

Übung

Andreas Buckenhofer

Mercedes-Benz Tech Innovation



Über mich



Andreas Buckenhofer

Lead Expert Vehicle Data Platforms

Seit 2009 bei Mercedes-Benz Tech Innovation
Team: Vehicle Data Platforms
Business Unit: Vehicle Platforms

Contact/Connect



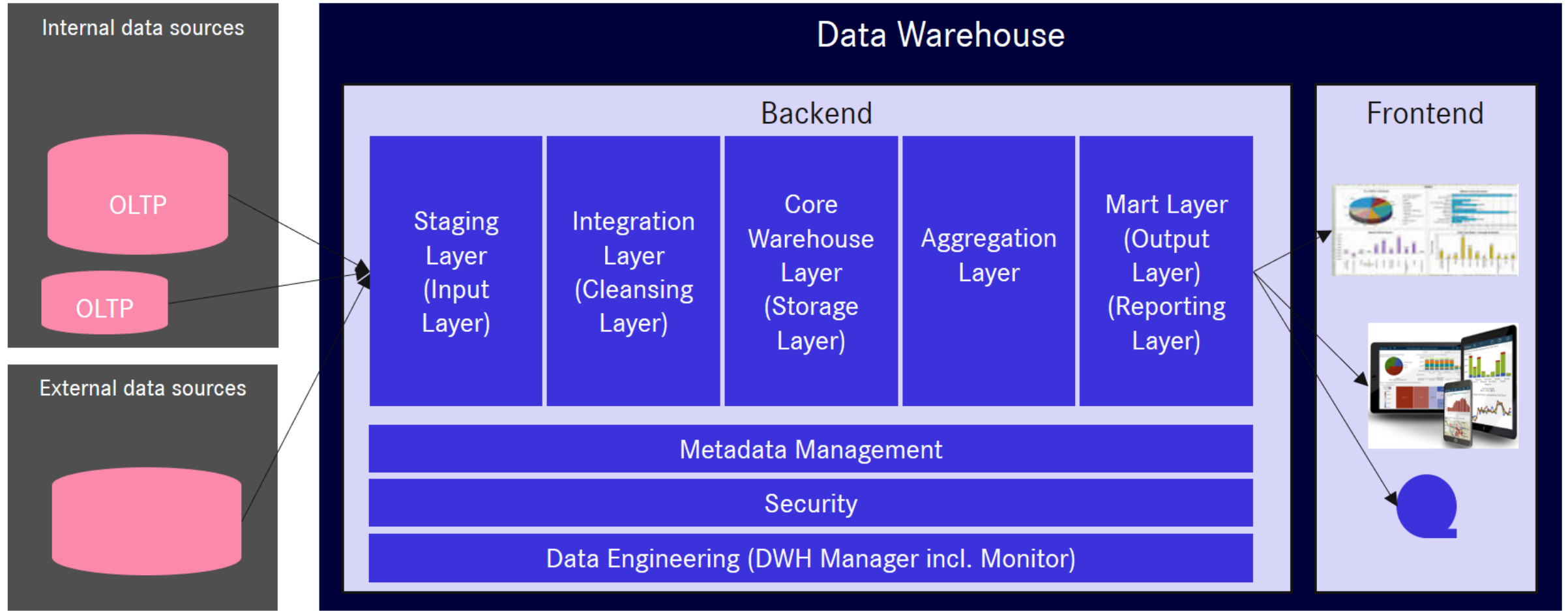
- Über 20 Jahre Erfahrung mit daten-intensiven Anwendungen
- Über 20 Jahre Erfahrung mit Datenbank Technologien
- Internationale Projekterfahrung

- Oracle [ACE Associate](#)
- [DOAG](#) Delegierter und DBC Mitglied
- Dozent an der [DHBW](#)
- Zertifiz. Data Vault Practitioner 2.0
- Zertifiz. Oracle Professional
- Zertifiz. IBM Big Data Architect

Change Log

Date	Changes

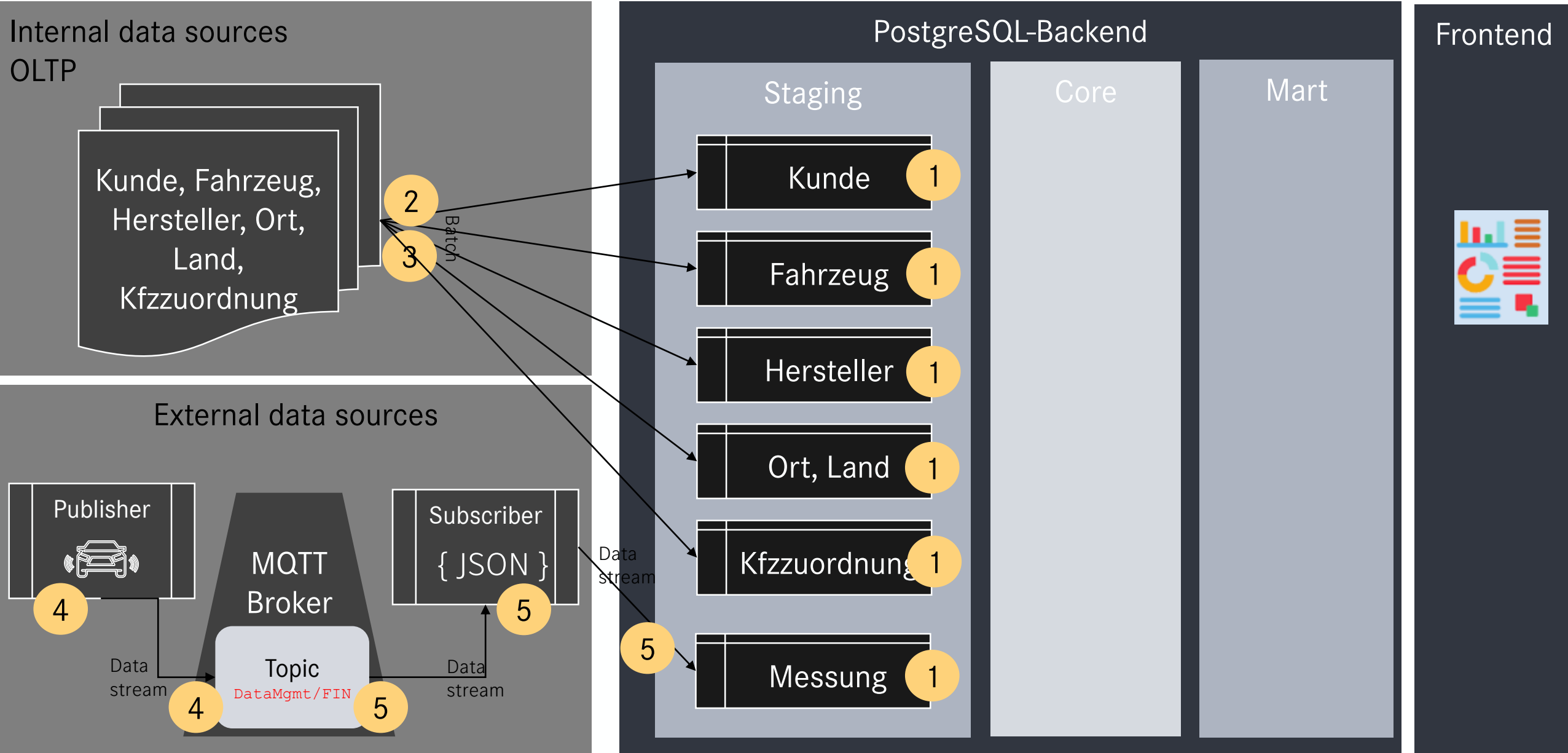
Standard DWH architecture



Fallstudie

- Erstellen Sie ein DWH aus internen und externen Daten
 - Interne Daten: Kunden, Fahrzeuge, u.ä
 - Externe Daten: Messdaten / Fahrzeugdaten
- Komponenten
 - RDBMS (PostgreSQL) zum Aufbau des DWH
 - Python, Javascript, SQL für Datenerzeugung, Datentransfer, Datentransformation
 - Batch: csv-Dateien (hier: SQL-Skripte) für interne Daten
 - Stream: Sensordaten, die über MQTT (Message Queue Telemetry Transport) übertragen werden, für externe Daten

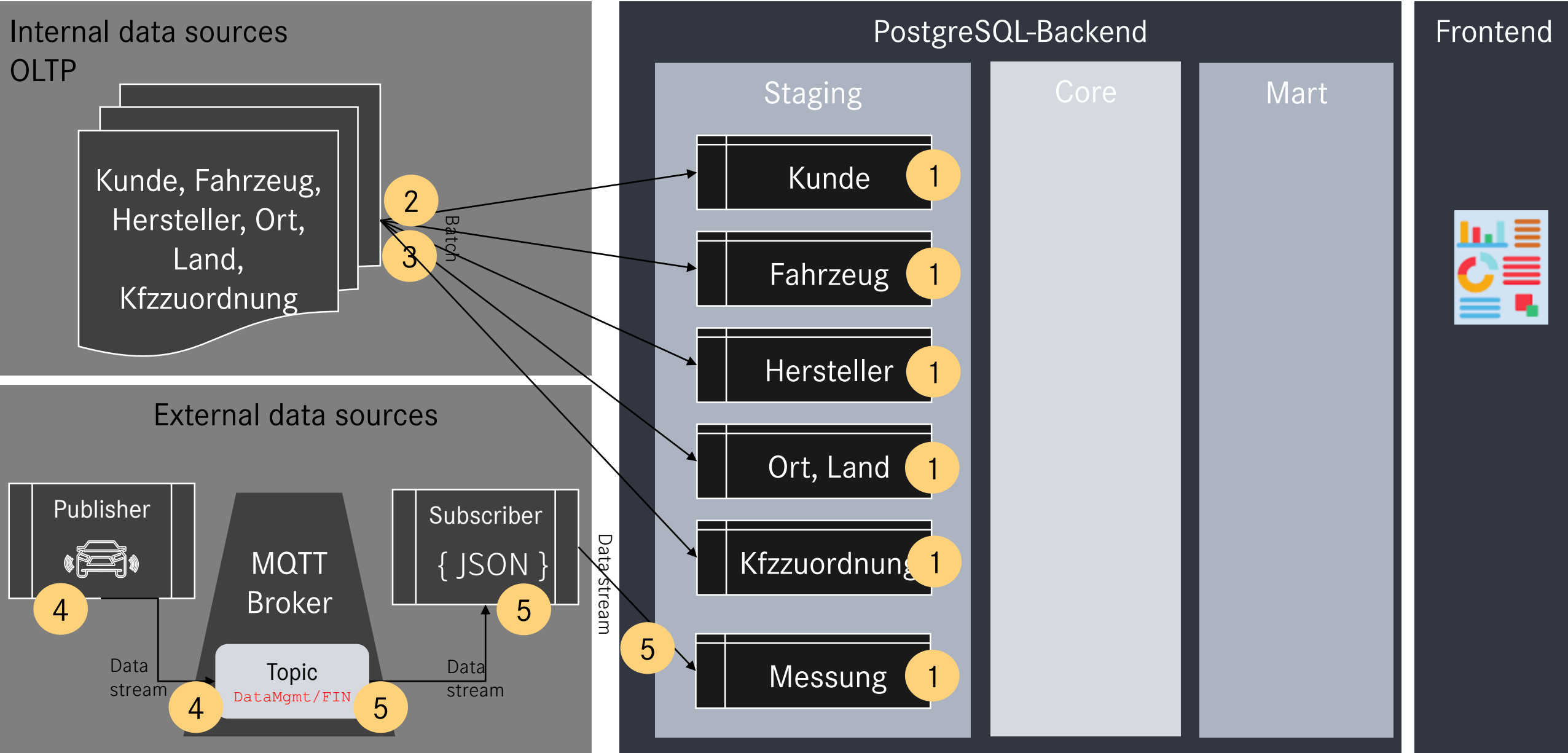
Architektur Fallstudie



Übung 1

- 1 Erstellen Sie die Tabellen im staging-Layer
- 2 Laden Sie die Daten in den staging Layer
- 3 Legen Sie Ihre eigenen Daten an und laden Sie diese in den staging-Layer
- 4 Erstellen Sie einen MQTT Publisher
- 5 Erstellen Sie einen MQTT Subscriber

Architektur Fallstudie



Tabellen im staging Layer erstellen

Führen Sie die zur Verfügung gestellten Skripte aus:

- 01_create_staging.sql

Anmerkung: Die Skripte enthalten constraints.

