a              |  |
| JavaScript                | prototypes  | n/a             | dynamic | n/a            | n/a              |  |

Abbildung 186:

Die verschiedenen Sprachen können unterschiedlich mit den Prinzipien, die wir besprochen haben, umgehen.

- Inheritance: Kann eine Klasse vererbt werden. Bei C++ können auch mehrere Klassen einer Klasse vererbt werden.

- Call to super: Wie lautet die Syntax, um eine Vererbung anzuzeigen.
- Typing: ? Das Thema ist noch unklar. Stichphrase: Wie wird kompiliert? Es gibt hierzu Vor- und Nachteil. Zum Beispiel: Bei static languages müssen alle Variablen vom Typ definiert werden, da sie vor dem Ausführen vorliegen. Bei dynamischen Sprachen wird der Typ bei Ausführung erkannt. Dies kann jedoch zu Problemen führen, falls die Erkennung nicht richtig erfolgt. Das Schreiben des Codes ist aber einfacher.

**Teil V**

**Python**

# 1 Python Introduction

## 1.1 Running Python from VS Code

### 1.1.1 Befehle

**Python –version** Zeigt die aktuelle Version an

**py** In PowerShell/Terminal/cmd wird Python über **py** gestartet.

### 1.1.2 Launch Json File

Unter Run-Debug/Settings werden die globalen Debug Lauch Konfigurationen eingelesen.

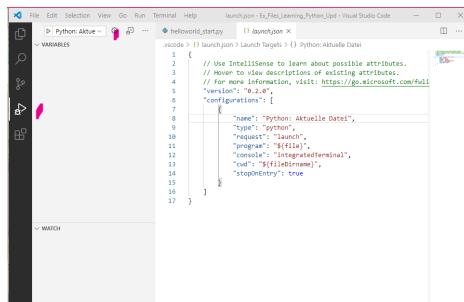


Abbildung 187:

- **cwd** gibt an wo CurrentWorkingDirectory ist.
- **stopOnEntry** Stop den Vorgang zum Anfang der Datei

### 1.1.3 Internal Console instead of Terminal

Eine verschlankte Anzeige ist, wenn die Console nicht über das Terminal gestartet wird. Eine Einschränkung gibt es, Python kann keine Eingaben erhalten.



- **console** integratedTerminal - Vollumfang
- **console** internalConsole - Verschlankte Darstellung in Debug Console, Kein Input

## 1.2 Basis Concepts

### 1.2.1 Variables and Expressions

- Keine spezifische Typ Zuweisung
- Die **print()** Funktion muss gleiche Typen erhalten

```
1 # Declare a variable and initialize it
2 f = 0
3 print(f)
4
5
6 # re-declaring the variable works
7 f = "abc"
```

```

8 print(f)
9
10
11 # ERROR: variables of different types cannot be combined
12 'print("string" + 2)
13 print("string"+str(2))

```

---

- Innerhalb einer Funktion wird eine lokale Variable abgelegt.

```

1 # Global vs. local variables in functions
2 def someFunction():
3 f = "def"
4 print(f)
5
6 someFunction()
7 print(f)

```

---

Output:

```
def
f
```

- Deklariert man die als *global* wird die Variable außerhalb der Funktion ebenfalls neu definiert.

```

1 \begin{lstlisting}[style=python]
2 # Global vs. local variables in functions
3 def someFunction():
4 global f
5 f = "def"
6 print(f)
7
8 someFunction()
9 print(f)

```

---

Output:

```
def
def
```

- Eine Inputvariable mit einem \* fungiert als Array.

### 1.2.2 Function

Funktion werden anders als in C++ nicht mit einem Funktionsruf versehen. Ab wann die Rupf in pyhton anfängt wird mit : und einer Einrückung versehen.

Die Print-Funktion ruft eine als Argument übergebenen Funktion auf und gibt den Wert der Funktion wieder. Damit ist der Return Wert gemeint, sondern der Allgemeine Wert einer Funtion - None.

Wenn die Funktion einen Return Wert besitzt, gibt Print() diesen wieder. Eine separate Ausgabe wird nicht durchgeführt.

```

1 # function that returns a value
2 def cube(x):
3 return x*x*x
4
5 print(cube(3))
6 # 27

```

---

In Python kann die Adressierung der Inputvariablen in einer Funktion getauscht werden.

---

```
1 def func3(a,b):
2 return a/b
3
4 print(func3(3,2)) # 1.5
5 print(func3(b=2, a=3)) 1.5
```

---

### 1.2.3 Conditionals

Der if-else Block unterscheidet sich zu C++ von der Syntax. Die Teilbereich des `if` Blocks wird mit : beendet. Ebenso gibt dies für `else`. Weil Python eine Leerzeichen sensible Sprache ist, muss auf die Einrückung geachtet werden.

---

```
1 def main():
2 x, y = 1000, 100
3
4 # conditional flow uses if, elif, else
5 if x < y:
6 str = "x is less than y"
7 elif x==y:
8 str = "x is equal to y"
9 else:
10 str = "x is greater than y"
11 print(str)
12
13 # conditional statements let you use "a if C else b"
14
15 if __name__ == "__main__":
16 main()
```

---

### 1.2.4 Loop

- While Loop

---

```
1 # define a while loop
2 while (x<5):
3 print(x)
4 if (x==3): break
5 x = x + 1
```

---

Nach der Bedingung wird in Python mit einem : beendet.

- For Loop

---

```
1 # define a for loop
2 for x in range(5,10):
3 print(x)
```

---

mit der Rang Funktion kann der Bereich definiert werden, welcher durchlaufen werden soll. Arrays können ohne Indexierung, wie in C++, durchlaufen werden.

---

```
1 # use a for loop over a collection
2 days = ["Mo", "Di", "Mi", "Do", "Fr"]
3 for d in days:
4 if d == "Mi": break
5 print (d)
```

---

oder

---

```
1 # use a for loop over a collection
2 days = ["Mo", "Di", "Mi", "Do", "Fr"]
3 for d, i in enumerate(days):
4 print (i, d)
```

---

## 1.3 OOP - Class

### 1.3.1 Structure of a Class

Der einfachste Fall ist eine Klasse, die leer ist. Der Befehl nach der Definierung der Klasse ist `pass`.

Ein Beispiel für eine Klasse mit zwei Methoden und einem belegten Attribute ist

---

```
1 class myClass():
2 # Attribute
3 x = "Python"
4
5 # Methode 1
6 def Methode1(self):
7 print("Output Methode 1 myClass")
8
9 # Methode 2
10 def Methode1(self, x):
11 print("Output Methode 1 myClass" + x)
12
13 object = myClass
14 print(object.x) # Python
```

---

### 1.3.2 Inheritance and Override

Die Vererbung einer Klasse funktioniert von der Syntax ähnlich wie in C++ (CPP). Die Superklasse oder Parent-class wird als Objekt in die Klasse übergeben.

---

```
1 class ChildClassName(SuperClassName):
```

---

Die Methoden und Attribute der Superklasse werden vererbt. Mit Hilfe von Methode Override können die vererbten Methoden geändert werden.

Im Folgenden wird die Superklasse `myClass()` an die Klasse `anotherClass()` übergeben.

---

```
1 class myClass():
2 def method1(self):
3 print("myClass method1")
4
5 def method2(self, someString):
6 print("myClass method2: " + someString)
7
8 def method3(self):
9 print("myClass method 3")
10
11
12 class anotherClass(myClass):
13 def method1(self):
14 print("anotherClass method1")
15
16 def method2(self):
17 print("anotherClass method2")
```

---

Hierbei werden *method1* und *method2* überschrieben. Die *methode3* wird vererbt. Die Instanzierung beider Klassen erfolgt auf dem üblichen Weg

---

```
1 instanzSK = myClass()
2 instanzCK = anotherClass()
```

---

Die Methoden der Superklasse können über das erzeugt Objekt **InstanzSK** aufgerufen werden.

---

```
1 instanzSK.methode1() # myClass method1
2 instanzSK.methode2("Test") # myClass method2: Test
3 instanzSK.methode3() # myClass method3
```

---

Die Instanz der Kindklasse hat die Möglichkeit die zwei überschriebenen Methode und die vererbte Methode aufzurufen.

---

```
1 instanzCK.methode1() # anotherClass method1
2 instanzCK.methode2() # anotherClass method2
3 instanzCK.methode3() # myClass method3
```

---

Die Möglichkeit Methode Overload bei vererbten Methoden anzuwenden ist nicht möglich, siehe folgendes Beispiel:

---

```
1 instanzCK.methode2("Test") # Error
```

---

### 1.3.3 Methode Overload

Das Prinzip von Methode Overload wird in Python nicht wie in CPP mit zwei gleichnamigen Funktionen mit unterschiedlicher Anzahl von Parametern umgesetzt. In Python wird Methode Overload in einer Funktion vereinigt. Die Inputvariablen werden mit einem Default Wert besetzt. Wird dieser nicht übergeben, kann zwischen verschiedenen Bereichen in der Funktion unterschieden werden.

Würde die Logik von CPP gelten, würde eine Ausführung wie folgt aussehen:

---

```
1 class myClass():
2 def method1(self):
3 print("myClass method1")
4
5 def method1(self, someString): # Zwei Parameter
6 print("myClass method1: " + someString)
7
8 c = myClass()
9 c.method1() # myClass methode1
10 c.method1("Test") # myClass methode1: Test
```

---

Mit Python wird dies in der Funktion mit einem Default Wert für *someString* erreicht.

---

```
1 class myClass():
2 def method1(self, someString = None):
3 if someString is None:
4 print("myClass method1")
5 else:
6 print("myClass method1: " + someString)
7
8 c = myClass()
9 c.method1() # myClass methode1
10 c.method1("Test") # myClass methode1: Test
```

---

#### 1.3.4 The self Argument

Klassen Methoden benötigen einen Parameter, welcher beim Aufrufen der Instanz nicht übermittelt werden muss. Dieser Parameter heißt *self*. Er ist vergleichbar mit dem *Me* Objekt im VBA. In CPP wird dies nicht benötigt. Die Instanz der Klasse übergibt den Wert von sich selbst an den *self* Parameter.

Wird eine Klasse mit einer Methode erstellt:

```
1 class myClass():
2 def method1(self):
3 print("myClass method1")
```

so kann diese über aufgerufen werden, indem eine Instanz der Klasse, *c*, erstellt wird und die Methode direkt aufgerufen wird.

```
1 c = myClass()
2 c.method1() #
```

Der Wert des Objektes *c* wird hierbei übergeben, ohne explizit genannt zu werden. Dies liegt an der Syntax. Voll ausgeschrieben wird die Methode *method1()* über die Klasse selbst aufgerufen.

```
1 myClass.method1(c) # myClass method1
```

#### 1.3.5 Attributes

Die Attribute einer Klasse können entweder einer Instanz oder der Klasse selbst zugeordnet werden. Eine Variable deklariert wird in Python nicht in der Form getan, wie in CPP.

##### 1.3.5.1 Ohne Constructur init ist falsch

Um ein Attribut für jede Instanz ansteuerbar zu machen, muss das *self* verwendet werden.

```
1 class myClass():
2 self.attribute # ERROR
3
4 c = myClass()
5 c.attribute = 5
6 print(c.attribute)
```

##### 1.3.5.2 Mit Constructor init ist richtig

Ein Attribut kann nur über die *init* Methode deklariert werden.

```
1 class myClass():
2 def __init__(self,attribute)
3 self.attribute = attribute
4
5 #c = myClass() # Error: Missing Input
6 c = myClass(5)
7 #c.attribute = 5 # Nicht mehr benötigt
8 print(c.attribute) # 5
9 c.attribute = 6
10 print(c.attribute) # 6
```

##### 1.3.5.3 Attribut einer Klasse direkt zuordnen.

Ein Attribut kann auch einer Klasse direkt zugeordnet werden. Dabei hat jede Instanz die Möglichkeit auf das Attribut zuzugreifen und Änderungen vorzunehmen.

---

```

1 class myClass():
2 attribute1 = None
3 def __init__(self,attribute2)
4 self.attribute1 = attribute
5
6 c = myClass(5)
7 cc = myClass(6)
8 myClass.attribute1 = 3
9
10 # Check Attribute1
11 print(c.attribute1) # 3
12 print(cc.attribute1) # 3
13 print(myClass.attribute1) # 3
14
15 # Check Attribut2
16 print(c.attribute2) # 5
17 print(cc.attribute2) # 6

```

---

Es ist Achtsamkeit geboten, wenn eine Klassen Attribute von einer Instanz geändert wird, wird dieser Instanz das Attribut als eigenständige Variable zugeordnet.

---

```

1 class myClass():
2 attribute1 = None
3 def __init__(self,attribute2)
4 self.attribute1 = attribute
5
6 c = myClass(5)
7 cc = myClass(6)
8
9
10 # Change Class Attribute over class itself
11 myClass.attribute1 = 3
12 print(c.attribute1) # 3
13 print(cc.attribute1) # 3
14 print(myClass.attribute1) # 3
15
16 # Class attribute changes to instanz attribute
17 c.attribute1 = 9
18 print(myClass.attribute1) # 3
19 print(c.attribute1) # 9
20 print(cc.attribute1) # 3
21
22 # New instanz varialbe stays the same
23 myClass.attribute1 = 10
24 print(myClass.attribute1) # 10
25 print(c.attribute1) # 9
26 print(cc.attribute1) # 10

```

---

### 1.3.6 Call methode from another Class

Jede Methode einer Klasse ist ebenfalls von Instanzen andere Klassen nutzbar. Die zu übergebenen Parameter müssen von der Fremde-Instanz auch bereit gestellt werden. Es müssen nicht alles Eigenschaften Fremden-Instanz gleich sein, nur die, die für die aufgerufenen Methode benötigt wird.

Der Parameter *self* bezieht sich auf die Fremde-Instanz.

```

1 class myClass():
2 def __init__(self, a, b):
3 self.a = a
4 self.b = b
5 def method1(self):
6 print("Method1 myClass: ", self.a, self.b)
7 self.a = self.a + 1
8 print("Method1 myClass: ", self.a, self.b)
9
10 class anotherClass():
11 def __init__(self, a, b):
12 self.a = a
13 self.b = b
14 def method1(self):
15 A = myClass(2,2)
16 print("Method1 anotherClass")
17 myClass.method1(A) # equal to self.method1()
18
19 class tanotherClass():
20 def __init__(self, a, b):
21 self.a = a
22 self.b = b
23
24 def method1(self):
25 print("Method1 tanotherClass")
26 myClass.method1(self)
27
28 c = myClass(1,1)
29 cc = anotherClass(4,4)
30 ccc= tanotherClass(8,8)
31
32 c.method1() # Method1 myClass: " 1 1
33 # Method1 myClass: " 2 1
34 cc.method1() # Method1 anotherClass
35 # Method1 myClass: " 2 2
36 # Method1 myClass: " 3 3
37
38 cc.method1() # Method1 tanotherClass
39 # Method1 myClass: " 8 8
40 # Method1 myClass: " 9 8

```

---

Wird die Klasse *myClass* als Superklasse übergeben, benötigt es *init* nicht mehr.

