Datenbanken 2

Oliver Lindemann

Hochschule Harz  
University of Applied Science  
u33873, m26264  
u33873@hs-harz.de

**ABSTRACT**

This paper will introduce you to relational database systems and Data-Warehouses. It explains what they are used for and uses an Event-Shop to demonstrate how they are built.

Modul: Datenbanken 2

Prüfer: Prof. Dr. Kerstin Schneider

Abgabedatum: 28.02.2020

# EINLEITUNG

Datenbanken finden sich heutzutage in fast allen Anwendungsbereichen wieder. Sei es die Kundenverwaltung eines Unternehmens, der Warenbestand in einem Supermarkt oder die bevorzugte Suchmaschine. Überall werden Datenbanken zum Verwalten und Organisieren der Mengen an Daten eingesetzt. In dieser Facharbeit werden relationale Datenbanksysteme (RDBS) vorgestellt, die Anwendungsmöglichkeiten von Data-Warehouses und die Tabellenstruktur des Event-Shop-Empfehlungssystems erläutert.

# GRUNDLAGEN

Eine relationale Datenbank ist eine Ansammlung strukturierter Daten, die miteinander in Relation stehen (können). Diese Daten werden in Tabellen organisiert, welche die Datenstruktur abbilden. Eine Tabelle besteht aus mehreren Spalten, welche jeweils eine Eigenschaft der Daten abbildet, und den Reihen, die jeweils einen vollständigen Datensatz darstellen. Die Besonderheit einer relationalen Datenbank sind die Relationen / Beziehungen zwischen den Daten bzw. den Tabellen. Diese erlauben das Referenzieren von Datensätzen anderer Tabellen in einem Datensatz[[1]](#footnote-1). Allgemein gesprochen können durch Relationen Redundanzen vermieden werden, da auf einen einmal eingefügten Datensatz von anderen Stellen verwiesen werden kann und dieser nicht an jeder dieser Stellen erneut eingefügt werden muss. Dieses Prinzip der Redundanzvermeidung wird *Normalisierung* genannt. Damit eine Datenbank *normalisiert* ist, muss diese die drei aufeinander aufbauenden *Normalformen (NF)* erfüllen:

**1. NF**: Alle Attribute sind atomar (kleinste mögliche Einheit)

**2. NF**: Alle Nichtschlüsselattribute sind vom Primärschlüssel abhängig

**3. NF**: Alle Nichtschlüsselattribute sind voneinander unabhängig

Normalisierte und somit auch redundanzfreie Daten haben den Vorteil, dass bei einer Änderung eines Datensatzes nur eine einzige Stelle aktualisiert werden muss.

Der Nachteil von normalisierten Daten ist allerdings die Geschwindigkeitseinbuße im operativen Betrieb. Einen vollständigen Datensatz mit all seinen Referenzen zu sammeln ist aufwändig, da die einzelnen Datensätze über mehrere Tabellen verteilt liegen. Es muss also über jede dieser Tabellen iteriert und nach den passenden Daten gesucht werden, um einen vollständigen Datensatz zu erhalten. Aus diesem Grund existieren andere Datenbankansätze, die bewusst gegen die Normalisierung verstoßen, um die Laufzeit zu verbessern. Diese Systeme sind *denormalisiert*. Ein typisches Beispiel eines solchen Systems ist das Data-Warehouse (DWH).

Eine Datenbank ist *denormalisiert*, sobald diese gegen eine der drei Normalformen verstößt, beispielsweise bei Einführung bewusster Redundanzen.

# DATA-WAREHOUSE

Ein Data-Warehouse (DWH) ist *denormalisiert* und bildet in den meisten Fällen bereits existierende Daten von einer oder mehreren normalisierten Datenbanken ab. Die Daten werden i.d.R. noch verarbeitet und erweitert, indem Ableitungen von diesen erstellt und ebenfalls in den Tabellen gespeichert werden. Dies nimmt eventuelle Berechnung von Abfragen und Analysen vorweg und verbessert somit die Geschwindigkeit. Aus diesem Grund werden in das DWH vom Anwender keine neuen Daten eingefügt, sondern lediglich auf die in den operationalen Systemen verfügbaren Daten zugegriffen. Dem Data-Warehouse liegen also in den meisten Fällen, wie in *Abbildung 1 Data-Warehouse Architektur* dargestellt, bereits ein oder mehrere bestehende Datenbanken zugrunde.