Üblicherweise werden keine constraints im staging-Layer angelegt, um alle Daten aufnehmen zu können – auch fehlerhafte. Für die Übung werden jedoch einige Constraints verwendet, um die erzeugten Daten zu validieren.

```

C:\Eingabeaufforderung - docker exec -it columnarpostgresql bash
DROP SCHEMA
postgres=#
postgres=# create schema staging;
CREATE SCHEMA
postgres=#
postgres=# create table staging.hersteller (
postgres=#     hersteller_code char(3) not null
postgres=#     , hersteller varchar(200) not null
postgres=#     , erstellt_am TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
postgres=#     , quelle varchar(20) not null
postgres=#     , CONSTRAINT pk_hersteller PRIMARY KEY(hersteller_code)
postgres=# );
CREATE TABLE
postgres=#
postgres=# create table staging.land (
postgres=#     land_id integer not null
postgres=#     , land varchar(200) not null
postgres=#     , erstellt_am TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
postgres=#     , quelle varchar(20) not null
postgres=#     , CONSTRAINT pk_land PRIMARY KEY(land_id)
postgres=# );
CREATE TABLE
postgres=#
postgres=# create table staging.ort (
postgres=#     ort_id integer not null
postgres=#     , ort varchar(200) not null
postgres=#     , land_id integer not null
postgres=#     , erstellt_am TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
postgres=#     , quelle varchar(20) not null
postgres=#     , CONSTRAINT pk_ort PRIMARY KEY(ort_id)
postgres=#     --, CONSTRAINT fk_o_land FOREIGN KEY(land_id) REFERENCES staging.land(land_id)
postgres=# );
CREATE TABLE
postgres=#
postgres=# create table staging.kunde (
postgres=#     kunde_id integer not null
postgres=#     , vorname varchar(200) not null
postgres=#     , nachname varchar(200) not null
postgres=#     , anrede varchar(20) CHECK (anrede in ('Herr', 'Frau'))
postgres=#     , geschlecht varchar(20) CHECK (geschlecht in ('männlich', 'weiblich'))
postgres=#     , geburtsdatum date
postgres=#     , wohnort integer not null
postgres=#     , erstellt_am TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
postgres=#     , quelle varchar(20) not null
postgres=#     , CONSTRAINT pk_kunde PRIMARY KEY(kunde_id)
postgres=#     , CONSTRAINT fk_k_ort FOREIGN KEY(wohnort) REFERENCES staging.ort(ort_id)
postgres=# );
CREATE TABLE
postgres=#
postgres=# create table staging.fahrzeug (
postgres=#     fin char(17) not null
postgres=#     , hersteller_code char(3) not null
postgres=#     , kunde_id integer not null
postgres=#     , baujahr integer not null
postgres=#     , modell varchar(200) not null
postgres=#     , erstellt_am TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
postgres=#     , quelle varchar(20) not null
postgres=#     , CONSTRAINT pk_fahrzeug PRIMARY KEY(fin)
postgres=#     , CONSTRAINT fk_f_hersteller FOREIGN KEY(hersteller_code) REFERENCES staging.hersteller(hersteller_code)
postgres=#     , CONSTRAINT fk_f_kunde FOREIGN KEY(kunde_id) REFERENCES staging.kunde(kunde_id)
postgres=# );
CREATE TABLE
postgres=#
postgres=# create table staging.kfzzuordnung (
postgres=#     kfz_kennzeichen char(17) not null
postgres=#     , fin char(17) not null
postgres=#     , erstellt_am TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
postgres=#     , quelle varchar(20) not null
postgres=#     , CONSTRAINT pk_kfzzuordnung PRIMARY KEY(kfz_kennzeichen)
postgres=#     , CONSTRAINT fk_kfz_fahrzeug FOREIGN KEY(fin) REFERENCES staging.fahrzeug(fin)
postgres=# );
CREATE TABLE

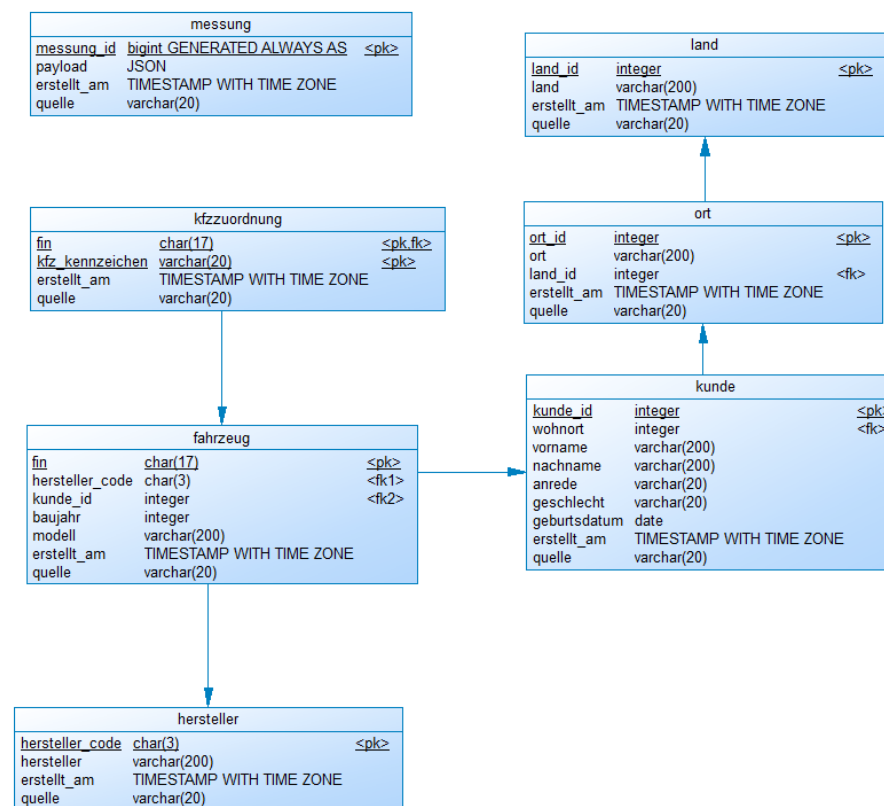
```

Wiederherstellen
Verschieben
Größe ändern
Minimieren
Maximieren
Schließen

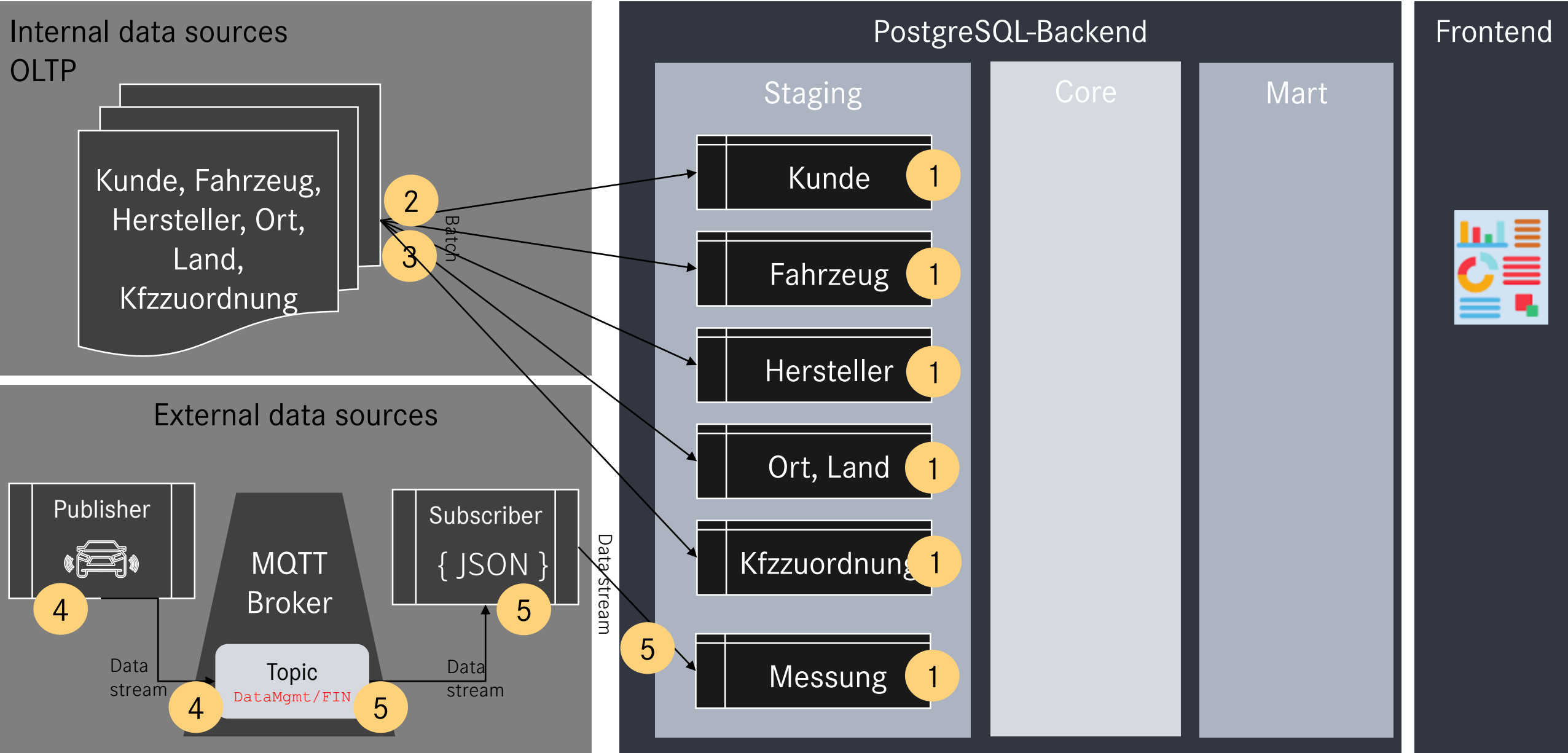
Bearbeiten
Standardwerte
Eigenschaften

Markieren
Kopieren
Einfügen
Alle auswählen
Bildlauf
Suchen...

STRG+M
EINGABE
STRG+V
STRG+A
STRG+F



Architektur Fallstudie



Daten in den staging Layer laden

Führen Sie die zur Verfügung gestellten Skripte aus:

- 02_load_staging.sql

Anmerkung: üblicherweise werden csv-Dateien in den staging-Layer geladen. In der Übung werden insert-Befehle genutzt. csv-Dateien können mittels Bulk-Befehlen geladen werden für eine deutlich bessere Performanz.