---

```

1 class anotherClass(myClass): # Vererbung von __int__
2 def method1(self):
3 A = myClass(2,2)
4 print("Method1 anotherClass")
5 myClass.method1(A) # equal to self.method1()

```

---

### 1.3.7 Public, Private and Protected

**Protected** Eine Variable kann als privat deklariert werden. Es gibt aber von Python kein Einschränkungen. Syntax `_`. Die Konvention ist, dass der Zugang zu den geschützten Attributen (Variablen) nur über get und set Funktionen erfolgt.

---

```

1 class myClass():
2 def __init__(self, a, b):

```

```
3 self._a = a
4 self.b = b
5
6 c = myClass(5,6)
7 print(c._a, c.b) # 5 6
```

---

**Private** Eine Variable kann als privat deklariert werden. Eine direkte Aufrufen

```
1 c = myClass(5,6)
2 print(c._b) # Error, wenn self._b = b
```

---

Über eine andere Syntax ist dies aber möglich.

```
1 class myClass():
2 def __init__(self, a, b):
3 self._a = a
4 self.__b = b
5
6 c = myClass(5,6)
7 print(c._a, c._myClass__b)
```

---

### 1.3.8 From ... import

Der erste Teil *From ...* verweist auf das komplette Modul. Der zweite Teil referiert auf eine spezifische Klasse.

## 1.4 Modul Formating Date and Time

- Das Packet *datetime* besteht aus mehreren Klassen. Eine davon lautet ebenso *datetime*.
- Information zu dem Module findet sich unter [Python Library](#)

### 1.4.1 Retriev information form datetime module with date, time and datetime class

```
1 from datetime import date
2 from datetime import time
3 from datetime import datetime
4
5 def main():
6 ## DATE OBJECTS
7 # Get today's date from the simple today() method from the date class
8 today = date.today()
9 todaytime = datetime.today()
10
11 # print out the date's individual components
12 print("Day:", today.day) # equal to date.today().day
13 print("Month:", today.month)
14 print("Year:", today.year)
15
16 print("Datetime: ", datetime.today())
17 print(datetime.today().day) # equal to todaytime.day
18 print(datetime.today().month)
19 print(datetime.today().year)
20 print(datetime.today().hour)
21
22
23 # retrieve today's weekday (0=Monday, 6=Sunday)
```

---

```

24 print("Weekday:", today.weekday())
25 print("Weekday:", date.today().weekday())
26
27 weekdays = ["Mo", "Di", "Mi", "Do", "Fr", "Sa", "So"]
28 print("Weekday full name: ", weekdays[date.today().weekday()])
29
30 ## DATETIME OBJECTS
31 # Get today's date from the datetime class
32 print("Date form Datetime class:", datetime.date(datetime.today())) #
33 .date() benötigt eine Input
34
35 # Get the current time
36 print("Current Time:", datetime.time(datetime.today()))

```

---

#### 1.4.2 Formating

Die Funktion `str()` wandelt eine Input Objekt in einen String um.

```

1 str(5) # converts a integer to a string
2 int("5") # converts a string to an integer

```

---

Die Funktion `strftime()` bekommt vom vererbten Objekt den datetime oder date Input. Die Methode funktioniert, in der Hinsicht, dass die Methode formattierte Strings ausgeben kann. Dabei können auch mehrere Formatierungscodes angegeben werden:

- \* Alle Codierungen werden mit einem führenden Prozentzeichen angegeben.
- m - Monat, b - Monatsname Kurz, B - Monatsname Lang
- d - Tag, w - weekdat
- y - Jahr
- H - Stunde, I - Stunden (12-Stunden Uhr) mit einer führenden Null, -I - Stunden (12 Uhr) als Dezimazahl, p - Local AM or PM
- M - Minute
- S - Sekunde

```

1 #
2 # Example file for formatting time and date output
3 #
4
5 from datetime import datetime
6
7 def main():
8 # Times and dates can be formatted using a set of predefined string
9 # control codes
10 print("Now", datetime.now())
11 print("Now", datetime.now().today()) # returns the same as before and
12 after this linie
13 print("Today", datetime.today()) # returns the same output as now()
14
15 ##### Date Formatting #####
16
17 ### strftime() datetime, date to string
18 # %Y, %m, %d, %y/%Y - Year, %a/%A - weekday, %b/%B - month, %d - day
19 # of month are Format Codes.

```

```

18 print(datetime.today().strftime("%Y")) # 2020
19 print(datetime.now().strftime("%H:%M:%S")) # 20:49:31
20
21
22 # %c - locale's date and time, %x - locale's date, %X - locale's time
23 print(datetime.today().strftime("%c")) # Full Formation
24 print(datetime.today().strftime("%x"))
25 print(datetime.today().strftime("%X"))
26 if __name__ == "__main__":
27 main()

```

---

Der umgedrehte Fall folgt aus `strptime()`. Die Methode nimmt einen String und wandelt in ein `datetime` Objekt um.

#### 1.4.3 Timedelta

Mit `timedelta()` kann eine Zu- oder Abbuchung auf ein `date` oder `datetime` Objekt erfolgen.

```

1 from datetime import datetime
2 from datetime import date
3 from datetime import time
4 from datetime import timedelta
5
6 print("One year form now:", date.today() + timedelta(days=365)) #
 08.12.2021
7 print("One year form now:", datetime.now() + timedelta(days=365)) #
 08.12.2021 10:11:223

```

---

Die Funktion `retrieve()` ermöglicht es bestimmte Bestandteile einer `datetime()` und oder `date()` Variable zu ändern.

```

1 ##### How many days until April Fools' Day?
2 xmas = date(2020,12,24)
3
4 if date.today() == xmas:
5 print("Today ist Christmas.")
6 xmas = xmas + timedelta(weeks=52)
7 print("Nächstes Weihnachten ist ", xmas)
8 else:
9 print("Es sind noch", (date.today() - xmas).days, "Tage")
10
11 # use date comparison to see if April Fool's has already gone for
12 # this year
13 # if it has, use the replace() function to get the date for next year
14 print("Alternativ kann das Datum mit replace() ermöglicht werden.",
 xmas.replace(year=date.today().year +2))

```

---

Eine Subtraktion von zwei Daten wird als ein Objekt umgewandelt, welches die Tage zwischen den zwei Daten zählt.

```

1 d = date.today()
2 b = date(2019,6,15)
3 print(d-b) # Returns 549 days, 0:00:00
4 print(d+b) # Returns Error

```

---

Die Operation `-` ist für `datetime.date` Objekte definiert. Die Addition nicht. Die Operation Subtraktion wandelt  $d - b$  in ein Objekt `datetime.timedelta` um. Es handelt sich hierbei nicht um ein Integer. Um einen Vergleich mit zwei Zahlen zu ermöglichen, muss ein Tag, Jahr oder Monat aus dem `Timedelta` Objekt extrahiert werden.

---

```
1 d = date.today()
2 b = date(d.year,6,30)
3 diff = (d-b).days # OHNE Days kommt es zu einen Instanzen-Fehler
4 if diff > 0:
5 print("Birthday in %d days" % diff)
6 else:
7 print("Birthday in %d days" % diff)
```

---

#### 1.4.4 Calendar

Unter **Library Calendar** wird der Aufbau des Moduls *calendar*.

Die Aufbau sieht wie folgt aus:

- Über das Modul selbst können einfache Kalender erstellt werden.

---

```
1 c = calendar
```

---

- Das Objekt *c* besitzt Methoden und Attribute. Die Klasse dient zur Vorbereitung von Kalender Informationen. Eine Formatierung ist nicht über die Klasse vorgesehen.

---

```
1 import calendar
2 c = calendar.calendar
```

---

- Detailiere Kalender können über

---

```
1 import calendar
2 c = calendar.TextCalendar(firstweekday=) # oder
3 c2 = calendar.HTMLCalendar(firstweekday=)
```

---

erstellt werden.

##### 1.4.4.1 Beispiele

- `calendar.weekday(year, month, day)` Returns the day of the week.
- `calendar.weekheader(n)` Returns header abbreviated
- `calendar.monthcalendar(year,month)` Returns a matrix representing a tupels of days. Example `.monatcalendar(2020,2)`

$$\rightarrow [[0, 0, 0, 0, 0, 1, 2], [3, 4, 5, 6, 7, 8, 9], [10, 11, 12, 13, 14, 15, 16], \quad (1.1)$$

$$[17, 18, 19, 20, 21, 22, 23], [24, 25, 26, 27, 28, 29, 0]] \quad (1.2)$$

- `calendar.MONDAY` gibt den Integer 0 wieder. Von Montag bis Sonntag ist dies möglich mit 0-6
- `calendar.week(theweek, w)` gibt einen cal Kalender mit Titel und Wochen Bezeichnung wieder.
- `calendar.TextCalendar(calendar.SUNDAY)` gibt ein HTML Objekt wieder, welches weiter formatiert und dargestellt wird.
- `calendar.TextCalendar(calendar.SUNDAY).formatyear(theyear=2020, w=4)` gibt einen komplette Kalender des Jahres 2020 wieder.
- `calendar.TextCalendar(calendar.SUNDAY).formatmonat(theyear=2020, themonth=12, w=4)` gibt einen kompletten Monatskalender wieder.
- `calendar.TextCalendar(calendar.SUNDAY).formatweek(theweek=[(0,6),...], w=4)` gibt eine Liste von Datumstagen wieder.

## 1.5 Working with files

Python bietet einen Sammelsurium von Build-In Funktionen an, siehe [Python Library](#).

### Built-in Functions

The Python interpreter has a number of functions and types built into it that are always available. They are listed here in alphabetical order.

| Built-in Functions |             |              |              |                |
|--------------------|-------------|--------------|--------------|----------------|
| abs()              | delattr()   | hash()       | memoryview() | set()          |
| all()              | dict()      | help()       | min()        | setattr()      |
| any()              | dir()       | hex()        | next()       | slice()        |
| ascii()            | divmod()    | id()         | object()     | sorted()       |
| bin()              | enumerate() | input()      | oct()        | staticmethod() |
| bool()             | eval()      | int()        | open()       | str()          |
| breakpoint()       | exec()      | isinstance() | ord()        | sum()          |
| bytearray()        | filter()    | issubclass() | pow()        | super()        |
| bytes()            | float()     | iter()       | print()      | tuple()        |
| callable()         | format()    | len()        | property()   | type()         |
| chr()              | frozenset() | list()       | range()      | vars()         |
| classmethod()      | getattr()   | locals()     | repr()       | zip()          |
| compile()          | globals()   | map()        | reversed()   | __import__()   |
| complex()          | hasattr()   | max()        | round()      |                |

#### 1.5.1 Open(File,Mode)

Eine Funktion *open()* gibt einen Zugang zu Dateien, um diese zu ändern und anzupassen. Die wichtigsten Parameter, welche die Datei aufnimmt, sind

- file - gibt an, wo die Datei zu finden ist und den Namen der Datei. Wir kein direkter Pfad angegeben, muss die Datei im gleichen Verzeichnis liegen, wo auch die .py liegt.
- modus - gibt an, in welchem Modus die Datei verwendet werden soll.
  - "r" - Read
  - "w" - Write and Truncation (Kürzen: Datei wird geleert.)
  - "a" - Append
  - "r+" - Read and Write
  - "x" - Creating (Error, if file exist)

Die Funktion erzeugt ein File-Objekt. Die einfachste Objekt-Methode ist *.close()*.

---

```
1 # open(file,modus)
2 fileObject = open("Testfile.txt","r")
3 fileObject.close() # Some Changes are only taking effect after the
 file is closed.
```

---

Dies ist wichtig, um Überschreibungen, Fehler und Ladungen richtig durchzuführen.

Das Objekt, je nach Modus, besitzt mehrere Attribute und Methoden.

---

```
1 # open(file,modus)
2 file = open("Testfile.txt","r")
3 print(file.name) # Testfile.txt
4 file.close()
```

---

Die Ausgabe der Datei funktioniert über *.read()*. Der Modus *r* muss aktiviert sein, damit eine Ausgabe möglich ist. Der Modus *w* erlaubt die Methode nicht.

---

```
1 # open(file,modus)
2 file = open("Testfile.txt","r")
3 print(file.read()) # Ausgabe des Inhaltes der Datei
4 file.close()
```

---

Nach dem die Methode durchgeführt wird, befindet sich der Pointer (Cursor) am Ende der Datei ein mehrmaliges Auslesen ist damit nicht möglich. Um genauer zu sein, die Ausgabe bleibt leer. Um den Pointer wieder auf den Start der Datei zu legen, wird `.seek()` benötigt.

```
1 # open(file,modus)
2 file = open("Testfile.txt","r")
3 print(file.read()) # Ausgabe des Inhaltes der Datei
4 file.seek(0) # setzt den Point auf den Anfang der Datei.
5 print(file.read()) # Die Ausgabe kann komplett durchgeführt werden.
6 file.close()
```

Um spezifische Zeilen anzusteuern oder mehrmaliges Auslesen zu ermöglichen, wird `.readline()` oder `.readlines()`. Die nächste Aufgabe befasst sich damit mehrer Zeilen in die Datei zu schreiben.

```
1 file = open("test.txt","r+")
2 for i in range(3,10):
3 file.write("Das ist brilliant hoch " + str(i) + \n \r) file.close()
```

Überträgt man die Daten der Auslesung, können die Daten in einer separaten Variable gespeichert werden.

```
1 file = open("test.txt","r")
2 if file.mode == "r": # Überprüft, ob die Datei richtig geöffnet wurde
3 fileArray = file.readlines()
4 for x in fileArray:
5 print(x)
6
7 file.close()
```

Jede Zeile wird als Eintrag in dem Array abgespeichert, und kann auch als solches abgerufen werden.