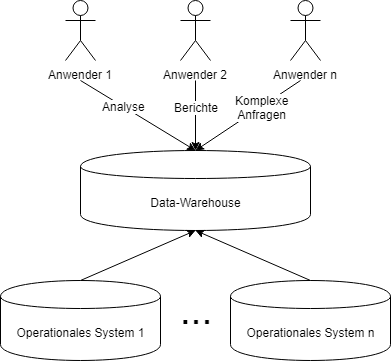


Abbildung 1 Data-Warehouse Architektur

Der Unterschied zwischen der *Normalisierung* und der *Denormalisierung* lässt sich anhand eines Datums in einer Tabelle darstellen:

In einer *normalisierten* Tabelle liegt nur das einzelne Datum ab. Aus diesem Datum lassen sich später Tag, Monat, Jahr und noch weitere Daten ableiten.

In einer *denormalisierten* Tabelle liegen neben dem Datum nun auch noch die abgeleiteten Eigenschaften des Datums ab, wie der Tag, der Monat, das Jahr, die Kalenderwoche und vieles weitere. Dies erspart für zukünftige Abfragen und Analysen die Berechnung der abgeleiteten Daten und erhöht die Geschwindigkeit des DWH.

## DWH Schemas

Für die Realisierung von Data-Warehouses gibt es verschiedene Ansätze und damit einhergehend unterschiedliche Schemas [1]:

* Star Schema
* Snowflake Schema

Die Namen der Schemas ergeben sich aus der Anordnung der Tabellen und der damit entstehenden Figuren / Objekte.

### Star Schema

Das Star-Schema ist ein einfaches Schema, welches sternförmig aufgebaut ist. Es besteht aus einer Faktentabelle, welche den Mittelpunkt des Sternes symbolisiert, und mehreren Dimensionstabellen, die sternförmig um die Faktentabelle positioniert sind.

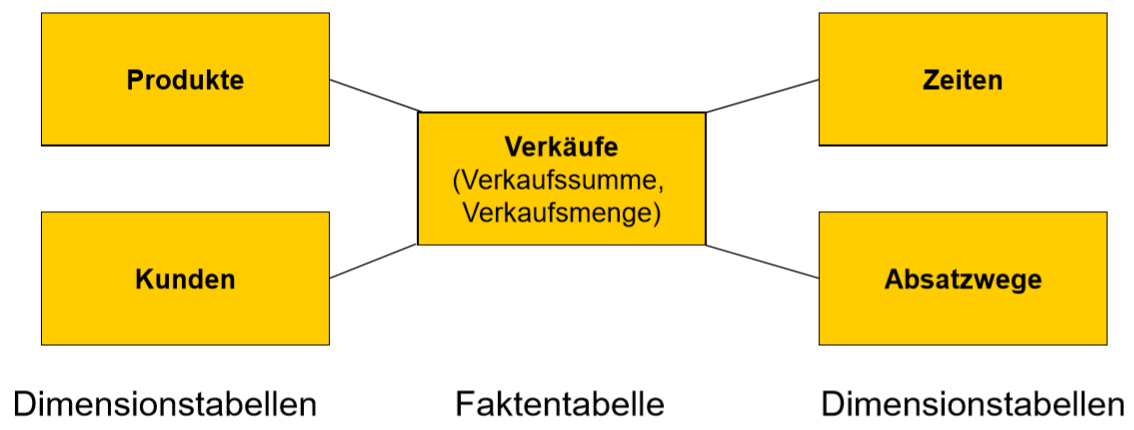


Abbildung 2 DWH Star Schema

Die Dimensionstabellen in einem Star Schema sind nicht in der 3. Normalform, sondern lediglich in der 2. Dies hat den Vorteil, dass alle relevanten Daten von der Faktentabelle mittels nur eines *Joins*[[2]](#footnote-2) erreicht und gesammelt werden können.