Außerdem enthalten die Tabellen im staging-layer in der Übung bereits Datentypen. Sonst werden häufig varchar-Datentypen verwendet, um ggf. fehlerhafte Daten aus den csv-Dateien laden zu können.

```
insert into staging.hersteller (hersteller_code, hersteller, quelle)
values
  ('WDD', 'Mercedes-Benz', 'WMI database')
, ('WVW', 'Volkswagen', 'WMI database')
, ('WBA', 'BMW', 'WMI database')
, ('WB1', 'BMW Motorrad', 'WMI database')
, ('1FM', 'Ford', 'WMI database')
, ('ZFF', 'Ferrari', 'WMI database')
, ('WMA', 'MAN', 'WMI database')
, ('WAU', 'Audi', 'WMI database')
, ('WPO', 'Porsche', 'WMI database')
, ('SCC', 'Lotus', 'WMI database')
, ('WOL', 'Opel', 'WMI database')
, ('VF1', 'Renault', 'WMI database')
, ('SNT', 'Sachsenring Automobilwerke Zwickau GmbH', 'WMI database')
;

insert into staging.land (land_id, land, quelle) values
  (101, 'Deutschland', 'Geo System')
, (102, 'Österreich', 'Geo System')
, (103, 'Mittelerde', 'Geo System')
, (104, 'Schweiz', 'Geo System')
, (105, 'China', 'Geo System')
;

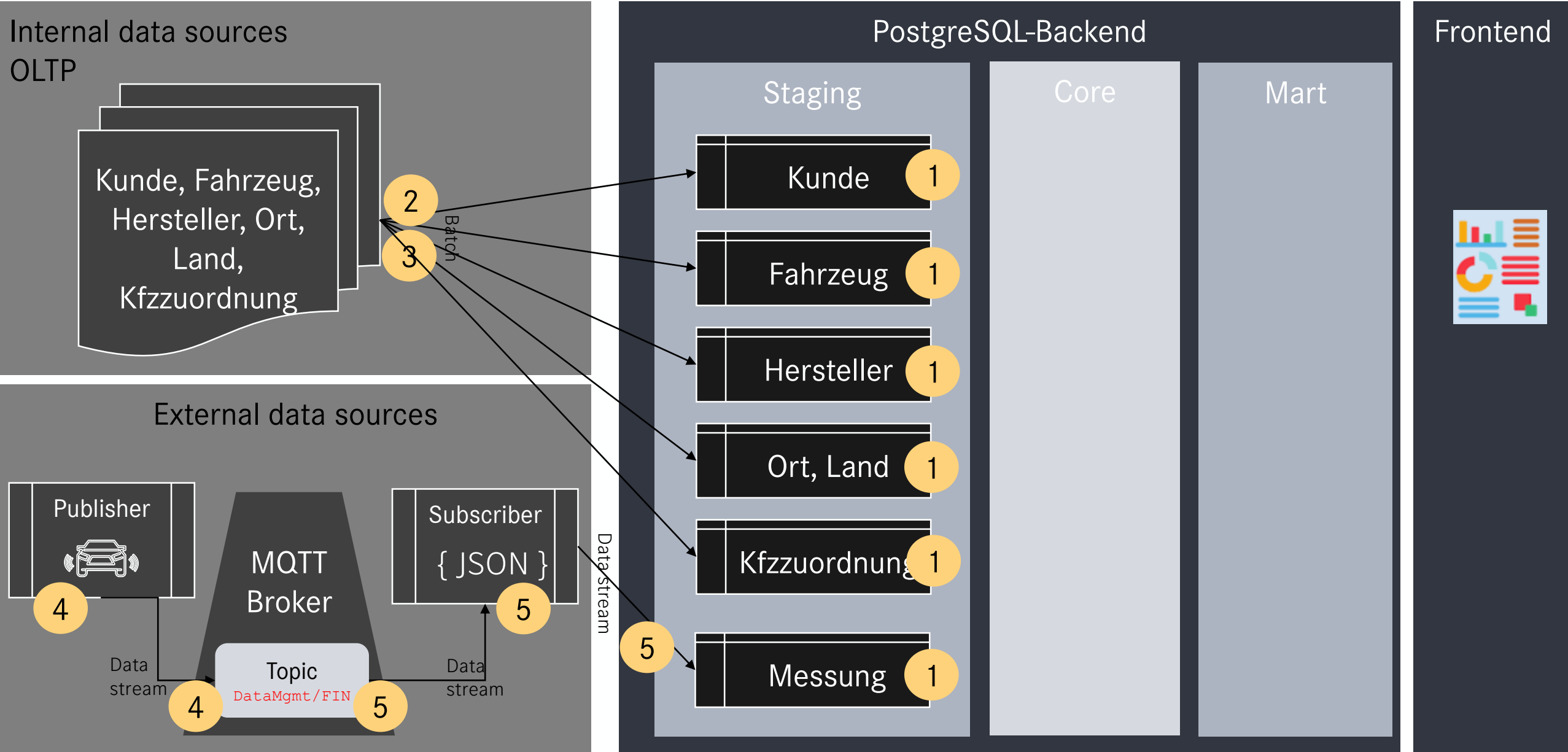
insert into staging.ort (ort_id, ort, land_id, quelle) values
  (1, 'Stuttgart', 101, 'Geo System')
, (2, 'München', 101, 'Geo System')
, (3, 'Frankfurt', 101, 'Geo System')
, (4, 'Türmli', 104, 'Geo System')
, (5, 'Xian', 105, 'Geo System')
, (6, 'Peking', 105, 'Geo System')
, (7, 'Valmar', 103, 'Geo System')
, (8, 'Númenor', 103, 'Geo System')
, (9, 'Wien', 102, 'Geo System')
, (10, 'Rot a.d. Rot', 100, 'Geo System')
;

insert into staging.kunde (kunde_id, vorname, nachname, anrede, geschlecht, ge
values (171893, 'Kevin', 'Minion', 'Herr', 'männlich', to_date('12.02.1990',

insert into staging.fahrzeug (fin, kunde_id, baujahr, modell, quelle)
values ('SNTU411STM9032150', 171893, 1985, 'Trabant 601 de luxe', 'Fahrzeug

insert into staging.kfzuzuordnung (fin, kfz_kennzeichen, quelle)
values ('SNTU411STM9032150', 'UL-DV 111', 'Fahrzeug DB');
```

Architektur Fallstudie



Eigene Daten anlegen und in den staging Layer laden

Legen Sie jeweils einen Datensatz für

- einen Kunden
- ein Fahrzeug (die ersten drei Zeichen der FIN müssen in der Tabelle hersteller existieren)
- eine Fahrzeugzuordnung

neu an.

Erstellen Sie SQL Befehle nach dem unten aufgeführten Muster. Nutzen Sie eigene Daten.

Führen Sie die insert Befehle aus.

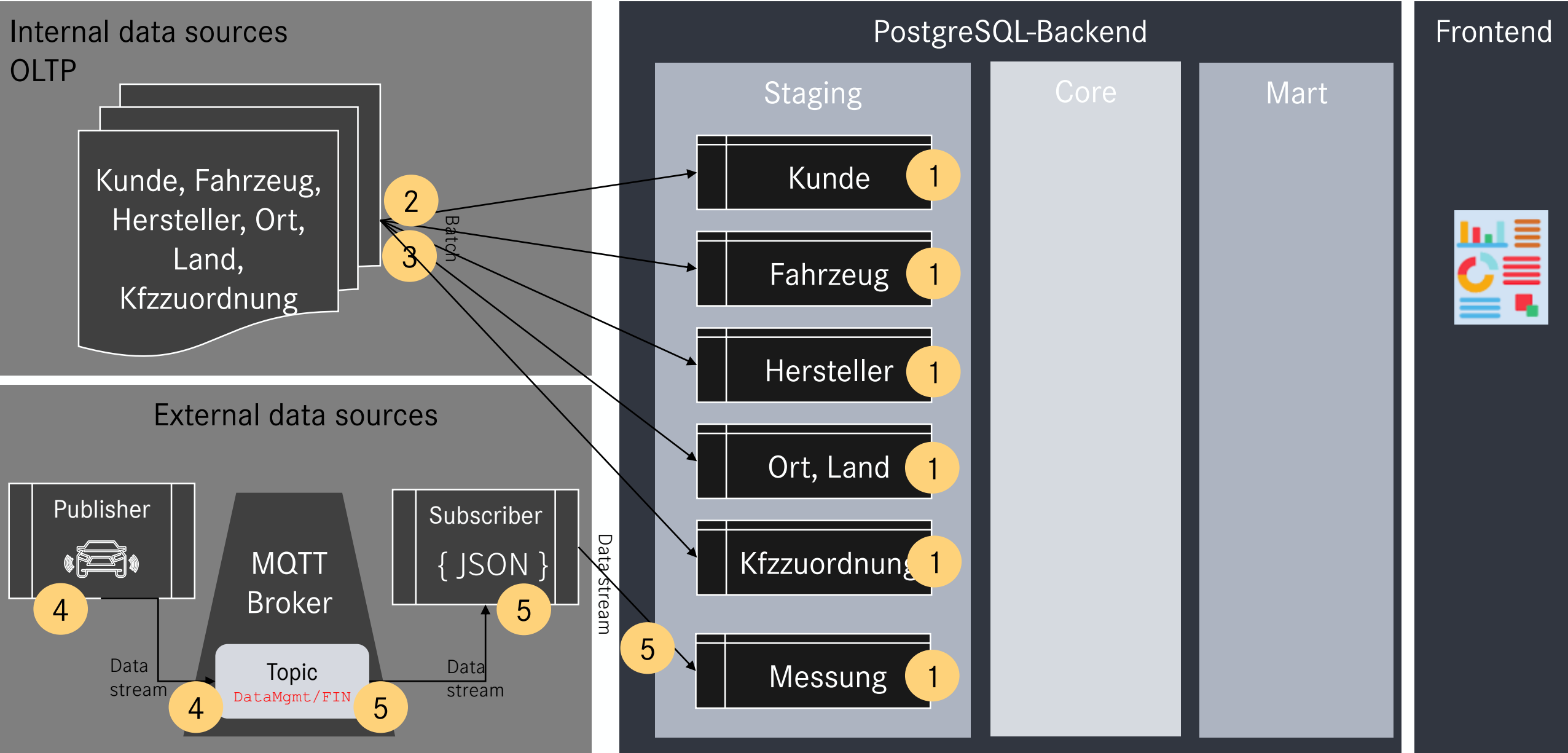
Laden sie die Befehle im Skript 03_staging.sql in moodle hoch!

```
insert into staging.kunde (kunde_id, vorname, nachname, anrede, geschlecht, geburtsdatum, wohnort, quelle)
values (171893, 'Kevin', 'Minion', 'Herr', 'männlich', to_date('12.02.1990', 'DD.MM.YYYY'), 7, 'CRM');

insert into staging.fahrzeug (fin, kunde_id, baujahr, modell, quelle)
values ('SNTU411STM9032150', 171893, 1985, 'Trabant 601 de luxe', 'Fahrzeug DB');

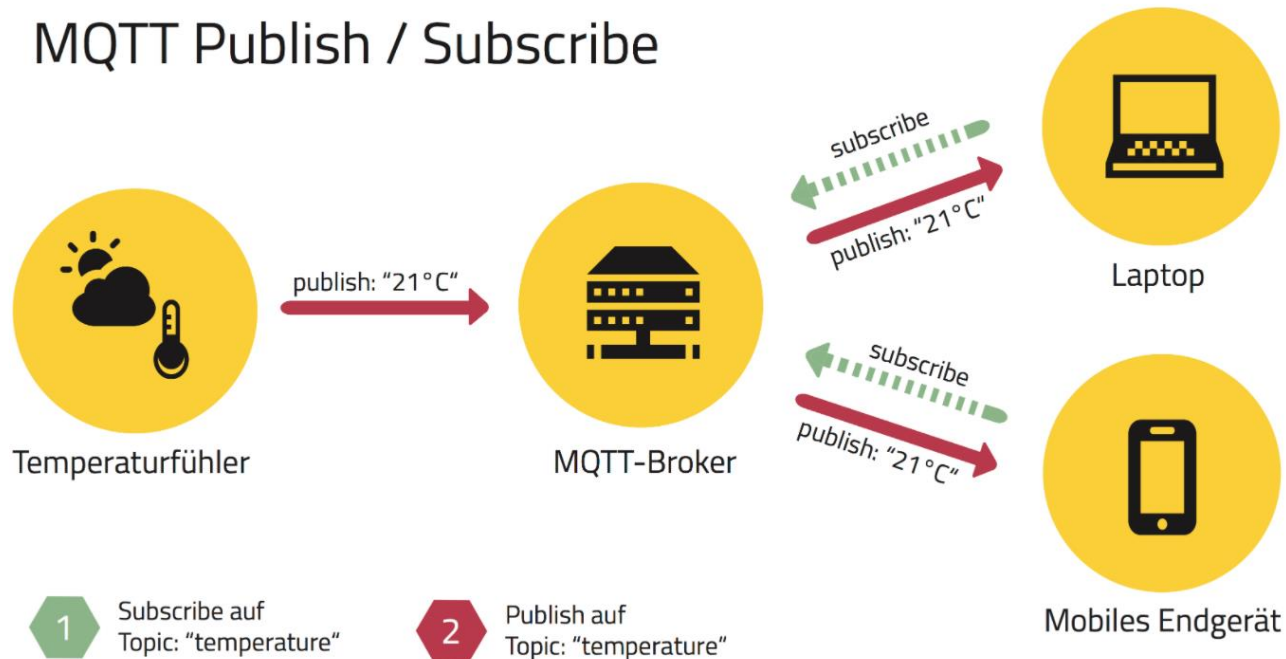
insert into staging.kfzzuordnung (fin, kfz_kennzeichen, quelle)
values ('SNTU411STM9032150', 'UL-DV 111', 'Fahrzeug DB');
```