### 1.5.2 OS and OS.Path

Das Paket OS erlaubt Zugriff auf das Datensystem. Path hat Funktionen implementiert, welche Operationen auf den Pfad und verbundenen Attribute erlaubt.

```
1 from os import path
2
3 # Check for item existence and type
4 var1 = path.exists("textfile.txt")
5 var2 = path.isfile("textfile.txt")
6 var3 = path.isdir("textfile.txt")
7
8 # Work with file paths
9 var1 = path.realpath("textfile.txt")
10 var2 = path.split(path.realpath("textfile.txt"))
11
12 # Get the modification time
13 var3 = path.getmtime("textfile.txt")
14 var4 = path.getatime(path.realpath("textfile.txt")) # Der Path ist
15 gleich gesetzt mit dem
```

### 1.5.3 Shutil and Zip

Mit dem vorherigen Modul path konnten Dateinamen und Dateiverzeichnisse bearbeitet werden. Um Dateien zu bearbeiten, kopieren, verschieben und Operationen an mehrere Dateien durchzuführen, hilft *Shutil*. Dateien können kopiert werden.

Im Folgenden wird eine Kopie von der Beispieldatei `textfile.txt` erstellt und ein Archive des Ordners als Zip erstellt. Hinweis: Die Funktion `.realpath()` wirkt nur auf das aktuelle Verzeichnis.

---

```

1 #
2 # Example file for working with filesystem shell methods
3 #
4 import os
5 from os import path
6 import shutil
7
8 def main():
9 # make a duplicate of an existing file
10 if path.exists("textfile.txt"):
11 # get the path to the file in the current directory
12 soriginal = path.realpath("textfile.txt")
13 scr = path.split(soriginal)
14 scrG = str(scr[0])+str("/") + str(scr[1])
15 dcr = scr[0] + str("/") + str("textfile-Copy.txt")
16 # let's make a backup copy by appending "bak" to the name
17
18 shutil.copy(soriginal, dcr)

```

---

Die Funktion `.copystat()` transferiert Metadaten einer Datei zu einer anderen.

---

```

1 shutil.copy(soriginal, dcr)
2 shutil.copystat(soriginal, dcr) # Transfer Metadaten

```

---

Um eine den Namen zu ändern, wird das Paket `OS` verwendet. Mit `.rename(scr,dcr)` wird das Attribut umgeschrieben.

---

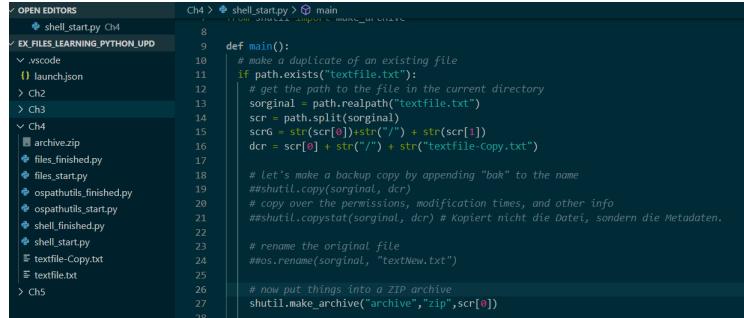
```

1 os.rename(soriginal, "textNew.txt")

```

---

Shutil enthält `make_archive`. Zip-Dateien können somit gebildet werden.



```

OPEN EDITORS
shell_start.py Ch4
EX_FILES_LEARNING PYTHON_UPD
vscode
launch.json
Ch2
Ch3
CH4
archive.zip
files_finished.py
files_start.py
ospathutils_finished.py
ospathutils_start.py
shell_finished.py
shell_start.py
textfile-Copy.txt
textfile.txt
Ch5

Ch4 X shell_start.py X main
8
9 def main():
10 # make a duplicate of an existing file
11 if path.exists("textfile.txt"):
12 # get the path to the file in the current directory
13 soriginal = path.realpath("textfile.txt")
14 scr = path.split(soriginal)
15 scrG = str(scr[0])+str("/") + str(scr[1])
16 dcr = scr[0] + str("/") + str("textfile-Copy.txt")
17
18 # let's make a backup copy by appending "bak" to the name
19 ##shutil.copy(soriginal, dcr)
20 ##shutil.copystat(soriginal, dcr) # kopiert nicht die Datei, sondern die Metadaten.
21
22 # rename the original file
23 ##os.rename(soriginal, "textNew.txt")
24
25 # now put things into a ZIP archive
26 shutil.make_archive("archive", "zip",scr[0])
27
28

```

Speziell für Zip Archive gibt es das Modul `zipfile`. Aus dem Modul wird im folgenden Beispiel das Objekt `ZipFile` importiert. Dies erlaubt eine genauere Bestimmung, was in das Archive hinein soll und was nicht.

---

```

1 from zipfile import ZipFile
2 # more fine-grained control over ZIP files
3 with ZipFile("testZip.zip","w") as newzip:
4 newzip.write(path.realpath("files_start.py"))
5 newzip.write(soriginal)

```

---

## 1.6 Working with Webdata

### 1.6.1 Urllib Request

Das Paket `urllib` enthält mehrere Module

- `urllib.request`

- urllib.error
- urllib.parse
- urllib.robotparser

das Modul `urllib.request` hat die Funktion Uniform Resource Locator (URL)s in Hypertext Transfer Protokoll (HTTP) abzurufen. Mit `.getcode()` wird eine Zahl wiedergeben, welche Auskunft über den Status des Abrufen der URL gibt. Es gibt folgende Klassen, welche angeben, ob eine Anfrage erfolgreich war:

- Informational responses (100–199)
- Successful responses (200–299)
- Redirects (300–399)
- Client errors (400–499)
- Server errors (500–599)

Am Beispiel der [Google-Startseite](#), ist die Abfrage erfolgreich.

---

```
1 webUrl = urllib.request.urlopen("http://www.google.com")
2 print(str(webUrl.getcode())) # Returns 200 (if the request was
 successful)
```

---

Mit `.read()` wird die komplette HTTP-Seite ausgelesen. Die Daten werden in dem Beispiel der Variable `data` übergeben.

---

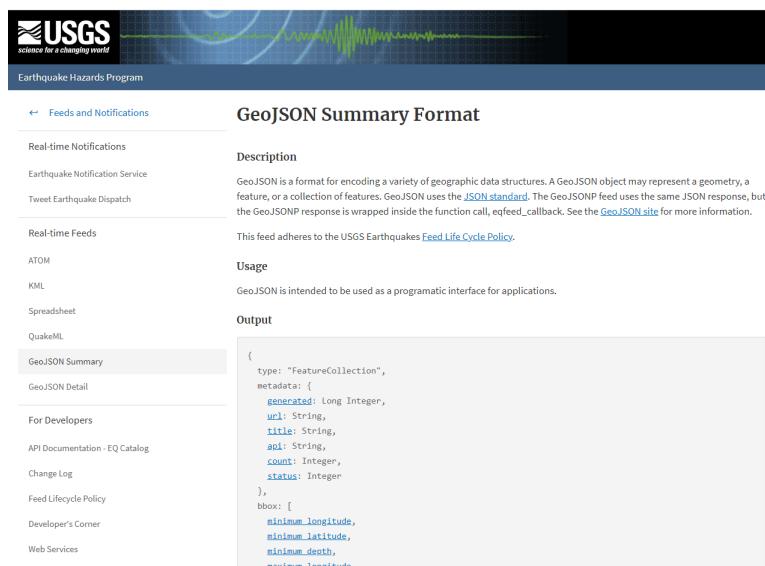
```
1 data = webUrl.read()
2 print(data)
```

---

Achtung: Selbst bei einfachen Seiten ist die Menge der ausgelesenen Daten hoch. Eine Darstellung über die Konsole kann daher eine hohe Rechenzeit benötigen.

### 1.6.2 JSON

Im diesem Beispiel soll es darum gehen, wie Daten aus dem Web per JavaScript Object Notation (JSON) Format ausgelesen werden können. Die Daten beziehen sich auf die Seismischen Erdbeben Daten.



The screenshot shows the USGS Earthquake Hazards Program website. The main navigation bar includes links for "Feeds and Notifications", "Real-time Notifications", "Earthquake Notification Service", "Tweet Earthquake Dispatch", "Real-time Feeds", "ATOM", "KML", "Spreadsheet", "QuakeML", "GeoJSON Summary" (which is highlighted), "GeoJSON Detail", "For Developers", "API Documentation - EQ Catalog", "Change Log", "Feed Lifecycle Policy", "Developer's Corner", and "Web Services". The "GeoJSON Summary" page has a header "GeoJSON Summary Format" and sections for "Description" and "Usage". The "Description" section explains that GeoJSON is a format for encoding a variety of geographic data structures. The "Usage" section states that GeoJSON is intended to be used as a programmatic interface for applications. On the right side of the page, there is a code snippet of GeoJSON:

```

type: "FeatureCollection",
metadata: {
 generated: Long Integer,
 url: String,
 title: String,
 api: String,
 count: Integer,
 status: Integer
},
bbox: [
 minimum_longitude,
 minimum_latitude,
 minimum_depth,
 maximum_longitude,
 maximum_latitude
]

```

Der Aufbau der Daten findet sich teilweise wie folgt aus:

```

{
 type: "FeatureCollection",
 metadata: {
 generated: Long Integer,
 url: String,
 title: String,
 azi: String,
 count: Integer,
 status: Integer
 },
 bbox: [
 minimum_longitude,
 minimum_latitude,
 minimum_depth,
 maximum_longitude,
 maximum_latitude,
 maximum_depth
],
 features: [
 {
 type: "Feature",
 properties: {
 mag: Decimal,
 place: String,
 time: Long Integer,
 updated: Long Integer,
 id: Integer,
 url: String,
 detail: String,
 felt: Integer,
 cdi: Decimal,
 mmi: Decimal,
 alert: String,
 status: String,
 tsunami: Integer,
 sig: Integer,
 net: String,
 }
 }
]
}

```

Im ersten Schritt wird das Objekt erzeugt, welches die URL öffnet: `.request.urlopen()`. Um zu prüfen, ob die Webseite sich anfragen lässt, wird der HTTP Code abgefragt. Ist dies Kontrolle positiv, wird die Webseite, die die JSON Datei enthält, ausgelesen: `.request.urlopen().read()`. Die Daten werden der Funktion `printResults` übergeben.

```

def main():
 # define a variable to hold the source URL
 # In this case we'll use the free data feed from the USGS
 # This feed lists all earthquakes for the last day larger than Mag 2.5
 urlData = "http://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/2.5_"

 # Open the URL and read the data
 webUrl = urllib.request.urlopen(urlData)
 if webUrl.getcode() == 200:
 printResults(webUrl.read())
 else:
 print("Seite lässt sich nicht öffnen.")

```

Mit dem Paket `JSON` welches JSON Formate entschlüsselt oder Daten in solche umwandelt.

- `String ()`
- `Boolom (false, true)`
- `Integer (10, 1.5, -4, ...)`
- `null`
- `Array ([...])`
- `Object ({...})`

Die Funktion `.loads()` ließt die Erdbebendaten ein. Diese können jetzt wie im Beispiel direkt angesteuert werden. Im Glossar wird auf die Logik von JSON eingegangen.

---

```

1
2 def printResults(data):
3 # Use the json module to load the string data into a dictionary
4 theJSON = json.loads(data)
5
6 # now we can access the contents of the JSON like any other Python
 object

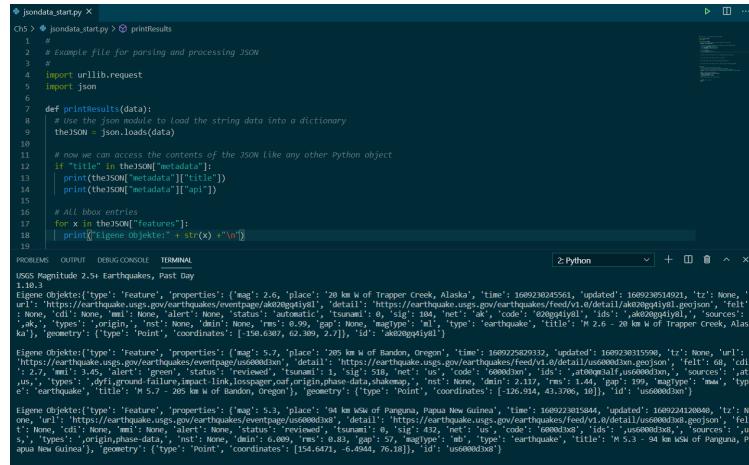
```

```

7 if "title" in theJSON["metadata"]:
8 print(theJSON["metadata"]["title"])
9 print(theJSON["metadata"]["api"])
10
11 # All bbox entries
12 for x in theJSON["features"]:
13 print("Eigene Objekte:" + str(x) +"\n")

```

---



The screenshot shows a terminal window with the following content:

```

jsondata_startpy X
GHS > jsondata_startpy > printResults
1 #
2 # Example file for parsing and processing JSON
3 #
4 import urllib.request
5 import json
6
7 def printResults(data):
8 # Use the json module to load the string data into a dictionary
9 theJSON = json.loads(data)
10
11 # now we can access the contents of the JSON like any other Python object
12 print(theJSON["metadata"]["title"])
13 print(theJSON["metadata"]["api"])
14
15
16 # ALL bbox entries
17 for x in theJSON["features"]:
18 print("Eigene Objekte:" + str(x) +"\n")
19
20
21
22
23
24
25
26
27

```

Below the code, the terminal displays several JSON objects representing earthquake events. Each event has properties like 'mag', 'place', 'time', 'updated', 'id', 'type', and 'geometry'. The 'place' field often includes descriptive text such as '20 km W of Trapper Creek, Alaska' or '94 km WSW of Panguna, Papua New Guinea'.

Abbildung 188: Ausgabe: JSON

Weite Aufgaben wurden gestellt, um spezifische Anfragen an die Erdbebendaten zu stellen.

```

1 def printResults(data):
2 # Use the json module to load the string data into a dictionary
3 theJSON = json.loads(data)
4
5 # now we can access the contents of the JSON like any other Python
6 # object
7 if "title" in theJSON["metadata"]:
8 print(theJSON["metadata"]["title"])
9 print(theJSON["metadata"]["api"])
10 print(theJSON["metadata"]["count"])
11
12 # All bbox entries
13 y = 0
14 for x in theJSON["features"]:
15 y += 1
16 print("Eigene Objekte:" + str(x) +"\n")
17
18 # output the number of events, plus the magnitude and each event name
19 print(str(y))
20
21 # for each event, print the place where it occurred
22 for x in theJSON["features"]:
23 print("Ort: " + str(x["properties"]["place"]) + " mit " + str(x["
24 properties"]["mag"]))
25
26 # print the events that only have a magnitude greater than 4
27 for x in theJSON["features"]:
28 if x["properties"]["mag"] >=4:

```

```
28 print(x["properties"]["place"])
```

---

### 1.6.3 HTML Parser (Plus: `itertools`)

Mit dem Modul `html.parser` ist eins von 4 Submodul von Hypertext Markup Language (HTML) :

- `html.escape` converts special HTML in strings.
- `html.unescape` converts named and numeric character references in strings.
- `html.entities` defines dictionaries.