### Snowflake Schema

Das Snowflake Schema ist ein komplexeres DWH Schema. Dimensionstabellen liegen in der 3. Normalform vor und können, um Redundanzen zu vermeiden, in kleinere Tabellen unterteilt sein. Dies bedeutet auch, dass Dimensionstabellen mit weiteren Dimensionstabellen in Relation stehen können. Allerdings werden dadurch bei einer Abfrage auch mehrere *Joins* benötigt, um diese zu vereinen.

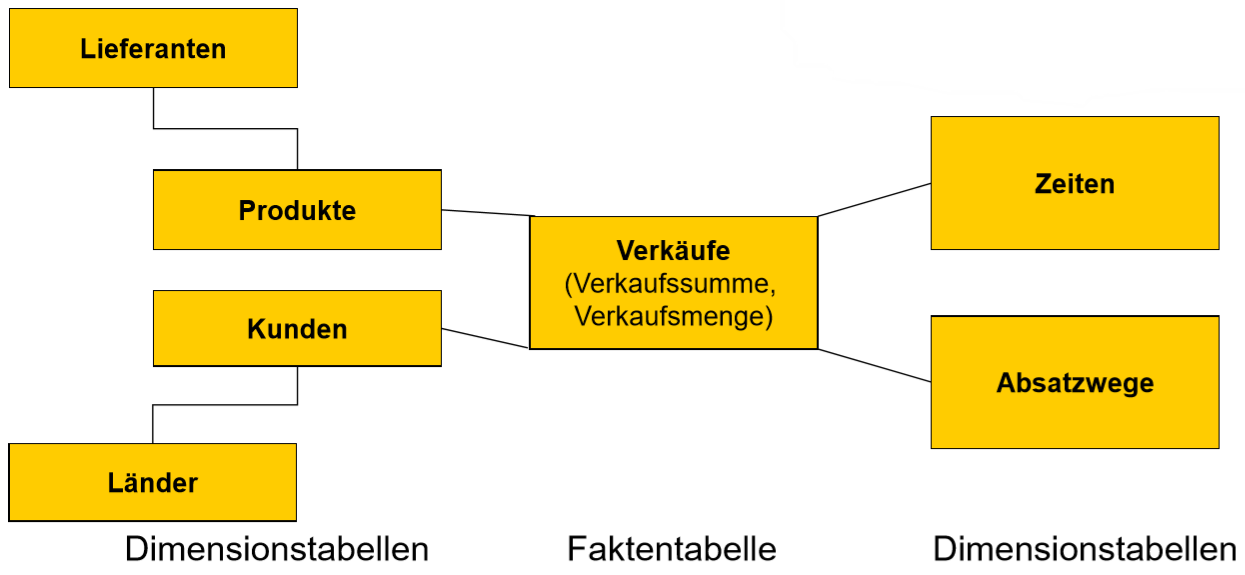


Abbildung 3 DWH Snowflake Schema

*Faktentabelle*: speichert die zu verwaltenden bzw. die zu analysierenden Daten, z.B. Ticketverkäufe.

*Dimensionstabelle*: speichert die Attribute und Eigenschaften von Fakten (bzw. beschreiben Fakten im Detail).

Faktentabellen enthalten Primärschlüssel der Dimensionstabellen.

# EVENT-SHOP DATENBANK

Die Event-Shop Datenbank ist nach dem Data-Warehouse Prinzip aufgebaut. Allerdings fallen aus Gründen der Einfachheit die unterliegenden operationalen Datenbanken weg. Das DWH dient somit zugleich als operatives System.

Der Event-Shop hat folgende Anforderungen an die Datenbank:

* Ein Kunde (Customer) kann sich registrieren und mittels seiner E-Mail-Adresse und Passwortes anmelden.
* Der Kunde kann Veranstaltungen (Events) auswählen und kaufen.
* Der Kauf (Purchase) erfordert eine Anzahl an Tickets, eine Bezahlmethode (Paymethod) und eine Ticketart (Tickettype), die etwaigen Rabatt ermöglicht.
* Gekaufte Events kann der Kunde bewerten (Rating).