Architektur Fallstudie



MQTT Protokoll im Internet der Dinge

MQTT Publish / Subscribe



Leichtgewichtiges Protokoll für den Transport von Nachrichten über ein unzuverlässiges Netzwerk

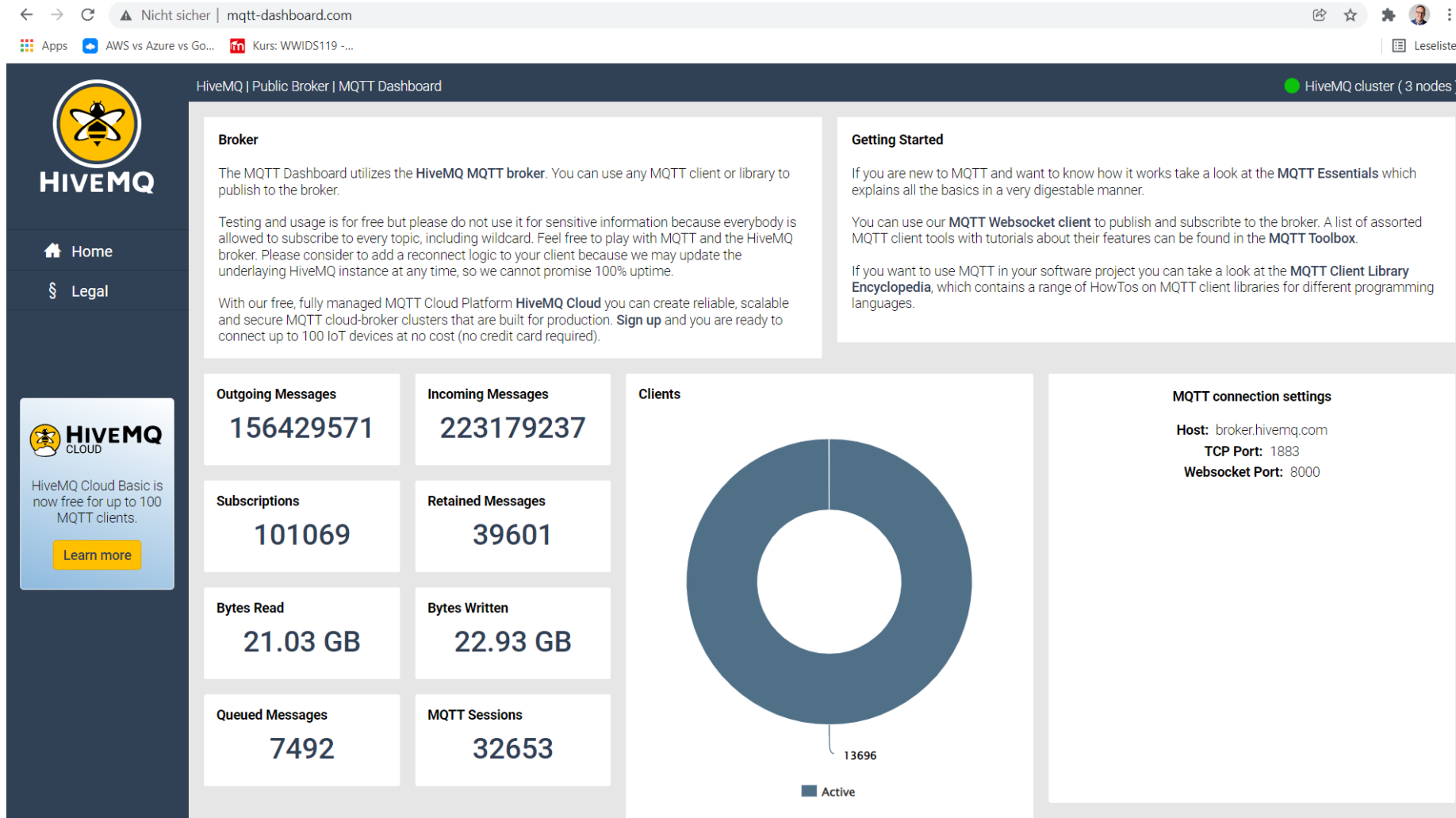
MQTT Broker

- Aufgaben: Speicherung, Verwaltung und Verteilung von Informationen (Topics)
- Ausfallsicherer Betrieb notwendig

MQTT Client

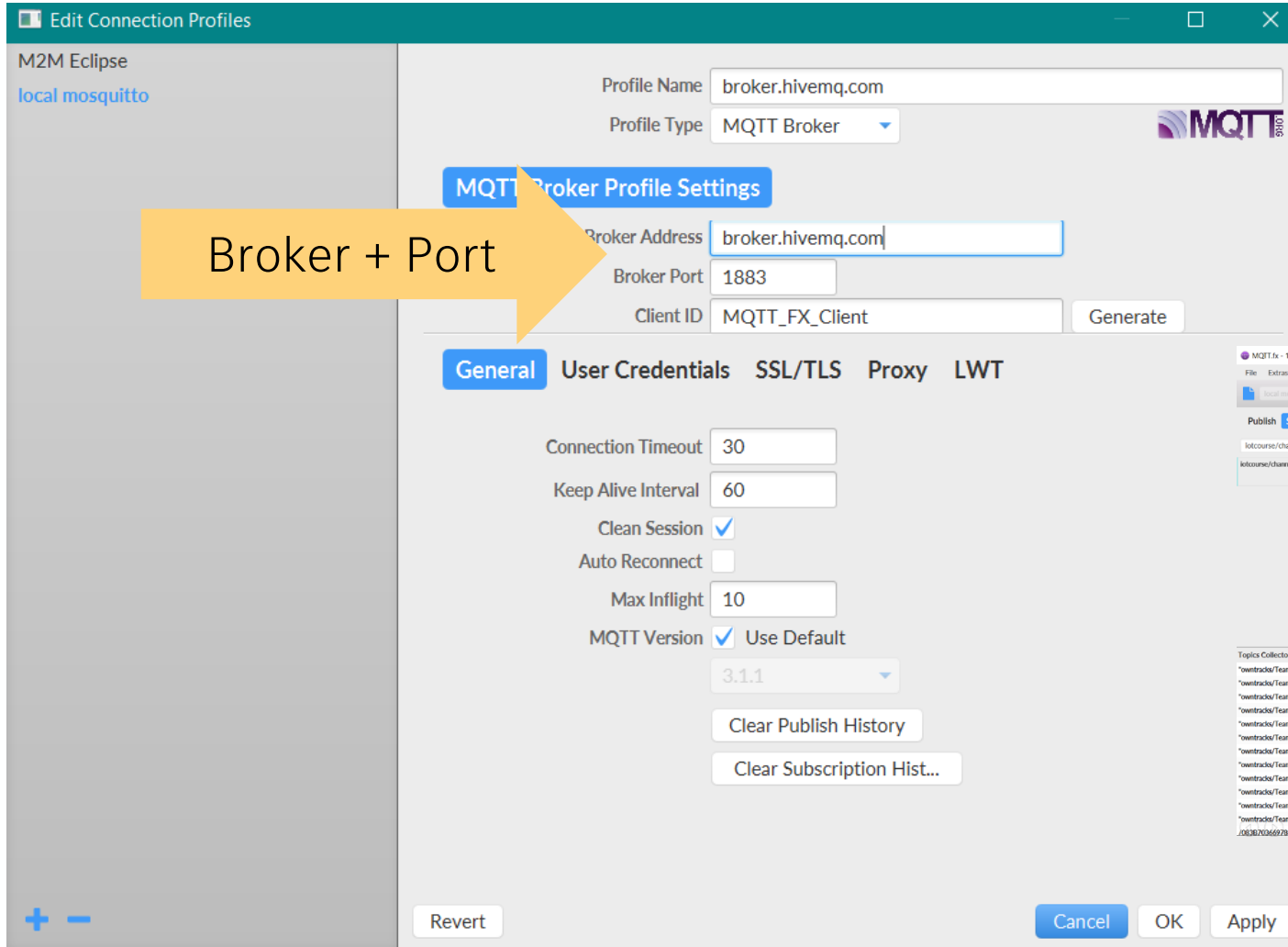
- Publisher: sendet Nachrichten an ein Topic
- Subscriber: empfängt/abonniert Nachrichten

Öffentlicher MQTT Broker zum Testen



Host:
broker.hivemq.com
Port: 1883

MQTT.fx (optional)

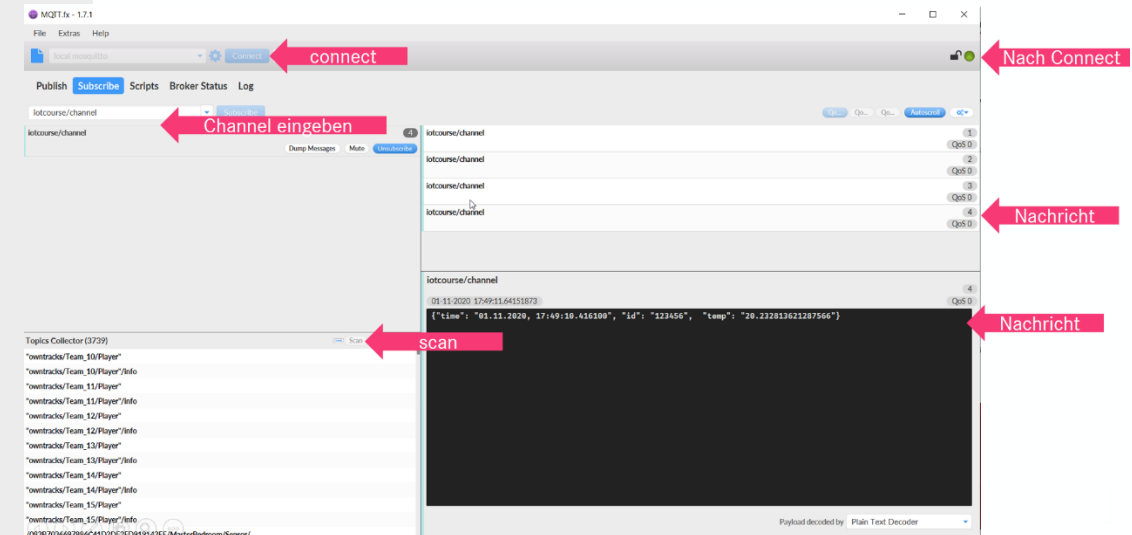


Optional, kann für Tests hilfreich sein:

MQTT.fx:

<https://web.archive.org/web/20220303040559/http://www.jensd.de/apps/mqttfx/1.7.1/>

(neue Versionen lizenzpflichtig)



MQTT Client - Beispiele

Publisher

```
import paho.mqtt.client as mqtt
broker_address="broker.hivemq.com"
client = mqtt.Client("P1", clean_session=False) #use your own unique ID
client.connect(broker_address)
client.publish("DataMgmt/FIN", <JSON>, qos=1)
```

Beispiele und Anleitungen, z.B.:

<http://www.steves-internet-guide.com/into-mqtt-python-client/>
<https://www.emqx.com/en/blog/mqtt-js-tutorial>

pip install paho-mqtt

Subscriber

```
import paho.mqtt.client as mqtt
broker_address="broker.hivemq.com"

def on_message(client, userdata, message):
<JSON Message in DB schreiben>

client = mqtt.Client("S1", clean_session=False) #use your own unique ID
client.on_message=on_message
client.connect(broker_address)
client.subscribe("DataMgmt/FIN", qos=1)
...
```

Persistente Sitzungen

- Client verbindet sich mit MQTT Broker und erstellt abonniert Topics (subscriptions)
- Falls Verbindung unterbrochen wird, gehen Nachrichten verloren
- Der Verlust von Nachrichten kann vermieden werden durch persistente Sitzungen durch Setzen von cleanSession flag beim Verbindungsaufbau
 - True: Client möchte keine persistente Sitzung
 - False: Client fordert persistente Sitzung an

Qualitätsstufen von Nachrichten

- QoS=0: eine Nachricht wird maximal einmal geliefert
- QoS=1: mindestens einmal geliefert
- QoS=2: Garantie, dass eine Nachricht "exakt einmal geliefert" wurde

MQTT Publisher erstellen

Erstellen Sie einen MQTT-Publisher. Erzeugen Sie Daten der Form

```
{  
  „fin“: "SNTU411STM9032150",  
  "zeit": "01.01.2020 15:34:05.123",  
  "geschwindigkeit": 60,  
  "ort": 10  
}
```

```
{  
  „fin“: "<FIN>",  
  "zeit": "<DD.MM.YYYY HH24:MI:SS.MS>",  
  "geschwindigkeit": <Geschwindigkeit>,  
  "ort": <Ort-ID>  
}
```

Schreiben Sie alle 5 Sekunden Daten in das Topic DataMgmt/FIN

- FIN: die von Ihnen festgelegte FIN (siehe Folien zu „Daten in staging Layer laden“)
- Zeit: aktuelle Systemzeit (Millisekunden oder Mikrosekunden)
- Geschwindigkeit: zufälliger Integer-Wert im Bereich 0 bis 200
- Ort: ID (Feld ort_id aus der Tabelle staging.ort; verwendeter Wert darf konstant sein)

Achten Sie auf valide JSON-Dokumente! (JSON überprüfen: <https://jsonformatter.curiousconcept.com/>)

Sie können optional weitere Key-Value-Paare hinzufügen.

MQTT Subscriber erstellen

Erstellen Sie einen MQTT-Subscriber

- Der Subscriber liest Nachrichten aus dem MQTT Topic `DataMgmt/FIN` aus (d.h. nicht nur Ihre eigenen!)
- Die Nachrichten werden in der PostgreSQL Datenbank in die Tabelle `staging.messung` geschrieben

Verbindungsdaten zur Datenbank sind üblicherweise:

```
pip install psycpg2-binary
```

```
import psycpg2
```

```
conn = psycpg2.connect("dbname='postgres' user='postgres' password='...' host='localhost' port='5432'")
```

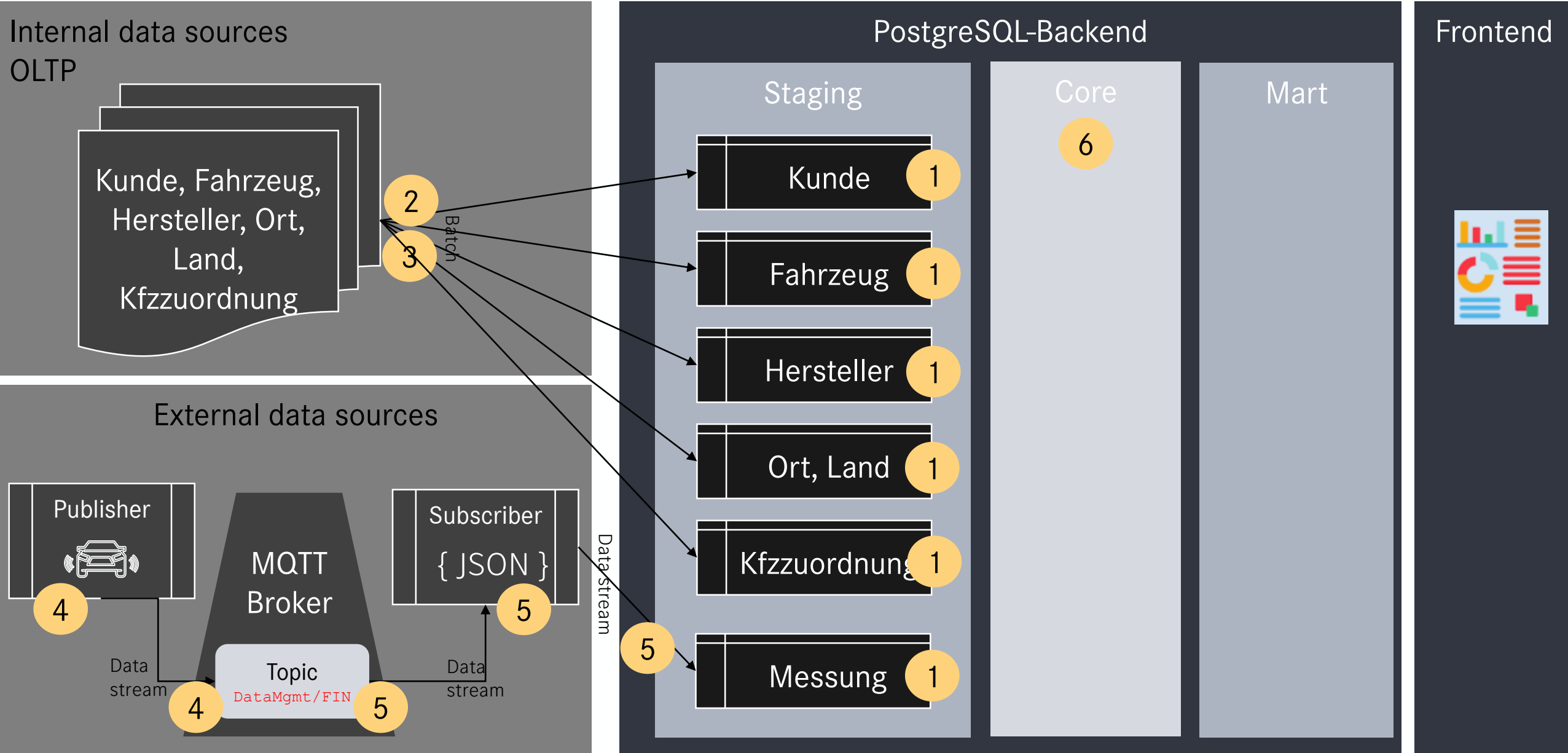
Abgabe

- Abgabe der Skripte über moodle oder github (Abgabe in moodle: txt-Datei mit dem github-Link)

3 Abgabe eines sql-Skripts 03_staging.sql

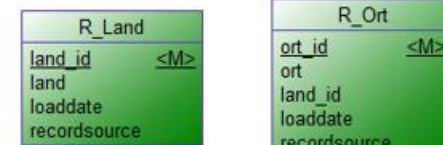
4 5 Abgabe Skript(e)

Architektur Fallstudie



Erstellung Data Vault Modell

- Entity Relationship Diagramm mittels PlantUML <https://plantuml.com/de/ie-diagram>
 - Tabellen (mit **Nutzung von Farben** mit den Vorgaben HUB in blau, LINK in rot, SAT in gelb)
 - Attribut/Spaltennamen inkl. **Datentyp** sowie
 - **Primärschlüssel**
 - Zwingende (**mandatory, NOT NULL**) Schlüssel (Kennzeichnung durch * vor dem Attributnamen)
 - Fremdschlüssel (<<FK>>), erzeugte Schlüssel (<<generated>>), eindeutige Schlüssel (<<unique>>)
 - Angabe der **Kardinalitäten** zwischen Tabellen
- Fachliche Vorgaben
 - Das JSON-Feld (payload) in der Staging-Tabelle Messung wird aufgelöst.
 - Land und Ort stehen als Referenztabellen R_Land und R_Ort zur Verfügung mit integer-Werten als Primärschlüssel. In einer SAT-Tabelle kann z.B. ein Feld ort_id als Fremdschlüssel verwendet werden. Die beiden Referenztabellen müssen nicht ins Diagramm aufgenommen werden. Es genügt eine Angabe in einer SAT-Tabelle wie „ort_id : integer <<FK>>“



PlantUML

„Documentation as Code“ mit ASCII Editor siehe z.B.