Mit `parser` wird einen HTML String durchsucht - dies kann zu einer Datei oder direkt zu einer Webseite referieren. Das Modul besitzt nur eine Klasse - `HTMLParser`. Anders, als bei anderen Modulen, wird hier eine abgeleitete (Sub-Klasse) Klasse gebildet.

```
import urllib.request
from html.parser import HTMLParser
import itertools # erlaubt über mehrere List gleichzeitig zu interieren
class HTML_Parser_My(HTMLParser):
```

Der Input `HTMLParser` dient als Referenzpunkt zur Hauptklasse. Die Klasse, `HTMLParser` bietet Methoden an, welche überschrieben werden können. Diese nennen sich `handler`

```
The following methods are called when data or markup elements are encountered and they are meant to be
overridden in a subclass. The base class implementations do nothing (except for handle_startendtag()):
HTMLParser.handle_starttag(tag, attrs)
This method is called to handle the start of a tag (e.g. <div id="main">).

The tag argument is the name of the tag converted to lower case. The attrs argument is a list of (name,
value) pairs containing the attributes found inside the tag's <> brackets. The name will be translated to
lower case, and quotes in the value have been removed, and character and entity references have been
replaced.

For instance, for the tag , this method would be called as
handle_starttag('a', [('href', 'https://www.cwi.nl/')]).

All entity references from html.entities are replaced in the attribute values.

HTMLParser.handle_endtag(tag)
This method is called to handle the end tag of an element (e.g. </div>).

The tag argument is the name of the tag converted to lower case.

HTMLParser.handle_startendtag(tag, attrs)
Similar to handle_starttag(), but called when the parser encounters an XHTML-style empty tag (). This method may be overridden by subclasses which require this particular lexical information; the
default implementation simply calls handle_starttag() and handle_endtag().

HTMLParser.handle_data(data)
This method is called to process arbitrary data (e.g. text nodes and the content of <script>...</script>
and <style>...</style>).

HTMLParser.handle_entityref(name)
```

Beim Einlesen des HTML-Strings wird durch die Methode `.feed()` die Variablen

- tag
- attrs
- data
- etc.

erstellt. Weiter werden die Handler-Methoden angewandt auf den String. Im folgenden Beispiel wird beim Durchlaufen jeder Start und End Tag sowie die Daten zwischen ihnen in einer Liste gespeichert.

```
1 class HTML_Parser_My(HTMLParser):
2 def handle_starttag(self, tag, attrs):
3 global startTagList
4 startTagList.append(tag)
5
6 def handle_endtag(self, tag):
7 global endTagList
8 endTagList.append(tag)
9
10 def handle_data(self, data):
11 global dataList
12 dataList.append(data)
```

---

Um diese außerhalb der Methode aufzurufen, müssen die Variablen definiert werden.

---

```
1 startTagList = []
2 endTagList = []
3 dataList = []
4 Limit = [None, None, None] # Setzt die Minimal Anzahl der
 auszugebenen Einträge dar. In dem Fall 3
```

---

Da Paket *itertools* wird verwendet, um durch die verschiedenen Liste gleichzeitig zu iterieren.

---

```
1 print(startTagList)
2 print(endTagList)
3 print(dataList)
4 for (a,b,c,d) in zip(startTagList, endTagList, dataList, Limit):
5 print(a,b,c)

6
7 ### Output
8 # ['html', 'head', 'meta', 'title', 'meta', 'meta', 'meta', 'link', 'link',
 'body', 'div', 'header', 'h1', 'nav', 'p', 'a', 'p', 'a', 'div',
 'footer', 'p']
9 #[['meta', 'title', 'meta', 'meta', 'link', 'link', 'head', 'h1',
 'header', 'a', 'p', 'a', 'p', 'nav', 'div', 'p', 'footer', 'div',
 'body', 'html']]
10 #['\n', '\n ', '\n ', '\n ', 'Sample HTML Document', '\n ',
 '\n ', '\n ', '\n ', '\n ', '\n ', '\n ', '\n\n',
 '\n ', '\n ', '\n ', '\n ', 'HTML Sample File', '\n
 ', '#'\n ', '\n ', '\n ', '\n ', 'Home', '\n
 ', '\n ', '\n ', '\n ', 'Contact', '\n
 ', '\n ', '\n ', '\n\n'
11 #', '\n ', '\n ', '\n ', '© Copyright by Administrator', '\n
 ', '\n ', '\n ', '\n ', '\n', '\n']
12 #html meta
13 #
14 #head title
15 #
16 #meta meta
```

---

#### 1.6.4 xml

Das Paket *xml.dom.minidom* oder *xml.dom* mit *minidom* bietet Möglichkeiten den eine Datei direkt Zeile für Zeile zu bearbeiten. Es ist Vorsicht beim *minidom* Paket zu bewahren. Bei einer Auslesung von Mailware ist das Paket nicht sicher.

## 2 Python with Excel

### 2.1 Pandas

**Teil VI**

**GitHub**

# 1 Konzepte

## 1.1 Befehle

**my-slide Verzeichnis**

**my-slide Verzeichnis**

**git ls-files** Zeigt alle Dateien in einem Verzeichnis an.

**copy con <file>** Dieser Befehl funktioniert unter Git bash. Die entsprechenden Datei wird in den aktiven Verzeichnis erstellt.

**git status** Gib alle Änderungen eines Verzeichnisses wieder.

**git clone <URL.git>** Das URL mit .git wird in dem aktiven Verzeichnis geklont.

**git branch <name>** Ein neuer Branch wird im lokalen Bereich erzeugt.

**git checkout <local branch>** Ändert den aktiven lokalen Branch.

**git push -set-upstream origin <Branch>** Der erzeugte Branch wird zum Remote Repository geladen.

**git pull origin <Branch>** Ein Update (pull) eines spezifischen Branch wird gezogen.

**git branch -all** Alle Branches lokal und remote werden angezeigt.

**git branch -m <Name>** Der Name des Branches, welcher gerade aktiv ist, kann mit dem Befehl geändert werden.

**git -add <file>** Bereitet Änderungen aus dem Working Bereich für den nächsten Commit vor.

**git commit -m "..."** Mit jedem Commit wird ein Snapshot der Dateien gemacht. Diese werden unter Version Control System (VCS) abgespeichert. Eine Nachricht ist notwendig.

**git push** Alle Commit werden für den ausgewählten Branch gepusht.

**git pull origin <Branch>** Aktualisiert den spezifischen Branch.

**git merge <Branch>** Der Master Branch wird mit dem ausgewählten Branch vereinigt.

**git branch -d <Branch>** Löscht den lokalen Branch, welcher schon mit master vereinigt ist.

**git branch -D <Branch>** Löscht den lokalen Branch.

**git push origin --delete <Branch>** Löscht den remote Branch, solang er auf Git Hub noch nicht "archiviert" wurde, sonst kann der Branch nicht gefunden werden.

**git remote prune origin** Löscht leere Branch Hüllen.

## 1.2 Unterschied zwischen git bash, cmd und gui

**Git bash** Es handelt sich hier um eine UNIX Shell. Dies bietet sich an, wenn Linux schon bekannt ist.

**Git CMD** Es handelt sich hier um eine UNIX Shell. Dies bietet sich an, wenn Windows bekannt ist.

**Git GUI** Bietet die Funktionalität von git in einer grafische Oberfläche an.

*stackoverflow o. D.*

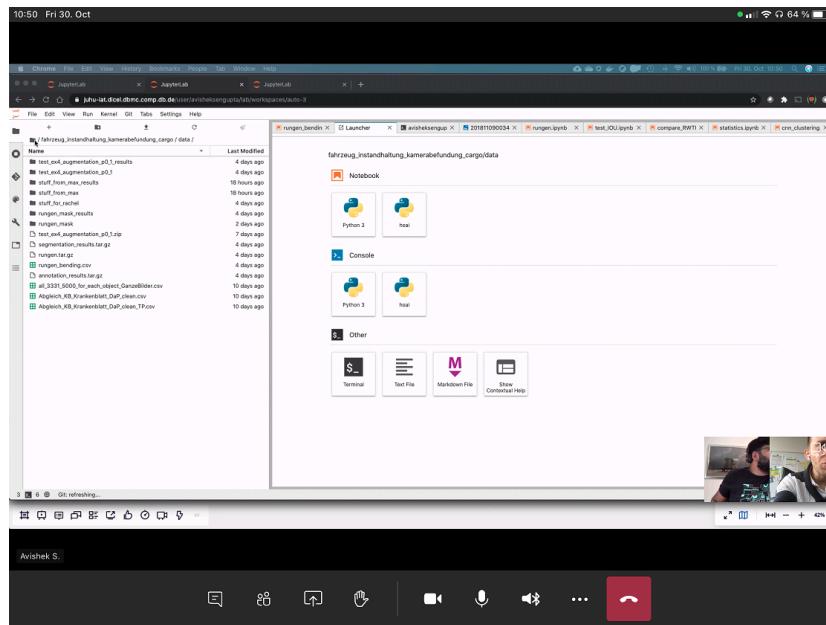
### 1.3 Staging Area / Index / Cache

Der Staging Bereich oder Index erlaubt inhaltlich verbundenen Dateien/ Änderungen an Dateien aus dem Working Bereich vorzuhalten und per **git add**.

```
1 C:\Users\Paul J\Documents\GitHub\github-slideshow> git status
2 On branch dell_Branch
3 Your branch is up to date with 'origin/dell_Branch'.
4
5 Untracked files:
6 (use "git add <file>..." to include in what will be committed)
7 - posts/test-dell.md - Copy.txt
8
9 nothing added to commit but untracked files present (use "git add" to track)
```

Dateien, welche diesem Index zugeordnet wurden, werden per **git commit** mit einer Nachricht zum lokalen Git Repository übertragen. Die gesammelten Commits können per **git push** zum Remote Repository übertragen werden.

```
1 C:\Users\Paul J\Documents\GitHub\github-slideshow> git status
2 On branch dell_Branch
3 Your branch is ahead of 'origin/dell_Branch' by 1 commit.
4 (use "git push" to publish your local commits)
5
6 nothing to commit, working tree clean
```

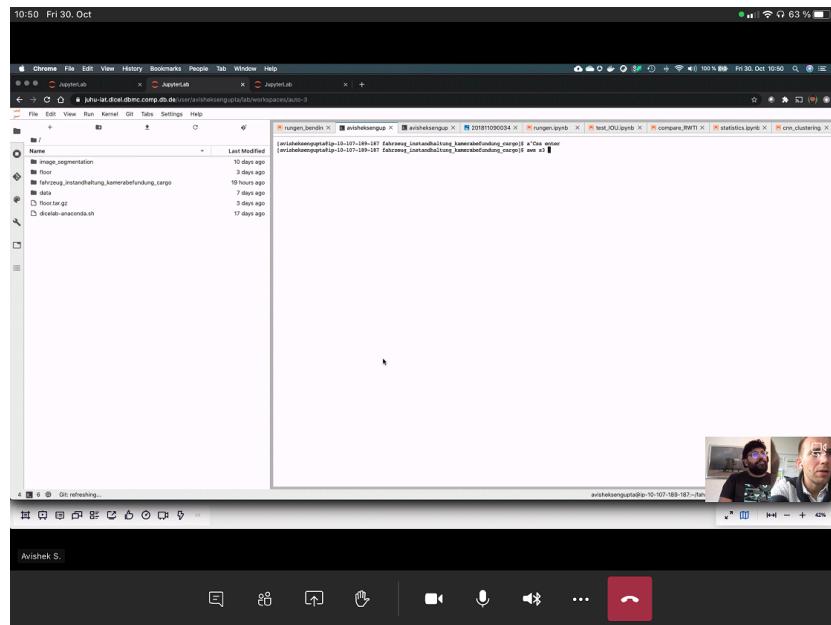


Eine Datei im Git Ordner kann in zwei Kategorien untergliedert werden:

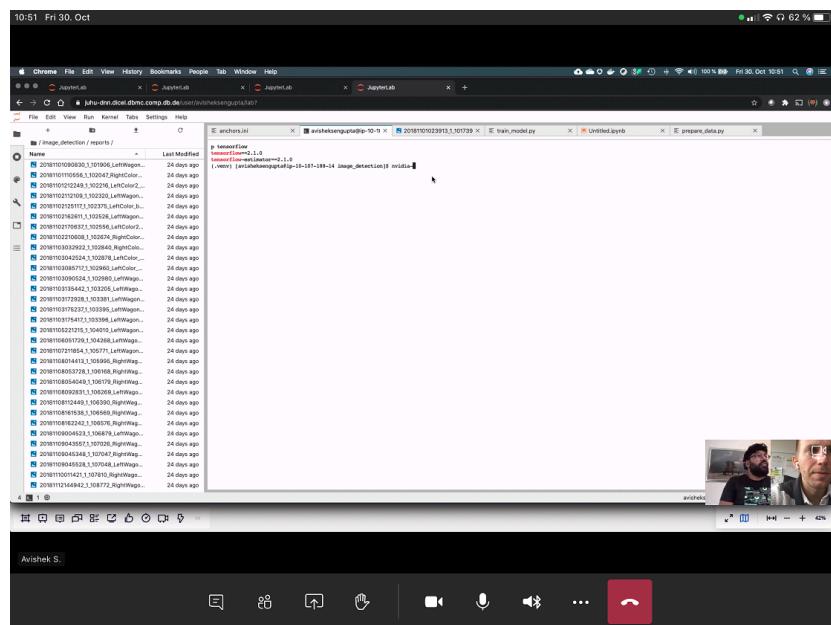
- tracked und
- untracked.

In früheren Git Versionen konnte eine ungetrackte Datei Git Repository zugeordnet werden ohne als *staged* markiert zu werden, sodass sie Teil der VCS wurde. Mit **git add <file>** wird diese Datei heute zum Staging Bereich zugeordnet und von Git getrackt. Beim nächsten **git push** wird diese Datei ans Remote Git Repository gesendet.