Aus diesen Anforderungen lassen sich verschiedene Tabellen entwickeln, die i m weiteren Verlauf dieser Facharbeit erläutert werden.

## Tabellen des Event-Shops

### Events

Die Event-Tabelle beinhaltet alle verfügbaren Events. Diese Events sind gegliedert in Titel (TITLE), Beschreibung (DESCRIPTION), Typ (TYPENAME), Genre (GENRE) und Sub-Genre (SUB\_GENRE). Zudem kommt der Veranstaltungsort mit der Straße (STREET), Hausnummer (HOUSENUMBER), Postleitzahl (ZIP), dem Ort (CITY) und Land (COUNTRY) inkl. Länderkürzel (COUNTRY\_SHORT). Des Weiteren wird der Veranstalter (ORGANIZERNAME), das Veranstaltungsdatum (DATE) inkl. julianischer Datumsangabe (DATE\_JULIAN) und die Dauer (DURATION\_IN\_MINUTES) des Events in Minuten angegeben. Zuletzt hat jedes Event noch einen Eintrittspreis (PRICE), der noch einmal als Centbetrag angegeben wird (PRICE\_CENT).

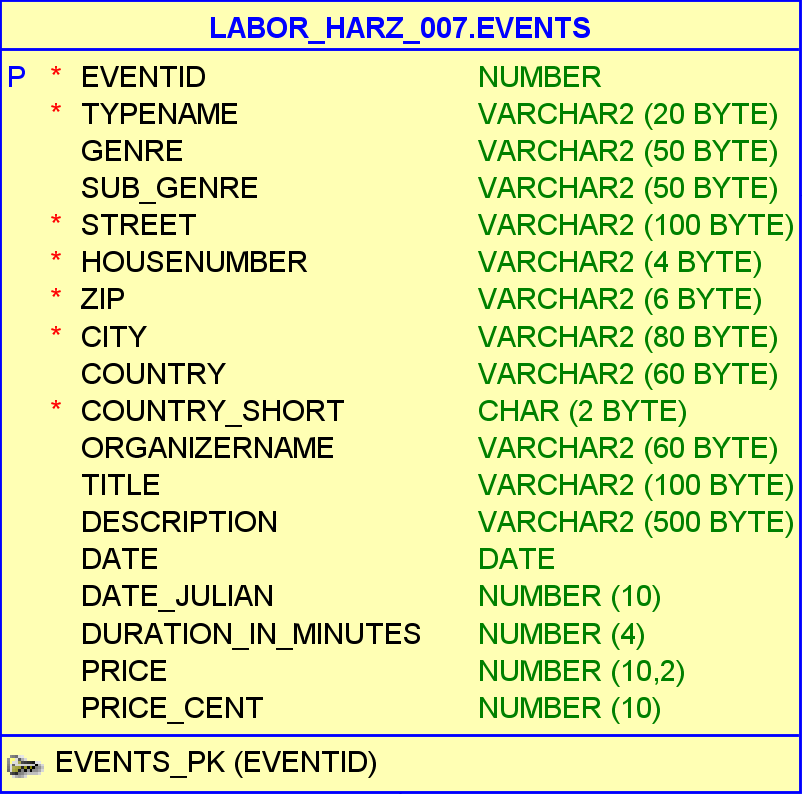


Abbildung 4 Tabelle EVENTS

### Customer

Die Tabelle Customer beinhaltet alle registrierten Kunden. Die Spalten der Tabelle enthalten den Namen (FIRSTNAME, LASTNAME), das Geschlecht (GENDER, GENDER\_SHORT), ihr Geburtsdatum als Datum (BIRTHDAY\_DATE) und in julianischem Datumsformat (BIRTHDAY\_JULIAN), das Geburtsjahr (BIRTHDAY\_YEAR), die vollständige Adresse (STREET, HOUSENUMBER, ZIP, CITY, COUNTRY, COUNTRY\_SHORT), die E-Mail-Adresse (EMAILADDRESS) und Provider (EMAIL\_PROVIDER), ihr gewähltes Passwort (PASSWORD) und die ausgewählten Präferenzen (PREFERENCE1, PREFERENCE2, PREFERENCE3). Abbildung 5 stellt alle Spalten detailliert dar:

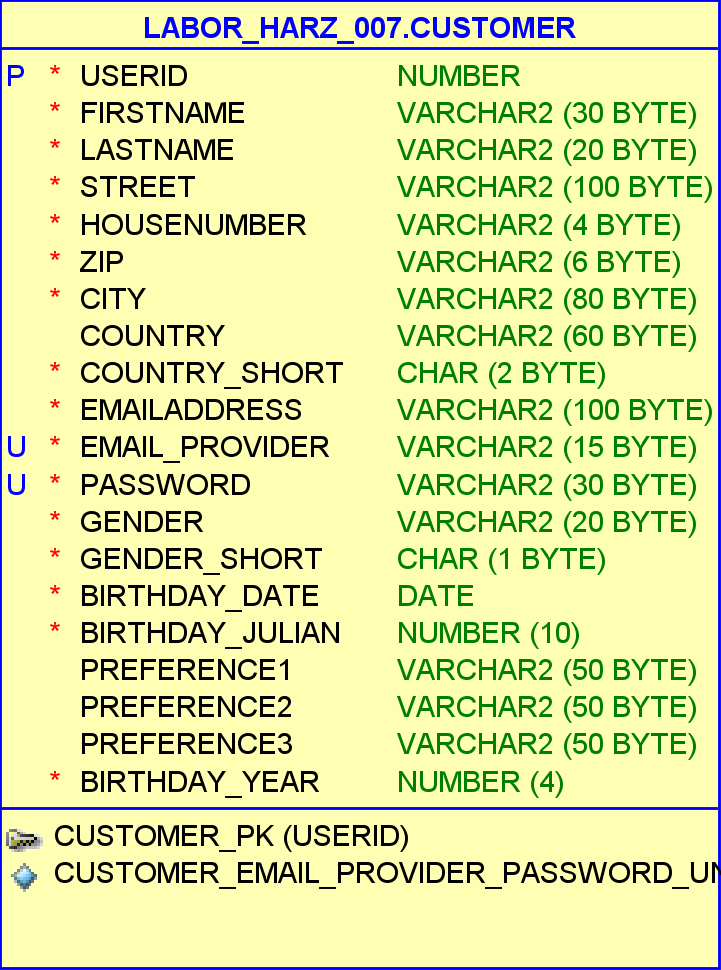
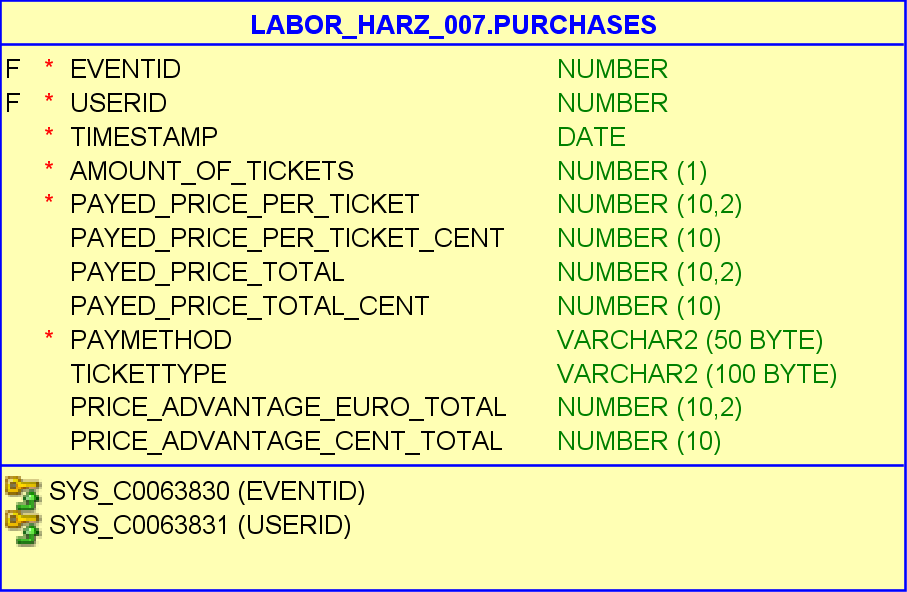