<https://www.heise.de/hintergrund/Documentation-as-Code-mit-Asciidoctor-4642013.html?seite=3>

```
@startuml
!theme cyborg
hide circle
'avoid problems with angled crows feet
skinparam linetype ortho

entity "H_Entity01" as h01 #line:purple;back:gold {
    *e1_id : integer <<generated>>
    --
    *name : varchar(50) <<unique>>
    *loaddate : timestamp
    *recordsource : varchar(10)
}

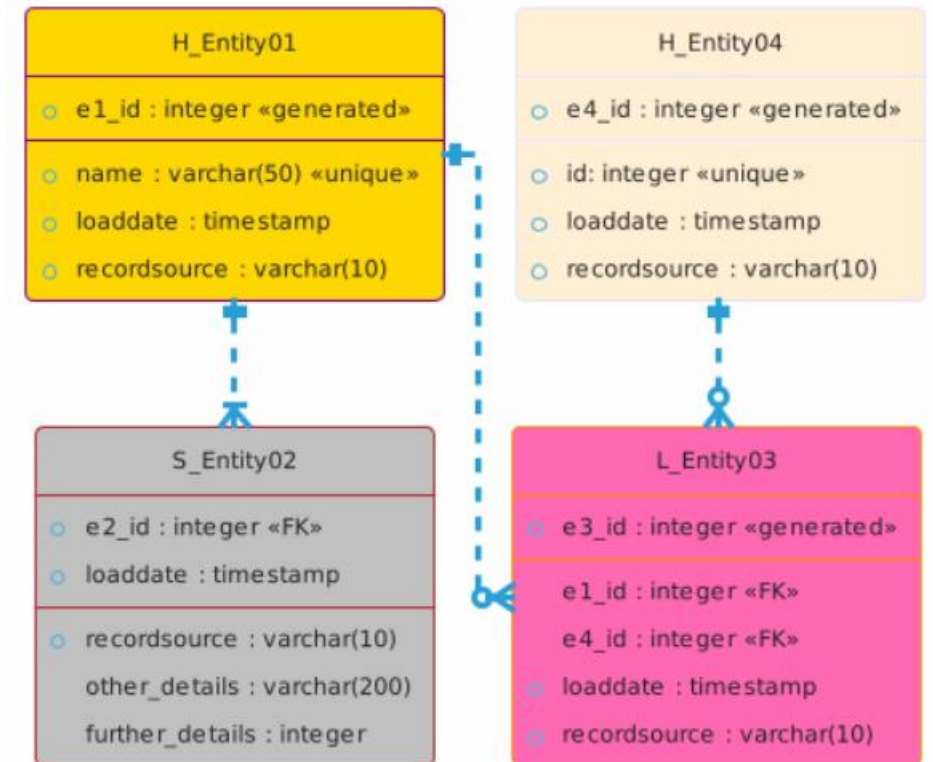
entity "H_Entity04" as h04 #line:Lavender;back:PapayaWhip {
    *e4_id : integer <<generated>>
    --
    *id: integer <<unique>>
    *loaddate : timestamp
    *recordsource : varchar(10)
}

h01 ||..|| s02
h01 ||..o{ 103
h04 ||..o{ 103
@enduml
```

Beispielcode: Auszug,
unvollständig

Farben für Rahmen
und Hintergrund;
Farbnamen z.B.

<https://htmlcolorcodes.com/color-names/>



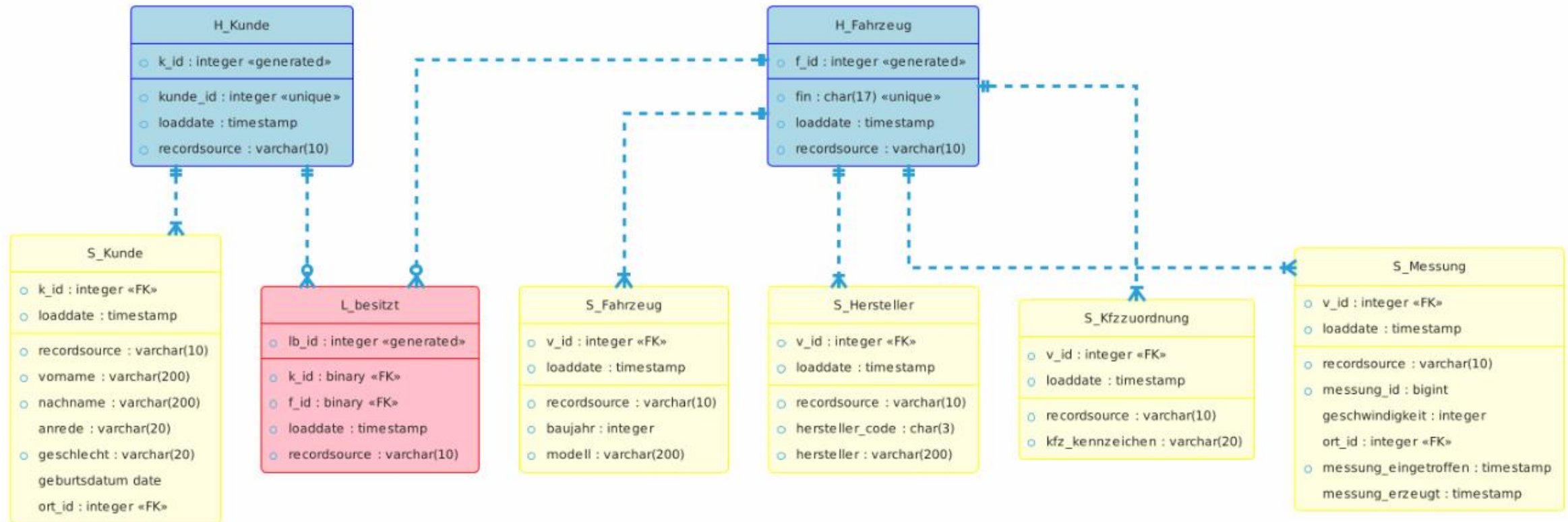
Abgabe Data Vault Modell

- Abgabe in moodle oder github (bei Nutzung github reicht in moodle: txt-Datei mit dem github-Link)
 - txt-Datei mit Anweisungen für PlantUML

6 UND

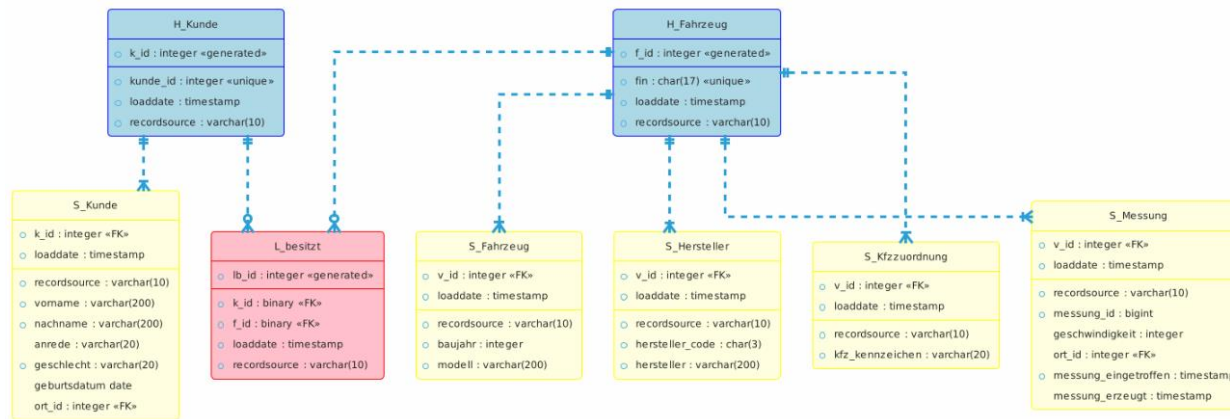
- Screenshot des Diagramms (jpg-Datei)

Datenmodell Data Vault mit loaddate (Data Vault 2.0)



- SAT-Tabelle S_Hersteller könnte auch als HUB und SAT modelliert werden
- Die Attribute in SAT-Tabelle S_Messung sind nicht mandatory, da die Werte aus dem JSON-payload fehlerhaft sein könnten (z.B. keine Zeitstempel sondern string)
- 2 Zeitstempel in SAT-Tabelle S_messung (aus der Differenz kann z.B. die Laufzeit berechnet werden)
 - messung_eingetroffen ist der Zeitstempel, wann die Daten in staging.messung eingetragen wurden
 - messung_erzeugt ist der Zeitstempel, wann der Publisher die Daten erzeugt und gesendet hat

Datenmodell Data Vault mit validfrom/validto (Data Vault < 2.0)



S_kfzzuordnung

Loaddate vs validfrom/validto
(die aktuell gültigen Datensätze sind jeweils Zeilen 2 und 3)

F_ID	Loaddate	Kfz_kennz
1	15.01.2000	UL-ZX 5
1	18.05.2015	UL-JK 25
2	27.02.2012	M-KR 4124

F_ID	Validfrom	validto	Kfz_kennz
1	15.01.2000	17.05.2015	UL-ZX 5
1	18.05.2015	NULL	UL-JK 25
2	27.02.2012	NULL	M-KR 4124

Data Vault loaddate vs validfrom/validto

Neue Versionen (Data Vault 2.0)
nicht immer besser. Hängt von den
Anforderungen ab!

F_ID	Loaddate	Kfz_kennz
1	15.01.2000	UL-ZX 5
1	18.05.2015	UL-JK 25
2	27.02.2012	M-KR 4124

F_ID	Validfrom	validto	Kfz_kennz
1	15.01.2000	17.05.2015	UL-ZX 5
1	18.05.2015	NULL	UL-JK 25
2	27.02.2012	NULL	M-KR 4124

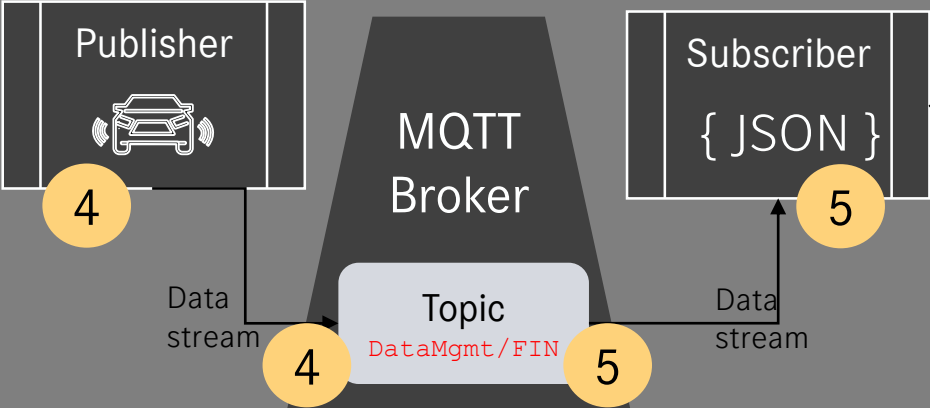
Operation	Loaddate (Data Vault 2.0)	Validfrom/validto (Data Vault < 2.0)
Daten einfügen	INSERT INTO ...	INSERT INTO ... Und Vorgänger-Datensatz abschließen UPDATE ... set validto = (ggf. Vereinfachung mit Time-Travel Funktion der DB)
Daten lesen	SELECT * FROM sat Join (select max(loaddate) from sat group by ...) ON <Primärschlüsselattribute> (oder besser: analytische Funktionen statt max und group by)	Select * From sat Where validto is NULL

Architektur Fallstudie

Internal data sources
OLTP

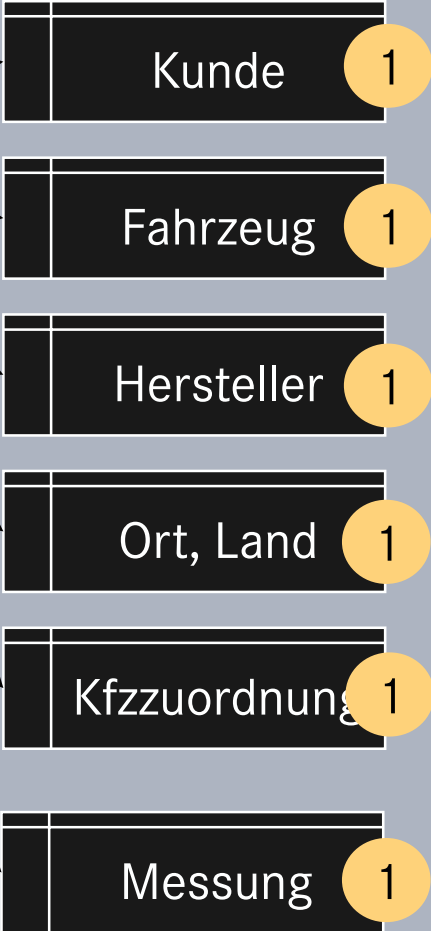


External data sources



PostgreSQL-Backend

Staging



Core

6

Mart

7

Frontend



Erstellung Starschema

- Entity Relationship Diagramm mittels PlantUML <https://plantuml.com/de/ie-diagram>
 - Tabellen (mit **Nutzung von Farben** mit den Vorgaben: Dimensionen = grau und Fakten = grün)
 - Attribut/Spaltennamen inkl. **Datentyp** sowie
 - **Primärschlüssel**
 - Zwingende (**mandatory**, **NOT NULL**) Schlüssel (Kennzeichnung durch * vor dem Attributnamen)
 - Fremdschlüssel (<<FK>>), erzeugte Schlüssel (<<generated>>), eindeutige Schlüssel (<<unique>>)
 - Angabe der **Kardinalitäten** zwischen Tabellen
- Fachliche Vorgaben
 - Eine Faktentabelle mit Auswertung von Geschwindigkeiten (Kennzahl) nach Kunde, Fahrzeug und Ort der Messung.
 - Unterscheiden Sie zwischen den Zeitstempeln „Messung wurde erzeugt“ und „Messung ist im DWH eingetroffen“. Diese Zeitstempel dürfen in der Faktentabelle als timestamp modelliert werden. Sie benötigen somit keine eigene d_date bzw d_timestamp Dimension. Alle Dimensionen sind vom Typ slowly-changing-dimension 1.