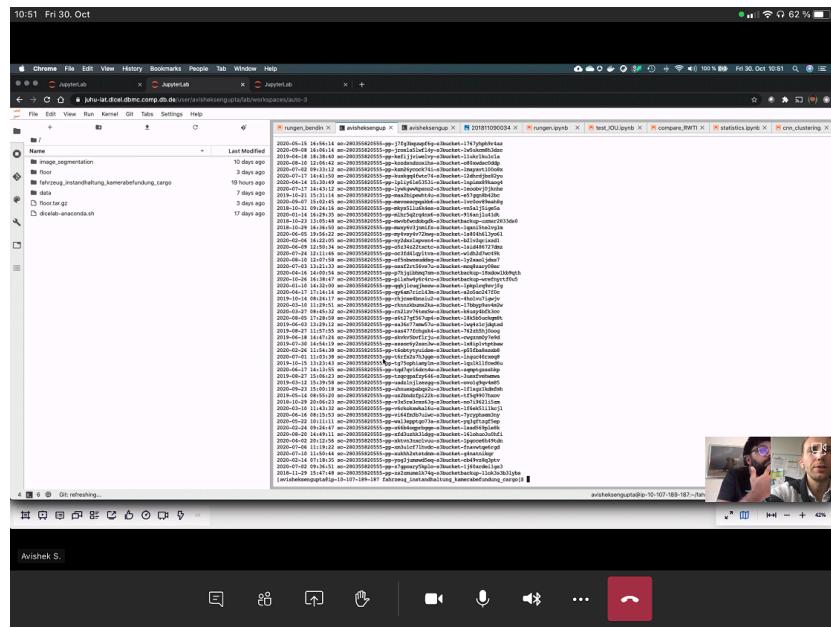
Mit Git GUI ist eine Übersicht gegeben, welche Dateien noch nicht im Stage Modus sind und welche schon.



Die Commits werden pro Branch aufgeteilt.



Zu jedem Schritt existiert ein Befehl in Git Repository.



## 1.4 Committing

Commits sind Snapshots/ Milestones des Projektes/ Ordners oder Dateien. Dies werden werden auf dem lokalen Git Repository abgespeichert. Commits können auch wieder gelöscht werden. Wenn alle Commits bereit sind, geteilt zu werden, können sie per **git push origin <Branch>** auf das remote Git Repository geladen werden. Über Git Hub stehen die Veränderungen und die History bereit zur Bearbeitung.

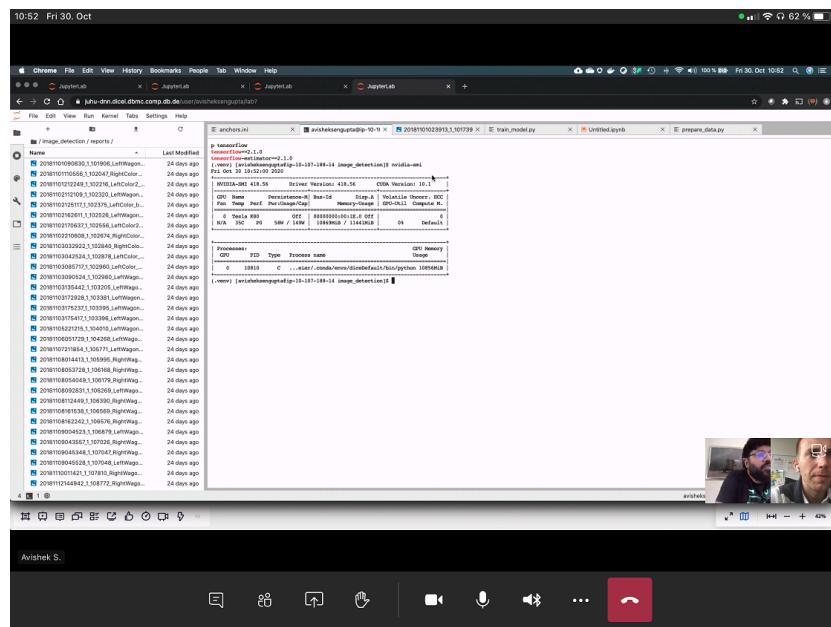


Abbildung 189: Richtig Auswahl des Branches & History anzeigen lassen.

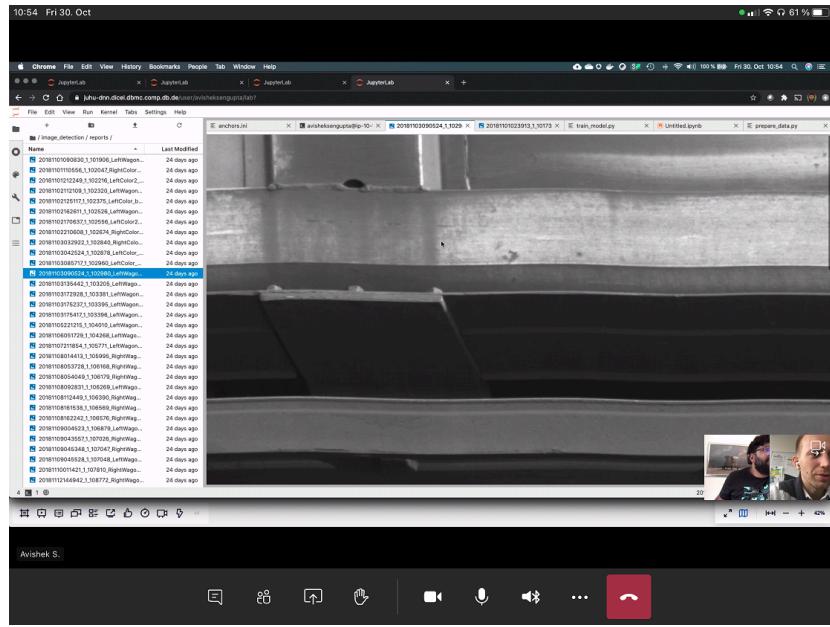


Abbildung 190: Zu jedem Milstone/ Linie besteht die Möglichkeit Kommentare anzubringen.

## 2 Example: github-slideshow

### 2.1 Clone Repository and Create a Branch

Ziel in diesem Abschnitt ist es, das Mit der Clone Anweisung wird das Git Repository auf den lokalen Server des Users gespielt. Die VCS wird in Git hinterlegt und speichert die Änderungen an den Daten. Damit die Änderungen sich einfach einbinden lassen und von verschiedenen User erstellt werden, wird ein **Branch** erstellt.

```

1 /// Change directory to where the repository should be saved
2 git cd C:\Users\PaulJulitz\Documents\GitHub
3 /// Clone repository to local server (.../Username/NameRepository)
4 git clone https://github.com/pauljulitz/github-slideshow.git /**
5 /// Navigate to the repository in your shell
6 git cd C:\Users\PaulJulitz\Documents\GitHub\github-slideshow
7 /// Create a new branch
8 git branch my-slide

```

### 2.2 Check all the branches of one repository

In einem Git Repository können mehrere Branches erstellt werden. Welche für das gewissen Git Repository zugewiesen sind, kann über `branch -all` in einen Git Repository dargestellt werden.

```

1 C:\Users\PaulJulitz>cd C:\Users\PaulJulitz\Documents\GitHub\github-slideshow
2
3 C:\Users\PaulJulitz\Documents\GitHub\github-slideshow> git branch --all
4 master
5 *my-slide
6 secoundBranch
7 remotes/origin/HEAD -> origin/master
8 remotes/origin/master
9 remotes/origin/my-slide
10 remotes/origin/secoundBranch

```

Der aktuelle Branch wird in Grün mit einer \* angezeigt.

## 2.3 Change branch name

Mit dem Befehl **-m <new name>** kann der Name des lokalen, ausgewählten Branch geändert werden. Wenn jedoch der gleiche Name für einen Branch ausgewählt wurde, wird der Zusatz *-lokal* angefügt.

```
1 C:\Users\PaulJulitz\Documents\GitHub\github-slideshow> git branch -m newName
```

## 2.4 Change aktiv branch

Ein Branch kann mit **git checkout <local branch>** gewechselt werden.

```
1 C:\Users\PaulJulitz\Documents\GitHub\github-slideshow> git branch --all
2 master
3 * my-slide
4 secoundBranch
5 remotes/origin/HEAD -> origin/master
6 remotes/origin/master
7 remotes/origin/my-slide
8 remotes/origin/secoundBranch
9
10 C:\Users\PaulJulitz\Documents\GitHub\github-slideshow> git checkout secoundBranch
11 Switched to branch 'secoundBranch'
12 Your branch is up to date with 'origin/secoundBranch'.
13
14 C:\Users\PaulJulitz\Documents\GitHub\github-slideshow> git branch --all
15 master
16 my-slide
17 * secoundBranch
18 remotes/origin/HEAD -> origin/master
19 remotes/origin/master
20 remotes/origin/my-slide
21 remotes/origin/secoundBranch
```

## 2.5 Push new local branch

Mit **--set-upstream origin <branch>** wird der angesteuerte Branch hochgeladen.

```
1 git push --set-upstream origin my-slide
```

## 2.6 Check the status of a branch

Der Befehl **git status** gibt an,

- welcher Branch aktuell aktiv ist,
- ob es Änderungen im Origin Projekt gibt und
- welche Änderungen gepusht werden müssen.

```
1 C:\Users\PaulJulitz\Documents\GitHub\github-slideshow> git status
2 On branch secoundBranch
3 Your branch is up to date with 'origin/secoundBranch'.
4
5 nothing to commit, working tree clear
```

## 2.7 Create a new file

Unter Git CMD wird der Befehl **copy con** nicht ausgeführt. Der Befehl **ls** ebenso nicht. Mit

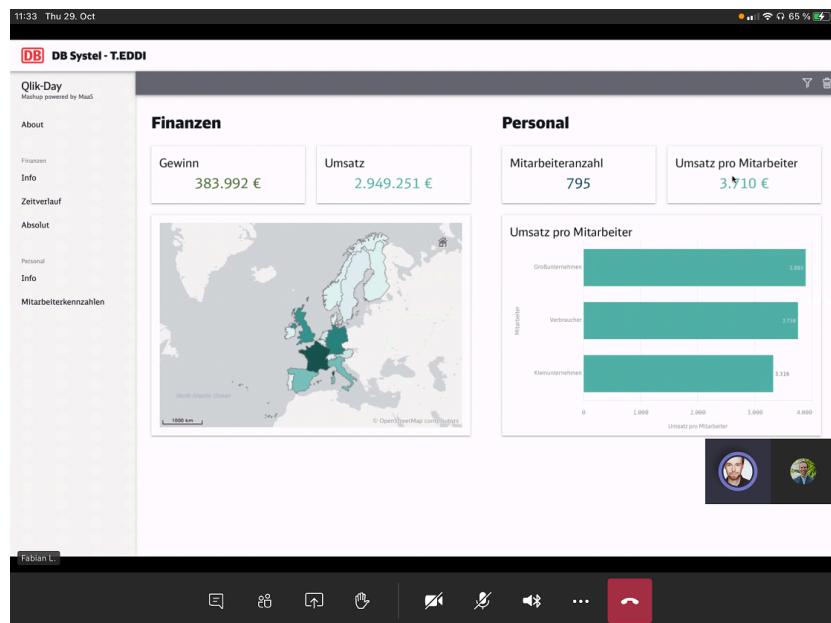
## 2.8 Update Branch

Mit dem Befehl **git pull origin <branch>** wird der spezifische Branch aktualisiert.

```
1 C:\Users\PaulJulitz\Documents\GitHub\github-slideshow> git pull origin my-slide
2 remote: Enumerating objects: 10, done.
3 remote: Counting objects: 100% (10/10), done.
4 remote: Compressing objects: 100% (7/7), done.
5 remote: Total 8 (delta 3), reused 0 (delta 0), pack-reused 0
6 Unpacking objects: 100% (8/8), 1.46 KiB | 14.00 KiB/s, done.
7 From https://github.com/pauljulitz/github-slideshow
8 * branch my-slide -> FETCH_HEAD
9 d64d186..e0c65fb my-slide -> origin/my-slide
10 Updating d64d186..e0c65fb
11 Fast-forward
12 _posts/0000-01-02-PaulJulitz.md | 6 ++++++
13 1 file changed, 6 insertions(+)
14 create mode 100644 _posts/0000-01-02-PaulJulitz.md
15
16 C:\Users\PaulJulitz\Documents\GitHub\github-slideshow> git status
17 On branch my-slide
18 Your branch is up to date with 'origin/my-slide'.
19
20 nothing to commit, working tree clean
```

## 2.9 Pull Request

Die kann über Github selbst erfolgen:



Oder über command-line interpreter (cmd) sind die folgenden Schritte notwendig. Der Master Branch wird aktiviert.

```
1 C:\Users\PaulJulitz\Documents\GitHub\github-slideshow> git checkout master
```

Der benötigte Branch wird über **git merge** vereinigt.

```
1 C:\Users\PaulJulitz\Documents\GitHub\github-slideshow> git merge my-slide
```

Dieser muss jetzt gepusht werden.