Abbildung 5 Tabelle Customer

### Purchases

Die Tabelle Purchases beinhaltet alle von dem Benutzer getätigten Käufe. Bei einem Kauf wird das Kaufdatum (TIMESTAMP), die Anzahl der Tickets (AMOUNT\_OF\_TICKETS), der Preis für jedes Ticket (PAYED\_PRICE\_PER\_TICKET) und der gezahlte Gesamtpreis (PAYED\_PRICE\_TOTAL) gespeichert. Diese beiden Preise werden jeweils noch einmal in Cent-Beträgen aufgelistet (PAYED\_PRICE\_TOTAL\_CENT, PAYED\_PRICE\_PER\_TICKET\_CENT). Außerdem wird die gewählte Bezahlmethode (PAYMETHOD), die Art der gekauten Tickets (TICKETTYPE) und der gewährte Rabatt (PRICE\_ADVANTAGE\_EURO\_TOTAL, PRICE\_ADVANTAGE\_CENT\_TOTAL) gespeichert.

Abbildung 6 Tabelle Purchases



### Ratings

In einer RATINGS-Tabelle wird die vom Benutzer (USERID) vergebene Bewertung (RATING) für ein bestimmtes Event (EVENTID) mit dem Bewertungsdatum (TIMESTAMP) gespeichert.

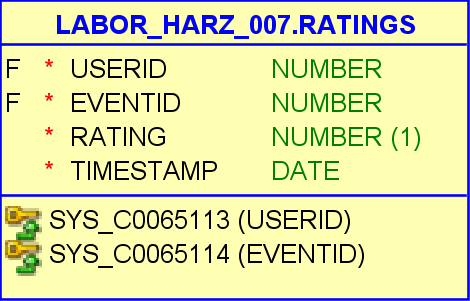


Abbildung 7 Tabelle Ratings

### Hilfstabellen

Hilfstabellen sind Tabellen, die in keiner Relation zu anderen Tabellen stehen. Der Inhalt der Tabellen wird allerdings für die Generierung weiterer Datensätze anderer Tabellen verwendet.

#### Countries

In der Countries-Tabelle sind Ländercodes (CODE) und deren deutsche Bedeutung (DE) hinterlegt (z.B. AT = Österreich).

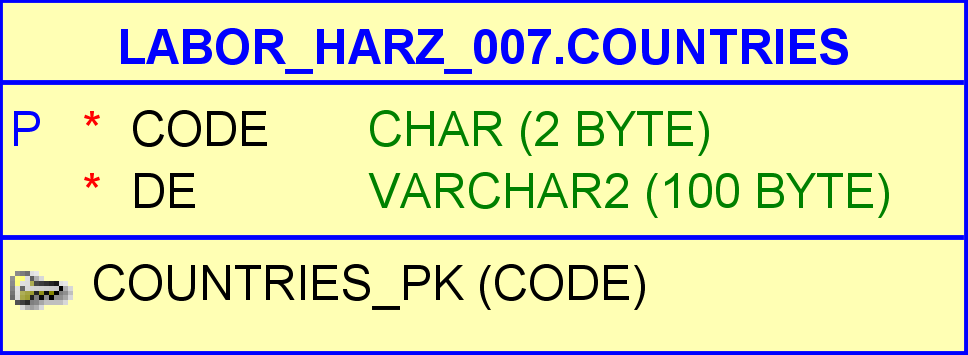


Abbildung 8 Tabelle Countries

#### Paymethod

Die Tabelle Paymethod beinhaltet alle Namen von möglichen Bezahlmethoden, zwischen denen der Benutzer beim Kauf eines Tickets wählen kann (NAME).