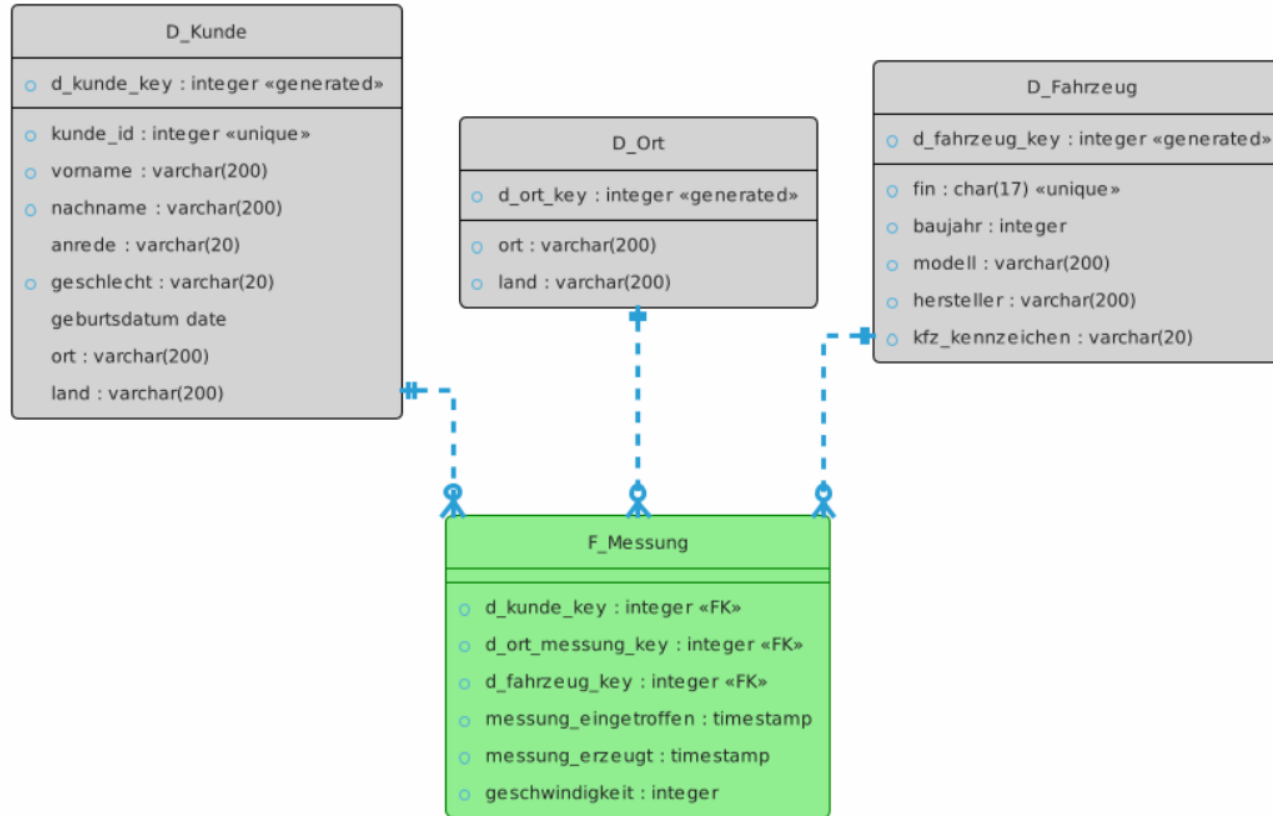
Abgabe Starschema

- Abgabe in moodle oder github (bei Nutzung github reicht in moodle: txt-Datei mit dem github-Link)
 - txt-Datei mit Anweisungen für PlantUML

7 UND

- Screenshot des Diagramms

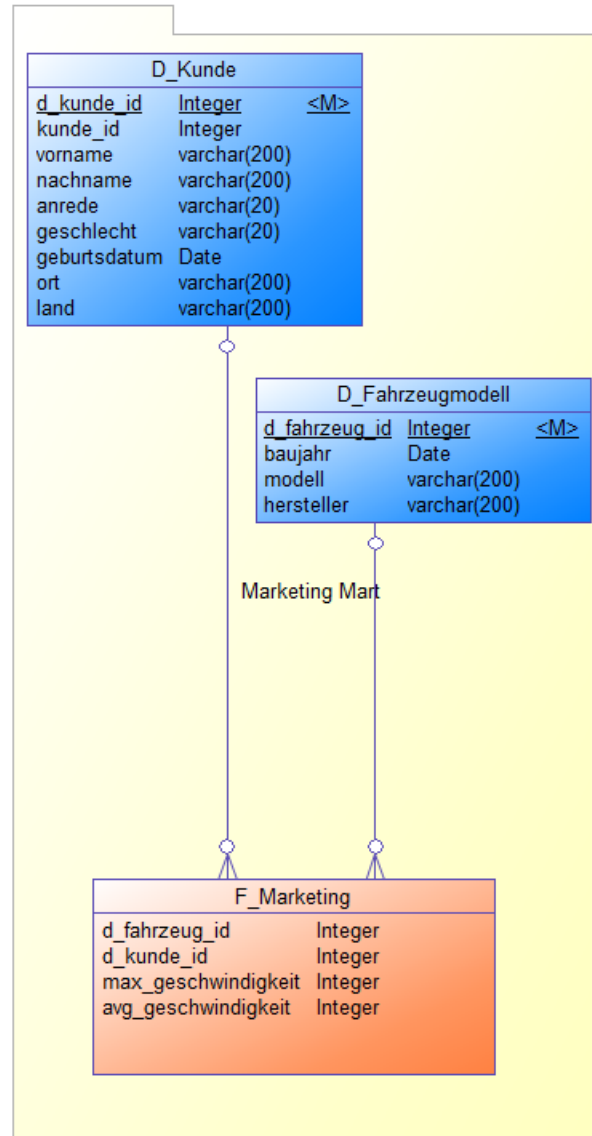
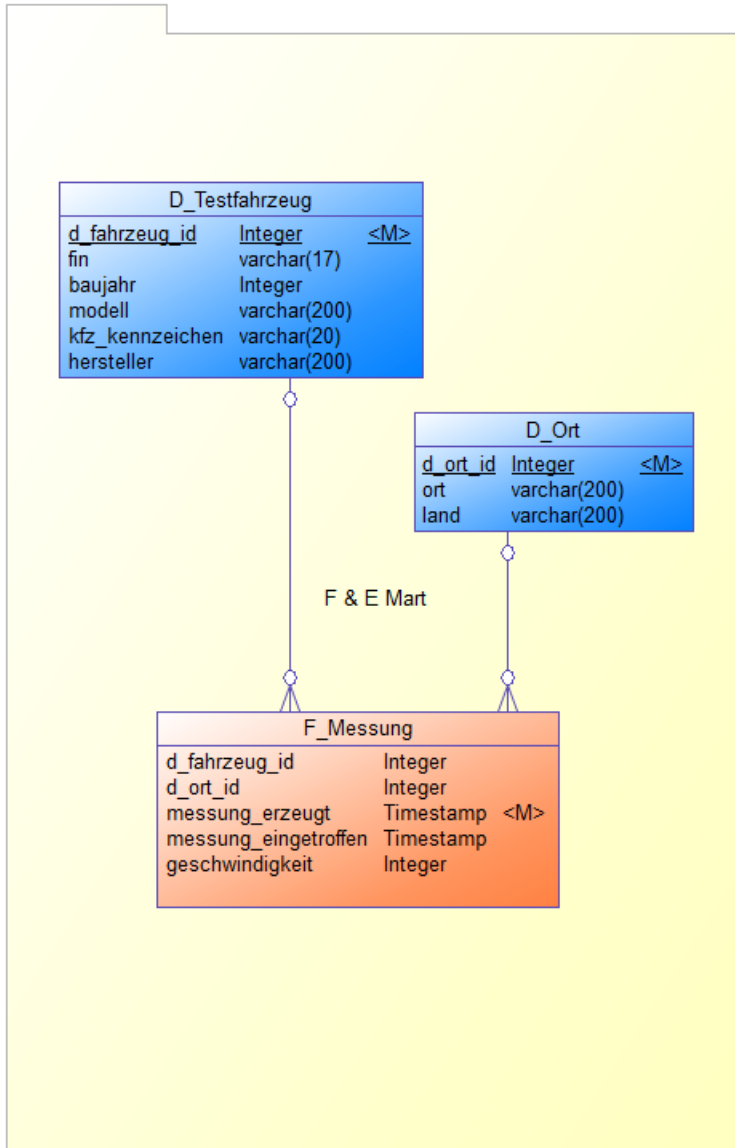
Datenmodell Star-Schema



Anmerkungen

- Künstliche Schlüssel in den Dimensionen mit Datentyp int
 - Performanter gegenüber z.B. varchar
 - DSGVO-relevante Daten wie FIN sollten generell nicht als Primärschlüssel verwendet werden (z.B. falls Daten anonymisiert werden müssen)
- Dimensionstabellen und Faktentabellen entstehen aus den Anforderungen, was eine Fachbereich analysieren möchte (siehe folgende Folie)
- Faktentabellen haben keine Primärschlüssel!

Starschema & Data Mart



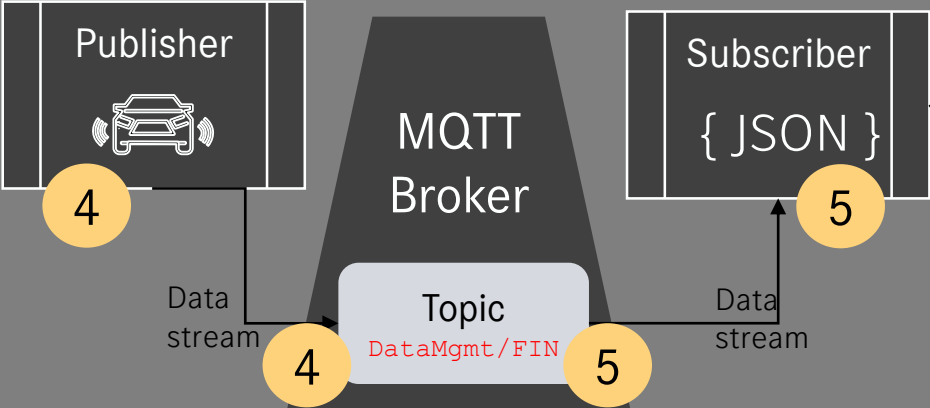
- F&E Mart
 - Analyse von Anomalien in Testfahrzeugen (z.B. Erlkönige)
 - Kundendaten irrelevant, außerdem Datenschutz
- Marketing Mart
 - Analyse von Kunden, welches Fahrzeug angeboten werden könnte
 - Fahrzeugdaten wie FIN oder Kfz-Kennzeichen irrelevant
 - Geschwindigkeit ggf. interessant: schneller Fahrer? Defensiver Fahrer?