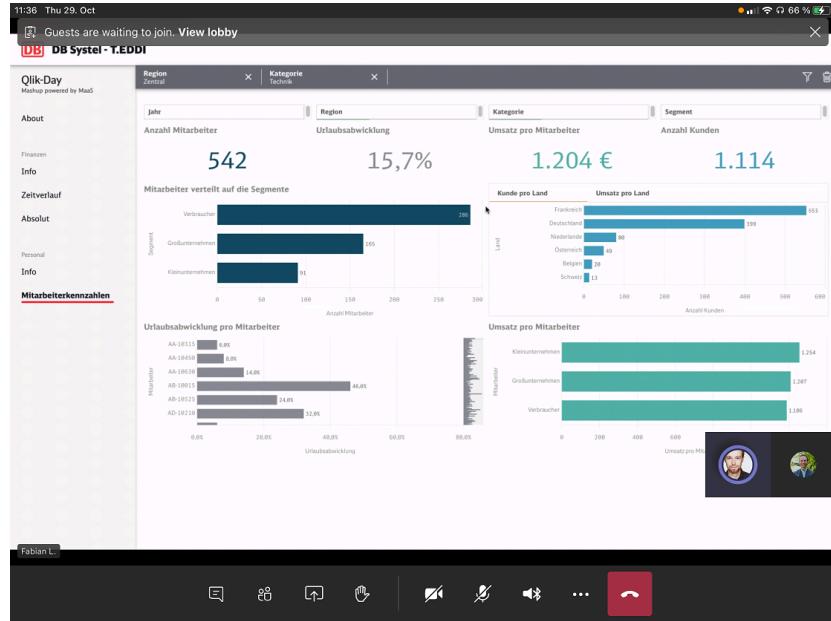
```
1 C:\Users\PaulJulitz\Documents\GitHub\github-slideshow> git push
```

Der alte Branch kann jetzt gelöscht werden.

```
1 C:\Users\PaulJulitz\Documents\GitHub\github-slideshow> git branch -d my-slide
```

## 2.10 Delete local branch

Wenn die Veränderungen überprüft wurden und der Branch vereinigt wurde, wird das Prinzip verfolgt, den Branch zu löschen. Wenn dies über die Git Hub Plattform passt, so bleibt eine "Ablage" weiter bestehen.



```
1 C:\Users\Paul J\Documents\GitHub\github-slideshow>git push origin --delete dell_Branch
2 error: unable to delete 'dell_Branch': remote ref does not exist
3 error: failed to push some refs to 'https://github.com/PaulJulitz/github-slideshow.git'
```

Wenn dies nicht getan wurde oder der Branch wieder zurückgesetzt wurde, wird er remote Branch gelöscht, durch:

```
1 C:\Users\Paul J\Documents\GitHub\github-slideshow>git branch --all
2 * dell_Branch
3 master
4 remotes/origin/HEAD -> origin/master
5 remotes/origin/dell_Branch
6 remotes/origin/dell_Branch_II
7 remotes/origin/master
8 remotes/origin/my-slide
9 remotes/origin/secoundBranch
10
11 C:\Users\Paul J\Documents\GitHub\github-slideshow>git pull
12 Already up to date.
13
14 C:\Users\Paul J\Documents\GitHub\github-slideshow>git push origin --delete dell_Branch_II
15 To https://github.com/PaulJulitz/github-slideshow.git
16 - [deleted] dell_Branch_II
17
18 C:\Users\Paul J\Documents\GitHub\github-slideshow>git push origin --delete dell_Branch
19 To https://github.com/PaulJulitz/github-slideshow.git
20 - [deleted] dell_Branch
21
22 C:\Users\Paul J\Documents\GitHub\github-slideshow>git branch --all
23 * dell_Branch
24 master
25 remotes/origin/HEAD -> origin/master
26 remotes/origin/master
27 remotes/origin/my-slide
28 remotes/origin/secoundBranch
```

## 2.11 Delete remote branch

Der Befehl ist **git push origin --delete origin/<branch>**. Dieser ist an ein Pull Request welcher den Branch im remote Git Repository löscht. Wenn dieser Branch nicht mehr existiert, kommt ein Fehler zurück.

## 2.12 Prune deletet remote branches

Der Befehl **git remote prune origin** löscht alle Hüllen eines Branches.

## 3 Designing your Github Page

Formatierung einer .md Datei. Das README.md wir als Startseite unter der Auflistung der Daten angezeigt. Dies zu Formatieren kann helfen, eine Einführung in das Git Repository zu geben.

- Ein Hashtag oder mehrere Hastagss geben die Größe der Überschrift an.
- Bilder aus dem Web werden per ![profile-image](<https://apod.nasa.gov/>...)eingefügt.
- Wenn nur ein Bildlink eingefügt werden soll, muss das ! vor dem Platzhaltern [...].
- Die Logik von Whatsapp ist auch hier Teilweise bei .md Formartierung anzuwenden (Kursiv, Fett, List).

**Teil VII**

**Qlik Sense**

# 1 Sammelsurium - Qlik Sense

## 1.1 Load

**Load and Select Statement** Was macht den Unterschied aus und wo liegen die Gemeinsamkeiten.

```
1 TabellenName :
2 Load
3 Spaltenname_1 ,
4 Spaltenname_2 ,
5 [
6] ;
7
```

## 1.2 Incremental Loading

Es ist möglich, nicht eine komplette Datenbank immer und immer wieder zu laden, um an die neuesten Datensätze zu kommen. Es ist möglich, durch [Incremental Loading](#). In dem Beispiel, [Qlik Snippet - Incremental Loading](#), wird über einen aufgesetzten SQL-Server gezeigt, wie neue Datensätze geladen werden können.

- Erst wird die Initialladung vorgenommen. Diese wird in einem QlikView Data (.qvd) gespeichert.
- Im zweiten Schritt wird die Incremental Loading Prozedur vor der Laden-Prozedur geschalten. Diese prüft jetzt, ob in der SQL Datenbank neuere Werte als in der dazugehörigen .qvd vorliegen. Diese werden dann geladen und in die alte .qvd gespeichert.

## 1.3 Data connection types

**Folder** Daten die lokal auf dem Server hinterlegt werden, können so abgerufen werden.

**Open Database Connectivity (ODBC)** Datenbank-Verbindungen, die am Server angelegt werden, können so mit Qlik Sense verbunden werden.

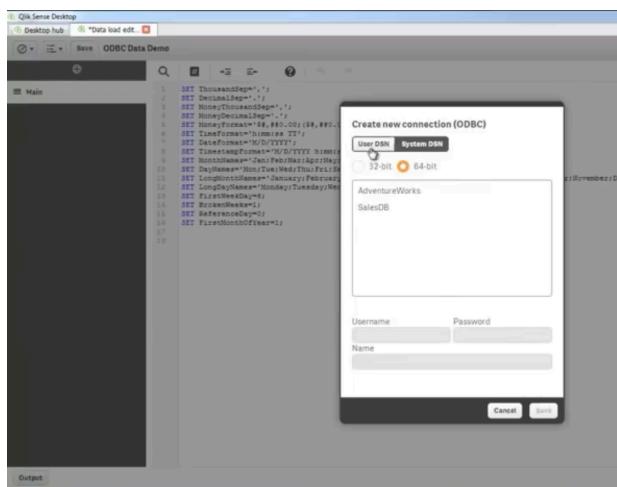


Abbildung 191:

Qlik bietet an, dass ODBC über Qlik Sense direkt angesteuert werden können oder über Microsoft.

**Load/ Select** Beide Ausdrücke generieren Tabellen. Dabei wird der Load Ausdruck verwendet, wenn es sich um lokale Dateien handelt. Der Select Ausdruck wird verwendet, wenn Datenbanken adressiert werden. Oft wird dabei auf SQL zurückgegriffen.

## 2 3 - Udemy - Certificate in Qlik Sense Analytics Development

### 2.1 Budgeting and KPI (goal) setting

#### 2.1.1 DevHub

Im Hauptmenü von Qlik Sense kann die DevHub Umgebung angesteuert werden.

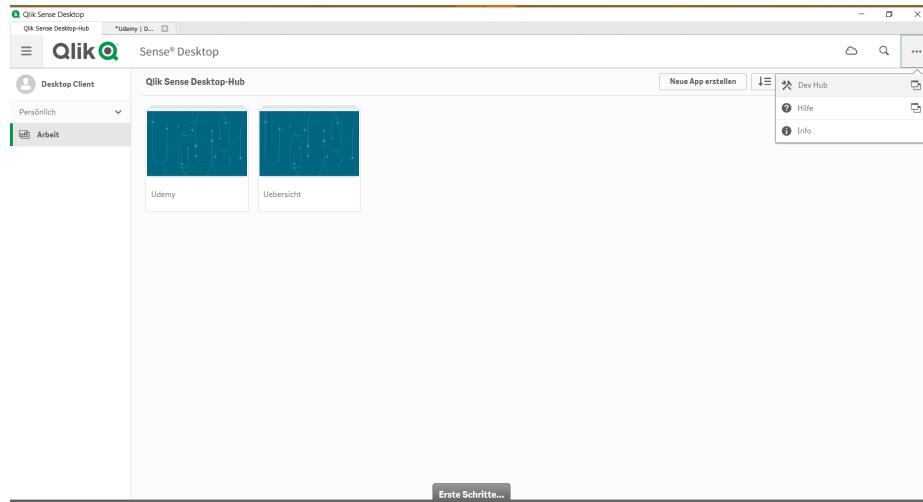


Abbildung 192:

Diese bietet an

- MashUps,
- Erweiterungen von Visualisierungen und
- Widget zur Verfügung zu stellen.

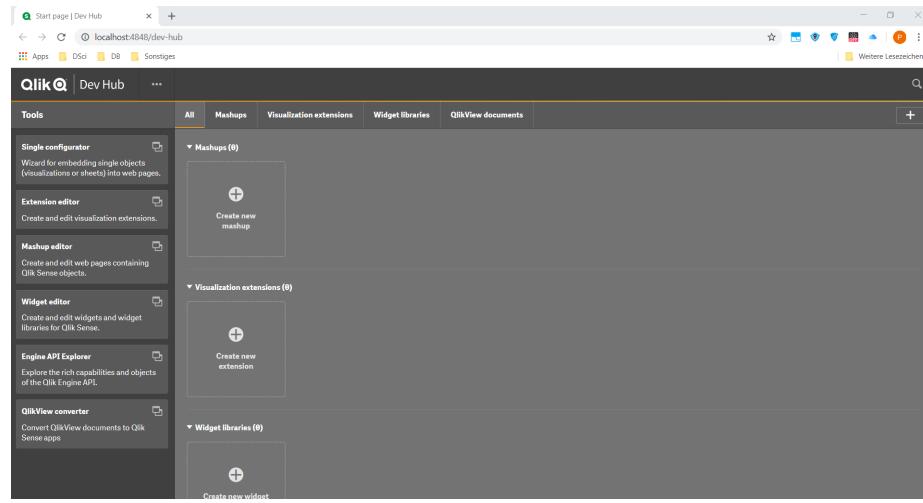


Abbildung 193:

Wichtige Punkte sind:

- Diese Features werden verwendet, um sie in andere Umgebungen einzubinden. Mashups werden so verwendet, um sie in SharePoint einzubinden.

- Die Plattform **Qlik Branch** gibt eine Übersicht über die verschiedenen Anbindungsmöglichkeiten zu Qlik Sense.
- Visualisierungserweiterungen werden über JavaScript (JS) programmiert.

### 2.1.2 Von Hub bis Sheet

Die Hub Oberfläche unterscheidet sich von dem Qlik Management Console (QMC), dass die Datenaufbereitung über diese Oberfläche zur Verfügung gestellt wird und QMC die Verwaltung übernimmt.

**Stream** In einem Stream können mehrere Apps veröffentlicht werden. Die Berechnungen wird über die Stream festgelegt.

**Apps** In einer App wird die Datengrundlage bestimmt. Das zugrundeliegende Datenmodell dient den Sheets.

**Sheets** Über Sheets können Visualisierungen erstellt werden.

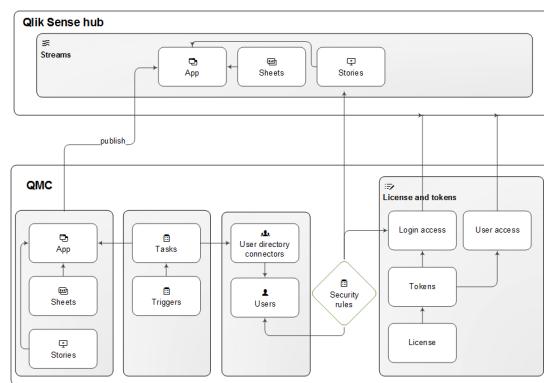


Abbildung 194:

Am Beispiel des Streams **Arbeit** wird die App **Udemy** erstellt.

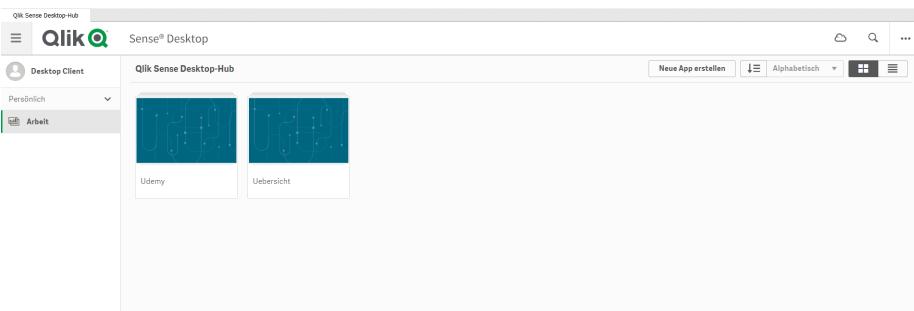


Abbildung 195:

Beim Öffnen der App erscheint die der Datenmanager.

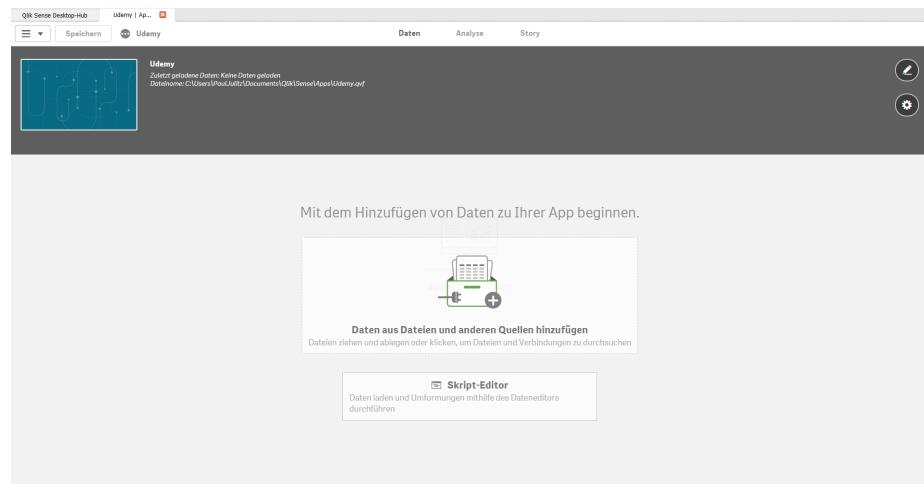


Abbildung 196:

Über diesen können verschiedenen Verbindung angesteuert werden.