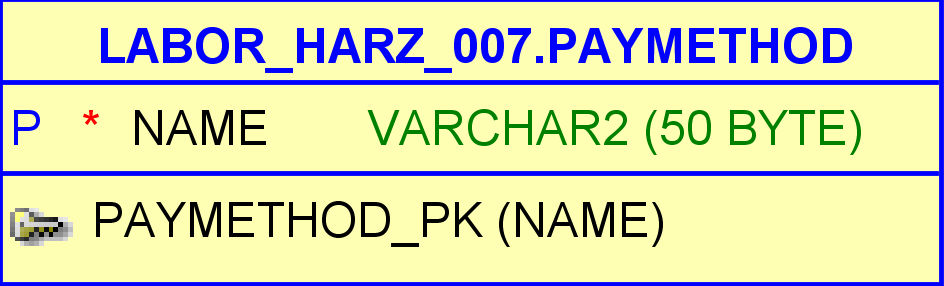


Abbildung 9 Tabelle Paymethod

#### Tickettypes

Tickettypes beinhaltet alle möglichen Arten von Tickets und deren Rabatt. Dies beinhaltet den Namen des Tickettypen (NAME) und den damit verbundenen Preisnachlass in Prozent (PERCENTAGE).

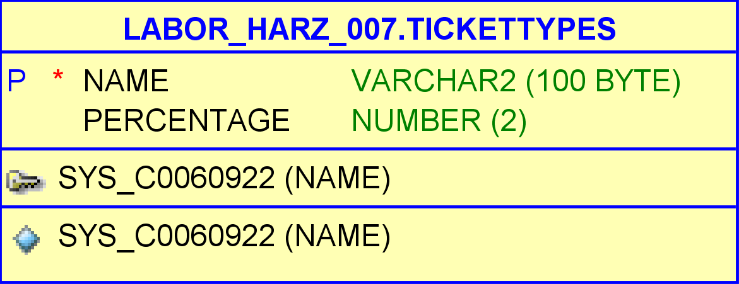


Abbildung 10 Tabelle Tickettypes

### Analysis\_ Queries

In der Tabelle Analysis\_Queries sind verschiedene Analyse-Abfragen (SQLQUERY) unter einem Namen (NAME) im JSON-Format abgelegt. Um eine solche Analyse auszuführen, wird die gewünschte SQL-Anfrage aus der Tabelle geholt und auf der Datenbank ausgeführt. Diese liefert dann eine Analyse mit stets aktuellen Daten, welche daraufhin visualisiert werden können.

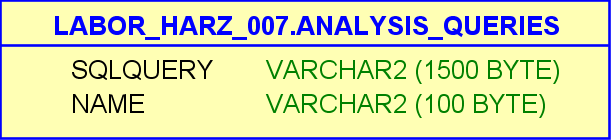


Abbildung 11 Tabelle Analysis\_Queries

### Times

Die Tabelle Times enthält pro Datensatz ein Datum und all dessen abgeleiteten Eigenschaften, wie z.B. das Jahr, den Monat, den Tag der Woche / Monat / Jahr, die Kalenderwoche, das Quartal u.v.m.:

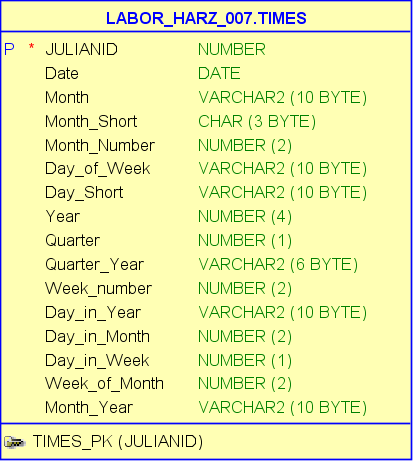


Abbildung 12 Tabelle Times

## Sequenzen

Eine Sequenz erzeugt numerische Werte, welche automatisch mit einer definierten Schrittweite inkrementiert werden. Diese erzeugten Werte werden häufig für eindeutige Primärschlüsselwerte verwendet. Dabei wird in der Regel eine Sequenz für eine einzige Tabelle verwendet. Bei der Erzeugung eines neuen Sequenzwertes wird die Sequenz automatisch inkrementiert, unabhängig eines *Commits* oder *Rollbacks* einer *Transaktion* [2].

Eine *Transaktion* ist eine Sammlung von SQL-Statements, die gemeinsam ausgeführt werden. Falls alle Statements ohne Fehler beendet werden, wird das Ergebnis mittels eines *Commits* in die Datenbank übertragen. Beim Auftreten eines Fehlers wird mit einem *Rollback* die Änderungen der zuvor ausgeführten Statements rückgängig gemacht und die Datenbank bleibt unverändert.