Architektur Fallstudie

Internal data sources
OLTP

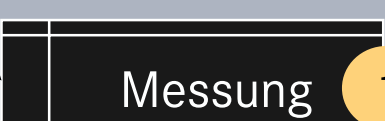
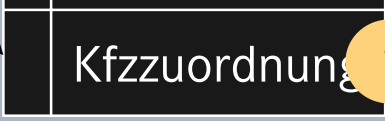
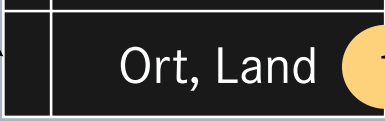
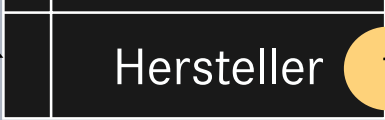


External data sources



PostgreSQL-Backend

Staging



Core



Mart



Frontend

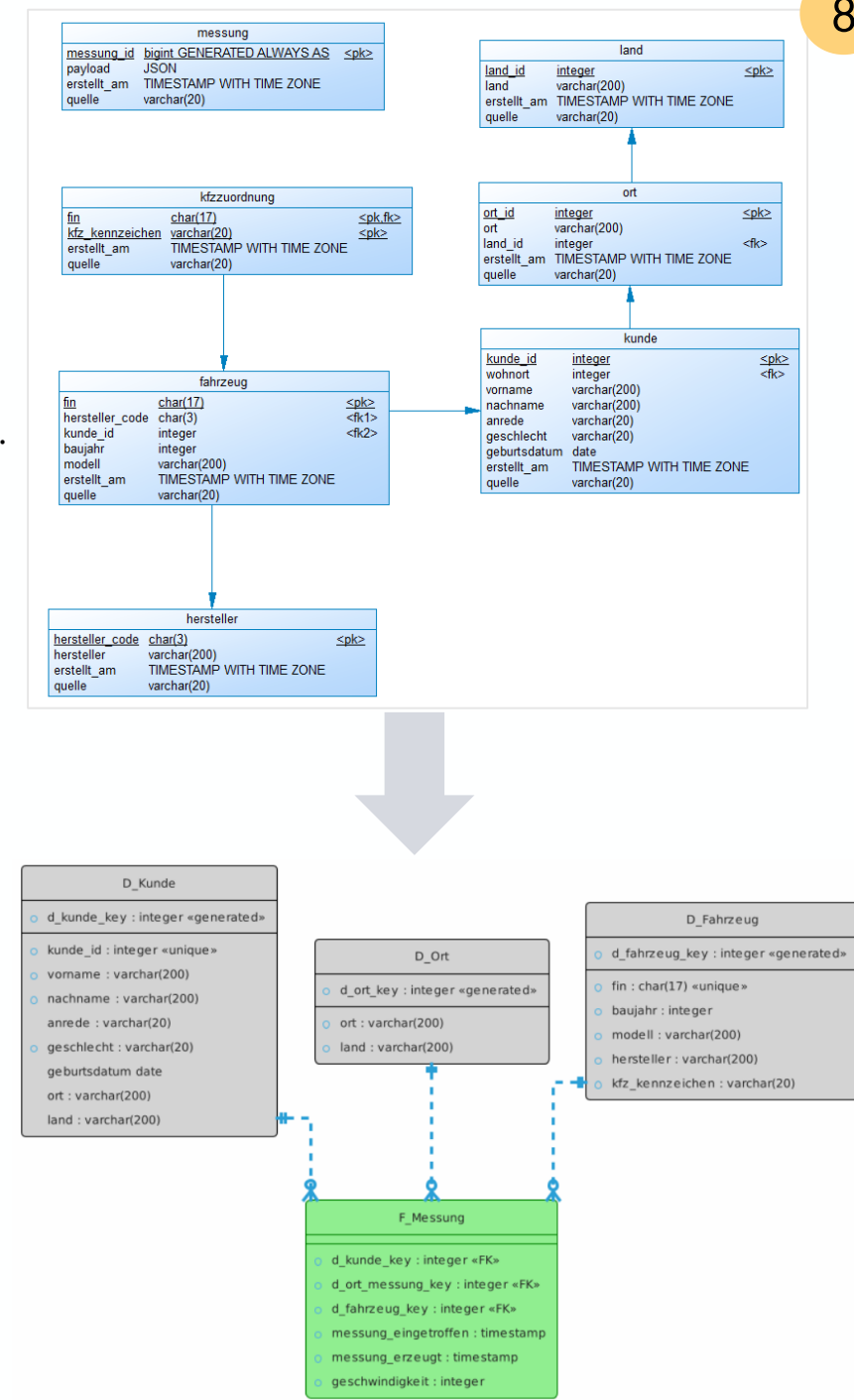


Data Engineering (ETL)

Erstellen Sie ETL/ELT-Prozesse mit Hilfe von Python dbt (data build tool)

- Kein „create table“ notwendig, Views oder Tabellen werden implizit durch dbt erstellt („Materialization“). Die Tabellen dürfen im Schema staging angelegt werden.
- Dbt erstellt keine Primärschlüssel, Fremdschlüssel, usw. Daher müssen diese nicht erstellt werden.
- Die Tabellen in staging können als Quellen („sources.yml“) definiert werden.
- Die Erzeugung der KEY-Werte in den Dimensionstabellen kann z.B. durch „select ROW_NUMBER () OVER ()“ erfolgen
- Dbt unterstützt verschiedene „Materializations“. Zuordnung:
 - D_Ort: table
 - D_Fahrzeug: incremental
 - D_Kunde: snapshot
 - F_Messung: view

Bei incremental und snapshot werden weitere technisch notwendige Spalten in den Tabellen hinzukommen!



Data Engineering - dbt

Infos zu dbt:

- Dbt core
<https://docs.getdbt.com>
- Erste Schritte
<https://timeflow.academy/dbt/labs/creating-first-dbt-project>
<https://docs.getdbt.com/docs/get-started/getting-started-dbt-core>
<https://www.buckenhofer.com/2022/11/data-engineering-with-dbt-first-steps-using-postgresql-and-oracle/>
- Materialization
<https://blog.jetbrains.com/big-data-tools/2022/02/22/dbt-deeper-concepts-materialization/>
<https://www.buckenhofer.com/2022/11/materialization-examples-of-data-engineering-with-dbt/>

pip install dbt-core dbt-postgres

Abgabe Data Engineering

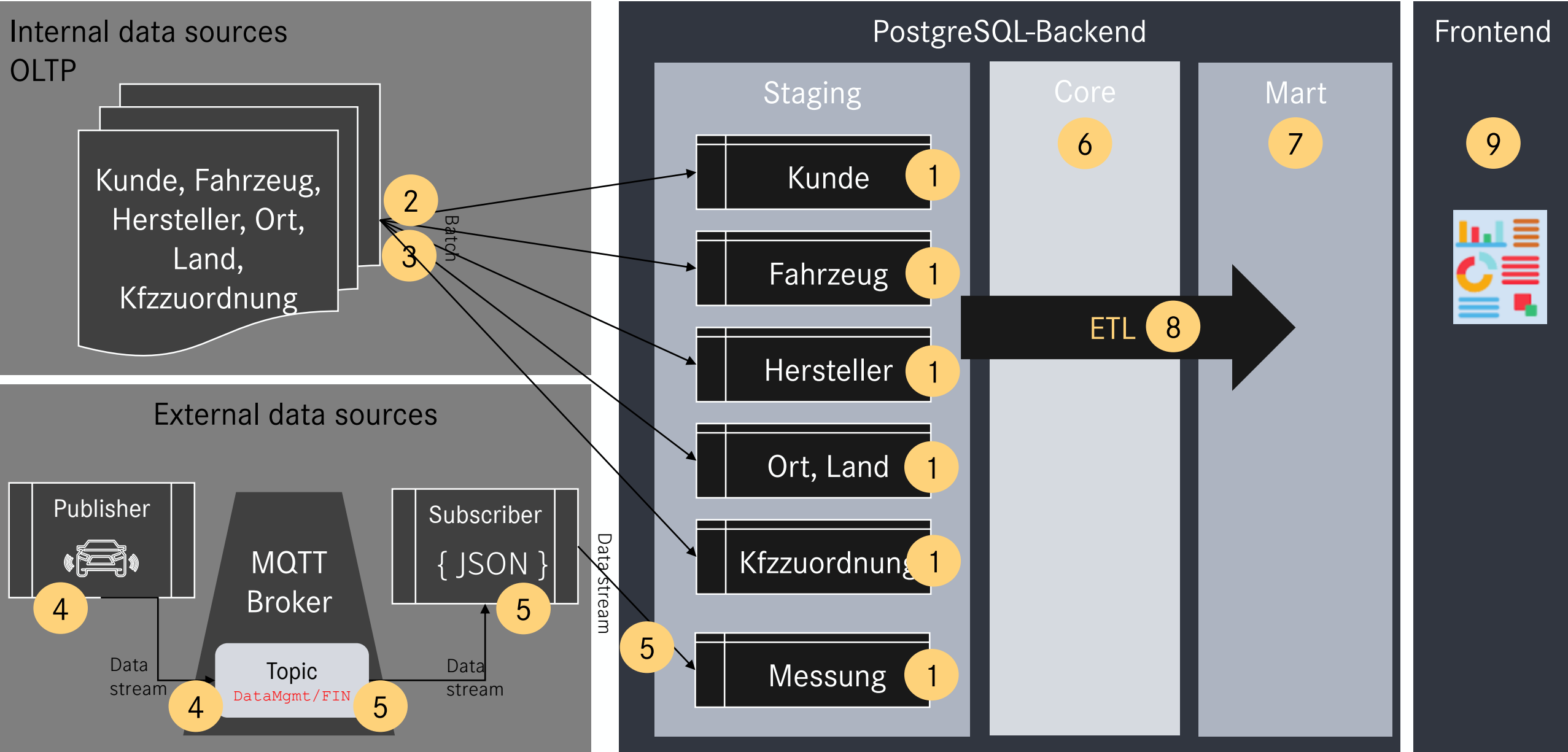
Führen Sie zur Prüfung der Daten aus:

```
select distinct geschwindigkeit from staging.f_messung order by geschwindigkeit;
```

Werden Geschwindigkeiten sortiert ausgegeben?

- Abgabe in moodle oder github (bei Nutzung github reicht in moodle: txt-Datei mit dem github-Link)
 - Dbt Verzeichnis (models und snapshots)


Architektur Fallstudie



Frontend und Abgabe

Erstellen Sie eine Grafik aus den Daten im Data Mart mittels Python/Javascript.

- Abgabe in moodle oder github (bei Nutzung github reicht in moodle: txt-Datei mit dem github-Link)
 - Quellcode + Screenshot der Grafik



Mercedes-Benz Tech Innovation GmbH
Wilhelm-Runge-Straße 11, 89081 Ulm
Telefon +49 731 505-06 / techinnovation@mercedes-benz.com / www.mercedes-benz-techinnovation.com
Sitz und Registergericht: Ulm / HRB-Nr.: 3844 / Geschäftsführung: Daniel Geisel (Vorsitzender), Isabelle Krautwald

Mercedes-Benz Tech Innovation