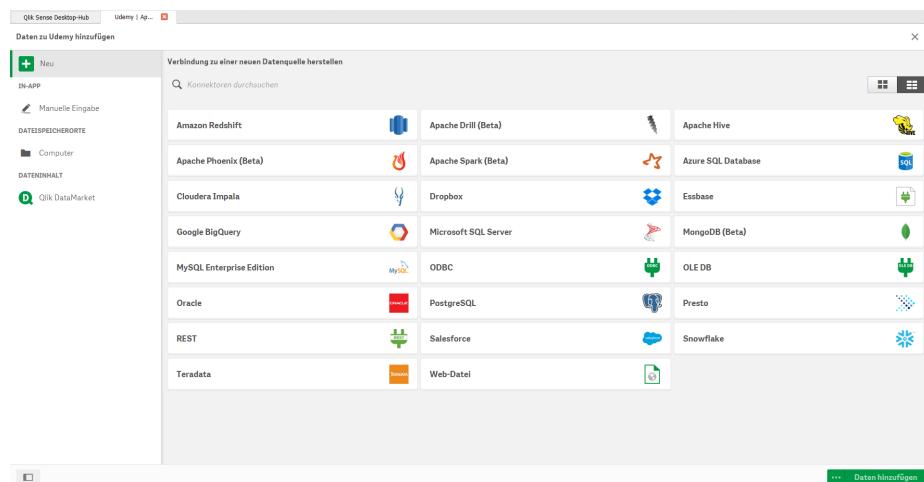


Abbildung 197:

Wird auf lokale Dateien zugegriffen, sieht das wie folgt aus:

Daten zu Udemy hinzufügen

← CalendarAndMonthlyBudget.xlsx

Tabellen

Datelformat: Excel (XLSX)

Feldnamen: Eingebettet. Feldname

Größe des Headers: 0

Tabellen filtern

Folder filtern

Tabellen

Filter

YearMo... Year Mo... Month... BudgetCategory BudgetSubCategory BudgetValue MarketingB...

20151 2015 January 1 FIXED STARTUP COSTS Fixtures and Equipment 10000  
20151 2015 January 1 FIXED STARTUP COSTS Decoration and remodeling 10000  
20151 2015 January 1 FIXED STARTUP COSTS Utilities and equipment 500  
20151 2015 January 1 FIXED STARTUP COSTS Installation costs 500  
20151 2015 January 1 FIXED STARTUP COSTS Starting inventory 5000  
20151 2015 January 1 FIXED STARTUP COSTS Deposits with public utilities 10000  
20151 2015 January 1 FIXED STARTUP COSTS Legal and other professional fees 500  
20151 2015 January 1 FIXED STARTUP COSTS Licenses and permits 500  
20151 2015 January 1 FIXED STARTUP COSTS Advertising and promotion for open 500  
20151 2015 January 1 FIXED STARTUP COSTS Cash 750  
20151 2015 January 1 FIXED STARTUP COSTS Other 200  
20151 2015 January 1 FIXED STARTUP COSTS Domain.com Registration 55  
20151 2015 January 1 INCOME Sales forecast 60000  
20151 2015 January 1 INCOME Interest income 2500  
20151 2015 January 1 INCOME Actual sales (gain/loss) 0  
20151 2015 January 1 PERSONNEL EXPENSES Wages 9500  
20151 2015 January 1 PERSONNEL EXPENSES Employee benefits 4000  
20151 2015 January 1 PERSONNEL EXPENSES Commission 5000  
20151 2015 January 1 OPERATING EXPENSES Advertising 3000  
20151 2015 January 1 OPERATING EXPENSES Rent 0  
20151 2015 January 1 OPERATING EXPENSES Cash discounts 1500  
20151 2015 January 1 OPERATING EXPENSES Delivery costs 2000  
20151 2015 January 1 OPERATING EXPENSES Depreciation 1000

... Daten hinzufügen

Abbildung 198:

Nachdem die Daten geladen werden.

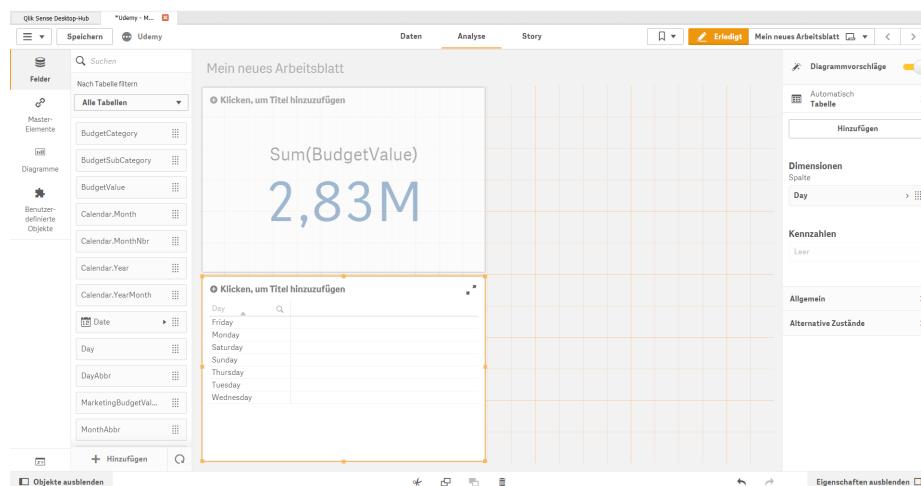


Abbildung 199:

kann ein neues Arbeitsblatt angelegt werden.

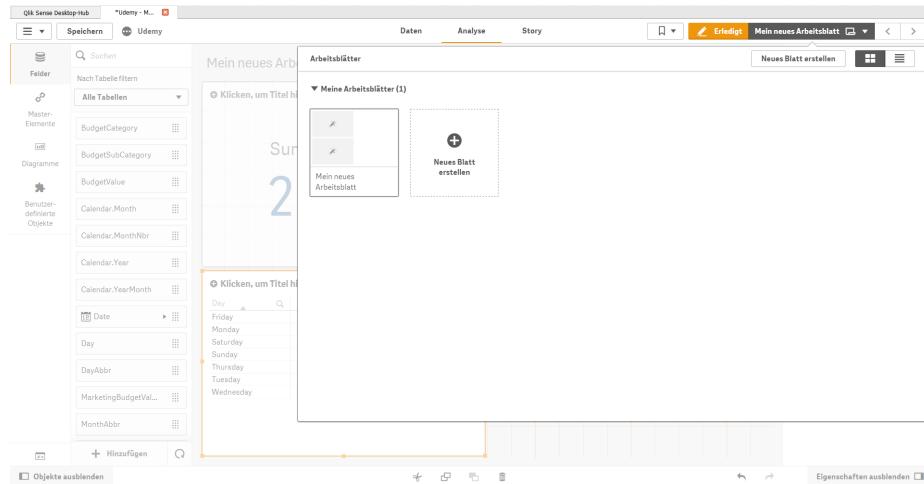


Abbildung 200:

Die Qlik Sense speichert die Daten für eine App mit allen Informationen zu den Arbeitsblättern in Qlik Sense Desktop App File (.qvf) Dateien. In Qlik Sense Desktop ist der vordefinierte Pfad:

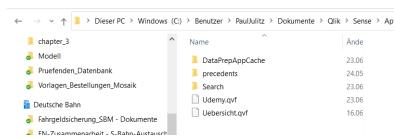


Abbildung 201:

### 2.1.3 Selection

Die QIX Engine agiert über Associationen. Tabellen werden über Schlüssel verbunden. Filter werden auf das gesamte Datenmodell angewandt. Es gibt verschiedenen Farben, die die Auswahl markieren.

Alle möglichen Auswahlen erscheinen weiß (Possible). Wir ein oder mehrere Auswahlen getroffen, so erscheinen diese in grün (Selected). Alle anderen Möglichkeiten erscheinen in einem leichten grau (Alternative). Durch das Assoziativ Modell werden alle andern Filter angepasst. Auch hier erscheinen alle möglichen Auswahlen in weiß (Possible). Die Möglichkeiten die nicht ausgewählt werden können, erscheinen in einem dunklen grau (Excluded)

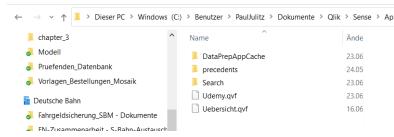


Abbildung 202:

## 2.2 Qlik Key

### 2.2.1 Composite Key

Die Assoziationen zwischen Tabellen kann auch manuell angestoßen werden. Die drei Begrifflichkeit sind Synonyme für den gleichen Sachverhalt.

Am folgenden Beispiel kann gezeigt werden, wie ein Composite Key verwendet wird, eine Kombination von Felder aus einer Tabelle mit einem anderen Feld aus einer anderen Tabelle verbunden werden kann. 1) Zu erst werden die beiden Tabellen geladen.

```

LOAD
 id,
 first_name,
 last_name,
 email,
 gender,
 salary,
 Company
FROM [lib://Qlik-Daten/smart_compound.xlsx]
(Excel, embedded labels, table is Employee);

LOAD
 StudentName,
 CourseDescription,
 CourseDate,
 CourseGrade
FROM [lib://Qlik-Daten/smart_compound.xlsx]
(Excel, embedded labels, table is Course);

```

Abbildung 203:

2) Im Datenmanager kann keine Verbindung automatisch erzeugt werden, da keine Felder übereinstimmen.

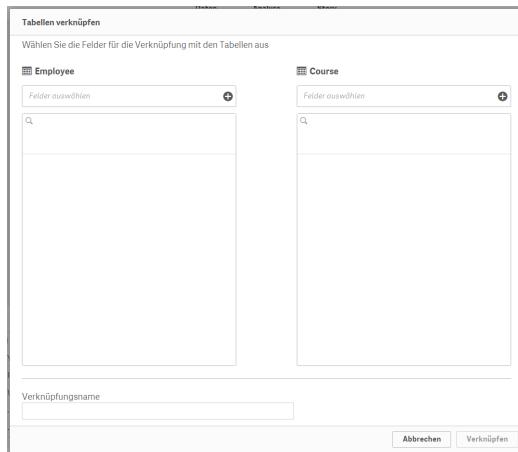


Abbildung 204:

3) Ein Composite Key kann jetzt direkt eingegeben werden. Dabei wird mit Hilfe eines Leerzeichen der [Vorname] und [Nachname] in der Employee Tabelle mit dem [StudentName] in der Course Tabelle verbunden.

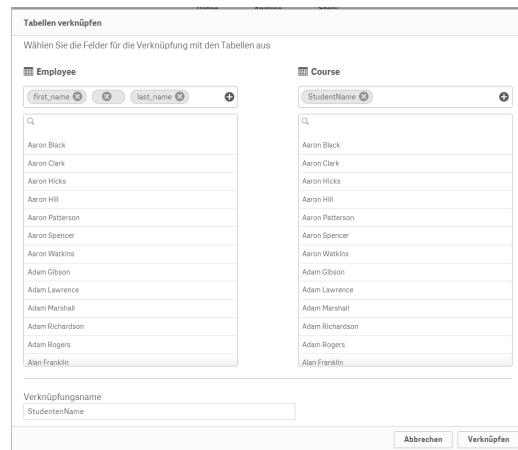


Abbildung 205:

4) Im Datenmanager wird die Verbindung jetzt rot angezeigt.

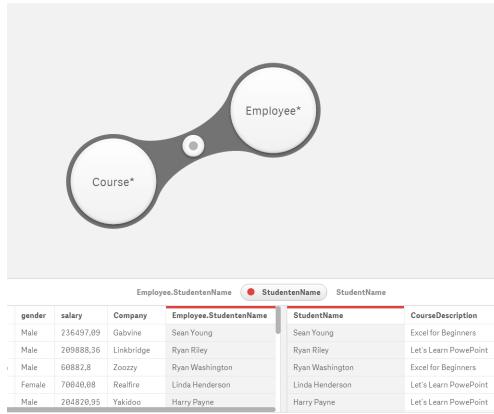


Abbildung 206:

### 5) Sheet Composite Key

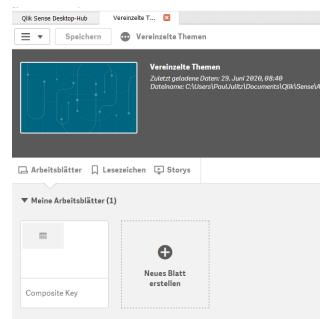


Abbildung 207:

Die App hat jetzt alle relevanten Daten geladen. Als nächstes kann ein Datenblatt angelegt werden. Dabei beinhaltet die App das Datenmodell und die Sheets geben Auskunft über die Visualisierungen.

### 6) Als Objekt der Wahl wird in dem Beispiel eine Tabelle eingefügt.

The screenshot shows the Qlik Sense editor interface. On the left, there is a toolbar with various objects like 'Felder', 'Muster-Elemente', 'Diagramme', and 'Benutzerdefinierte Objekte'. A dropdown menu 'Diagramme' is open, showing options like 'Karte', 'Kombi-Diagr.', '#1 KPI', 'Kreisdiagramm', 'Liniendiagramm', 'Marimekko-Di...', 'Pivottabelle', 'Punktdiagramm', 'Sammelbox', 'Tabelle', 'Text und Bild', 'Verteilungsdiag...', and 'Wasserfalldiag...'. Below this is a button 'Objekte ausblenden'. The main area is titled 'Composite Key' and contains a search bar and a list of items. To the right, there is a panel titled 'Eigenschaften des Arbeitsblatts' with fields for 'Titel' (set to 'Composite Key'), 'Titelformel' (set to '=fx'), 'Beschreibung' (set to '=fx'), 'Gitternetzabstand' (set to 'Weit'), 'Arbeitsblattgröße' (set to 'Reaktionssfähig'), and 'Layout für kleinen Bildschirm'. At the bottom right is a button 'Eigenschaften ausblenden'.

Abbildung 208:

7) Anhand der Verknüpfung [StudentenName] kann das Feld [CourseDescripton] von der Tabelle [Course] mit [Gender] von der Tabelle [Compony] in Verbindung gesetzt werden.

Abbildung 209:

Ein Composite Key kann auch direkt über den Daten Editor erstellt werden. Dabei wird die Spalte direkt über das **Load** Statement abgeändert.

```
[Employee_temp_c3d2d839-a067-47b3-b5d4-1d66c925];
LOAD
[id],
[first_name],
[last_name],
[email],
[gender],
[city],
[company],
[first_name] & '-' & [last_name] AS [StudentenName]
RESIDENT [Employee];
DROP TABLE [Employee];

[Course_temp_b6df4df7-0f9b-7074-75cf-187e3485];
LOAD
[StudentName] AS [StudentenName],
[CourseDescription],
[CourseDate],
[CourseGrade];
RESIDENT [Course];
DROP TABLE [Course];
```

Abbildung 210:

Die Spalte [First Name] und [Last Name] wir verbunden und über **as** als neue Spalte definiert.

## 2.2.2 Synthetic Key

Ein synthetischer Schlüssel wird erstellt, wenn zwei oder mehrere Felder in einer oder mehreren Tabellen gleich sind.