Nach Erstellung einer Sequenz kann über den vergebenen Namen und des SQL-Statements *CURRVAL* der aktuelle Wert der Sequenz ermittelt werden, wohingegen mittels *NEXTVAL* ein neuer Wert erzeugt und zurückgegeben wird.

Die Datenbank des Event-Shops beinhaltet mehrere Sequenzen, die für die Erzeugung von Primärschlüsseln verwendet werden. Jede Dimensionstabelle hat eine eigene Sequenz mit einem entsprechenden Namen. Beim Einfügen eines Datensatzes wird mittels der zu der Tabelle zugehörigen Sequenz die eindeutige ID generiert, welche zugleich als Primärschlüssel dient.

## Verwendetes DWH-Schema

Das Event-Shop-DWH verwendet das Star-Schema. Die Käufe der Benutzer (Purchases-Tabelle) dienen als Faktentabelle und die Events, Customer, Tickettype, Paymethod sowie das Datum des Kaufes repräsentieren die Dimensionstabellen. In der *Abbildung 13 Star-Schema der Tabellen* sind diese Tabellen nach dem Sternschema angeordnet:

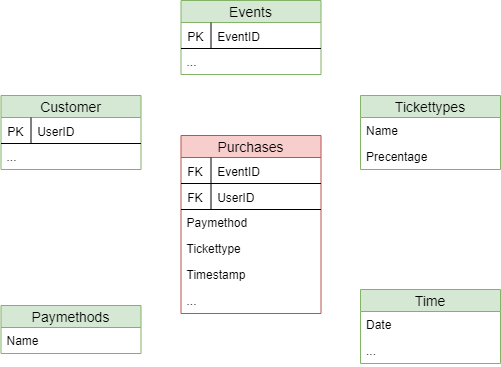


Abbildung 13 Star-Schema der Tabellen

# ZUSAMMENFASSUNG

In dieser Facharbeit wurden die Grundlagen von Datenbanken, des Data-Warehouses und der verschiedenen Schemas erläutert. Zudem wurden die Anforderungen an die Event-Shop-Datenbank vorgestellt und die daraus erstellten Tabellen erklärt. Außerdem wurden Sequenzen und ihre Rolle in der erstellten Datenbank dem Leser nähergebracht. Abschließend wurde erläutert, welches DWH-Schema für den Event-Shop verwendet wurde und welche Tabellen als Fakten- und welche als Dimensionstabellen fungieren.

# AUSBLICK

Für große und weitreichende Analysen sind die Datensätze in der Event-Shop-Datenbank zu gering. Zudem müssten die Tabellen für performante Analysen noch die Zeitunterteilung aus der Tabelle *Times* übernehmen und beispielsweise der Zeitstempel noch auf Jahr, Monat, Tag, Quartal, Kalenderwoche und mehr erweitert werden. Dies erspart Berechnungszeit. Allerdings ist dies in dieser Facharbeit nicht das primäre Ziel gewesen.

Außerdem könnte noch bei Analysen eine Unterteilung der Tabellen Purchases und Ratings in Data-Cubes erfolgen. Diese bilden aus den Daten sog. mehrdimensionale Würfel, wobei eine Dimension jeweils ein Produkt, Kunde oder Zeit darstellt.

# QUELLENVERWEIS

1. Schneider, K. *Datenbanksysteme 2* *(ehem. Data Warehouse).* Hochschule Harz, (2019).
2. Lorentz, D., Oracle© Database SQL Language Reference, *CREATE SEQUENCE*,11g Release 1 (11.1) B28286-07, *Oracle and/or its affiliates*, pages 1105-1108 (2013). [online] Available at: https://docs.oracle.com/cd/B28359\_01/server.111/b28286.pdf

1. Beispielsweise referenziert eine Bewertung ein spezielles Event, für das diese Bewertung geschrieben wurde. [↑](#footnote-ref-1)
2. Ein *Join* verknüpft / vereint zwei in Relation stehende Tabellen [↑](#footnote-ref-2)