Abbildung 211:

Im gewählten Beispiel sind die Felder **[Region]** und **[Region Code]** in zwei Tabellen gleich. Qlik Sense erstellt eine Synthetic Tabelle **\$Syn 1 Table**. Diese enthält das Feld **\$Syn1**, und **[Region]** und **[Region Code]**.

| Pointer | Value |
|---------|-------|
| 000     | aa    |
| 001     | asdf  |

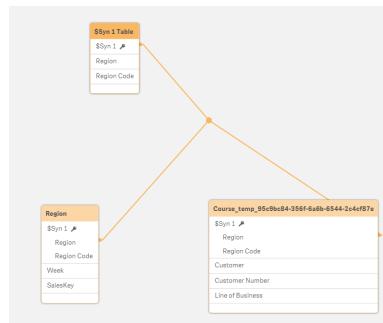


Abbildung 212:

Gleiches wird auch in den anderen Tabellen angelegt, obwohl diese nur über den synthetischen Schlüssel verbunden sind.

Sind Tabellen identisch, so werden diese verbunden und erscheinen im Modell als eine Tabelle.

### 2.2.3 Speicherung

Damit Qlik effizient und schnell Berechnungen durchführen kann, werden alle geladenen Tabellen in zwei Typen gespeichert. Die eigentlichen Datentabellen und die Symboltabellen. Jeder Tabellenspalte wird eine Symboltabelle zugeordnet. In diesen gibt es zwei Spalten, die Pointer und Daten Spalte. Die Daten Spalte greift alle unterschiedlichen Werte einer Spalte auf. Die Anzahl der Zeilen bestimmt die Größe des Pointers. Dieser wird binär dargestellt. Qlik geht hier platzsparend vor. Der Pointer ist nämlich nur so lang - verbraucht nur so viele Stelle, wie benötigt wird. Gibt es vier unterschiedliche Werte, so ist der Pointer drei Stellen lang - drei Bit

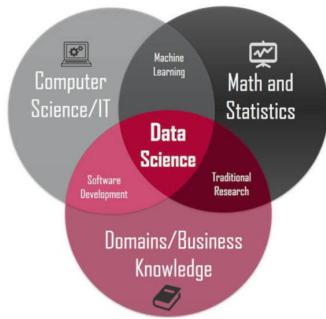
# **Teil VIII**

# **Machine Learning**

# 1 Konzepte

## 1.1 Anwendungsfall

Maschinelles Lernen ist nur ein Teil des Portfolios um Probleme in Unternehmen an Hand von Daten zu lösen.



Die bisher meist verwendeten sind

- Data Analysis: Die traditionelle Bewertung und Auswertung von Geschäftlichen Fragestellungen.
- Data Processing: Software Entwicklung, Transport, Verwaltung und Darstellung von Daten ist ebenso eine bisherige Lösung.

Die Vorhersage Fähigkeit muss zum Geschäftsproblem passen, erst können diese geschaffenen Informationen genutzt werden.

## 2 Unsupervised Learning

## 3 Reinforcement Learning

## 4 Supervised Learning

Überwachtes Lernen setzt

**Teil IX**

**Anhang**

## Anhang A

# Abkürzungsverzeichnis

### **Symbole**

- .qvd** QlikView Data. 1, 166
- .qvf** Qlik Sense Desktop App File. 1, 171

### **A**

- ABB** Arbeitsbereich. 1
- API** Application programming interface. 1
- AWS** Amazon Web Service. 1, 23, 30
- AZM** Arbeit Zeitmanagement. 1

### **B**

- Bahn Enterprise Architecture Management** Bahn Enterprise Architecture Management. 1
- BB** Beförderungsbedingungen. 1
- BCM** Bahn Content Management. 1
- BEG** Bayrische Eisenbahngesellschaft. 1, 17
- BKU** Bürokommunikation Unternehmensweit. 1

### **C**

- cmd** command-line interpreter. 1, 162
- CPP** C++. 1, 138–140
- csv** comma-separated values. 1, 45

### **D**

- DAX** Data Analysis Expressions. 1, 45, 58, 61, 64, 70, 71
- DB** Deutsche Bahn AG Konzern. 1
- DB MC** DB Modular Cloud. 1, 30, 31
- DBS** DB Sicherheit. 1
- DBSE** Externe zivile Prüfer der DB Sicherheit. 1
- DFO** DAX-Filter-Option. 1, 70, 71

### **E**

- EBL** Eisenbahnbetriebsleiter. 1
- EC2** Amazon Elastic Computing Cloud. 1, 26, 30, *Glossar: Amazon Elastic Computing Cloud*
- EfA** Endgeräte für Alle. 1
- ETL** Extract, Transform, Load. 1, 45
- EuLe** Erlös- und Leistungsdatenbank. 1, 25

### **F**

**FGMT** Folgegeneration MT. 1, 180  
**FIS-DV** Fahrgastinformationssystem Datenversorgung. 1  
**FP-Nr** Fahrausweisprüfer Nummer. 1  
**FV** Fernverkehr. 1

## H

**HTML** Hypertext Markup Language. 1, 153  
**HTTP** Hypertext Transfer Protokoll. 1, 150, 151, *Glossar:* Python

## I

**IFM** infoscore Forderungsmanagement GmbH (Aktuell: paigo). 1  
**IoT** Internet of Things. 1  
**iPD** integriertes Planungs- und Dispositionssystem. 1, 13, 14

## J

**JS** JavaScript. 1, 168  
**JSON** JavaScript Object Notation. 1, 150, 151, *Glossar:* JSON  
**json** JavaScript Object Notation. 1, *Glossar:* JSON

## K

**KiN-A** Kundenbetreuer im Nahverkehr. 1

## L

**LV** Leistungsvereinbarung. 1

## M

**MLR** Marketing Leiter Runde. 1  
**MOSAIK** Mobile Strategie im Nahverkehr (vorherige Bezeichnung: Folgegeneration MT (FGMT) Kontroll App für Prüfer). 1  
**MVG** Münchener Verkehrsgesellschaft mbH. 1

## O

**ODBC** Open Database Connectivity (Connection). 1, 166, *Glossar:*  
**OE** Organisationseinheit. 1  
**OfW** Ohne festen Wohnsitz. 1  
**OLT** Online Ticket. 1  
**OOP** Objekt-Orientierte-Programmierung. 1, 76, 89, 121, 122

## P

**P.RM-SBM-M1** Kundeninteraktion. 1  
**P.RM-SBM-M11** Fahrgast- und Verbundmarketing. 1  
**P.RM-SBM-M12** Prüfdienst. 1  
**P.RM-SBM-M13** Fahrgastinformation. 1  
**PDOS** Prüfdienst ohne Sicherheit (Externe zivile Prüfer der DB Sicherheit). 1  
**PKF** Pivot-Koordinaten-Filter. 1, 70, 71, 73, 74

## Q

**QMC** Qlik Management Console. 1, 168

## R

**RDG** Rechtsdienstleistungsgesetz. 1

**RFU** Rahmen des jährlichen Fortbildungsunterrichts. 1

**RiL** Richtlinie. 1

**RIS** Reise- und Informationssystem. 1

**RTA** Reisenden-Ticket-Automat. 1

## S

**S3** Simple Storage Services. 1, 33

**SBF** Schutzbedarfsfeststellung. 1, 31

**SBM** S-Bahn München. 1

**SDVS** Securitas Deutsche Verkehrssicherheit und Service GmbH (Externe Prüfer von Securitas). 1

**SECU** Externe Prüfer von Securitas. 1

**SEV** Schienenersatzverkehr. 1

**SFTP** Secure File Transfer Protocol. 1, 33

**SQL** Structured Query Language. 1, 113, 166, 183, *Glossar: SQL*

## T

**TV** Tarifverstoß (ursprünglich Vertragsstrafe). 1

## U

**UI** User Interface. 1

**URL** Uniform Resource Locator. 1, 150, 151, *Glossar: URL*

**UX** User Experience. 1

## V

**VAS** Vertriebs- und Abrechnungssoftware. 1

**VB** Verkehrsvertrieb (VB). 1

**VBA** Visual Basic Application. 1, 76–78, 84, 87, 90–92, 140

**VCS** Version Controll System. 1, 156, 157, 160, 182

**VDV** Verband Deutscher Verkehrsunternehmen. 1

## X

**xlsx** Microsoft Excel Open XML Spreedsheet with Macros. 1

**xlsx** Microsoft Excel Open XML Spreedsheet. 1, 45

## Z

**ZVT** Zahlung-Verkehr-Terminal. 1

## ÖPNV

Öffentlicher Personennahverkehr. 1

# Anhang B

## Glossar

### A

**Amazon Elastic Computing Cloud** Amazon Elastic Compute Cloud (Amazon EC2) bietet eine skalierbare Rechenkapazität in der Amazon-Web-Services(AWS)-Cloud. Amazon EC2 beseitigt die Notwendigkeit, im Voraus in Hardware investieren zu müssen. Daher können Sie Anwendungen schneller entwickeln und bereitstellen. Sie können Amazon EC2 verwenden, um so viele oder so wenige virtuelle Server zu starten, wie Sie benötigen, Sicherheit und Netzwerk zu konfigurieren und den Speicher zu verwalten. Amazon EC2 können Sie auf- oder abwärts skalieren, um auf geänderte Anforderungen oder Datenverkehrsspitzen zu reagieren. Dies reduziert die Notwendigkeit, den Datenverkehr vorauszusagen. . 1

**Application Programming Interface** Ein Programmschnittstelle ist eine Interaktionsmöglichkeit, die ein Programm anderen Programmen bietet auf bestimmte Teile der Software und oder Daten zuzugreifen. . 1

### B

**Berechnende Spalte** Diese ergänzen Tabellen, welche im Datenmodell eingefügt sind. Dabei werden Daten erzeugt und auch im Arbeitsspeicher geladen. Diese Spalten lassen keine Aggregations zu und beziehen sich ausschließlich auf jede Zeile in einer Tabelle. . 1, 61

### G

**Git** Git is a free and open-source distribution VCS. It tracks changes in source code during software development. . 1, 156, 157, 159, 163

**Git Repository** Repositories in GIT contain a collection of files of various different versions of a Project. These files are imported from the repository into the local server of the user for further updatons and modifications in the content of the file. A VCS or the Version Control System is used to create these versions and store them in a specific place termed as a repository **Geeks.2020**. . 1, 158

**Git Repository** Repositories in GIT contain a collection of files of various different versions of a Project. These files are imported from the repository into the local server of the user for further updatons and modifications in the content of the file. A VCS or the Version Control System is used to create these versions and store them in a specific place termed as a repository **Geeks.2020**. . 1, 157, 159, 160, 164

### I

**Intellisense** Dies ist eine Anwendung zur Auto-Vervollständigung . 1, 78, 84

### J

**JSON** JavaScript Object Notation strukturiert Daten. Dieses Datenformat hat sich in der Webentwicklung verbreitet. Bei Anfragen von Rest-API kann man ein JSON Format erwarten. Ebenso finden JSON-Formate in Log und Confi-Datein ihren Nutzen. Es gibt 6 Datenformate, welche in einer JSON Datei abgespeichert werden können. String, Boolom, Integer, null, Array und Object . 1

## M

- M Formula Language** Power Query nutzt die Sprache M. Diese ist eine funktional Sprache ähnlich zu F. Es können mit den Ausdrücken Daten gefiltert und transformiert werden. . 1, 108
- Measure** Dies ist eine berechnenden DAX-Funktion. Dies ist zu unterscheiden zu den *Berechneten Feld* Funktionen, die für Tabellen die nicht im Datenmodell enthalten sind, zu nutzen sind. . 1, 61, 66, 67, 73, 74

## O

- Open Database Connectivity (or Connector)** Es handelt sich hierbei um eine API Verbindung zu einem DBMS. Das Ziel ist, dass eine Datenverbindung zu Datenbanksystem hergestellt werden kann, die unterschiedlicher Natur und oder auf unterschiedlichen Plattformen laufen. . 1

## P

- Python** The Hypertext Transfer Protocol is an application layer protocol for distributed, collaborative, hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web, where hypertext documents include hyperlinks to other resources that the user can easily access, for example by a mouse click or by tapping the screen in a web browser. [Wikipedia o. D.](#) . 1
- Python** Python is an interpreted, high-level and general-purpose programming language. Python's design philosophy emphasizes code readability with its notable use of significant whitespace. [Wikipedia o. D.](#) . 1, 136, 137, 139, 140, 142

## Q

- Qlik Sense** Qlik Sense ist eine Business Analytics Software, welche die QIX-Engine Technologie nutzt. . 1, 167, 168, 171, 174

## S

- Shell** Eine Shell ist ein Befehl Interpretator (Commandline Interpretar). Diese gibt Zugang zum Betriebssystem. Im Regelfall wird dieser über ein command-line or graphical user interface gesteuert. . 1, 156
- SQL** SQL ist eine Sprache welche strukturierte Abfragen an Datenbanken stellt. Diese Sprache wird von Datenbanken wie MySQL, Oracle, Microsoft SQL Server, etc. verwendet. . 1
- SQLite** SQLite ist ein relationales Datenbanksystem. . 1, 115

## U

- URL** A Uniform Resource Locator (URL), colloquially termed a web address, is a reference to a web resource that specifies its location on a computer network and a mechanism for retrieving it. A URL is a specific type of Uniform Resource Identifier (URI), although many people use the two terms interchangeably. URLs occur most commonly to reference web pages (http), but are also used for file transfer (ftp), email (mailto), database access (JDBC), and many other applications. [Wikipedia o. D.](#) . 1

## Anhang C

### Literatur

*stackoverflow* (o. D.). URL: [https://stackoverflow.com/questions/45034549/difference-between-git-gui-git-bash-git-cmd#:~:text=\(Command%20Line%20prompt\)%20is%20the,cmd%20to%20run%20git%20commands..](https://stackoverflow.com/questions/45034549/difference-between-git-gui-git-bash-git-cmd#:~:text=(Command%20Line%20prompt)%20is%20the,cmd%20to%20run%20git%20commands..)

*Wikipedia* (o. D.). URL: [https://en.wikipedia.org/wiki/Hypertext\\_Transfer\\_Protocol](https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol).

*Wikipedia* (o. D.). URL: [https://en.wikipedia.org/wiki/Python\\_\(programming\\_language\)](https://en.wikipedia.org/wiki/Python_(programming_language)).

*Wikipedia* (o. D.). URL: <https://en.wikipedia.org/wiki/URL>.